Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Dottorato di Ricerca in Ingegneria dell'Informazione

Humans in the loop: Optimization of active and passive crowdsourcing

Advisor:	**Prof. Piero Fraternali**
Co-Advisor:	**Prof. Davide Martinenghi**
	Prof. Marco Tagliasacchi
Tutor:	**Prof. Barbara Pernici**
Supervisor of the Doctoral Program:	**Prof. Carlo Fiorini**

Doctoral dissertation of:
Eleonora Ciceri

2015 - XXVII

Abstract

In the last years, social media have attracted millions of users and have been integrated in people's daily practices. They enable users to create and share content or to participate in social networking. User-generated content, i.e., the various forms of media assets publicly available and created by end-users, is published every day on the Web and mostly in social media at a massive scale, either in the form of textual documents (e.g., blog articles, posts on social networks, comments and discussion) or in the form of multimedia items (e.g., images and videos). Most user-generated content is about personal lives and facts about users. However, users often publish more structured and complex information.

Crowdsourcing has gained increasing importance in the last years. The term crowdsourcing generally refers to the outsourcing of a non-automatable task to people. The growth of the time spent online has led to a growth of interest in crowdsourcing. Several works have been developed, either making users actively participate in the resolution of tasks or exploiting data they generate and publish over the Web. We refer to these approaches as, respectively, active crowdsourcing (i.e., active participation of motivated users in task execution) and passive crowdsourcing (i.e., exploitation of user-generated content to extract useful information).

On the one hand, active crowdsourcing is the process of outsourcing tasks to a large group of people, called workers. In this scenario, human workers are asked to perform very specific tasks (called crowd tasks), which usually are easy to be solved by humans but hard to be solved by machines. In the context of active crowdsourcing, only tasks difficult to be performed by a machine are submitted as crowd tasks. They are often based on uncertain data, since these data can hardly be processed by computers, due to their unstructured nature. Unfortunately, an appropriate modeling of the impact of a crowd task answer on uncertain data is yet to be defined. Moreover, similarly to the use of machine resources, which cost, also human computational resources are not freely available in any amount, and may provide erroneous answers. Consequently, an approach for the selection of the best candidate set of tasks to submit to the crowd under some fixed constraints (e.g., costs and time) needs to be devised, together with quality assurance procedures that guarantee an appropriate result quality level.

On the other hand, passive crowdsourcing denotes an alternative approach for leveraging the online activity of users for task resolution, which amounts to analyzing a huge amount of publicly available contents, to extract information about behaviors, interests and activities of the social media population. Researchers from different fields (e.g., social science, economy and marketing) analyze a variety of user-generated datasets to understand human behaviors, find new trends in society and possibly formulate adequate policies in response. However, due to the uncontrolled nature of users' participation on the Web, the huge mass of available data contains replicated information, as well as low quality or irrelevant content. Moreover, content is often replicated maliciously: users copy content created by others (and often subject to copyright laws), rename it and pretend they are the authors of the corresponding original content.

In this Thesis, we propose methods to overcome these problems, both in the active and passive crowdsourcing field, with the objective of maximizing the quality of results, under the assumption of budget and time constraints.

Sommario

Durante gli ultimi anni, i social media hanno attratto milioni di utenti, e sono stati integrati progressivamente nella loro routine quotidiana. Essi permettono di creare e condividere contenuto, o di connettersi con altre persone. Lo *user-generated content*, ovvero le varie forme di contenuto creato dagli utenti e disponibile sui media, è pubblicato ogni giorno sul Web in quantità enormi, sia in formato testuale (ad esempio sottoforma di articoli in blog, post sui social network, commenti e discussioni) sia sottoforma di file multimediali (ad esempio immagini e video). La maggior parte del contenuto parla di fatti personali che gli utenti vogliono condividere con altre persone nella propria cerchia sociale. Tuttavia, altri utenti pubblicano spesso informazione più strutturata e complessa.

Il *crowdsourcing* è diventato uno degli argomenti più discussi degli ultimi anni. Il termine crowdsourcing generalmente fa riferimento all'esecuzione da parte di un gruppo di persone di un compito non automatizzabile (detto anche *task*). La crescita del tempo che le persone spendono online ha aumentato via via l'interesse che le persone mostrano nei confronti del crowdsourcing. Di conseguenza, molti lavori di ricerca hanno studiato a fondo aspetti riguardanti il crowdsourcing, sia creando politiche di partecipazione attiva degli utenti alla risoluzione di task, sia sfruttando i dati che gli utenti pubblicano tutti i giorni sul Web. Ci riferiamo a questi approcci come, rispettivamente, *active crowdsourcing* (cioè: partecipazione attiva di utenti motivati nell'esecuzione di task) e *passive crowdsourcing* (cioè: sfruttamento dello user-generated content per l'estrazione di informazione altrimenti non nota).

L'*active crowdsourcing* è il processo di esecuzione di task da parte un largo gruppo di persone, chiamati lavoratori. In questo scenario, si chiede ad un lavoratore umano di eseguire un compito specifico (chiamato *crowd task*), che solitamente è semplice da eseguire per il lavoratore, ma difficile da eseguire per una macchina. Nel contesto dell'active crowdsourcing, solo i compiti difficilmente automatizzabili vengono fatti eseguire da lavoratori umani. Di conseguenza, i crowd task vanno spesso a trattare dati incerti, non strutturati e non comprensibili da componenti automatiche. Sfortunatamente, non è ancora stato quantificato appropriatamente l'impatto che la risoluzione di un crowd task su dati incerti potrebbe avere sul grado di incertezza dei dati. Inoltre, come accade anche con l'impiego di risorse automatiche, anche l'impiego di lavoratori umani ha un certo costo, ed inoltre i lavoratori non sono sempre disponibili e potrebbero fornire risposte errate. Di conseguenza, è necessario progettare un approccio per la selezione dell'insieme di crowd task più promettente, dati alcuni vincoli (ad esempio, di tempo e di costo), così che l'esecuzione dei suddetti possa garantire una alta qualità del risultato.

Il *passive crowdsourcing* denota invece un approccio alternativo per sfruttare l'attività online degli utenti: esso richiede di analizzare grosse quantità di dati (resi pubblici dagli utenti stessi), per estrarre informazione sui comportamenti, sugli interessi e sulle attività svolte dalla popolazione dei social media. Ricercatori di diversi campi (ad esempio delle scienze sociali, economiche e di marketing) analizzano una grande varietà di dati creati dagli utenti per comprenderne il comportamento, per trovare nuovi trend nella società e per formulare politiche adeguate in risposta. Tuttavia, data la natura incontrollata della partecipazione degli utenti sul Web, una grande quantità di dati contiene informazione replicata, di bassa qualità ed irrilevante. Inoltre, il contenuto viene spesso replicato senza permesso: gli utenti copiano contenuto creato

da altri (anche se soggetto a copyright), lo rinominano e fingono di esserne gli autori.

In questa Tesi, proponiamo metodi per superare i problemi citati, sia nel campo dell'active crowdsourcing sia in quello del passive crowdsourcing, con l'obiettivo di massimizzare la qualità del risultato, anche in presenza di vincoli di budget e di tempo.

Contents

Chapter 1

Introduction

In the last years, social media have attracted millions of users and have been integrated in people's daily practices. They enable users to create and share content or to participate in social networking. Social networks are web-based services where individuals can create a virtual profile, list users with whom they share a connection, and visualize profiles made by others in the same system [Ellison et al., 2007]. Classical examples of these social applications are Facebook[1], Twitter[2] and Instagram[3]. A social network allows users to maintain a virtual copy of their real-world social graph, i.e., a representation of the relationships with other people (e.g., friends and family). However, these websites often encourage users to interact also with strangers, based on shared interests. Twitter is a typical example, where users follow (i.e., create a relationship with) other users just based on the interest in the content they publish. Some sites are thought for the general public, while others attract users based on common language, religious identity, ideas and nationality.

With the diffusion of mobile devices, a new form of social networking has started: mobile social networking. People with similar interests converse and connect with one another through their mobile devices (e.g., smartphones and tablets), which makes social networks more pervasive: in any moment, either at home or in mobility, users can connect to their profiles, check updates from their friends, publish fresh content.

User-Generated Content, i.e., the various forms of media assets publicly available and created by end-users, is published every day on the Web and mostly in social media at a massive scale, either in the form of textual documents (e.g., blog articles, posts on social networks, comments and discussion) or in the form of multimedia items (e.g., images and videos).

Most user-generated content is about personal lives and facts about users. However, users often publish more structured and complex information. For instance, when breaking-news events happen (e.g., earthquakes, shootings, attacks), people start to publish text, photos and videos massively, so as to describe the ongoing event to their own friends and acquaintances [Sakaki et al., 2010]. Sometimes the publishing rate is so powerful that content reaches users before the actual event hits: [The Washington Post, 2011] showed that tweets talking about earthquakes can travel faster than earthquakes themselves. As another example, when new products are released, expert users massively create product reviews that contain textual descriptions, images and videos commenting the product. These materials always generate great interest in consumers, who want to retrieve detailed information about products before buying them. For instance, after the release of iPhone6 in September 2014, a user realized that the released phone could be bended with a little pressure, and demonstrated his discovery with a video, reaching 53.4 millions of views in two weeks.

The booming interest on social networks multiplies the number of people that every day

[1]`http://www.facebook.com`
[2]`http://twitter.com`
[3]`http://instagram.com`

interconnect themselves on the Web. Somehow, this relates with human nature: humans are innately social and cooperate to achieve objectives [Porter, 2010]; they tend to form social circles (i.e., a group of socially interconnected people) and obtain benefits from them [Levine and Kurzban, 2006]. Social circles existed well before social networks were invented. People clustered in social circles in the real world (i.e., communities sharing the same interests, religions, ideas), and the information a person had access to was that flowing through her social circle. Information generated by other circles was not visible. Now, with the diffusion of new media (e.g., the Internet and social media), the situation is the opposite: the amount of available information has grown substantially, data coming from all circles is always available, and each social circle acts as a filter for the information delivered every day [Ma et al., 2011]. Social networks promote content coming from people close in one's circle, who usually share the same interests and ideas.

A direct benefit of real-world social circles is that they allow users to ask directly their friends for advice or opinions. A similar mechanism can be applied to online social networks, where data is automatically filtered according to the opinions coming from social circles, so that users are provided only with information that can potentially answer their needs. The benefit of this approach is that, while the number of advices and opinions that can be handled manually by a person is limited, the introduction of machines allows millions of items, covering a large social circle [Luz et al., 2014].

Socially filtered recommendations are an outstanding example of information retrieval implemented by the harmonic cooperation of humans and computer systems. Back in the 1960s, [Licklider, 1960] believed that machines and computers were just part of a scale which weighs humans on one side, and computers on the other [Luz et al., 2014]. According to his vision, humans and computers should work together with complementary roles: massive data computation is left to computers, while tasks that require some additional knowledge and reasoning are performed by humans. Recently, this idea was further developed by other authors [Gruber, 2008], with the introduction of the social Web and collective intelligence applications [Malone et al., 2009]. In these works, the authors argue that humans and computers have complementary abilities, and that users can act as computational units. Even nowadays, many tasks that are trivial for humans continue to challenge the most sophisticated computer programs. These tasks, due to their intrinsic complexity, cannot be fully automatized, and it is thus necessary to perform them manually. Typically they require some degree of creativity or common sense, plus some background knowledge [Singh et al., 2002, Chklovski, 2003]. Image annotation is a common example: tagging images requires the ability of recognizing objects and scenes in photos, which is typical of humans. Machines, on the other hand, if provided with a large dictionary of shapes, can only state if two objects are similar. This automatic decision is usually based on the comparison of key points (i.e., significant points in the image), which is obviously not robust to illumination, rotation and position change, and thus prone to errors.

For these reasons, *crowdsourcing* has gained increasing importance in the last years [Howe, 2008, Von Ahn, 2009, Quinn and Bederson, 2011]. The term "crowdsourcing" [Howe and Robinson, 2006] generally refers to the outsourcing of a non-automatable task to people.

The growth of the time spent online has led to a growth of interest in crowdsourcing. In this inception phase, most business crowdsourcing applications are developed in an ad-hoc manner; in the academy most work has focused on such aspects as mixed human-machine computational techniques and performance analysis. Several works have been developed, either making users actively participate in the resolution of tasks [Yuen et al., 2011, Luz et al., 2014] or exploiting data they generate and publish over the Web [Xintong et al., 2014]. We refer to these approaches as, respectively, *active crowdsourcing* (i.e., active participation of motivated users in task execution) and *passive crowdsourcing* (i.e., exploitation of user-generated content to extract useful information).

The common trait of active and passive crowdsourcing is that they rely on the coordinate work of humans and machines to solve a specific task meeting quality standards and respecting budget constraints. In active crowdsourcing humans are engaged explicitly and quality of work can be achieved with techniques that work a priori (e.g., task design). In passive crowdsourcing human work is exploited a posteriori, and optimization techniques work by filtering out irrelevant data to retain only information that can actually contribute to resolving the task.

Active crowdsourcing

Active crowdsourcing is the common notion of crowdsourcing found in the literature, which we qualify as *active* in order to distinguish it from the *passive* collection of contributions from user-generated content. Active crowdsourcing is the process of outsourcing tasks to a large group of people [Quinn and Bederson, 2011], called *workers*. In this scenario, human workers are asked to perform very specific tasks (called *crowd tasks*), which usually are easy to be solved by humans but hard to be solved by machines. An example of crowd task is image annotation, in which workers are provided with an image and are required to state whether the image contains a specific object.

Usually, crowd tasks are submitted to crowdsourcing platforms, where workers are paid with a fixed (typically monetary) reward, do not cooperate among themselves in solving tasks [Von Ahn and Dabbish, 2008], and can pick and execute the tasks they prefer. Workers generally tend to select tasks that require a short execution time, so as to maximize the reward. Single task executions are generally avoided because in this way a worker wastes more time in selecting the task than in executing it, reducing profit.

Work quality varies according to both task design and user motivation in performing the task. The simpler the task and the more motivated the user, the better the expected work accuracy. Several works in the literature focus on how to induce workers to answer tasks with the best possible accuracy: user motivation [Harris, 2011a,Mason and Watts, 2010,Kazai, 2010], clean task design [Grady and Lease, 2010,Yang et al., 2008], quality management policies [Sheng et al., 2008,Snow et al., 2008,Rashtchian et al., 2010,Nowak and Rüger, 2010] and cheating detection [Eickhoff and de Vries, 2011] are the most studied factors. A common approach used to ensure quality is redundancy with *majority voting* [Sheng et al., 2008,Snow et al., 2008,Nowak and Rüger, 2010]. This approach replicates tasks so as to collect multiple answers for the same task instance, and averages the collected answers to extract an aggregated opinion. This results in the reduction of erroneous answers: if the hypothesis that the majority of workers provides the correct answer holds, the damage caused by wrong answers is limited.

In the context of active crowdsourcing, only tasks difficult to be performed by a machine are submitted as crowd tasks. They are often based on uncertain data, since these data can hardly be processed by computers, due to their unstructured nature. Some famous examples of crowd tasks on unstructured data are: image tagging tasks [Nowak and Rüger, 2010] (in which workers are asked to detect a specific visual component in an image), OCR tasks [Von Ahn et al., 2008] (in which workers are asked to convert images of typewritten text into text), matching of complex structures [Zhang et al., 2013] (e.g., object matching, schema matching), fuzzy query answering [Franklin et al., 2011]. However, the concept of uncertain data is broad [Aggarwal and Yu, 2009]. Uncertainty may arise in different forms (e.g., uncertain attribute values of tuples in databases, uncertainty on data correctness, uncertainty on the existence of tuples in real world) and may affect the results of different applications (e.g., uncertain data mining, uncertain query answering, classification).

Several works have shown preliminary evidence that the application of crowdsourcing techniques can improve the performance of state-of-the-art automatic algorithms for uncertain data processing [Bozzon et al., 2012]. Since automatic uncertain content analysis components (e.g., multimedia content processors [Nowak and Rüger, 2010], uncertain data validators [Zhang et al.,

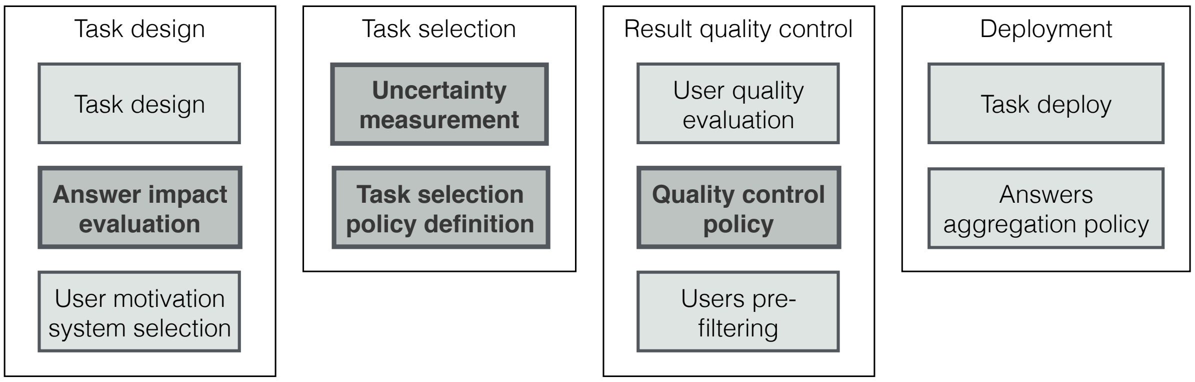


Figure 1.1: Overview of the steps of active crowdsourcing. In bold, the components of the pipeline we will optimize in this Thesis

2013]) have often low accuracy, they introduce errors in the output. Thus, humans are asked to replace or integrate them, hopefully increasing the ultimate accuracy. Similarly to the use of machine resources, which cost, also human computational resources are not freely available in any amount. The larger the number of tasks to perform, the larger the amount of money the requester has to pay to ensure the desired number of workers engage and deliver the requested level of accuracy. Thus, it is necessary to devise a strategy to minimize the number of posed questions, while guaranteeing a high result quality [Parameswaran et al., 2011].

Figure 1.1 shows the basic steps of active crowdsourcing. We summarize them as follows:

- *Task design*: design of crowdsourcing tasks and evaluation of the impact of a worker answer;
- *Task selection*: selection of the set of tasks that minimizes the underlying residual data uncertainty;
- *Result quality control*: implementation of quality control policies to guarantee quality of the result;
- *Deployment*: publication of tasks on crowdsourcing platform and collection of contributions.

Passive crowdsourcing

The term *Passive crowdsourcing* denotes an alternative approach for leveraging the online activity of users for task resolution, which amounts to analyzing a huge amount of publicly available contents, to extract information about behaviors, interests and activities of the social media population. Researchers from different fields (e.g., social science, economy and marketing) analyze a variety of user-generated datasets to understand human behaviors, find new trends in society and possibly formulate adequate policies in response [Xintong et al., 2014]. Two applications of these techniques are the prediction of user interests in products (e.g., given the preferred dress colors of past seasons, predict which color will be chosen by users in the next season) and the tracking of felonious behaviors (e.g., track the insurgence of extremist groups, which often use Twitter as a platform for the diffusion of messages and discussions [Lynch et al., 2014]).

However, due to the uncontrolled nature of users' participation on the Web, the huge mass of available data contains replicated information, as well as low quality or irrelevant content. Thus, in recent years, content relevance has emerged as a relevant topic in the context of passive crowdsourcing [De Choudhury et al., 2011b, Ghosh et al., 2013, Lerman and Hogg, 2010, Bakshy et al., 2011]. When data need to be analyzed (either to extract relevant content to be shown to

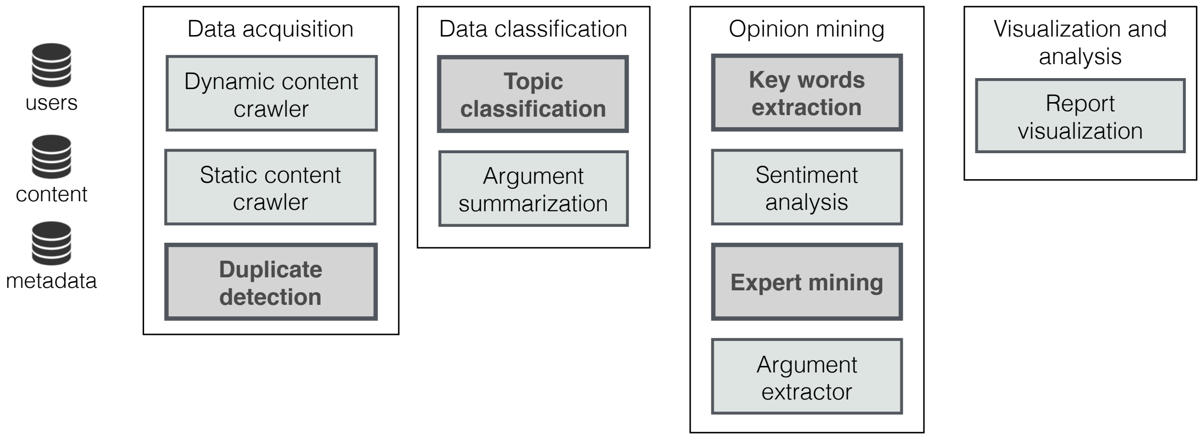


Figure 1.2: Overview of the steps of passive crowdsourcing, inspired by [Charalabidis et al., 2013]. In bold, the components of the pipeline we will optimize in this Thesis

users or to extract relevant information to build reports) it is necessary to pre-filter them so as to discard irrelevant content and analyze only pertinent data. Several works in the literature apply machine learning techniques and graph structure analysis to extract relevant content from social media [Sun and Ng, 2013,Silva et al., 2013,Schenkel et al., 2008,Gou et al., 2010,Yin et al., 2010]. Their analysis reveals relationships between content and content creators: influential users (i.e., the ones with the largest visibility in the network) are the ones producing more exploitable content (e.g., original content characterized by novelty and relevance to a topic).

Figure 1.2 shows the basic steps of passive crowdsourcing. We summarize them as follows:

- *Data acquisition*: definition of a crawling architecture that retrieves user-generated content from social media;
- *Data classification*: classification of data so as to recognize topic-related content;
- *Opinion mining*: analysis of content to extract significant information;
- *Visualization and analysis*: visualization of the extracted reports.

1.1 Problem statement

In this Section we present the problems related to the contexts of active and passive crowdsourcing, which will be addressed in this Thesis.

1.1.1 Active crowdsourcing

We restrict the range of active crowdsourcing applications to those addressing a specific, yet extremely relevant task: reducing the uncertainty inherent to some input data to improve its usability, e.g., to build predictive models or to extract information actionable by human decision-makers. Uncertainty reduction can not be fully addressed with automated tools alone, due to such factors as the fuzzy definition of the data usage goal, the probabilistic nature of the resolution algorithms, or even the unavailability of data processing algorithms.

Therefore, uncertain data processing lends itself as a natural playground for active crowdsourcing applications, where machines are employed to make a first processing of data at a very large scale, and humans are called to the rescue after such initial screening, to resolve the residual uncertainty of a smaller, hopefully significant, pool of data.

However, several questions still need to be addressed, related to the interplay between data uncertainty, the nature of the task submitted to the crowd and the uncertainty on crowd workers behavior (e.g., human errors, cheating, fatigue).

First, an appropriate modeling of the impact of a crowd task answer on uncertain data needs to be defined. Answers collected by the crowd help requesters in gaining unknown information, which in turn can be exploited to reduce the underlying data uncertainty. However, it is not yet clear how the requester can take advantage of this information once it has been acquired.

Second, requesters need to devise an approach for the selection of the best candidate set of tasks to submit to the crowd under some fixed constraints. Several tasks that involve different data may contribute differently to uncertainty reduction, since some of them may be more informative than others. Therefore, an efficient and effective strategy that takes into account constraints of latency and cost has to be designed.

Third, task answers may be affected by low quality, due to either complex task design, cheating behaviors or lack of user motivation. Consequently, quality assurance procedures are necessary to guarantee an appropriate result quality level.

Problem Statement:
Given a crowd task that aims at reducing the uncertainty of a data corpus, devise a crowdsourcing strategy that ensures a maximal reduction of residual uncertainty, taking into account constraints, such as latency, cost and result quality

1.1.2 Passive crowdsourcing

Passive crowdsourcing is the approach that considers all users of social media as potential crowd workers and thus aims at extracting useful information from the massive amount of user-generated content available online.

Like in active crowdsourcing, also passive crowdsourcing has problems of quality and uncertainty of results and is subject to time and processing cost constraints. However, the challenges posed by passive crowdsourcing and the techniques to face them differ from those found in active crowdsourcing.

First, a problem related to user-generated content is that it contains a large percentage of redundant, low-quality and non-relevant information. Nevertheless, a small part of it is highly informative and provides clues about who are the experts and which is the most relevant content for each topic. A simple approach that requires to crawl the whole content and filter out the irrelevant one would ensure maximum recall, but it would not be feasible, due to the huge quantity of data to be processed. A manual validation would also be infeasible, because of its high cost (although some real scenarios such as the study of socially mediated civil war in Syria via social networks [Lynch et al., 2014] may be so important to justify such a high level of investment). Thus, a way of automatically discarding irrelevant material and retaining only relevant content has to be devised.

Second, content is often replicated maliciously: users copy content created by others (and often subject to copyright laws), rename it and pretend they are the authors of the corresponding original content. Causes of this behavior may be found in the interest of receiving visibility over the Web, where other users visualize, comment and share the copies as if these were original. As a result, users fake their expertise in one topic, and the volume of required resources to store all these copies exceeds the capacity of the processing system available. Thus, a mechanism that allows content analysts to discard all the copies and identify the original content is needed.

Problem Statement:
Given a passive crowdsourcing goal, devise techniques for the automatic analysis of user-generated content, so as to identify relevant entities (e.g., content and users) about a specific topic respecting time and cost (e.g., storage) constraints

1.2 Research questions

In this Section, we formulate the research questions of this Thesis in order to address the aforementioned problems.

1.2.1 Optimizing active crowdsourcing

In this Section we enumerate the research questions related to the active crowdsourcing context. Our contributions will concern the steps indicated in bold in Figure 1.1. All the other steps will employ known techniques in the state of the art. In the following, we list the essential research questions, together with the architecture steps the associated research work will improve.

Research question 1. *How can uncertainty of structured data be modeled?*

Different forms of uncertainty may affect data, and different crowdsourcing techniques may be designed so as to reduce the uncertainty of the underlying data. However, in order to devise a proper uncertainty reduction strategy, a way of modeling and measuring data uncertainty is needed. Answering this question will improve the performance of the *Uncertainty measurement* step in Figure 1.1.

Research question 2. *How do crowd task answers impact on data uncertainty?*

Task design is a crucial dimension of crowdsourcing. Small tasks that involve few concepts are preferred to tasks that require workers to do complicate reasoning. Moreover, a good motivation mechanism (typically monetary) assures that workers will be prompted in providing good answers for a large amount of tasks. Nevertheless, a good task design has to be matched with a good integration strategy for the information coming from workers' answers. Answers provide information that was unknown before the related tasks were answered. If harnessed correctly, this information can greatly reduce the uncertainty on the underlying data. Thus, it is necessary to design a methodology for the exploitation of such information. Answering this question will improve the performance of the *Answer impact evaluation* step in Figure 1.1.

Research question 3. *How do task selection and budget constraint affect uncertainty on structured data?*

Several tasks can be posted to workers, but some of them are more informative than others, and some others are redundant. Requesters usually have constraints on task execution, such as latency or cost, hence asking all the available tasks does not suit the requirements. Therefore, the success of a crowdsourcing campaign depends not only on task design, but also on task selection. We argue that a good solution should accurately select the tasks that assure the maximum expected uncertainty reduction. Answering this question will improve the performance of the *Task selection policy definition* step in Figure 1.1.

Research question 4. *How does worker quality impact on crowd tasks effectiveness?*

Workers are not oracles, and thus can provide wrong contributions. When collecting answers from workers, requesters usually apply replication policies to retrieve multiple answers for the same task instance, which are in turn averaged out to obtain a unique opinion. However, this replication has its own costs, since multiple workers have to be paid to execute the same task. Thus, more refined techniques that weigh a worker's answer impact according to her accuracy may be applied, so as to guarantee result quality while still asking a small number of tasks. Answering this question will improve the performance of the *Quality control policy* step in Figure 1.1.

1.2.2 Optimizing passive crowdsourcing

In this Section we enumerate the research questions related to the passive crowdsourcing context. Our contributions will concern the aspects indicated in bold in Figure 1.2. All the other steps will employ known techniques in the state of the art. In the following, we list the essential research questions, together with the architecture steps the associated research work will improve.

Research question 5. *What seed queries can be used to initialize topic-related information retrieval?*

Crawlers are components used to retrieve user-generated data from social media. These components work by specifying a set of search criteria to follow, so that any content that matches at least one of the search criteria is retrieved. However, user conversations are dynamic and thus fixing a set of predefined search criteria limits the information a crawler can retrieve. Hence, a dynamic modification of search criteria is needed, so that search criteria change in response to what is currently discussed by users in social media. Answering this question will improve the performance of the *Key words extraction* step in Figure 1.2, which will in turn retrofit positively on the Data Acquisition step.

Research question 6. *What type of data can be analyzed to improve the accuracy of topic-related information retrieval?*

Simple validation techniques used to state whether a content is topic-related require manual validation, where annotators (generally experts) evaluate content to state whether it is credible and relevant. However, manual validation has a huge drawback: since the amount of data is huge and experts are costly, manually evaluating user-generated content requires a large amount of money and a significant waste of time. Thus, an automatic solution that employs machine learning techniques has to be applied to classify content as either relevant or not relevant for a topic. In the literature, several works are based on text-classification only. However, multimedia content could provide further insights on content relevance, and thus we argue that a proper solution should consider multimedia content classification too. Answering this question will improve the performance of the *Topic classification* step in Figure 1.2.

Research question 7. *How are influential content producers defined and identified?*

Influential users are expert users that produce relevant and interesting content for a specific topic. Influential users change over time and may produce topic-unrelated content. Thus, it is necessary to devise a strategy to define what is influence and track influential users over time. This strategy should take into account the amount of original, topic-related content produced by those users: the larger the amount of created relevant content, the higher the influence. However, when content is replicated maliciously, the expertise and influence degree of a user are wrongly estimated, since copying content from others increases the amount of original content attributed to that user. Thus, given a set of duplicates, a strategy for the recognition of the original content (and the original author) is necessary. Answering this question will improve the performance of the *Expert mining* and *Duplicate detection* steps in Figure 1.2, which will in turn retrofit positively on the Data Acquisition step.

Research question 8. *What is the impact of considering content producers' influence level on the accuracy of topic-related information retrieval?*

According to works in the literature, influential users produce relevant content. Thus, we argue that tracking content produced by influential users and consequently using it to change crawlers' search criteria should improve the accuracy of the retrieval system. However, it is necessary to define an evaluation metrics for the impact of influence level on the accuracy of the retrieved results. Answering this question will improve the performance of the *Expert mining* step in Figure 1.2, which will in turn retrofit positively on the Data Acquisition step.

1.3 Contributions

The contribution of this Thesis are as follows:

- We present a framework for the reduction of data uncertainty via crowdsourcing, with specific interest in the top-K query context [Ilyas et al., 2008]. In this context, uncertainty on data induces multiple possible orderings. Thus, crowdsourcing is used to gather information about the relative order of objects, which in turn is used to discard incompatible orders. Contributions can be summarized as follows:
  - we model data uncertainty (this answers to Research Question 1);
  - we design crowd tasks and evaluate worker answers impact on data uncertainty (this answers to Research Question 2);
  - we give a formal definition of Uncertainty Resolution problem;
  - we define strategies to solve the Uncertainty Resolution problem by selecting the set of B questions that ensures the maximum (expected) uncertainty reduction (this answers to Research Questions 3 and 4).
- We present a pipeline for the automatic analysis of user-generated content, with the objective of recognizing relevant content and influential users for a specific topic. Contributions can be summarized as follows:
  - we define a topic-related real-time crawling system for multimodal data, which automatically discards non-relevant content;
  - we mix multimedia content analysis and text content analysis to find relevant, topic-related content (this answers to Research Question 6);
  - we extract statistics from the crawled relevant content, such as the most used topic-related terms, to be used as content search criteria (this answers to Research Question 5);
  - we identify and track influencers for the selected topic (this answers to Research Question 7);
  - we evaluate the impact of considering influential users on the accuracy of topic-related information retrieval (this answers to Research Question 8).
- We study the detection of original content in a set of duplicates, with specific interest in duplicated videos. Contributions can be summarized as follows:
  - we define video similarity graph, where nodes are videos and (directed) edges (a, b) indicate candidate copy relationships (i.e., b could be a duplicate of a);
  - we translate video similarity graph in video phylogeny, where roots identify original content (this answers to Research Question 7).

1.3.1 Summary of findings

Active crowdsourcing

In this Thesis we prove that, given a set of uncertain and incomplete data, active crowdsourcing techniques help in reducing the underlying uncertainty, and that an accurate selection of crowd tasks allows the requester to save budget and maximize the expected uncertainty reduction.

Specifically, we introduced several algorithms that help in reducing uncertainty under budget constraints:

- some algorithms require the full materialization of the space of possible orderings, so as to identify the best crowd task among all the available ones;
- some other algorithms avoid the full materialization of the space of possible orderings, which may be huge, selecting the best crowd task among the subset of available tasks.

The best crowd task is defined as the task that allows us to gain the highest expected uncertainty reduction. In order to identify it, we devise four different uncertainty metrics to measure the uncertainty of the space of possible orderings, which take into account the probability of each ordering and the structure of the space of possible orderings: spaces that contain a large number of highly different orderings are more uncertainty than spaces with a small number of similar orderings.

Our findings are as follows:

- the algorithms that require the full materialization of the space perform better than incremental algorithms, since they have a larger choice of crowd tasks to be asked to workers, and thus can identify the most promising questions in early stages (saving up to 50% of budget), although requiring a long time to materialize the whole space (even hours in case of trees containing thousands of orderings);
- in general, our algorithms allow one to save up to 80% of budget compared to techniques that choose randomly crowd tasks;
- the state-of-the-art uncertainty metrics that consider only ordering probabilities (e.g., entropy) model uncertainty worse than metrics that consider also the structure of the space of possible orderings; thus, algorithms using state-of-the-art uncertainty metrics have low performance and require larger budgets to eliminate uncertainty.

Passive crowdsourcing

In this Thesis we show that a multimodal (text + visual content) automatic classification process of user-generated content on social media allows us to discover influential users on a selected topic, that in turn generate high-quality content for the same topic.

First of all, multimodal classification proves to be more accurate than text classification only, since it captures aspects of content that are hidden in multimedia items (i.e., images, videos and audio) and that could give some extra hints on content relevance. Specifically, the accuracies of multimodal classification and text classification are respectively 82% and 73% on real-time Twitter data and 87% and 89% on offline, manually filtered data.

Furthermore, we show that an elaborate influence metrics that mixes different aspects such as originality, activity and communicativeness of users is effective: users retrieved with this influence metrics produce a large amount of relevant content (i.e., 76% of produced content is topic-related) and are particularly influential for the selected topic. On the other hand, simpler state-of-the-art measures based on popularity and activity retrieve users that produce a small amount of relevant content (i.e., 25% of produced content is topic-related).

Finally, it is clear how users that produced original content are potentially influencers for the selected topic, while others that mainly replicate content should have a lower influence score.

1.4 Outline of the Thesis

The rest of the Thesis is organized as follows.

Chapter 2 defines the context of the work in terms of crowdsourcing techniques, mathematical background, uncertain data features and machine learning. For each topic, we describe the concepts that will be used throughout the following Chapters.

Chapter 3 discusses an application of active crowdsourcing techniques in the context of uncertain structured data, with particular emphasis on top-K query applications, where the available information is not sufficient to characterize a unique ordering for the underlying data. In this Chapter, we define the space of possible orderings for a set of uncertain structured data. Then, we propose a metrics to measure the uncertainty of that space. Finally, we devise several approaches to select the set of questions that, if asked to the crowd, minimize the uncertainty of the space of possible orderings. The approaches consider also the possibility that crowd answers may be incorrect.

Chapter 4 introduces the automatic procedure for the extraction of a set of influencers for a specific topic from a microblogging platform. First, we define the architecture of the crawling pipeline used to gather potentially topic-related content. Then, we introduce the filtering chain used to filter out irrelevant content, which considers both textual and multimedia content.

Chapter 5 explores the techniques useful for extracting ancestry relationships between user-generated content, so as to identify malicious behavior of users that copy content from other users. First, we give a definition of content similarity. Then, we build a graph similarity that shows similarity relationships between multiple multimedia objects generated by users. Finally, we devise a strategy for the extraction of ancestry relationships from the similarity graph.

Chapter 6 presents the experimental evaluation that covers Chapters 3, 4 and 5. Here, the performance of the proposed techniques is assessed, and results show the advantages of applying crowdsourcing techniques to the presented scenarios.

Chapter 7 presents the related work for the proposed scenarios.

Finally, in **Chapter 8** conclusions are drawn and future work directions are presented.

Chapter 2

Background

In this Chapter, we present the context of the work in terms of crowdsourcing techniques, mathematical background, uncertain data features and machine learning basic techniques. These concepts will be used throughout the following Chapters.

2.1 Crowdsourcing: active and passive contexts

In the following, we introduce some key concepts on active and passive crowdsourcing, which will be used in the following Chapters.

2.1.1 Social media: a classification

Recently, the Web has shifted towards user-driven technologies such as blogs, social networks and video-sharing platforms [Smith, 2009]. According to Forrester Research, 75% of Internet surfers use social media in 2008 by reading blogs, contributing reviews to shopping sites or joining social networks [Kaplan and Haenlein, 2010]. As a result, social media have gained large popularity, producing an unprecedented mass of data and generating large participation on the Web. People are prone to the participation on conversations, share information about their relations with others and their lives, cooperate in the production of new content.

This shift towards social media was caused by a shift in the technology used to build Web applications, generally referred to as Web 2.0 [O'reilly, 2009]. With this term, we describe a new way in which software developers and end-users started to utilize the Web: as a platform whereby content and applications are no longer created and published by individuals, but instead are continuously modified by all users in a participatory and collaborative fashion [Kaplan and Haenlein, 2010]. This helps users in the creation of User-Generated Content [Krumm et al., 2008], i.e., the various forms of media content that are publicly available and created by end-users. Content is considered as user-generated if it is compliant to the following criteria: *i)* it is published either on a publicly accessible website or on a social networking site accessible to a selected group of people; *ii)* it shows a certain amount of creative effort; *iii)* it has been created outside of professional routines and practices. Hence, social media is a group or Internet-based applications that build on the ideological and technological foundations of Web 2.0, and that allow the creation and exchange of User-Generated Content.

In the following, we provide a classification of social media based on the taxonomy by [Kaplan and Haenlein, 2010], as shown in Figure 2.1.

Collaborative projects

Collaborative projects enable the joint and simultaneous creation of content by many end-users. Collaborative projects are differentiated between wikis (i.e., websites which allow users to add, remove and change text-based content) and social bookmarking applications (i.e., websites that

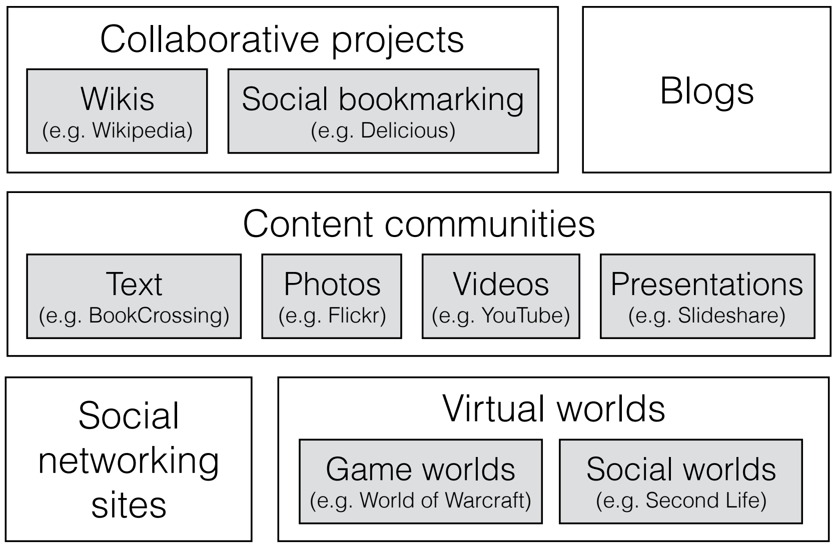


Figure 2.1: Social media classification

enable the group-based collection and rating of Internet links or media content). The main idea underlying collaborative projects is that the joint effort of many actors leads to a better outcome than any actor could achieve individually.

Two classical examples of collaborative projects are Wikipedia[1] and Delicious[2].

Blogs

Blogs are special types of websites that usually display date-stamped entries in reverse chronological order. They are a modern version of personal web pages, and can come in a multitude of different variations, from personal diaries describing the author's life to summaries of all relevant information in one specific content area.

Content communities

The main objective of content communities is the sharing of media content between users. Content communities exist for a wide range of different media types, including text (e.g., BookCrossing[3]), photos (e.g., Flickr[4]), videos (e.g., YouTube[5]) and PowerPoint presentations (e.g., SlideShare[6]).

Social networking sites

Social networking sites are applications that enable users to connect by creating personal information profiles, inviting friends and colleagues to have access to those profiles, and sending e-mails and instant messages between each other.

The most famous social network is Facebook[7].

[1]`http://en.wikipedia.org`
[2]`https://delicious.com`
[3]`http://www.bookcrossing.com`
[4]`http://flickr.com`
[5]`http://youtube.com`
[6]`http://slideshare.net`
[7]`www.facebook.com`

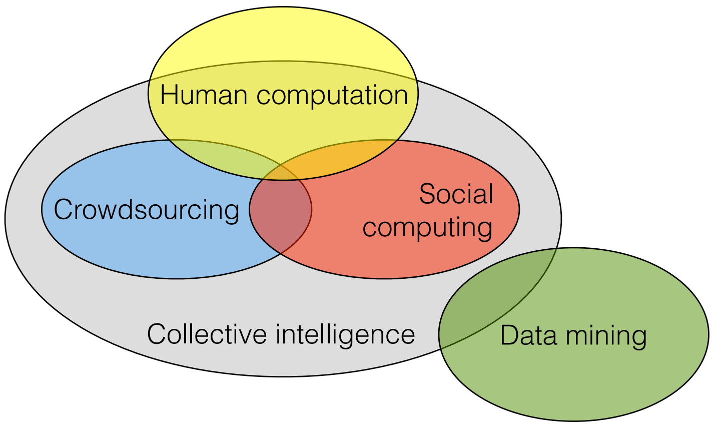


Figure 2.2: Human computation taxonomy

Virtual worlds

Virtual worlds are platforms that replicate a three-dimensional environment in which users can appear in the form of personalized avatars and interact with each other as they would in real life.

Virtual game worlds. A virtual game world requires its users to behave according to strict rules in the context of a massively multiplayer online role-playing game. An example of virtual game world is World Of Warcraft.

Virtual social worlds. A virtual social world allows inhabitants to choose their behavior mode freely and essentially live a virtual life similar to their real life. Virtual social world users appear in the form of avatars and interact in a three-dimensional virtual environment. However, there are no rules restricting the range of possible interactions, except for basic physical laws such as gravity. The most prominent example of virtual social world is the Second Life application.

2.1.2 Human computation: taxonomy

The idea of humans and machine working together to achieve a common goal was envisioned in 1960 in the work [Licklider, 1960]. However, only in the last years, the words *human computation* and *crowdsourcing* put the idea of cooperation between humans and computers in practice. This shift was mainly caused by the evolution of the Web from a publishing platform, where people were interested in publishing content or accessing to content published by others, to a collaborative tool, in which users share opinions, cooperate in the production of content and the execution of tasks, play games and participate to communities.

In the following, we present a taxonomy for human computation systems, inspired by the work [Quinn and Bederson, 2011].

Human computation

The modern use of the term *human computation* is inspired by the work [Von Ahn, 2009], where human computation is defined as:

> *a paradigm for utilizing human processing power to solve problems that computers cannot yet solve.*

Human computation can be described by two distinctive features: *i)* the treated problems fit the general paradigm of computation, and as such they could be someday solved by machines rather than humans; *ii)* the human participation is controlled by the computational system, who manages the creation of tasks and the retrieval of answers by users.

However, human computation do not include many of the activities that users perform every day on the Web: online discussions and creative projects do not fit into the paradigm of human computation, since they require creativity and innovation, which does not play a role in a predefined plan to solve a computational problem.

Crowdsourcing

The term crowdsourcing was coined by Howe and Robinson [Howe and Robinson, 2006]:

> *Crowdsourcing represents the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively), but is also often undertaken by sole individuals. The crucial prerequisite is the use of the open call format and the large network of potential laborers.*

In the following, other works gave alternative definitions. Brabham [Brabham, 2008] stated that crowdsourcing is an online, distributed problem-solving and production model. Doan [Doan et al., 2011] defined crowdsourcing as a system that enlists a crowd of humans to help solve a problem defined by the system owners, and in doing so, it addresses the challenges of recruiting and retaining users, defining which contributions can be made by users, combining these contributions and evaluating user performance [Luz et al., 2014]. More recently, the work [Estellés-Arolas and González-Ladrón-de Guevara, 2012] integrated older definition in a new one, where crowdsourcing is seen as an online activity where individuals, institutions, companies propose to a group of individuals the voluntary undertaking of a task. These individuals, also called workers, have different knowledge and are heterogeneous. Moreover, according to this definition, the task has mutual benefit: the workers perform the task for the requester (i.e., the individual submitting the task), while the task brings knowledge, money or entertainment to the workers.

There is some overlap between human computation and crowdsourcing [Quinn and Bederson, 2011]: this intersection represents those applications that could reasonably be considered as replacements for either traditional human roles or computer roles. For example, translation is a task that can be done either by machines (when speed and cost are the priority) or by professional translators (when quality is the priority).

Crowdsourcing has two separate branches: active crowdsourcing, where users answer actively to tasks posted on crowdsourcing platforms by requesters (Section 2.1.3), and passive crowdsourcing, where user-generated content is exploited so as to extract interesting insight about specific topics and contexts (Section 2.1.4).

Social computing

Blogs, wikis and online communities are examples of social computing [Parameswaran and Whinston, 2007]. Their scope is broad, although it always include communication between humans that is mediated by technology.

The key distinction between human computation and social computing is that social computing facilitates natural human behavior with technology, while participation in a human computing system is primarily directed by the system itself.

Data mining

Data mining is the application of specific algorithms for extracting patterns from data [Fayyad et al., 1996]. These algorithms do not constitute human computation, since they do not necessarily require the collection of data, while human computation does. Moreover, challenges common to human computation systems (i.e., resistance to cheating, user motivation etc.) do not apply to the context of data mining.

Collective intelligence

Collective intelligence has a broad meaning, which is: people can accomplish great objectives when working together. Thus, collective intelligence is a superset of social computing and crowdsourcing, because both are defined in terms of social behavior. Moreover, some data mining applications benefit from groups.

The key distinction with human computation is that collective intelligence requires a group of people working together, while users may work individually when performing tasks in a human computation system.

2.1.3 Active crowdsourcing

Active crowdsourcing (generally referred to as crowdsourcing) is the process of obtaining services, ideas and information by soliciting active contribution from a large group of people, rather than from employees or suppliers.

Active crowdsourcing in the state of the art

Crowdsourcing can be seen from two different perspectives: a business domain-specific perspective, and a technical domain-independent perspective [Luz et al., 2014]. Several works in the state of the art try to solve problems related to these perspectives. From a business perspective, there is great interest on accomplishing specific business tasks efficiently and effectively in terms of time and monetary costs. Thus, works in this area focus on user motivation and quality-control aspects. From a technical perspective, the emphasis is on methods, techniques and frameworks for solving problems that machines are not able to solve. Thus, works in this area focus on creation and deployment of mechanisms that efficiently and effectively facilitate crowdsourcing and human computation process.

Crowdsourcing platforms

With the term crowdsourcing platform we generally indicate an online platform on which workers perform microtasks. A microtask is a small task which is simple to be executed by a person, but difficult for a machine. Tasks are usually created to perform those small jobs for which there does not exist an algorithm that can perform them efficiently, but that can be completed easily and reliably by humans. In return, each worker receives a small amount of money for each executed task. A classical example of microtask is the one in which workers are asked to classify a set of images as belonging to a class or not (e.g., discriminate between images containing a dog and images not containing a dog).

We call *explicit crowdsourcing* crowdsourcing activities performed via these platforms: during these activities workers are aware that they are working towards solving specific tasks [Luz et al., 2014].

Worker selection. Workers usually belong to at least one community and may be filtered according to different characteristics such as their profile information, demographics (e.g., country of residence) and expertise. Thus, six types of worker selection strategies can be identified: workers can be selected either from multiple communities or from a single community; filtering can be performed based on either assessment, or expertise test, or profile characteristics, or demographic characteristics. Assessment strategies are either manual (i.e., performed by the requester) or automatic (i.e., performed by the crowdsourcing systems depending on a set of attributes) [Luz et al., 2014].

Worker demographics. Some authors [Downs et al., 2010, Ross et al., 2010, Ipeirotis, 2010] studied the demographics of these platforms, to discover that the population is always changing over time, and that professionals, students and non-workers tend to take tasks more seriously than financial workers, hourly workers and other workers.

Examples of crowdsourcing platforms. In the following, we list the most known crowdsourcing platforms. Amazon Mechanical Turk[8] enables requesters to post tasks (called HITs, i.e., Human Intelligence Tasks), which are performed by human workers. Workers can browse among existing tasks and complete them for a small monetary payment (decided by the requester). Crowdflower[9] is another platform where workers are asked to perform microtasks such as labeling, data cleaning, data categorization, sentiment analysis, transcription. Samasource[10] is a nonprofit organization whose mission is to alleviate poverty by connecting unemployed youth and women in impoverished countries to digital work. A similar company is Mobileworks[11], which offers online works to underemployed communities around the world. Microtask[12] is a Finnish company with a private crowd where requesters can submit groups of small tasks. Since the crowd is private, Microtask assures the quality of its workers.

Motivation of users

One of the challenges of crowdsourcing (and human computation) is finding a way to motivate people to perform tasks. Unlike traditional jobs, which almost always pay with money, human computation workers may be motivated by a number of factors [Quinn and Bederson, 2011], from providing enjoyment and relying on altruism, to receiving monetary rewards [Faradani et al., 2011].

[Harris, 2011a] found that financial incentives actually encourage quality if the task is designed appropriately, although [Mason and Watts, 2010] proved that increased financial incentives increase the quantity, but not the quality, of work performed by crowdsourcing workers. [Kazai, 2010] found that low pay conditions result in increased levels of unusable and spam answers, and [Moreno et al., 2009] concluded that question-answering sites should function better with both long-term and short-term rewards. However, although monetary crowdsourcing incentive is dominant, some crowdsourcing systems do not offer monetary rewards to their workers: attention (e.g., number of downloads) is an important driver of contribution [Huberman et al., 2009], while altruism, learning and competency are frequent motivations for community question answering top answerers to participate [Nam et al., 2009].

In the following, we list all the types of incentives works in the state of the art used to induce workers to perform crowd tasks.

Money. Financial rewards are the easiest way to recruit workers. These are usually gained in the form of money, gift certificates, or virtual currency. Unfortunately, in order to gain more, workers are somehow tempted to cheat, and this is also caused by the fact that they are generally working as anonymous workers. Thus, a way of performing quality control needs to be integrated in the system.

Payment in this context has been the subject of some criticism due to the creation of cheap labour marketplaces [Harris, 2011b]. In particular, the current monetary rewards are not sufficient to be a primary source of income, and often they are not enough to serve as a motivator [Luz et al., 2014, Mason and Watts, 2010, Paolacci et al., 2010].

[8]`https://www.mturk.com/mturk/welcome`
[9]`http://www.crowdflower.com`
[10]`http://samasource.org`
[11]`https://www.mobileworks.com`
[12]`http://www.microtask.com`

Altruism. Altruism could motivate people to perform tasks. However, this is feasible only if workers think the problem is important and interesting.

Enjoyment. By making a task entertaining, it is possible to engage humans to do tasks that contribute to a computational goal (see Section 2.1.3 for games with a purpose).

Reputation. Where the problem is associated with an organization of some prestige, human workers may be motivated by the chance to receive public recognition for their efforts.

Implicit work. It is possible to hide a computational task in the activity users are already performing. ReCAPTCHA is a famous example of this approach [Von Ahn et al., 2008], in which users are asked to perform an OCR task on a photo of old books and newspapers.

Task design

The work [Alonso and Lease, 2011] gives some generic tips for task design: *i)* experiments should be self-contained; *ii)* instructions for task must be short and simple, brief and concise (too complex tasks or ambiguous instructions for workers to understand will generate poor quality answers); *iii)* if a similar task has been published, there is no need to replicate it; *iv)* always ask for feedback in an input box; *v)* highlight important concepts on the user interface.

Many workers complete tasks in multiple batches, introducing effects on training or fatigue, and thus it is necessary to ensure that each experiment involves a different set of workers in order to increase output accuracy [Grady and Lease, 2010]. Moreover, most workers become inactive after only a few submissions, others keep attempting tasks, and others tend to select tasks with higher expected rewards [Yang et al., 2008].

Several types of tasks can be executed by humans instead of machines. In particular, a *voting task* is a task in which the user has to select the correct answer among a set of possible answers. Some examples are: geometric reasoning tasks [Heer and Bostock, 2010] (i.e., the ability to interpret and reason about shapes), named entity annotation [Finin et al., 2010] (i.e., identification and categorization of textual references to objects in the world), opinions [Mellebeek et al., 2010] (i.e., subjective preferences gathering), relevance evaluation [Alonso et al., 2008] (i.e., determination of document relevance for a topic), natural language annotation [Akkaya et al., 2010] and spam identification.

Handling noisy workers

Since workers are not oracles and can introduce errors while answering to tasks, it is necessary to consider worker quality while retrieving answers from the crowd.

Majority voting. Several works in the state of the art try to overcome the problem of noisy workers (i.e., workers that answer wrongly to the submitted questions) by applying very simple techniques such as majority voting, which requires that a single task is replicated and performed by multiple workers. All the answers are aggregated to find the most recurrent one, which becomes the final answer for that task [Sheng et al., 2008, Snow et al., 2008, Nowak and Rüger, 2010]. Task replication improves data quality at low cost [Sheng et al., 2008], especially when answers are noisy. [Snow et al., 2008] proved that it is required to collect an average of four non-expert labels to emulate expert-level answer quality, especially in labeling tasks. Moreover, when different annotators judge the same data, the inter-annotator agreement among different annotators can ensure quality [Nowak and Rüger, 2010].

Compute worker's error probability. However, other works apply more refined techniques to model the noise on the answer. Several of them, such as [Zhang et al., 2013], assume to know the error probability of the user, and act accordingly, making the answer effect more shallow as the error probability grows. Still, this approach has one problem, which is how to compute the error probability for the set of workers. In fact, all the workers are different and act differently when the task difficulty and the number of tasks change. Thus, works such as [Joglekar et al., 2013] and [Venetis and Garcia-Molina, 2012] propose a way of estimating the error probability from data. Here, a probability estimate for each worker is computed depending on the number of disagreements the worker had with other worker answers: the larger the number of disagreement, the larger the error probability.

Compute a score for each worker. Other types of work compute a quality score for each worker while still gathering answers. An example of these works is [Marcus et al., 2012]. Here, the quality score is proportional to the number of answers by the worker that are close to the average of all the answers gathered by the crowd, i.e., if the worker deviates from the average answer, then she is a spammer. The work proposes also an algorithm that assures robustness to coordinated attacks by a set of workers.

This is similar to the idea of computing a reputation score for the user. In Amazon Mechanical Turk, requesters can require a minimum quality level for the workers, and some other qualifications (e.g., language skills). When cheating is detected, the reputation reduces. Finally, when the reputation is too low, the worker is put in a blacklist [Heimerl et al., 2012]. [Allahbakhsh et al., 2012] proposes a reputation management framework which adequately takes into account the values of the tasks completed, the trustworthiness of the assessors, the results of the tasks and the time of evaluation in order to achieve more credible quality metrics for workers.

Pre-filtering of humans. Finally, the worker may lack expertise or skills to handle some kind of complex job. Thus, some works [Liu et al., 2012] provide some basic knowledge to workers before they are put into work, or require some qualifications to prove themselves qualified to finish the task. Some other works [Rashtchian et al., 2010] propose an approach in which the use of a qualification tests (i.e., screening users before letting them performing tasks) provides the highest improvement of quality of linguistic data.

Gold standards. Obviously, an easy way of measuring workers' quality is to build a gold standard (i.e., a small group of data annotated by experts, so that it contains only reliable answers), and then compare the gold standard data with the workers' answers, so as to spot the workers with low quality [Bernstein et al., 2012, Le et al., 2010, Bernstein et al., 2011].

Recognition of malicious users. Malicious workers often try to maximize their financial gains by producing generic answers rather than actual working on the task. [Eickhoff and de Vries, 2011] concluded that malicious workers are less frequently encountered in novel tasks that involve a degree of creativity and abstraction, and crowd filtering can reduce the number of malicious workers.

Games with a purpose: an alternative way of asking questions

A Game With A Purpose (GWAP) is a human-based computation technique in which a computational process performs its function by outsourcing certain steps to humans in an entertaining way [von Ahn, 2006]. The aim of a game with a purpose is to exploit the enormous amount of time people spend online playing games, to solve complex problems that involve human intelligence. As a result, gamers become workers, and games become tasks, so that crowdsourcing is hidden in the gameplay and the requester ‘pays ’workers by providing them a tool (i.e., the game) that entertains them.

We call *implicit crowdsourcing* the crowdsourcing activities performed via these games: users are not aware of the crowdsourcing work they are performing [Luz et al., 2014].

Some examples of tasks hidden in games with a purpose are: object recognition in images, OCR results verification, protein folding and multiple sequence alignment algorithms in molecular biology [Cooper et al., 2010]. Another classical example is ReCAPTCHA [Little et al., 2010].

Several game designs have been proposed in the last years [Von Ahn and Dabbish, 2008]. One of the most famous serious game by von Ahn is the ESP-game. This game, which is played by two partners, encourages players to assign obvious tags to a non-tagged photo, which are most likely to lead to an agreement between the partner. When an agreement on the tag is found (i.e., the players propose the same tag for the photo), they gain points. ESP-game authors presented evidence that the labels produced by the players were useful descriptions of the photos. Thus, ESP-game proved itself to be a valid tool for performing image tagging efficiently and at no cost.

2.1.4 Passive crowdsourcing

Passive crowdsourcing techniques are used to analyze automatically data users publish on social networks and social media to extract some useful information. In the state of the art, passive crowdsourcing is applied to several context.

Environment analysis. Users are producing every day a massive amount of photos on social media (e.g., Flickr[13]), and several of them regard landscapes and cities. Several works analyze those photos to extract information about level of rainfall and snowfall, population density, plant and flower species distribution, population density in cities. In the following, we list some examples. The work [Hays and Efros, 2008] estimates land cover and population density on woods and cities based on visual descriptors of geotagged uploaded images. The works [Zhang et al., 2012] and [Wang et al., 2013a] quantify snow cover, snowfall, vegetation density and flower species distribution by analyzing multimedia content of photos in Twitter. The work [Goodchild and Glennon, 2010] applies crowdsourcing to the harnessing of geographical information for disaster response, such as occurrences of wildfires in Santa Barbara (California). Finally, the works [Fedorov et al., 2013] and [Fedorov et al., 2014] identify mountain peaks in photos crawled from Flickr, and estimate the snow cover on each peak to quantify the amount of water that will be available in summer in those areas.

Localization. Several photos are uploaded on social media without any geographical reference about where they were taken. Thus, positioning those photos over a map becomes a challenge. Several works accept that challenge and analyze similarity between photos and associated tags to find a geographical location to each photo [Serdyukov et al., 2009,Gallagher et al., 2009,Crandall et al., 2009].

Fashion trend. During the last years, some authors [Galli et al., 2012] developed several works that segment user-generated images to extract dress colors, texture features of garments and their association with the features of the subject wearing them, to analyze automatically the current fashion trends. This analysis can be useful for fashion industries, that, by analyzing content with an automatic pipeline, understand which are the product characteristics people prefer, and adapt their products accordingly.

[13]`https://www.flickr.com`

Event detection. When a noteworthy event happens, microblogs such as Twitter and social media like Flickr get filled with content talking about that event. Several earthquakes hit a place after tweets describing the same incoming earthquake arrived to it [The Washington Post, 2011]. Several works treated this topic. The work [Nitta et al., 2014] studied event detection by using time, location information and text tags attached to images in Flickr. The work [Gupta et al., 2014] performed the detection of events baed on tweet content and external sources. The works [Zhou and Chen, 2014] and [Sakaki et al., 2013] performed real-time detection of events over streams such as Twitter.

User profiling. Some works perform tweet content and profiles analysis to extrapolate demographic information about the user. Such information can be useful to identify the targets for political campaign, product campaign and recruiting. Two examples can be found in [Gupta et al., 2014] and [Ikeda et al., 2013], where user profiling is performed by focusing on location prediction of the user and on tweet content and community relationships, respectively.

Topic classification. In other cases (e.g., Twitter) the social networking site gives humans the permission to create friends categories: users assign tags to their posts in order to follow up the trending topics, and this facilitates quick retrieval and summary report on posts [Barbier et al., 2012].

Influence evaluation. Influencers, or opinion leaders, are people who can influence people's thought and sentiment about a specific topic. Chapter 4 will talk extensively about this subject. In the state of the art many works that cover this topic can be found [Jabeur et al., 2012, Cha et al., 2010, Agarwal et al., 2008]. Here, published content and social graph are jointly analyzed to capture the capability of each user to spread information over the network.

Sentiment analysis. User-generated content can be analyzed to determine consumer sentiment towards a brand or a topic. The work [Ghiassi et al., 2013] determines the sentiment towards a brand by automatically analyze tweets citing the name of that brand. The work [Kontopoulos et al., 2013] perform sentiment analysis of Twitter posts to discover sentiments and opinions on topics.

Video phylogeny. Multimedia content can be analyzed automatically to discover copies of the same content that are shared through different media. In fact, it happens often that users who want to gain visibility copy content from others and repost it as it was created by them. Many examples of works that cope with this problem can be found for videos [Dias et al., 2011, Ngo et al., 2013], images [Dias et al., 2010] and audio [Nucci et al., 2013].

Crisis map. A typical crowdsourcing application is the crisis map, where mass data is analyzed and displayed in a straightforward way in real time during a crisis. User-generated content is analyzed and clustered into meaningful categories [Goolsby, 2010]. As an example, people generated numerous messages and photos after the devastating earthquake in Haiti in 2010, through social media networking [Gao et al., 2011].

Homeland security. Crowdsourcing can benefit homeland security: crowds can contribute in delivering quality information and identifying the suspects. When the Boston marathon bombing happened in 2013, citizens submitted photos and videos they might have taken during the marathon. Then, by aggregating the available data, the suspects were identified actively by the crowd [Markowsky, 2013].

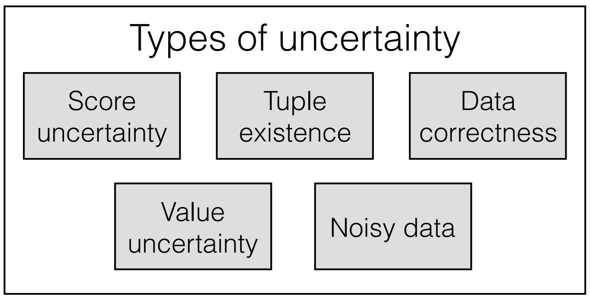


Figure 2.3: Uncertainty types

2.1.5 Other types of crowdsourcing

Crowd voting. Crowd voting [Kirkels and Post, 2013] is the process of asking opinions on a topic. Usually, these opinions are collected through the Web and used by producers to change their products, capture the mass' view on politics, give opinion about a leadership. Two famous examples of crowd voting are the one of Threadless[14] (which selects the t-shirt it sells by having users provide their own design and vote the ones they like) and the one of Lego[15] (which collects ideas from users about new toy kits and let the crowd vote the ones they would buy in stores).

Crowd searching. Crowd searching is a virtual search party in which internet users cooperate in looking for lost items, pets or people. This is usually done by joining a group of users through dedicated social networks, such as Crowdfynd[16].

Crowd funding. Crowd funding is the process of funding projects by a large group of people contributing a small amount of money. The most known crowd funding website is Kickstarter[17].

2.2 Uncertainty on data: taxonomy

Data uncertainty has become a well-studied problem in the last years, since it affects the result quality of different algorithms, e.g., query processing, data mining and clustering. In the following, we enumerate the uncertainty types works in the state of the art study, the possible representations of uncertainty that one can find in other works, and a set of problems related to uncertainty.

2.2.1 Types of uncertainty

Uncertainty on data may arise in different forms. In the following, we enumerate the types of uncertainty reviewed in the state of the art, as synthesized in Figure 2.3.

Score uncertainty [Soliman et al., 2010]. A top-K query is a query that retrieves the best K tuples matching the user's information need. In order to compute the score of an object for a query, a *scoring function* aggregates the objects' attribute values in a deterministic value, which reflects the importance of that object for the user need.

When score is not uncertain (i.e., it is deterministic), it induces a total order over the available objects, i.e., it is possible to order the objects from the most relevant to the least relevant. However, there are cases in which some sources of uncertainty (e.g., not known attributes)

[14]`https://www.threadless.com/make/submit/`
[15]`https://ideas.lego.com`
[16]`https://www.crowdfynd.com`
[17]`https://www.kickstarter.com`

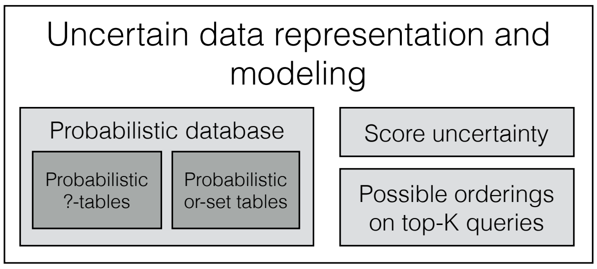

Uncertain data representation and modeling
Probabilistic database
Probabilistic ?-tables
Probabilistic or-set tables
Score uncertainty
Possible orderings on top-K queries

Figure 2.4: Uncertainty representation and modeling

influence the score computation, so that the score becomes not deterministic and thus described as a set of possible values. Score uncertainty induces a partial order over the underlying records, where multiple rankings are valid.

Tuple existence [Soliman et al., 2008]. In probabilistic databases, tuples may be related with an existence probability, i.e., a tuple actually exists in the real world with a specified probability. This means that not all the tuples enlisted in the database necessarily corresponds to real data.

This type of uncertainty is usually referred to as *membership uncertainty*. It treats tuples as uncertain events capturing the belief that they belong to the database. The probabilities of such events originate from different sources, e.g., reliability of data source in data integration environment, or similarity measures in approximate-matching.

Data correctness [Soliman et al., 2008]. Sometimes a tuple in the database is associated with the probability of giving correct information. This probability models the fact that the information contained in the tuple may be inadequate, due to some noise in the data.

Value uncertainty [Soliman et al., 2008, Ilyas et al., 2008]. A tuple attribute could be defined probabilistically as multiple possible values drawn from discrete or continuous domains. Value uncertainty represents attributes as probability distributions on continuous or discrete domains of possible values.

We assume that value uncertainty is transformed into a single probability distribution over tuple score [Li and Deshpande, 2010]. If an attribute does not contribute to the score, its uncertainty can be ignored for ranking purposes.

Noisy data [Bi and Zhang, 2004]. In traditional formulations of supervised learning, we seek a predictor that maps input x to output y. Sometimes errors are not only confined to the output y, i.e., noise is present in the input x too. Noise sources in this case could be identified as errors in the initial phases of data processing, e.g., in image classification applications some features may rely on image processing outputs that introduce errors.

2.2.2 Uncertain data representations and modeling

In uncertain data management [Aggarwal and Yu, 2009], data records are typically represented by probability distributions rather than deterministic values. A key issue is modeling uncertain data: the database designer has to find a way of capturing the underlying uncertainty, while keeping the data usable for database management applications. In the following, we introduce some typical representation of data uncertainty, as synthesized in Figure 2.4.

Probabilistic database

Uncertain data is not deterministic, and thus it can take many values. Each value is a possible materialization (i.e., instance) of the data in the real world. Thus, several instances of the same data are valid.

A probabilistic database [Aggarwal and Yu, 2009] is a finite probability space whose outcomes are all possible database instances consistent with a given schema. A database instance, also called *possible world*, is a possible materialization of the database in the real world. A probabilistic database can be represented as the pair $(\mathcal{X}, p)$ where: *i)* $\mathcal{X}$ is a finite set of possible database instances consistent with a given schema; *ii)* $p(I)$ is the probability associated with any instance $I \in \mathcal{X}$. We note that, since $p(\cdot)$ represents the probability vector over all instances in $\mathcal{X}$, we have that $\sum_{I \in \mathcal{X}} p(I) = 1$.

The validity of some tuple combination is determined based on the underlying tuple dependencies [Ilyas et al., 2008]. For instance, two tuples might never appear together in the same world if they represent the same real-world entity.

The uncertainty associated to tuples in the database may be represented in different ways [Ilyas et al., 2008]. Most applications work on two kinds of uncertainty:

1. *Existential uncertainty:* a tuple may or may not exist in the database, and the presence or absence of one tuple may affect the presence or absence of another tuple in the database. In some cases, tuple independence assumption is used. Furthermore, there may be constraints that correspond to mutual exclusivity of certain tuples in the database

2. *Attribute level uncertainty:* a number of tuples and their modeling have already been determined. Uncertainties of the individual attributes are modeled by a probability density function, or other statistical parameters such as the variance

Tuples' probabilities arise as an additional ranking dimension that interacts with tuples' scores [Ilyas et al., 2008]. Consequently, both probabilities and scores need to be factored in the interpretation of queries in probabilistic databases. Combining scores and probabilities using some score aggregation function could eliminate uncertainty completely, but this could not be meaningful in some cases.

Sometimes, probability is the only ranking dimension, since tuples are not scored by a scoring function. Tuple are then treated as probabilistic events.

Probabilistic ?-tables. Since each tuple t comes with the probability that t is present in the database, the probability of the instance I is defined as:

$$Pr(I) = \prod_{t \in I} Pr(t) \cdot \prod_{t \notin I} (1 - Pr(t))$$

If $\mathbf{Q}$ is a query and $\mathbf{Q}(I)$ is the output of $\mathbf{Q}$ restricted to a single possible world, the probability of an output tuple t_q is:

$$Pr(t_q) = \sum_{I : t_q \in \mathbf{Q}(I)} Pr(I)$$

Probabilistic or-set tables. A probabilistic or-set table models the probabilistic behavior of each attribute for a tuple that is known to be present in the database. Each attribute is represented as an `or` over various possibilities along with corresponding probability values.

An instance of the database is obtained by picking each outcome for an attribute independently.

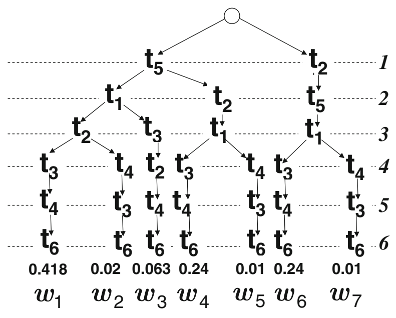


Figure 2.5: Space of possible orderings for a set of tuples

Score uncertainty

The score $s(t_i)$ of the record t_i is modeled as a probability density function f_i defined on a score interval $[l_i, u_i]$ [Soliman et al., 2010].

The density function can be: *i)* directly obtained from uncertain attributes (e.g.,a uniform distribution), or *ii)* computed from the predictions of missing/incomplete attribute values, or *iii)* constructed from histories and value correlations.

The space of possible orderings is viewed as a probability space generated by a probabilistic process that draws, for each record t_i, a random score $s(t_i) \in [l_i, u_i]$ based on the density f_i. Ranking the drawn scores gives a total order on the database records, where the probability of such order is the joint probability of the drawn scores.

Possible orderings

Uncertainty induces a partial order over the underlying records [Soliman et al., 2010].

The *linear extensions* of a partial order are all possible topological sorts of the partial order graph.

Linear extensions can be viewed as tree where each root-to-leaf path is one linear extension. Each occurrence of the record t in the tree represents a possible ranking of t.

Possible world probabilities are determined based on the probabilistic correlations among tuples (e.g., mutual exclusion of tuples that map to the same real world entity). We call such correlations generation rules, since they control how the possible worlds space is generated.

Examples of generation rules [Soliman et al., 2007] are:

- *Exclusiveness*: tuples never co-exist in the same world
- *Implication*: the existence of one tuple in some world leads to the existence of another specific tuple in the same world
- *Equivalence*: both tuples always exist together

Whenever tuples are *independent*, the joint probability of any combination of tuple events is computed by multiplying the probabilities of the corresponding tuple events. On the other hand, working with *correlated tuples* means that generation rules are present too. In this case, the joint probability of any combination of tuple events is computed based on tuples' probabilities and rules semantics.

2.2.3 Problems related to uncertainty

In case of data uncertainty, one wishes to adapt traditional database management techniques so as to handle both uncertain data and deterministic data. For this reason, since the results of data mining applications are affected by the underlying uncertainty in the data, it is critical to design

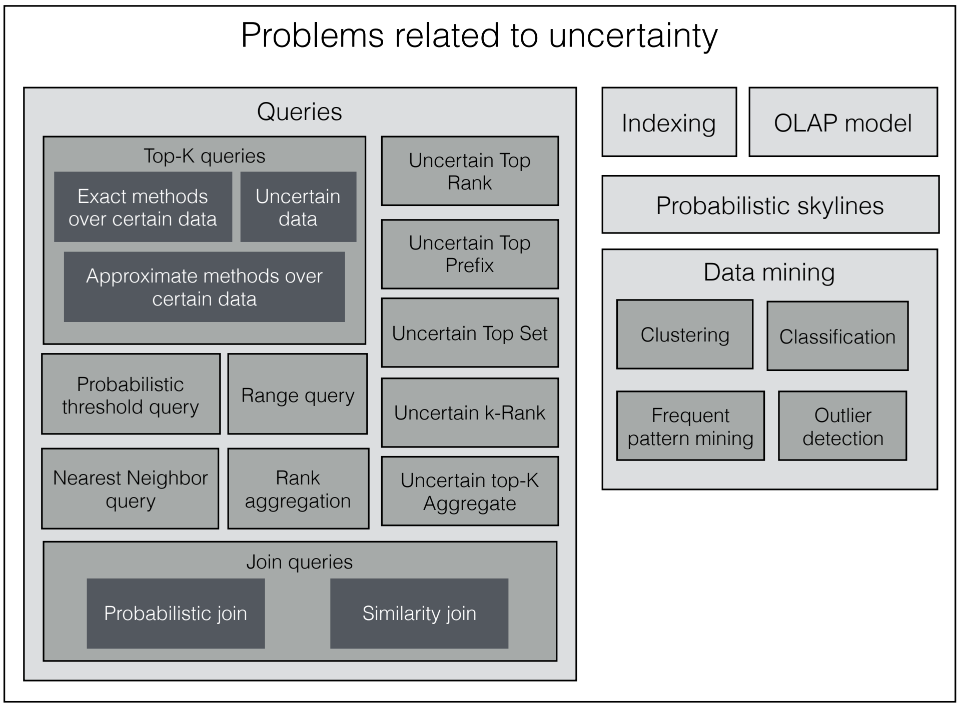


Figure 2.6: Problems related to uncertainty

data mining techniques that can take such uncertainty into account during the computations. In the following we enlist how classical data retrieval problems are affected by data uncertainty (as synthesized in Figure 2.6), and how to overcome such issues.

Queries

In traditional database management, queries are typically represented as SQL expressions which are then executed on the database according to a query plan [Aggarwal and Yu, 2009].

A given query on an uncertain database may require computation or aggregation over a large number of possible instances of the uncertain data. In some cases, queries may be nested, which greatly increases the complexity of the computation. Thus, two semantic approaches to overcome these problems were introduced in the state of the art. On the one hand, the intensional semantics models the uncertain database in terms of an event model, which defines the possible worlds, and uses tree-like structures of inferences on these event combinations. The tree structure enumerates all the possibilities over which the query may be evaluated and subsequently aggregated. In this case, developing a probabilistic relational algebra is required, and correct results are always obtained. On the other hand, the extensional semantics designs a plan which can approximate these queries without having to enumerate the entire tree of inferences. In this case uncertainty is represented as a generalized truth value attached to formulas. We attempt to evaluate, or approximate, the uncertainty of a given formula based on that of its sub-formulas. The extensional semantics is not able to capture the underlying complexity and dependencies in the query results. Consequently, it is useful in case of evaluating simple expressions.

Example. Consider a possible worlds model in which k possible tuples are present: $t_1, \ldots, t_k$. Each tuple has an existence probability p_{t_i}. We want to compute the probability of having both t_1 and t_2 in the database. An intensional plan would require us to create the event variables $e(t_1)$, $e(t_2)$ and to compute the probability $Pr(e(t_1) \cap e(t_2))$. An extensional plan would just compute the probability $P(t_1) \cdot P(t_2)$, without taking into account any correlation between t_1 and t_2.

Top-k queries. We classify top-k processing techniques based on query and data certainty as follows [Aggarwal and Yu, 2009, Ilyas et al., 2008]:

- **Exact methods over certain data**: deterministic top-k queries are processed over deterministic data
- **Approximate methods over certain data**: techniques that operate on deterministic data, but report approximate answers in favor of performance. The approximate answers are associated with probabilistic guarantees indicating how far they are from the exact answer
- **Uncertain data**: techniques that work on probabilistic data. Some approaches treat probabilities as the only scoring dimension, while others study the interplay between the scoring and probability dimension

In case of uncertain data, the aim is to find the top-k answers for a particular query. The top-k ranking is based on some scoring function in deterministic applications. However, in uncertain applications, a tuple in a top-k answer does not depend only on its score but also on its membership probability.

The usage of the possible world semantics allows complex correlations among tuples in the database. We call generation rules the logical formulas that determine valid worlds. Consequently, all responses to the queries need to be defined in valid possible worlds in order to avoid answers inconsistent with generation rules and other database constraints.

The goal is to evaluate the top-k query in the most probable world. This leads to two possible interpretations: *i)* extract the top-k tuples in the most probable world, or *ii)* extract the most probable top-k tuples that belong to the valid possible worlds. Both the interpretations involve both ranking and aggregations across possible worlds.

Top-k queries - Deterministic data, approximate methods Reporting the exact top-k query answers could be neither cheap nor necessary for some applications [Ilyas et al., 2008]. Users may thus sacrifice the accuracy of query answers in return of savings in time and resources.

Tuples are partitioned into *certain* and *possible* tuples. As more certain data is read from base tables, more certain tuples are inserted into approximate relations while more possible tuples are deleted.

The problem is that, since usually a parameter θ is chosen in order to denote the required level of approximation, the selection of this parameter is mostly application-oriented. Approximate answers are more useful when they are associated with some accuracy guarantees. Consequently, a probability p is associated with a candidate tuple t_i, stating that t_i can be in the top-k set with probability p.

An example of approximate query is the similarity search query. A similarity search algorithm uses a distance metrics to rank objects according to their distance from a target query object. In this case, a query is represented as a hypersphere centered in the target object. Since in many cases no actual data points (or a small number of data points) are included in the hypersphere, there is no way to precisely determine the useless regions, i.e., regions that do not contain data and thus that cannot provide useful results. A proximity measure is used to decide if a data region should be inspected or not; data regions whose proximity to the query region is greater than a specified threshold are accessed.

Uncertain Top Rank. A UTop-Rank(i,j) [Soliman et al., 2010] query reports the most probable record to appear at any rank $i, \ldots, j$ in possible orderings.

Uncertain Top Prefix. A UTop-Prefix(k) [Soliman et al., 2010] query reports the most probable linear extension prefix of k records.

Uncertain Top Set. A UTop-Set(k) [Soliman et al., 2010] query reports the most probable set of top-k records of linear extensions.

Rank Aggregation. Rank aggregation [Soliman et al., 2010] is the problem of computing a consensus ranking for a set of candidates $\mathcal{C}$ using input rankings of $\mathcal{C}$ coming from different voters. The optimal rank aggregation is the ranking with the minimum average distance to all input rankings.

Uncertain Top-k query. A UTop-k query [Soliman et al., 2008] reports a top-k vector with the highest aggregated probability across all worlds.

Let $\mathcal{D}$ be a probabilistic database, $\mathbf{Q}$ a query to be processed on $\mathcal{D}$ and $\mathbf{Q}(\mathcal{D})$ the output tuples of the query. Let $\mathcal{PW}$ be the set of possible worlds for a probabilistic database. Let $\mathcal{T} = \{T^1, \ldots, T^m\}$ be the set of all k-length tuple vectors in $\mathbf{Q}(\mathcal{D})$ such that each $T^i \in \mathcal{T}$ is the top-k answer in a set of possible worlds $W(T^i) \subseteq \mathcal{PW}$.

The probability of a top-k vector $T^j \in \mathcal{T}$ is

$$Pr(T^j) = \sum_{PW^i \in W(T^j)} Pr(PW^i)$$

The UTop-k query returns a k-length tuple vector $T^* \in \mathcal{T}$ with the highest probability among all tuple vectors in $\mathcal{T}$.

Uncertain k-Rank query. A U-kRanks query reports a set of tuples that might not form together the most probable top-k vector. However, each tuple is a clear winner at its rank over all worlds.

Uncertain Top-k Aggregate Query [Soliman et al., 2008] Let $\mathbf{Q}$ be a group-by query, $\mathcal{A}$ be a set of grouping attributes. Let $\mathcal{G}_i = \langle g_1, \ldots, g_k \rangle$ be the top-k group vector in a nonempty set of possible worlds $W(\mathcal{G}_i) \subseteq \mathcal{PW}$, based on $\mathcal{A}$. We define the probability of a top-k-group vector $\mathcal{G}_i$ as

$$Pr(\mathcal{G}_i) = \sum_{PW^l \in W(\mathcal{G}_i)} Pr(PW^l)$$

A UTop$k$-Agg query [Soliman et al., 2007] returns a k-length vector $\mathcal{G}^*$ with the highest probability.

Other uncertain queries. In the following, we list some of the queries one can perform on uncertain data.

The range queries require to find all the objects in a given range. Since the objects are uncertain, their exact positions are not known, and thus their membership in the range cannot be known deterministically.

The nearest neighbor queries require to determine the objects with the least expected nearest neighbor distance to the target. The probabilistic NN queries are formulated in terms of the nonzero probability that a given object is the nearest neighbor to the target. The technique involves evaluating the probability of each object being closest to the query point.

In the case of the aggregate queries one wants to determine the aggregate statistics from queries such as the sum of the max.

The probabilistic threshold queries are queries that determine all objects whose behavior satisfies certain conditions with a minimum probability.

Indexing uncertain data

The problem of indexing uncertain data [Aggarwal and Yu, 2009] arises when data is updated only periodically in the index, and therefore the current attribute values cannot be known exactly (they can only be estimated).

There are many types of queries that can be answered with the use of index structures. All the queries can be classified in two categories:

- An *entity-based query* returns a set of objects that satisfy the condition of the query
- A *value-based query* returns a single value, which can be for instance the value of a specific dimension, or a statistical function of a set of objects satisfying query constraints.

OLAP Model

The queries that are most relevant for the OLAP setting [Aggarwal and Yu, 2009] are the aggregation queries, in which one attempts to aggregate a particular function of the data on a part of the data cube.

A set of criteria has been identified in order to handle ambiguities:

- **Consistency:** the consistency criterion from the OLAP perspective
- **Faithfulness:** more precise data should lead to more accurate results
- **Correlation preservation:** the correlation properties of the data should not be affected by the allocation of ambiguous data records

An extension is to introduce a new measure attribute which represents uncertainty. This is in the form of a probability distribution function over the base domain. In this case, the query needs to aggregate over different probability density functions. The problem of aggregating PDFs is closely related to a problem studied in the statistics literature, which is that of opinion pooling. The opinion pooling problem is to form a *consensus opinion* from a given set of opinions θ. The set of opinions as well as the consensus opinion are presented as pdf's over a discrete domain O.

Join processing

In the following, we illustrate some of the techniques used to perform join queries on uncertain data [Aggarwal and Yu, 2009].

Probabilistic Join Queries. In these queries, each item is associated with a range of possible values and a pdf, which quantifies the behavior of the data over that range. Each join-pair is associated with a probability, to indicate the likelihood that the two tuples are matched.

The join may contain a number of false positives. Since each tuple-pair is associated with a probability that indicates the likelihood of the join, those pairs which have low probability values can be discarded in order to reduce the number of false positives. This variant is referred to as Probabilistic Threshold Join Query. We note that the use of thresholds reduces the number of false positives, but it may also result in the introduction of false negatives. A tradeoff between the number of false positives and false negatives is chosen.

A key operator in the case of joins is that of equality, since a join is performed only when the corresponding attribute values are equal. In this case, a pair of attributes are defined to be equal to one another within acceptable resolution c.

Similarity join. In these queries, similarity is measured by the distance between the two feature vectors. The join is performed on this distance.

The most popular similarity join is the distance-range join. In this case, we perform the join between two records if the distance between the two does not exceed a parameter ϵ. However, the expected distance may not reflect the true likelihood that a given pair of records may join on a particular attribute. Consequently, different joins which have similar probability of lying within the range of ϵ may be treated inconsistently. Therefore, a probability value is assigned to each pair of objects, reflecting the likelihood that the pair belongs to the join result set.

Probabilistic Skylines on Uncertain Data

The skyline computation [Aggarwal and Yu, 2009] allows to recognize the best object in a set of objects that are defined by d different dimensions. For instance, if different NBA players are considered, different parameters are used to evaluate their performance, e.g., the number of assists, the rebounds, the baskets. We want to identify the best player among the set of players.

We say that an object u dominates another object v if, for each dimension $i \in \{1, \ldots, d\}$ we have $u_i \leq v_i$, and for some dimension $i_o \in \{1, \ldots, d\}$ we have $u_{i_0} \leq v_{i_0}$. Then, given a set of points S, a point u is a skyline point if there exists no other point $v \in S$ such that v dominates u. The skyline of S is the set of all skyline points.

A probabilistic skyline captures the dominance relationship between uncertain objects. In this case, the probability of an object being in the skyline is the probability that the object is not dominated by any other objects. The challenge in this case is that the probability density function of uncertain data objects is not available explicitly.

Mining applications

The presence of uncertainty can affect the results of data mining applications significantly [Aggarwal and Yu, 2009]. In the case of a *classification* application, an attribute which has lower uncertainty is more useful than an attribute which has a higher level of uncertainty. In a *clustering* application, instead, the attributes which have a higher level of uncertainty need to be treated differently from those which have a lower level of uncertainty.

Clustering uncertain data. The presence of uncertainty changes the nature of the underlying clusters [Aggarwal and Yu, 2009]. We need to compute uncertain distances effectively between objects which are probabilistically specified. The fuzzy distance is defined in terms of the distance distribution function, that encodes the probability that the distances between two uncertain objects lie within a certain user-defined range.

Classification of uncertain data. A closely related problem is that of classification of uncertain data in which the aim is to classify a test instance into one particular label from a set of class labels [Zhang, 2005].

The support vector machine (SVM) technique functions by constructing boundaries between groups of data records. The margin created by the SVM can be modified by using the uncertainty of the points which lie in the boundary.

A geometric algorithm has to be found, in which we optimize the probabilistic separation between the two classes on both sides of the boundary.

Frequent pattern mining. In this model we assume that each item has an existential uncertainty in belonging to a transaction [Aggarwal and Yu, 2009]. The probability of an item belonging to a particular transaction is modeled in this approach.

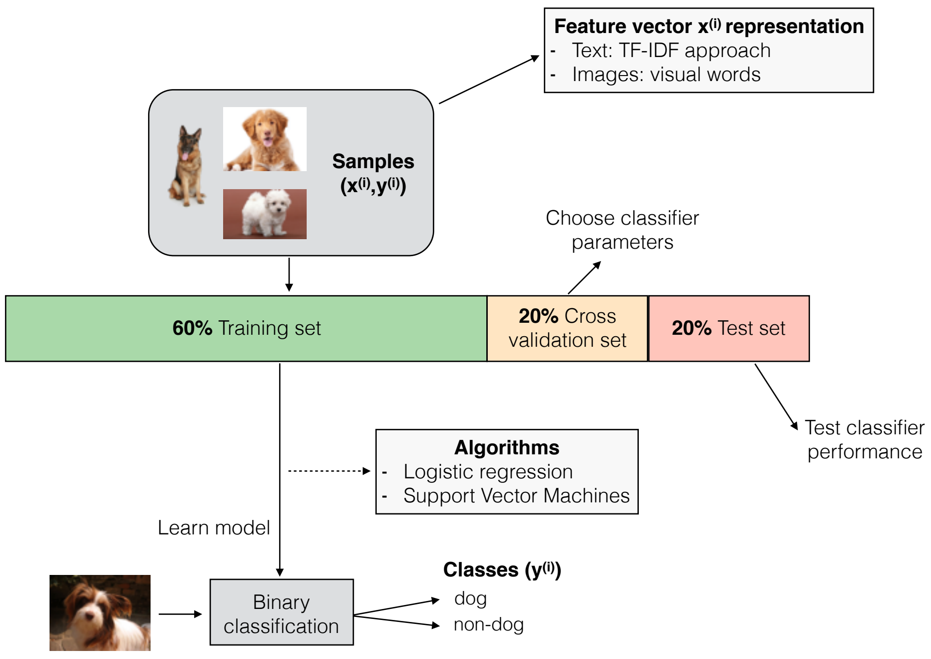


Figure 2.7: Supervised learning process

Outlier detection with uncertain data. In the problem of outlier detection differing levels of uncertainty across different dimensions may affect the determination of the outliers in the underlying data [Aggarwal and Yu, 2009]. It is possible to define the concept of an outlier in terms of the probability that a given data point is drawn from a dense region of the overall data distribution.

2.3 Machine learning

In the following, we introduce some background concepts on state-of-the-art machine learning techniques, and more specifically on text and image classification, which will be used in Chapter 4.

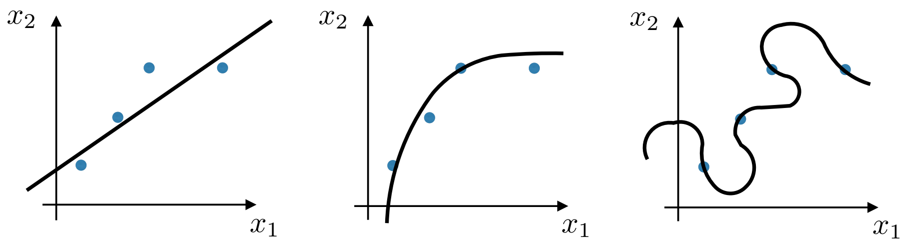
2.3.1 Supervised learning

In this Section, we illustrate come key concepts of supervised learning, as shown in Figure 2.7.

Let $x^{(i)} \in \mathcal{X}$ be a set of input variables, called features, and $y^{(i)} \in \mathcal{Y}$ the output variable we would like to predict, called target. For instance, $x^{(i)}$ could represent some characteristics for the house i (i.e., number of bathrooms, number of bedrooms), while $y^{(i)}$ may characterize its price. Our objective is to learn the model used to give a price to a house with some characteristics, so that, given a house j and its characteristics $x^{(j)}$, we are able to predict its price $y^{(j)}$. When y is continuous, the learning problem we would like to solve is called a *regression problem*. On the other hand, when y takes only a small number of discrete values, then we are solving a *classification problem*.

Let $(x^{(i)}, y^{(i)})$ be a training sample. The value $y^{(i)}$ is called label for the value $x^{(i)}$. Then, the set of training samples $\mathcal{T} = \{(x^{(i)}, y^{(i)}) : i = 1, \ldots, m\}$ is called *training set* and represents the dataset we will use to learn y.

More formally, given a training set $\mathcal{T}$, we would like to learn a function $h : \mathcal{X} \to \mathcal{Y}$, called *hypothesis*, so that $h(x)$ is a good predictor for the corresponding y.



(a) High bias (underfitting) (b) Right model (c) High variance (overfitting)

Figure 2.8: Model choice to fit the training samples $(x^{(i)}, y^{(i)})$

Binary classification problem

We will focus on the *binary classification problem*, in which y can take only two values, i.e., 0 or 1.

For instance, if we are building a topic-related content classifier, then $x^{(i)}$ can be the text to be classified, and y can be 1 if the text is related to a specific topic T, or 0 otherwise. Here, 0 is called *negative class*, while 1 is called *positive class.*

The problem of overfitting and underfitting

Given a set of training samples $(x^{(i)}, y^{(i)})$, we would like to find the model that best subdivides the points $(x^{(i)}, y^{(i)})$ in the positive and negative classes.

As a first step, what one could do is to select manually a family of possible models (e.g., linear, polynomial) and then find the parameters that best fit for that model family and those training samples.

For instance, consider Figure 2.8.

In Figure 2.8(a) a simple, linear model was selected. Unfortunately, this model is too poor to fit the data in the training set: this situation is characterized by the phenomenon called *underfitting*, or *high bias.*

On the other hand, in Figure 2.8(c) a complex polynomial model was selected. Here, the model is rich enough to cover all the samples $(x^{(i)}, y^{(i)})$ in the training set, and consequently it reveals to be perfect for the available data. Unfortunately, the model 'learned 'the data, and thus when other training samples $(x^{(j)}, y^{(j)})$ are selected, the model is not generalized enough to be able to fit them. This situation is characterized by the phenomenon called *overfitting*, or *high variance.*

Cross validation set and test set

Let $\mathcal{T}$ be the training set used to train the classifier. If several models are available, then a possible solution could be to train each available model M_i on $\mathcal{T}$, to get some hypothesis h_i, and then pick the hypotheses with the smallest training error, i.e., the hypotheses that best fit the available data. Unfortunately, this solution does not work: the higher the order of the polynomial, the better it will fit the training set $\mathcal{T}$, the lower the training error. Consequently, this method will always select a high-variance model.

A smarter approach is the one of dividing the available data in two sets: the *training set* $\mathcal{T}_{\text{train}}$, containing 70% of the data, and the *test set* $\mathcal{T}_{\text{test}}$, containing the remaining 30% of the data. With this method, each model M_i is trained on $\mathcal{T}_{\text{train}}$, to get some hypothesis h_i, and then we select the hypothesis that generates the smallest test error on the test set $\mathcal{T}_{\text{test}}$. However, this methodology has its drawbacks too: we selected a model according to the error that is

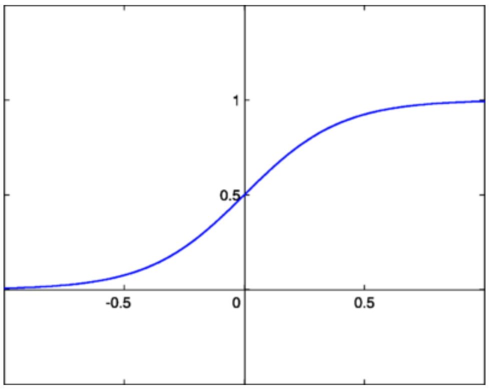


Figure 2.9: Sigmoid function $g(z)$

generated on already seen data (the test set $\mathcal{T}_{\text{test}}$), and thus we cannot have proof of which is the performance on new data.

A third solution is to subdivide the set of data in three sets: the training set $\mathcal{T}_{\text{train}}$, the test set $\mathcal{T}_{\text{test}}$ and the *cross validation set* $\mathcal{T}_{\text{CV}}$. In this case, each model M_i is trained on $\mathcal{T}_{\text{train}}$ to get some hypothesis h_i, the one with the highest performance on $\mathcal{T}_{\text{CV}}$ is selected and the final classifier performance (on new data) is computed on $\mathcal{T}_{\text{test}}$. Notice that in this case $\mathcal{T}_{\text{train}}$ contains 60% of the data, while $\mathcal{T}_{\text{CV}}$ and $\mathcal{T}_{\text{test}}$ contain both 20% of the data.

Algorithms

Logistic regression.

The hypothesis $h_\theta(x)$ is generally expressed as:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{2.1}$$

where $g(z)$ is called sigmoid function. Figure 2.9 shows a plot of $g(z)$, where $g(z)$ tends toward 1 as z tends toward ∞, and $g(z)$ tends toward 0 as z tends toward $-\infty$.

To fit the parameters θ, the maximum likelihood approach can be used. At first, we assume that:

$$\Pr(y = 1|x; \theta) = h_\theta(x)$$

and:

$$\Pr(y = 0|x; \theta) = 1 - h_\theta(x)$$

Consequently,

$$p(y|x; \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

Assuming that the training samples were generated independently, the likelihood of the parameters can be written as follows:

$$L(\theta) = \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \tag{2.2}$$

Finally, the parameters θ will be chosen so that they maximize the log-likelihood $\ell(\theta) = \log L(\theta)$.

Support Vector Machines.

This Section presents the main concepts behind the construction of a Support Vector Machine (SVM) learning algorithm.

Considering the definition of logistic regression, we predict $y = 1$ on the input x if and only if $h_\theta(x) \geq 0.5$, or equivalently, if $\theta^T x \geq 0$. Moreover, it is natural to say that the larger is $\theta^T x$,

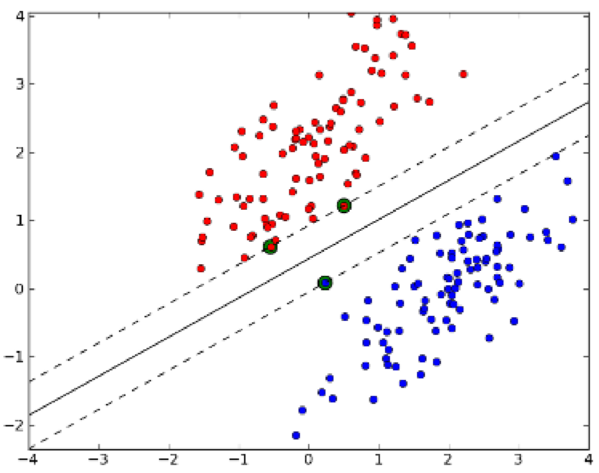


Figure 2.10: Decision boundary for a support vector machine. In green: support vectors, i.e., those training samples $(x^{(i)}, y^{(i)})$ at the minimum distance from the decision boundary (Source: `http://www.mblondel.org/images/svm_linear.png`)

the higher is our confidence that the label for x is 1. The same holds for the negative class: if $\theta^T x < 0$, then $y = 0$, and the outcome is correct with a high confidence if $\theta^T x \ll 0$.

The objective of an SVM is the one of finding a decision boundary that, while separating the positive class from the negative class, allows us to make confident predictions on the training samples. This is obtained by finding the decision boundary that maximizes the *geometric margins*, i.e., the distance of each training sample from the decision boundary itself.

Kernels. Let x be the set of input attributes of a problem. It is possible to map these values to other quantities, called input features. This is generally done via a feature mapping ϕ, which maps the attributes to the features.

Rather than applying the SVM on the original input attributes x, we could want to use the features $\phi(x)$. A *kernel* is a function $K(\cdot)$ that enable us to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space.

Several kernel functions can be used with an SVM. Generally, we can decide to avoid the usage of a complex kernel (i.e., we use a linear kernel) in case the number of training samples is high and the number of features is low. However, in case the number of tracked features is high and the training set is small, it is convenient to use a more refined kernel, e.g., the Gaussian kernel.

Parameters. An SVM classifier is mainly characterized by two parameters: the *cost parameter* C and the *kernel width* σ.

When σ is large, then features vary smoothly, thus leading to high bias and low variance. On the other hand, small values of σ lead to a sharp variation of the features, a lower bias and a higher variance.

Furthermore, when C is large, we come up with a situation of low bias and high variance. On the other hand, small values of C lead to a situation of high bias and low variance.

Build feature vectors: text-based approach

When classifying text, we would like to find a way of translating the content of each document in a feature vector, that can be fed as an input to a classification algorithm and then processed as a training sample $(x^{(i)}, y^{(i)})$. Thus, text (intended as sequence of words and symbols) needs to be processed so as to compute a vector of numerical values (representing the feature vector).

One of the classical processing techniques for textual documents is the *term frequency-inverse document frequency (tf-idf) approach.* This approach is a statistic intended to reflect the importance of a word in a document belonging to a collection of documents. The tf-idf value is meant to increase proportionally to the number of times a word appears in a document; however, a penalty term (i.e., the idf) is added to the computation, to penalize words that appear very frequently in most of the documents in the collection. Using this approach, every pair term-document t_i, d_i is associated with a value $\texttt{tf} - \texttt{idf}(i, j)$, which indicates the importance of t_i for d_i, weighted by the representativeness of t_i for the whole document collection $\mathcal{D}$:

$$\texttt{tf} - \texttt{idf}(i,j) = \texttt{tf}(i,j)\texttt{idf}(i) = \texttt{frequency}(t_i, d_j) \log \frac{|\mathcal{D}|}{|\{d_j \in \mathcal{D} | t_i \in d_j\}|}$$

Thus, given a document d_j, we subdivide it in words. Each word extracted from a document d_j is used as a term t_i, and is thus associated with a value $\texttt{tf} - \texttt{idf}(i, j)$.

To build the feature set, every document d_j in the training set $\mathcal{T}_{\text{train}}$ is subdivided in words: every word will correspond to a feature.

The feature vector for a document d_j is such that the i-th value is computed as $\texttt{tf} - \texttt{idf}(i, j)$.

Stop words. A *stop word* is a word that is filtered out prior to processing of text. This is usually done since stop words are not discriminative to any topic, due to their nature, and thus it would be useless to consider them as features describing a text. Usually we consider stop words terms as *the*, *as*, *on*, *which.* Obviously this concept can be extended to include other types of terms which are generally diffused and not particularly discriminative (e.g., *want* or *like*).

Stemming. Stemming is the process used for reducing inflected words to their stem (i.e., root form). A stemmer for English for instance should identify that the words *cat*, *cats*, *catlike* are all related to the same term *cat.* Stemming is used to group together words with a similar basic meaning. Consequently, by applying this process we can understand that two documents reporting two different words that resemble the same term (e.g., *cat* and *cats*) are probably referring to the same concept. Note that the extracted stem does not have to be identical to the morphological root of the word: it is sufficient that related words map to the same stemmed term. One of the most diffused stemming algorithms is the Porter stemmer.

Image classification

Classifying images into semantic categories is a problem of great interest [Yang et al., 2007]. For instance, if an online collection of photos is available, we could need to classify those photos as belonging to a specific topic T or not.

In the last ten years, there has been a trend of using image key-points (or local interest points) in image retrieval and classification. A *key-point* is a salient image patch that contains rich local information of an image. Usually, a key-point can be automatically detected using various detectors and represented using various descriptors. In Figure 2.11, the key-points are denoted by small crosses in the three images.

An image can be represented by a set of key-point descriptors, i.e., the set of descriptors for all the points in the image. However, this set varies in cardinality and lacks meaningful ordering. This creates difficulties for many learning methods, that require feature vectors of fixed dimension as input. Thus, it is still desirable to transform raw key-point features into an image feature with a fixed dimension. To do so, key-points are grouped into a large number of clusters using the k-means clustering algorithm [Hartigan, 1975], so that those with similar descriptors are assigned into the same cluster. We call each cluster *visual word*: a visual word is

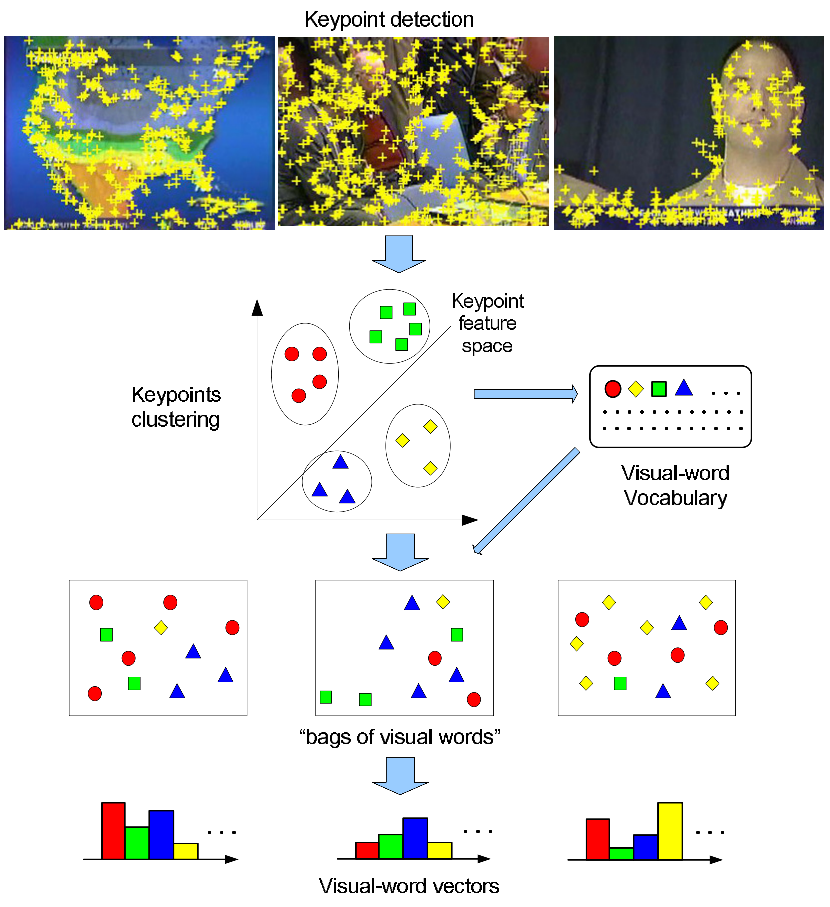


Figure 2.11: Extraction of a bag of visual words [Yang et al., 2007]

an entity that represents a specific local pattern shared by the key-points in that cluster. From this point on, each key-point is referred to as the index of the cluster it belongs to.

There exists a correspondence between visual words in images and words in textual documents. In textual documents, a word represents a concept, and a concept can be expressed by means of several synonyms. In the same way, a visual word represents a visual concept (i.e., a visual pattern), and it is such that every patch comprised in the related cluster is a synonym for the same concept.

Then, given a large collection of photos, one could extract all the contained visual words and build the so-called *visual-word vocabulary*, i.e., a collection of visual word describing all kinds of local image patterns. The number of clusters determines the size of the vocabulary, which usually varies from hundreds to over tens of thousands.

When a new image is submitted to the system, its key-points are extracted and mapped to a visual word. That is, for each key-point, we find the visual word that is most similar to the key-point descriptor. What we obtain is a collection of words that describe the image, so that the image representation is now a series of textual term and the image can be treated as a textual document itself. We call *bag of visual words* the set of visual words in that image. This representation is analogous to the bag of words document representation in terms of form and semantics. Both representations are sparse and high dimensional, and just as words convey meanings of a document, visual words reveal local patterns characteristic of the whole image.

The bag of visual words representation can be then converted into a visual word vector similar to the term vector of a document. The visual word vector is such that each bin represents the (normalized) count of the corresponding visual word in the vocabulary. That is, it represents the number of image key-points in the corresponding cluster. Such bag of visual word can be thus used as a feature vector in classification task, as explained in Section 2.3.1.

2.3.2 Decision trees

When performing a classification task we are provided with a set of K *objects* $o \in \mathcal{O}$, each one described by a finite set of attributes $\mathcal{A}_o$. Each *attribute* $a \in \mathcal{A}_o$ measures an important feature of o. For instance, suppose that the objects were days and the classification task involved the weather. In this case, an effective set of attributes could be the following:

$$\mathcal{A}_o = \{outlook, temperature, humidity, windy\}$$

where each attribute may have values:

$$\begin{aligned}&outlook : \texttt{sunny}, \texttt{overcast}, \texttt{rain}\\ &temperature : \texttt{cool}, \texttt{mild}, \texttt{hot}\\ &humidity : \texttt{high}, \texttt{normal}\\ &windy : \texttt{true}, \texttt{false}\end{aligned}$$

Each object $o \in \mathcal{O}$ belongs to a class $c \in \mathcal{C}$.

In our setting, $\mathcal{O}$ is our *training set*, i.e., a set of objects whose class is known. A classification rule need to be defined, so that, when a new object $o_{new} \notin \mathcal{O}$ is fed as an input, its class can be determined. The classification rule will be expressed as a decision tree built on the training set.

A *decision tree*, as defined in [Quinlan, 1986], is a tree in which leaves are labeled with class names and other nodes represent tests that will be performed on some object attribute $a \in \mathcal{A}_o$. Each test T produces N branches, where N is the number of possible values for a. For instance, if T is a test on 'humidity', its two possible branches correspond to the values 'high' and 'normal'. The classification process for the object o starts from the root of the tree and goes down to the leaves. When a test T on the attribute a is evaluated for the object o, the branch corresponding to the value $v_{a,o}$ of the attribute in o is taken. The process terminates when a leaf (labeled with class c) is encountered, at which time the object is said to belong to c.

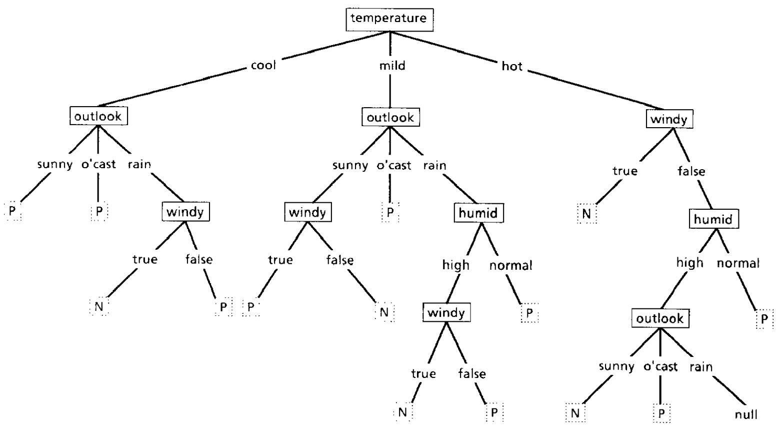


Figure 2.12: Example of decision tree. Here, two classes are present: P (positive class) and N (negative class)

It is always possible to build at least one decision tree that correctly classifies each object in the training set, but usually there are many correct decision trees. The *ID3 algorithm* allows one to find a reasonably good decision tree without much computation. ID3 is iterative: in the first step a selection of objects $\mathcal{O}_w$, called window, is extracted from the training set, and the decision tree is built on it; then, if the tree classifies correctly all the objects $o \in \{\mathcal{O} \backslash \mathcal{O}_w\}$, the algorithm terminates, otherwise a subset of incorrectly classified objects is added to the window and the process continues.

Let T_k be a test on the attribute a, whose possible outcomes are $v_{a,1}, \ldots, v_{a,N}$. If $\mathcal{O}$ is the set of objects on which T is evaluated, $\mathcal{O}$ will be split into N groups $\mathcal{O}_1, \ldots, \mathcal{O}_N$. The i-th outgoing branch of T_k will end into another test T_{k+1}, which will be evaluated on $\mathcal{O}_i$.

The open question is about how to choose a test for a node in the tree. The decision tree can be seen as a source of messages, whose content is the retrieved class label. Let $P = \langle p_1, \ldots, p_M \rangle$ be the probabilities of belonging to the classes $c_1, \ldots, c_M$ respectively. Let $I(P)$ be the information needed to generate this message (i.e., the entropy of the probabilities P, $H(P) = -\sum_{i=1}^{M} p_i \log p_i$). Let $E(a)$ be the expected information required for the tree with a as root, i.e., $E(a) = \sum_{i=1}^{N} \frac{|\mathcal{O}_i|}{|\mathcal{O}|} I(P)$, where N is the number of possible values for a. Then, the information gained by branching on a is:

$$G(a) = I(P) - E(a)$$

The attribute that will be chosen to fit in the root of the tree will be the one that maximizes the information gain:

$$T_{root} = \arg\max_a G(a)$$

Algorithm 1 shows the procedure for the construction of a decision tree. The complexity of ID3 is evaluated in [Quinlan, 1986]. At each non-leaf node the gain of the attribute a must be computed. This means that each object need to be analyzed in order to determine how many objects belong to each set $\mathcal{O}_i$. Consequently, for each internal node the complexity is $O(K|\mathcal{A}|)$. Since no exponential growth in time has been observed, ID3 can be applied to large tasks.

The analysis conducted by [Mitchell, 1997] shows that this algorithm may produce trees that overfit the training set. This happens mostly when the number of training samples is either too small to characterize a significant sample of the true target function or when data is noisy. Figure 2.13 shows the result of this analysis. The x-axis indicates the total number of nodes in the decision tree. The solid line shows the accuracy in the classification of training samples, while the broken line shows the accuracy over new data. Obviously the accuracy in the classification of training samples increases monotonically as the tree grows, since the algorithm is 'learning' the data. Unfortunately this is not valid for new data: once the number of nodes exceeds 25, the accuracy in the classification decreases. This is due to the fact that data could present coincidental regularities, and consequently some attribute may partition the training set very well, despite being unrelated to the actual target function. A way to prevent overfitting may be the post-pruning of the tree: first of all the tree is left growing and overfitting the data, then it is pruned using some predefined approach. A way to verify whether the correct tree size is reached is the *training and validation set* approach. In this approach, the available data are separated into two sets: the training set (used to form the decision tree) and the validation set (used to evaluate the accuracy of the tree).

Decision trees for uncertain data

[Tsang et al., 2011] applies the decision tree algorithm in the field of uncertain data.

Usually, when considering an attribute a of an object, we think about values that are either categorical or numerical. However, there are many scenarios in which uncertainty is introduced in the data set, e.g., when a temperature is captured by a sensor network it could be uncertain because of noise. In these cases the value of a is captured by a range of possible values instead of a single point value. Each possible value v can be assumed with a given probability $p(v)$; consequently, a probability distribution $f(a)$ over the range of possible values $[l, u]$ is defined, and this distribution will synthesize the value of a.

In this scenario, a test object o_t contains uncertain attributes, each one described by a PDF: $o_t = \langle f(a_1), \ldots, f(a_N) \rangle$. A *classification model* is a model that maps o_t to a probability distribution over $\mathcal{C} = \{P, N\}$, where P and N are classes.

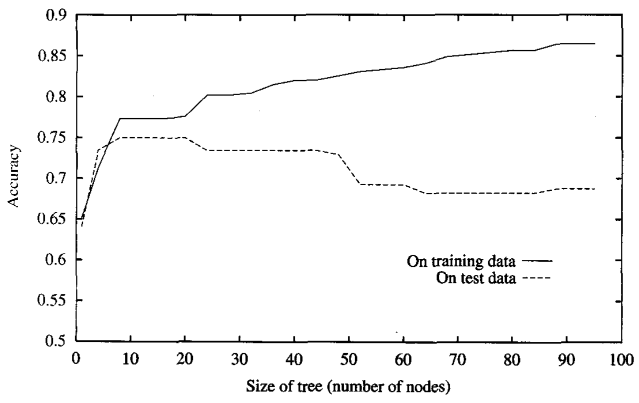


Figure 2.13: Analysis on the performance of the decision tree algorithm

Algorithm 1: ID3 algorithm for the construction of a decision tree. A is the target attribute that we have to learn

```
Input: Examples $\mathcal{O}$, Attributes $\mathcal{A}$, Target attribute $A$
Output: Decision tree $D$

Create root $r$ for the tree $D$;
if all elements in $\mathcal{O}$ are of class $P$ then
    $label(r) = P$;
else if all elements in $\mathcal{O}$ are of class $N$ then
    $label(r) = N$;
else if $\mathcal{A}$ is empty then
    $label(r) =$ most common value of $A$ in $\mathcal{O}$;
else
    $T = \arg\max_{a \in \mathcal{A}} G(a)$;
    $test(r) = T$;
    for all values $v_i$ of $a$ do
        Add a new branch to $r$;
        $\mathcal{O}_{v_i} = \{o \in \mathcal{O} : v_{a,o} = v_i\}$;
        if $\mathcal{O}_{v_i} = \emptyset$ then
            $label(D_{sub}) =$ most common value of $A$ in $\mathcal{O}$;
        else
            $D_{sub} =$ ID3($\mathcal{O}_{v_i}$, $A$, $\mathcal{A} \setminus \{a\}$);
        Add $D_{sub}$ as child of the new branch;
return $r$
```

Given a node n associated to the attribute a_{j_n}, the following rule is applied:

$$T = \begin{cases} v_{a,o} \leq z_n & go\ right \\ otherwise & go\ left \end{cases}$$

z_n is called *split point* for the attribute a_{j_n}, since this is the value used to split between the left and the right branch.

An object o_x is associated to a weight $w_x \in [0, 1]$, that is updated while running the algorithm. The quantity $\phi_n(x; o_x, w_x)$ is the conditional probability that o_x is labeled c when a tree with root at n is used to classify o_x with weight w_x. If n is associated to the attribute a_{j_n} and the split point z_n, then we can compute the probability p_L of going to the left branch after the test as:

$$p_L = \int_{l_{jn}}^{z_n} f(t)dt$$

Consequently, the probability p_R of going to the right branch after the test is:

$$p_R = 1 - p_L$$

o_x is then divided into two sub-objects o_L and o_R. Object o_L will have weight $w_L = w_x \cdot p_L$; moreover, its feature vector is copied from the one of o_x, apart from $f(a_{j_n})$, which is computed as follows:

$$f(a_{j_n}) = \begin{cases} f(a_{j_n})/w_L & x \in [l_{j_n}, z_n] \\ 0 & otherwise \end{cases}$$

Object o_R is built in the same way.

Thus, we can define:

$$\phi_n(c; o_x, w_x) = p_L \cdot \phi_{n_L}(c; o_l, w_L) + p_R \cdot \phi_{n_R}(c; o_R, w_R)$$

where n_L and n_R are the left and right child of node n respectively. If n is a leaf node,

$$\phi_n(c; o_x, w_x) = w_x \cdot P_n(c)$$

Finally, for each class c, $P(c) = \phi_r(c; o_t, 1)$, where r is the root node of the decision tree. This probability indicates how likely it is that the test object o_t has class label c.

The simpler way of handling uncertain data is called *averaging*. In this case, for each object o_i and attribute a_j the mean value $v_{i,j} = \int_{l_{i,j}}^{u_{i,j}} x f_{i,j}(x)dx$ is taken as representative value.

At each node n, the objects are checked in order to verify whether they all have the same class label c. If this is the case, then n is set as a leaf node and $P_n(c) = 1$. Otherwise, an attribute a_j is selected together with a split point z_n and the objects are divided into the left and right subsets; $P_n(c)$ is set equal to the fraction of objects that are labeled as c. All the attributes a_j are evaluated, each one using its possible values $\{v_{j,1}, \dots, v_{j,N}\}$ as split points z_n. At the end, the pair $\langle a_j, z_n \rangle$ that minimizes the entropy of the probabilities is chosen as a test for the current node.

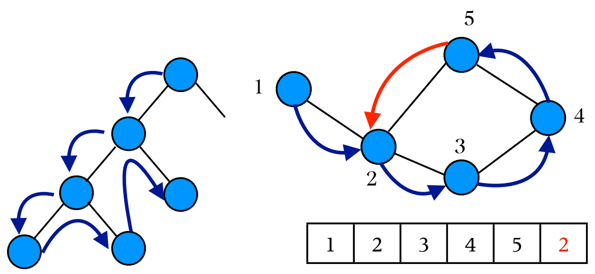
2.4 Mathematical background

2.4.1 Depth-First Search in graphs

In the following, we introduce the main concepts behind the Depth-First Search algorithm in graphs, which will be used in Chapter 5.

A *search algorithm* is an algorithm used for traversing trees and graphs. The idea is to incrementally explore paths from the root(s). A *frontier* is defined as the set of paths from the start node that have been explored. Initially, the frontier contains the root node (in case of multiple root nodes, the algorithm is iterated for each of them). Then, as the algorithm proceeds exploring the graph, the frontier expands into unexplored nodes until a leaf is encountered.

Depth-First search is a search graph whose search starts at the root, and navigates as deep as possible in the graph before backtracking. In a phylogeny, the search starts at nodes containing only original content (i.e., the roots) and proceeds visiting those nodes that are duplicates of the root. The frontier acts like a stack, in which nodes are stored and then visited one at a time. Paths are visited in a depth-first manner: a path is completely visited, from the root to the leaf, before trying an alternative path. The *backtracking* allows to select a first alternative at each node, and to backtrack to the next alternative when all the paths from the first selection have been visited (see Figure 2.14(a)).



(a) DFS algorithm running on a tree (b) DFS for cycle detection

Figure 2.14: DFS algorithm

Let $G(V, E)$ denote a graph, where V is the set of vertices and E is the set of edges. The exploration made by DFS takes time $O(|E|)$ and uses space $O(|V|)$ in the worst case for stacking vertices on the current search path.

In case of cyclic graphs DFS allows to detect cycles, since each time a node is visited it is queued in the stack, and whenever the same node appears twice in the stack that means that a cycle has just been visited.

Chapter 3

Optimizing active crowdsourcing: Fighting uncertainty on structured data

An emerging trend in data processing is *active crowdsourcing* [Ipeirotis and Gabrilovich, 2014], defined as the systematic engagement of humans in the resolution of tasks through online distributed work. This approach combines human and automatic computation in order to solve complex problems, and has been applied to a variety of data and query processing tasks, including multimedia content analysis, data cleaning, semantic data integration, and query answering.

Usually the data that is treated via crowdsourcing techniques is neither structured nor standard, and thus difficult to be processed in an automatic way. Two classical examples of non-structured data are multimedia content and uncertain data.

In the case of multimedia content, its analysis requires to posses a complex knowledge base to understand what it contains, which is not typical of machines. For instance, if we ask a machine to identify a dog in an image, we will be unlikely to receive a correct answer, since a machine does not have the concept of 'what is a dog ', and for sure even if it is trained to recognize N types of dogs, since dogs are all different between each other, it is not assured that the machine will recognize the $(N + 1)$-th type. Moreover, this automatic analysis becomes more and more difficult as the multimedia content becomes more and more complex (e.g., from image to audio content to video). On the other hand, a human possesses the required knowledge base, and thus is able to recognize multimedia content with little effort.

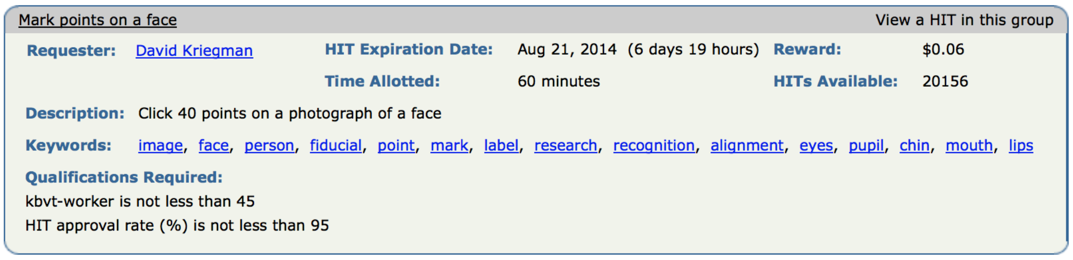
In the case of uncertain data processing, data is probabilistic instead of deterministic. This uncertainty may depend on either errors in the data collection, or low precision on the tools that were used to collect the data. In order to deal with this data, an automatic process needs to redefine it, reduce its uncertainty and approach as much as possible the deterministic version of it. This is usually done via the collection of further information on the data, which may be collected from other sources and used to reduce errors, or introduce more precision. Again, humans could possess this new information and deliver it to us with little effort.

In all these scenarios, we need a standard way to ask the required information to users: *i)* as many time as needed; *ii)* using always the same interface.

Consequently, what is usually done is to design a *crowd task*, i.e., a task that is submitted to a crowd of users and allows us to collect the required information. Each task can be usually solved by any user in the crowd, so that, when the answer is collected, we do not know the identity of the answer's author.

Crowd tasks are published on *crowdsourcing platforms*, such as Amazon Mechanical Turk[1], Crowdflower[2], Microtask[3]. Humans that participate are usually paid a certain amount of money for each task (or set of tasks), so that the reward prompt them in answering more tasks. An

[1]www.mturk.com
[2]http://www.crowdflower.com
[3]http://www.microtask.com


Mark points on a face | View a HIT in this group

Requester:	David Kriegman	**HIT Expiration Date:**	Aug 21, 2014 (6 days 19 hours)	**Reward:**	$0.06
		Time Allotted:	60 minutes	**HITs Available:**	20156

Description: Click 40 points on a photograph of a face

Keywords: image, face, person, fiducial, point, mark, label, research, recognition, alignment, eyes, pupil, chin, mouth, lips

Qualifications Required:
kbvt-worker is not less than 45
HIT approval rate (%) is not less than 95

Figure 3.1: An example of task on Amazon Mechanical Turk. Here, a task is called a HIT. Notice that in this case the requirement is the one of processing multimedia content manually, while the reward for each task is 0.06$

example of task posted on Amazon Mechanical Turk is shown in Figure 3.1.

Unfortunately for us, choosing tasks whose answers will bring us little information makes us waste budget, and would require us to post more and more tasks to the crowdsourcing platform, in need of new information. Moreover, crowdsourcing has problems of its own [Allahbakhsh et al., 2013]: the output of humans is uncertain, too, and thus the additional knowledge must be properly integrated, notably by explicitly taking into account the noisiness of human judgment and by aggregating the responses of multiple contributors. Due to this redundancy, further significant budget savings may be achieved by avoiding to post even a small amount of tasks.

These problems require an appropriate policy in the formulation of the tasks to submit to the crowd, aimed at reaching the maximum reduction of uncertainty with the smallest number of crowd task executions.

In this Chapter we introduce a use case for the application of active crowdsourcing techniques, based on the context of top-K queries.

3.1 The scenario: Uncertainty on top-K query results

Both social media and sensing infrastructures are producing an unprecedented mass of data that are at the base of numerous applications in such fields as information retrieval, data integration, location-based services, monitoring and surveillance, predictive modeling of natural and economic phenomena, public health, and more. The common characteristic of both sensor data and user-generated content is their uncertain nature, due to either the noise inherent to sensors or the imprecision of human contributions. Therefore query processing over uncertain data has become an active research field [Wang et al., 2013c], where solutions are being sought for coping with the two main uncertainty factors inherent in this class of applications: the approximate nature of users' information needs and the uncertainty residing in the queried data.

In the well-known class of applications commonly referred to as "top-K queries" [Li et al., 2011], the objective is to find the best K objects matching the user's information need, formulated as a scoring function over the objects' attribute values. If both the data and the scoring function are deterministic, the best K objects can be univocally determined and totally ordered so as to produce a single ranked result set (as long as ties are broken by some deterministic rule).

However, in application scenarios involving uncertain data and fuzzy information needs, this does not hold. For example, in a large social network the importance of a given user may be computed as a fuzzy mixture of several characteristics, such as her network centrality, level of activity, expertise, and topical affinity. A viral marketing campaign may try to identify the "best"K users and exploit their prominence to spread the popularity of a product [Kempe et al., 2003]. Another instance occurs when sorting videos for recency or popularity in a video sharing site [Cha et al., 2009]: for example, the video timestamps may be uncertain because the files were annotated at a coarse granularity level (e.g., the day), or perhaps because similar but not identical types of annotations are available (e.g., upload instead of creation time). Sometimes, data processing may also be a source of uncertainty; for example, when tagging images with a visual quality or representativeness index, the score may be algorithmically computed as a probability distribution, with a spread related to the confidence of the algorithm employed to estimate quality [Kennedy and Naaman, 2008, Sheikh et al., 2006].

Furthermore, uncertainty may also derive from the user's information need itself; for example, when ranking apartments for sale, their value depends on the weights assigned to price, size, location, etc., which may be uncertain because they were specified only qualitatively by the user or estimated by a learning-to-rank algorithm [Yu et al., 2005].

When either the attribute values or the scoring function are nondeterministic, there may be no consensus on a single ordering, but rather a space of possible orderings. For example, a query for the top-K most recent videos may return multiple orderings, namely all those compatible with the uncertainty of the timestamps. To determine the correct ordering, one needs to acquire additional information so as to reduce the amount of uncertainty associated with the queried data.

When data ambiguity can be resolved by human judgment, crowdsourcing becomes a viable tool for converging towards a unique or at least more determinate query result. For example, in an event detection and sorting scenario, a human could know the relative order of occurrence of two events; with this information, one could discard the incompatible orderings.

In this Chapter, we would like to define and compare task selection policies for uncertainty reduction via crowdsourcing, with emphasis on the case of top-K queries. Given a data set with uncertain values, our objective is to execute the set of tasks (in the form of questions posed to a crowd) that, within an allowed budget, minimizes the expected residual uncertainty of the result, possibly leading to a unique ordering of the top K results.

3.2 Score distribution for uncertain data

We consider the problem of answering a top-K query over a relational database table T containing N tuples. The relevance of a tuple to the query is modeled as a score.

Let $t_i \in T$ be a tuple in the database, defined over a relation schema $\mathcal{A} = \langle A_1, \ldots, A_M \rangle$, where $A_1, \ldots, A_M$ are attributes. Let $s(t_i)$ denote the score of tuple t_i, computed by applying a *scoring function* over t_i's attribute values. Generally, $s(t_i)$ is computed by using an aggregation function

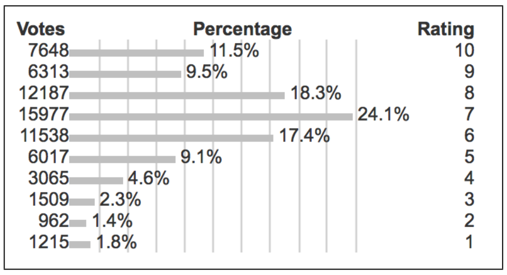
$$s(t_i) = F(s(t_i[A_1]), \ldots, s(t_i[A_M]); w_1, \ldots, w_M), \tag{3.1}$$

where $t_i[A_j]$ is the value of the j-th attribute of t_i, $s(t_i[A_j])$ its relevance with respect to the query, and w_j is the weight associated with the j-th attribute, i.e., the importance of A_j with respect to the user needs. It is common to define (3.1) as a convex sum of weighted attribute scores [Soliman et al., 2011].

When both the attribute values and the corresponding weights are known, the tuples in T can be totally ordered in descending order of $s(t_i)$ by breaking ties deterministically. Instead, if either the attribute values or the weights are uncertain, the score $s(t_i)$ can be modeled as a random variable. The corresponding probability density function (pdf) f_i can be obtained either analytically, from the knowledge of the application domain, or by fitting training data.

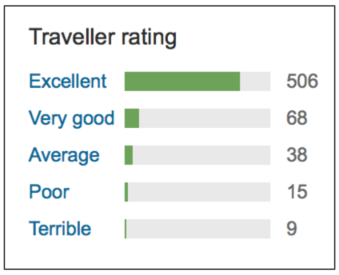
Some examples of uncertain score distributions can be found in real-life scenarios. For instance, Apartments[4] groups similar apartments, and consequently shows aggregated intervals of prices, number of bedrooms, etc (Figure 3.2(c)). Another example can be found in those websites that collect user opinions, e.g. on movies (IMDB[5], Figure 3.2(a)), hotels (TripAdvisor[6],

[4]`www.apartments.com`
[5]`www.imdb.com`
[6]`www.tripadvisor.com`



Votes	Percentage	Rating
7648	11.5%	10
6313	9.5%	9
12187	18.3%	8
15977	24.1%	7
11538	17.4%	6
6017	9.1%	5
3065	4.6%	4
1509	2.3%	3
962	1.4%	2
1215	1.8%	1

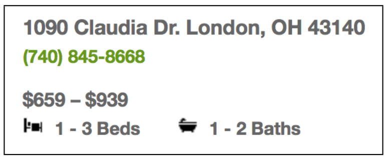
(a) IMDB



Traveller rating

Excellent	506
Very good	68
Average	38
Poor	15
Terrible	9

(b) TripAdvisor

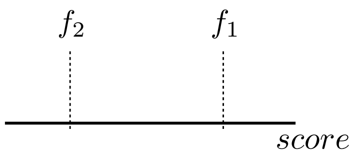


1090 Claudia Dr. London, OH 43140
(740) 845-8668

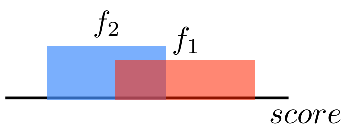
$659 – $939
1 - 3 Beds 1 - 2 Baths

(c) Apartments

Figure 3.2: Uncertain scores from real scenarios



(a) Deterministic score



(b) Uncertain score

Figure 3.3: Deterministic and uncertain scores for a pair of tuples. While a deterministic score allows one to define a total order between the tuples, an uncertain score induces a partial order over tuples

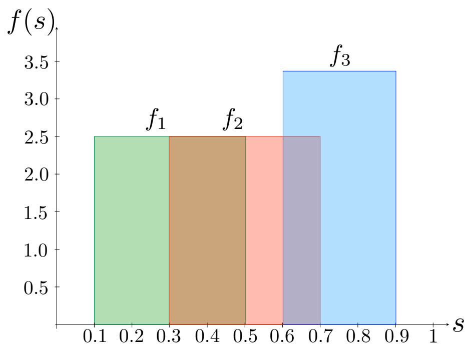
Figure 3.2(b)), in which user opinions may not coincide and thus generate score distributions for the analyzed tuples. In all these cases, it is not possible to characterize a deterministic score value, and thus a distribution can be inferred, either as an uniform distribution (as in the case of Apartments, in which we are just given with the interval extrema) or as a more complex distribution (as in the case of IMDB and TripAdvisor, in which the score distribution coincides with the distribution of votes by the users).

In the following we focus on the case in which f_i represents a continuous random variable, from which the simpler discrete case can be derived. We make very weak assumptions on the class of pdf's: f_i can be any function that can be approximated with a piecewise polynomial function defined over a finite support $[l_i, u_i]$, where l_i and u_i are the lowest and highest values that can be attained by $s(t_i)$. This approximation allows us to handle the most common probability distributions [Li and Deshpande, 2010].

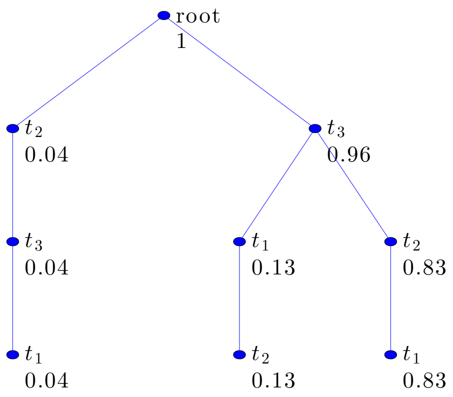
For ease of presentation, from now on we focus on uniform probability distributions. Let then δ_i denote the spread of the score distribution associated with the tuple t_i, i.e., $\delta_i = u_i - l_i$. Without loss of generality, we assume that the scores are normalized in the $[0, 1]$ interval. Therefore, $l_i \in [0, 1 - \delta_i]$ and $u_i \in [\delta_i, 1]$. Figure 3.4(a) illustrates an example with three tuples whose score is represented by means of a uniform pdf.

3.3 Tree of possible orderings

Let t_i, t_j be a pair of tuples. In the following, we will indicate with $t_i \prec t_j$ the situation in which t_i is ranked higher than t_j.



(a) Score pdf's



(b) Tree of possible orderings

Figure 3.4: (a) Score pdf's for tuples t_1, t_2 and t_3; (b) their tree of possible orderings $\mathcal{T}$. Each node is labeled with the probability of the corresponding prefix

A deterministic knowledge of the scores $s(t_i)$ induces a total order over the tuples. That is, given a pair of tuples, the tuple having a larger score is ranked higher than the tuple having a smaller score. As an example, Figure 3.3(a) shows two tuples having deterministic score. Here, $t_1 \prec t_2$, since $s_1 > s_2$.

However, the uncertain knowledge of the scores $s(t_i)$ induces a partial order over the tuples [Soliman and Ilyas, 2009]. Indeed, when the pdf's of two tuples overlap, their relative order is undefined. As a consequence, it is not possible to identify a unique ordering for the analyzed tuples. As an example, Figure 3.3(b) shows two tuples having an uncertain score. Here, we cannot state whether either $t_1 \prec t_2$ or $t_2 \prec t_1$, since the score distribution overlaps. Thus, Figure 3.3(b) generates two orderings: the one in which t_1 is ranked higher and the one in which t_2 is ranked higher.

We define the *space of possible orderings* as the set of all the total orderings compatible with the given score probability functions. This space can be represented by means of a *tree of possible orderings*, in which each node (except the root) represents a tuple t_i, and an edge from t_i to t_j indicates that t_i is ranked higher than t_j. Each path $t_{r(1)} \prec t_{r(2)} \prec \ldots \prec t_{r(N)}$, where $r(k)$ is the index of the tuple ranked at position k, is associated with a probability value, stating how probable is the prefix it represents. Complete paths from the root (excluded) to the leaf $t_{r(N)}$ (included) represent a *possible ordering* ω of the underlying set of tuples T.

For example, Figure 3.4(b) shows the tree of possible orderings obtained from the score distributions in Figure 3.4(a), where each ordering is associated with its probability value.

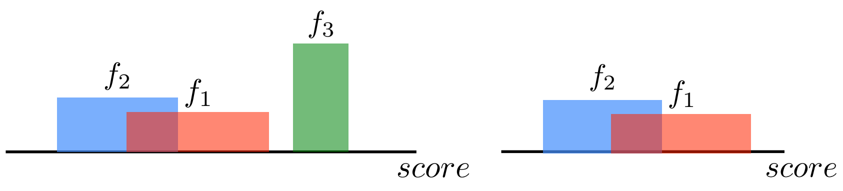
3.3.1 Construct the tree

Build the topology

A method for constructing a tree of possible orderings $\mathcal{T}$ was proposed in [Soliman and Ilyas, 2009].

Let T be a table containing the tuples $\{t_1, \ldots, t_N\}$ whose uncertain scores are represented by the pdf's $\{f_1, \ldots, f_N\}$. A tuple t_i is *dominated* if there exists at least one tuple $t_j \in T$ such that $l_j > u_i$. A *source* is a tuple $t_i \in T$ that is not dominated by any other tuple $t_j \neq t_i$. For instance, in Figure 3.5(a) t_3 is the only source, since there does not exist any other tuple that dominates it (while t_1 and t_2 are dominated by t_3). On the other hand, in Figure 3.5(b) both t_1 and t_2 are sources, since there does not exist any other tuple that dominates them.

The tree is built as described in Algorithm 2, by calling `buildTPO`(T). First, a dummy root node (not associated with any tuple) is created. Then, the sources are extracted from T and



(a) t_3 is a source (b) t_1 and t_2 are sources

Figure 3.5: A source is a tuple t_i that is not dominated by other tuples. Sources can be either unique (a) or multiple (b)

Algorithm 2: Building the tree of possible orderings

```
Function: buildTPO.
Input: Tuple set $T$. Output: The tree of possible orderings for the tuples in $T$.
  1. $r :=$ node with label 'root'; // the dummy root node
  2. addNodesToTPO($T$, $r$);
  3. return tree rooted in $r$;
------------------------------------------------------------------------
Subroutine: addNodesToTPO.
Input: Tuple set $T$, Root note $r$. Side effect: Adding descendants to $r$.
  4. for each $t_i \in T$
  5.   if $t_i$ is source for $T$
  6.     $n =$ node with label $t_i$;
  7.     Add $n$ as a child of $r$;
  8.     addNodesToTPO($T \setminus \{t_i\}$, $n$);
```

attached as children of the root. Next, each extracted source is used as a root for computing the next level of the tree. Finding the sources at each level of the tree requires comparing each tuple with all the other tuples in the current table T. Hence, the asymptotic time complexity of building the tree up to level K is $O(KN^2)$.

Compute the orderings probability

Once the structure of the tree is determined, the probability $\Pr(\omega)$ of any ordering ω in the tree can be computed, e.g., with the *generating functions* technique [Li and Deshpande, 2010] with asymptotic time complexity $O(N^2)$, or via Monte Carlo sampling.

In the following, we illustrate the procedure based on [Li and Deshpande, 2010] to compute $\Pr(\omega)$ for each ordering ω in the tree $\mathcal{T}$. For ease of presentation, we will focus on uniform distribution, although the same reasoning can be done for every distribution that can be approximated with a piecewise polynomial function. Without loss of generality, we consider an ordering $t_1 \prec \ldots \prec t_N$, call it ω, for which t_i appears at rank i (i.e., $r(i) = i$) for $1 \leq i \leq N$. Then, the probability $\Pr(\omega)$ is given by the integral:

$$\Pr(\omega) = \int_{-\infty}^{\infty} \int_{x_N}^{+\infty} \ldots \int_{x_2}^{+\infty} f_N(x_N) \ldots f_1(x_1) dx_1 \ldots dx_N, \tag{3.2}$$

where f_i is the score pdf associated with tuple t_i in ω.

Let I be the set of P *small intervals*, i.e., the intervals generated by the tuples pdf's extrema, as shown in Figure 3.6.

We start by integrating the pdf associated with the top-1 tuple in the ordering ω. Once the i-th integral is computed, we switch to the integration of the $(i+1)$-th pdf. At each step, the polynomial $\mathbf{P}_j(x_i)$ (corresponding to the result of the i-th integral on the small interval I_j) is stored for each small interval I_j.

Given a polynomial $\mathbf{P}_j(x_i)$ corresponding to the j-th small interval, there could be a set of other intervals I_{k_i} (usually adjacent) such that $\mathbf{P}_j(x_i) = \mathbf{P}_{k_i}(x_i)$. Let $I_{\mathbf{k}} = \{I_{k_i}\}$ be the set of

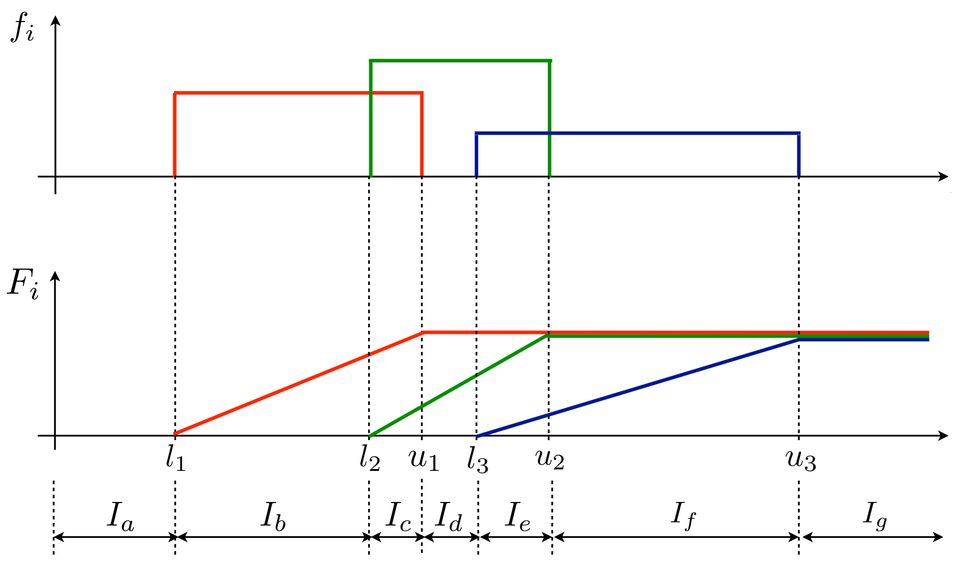


Figure 3.6: Small intervals: $I = \{I_a, I_b, I_c, I_d, I_e, I_f, I_g\}$. f_i is the score pdf for the tuple i, while F_i is its cdf

Algorithm 3: Computing ordering probabilities

```
Function: computeOrderingProbability.
Input: Ordering $\omega = \langle t_1, \dots, t_N \rangle$, Small Intervals $I$. Output: Probability $\Pr(\omega)$.
 1. Set $\mathbf{P}_j(x_1) = 1$ for every small interval $I_j \in I$
 2. for $i = 1$ to $N - 1$
 3.     for each $I_j \in I$
 4.         if $i <> 1$
 5.             $\mathbf{P}_j(x_i) = \int_{x_i}^{u_\mathbf{k}} \mathbf{P}_j(x_{i-1})dx_{i-1} + \sum_{n=k_M+1}^{P} \int_{l_n}^{u_n} \mathbf{P}_n(x_{i-1})dx_{i-1}$
 6.         else if overlap($I_j$,$f_i$)
 7.             $\mathbf{P}_j(x_i) = \mathbf{P}_j(x_i)\frac{1}{u_i - l_i}$;
 8.         else
 9.             $\mathbf{P}_j(x_i) = 0$;
10. $\Pr(\omega) = \sum_{I_{N_i} \in I_\mathbf{N}} \int_{l_{N_i}}^{u_{N_i}} \mathbf{P}_{N_i}(x_N)dx_N$
```

these small intervals, having cardinality M and extrema $[l_\mathbf{k}, u_\mathbf{k}]$. The polynomial $\mathbf{P}_j(x_i)$ is thus computed as:

$$\mathbf{P}_j(x_i) = \int_{x_i}^{u_\mathbf{k}} \mathbf{P}_j(x_{i-1})dx_{i-1} + \sum_{n=k_M+1}^{P} \int_{l_n}^{u_n} \mathbf{P}_n(x_{i-1})dx_{i-1}$$

When the i-th integral is evaluated, its result is multiplied by the pdf of the i-th tuple:

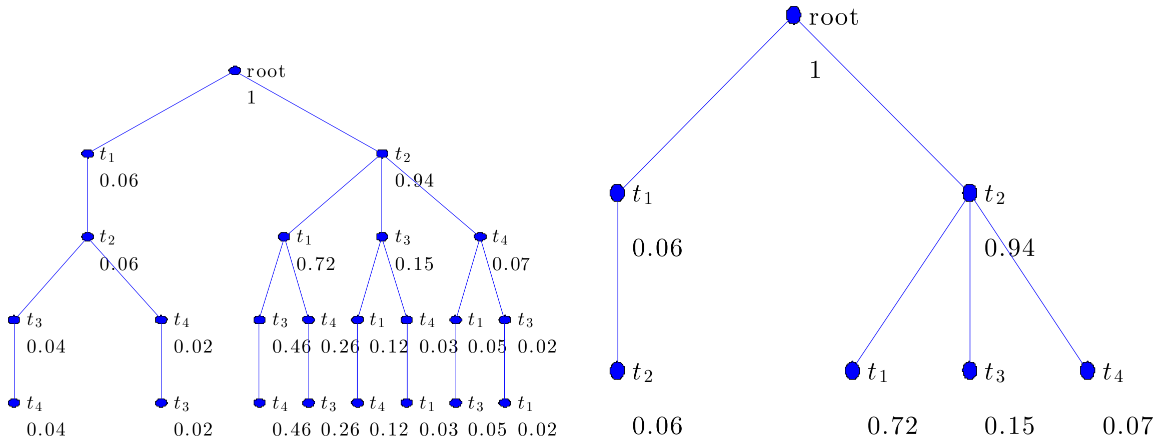
- the polynomials $\mathbf{P}_j(x_i)$ that overlap the pdf are multiplied by $\frac{1}{u_i - l_i}$
- the polynomials $\mathbf{P}_j(x_i)$ that do not overlap the pdf are set equal to 0

Evaluating the N-th integral in the chain results in the ordering probability $\Pr(\omega)$. If $I_\mathbf{N}$ is the set of small intervals that overlap the N-th tuple:

$$\Pr(\omega) = \sum_{I_{N_i} \in I_\mathbf{N}} \int_{l_{N_i}}^{u_{N_i}} \mathbf{P}_{N_i}(x_N)dx_N$$

Algorithm 3 shows how to compute the probability of an ordering ω.

The asymptotic time complexity of computing $\Pr(\omega)$ can be obtained by observing that the computation of (3.2) requires the evaluation of N sub-integrals; each sub-integral is solved by



(a) A complete tree of possible orderings $\mathcal{T}$ with $N = 4$ (b) Tree of possible orderings $\mathcal{T}_2$ cut at $K = 2$

Figure 3.7: (a) Tree of possible orderings $\mathcal{T}$; (b) its cut at depth $K = 2$.

splitting the domain of real numbers in up to $2N+1$ intervals delimited by $\{l_i, u_i\}$, $i = 1, \ldots, N$, and then iterating over all such intervals, which has a complexity of $O(N^2)$ [Li and Deshpande, 2010].

Finally, since a complete tree contains $N!$ leaves (in case every score distribution overlap with all the others), computing the probabilities for each ordering ω in the tree has complexity of $O(N^2 N!)$ (in the worst case).

Resulting tree

Figure 3.4(b) shows the tree of possible orderings obtained from the score distributions in Figure 3.4(a), along with the probability of each ordering, indicated next to each leaf. Each internal node n at depth d is associated with a probability $\Pr(n)$, obtained by summing up the probabilities of the children of n; such a value denotes the probability of the prefix of length d formed by the nodes along the path from the root to node n.

3.3.2 Limit the tree at depth K

We observe that processing a top-K query over uncertain data only requires computing the orderings of *the first* K tuples compatible with the pdf's of the tuple scores. In other words, when a top-K query is posed, only the sub-tree $\mathcal{T}_K$ of possible orderings up to depth K is relevant to answer the query.

Building the complete tree $\mathcal{T}$ of depth N is thus unneeded, as the probabilities $\Pr(\omega_K)$ for each $\omega_K \in \mathcal{T}_K$ can be computed without knowing $\mathcal{T}$ and its probabilities, and thus much more efficiently. This is done by generating the tree of possible orderings up to depth K and computing its orderings probabilities. Clearly this procedure returns the same result obtained by summing up all the probabilities of the orderings in $\mathcal{T}$ with prefix ω_K, but is much more efficient. Indeed, while $|\mathcal{T}|$ increases exponentially with N and δ, $|\mathcal{T}_K|$ is typically slightly larger than K.

Figure 3.7(a) shows an example tree of possible orderings with 4 tuples; Figure 3.7(b) shows the same tree of possible orderings when only the first $K = 2$ levels are considered.

3.3.3 Optimization: Database shrink

[Soliman and Ilyas, 2009] propose an extended approach for shrinking the database and avoid useless comparisons between tuples that will not be included in the top-K prefix. This speeds up the procedure used for building the tree of possible orderings, since as a result only K tuples will be included in T, and thus the asymptotic time complexity of building the tree is $O(K^3)$ instead of $O(KN^2)$ (where $K << N$).

Here, a tuple t_i is called *K-dominated* if at least K other records in T dominate t_i. The idea is that any K-dominated tuple can be ignored while building the tree of possible orderings up to depth K, since those tuples are not meant to be included in any ordering of length K.

The shrinking procedure assumes to build a list L in which tuples are ordered in descending upper bound (u_i) order, Moreover, we assume that the tuple t_K having the K-th largest lower bound l_K is known. Then, a binary search is conducted to find a tuple $\hat{t}$ such that $\hat{t}$ is dominated by t_K and $\hat{t}$ is the tuple located in the highest position of L. The tuple $\hat{t}$ is K-dominated by definition, and thus it can be discarded. Moreover, if p is the position of $\hat{t}$ in L, then all the tuples in L having positions greater than p can be discarded too, since they are K-dominated.

3.3.4 Computing $\Pr(r(t_i) = j)$

The work proposed by [Li and Deshpande, 2010] can be used to compute the probability of a tuple t_i to be located at level j in the tree, i.e., $\Pr(r(t_i) = j)$.

Let F_i and f_i be the cdf and the pdf of the score for t_i, respectively. The support of f_i is $[l_i, u_i]$. If the pdf is uniform, the cdf for t_i is defined as follows:

$$F_i(\ell) = \begin{cases} 0 & \ell \leq l_i \\ \frac{\ell - l_i}{u_i - l_i} & l_i < \ell < u_i \\ 1 & \ell \geq u_i \end{cases}$$

Moreover, let $\bar{F}_i = 1 - F_i$.

Let I be the set of small intervals, and I_i be the subset of those contained in the support of f_i.

Let $T_{\hat{i}} = T \backslash t_i$ and $F_{\hat{i},m,k}$ be the cdf of the score the m-th object in $T_{\hat{i}}$ over the k-th small interval. Moreover, let $B = \{B_1, \ldots, B_k\}$ be the set of binary sequences whose number of 1s is equal to $j-1$, i.e., $\sum_{m=1}^{|B_b|} B_b(m) = j-1$. For instance, when $j = 2$, $B = \{01, 10\}$.

Given $I_{ik} \in I_i$, a contribution for $\Pr(r(t_i) = j)$ is:

$$p_{ik} = \sum_{b=1}^{|B|} x \int_{l_{ik}}^{u_{ik}} \frac{1}{u_i - l_i} \prod_{m=1}^{|T_i|} [(1 - B_b(m))F_{\hat{i},m,k}(\ell) + B_b(m)\bar{F}_{\hat{i},m,k}(\ell)x] d\ell$$

The probability $\Pr(r(t_i) = j)$ is finally obtained by summing all the contributions p_{ik}:

$$\Pr(r(t_i) = j) = \sum_{I_{ik} \in I_i} p_{ik}$$

3.3.5 Statistics on the tree of possible orderings

Number of overlaps between score distributions

Uncertainty on the relative order of pairs of tuples happens when the associated score distributions overlap. We would like to estimate the expected number of overlaps between the tuples' score pdf's.

For this purpose, let δ be the spread of the score pdf's, i.e., $\delta = u_i - l_i$. The probability that a pair of score pdf's $s(t_i)$, $s(t_j)$ overlap is computed as the probability $\Pr(|l_i - l_j| < \delta)$, since whenever the distance between the lower bounds of the score pdf's is less than the pdf's spread δ, an overlap is encountered. This probability is evaluated by computing the highlighted area in Figure 3.8, leading to:

$$\Pr(|l_i - l_j| < \delta) = \frac{(1-\delta)^2 - \frac{(1-2\delta)^2}{2} \cdot 2}{(1-\delta)^2} = \frac{2\delta - 3\delta^2}{(1-\delta)^2}$$

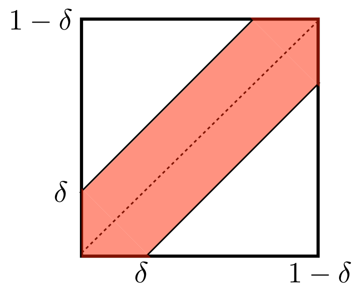


Figure 3.8: Computing the probability of having an overlap $\Pr(|l_1 - l_2| < \delta)$

When considering a pair of tuples in T, they may or may not overlap. Thus, the probability distribution of the number of overlaps can be seen as a binomial, whose expected value is:

$$\mathbb{E}[O] = \binom{n}{2} \Pr(|l_i - l_j| < \delta)$$

where n is the maximum number of overlaps between N pdf's, i.e., $n = \frac{N(N-1)}{2}$.

Number of dominant nodes in the tree

A *dominant tuple* is a tuple contained in one of the top-K levels of the tree. The expected number of dominant tuples $\mathbb{E}[D_K]$ in the first K levels of the tree can be computed as follows:

$$\mathbb{E}[D_K] = \sum_{n=1}^{N} n \Pr(n, K)$$

where $\Pr(n, K)$ is the probability of having exactly n tuples in the first K levels. This probability can be computed as:

$$\Pr(n, K) = \sum_{T_n \in T} \Pr(T_n \in \{1, \ldots, K\})$$

where T_n is a set of n tuples extracted from T and $\Pr(T_n \in \{1, \ldots, K\})$ is the probability that the n tuples in T_n are the tuples contained in the first K levels of the tree. It follows that:

$$\Pr(n, K) = \sum_{T_n \in T} \left[\prod_{t_i \in T_n} \Pr(t_i \in \{1, \ldots, K\}) \cdot \prod_{t_j \in T \setminus T_n} \Pr(t_i \in \{K+1, \ldots, N\}) \right]$$

where $\Pr(t_i \in \{1, \ldots, K\})$ is the probability that t_i is in one of the first K levels of the tree and $\Pr(t_i \in \{K+1, \ldots, N\})$ is the probability that t_j is not in the first K levels.

The probability $\Pr(t_i \in \{1, \ldots, K\})$ can be found by computing $\Pr(r(t_i) = \ell)$, i.e., the probability of t_i of being ranked at the level ℓ in the tree (see Section 3.3.4).

Thus, we find that:

$$\Pr(t_i \in \{1, \ldots, K\}) = \sum_{\ell=1}^{K} \Pr(r(t_i) = \ell)$$

Eventually, the expected number of tuples in the first K levels of the tree can be computed as follows:

$$\mathbb{E}[D_K] = \sum_{n=1}^{N} n \cdot \sum_{T_n \in T} \left[\prod_{t_i \in T_n} \sum_{\ell=1}^{K} \Pr(r(t_i) = \ell) \cdot \prod_{t_j \in T \setminus T_n} \sum_{\ell=K+1}^{N} \Pr(r(t_j) = \ell) \right]$$

3.4 Distance metrics

In the following, we introduce a distance metrics that quantifies the difference between two orderings ω_i and ω_j (both contained in the tree of possible orderings $\mathcal{T}_K$). This metrics is an adaptation of the weighted Kendall-Tau distance [Kumar and Vassilvitskii, 2010] to the top-K context, as follows.

Let ω_1 and ω_2 be two orderings in the tree $\mathcal{T}_K$. Let us indicate with $\sigma_i(t)$ the position of tuple t in ω_i, and with $\mathcal{I}(\omega_1, \omega_2)$ the set of pairs of tuples whose order is inverted in ω_1 and ω_2, i.e.

$$\mathcal{I}(\omega_1,\omega_2)= \{(t,t')|(\sigma_1(t)-\sigma_1(t'))(\sigma_2(t)-\sigma_2(t'))<0, t\in\omega_1, t'\in\omega_2\}. \tag{3.3}$$

We define:

$$d(\omega_1,\omega_2) = \sum_{(t,t')\in\mathcal{I}(\omega_1,\omega_2)} \pi(\sigma_1(t),\sigma_2(t))\cdot\pi(\sigma_1(t'),\sigma_2(t')) \tag{3.4}$$

where $\pi(\cdot,\cdot)$ is a weight that decreases as its arguments increase. In this way, inversions involving a tuple near the head of the orderings weigh more than near the tail.

In the following, we illustrate two possible choices of $\pi(\cdot,\cdot)$. The first choice of $\pi(\cdot,\cdot)$ let them vary with a logarithmic trend (adapted from [Kumar and Vassilvitskii, 2010] to the top-K context):

$$\pi(i,j) = \begin{cases} \frac{1}{\log(\min(i,j)+1)} & (i > K \vee \\ & j > K) \\ \frac{1}{\log_2(\min(i,j)+1)} - \frac{1}{\log_2(\max(i,j)+2)} & otherwise \end{cases} \tag{3.5}$$

The second choice of $\pi(\cdot,\cdot)$ let them vary with a linear trend:

$$\pi(i,j) = \begin{cases} \frac{K-\min(i,j)+2}{K} & i > K \vee j > K \\ \frac{|i-j|+1}{K} & otherwise \end{cases} \tag{3.6}$$

3.4.1 Application in the bioinformatic field

Our extension for the weighted Kendall Tau distance can be applied to many contexts in which lists are used to collect and order data. In the following, we introduce an application of the metrics in the bioinformatic field, where there is the need of comparing lists of gene function annotations (produced as output from different algorithms) to assess their difference [Chicco et al., 2014].

In bioinformatics, a *gene function annotation* $A = \langle g, t\rangle$ is the association of a gene g with a term t that represents a functional feature; such term is usually either part of a terminology or structured in an ontology (e.g., the Gene Ontology (GO) [Consortium, 2001]). The annotation A states that the gene g has a specific physical/behavioral characteristics represented by the annotation term t. For instance, $A = \langle$`VMA9`, `transmembrane transport`$\rangle$ is an annotation indicating that the gene `VMA9` is involved in the transmembrane transport.

Despite their biological significance, it may happen that either available annotations are erroneous (due to evaluation errors) or some annotations may be missing [Karp, 1998]. Thus, computational methods and software tools are often used to produce ranked lists of reliably predicted annotations, called *annotation lists*. These lists rank annotations according to their probability of being correct (from the most probable to the least probable).

Unfortunately, all these methods involve key parameters that influence the output, and there does not exist any study about how changing a specific parameter changes the resulting annotation lists. Thus, a similarity measure that compares different output lists in order to assess their difference is required. To accomplish this, the most consistent measures are the Spearman rank correlation coefficient and the Kendall-tau rank distance. In the following, we explain how the adapted top-K versions of these measures can be used for this purpose.

Annotation categories

Let $\mathbf{A} = [a_{ij}]$ be a $m \times n$ matrix, representing the *training set*, i.e., a matrix where the known annotations are enlisted. Here, each row i corresponds to a gene g_i and each column j corresponds to a functional feature term g_j of a terminology or ontology: $a_{ij} = 1$ if g_i is annotated to the feature term t_j, or $a_{ij} = 0$ otherwise.

Suppose that a *prediction algorithm* elaborates the matrix $\mathbf{A}$ to produce an output matrix $\widetilde{\mathbf{A}}$, with the same dimensions of $\mathbf{A}$ and containing the predicted annotations. Each element $\widetilde{a}_{ij}$, if not null, represents the probability that the corresponding predicted annotation $A = \langle g_i, t_j \rangle$ is true. Consequently, a high $\widetilde{a}_{ij}$ value indicates that the probability of g_i to be associated with t_j is high.

Finally, each predicted annotation $A = \langle g_i, t_j, \tilde{a}_{ij} \rangle$ is classified in one of the following categories.

- *Annotation Confirmed* (`AC`): $a_{ij} = 1 \wedge \widetilde{a}_{ij} > \theta$ (corresponding to True Positive)
- *Annotation Predicted* (`AP`): $a_{ij} = 0 \wedge \widetilde{a}_{ij} > \theta$ (corresponding to False Positive)
- *Non-Annotation Confirmed* (`NAC`): $a_{ij} = 0 \wedge \widetilde{a}_{ij} \leq \theta$ (corresponding to True Negative)
- *Annotation to be Reviewed* (`AR`): $a_{ij} = 1 \wedge \widetilde{a}_{ij} \leq \theta$ (corresponding to False Negative)

Extract the annotation list

The predicted annotations contained in the output matrix $\widetilde{\mathbf{A}}$ are extracted and sorted by descending values of $\widetilde{a}_{ij}$. The list can be fractioned in four parts: `AClist`, `APlist`, `NAClist` and `ARlist`, respectively containing those annotations that were classified as `AC`, `AP`, `NAC`, `AR`.

Compare different annotation lists

Each prediction algorithm has parameters that, when changed, lead to different output annotation lists. Thus, to understand how the selected parameter values influence the output lists, it is important to define similarity metrics that compare annotation lists resulting from different algorithm parameterizations. To this extent, in the following we present significant variants to two well-known similarity metrics, i.e. the *Spearman ρ coefficient* and the *Kendall τ distance.*

Spearman rank correlation coefficient Suppose that two annotation lists l_a and l_b are composed of the same elements (and consequently have the same length n). Given an element A_i, let x_i denote its position in l_a, y_i its position in l_b and $d_i = |x_i - y_i|$ the distance of its positions in l_a and l_b. The final normalized Spearman ρ value is computed as:

$$\rho = 1 - (6 \cdot \sum d_i^2)/(n \cdot (n^2 - 1)) \tag{3.7}$$

A maximum positive correlation $\rho = +1$ occurs when l_a and l_b are identical (i.e. with the same elements in the same order), while the maximum negative correlation $\rho = -1$ occurs when l_a and l_b contain the same elements but in inverse order.

Weighted Spearman rank correlation coefficient When two lists l_a and l_b have different length and/or contain different elements, they are not properly handled by the classical Spearman rank correlation coefficient. To manage these cases, we apply the *Weighted Spearman rank correlation coefficient* (ρ_w), which features penalty distance weights w_{si} for each element A_i absent from one of the two lists.

The penalty weight w_{si} for an element A_i in list l_a and/or l_b is computed as follows, by penalizing elements present in only one of the two lists:

$$w_{si} = \begin{cases} 1 - \frac{1}{|x_i - y_i| + 1} & \{A_i \in l_a \wedge A_i \in l_b\} \\ 1 & \text{otherwise} \end{cases} \tag{3.8}$$

Let $q = |l_a \cup l_b|$. The Weighted Spearman rank correlation coefficient ρ_w value is computed as:

$$\rho_w = \frac{\sum_{i=1}^{q} w_{si}}{q} \tag{3.9}$$

High correlation is found when $\rho_w \simeq 0$ (i.e. very few penalties are assigned), while low correlation is found when $\rho_w \simeq 1$ (i.e. many penalties are assigned). If the two lists are equal, $\rho_w = 0$, otherwise, if they have no common elements, $\rho_w = 1$.

Extended Weighted Spearman rank correlation coefficient Yet, the Weighted Spearman rank correlation coefficient has a flaw: as stated before, an annotation list can be divided in four sublists, and it would be reasonable to weigh differently the presence/absence of annotation in different sublists. In particular, since `APlist` and `NAClist` are the most relevant in bioinformatic sense, given two lists l_a and l_b (respectively containing the sublists $\langle \texttt{APlist}_a, \texttt{NAClist}_a \rangle$ and $\langle \texttt{APlist}_b, \texttt{NAClist}_b \rangle$), between the cases:

$$\begin{cases} A \in \texttt{APlist}_a \\ A \notin \texttt{APlist}_b \\ A \in \texttt{NAClist}_b \end{cases}$$

and

$$\begin{cases} A \in \texttt{APlist}_a \\ A \notin \texttt{APlist}_b \\ A \notin \texttt{NAClist}_b \end{cases}$$

we would like to penalize more this second scenario (since A falls off both $\texttt{APlist}_b$ and $\texttt{NAClist}_b$). Thus, we defined a new penalty weight v_i, more suitable to our application domain , where the first scenario gets a lower penalty than the second one. To do so, we first defined the penalized position of each element A_i in $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$), with respect to the length of the related list $\texttt{APlist}_a$ ($\texttt{APlist}_b$), as: $\hat{z}_i = z_i + n \cdot 2$, where z_i denotes the position of A_i in $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$), n is the length of the related $\texttt{APlist}_a$ ($\texttt{APlist}_b$) and 2 is a penalty factor for A_i not to be in $\texttt{APlist}_a$ ($\texttt{APlist}_b$) but in $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$). Then, we expressed the new penalty weight as follows:

$$v_{si} = \begin{cases} 1 - \frac{1}{|x_i - y_i| + 1} & \{A_i \in \texttt{APlist}_a, A_i \in \texttt{APlist}_b\} \\ 1 - \frac{1}{|x_i - \hat{z}_i| + 1} & \{A_i \notin \texttt{APlist}_a, A_i \in \texttt{NAClist}_a, A_i \in \texttt{APlist}_b\} \vee \\ & \quad \{A_i \in \texttt{APlist}_a, A_i \notin \texttt{APlist}_b, A_i \in \texttt{NAClist}_b\} \\ 1 & \{A_i \notin \texttt{APlist}_a, A_i \notin \texttt{NAClist}_a, A_i \in \texttt{APlist}_b\} \vee \\ & \quad \{A_i \in \texttt{APlist}_a, A_i \notin \texttt{APlist}_b, A_i \notin \texttt{NAClist}_b\} \end{cases} \tag{3.10}$$

where x_i is the A_i element position in $\texttt{APlist}_a$, y_i is its position in $\texttt{APlist}_b$ and $\hat{z}_i$ is its penalized position in $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$). The *Extended Spearman rank correlation coefficient* (ρ_e) is then computed as:

$$\rho_e = \frac{\sum_{i=1}^{q} v_{si}}{q} \tag{3.11}$$

As for the Weighted Spearman rank correlation coefficient, high ρ_e values lead to low correlation, while $\rho_e \simeq 0$ suggests high correlation.

Extended Kendall tau distance The *Kendall rank distance* (τ) [Kendall, 1938] is related to the number of pairwise disagreements between two lists l_a and l_b of ranked elements, i.e. the number of bubble-sort swaps needed to sort the two lists in the same order. Obviously, when the two lists are identical, $\tau = 0$. On the other hand, if the two lists are equal but in reverse order, then $\tau = 1$. Let l_a and l_b be two lists of length n. Given an element A_i, let x_i be its position in l_a and y_i its position in l_b. Thus, the set $\mathcal{K}$ of required swaps between list elements is computed as:

$$\mathcal{K}(l_a, l_b) = \{(i,j) : (x_i < y_i \wedge x_j > y_j) | (x_i > y_i \wedge x_j < y_j)\} \tag{3.12}$$

Then, the normalized Kendall rank distance τ is given by:

$$\tau = \frac{|\mathcal{K}|}{n \cdot (n-1)/2} \tag{3.13}$$

Notice that the Kendall rank distance τ does not express negative correlation between lists. Moreover, while the Spearman rank correlation coefficient ρ is focused on the distance between the ranks of each element in the two lists, the Kendall rank distance τ considers just the number of swaps in the element rank.

Weighted Kendall tau distance Similarly to what concerns the Spearman rank correlation coefficient, a flaw of the classical normalized Kendall rank distance is its usability only when the two lists have the same length and the same elements. To avoid this limitation, we introduce weights to consider, but with a penalty, the elements that are present in one list but absent from the other. The weight w_k for each swapped element A_i in a list is defined as:

$$w_{ki} = \begin{cases} \frac{1}{\log(x_i+2)} - \frac{1}{\log(x_i+3)} & \{A_i \in l_a \wedge A_i \in l_b\} \\ 0.5 & \text{otherwise} \end{cases} \tag{3.14}$$

We chose this weight function to make the weight higher if the element is in the list first positions, and lower if it is in the last ones; we consider bubble-sort swaps in the first positions more important. The *Weighted Kendall rank distance* τ_w is thus computed as:

$$\tau_w = \frac{\sum_{(i,j)\in\mathcal{K}(l_a,l_b)} w_{ki} w_{kj}}{|\mathcal{K}|} \tag{3.15}$$

As for the classical Kendall rank distance τ, low correlations correspond to high τ_w values, whereas $\tau_w = 0$ if the two lists are identical (since no swap occurs).

Extended Weighted Kendall tau distance Again, similarly to what concerns the Weighted Spearman rank correlation coefficient, all elements $\{A_i : A_i \notin \texttt{APlist}_a \vee A_i \notin \texttt{APlist}_b\}$ are weighted equally, independently if they are, or are not, included in a related list (i.e. $\texttt{NAClist}_a$ or $\texttt{NAClist}_b$). Thus, we also define a penalty weight v_{ki} for the swapped elements as:

$$v_{ki} = \begin{cases} \frac{1}{\log(x_i+2)} - \frac{1}{\log(x_i+3)} & \{A_i \in \texttt{APlist}_a, A_i \in \texttt{APlist}_b\} \\ 0.5 - h & \{A_i \notin \texttt{APlist}_a, A_i \in \texttt{NAClist}_a, A_i \in \texttt{APlist}_b\} \vee \\ & \quad \{A_i \in \texttt{APlist}_a, A_i \notin \texttt{APlist}_b, A_i \in \texttt{NAClist}_b\} \\ 0.5 & \{A_i \notin \texttt{APlist}_a, A_i \notin \texttt{NAClist}_a, A_i \in \texttt{APlist}_b\} \vee \\ & \quad \{A_i \in \texttt{APlist}_a, A_i \notin \texttt{APlist}_b, A_i \notin \texttt{NAClist}_b\} \end{cases} \tag{3.16}$$

where, when an element A_i missed from $\texttt{APlist}_a$ ($\texttt{APlist}_b$) is present in a related $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$) list, A_i gets a lower penalty weight, i.e. $v_{ki} = 0.5 - h$. In particular, the penalty reduction h can be defined as: $h = 0.5 - z_i/(m \cdot 2)$, where z_i is the position of A_i in $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$) and m is the length of $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$).

Alternatively, if the elements in $\texttt{NAClist}_a$ ($\texttt{NAClist}_b$) have an associated likelihood value, then $h = \widetilde{a}_{ij}$. (Notice that the likelihood value h used to define v_{ki} decreases along the rank). We chose the weight function in Equation 3.16 so that it decreases when the element gets a lower rank.

Thus, we introduce the *Extended Kendall rank distance* τ_e as:

$$\tau_e = \sum_{(i,j)\in\mathcal{K}(l_a,l_b)} v_{ki}v_{kj} \tag{3.17}$$

As for the Weighted Kendall rank distance, τ_e is high when $\texttt{APlist}_a$ and $\texttt{APlist}_b$ are very different, whereas $\tau_e \simeq 0$ when the two lists $\texttt{APlist}_a$ and $\texttt{APlist}_b$ are very similar.

3.5 Measure tree uncertainty

Reducing uncertainty via crowdsourcing requires acquiring additional knowledge from the crowd. Thus it becomes important to quantify the uncertainty reduction that can be expected by the execution of a crowd task.

Given a tree of possible orderings $\mathcal{T}_K$, we propose four measures to quantify its level of uncertainty. For convenience, we treat $\mathcal{T}_K$ as a set and write $\omega \in \mathcal{T}_K$ to indicate that ω is one of the orderings in $\mathcal{T}_K$, and $|\mathcal{T}_K|$ to denote the number of orderings in $\mathcal{T}_K$.

3.5.1 Entropy

The first measure relies on Shannon's entropy, which quantifies the average information conveyed by a source that emits symbols from a finite alphabet. In our context, the alphabet is represented by the orderings in $\mathcal{T}_K$. Each ordering $\omega \in \mathcal{T}_K$ is mapped into a symbol having probability $\Pr(\omega)$.

Then, $U_H(\mathcal{T}_K)$ measures the uncertainty of $\mathcal{T}_K$, based on the probabilities of its leaves:

$$U_H(\mathcal{T}_K) = -\sum_{\omega\in\mathcal{T}_K} \Pr(\omega)\log_2 \Pr(\omega). \tag{3.18}$$

When a unique total order can be determined, i.e., when the scores are deterministically known, then $U_H(\mathcal{T}_K) = 0$. Conversely, when all orderings are equally likely, which corresponds to the case of maximum uncertainty, then $U_H(\mathcal{T}_K) = \log_2 |\mathcal{T}_K|$. In general, $0 \le U_H(\mathcal{T}_K) \le \log_2 |\mathcal{T}_K|$.

Assuming the ordering probabilities are known, the procedure used for computing $U_H(\mathcal{T}_K)$ has a complexity of $O(|\mathcal{T}_K|)$.

3.5.2 Weighted entropy

The measure $U_H(\mathcal{T}_K)$ only considers the probabilities of the leaves of $\mathcal{T}_K$. However, in a top-K context, better ranked tuples are more important. Thus, we define weighted entropy as a weighted combination of entropy values at the first K levels of the tree of possible orderings:

$$U_W(\mathcal{T}_K) = \sum_{k=1}^{K} \eta(k) U_H(\mathcal{T}_k) \tag{3.19}$$

where $\eta(k)$ weighs the relevance of level k , i.e., the smaller k, the larger $\eta(k)$.

Computing $U_W(\mathcal{T}_K)$ has a complexity of $O(|\texttt{nodes}(\mathcal{T}_K)|)$.

3.5.3 ORA

The third measure is based on the idea of comparing all the orderings in $\mathcal{T}_K$ with an ordering that is *representative* in some sense. To this end, we adopt the Optimal Rank Aggregation (ORA) as the "average" ordering [Soliman et al., 2011]: the ORA is the ordering with minimum average expected distance from all other orderings in $\mathcal{T}_K$. That is,

$$\omega^{\text{ORA}} = \arg\min_{\omega \in \mathcal{T}_K} \frac{1}{|\mathcal{T}_K|} \sum_{\omega_i \in \mathcal{T}_K} d(\omega_i, \omega) \Pr(\omega_i), \tag{3.20}$$

where $d(\cdot,\cdot)$ measures the distance between two orderings. Here we use the weighted Kendall-Tau distance shown in Section 3.4.

The average expected distance from ω^{ORA} induces an uncertainty measure:

$$U_{\text{ORA}}(\mathcal{T}_K) = \frac{1}{|\mathcal{T}_K|} \sum_{\omega_i \in \mathcal{T}_K} d(\omega_i, \omega^{\text{ORA}}) \Pr(\omega_i) \tag{3.21}$$

Computing $U_{\text{ORA}}(\mathcal{T}_K)$ has a complexity of $O(|\mathcal{T}_K|^2)$.

3.5.4 MPO

The last measure is similar to U_{ORA} but refers to another representative ordering, i.e., the Most Probable Ordering (MPO) [Soliman et al., 2011]:

$$\omega^{\text{MPO}} = \arg\max_{\omega \in \mathcal{T}_K} \Pr(\omega). \tag{3.22}$$

In turn, this induces an uncertainty measure:

$$U_{\text{MPO}}(\mathcal{T}_K) = \frac{1}{|\mathcal{T}_K|} \sum_{\omega_i \in \mathcal{T}_K} d(\omega_i, \omega^{\text{MPO}}) \Pr(\omega_i) \tag{3.23}$$

Computing $U_{\text{MPO}}(\mathcal{T}_K)$ has a complexity of $O(|\mathcal{T}_K|)$.

In the following, we will drop the subscript and simply write $U(\mathcal{T}_K)$ to denote the uncertainty of a tree of possible orderings, because the implementation of our algorithms for uncertainty reduction is orthogonal to the specific way of measuring uncertainty.

3.6 Uncertainty Reduction (UR) problem

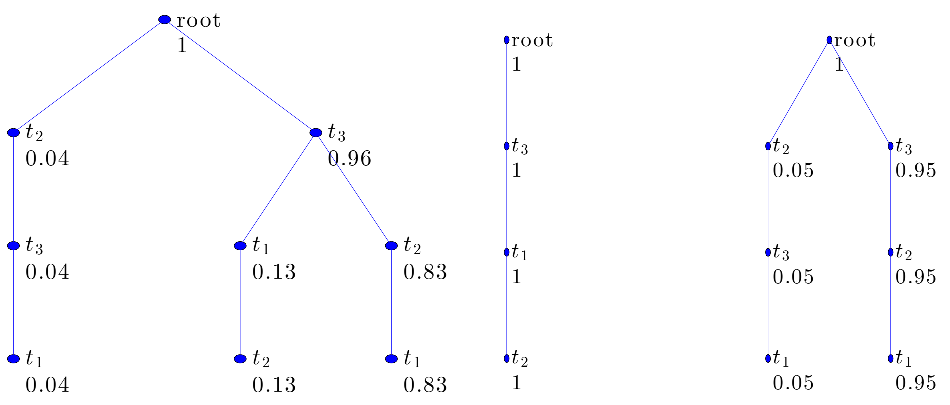
The number of orderings in a tree of possible orderings (i.e., paths from the root to a leaf in $\mathcal{T}$) depends on the number of tuples N and on the overlaps of their pdf's f_i, $i = 1, \dots, N$, and can be very large even for small values of N.

We have two main ways of reducing uncertainty in $\mathcal{T}$ to quickly converge to the correct ordering:

i) building only the first K levels of the tree of possible orderings, as was shown in Section 3.3.2, thereby focusing on $\mathcal{T}_K$ instead of $\mathcal{T}$

i) defining crowd tasks for disambiguating the relative order of tuples in order to reduce the number of orderings in $\mathcal{T}$

In the following, we illustrate how to design crowd tasks whose answers are used to disambiguate the relative order of tuples. Then, we give a definition for the Uncertainty Reduction problem.



(a) Tree of possible orderings $\mathcal{T}$ (b) Left: $\mathcal{T}^{t_1 \prec t_2}$. Right: $\mathcal{T}^{t_1 \nprec t_2}$

Figure 3.9: Tree pruning when it is known that either $t_1 \prec t_2$ or $t_2 \prec t_1$

3.6.1 Designing crowd tasks

Tree pruning

If the relative order of two tuples in a tree of possible orderings is known, e.g., as determined by a crowd answer assumed to be correct, we can prune all the paths incompatible with such an order.

In particular, when considering two tuples t_i and t_j in the full tree $\mathcal{T}$ (i.e., when $K = N$), each path in $\mathcal{T}$ agrees either with $t_i \prec t_j$ or with $t_i \nprec t_j$. Thus, $\mathcal{T}$ can be partitioned into two sub-trees:

- $\mathcal{T}^{t_i \prec t_j}$, which contains all the paths (from the root to a leaf) in $\mathcal{T}$ in which t_i is ranked higher than t_j, and
- $\mathcal{T}^{t_i \nprec t_j}$, which contains all the remaining paths.

As an example, Figure 3.9(b) shows two sub-trees derived from the tree in Figure 3.9(a) when either $t_1 \prec t_2$ or $t_1 \nprec t_2$.

Note that the leaf probabilities are always normalized so that they sum up to 1, i.e., each probability $\Pr(\omega)$ in a sub-tree $\mathcal{T}' \in \{\mathcal{T}^{t_i \prec t_j}, \mathcal{T}^{t_i \nprec t_j}\}$ is recomputed as $\frac{\Pr(\omega)}{\sum_{\omega' \in \mathcal{T}'} \Pr(\omega')}$. The sub-tree that agrees with the known relative order of t_i and t_j becomes the new tree of possible orderings.

Instead, when $K < N$, it may happen that some orderings in $\mathcal{T}_K$ are not affected by the knowledge of the relative order of some tuples. For instance, consider the tree of possible orderings $\mathcal{T}$ in Figure 3.7(a) and its restriction $\mathcal{T}_2$ to $K = 2$, shown in Figure 3.7(b). When considering the relative order of t_3 and t_4, the paths $\langle t_1, t_2 \rangle$ and $\langle t_2, t_1 \rangle$ in $\mathcal{T}_2$ belong to both $\mathcal{T}_2^{t_3 \prec t_4}$ and $\mathcal{T}_2^{t_3 \succ t_4}$, although with different probabilities, as shown in Figure 3.10. In such cases, each path ω_K in which the relative order of t_i and t_j is ambiguous must be inserted in both the sub-trees $\mathcal{T}_K^{t_i \prec t_j}$ and $\mathcal{T}_K^{t_i \succ t_j}$, and the probability of each path must be computed accordingly.

Task design

Several question types may be selected so as to gather new information about the real ordering from the crowd. Some examples are: *i)* comparison between two tuples t_i and t_j to select the most relevant; *ii)* comparison between a set of tuples $\{t_i\}$, with cardinality greater than 2 to select the most relevant; *iii)* gather new information on the score distribution, to reduce its uncertainty. However, when the tasks are too complex (e.g., finding the most relevant tuple in a huge set of tuples, manually redefining a score), they lose effectiveness, since giving an answer

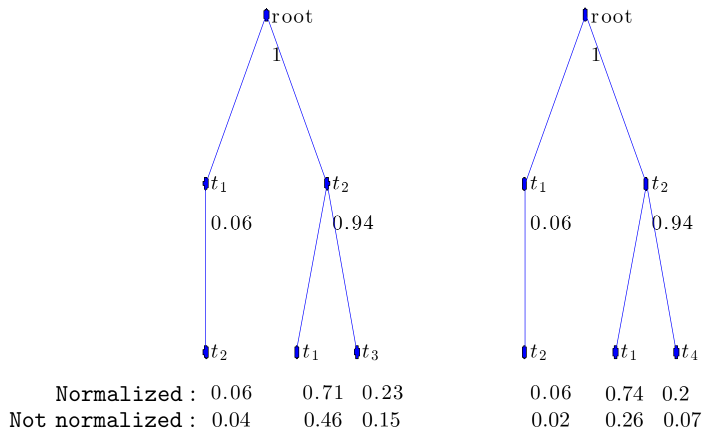

root
1
t_1
t_2
0.06
0.94
t_2
t_1
t_3
root
1
t_1
t_2
0.06
0.94
t_2
t_1
t_4
Normalized : 0.06 0.71 0.23 0.06 0.74 0.2
Not normalized : 0.04 0.46 0.15 0.02 0.26 0.07

Figure 3.10: Posing questions on trees $\mathcal{T}_K$ of depth K: normalizing probabilities

becomes more and more difficult, and thus human answer becomes more and more uncertain. Thus, among all the available tasks design, it is advisable to select the simplest one.

Hence, we consider crowd tasks expressed as questions of the form $q = t_i \overset{?}{\prec} t_j$, which compare t_i and t_j to determine which one ranks higher.

We initially assume that a crowd worker's answer $ans(q)$, which is either $t_i \prec t_j$ or $t_i \not\prec t_j$,[7] always reflects the real ordering of the tuples (we shall relax this assumption in Section 3.8). With that, we can prune from $\mathcal{T}_K$ all those paths that do not agree with the answer.

Computing $\Pr(t_i \prec t_j)$

We now show how to compute the probability $\Pr(t_i \prec t_j)$ of a tuple t_i of being ranked higher than another tuple t_j. This is equivalent to compute the probability $\Pr(s(t_1) > s(t_2))$.

Notice that $\Pr(t_i \prec t_j)$ can be equally computed by summing up the probabilities $\Pr(\omega)$ of every ordering ω in which $t_i \prec t_j$. Moreover, if we define a question $q = t_i \overset{?}{\prec} t_j$, then $\Pr(t_i \prec t_j)$ is the probability that the user will answer that $t_i \prec t_j$, leading to the output $\mathcal{T}_K^{t_i \prec t_j}$.

This probability is easily evaluated in the case $s(t_2)$ is fixed: when $\Pr(s(t_1) > s(t_2)|s(t_2))$ is known for a specific value of $s(t_2)$, then $\Pr(s(t_1) > s(t_2))$ can be found by making $s(t_2)$ vary between $-\infty$ and $+\infty$, as follows:

$$\Pr(s(t_1) > s(t_2)) = \int_{-\infty}^{+\infty} \left[\Pr(s(t_1) > s(t_2)|s(t_2)) \cdot \Pr(s(t_2))\right] ds(t_2)$$

We distinguish between six different contexts, as shown in Figure 3.11.

Context 1 (3.11(a)).

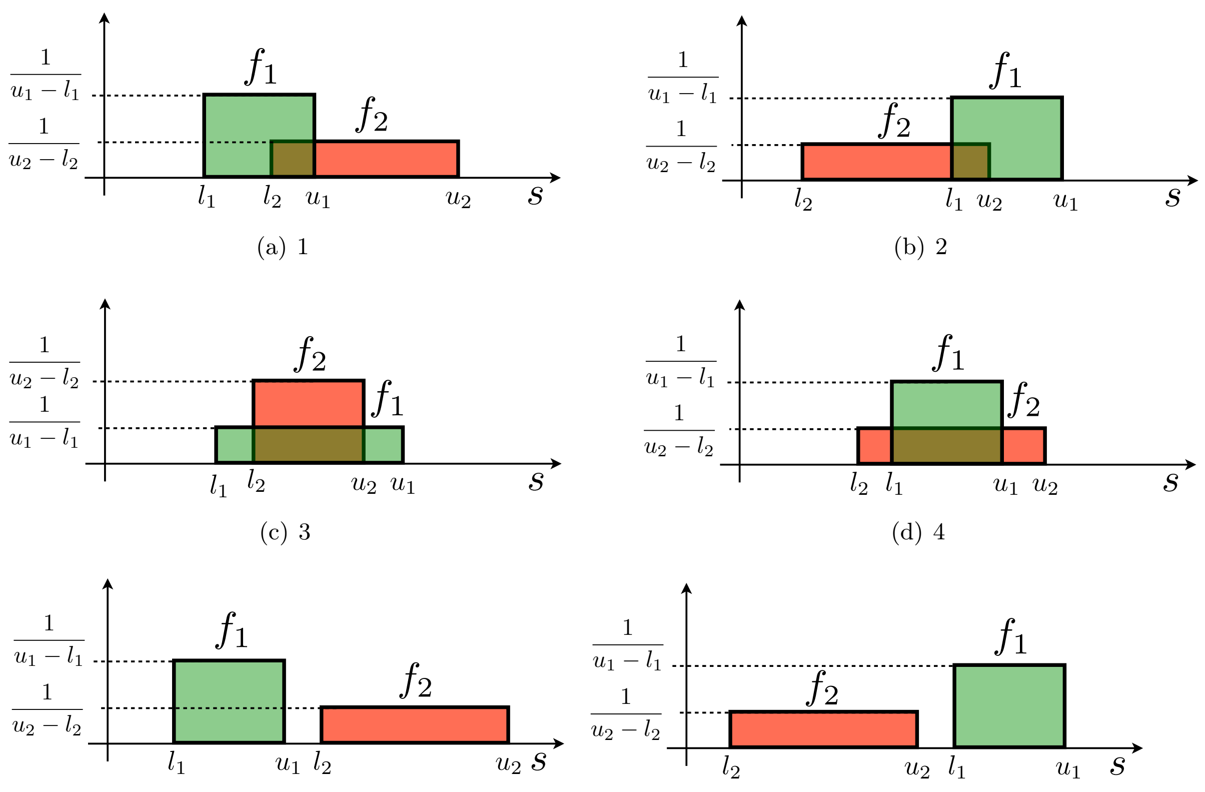
$s(t_1)$ can be greater than $s(t_2)$ only when $s(t_2)$ varies between l_2 and u_1. If $s(t_2)$ is fixed ($s(t_2) \in [l_2, u_1]$), then $\Pr(s(t_1) > s(t_2)|s(t_2)) = (u_1 - s(t_2))\frac{1}{u_1 - l_1}$, i.e., the area of the pdf of $s(t_1)$ between $s(t_2)$ and u_1.

$$\Pr(s(t_1) > s(t_2)) = \int_{l_2}^{u_1} \Pr(s(t_1) > s(t_2)|s(t_2)) \Pr(s(t_2)) ds(t_2) =$$

$$= \int_{l_2}^{u_1} \frac{u_1 - s(t_2)}{u_1 - l_1} \frac{1}{u_2 - l_2} ds(t_2) =$$

$$= \frac{\frac{1}{2}u_1^2 - u_1 l_2 + \frac{1}{2} l_2^2}{(u_1 - l_1)(u_2 - l_2)}$$

[7] We assume the existence of a deterministic rule known to the crowd worker for breaking ties and thus write equivalently $t_i \not\prec t_j$ and $t_j \prec t_i$.



(a) 1 (b) 2 (c) 3 (d) 4 (e) 5 (f) 6

Figure 3.11: Probability distribution context

Context 2 (3.11(b)).

$s(t_1)$ can be greater than $s(t_2)$ when $s(t_2)$ varies between l_1 and u_2. In this case, $\Pr(s(t_1) > s(t_2)|s(t_2)) = \frac{u_1 - s(t_2)}{u_1 - l_1}$. Moreover, $s(t_1)$ is surely greater than $s(t_2)$ when $s(t_2)$ varies between l_2 and l_1 ($\Pr(s(t_1) > s(t_2)|s(t_2)) = 1$).

$$\Pr(s(t_1) > s(t_2)) = \int_{l_2}^{l_1} \Pr(s(t_1) > s(t_2)|s(t_2)) \Pr(s(t_2)) ds(t_2) +$$

$$+ \int_{l_1}^{u_2} \Pr(s(t_1) > s(t_2)|s(t_2)) \Pr(s(t_2)) ds(t_2) =$$

$$= \int_{l_2}^{l_1} \frac{1}{u_2 - l_2} ds(t_2) + \int_{l_1}^{u_2} \frac{u_1 - s(t_2)}{u_1 - l_1} \frac{1}{u_2 - l_2} ds(t_2) =$$

$$= \frac{l_1 - l_2}{u_2 - l_2} + \frac{u_1 u_2 - \frac{1}{2}u_2^2 - u_1 l_1 + \frac{1}{2} l_1^2}{(u_1 - l_1)(u_2 - l_2)} =$$

$$= \frac{-\frac{1}{2}l_1^2 - \frac{1}{2}u_2^2 + l_1 l_2 + u_1 u_2 - u_1 l_2}{(u_1 - l_1)(u_2 - l_2)}$$

Context 3 (3.11(c)).

$s(t_1)$ can be greater than $s(t_2)$ when $s(t_2)$ varies between l_2 and u_2. In this case, $\Pr(s(t_1) > s(t_2)|s(t_2)) = \frac{u_1 - s(t_2)}{u_1 - l_1}$.

$$\Pr(s(t_1) > s(t_2)) = \int_{l_2}^{u_2} \Pr(s(t_1) > s(t_2)|s(t_2)) \Pr(s(t_2)) ds(t_2) =$$

$$= \int_{l_2}^{u_2} \frac{u_1 - s(t_2)}{u_1 - l_1} \frac{1}{u_2 - l_2} ds(t_2) =$$

$$= \frac{u_1 u_2 - \frac{1}{2}u_2^2 - u_1 l_2 + \frac{1}{2}l_2^2}{(u_1 - l_1)(u_2 - l_2)}$$

Context 4 (3.11(d)).
$s(t_1)$ can be greater than $s(t_2)$ when $s(t_2)$ varies between l_1 and u_1. In this case, $\Pr(s(t_1) > s(t_2)|s(t_2)) = \frac{u_1 - s(t_2)}{u_1 - l_1}$. Moreover, $s(t_1)$ is surely greater than $s(t_2)$ when $s(t_2)$ varies between l_2 and l_1.

$$\Pr(s(t_1) > s(t_2)) = \int_{l_2}^{l_1} \Pr(s(t_1) > s(t_2)|s(t_2)) \Pr(s(t_2)) ds(t_2) +$$

$$+ \int_{l_1}^{u_1} \Pr(s(t_1) > s(t_2)|s(t_2)) \Pr(s(t_2)) ds(t_2) =$$

$$= \int_{l_2}^{l_1} \frac{1}{u_2 - l_2} ds(t_2) + \int_{l_1}^{u_1} \frac{u_1 - s(t_2)}{u_1 - l_1} \frac{1}{u_2 - l_2} ds(t_2) =$$

$$= \frac{l_1 - l_2}{u_2 - l_2} + \frac{\frac{1}{2}u_1^2 - u_1 l_1 + \frac{1}{2}l_1^2}{(u_1 - l_1)(u_2 - l_2)}$$

$$= \frac{-\frac{1}{2}l_1^2 - \frac{1}{2}u_2^2 + l_1 l_2 - u_1 l_2}{(u_1 - l_1)(u_2 - l_2)}$$

Context 5 (3.11(e)).
$s(t_1)$ is never greater than $s(t_2)$.

$$\Pr(s(t_1) > s(t_2)) = 0$$

Context 6 (3.11(f)).
$s(t_1)$ is always greater than $s(t_2)$.

$$\Pr(s(t_1) > s(t_2)) = 1$$

Score distribution updated based on a prior knowledge

We now discuss how a prior knowledge on the relative order of two tuples t_i and t_j impacts on the associated score distributions $s(t_i)$ and $s(t_j)$.

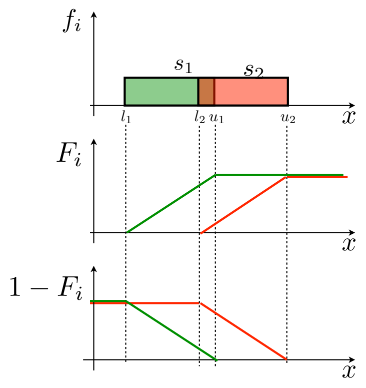
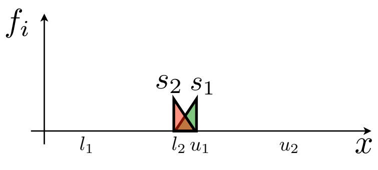
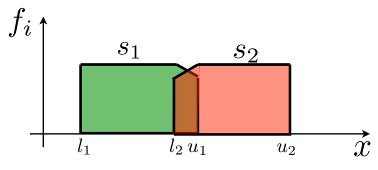


Figure 3.12: Context: pdf's and cdf's of the tuples t_1 and t_2

Consider the tuples t_1 and t_2 in Figure 3.12. Knowing that t_1 comes before or after t_2 modifies our knowledge about the pdf's of the two tuples according to the new information.



(a) $t_1 \prec t_2$



(b) $t_2 \prec t_1$

Figure 3.13: Modified pdf's of the tuples t_1 and t_2

Prior knowledge: $s(t_1) > s(t_2)$

The pdf of t_1 is modified as follows:

$$\begin{aligned}\Pr(s(t_1)|s(t_1) > s(t_2)) &= \frac{\Pr(s(t_1) > s(t_2)|s(t_1))\Pr(s(t_1))}{\Pr(s(t_1) > s(t_2))} = \\ &= \frac{\Pr(s(t_2) < x|s(t_1) = x)\Pr(s(t_1) = x)}{\Pr(s(t_1) > s(t_2))} = \\ &= \frac{F_2(x)f_1(x)}{\Pr(s(t_1) > s(t_2))}\end{aligned}$$

Similarly, the pdf of t_2 is modified as follows:

$$\begin{aligned}\Pr(s(t_2)|s(t_1) > s(t_2)) &= \frac{\Pr(s(t_1) > s(t_2)|s(t_2))\Pr(s(t_2))}{\Pr(s(t_1) > s(t_2))} = \\ &= \frac{\Pr(s(t_1) > x|s(t_2) = x)\Pr(s(t_2) = x)}{\Pr(s(t_1) > s(t_2))} = \\ &= \frac{[1 - F_1(x)]f_2(x)}{\Pr(s(t_1) > s(t_2))}\end{aligned}$$

Prior knowledge: $s(t_1) < s(t_2)$

The pdf of t_1 is modified as follows:

$$\begin{aligned}\Pr(s(t_1)|s(t_1) < s(t_2)) &= \frac{\Pr(s(t_1) < s(t_2)|s(t_1))\Pr(s(t_1))}{\Pr(s(t_1) < s(t_2))} = \\ &= \frac{\Pr(s(t_2) > x|s(t_1) = x)\Pr(s(t_1) = x)}{\Pr(s(t_1) < s(t_2))} = \\ &= \frac{[1 - F_2(x)]f_1(x)}{\Pr(s(t_1) < s(t_2))}\end{aligned}$$

Similarly, the pdf of t_2 is modified as follows:

$$\begin{aligned}\Pr(s(t_2)|s(t_1) < s(t_2)) &= \frac{\Pr(s(t_1) < s(t_2)|s(t_2))\Pr(s(t_2))}{\Pr(s(t_1) < s(t_2))} = \\ &= \frac{\Pr(s(t_1) < x|s(t_2) = x)\Pr(s(t_2) = x)}{\Pr(s(t_1) < s(t_2))} = \\ &= \frac{F_1(x)f_2(x)}{\Pr(s(t_1) < s(t_2))}\end{aligned}$$

Dependency between questions

While asking the second question in order to know whether t_1 comes before t_2 or not, the pdf's of the tuples are again modified:

$$\begin{aligned}
\Pr(s(t_1)|s(t_1) < s(t_2), s(t_2) < s(t_3)) =& \frac{\Pr(s(t_1) < s(t_2), s(t_2) < s(t_3)|s(t_1))\Pr(s(t_1))}{\Pr(s(t_1) < s(t_2), s(t_2) < s(t_3))} = \\
=& \frac{\Pr(s(t_1) < s(t_2), s(t_2) < s(t_3)|s(t_1))\Pr(s(t_1))}{\Pr(s(t_1) < s(t_2)|s(t_2) < s(t_3))\Pr(s(t_2) < s(t_3))} = \\
=& \frac{\Pr(s(t_2) > x, s(t_2) < s(t_3)|s(t_1) = x)\Pr(s(t_1) = x)}{\Pr(s(t_2) > x|s(t_2) < s(t_3))\Pr(s(t_2) < s(t_3))}
\end{aligned}$$

Equivalently:

$$\begin{aligned}
\Pr(s(t_1)|s(t_1) < s(t_2), s(t_2) < s(t_3)) =& \frac{\Pr(s(t_1), s(t_1) < s(t_2)|s(t_2) < s(t_3))\Pr(s(t_2) < s(t_3))}{\Pr(s(t_1) < s(t_2)|s(t_2) < s(t_3))\Pr(s(t_2) < s(t_3))} = \\
=& \frac{\Pr(s(t_1), s(t_1) < s(t_2)|s(t_2) < s(t_3))}{\Pr(s(t_1) < s(t_2)|s(t_2) < s(t_3))}
\end{aligned}$$

3.6.2 Problem definition

We now focus on the problem of Uncertainty Resolution (UR) and use $\mathcal{T}_K$ as a starting point of our investigation, knowing that $\mathcal{T}_K$ can be built with the techniques described in the previous sections.

For convenience, let $ans_\omega(q)$ indicate the answer to question q compatible with an ordering $\omega \in \mathcal{T}$. Also, for a sequence of answers $\mathbf{ans} = \langle ans_1, \ldots, ans_n\rangle$, let $\mathcal{T}_K^{\mathbf{ans}}$ indicate the tree $(\ldots(\mathcal{T}_K^{ans_1})\ldots)^{ans_n}$ obtained by pruning the tree of possible orderings $\mathcal{T}_K$ accordingly.

Formally, an underlying real ordering ω is part of the problem definition, although ω does not need to be known in practice.

Definition 3.6.1. *A UR* problem *is a pair $\langle\mathcal{T}_K, \omega\rangle$, where $\mathcal{T}_K$ is a tree of possible orderings and ω is an ordering of all tuples in $\mathcal{T}_K$, called* real ordering*, such that an ordering $\omega_K \in \mathcal{T}_K$ is a prefix of ω. A* solution *of $\langle\mathcal{T}_K, \omega\rangle$ is a sequence of questions $\langle q_1, \ldots, q_n\rangle$ such that $\mathcal{T}_K^{\langle ans_\omega(q_1),\ldots,ans_\omega(q_n)\rangle}$ only contains ω_K. A solution is* minimal *if no shorter sequence is also a solution.*

We consider two classes of algorithms: *i) offline algorithms*, which determine the solution a priori, before obtaining any answer from the crowd, and *ii) online algorithms*, whereby the solution is determined incrementally as the answers to previous questions arrive. These classes reflect two common situations in crowdsourcing markets: one where the interaction is limited to the publication of a batch of tasks, which is evaluated for acceptance as a whole; and one where the employer can inspect the outcome of crowd work as it becomes available and incrementally publish further tasks[8].

The difference between these two classes lies in the ability of each online algorithm A to probe the real ordering so as to choose at each step a new question, via a function ϕ_A, according to the answers to previously posed questions.

Definition 3.6.2. *A UR algorithm A takes as input a pair $\langle\mathcal{T}_K, B\rangle$, where $\mathcal{T}_K$ is a tree of possible orderings and B (the* budget*) is a nonnegative integer, and outputs a sequence $\mathcal{Q} = \langle q_1, \ldots, q_B\rangle$ of questions on the tuples in $\mathcal{T}_K$. A is* offline *if $\mathcal{Q}$ is a function of $\langle\mathcal{T}_K, B\rangle$. A is* online *if there is a function ϕ_A such that, for any real ordering ω and for $1 \le i \le B$, we have $q_i = \phi_A(\langle\mathcal{T}_K, \langle q_1, \ldots, q_{i-1}\rangle, \langle ans_\omega(q_1), \ldots, ans_\omega(q_{i-1})\rangle\rangle)$.*

An algorithm is optimal if it always finds a minimal solution.

[8]http://docs.aws.amazon.com/AWSMechTurk/latest/AWSMturkAPI/ApiReference_ExternalQuestionArticle.html

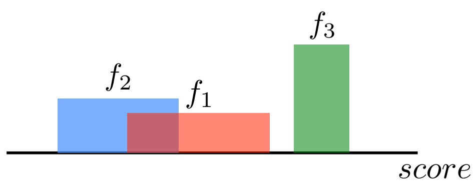


Figure 3.14: Asking the question $t_1 \overset{?}{\prec} t_2$ brings new information on the total order of the tuples, while asking either $t_1 \overset{?}{\prec} t_3$ or $t_2 \overset{?}{\prec} t_3$ does not bring any new information, and thus is useless

Definition 3.6.3. *A UR algorithm A is* optimal *if, for every UR problem $\mathcal{P} = \langle \mathcal{T}_K, \omega \rangle$ and minimal solution $\mathcal{Q}$ of $\mathcal{P}$, the output of A on $\langle \mathcal{T}_K, |\mathcal{Q}| \rangle$ is also a solution of $\mathcal{P}$.*

Theorem 3.6.4. *No deterministic algorithm is optimal.*

Proof. Let $T = \{t_1, t_2, t_3\}$ be such that t_1 dominates t_3, while t_2's pdf overlaps with both t_1's and t_3's. Only three orderings are possible: $\omega_1 = \langle t_1, t_2, t_3 \rangle$, $\omega_2 = \langle t_1, t_3, t_2 \rangle$, and $\omega_3 = \langle t_2, t_1, t_3 \rangle$. The only two questions on overlapping tuples are: $q_1 = t_1 \overset{?}{\prec} t_2$ and $q_2 = t_2 \overset{?}{\prec} t_3$. Each deterministic algorithm must commit to a choice of either q_1 or q_2 as the first question, with no prior knowledge on the real ordering. However, if the real ordering is ω_2, each deterministic algorithm choosing q_1 as the first question fails to identify the correct ordering with just one question, which could have been done by choosing q_2. Analogously, if the real ordering is ω_3, each deterministic algorithm choosing q_2 fails in a similar way (q_1 suffices to identify the real ordering). Therefore, for every deterministic algorithm there is a UR problem whose minimal solution consists of one question while the algorithm's solution includes two questions. □

3.7 Algorithms for UR problem resolution

With the result of Theorem 3.6.4 we cannot hope to find an optimal algorithm. Therefore we now turn our attention to an attainable form of optimality that is of a probabilistic nature, in that it refers to the *expected* amount of uncertainty that remains in the tree of possible orderings after posing the questions selected by an algorithm.

In the following, we firstly define the set of available questions that one could ask to the crowd. Then, we devise a method for selecting the most promising questions to be asked.

3.7.1 Available questions

Let $t_1, \ldots, t_N$ be the set of tuples in the tree of possible orderings, whose scores are defined by means of a probability distribution.

A question $q = t_i \overset{?}{\prec} t_j$ is defined as a comparison between a pair of tuples (see Section 3.6.1 for further details). Since we can build a question out of each pair of tuples in the tree, a naive way of building the question set is the following:

$$\mathcal{Q}_K^{\text{naive}} = \{t_i \overset{?}{\prec} t_j \mid t_i, t_j \in \mathcal{T}_K \wedge i < j\} \tag{3.24}$$

Here, the number of available questions is $|\mathcal{Q}_K^{\text{naive}}| = \binom{N}{2} = \frac{N(N-1)}{2}$.

However, there may be tuples whose score distribution do not overlap. In this case, asking a question that involves a pair of these tuples is not useful, since their relative order is not ambiguous. For instance, consider Figure 3.14. Here, receiving an answer for the question $t_1 \overset{?}{\prec} t_2$ allows us to disambiguate the relative order between t_1 and t_2, while receiving an answer for either $t_1 \overset{?}{\prec} t_3$ or $t_2 \overset{?}{\prec} t_3$ does not help in reducing uncertainty, since it is clear that t_3 has to be ranked higher than t_1 and t_2 (i.e., their relative order is not ambiguous). Asking these useless question would require to involve humans in the process, without gathering any benefit.

Thus, a more convenient approach requires to discard those questions that involve tuples whose score distributions do not overlap.

The set of relevant questions is thus:

$$\mathcal{Q}_K = \{t_i \overset{?}{\prec} t_j \mid t_i, t_j \in \mathcal{T}_K \wedge \texttt{overlap}(f_i, f_j) \wedge i < j\} \tag{3.25}$$

In this case, the number of available questions is $|\mathcal{Q}_K| \leq \binom{N}{2}$.

3.7.2 Approaches for residual uncertainty estimation

We now introduce the notion of residual uncertainty as a means of finding the best sequence of questions for a given tree of possible orderings $\mathcal{T}_K$.

The *residual uncertainty* $\mathcal{R}_{\mathcal{Q}}^{\texttt{appr}}(\mathcal{T}_K)$ that can be expected after a crowd worker has answered the questions $\mathcal{Q}$ can be estimated with three approaches (`appr`): average (`avg`), optimistic (`opt`), and pessimistic (`pess`).

Average approach

With the *average approach*, the expected uncertainty is:

$$\mathcal{R}_{\mathcal{Q}}^{\texttt{avg}}(\mathcal{T}_K) = \sum_{\mathbf{ans}} \Pr(\mathbf{ans}) U(\mathcal{T}_K^{\mathbf{ans}}) \tag{3.26}$$

where $\mathbf{ans} = \langle ans_1, \dots, ans_{|\mathcal{Q}|}\rangle$ ranges over all possible sequences of answers for the questions in $\mathcal{Q}$, $\mathcal{T}_K^{\mathbf{ans}}$ is the pruned tree obtained from $\mathcal{T}_K$ according to the answers **ans**, and $\Pr(\mathbf{ans}) = \Pr(\bigwedge_{j=1}^{|\mathcal{Q}|} ans(q_j) = ans_j)$ is the probability that each answer ans_j in **ans** is the actual worker's answer to question $q_j \in \mathcal{Q}$.

For a single question $\mathcal{Q} = \{t_i \overset{?}{\prec} t_j\}$, we have $\mathcal{R}_{\mathcal{Q}}^{\texttt{avg}}(\mathcal{T}_K) = \Pr(t_i \prec t_j) U(\mathcal{T}_K^{t_i \prec t_j}) + \Pr(t_i \not\prec t_j) U(\mathcal{T}_K^{t_i \not\prec t_j})$.

Optimistic and pessimistic approaches

The *optimistic approach* (resp. *pessimistic approach*) expects the crowd worker to always provide the answers that minimize (resp., maximize) the residual uncertainty:

$$\mathcal{R}_{\mathcal{Q}}^{\texttt{opt}}(\mathcal{T}_K) = \min \bigcup_{\mathbf{ans}} \{U(\mathcal{T}_K^{\mathbf{ans}})\} \tag{3.27}$$

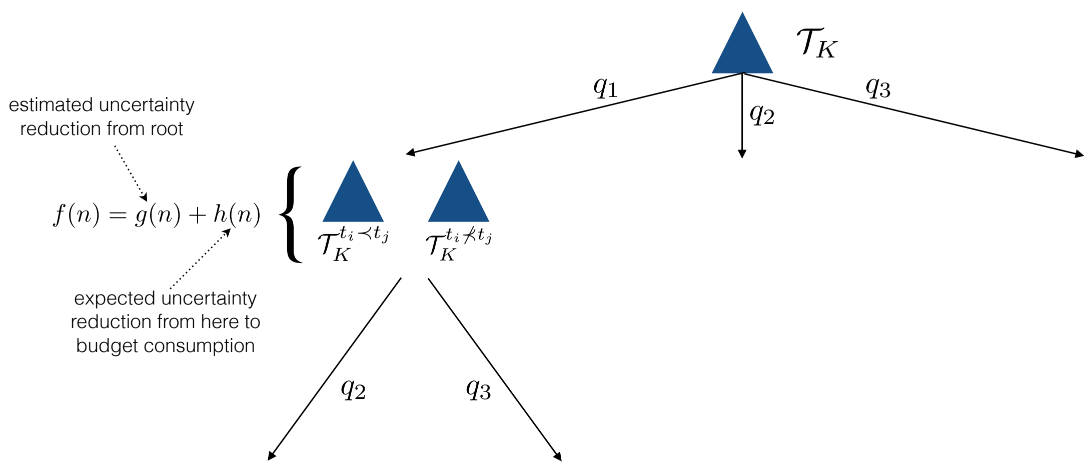
$$\mathcal{R}_{\mathcal{Q}}^{\texttt{pess}}(\mathcal{T}_K) = \max \bigcup_{\mathbf{ans}} \{U(\mathcal{T}_K^{\mathbf{ans}})\} \tag{3.28}$$

3.7.3 Select the best questions: online and offline approaches

In crowdsourcing applications, restrictions in the budget used for rewarding workers or in the available time allotted for collecting answers usually limit the number of questions that can be posted to the crowd. In practical scenarios, such restrictions are yet more significant if the crowd is noisy and thus same question needs to be posed to several workers to increase confidence.

Let B denote the maximum number of questions (budget) that can be asked to the crowd workers. Our goal is then to select the best sequence of questions $\mathcal{Q}^* = \langle q_1^*, \dots, q_B^*\rangle \subseteq \mathcal{Q}_K$ that causes the lowest amount of expected residual uncertainty.

Based on this, we introduce a relaxed version of optimality for offline algorithms, and show how to attain it. Then we discuss sub-optimal, but more efficient algorithms.



$\mathcal{T}_K$
q_1
q_2
q_3
estimated uncertainty reduction from root
$f(n) = g(n) + h(n)$
expected uncertainty reduction from here to budget consumption
$\mathcal{T}_K^{t_i \prec t_j}$
$\mathcal{T}_K^{t_i \not\prec t_j}$
q_2
q_3

Figure 3.15: Solution space of A*

Offline question selection strategies

We first observe that an offline algorithm determines all the questions to pose before obtaining any answer from the crowd. Therefore, permuting the order in which the questions selected by such an algorithm are asked always leads to the same uncertainty reduction on the tree $\mathcal{T}_K$. The order of questions is thus immaterial, and we shall then consider that the output of an offline algorithm is simply a set (instead of a sequence) of questions.

Definition 3.7.1. *An offline UR algorithm is* offline-optimal *wrt. approach* `appr` *if it outputs the set of B questions $\mathcal{Q}^*$ that minimizes the expected residual uncertainty, i.e.:*

$$\mathcal{Q}^* = \arg \min_{\mathcal{Q} \,:\, |\mathcal{Q}|=B \wedge \mathcal{Q} \subseteq \mathcal{Q}_K} \mathcal{R}_{\mathcal{Q}}^{\texttt{appr}}(\mathcal{T}_K). \tag{3.29}$$

In the following, we propose a set of offline algorithms whose aim is to reduce the uncertainty on the space of possible orderings. These algorithms, as demonstrated in the experimental evaluation, have different performance: the more performant is the algorithm, the higher its complexity, the higher is the required computational time. Thus, different options should be used to achieve different objectives: when the degree of uncertainty is too large, a less performant but faster algorithm (e.g., `TB−off`) should be selected, while more performant (and slower) algorithms (e.g., `C−off` and `A*−off`) should be selected when high performance is needed and the degree of uncertainty is limited.

Best-first search offline algorithm (`A*−off`). An implementation of an offline-optimal algorithm can be obtained by adapting the well-known A* best-first search algorithm [Hart et al., 1968] to UR. `A*−off` explores the solution space with the help of a *solution tree* defined as follows:

- each node n is associated with a gain function $f(n)$, which determines the priority of the node in the search process for the optimal solution;
- each edge outgoing from n represents the choice of a question $q \in \mathcal{Q}_K$;
- the maximum tree depth is B;
- for each pair of paths P and R of B questions, $|P \cap R| < B$ (so as to avoid considering the same set of questions multiple times).

An example of solution space is shown in Figure 3.15.

Let n be a node in the solution tree at depth d and let $\mathcal{Q}_n$ denote the sequence of questions on the path between the root and n. The gain function $f(n)$, defined as knowledge plus heuristic, is $f(n) = g(n) + h(n)$. The *knowledge* $g(n)$ is given by the uncertainty reduction expected by asking the questions in $\mathcal{Q}_n$:

$$g(n) = U(\mathcal{T}_K) - \mathcal{R}^{\mathtt{appr}}_{\mathcal{Q}_n}(\mathcal{T}_K). \tag{3.30}$$

Since the contribution to uncertainty reduction of a set of dependent questions is at most the sum of the single contributions (and at most the residual uncertainty), the following *heuristic* $h(n)$ is an upper bound on the uncertainty reduction expected by asking $(B-d)$ more questions:

$$h(n) = \min\{\mathcal{R}^{\mathtt{appr}}_{\mathcal{Q}_n}(\mathcal{T}_K), \sum_{q\in\mathcal{Q}_{\text{best}}} \mathcal{R}^{\mathtt{appr}}_{\mathcal{Q}_n}(\mathcal{T}_K) - \mathcal{R}^{\mathtt{appr}}_{\mathcal{Q}_n\circ\langle q\rangle}(\mathcal{T}_K)\} \tag{3.31}$$

where $\mathcal{Q}_{\text{best}}$ is the set of $(B-d)$ questions (obtained by enumeration) that, if asked after $\mathcal{Q}_n$, guarantee the highest uncertainty reduction. A* traverses the solution tree keeping a sorted priority queue of nodes. The higher the expected gain $f(n)$ achieved by traversing a node, the higher the node priority. Thus, high priority nodes will be explored before others. When the algorithm traverses a path of length B, the questions on that path become the selected set $\mathcal{Q}^*$.

Theorem 3.7.2. `A*−off` *is offline-optimal.*

Offline-optimality follows immediately from the fact that A* is *complete* [Hart et al., 1968], and thus considers all candidate solutions, while retaining only the optimal one. Yet, `A*−off` is computationally very expensive, so we shall also consider two sub-optimal, but more efficient algorithms.

Top-B offline algorithm (`TB−off`). This simpler method computes for each question $q \in \mathcal{Q}_K$ the expected residual uncertainty $\mathcal{R}^{\mathtt{appr}}_q(\mathcal{T}_K)$. Then, we sort the questions in ascending order of $\mathcal{R}^{\mathtt{appr}}_q(\mathcal{T}_K)$ and define $\mathcal{Q}^*$ as the set of top B questions.

Conditional offline algorithm (`C−off`). This method iteratively selects one question at a time based on the previous selections.

Let $\{q_1^*, \dots, q_i^*\}$ be the first i selected questions ($\emptyset$ when $i = 0$). The $(i+1)$-th question q_{i+1}^* is selected by `C−off` from $\mathcal{Q}_K \setminus \{q_1^*, \dots, q_i^*\}$ so as to minimize $\mathcal{R}^{\mathtt{appr}}_{\langle q_1^*,\dots,q_i^*,q_{i+1}^*\rangle}(\mathcal{T}_K)$, i.e., the residual uncertainty conditioned by the choice of the previously selected questions $q_1^*, \dots, q_i^*$.

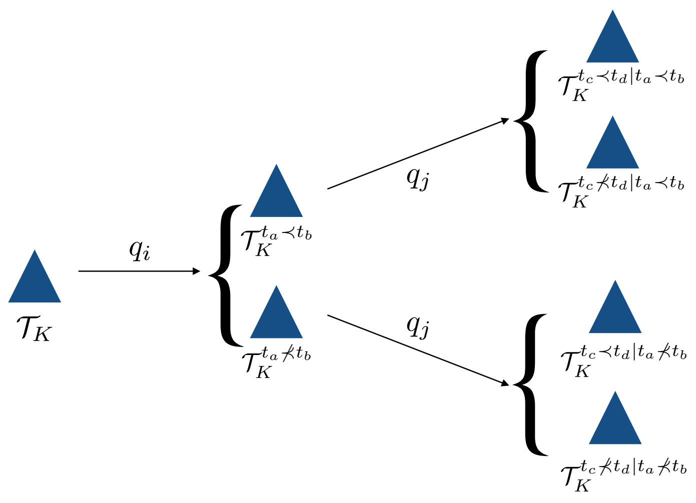
As an example, consider Figure 3.16. Here, we suppose question $q_i = t_a \overset{?}{\prec} t_b$ was already selected. Then, the residual uncertainty of $q_j = t_c \overset{?}{\prec} t_d$ conditioned by the previously selected question q_i is:

$$\mathcal{R}^{\mathtt{appr}}_{\langle q_i,q_j\rangle}(\mathcal{T}_K) = \Pr(t_a \prec t_b)\mathcal{R}^{\mathtt{appr}}_{\langle q_j\rangle}(\mathcal{T}_K^{t_a\prec t_b}) + \Pr(t_a \not\prec t_b)\mathcal{R}^{\mathtt{appr}}_{\langle q_j\rangle}(\mathcal{T}_K^{t_a\not\prec t_b}) \tag{3.32}$$

The final output is thus $\mathcal{Q}^* = \{q_1^*, \dots, q_B^*\}$.

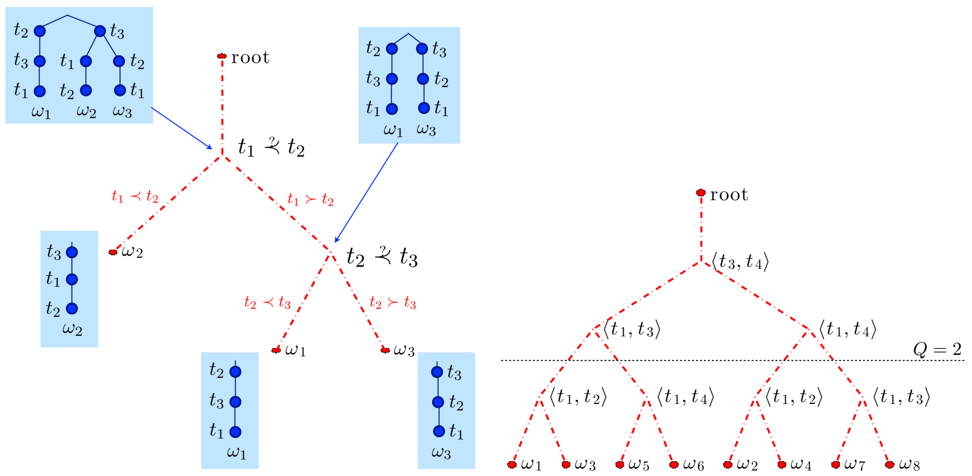
Decision trees (`DT`). In this method we build a decision tree $\theta(\mathcal{T}_K)$ in which each internal node represents a question comparing two tuples in $\mathcal{T}_K$. Each such node has two outgoing branches, corresponding to the two possible answers to the question. Then, each path from the root to a node represents a sequence of questions. Each leaf is associated with the sub-tree of possible orderings that includes only the orderings of $\mathcal{T}_K$ that agree with the answers encountered along the path from the root to the leaf.

The idea of this strategy is to construct the decision tree up to depth B and to choose the leaf associated with the best sequence of up to B questions (and up to $B-1$ corresponding answers). To do this, we shall introduce a measure of goodness of a leaf node.



$\mathcal{T}_K$
q_i
$\mathcal{T}_K^{t_a \prec t_b}$
$\mathcal{T}_K^{t_a \not\prec t_b}$
q_j
q_j
$\mathcal{T}_K^{t_c \prec t_d | t_a \prec t_b}$
$\mathcal{T}_K^{t_c \not\prec t_d | t_a \prec t_b}$
$\mathcal{T}_K^{t_c \prec t_d | t_a \not\prec t_b}$
$\mathcal{T}_K^{t_c \not\prec t_d | t_a \not\prec t_b}$

Figure 3.16: Application of conditional offline algorithm for two questions: q_i and q_j



root
$t_1 \overset{?}{\prec} t_2$
$t_1 \prec t_2$
$t_1 \succ t_2$
ω_2
$t_2 \overset{?}{\prec} t_3$
$t_2 \prec t_3$
$t_2 \succ t_3$
ω_1
ω_3
root
$\langle t_3, t_4 \rangle$
$\langle t_1, t_3 \rangle$
$\langle t_1, t_4 \rangle$
$Q = 2$
$\langle t_1, t_2 \rangle$
$\langle t_1, t_4 \rangle$
$\langle t_1, t_2 \rangle$
$\langle t_1, t_3 \rangle$
ω_1 ω_3 ω_5 ω_6 ω_2 ω_4 ω_7 ω_8

(a) Decision tree for the tree $\mathcal{T}_K$ in Figure 3.4(b)

(b) Decision tree for the tree $\mathcal{T}_K$ in Figure 3.7(a)

Figure 3.17: Example of decision tree for the tree in Figure 3.4(b). Each node represents a question $q \in \mathcal{Q}$, while each leaf represents an ordering ω_i of $\mathcal{T}$

The decision tree is constructed incrementally starting from the root, corresponding to the first selected question q_1^*, as in Algorithm 4. Its two children are then created and associated with the corresponding sub-trees. Then, the construction proceeds iteratively: for each internal node n, we determine the question that minimizes the residual uncertainty of the sub-tree associated with n:

$$q_n^* = \arg \min_{q \in \mathcal{Q} \setminus \mathcal{Q}_n} \mathcal{R}_q^{\texttt{appr}}(\mathcal{T}_n), \tag{3.33}$$

where $\mathcal{Q}_n$ represents the sequence of questions corresponding to the path from the root to node n, and $\mathcal{T}_n$ denotes the sub-tree of possible orderings associated with node n, which depends on both $\mathcal{Q}_n$ and the corresponding answers. No children are generated if n is at depth B or if $\mathcal{T}_n$ identifies a unique ordering. Note that Algorithm 4 leads to a sequence of questions that corresponds to a specific path of $\theta(\mathcal{T})$.

In order to illustrate the procedure, the tree $\mathcal{T}$ in Figure 3.7(a) and the corresponding decision tree $\theta(\mathcal{T})$ in Figure 3.17(b) will be used as examples, with a budget of $B = 2$ questions. Let $\theta_B(\mathcal{T})$ denote the first B levels of $\theta(\mathcal{T})$. The two leaf nodes in $\theta_B(\mathcal{T})$ correspond to the

Algorithm 4: Top-1 online algorithm (`T1−on`)

```
Input: Tree of possible orderings $\mathcal{T}_K$, Budget $B$
Output: Optimal sequence of questions $\mathcal{Q}^*$
Environment: Underlying real ordering $\omega$
Parameters: Question selection approach appr among avg, opt, pess
1. $\mathcal{Q}^* := \emptyset$;
2. for $i := 1$ to $B$
3.     if $|\mathcal{T}_K| = 1$ then break;
4.     $q_i^* := \arg\min_{q \in \mathcal{Q}_K \setminus \mathcal{Q}^*} \mathcal{R}^{\mathtt{appr}}_{\langle q \rangle}(\mathcal{T}_K)$; // see Equations (3.26)–(3.28)
5.     $\mathcal{Q}^* := \mathcal{Q}^* \circ \langle q_i^* \rangle$; // appending the selected question
6.     Ask $q_i^*$ to the crowd and collect the answer $ans_\omega(q_i^*)$
7.     $\mathcal{T}_K := \mathcal{T}_K^{ans_\omega(q_i^*)}$; // updating the tree
8. return $\mathcal{Q}^*$;
```

sequences of questions $\mathcal{P}_1 = \langle t_3 \overset{?}{\prec} t_4, t_1 \overset{?}{\prec} t_3 \rangle$ and $\mathcal{P}_2 = \langle t_3 \overset{?}{\prec} t_4, t_1 \overset{?}{\prec} t_4 \rangle$.

The goal is to select the leaf corresponding to the optimal sequence of questions $\mathcal{Q}^*$ (i.e., either $\mathcal{P}_1$ or $\mathcal{P}_2$). We assign a goodness measure $\mathcal{G}(\mathcal{P}_n)$ to each leaf node n, based on two criteria: *i)* the uncertainty reduction gained by asking the questions in $\mathcal{P}_n$ (and receiving the answers indicated in $\mathcal{P}_n$'s path); *ii)* the probability of observing $\mathcal{P}_n$'s path.

We define the residual uncertainty $\mathcal{R}^{\mathtt{appr}}(\mathcal{P}_n)$ of a leaf node n as $\mathcal{R}^{\mathtt{appr}}_{q_n^*}(\mathcal{T}_n)$, i.e., the value of the objective function in (3.33). Indeed, this is the expected uncertainty obtained by asking q_n^* given the tree of possible orderings $\mathcal{T}_n$ resulting from the answers to (up to) $B-1$ questions. For instance, the residual uncertainty $\mathcal{R}^{\mathtt{appr}}(\mathcal{P}_1)$ in Figure 3.17(b) can be computed by considering the expected residual uncertainty obtained by asking $t_3 \overset{?}{\prec} t_4$ (and receiving $t_3 \prec t_4$ as an answer), followed by $t_1 \overset{?}{\prec} t_3$.

The probability $\Pr(\mathcal{P}_n)$ associated with a leaf node n can be computed by considering the $B-1$ answers corresponding to the path from the root to n and summing up the probabilities of the orderings in $\mathcal{T}$ that are compatible with these answers. For example, $\Pr(\mathcal{P}_1) = \Pr(\omega_1) + \Pr(\omega_3) + \Pr(\omega_5) + \Pr(\omega_6)$.

The goodness measure $\mathcal{G}(\mathcal{P}_n)$ we adopt is computed as follows:

$$\mathcal{G}(\mathcal{P}_n) = \Pr(\mathcal{P}_n)(U(\mathcal{T}) - \mathcal{R}(\mathcal{P}_n)). \tag{3.34}$$

The best sequence of questions $\mathcal{Q}^*$ selected by this approach are therefore computed as follows:

$$\mathcal{Q}^* = \arg\max_{n \in \Lambda} \mathcal{G}(\mathcal{P}_n), \tag{3.35}$$

where Λ is the set of leaf nodes of $\theta_B(\mathcal{T})$.

Online question selection strategies

An online algorithm has the ability to determine the i-th question based on the answers collected for all the previously asked $i-1$ questions. Differently from the offline case, the output of an online algorithm is treated as a sequence and not as a set, since each received answer may influence the choice of the next question, and thus the order matters.

In the following, we propose two different online algorithms, which have different performance (as demonstrated in the experimental evaluation) and different complexities. Again, one should select the most appropriate algorithm according to the degree of uncertainty and the needed performance: `A*−on` is most suited for situation with limited uncertainty, achieving high performance, while `T1−on` should be selected in case of high uncertainty.

Best-first search online algorithm ($\texttt{A}^*\!-\!\texttt{on}$). An online UR algorithm can be obtained by iteratively applying $\texttt{A}^*\!-\!\texttt{off}$ B times. At the i-th step, $1 \leq i \leq B$, $\texttt{A}^*\!-\!\texttt{on}$ identifies the i-th question q_i^* in its output and asks it to the worker, thus obtaining the answer $ans_\omega(q_i^*)$. Question q_i^* is simply the first element of the sequence $\mathcal{Q}_i^*$ returned by $\texttt{A}^*\!-\!\texttt{off}$ for the tree of possible orderings $\mathcal{T}_K^{\langle ans_\omega(q_1^*),\ldots,ans_\omega(q_{i-1}^*)\rangle}$ with budget $(B-i+1)$, where ω is the real ordering and $q_1^*,\ldots,q_{i-1}^*$ are the previously selected questions (initially, $\texttt{A}^*\!-\!\texttt{off}$ is applied on $\mathcal{T}_K$ with budget B and the first question in its output is chosen as q_1^*). Intuitively, each step chooses the most promising question q_i^* within the horizon of the remaining $B-i+1$ questions to be asked based on the current knowledge of ω. Note that, as new answers arrive, the next most promising questions might no longer coincide with the rest of the previously planned sequence $\mathcal{Q}_i^*$.

Being based on $\texttt{A}^*\!-\!\texttt{off}$, $\texttt{A}^*\!-\!\texttt{on}$ is costly. Thus, we also consider a simpler but more efficient online algorithm.

Top-1 online algorithm ($\texttt{T1}\!-\!\texttt{on}$). Algorithm 4 illustrates the $\texttt{T1}\!-\!\texttt{on}$ algorithm, which builds the sequence of questions $\mathcal{Q}^*$ iteratively until the budget B is exhausted (line 2). At each iteration, the algorithm selects the best (Top-1) unasked question, i.e., the one that minimizes the expected residual uncertainty with budget $B = 1$ (line 4). The selected question q_i^* is then appended to $\mathcal{Q}^*$ and asked to the crowd. Depending on the answer, the tree of possible orderings $\mathcal{T}_K$ is updated to the sub-tree that agrees with the answer to q_i^* (line 7). Early termination may occur if all uncertainty is removed, i.e., the tree is left with a single path (line 3).

3.7.4 Strategies for quick question selection

Quantifying the uncertainty of a tree of possible orderings with the proposed techniques requires to build completely the tree, i.e., firstly building the structure and then filling it with the probabilities $\Pr(\omega)$ associated with its leaves (i.e., orderings). However, computing the whole tree may be costly, in particular when a large number of overlaps exists (i.e., large number of orderings in the tree).

In the following, we propose some strategies for quick tree construction and question selection, so as to speed up the proposed procedures.

Approximate probabilities via Monte Carlo sampling

One simple way to avoid this cost is to approximate $\Pr(\omega)$ by sampling, as suggested in Section 3.3.1. Given the tree of possible orderings structure, we can sample the tuple scores and count the frequency of occurrence of each leaf of the tree of possible orderings:

$$\Pr(\omega) = \frac{\text{occurrences of } \omega}{\#\text{ samples}} \tag{3.36}$$

Obviously, the higher the number of samples used, the better the approximation.

Incremental construction of the tree of possible orderings

The number of orderings in a tree of possible orderings can be large if there are many overlaps in the tuple score distributions, thereby affecting the execution time of our algorithms.

We propose a new algorithm, called *Incremental*, that does not start with a tree of possible orderings $\mathcal{T}_K$, but rather builds it incrementally, one level at a time, alternating a tree construction step with a cycle of question answering and tree pruning.

Online selection strategy. In the online version of out Incremental algorithm ($\texttt{Incr} - \texttt{On}$, shown in Algorithm 5), each tree of possible orderings $\mathcal{T}_k$, $1 \leq k \leq K$, is built by adding one

Algorithm 5: Incremental algorithm (`Incr`)

```
Input: Tuple set $T$, Budget $B$, Depth $K$
Output: Optimal sequence of questions $\mathcal{Q}^*$
Environment: Underlying real ordering $\omega$
Parameters: Question selection approach appr among avg, opt, pess
 1. $\mathcal{Q}^* := \emptyset$; ans $:= \emptyset$; $\mathcal{T}_0 :=$ tree of possible orderings with only the root node;
 2. for $k := 1$ to $K$
 3.     $\mathcal{T}_k :=$ tree of possible orderings obtained by extending $\mathcal{T}_{k-1}$ by one level;
 4.     $\mathcal{Q}_k :=$ relevant questions for $\mathcal{T}_k$ (as in Equation (3.24));
 5.     while $|\mathbf{ans}| < B$ and $|\mathcal{T}_k| > 1$
 6.         $q^* := \arg\min_{q \in \mathcal{Q}_k \setminus \mathcal{Q}^*} \mathcal{R}^{\mathtt{appr}}_{\langle q \rangle}(\mathcal{T}_k)$; // as in Equations (3.26)–(3.28)
 7.         $\mathcal{Q}^* := \mathcal{Q}^* \circ \langle q^* \rangle$; // appending the selected question
 8.         ans $:=$ ans $\circ \langle ans_\omega(q^*) \rangle$ // appending the crowd's answer
 9.         $\mathcal{T}_k := \mathcal{T}_k^{\mathbf{ans}}$; // updating the tree of possible orderings
10. return $\mathcal{Q}^*$;
```

level to $\mathcal{T}_{k-1}$ (line 3): in each ordering ω_{k-1} of $\mathcal{T}_{k-1}$, we add the unused sources as children of the leaf (we assume, for efficiency, that the probabilities are approximated by sampling, as mentioned in Section 3.3.1).

Then, we ask as many questions as possible until either the budget B is consumed or no uncertainty is left in $\mathcal{T}_k$ (line 5). We thus keep the tree of possible orderings as pruned as possible, and only proceed to computing the next level when the uncertainty in the previous levels is removed. Each time, we select the question that minimizes the residual uncertainty of $\mathcal{T}_k$ and immediately pose it to the crowd in an online fashion (lines 6-9).

As a result, since questions are asked in the top levels of the tree at first, many orderings are pruned from the tree at a time.

An example is shown in Figure 3.18. Figure 3.18(a) shows the complete tree of possible orderings. The real ordering ω is marked with a bold line. The incremental algorithm starts to build the first level of the tree (Figure 3.18(b)): here, two orderings are defined: $\omega_1 = t_1$ and $\omega_2 = t_2$, respectively with probabilities $\Pr(\omega_1) = \Pr(t_1 \prec t_2)$ and $\Pr(\omega_2) = \Pr(t_1 \not\prec t_2)$. Then, the question $t_1 \overset{?}{\prec} t_2$ is asked and the answer is used to delete the ordering ω_2. After that, the second level of the tree is built (Figure 3.18(c)), but since the tree contains only one path up to now (i.e., $\omega_1 = t_1 \prec t_2$), no questions are asked to the crowd. Next, another level of the tree is built (Figure 3.18(d)): here, two orderings are present ($\omega_1 = t_1 \prec t_2 \prec t_3$ and $\omega_2 = t_1 \prec t_2 \prec t_4$), and the question $t_3 \overset{?}{\prec} t_4$ is asked to discard the ordering ω_1. Finally, the fourth level of the tree is built (Figure 3.18(e)), and since the tree contains just one ordering, the procedure terminates and the real ordering $\omega = t_1 \prec t_2 \prec t_4 \prec t_3$ is returned.

Hybrid question selection strategy The hybrid version of the Incremental algorithm (called $\mathtt{Incr} - \mathtt{Hyb}$) is a generalization of $\mathtt{Incr} - \mathtt{On}$, where $1 \leq n \leq B$ questions are asked during each cycle of question answering and tree pruning. After each construction step, we select the n questions whose expected uncertainty reductions are higher. However, since the space of possible orderings is not fully materialized, it may happen that the number of available questions is smaller than n. In this case, we keep constructing new levels of $\mathcal{T}_k$ until at least n questions are available.

Estimation of the effectiveness of questions

We can avoid altogether the estimation of the probabilities $\Pr(\omega)$ by assessing the effectiveness of questions based solely on *i)* the overlap between score pdf's, and *ii)* the topology of the tree. While the former provides a rough indication of the amount of uncertainty that will be removed from the tree of possible orderings when the related question is answered (the larger the overlap, the larger the uncertainty reduction), the latter may help selecting questions that involve pairs

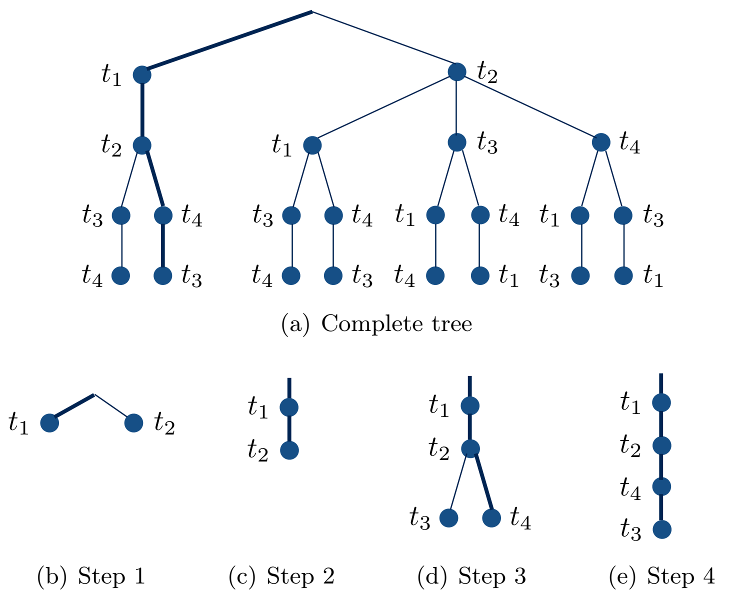


Figure 3.18: Incremental algorithm. Complete tree of possible orderings (a), where the real ordering is marked with a bold line, and steps required to find the real ordering

of tuples appearing in the top levels of the tree.

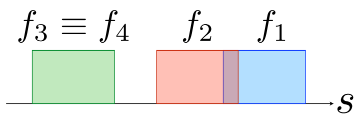


Figure 3.19: Score distributions for the objects t_1, t_2, t_3 and t_4

To illustrate, consider Figure 3.19, where four tuples t_1, t_2, t_3, t_4 are such that the only overlaps of their pdf's are between f_1 and f_2 (top 2 tuples, partial overlap) and f_3 and f_4 (bottom 2 tuples, total overlap). Only two questions are relevant: $q_1 = t_1 \overset{?}{\prec} t_2$ and $q_2 = t_3 \overset{?}{\prec} t_4$. Answering the question regarding the largest overlap (q_2, i.e., the most uncertain using the metrics we discussed earlier) identifies the top-3 tuple. However, answering q_1 identifies the top-1 tuple and thus is likely to be more relevant in a top-K context. This suggests that the overlap alone (which is used in [Zhang et al., 2013]) is not sufficient, and the topology of the tree of possible orderings should also be taken into account.

Let $\mu(t)$ indicate the median rank of tuple t in the orderings of $\mathcal{T}_K$, where t's rank is set to $K+1$ if t is not in the ordering.

We define the *effectiveness* of $q = t_i \overset{?}{\prec} t_j$ as:

$$\tilde{\mathcal{R}}^{\epsilon}_q(\mathcal{T}_K) = \frac{\mu(t_i) + \mu(t_j)}{2} \cdot (1 - H(\Pr(t_i \prec t_j))), \tag{3.37}$$

where $H(\Pr(t_i \prec t_j)) = -\Pr(t_i \prec t_j)\log_2(\Pr(t_i \prec t_j)) - \Pr(t_i \not\prec t_j)\log_2(\Pr(t_i \not\prec t_j))$.

Then, $\tilde{\mathcal{R}}^{\epsilon}_q(\mathcal{T}_K)$ can be used as a proxy for residual uncertainty that does not require the computation of the probabilities of $\mathcal{T}_K$'s leaves. The UR algorithms shown in Section 3.7.3 can thus be reformulated by using $\tilde{\mathcal{R}}^{\epsilon}_q$ rather than $\mathcal{R}^{\texttt{appr}}_q$.

3.8 Handle noisy workers

In a crowdsourcing scenario, the collected answers might be noisy.

Let p denote a crowd worker's accuracy (i.e., the probability that his/her answer is correct). When $p < 1$, it is possible that the answer ans_q given by the user for the question $q = t_i \prec t_j$ is not correct. Thus, when the techniques presented in the previous sections are applied, pruning those paths that are not consistent with ans_q could cause the deletion of paths that are in fact reporting the correct relative order of t_i and t_j.

Consequently, handling noisy workers requires a simple redefinition of the trees $\mathcal{T}_K^{t_i \prec t_j}$, $\mathcal{T}_K^{t_i \not\prec t_j}$. Here, we will not delete paths from the trees: both trees will represent the whole set of possible orderings included in $\mathcal{T}_K$, but the probabilities of the orderings need to be adjusted to reflect the incoming information.

The tree $\mathcal{T}_K^{\mathbf{ans}}$ obtained collecting all answers is defined as before, i.e., $(\dots(\mathcal{T}_K^{ans_1})\dots)^{ans_n}$, in which the probabilities of the orderings are recomputed for each answer, without performing pruning.

Let $\Pr(\omega)$ and $\Pr(\omega | ans_q = t_i \prec t_j)$ denote, respectively, the probability of the same ordering ω in $\mathcal{T}_K$ and $\mathcal{T}_K^{t_i \prec t_j}$. Then, by Bayes' theorem [Zhang et al., 2013]

$$\begin{aligned}\Pr(\omega | ans_q = t_i \prec t_j) &= \frac{\Pr(ans_q = t_i \prec t_j | \omega)\Pr(\omega)}{\Pr(ans_q = t_i \prec t_j)} \\ &= \frac{\Pr(ans_q = t_i \prec t_j | \omega)\Pr(\omega)}{p\Pr(t_i \prec t_j) + (1-p)\Pr(t_i \not\prec t_j)},\end{aligned} \tag{3.38}$$

where $\Pr(ans_q = t_i \prec t_j | \omega) = p$, if $t_i \prec t_j$ in ω; otherwise, $1 - p$.

A similar expression can be written for $\mathcal{T}^{t_i \not\prec t_j}$:

$$\begin{aligned}\Pr(\omega | ans_q = t_i \not\prec t_j) &= \frac{\Pr(ans_q = t_i \not\prec t_j | \omega)\Pr(\omega)}{\Pr(ans_q = t_i \not\prec t_j)} \\ &= \frac{\Pr(ans_q = t_i \not\prec t_j | \omega)\Pr(\omega)}{p\Pr(t_i \not\prec t_j) + (1-p)\Pr(t_i \prec t_j)},\end{aligned} \tag{3.39}$$

3.9 Summary

In this Chapter we proposed a methodology for the application of active crowdsourcing techniques so as to reduce uncertainty in a corpus of data, with specific focus on the top-K query context. We started the some state-of-the-art works that suggest how to materialize the space of possible orderings. We suggested how to adapt these techniques to the top-K context and how to model uncertainty in the resulting space. Finally, we proposed several techniques which can be used to maximize the expected uncertainty reduction on the space of possible orderings.

The proposed work answers to the following research questions:

- **Research question 1:** ***How can uncertainty of structured data be modeled?*** In this Chapter we proposed four uncertainty metrics which can be used to quantify uncertainty in a corpus of data. Specifically, we noticed that while some metrics in the state of the art (e.g., entropy) are based only on the probabilities of each ordering, without capturing the structure of the space of possible orderings, other measures can involve the tree structure in the computation. In this way, these metrics adhere more precisely to the problem structure, being able to judge differently trees having similar probabilities but different structure.

- **Research question 2:** ***How do crowd task answers impact on data uncertainty?*** In this Chapter we studied how an answer from a crowd worker can affect the degree of uncertainty in the space of possible orderings: each answer provides new information, which could not be extracted from data, and thus it can reduce uncertainty by removing some orderings from the space (in case of answers from experts) or modifying their probabilities (in case of answers from noisy workers).

- **Research question 3: *How do task selection and budget constraint affect uncertainty on structured data?*** In this Chapter we demonstrated that some tasks produce a larger expected uncertainty reduction, while others affect poorly uncertainty. Thus, it is possible to estimate the uncertainty reduction brought by each available task, and select the most promising one, so as to maximize the expected uncertainty reduction. Moreover, we proposed several task selection techniques which can be used when a budget constraint is provided. Some of them require the full materialization of the tree of possible orderings, with high performance but high costs, while others build incrementally the tree, with more limited performance but low costs. These techniques can be used both in online mode (i.e., by asking one question at a time) and offline mode (i.e., by selecting a set of questions to be asked in batch).

- **Research question 4: *How does worker quality impact on crowd tasks effectiveness?*** In this Chapter we proposed a technique to handle noisy answers, so that orderings are not blindly deleted from the tree if the workers suggest it: their probabilities are simply remodeled so as to take into account the possibility that the provided answers are wrong. Obviously wrong answers introduce noise, but with the proposed techniques we are able to limit its negative effect.

Chapter 4

Optimizing passive crowdsourcing: Influencers retrieval on multimodal dynamic data

Every day, a massive amount of content is produced by users in social media. These data are publicly available, and contain information about behaviors, interests and activities of the social media population. Passive crowdsourcing is the process of analyzing user-generated content and convert it into a comprehensible structure for further use. When a user expresses a need, in order to gather information that could answer to that specific need, one could decide to navigate through the massive amount of user-generated content and retrieve relevant and fresh data from it. However, when retrieving information about a specific topic, due to the high diversity of people posting every day, user-generated content could be redundant, not representative, not well presented and related to different topics. Thus, a way of filtering it is needed, so as to discard irrelevant content and focus on pertinent data.

In this Chapter, we introduce a first use case for the passive crowdsourcing application. Here, we filter content that is published on Twitter so as to retrieve influencers and relevant content about a specified topic.

4.1 The scenario: Twitter influencers

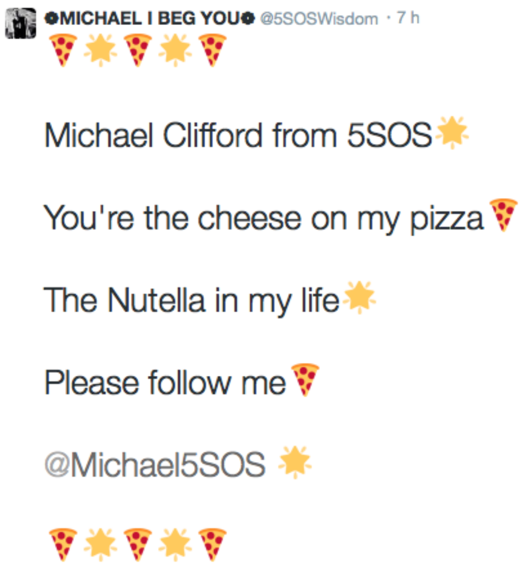
A preliminary definition of social influence can be found in [Katz and Lazarsfeld, 1970]. According to this work, a minority of people, called *opinion leaders*, act as intermediaries between the society and the mass media. An opinion leader is a subject which is very informed about a topic, well-connected with other people in the society and well-respected. Since information is passing through the opinion leaders before going to the society, this communication model is called *two-step communication model*, where in the first step information goes from mass media to opinion leaders, and in the last step reaches the society through the opinion leaders. After this analysis, the expression *influential users* (or *influencers*) [Merton, 1957] started to spread through different fields, such as innovation [Rogers Everett, 1995], communication research [Weimann, 1994] and marketing [Chan and Misra, 1990]. The power of this theory is behind the ability of influencers of spreading rapidly and easily information: a viral campaign can reach a large audience at a small cost (needed to identify influencers and convince them to diffuse the information).

However, more modern works suggested that influencers alone are nor necessary not sufficient to diffuse information through the society [Watts and Dodds, 2007]. In fact, influencers may be the ones that generate large flows of information to be spread through the network, but then average users are the one that diffuse it. Thus, influence is a key factor in the generation of a convincing information, but then the network is the last needed ingredient to make the information survive through the network.

More recently, computer scientist began creating theories about influence in social networks. The first works that considered the propagation of influence in a social network are [Domingos and Richardson, 2001, Richardson and Domingos, 2002]. In this works, the authors build a probabilistic model in which users are assumed to be influenced by their nearest users in the social network. Thus, a heuristic is proposed to identify influencers. Then, the works [Kempe et al., 2003, Chen et al., 2010, Leskovec et al., 2007] define and solve the influence maximization problem. This problem is defined as follows: given a social network, select a set of K users such that they influence the largest number of users in the network.

In recent years Twitter has gained a huge visibility as a platform on which users share ideas and opinions. Thus, several works focused their solutions for the influence maximization problem on this social network. These works focus their analysis on the graph structure and the textual content of each post, to state whether a person is an influencer for a specific topic. Typical metrics that are used to evaluate the influence of a user are her number of followers and friends, her number of posts related to the topic, the number of retweets, etc.

However, in the state of the art there is a lack of works that consider the whole published content, which comprises not only textual content, but also multimedia content and linked content. In fact, often the text of a post is ambiguous and hardly attributable to a topic, while the associated extra content (i.e., URLs and multimedia files) can be useful to disambiguate. This often causes wrong classification of the content, and thus wrong count of the topic-related posts for an author. Thus, more refined techniques are needed to get a rich view of the content and disambiguate the content topic.


●MICHAEL I BEG YOU● @5SOSWisdom · 7 h
🍕🌟🍕🌟🍕

Michael Clifford from 5SOS🌟

You're the cheese on my pizza🍕

The Nutella in my life🌟

Please follow me🍕

@Michael5SOS 🌟

🍕🌟🍕🌟🍕

Figure 4.1: False positive captured by a text classifier when the topic is `cooking`

Moreover, several works are focusing their analysis on data sets collected in fixed time spans. However, influencers change over time [Cha et al., 2010], and thus a real-time analysis of the social network posts and users is needed to capture the most updated influencers list.

In this Chapter, we introduce our solution for the influence maximization problem, with focus on real-time data and multi-modal analysis of content. In Section 4.3 we introduce the architecture of the topic-related content retrieval pipeline. Then, Section 4.4 and Section 4.6 give details about the crawling process and the multi-model classification process, respectively. Section 4.7 illustrates how to keep updated the crawling criteria, so that the retrieved content always reflect the active communications on Twitter. Section 4.8 presents the influence metrics used in this work. Finally, Section 4.9 discusses how to present the results and evaluate their quality.

4.2 Open problems in influencer detection

The concept of trust has been studied in sociology [Molm et al., 2000], psychology [Cook et al., 2005], economics [Huang, 2007] and computer science [Maheswaran et al., 2007]. Moreover, during the last years many companies have emerged in the market having as core business to analyze social media and extract various statistics about users and content, e.g., Blogmeter[1], Klout[2], PeerIndex[3] and ProSkore[4]. These statistics include online reputation, influence evaluation, engagement and content relevance. Usually, the offered services are grouped in software suites that allow one to analyze dynamically content (e.g., by filtering it by topic and relative subtopics, or by extracting the most diffused terms and concepts for a specific topic) and users (e.g., by visualizing who are the most active users in the field).

The most common strategies for influence evaluation in social media are: *i)* either to identify influencers, or *ii)* to study the maximization of influence's spread in a social network. In [Kiss and Bichler, 2008] the identification of influencers is achieved by considering the structural properties of networks. In [Lu et al., 2012] a graph-based framework is used to predict the evolution of influencers. [Scripps et al., 2009] investigated how different decisions such as selection and influence affect the dynamics of social networks. [Gomez Rodriguez et al., 2010] developed a method to trace paths of diffusion and influence through networks. Furthermore, some researchers investigated the problem of maximizing influence on a person network (ego-net) for applications such as viral marketing [Domingos and Richardson, 2001, Kempe et al., 2003, Goyal et al., 2010]. In [Tan et al., 2010], authors studied how to track and predict users' action according to a learning model. However, these works neither consider heterogeneous information nor learn topics and influence strength jointly: there are no works in the state of the art that analyze the full spectrum of multimedia content produced and consumed by users to estimate a local and contextual notion of trust. Moreover, they did not consider the topic-level influence: in most of these works, a user is influencer if she is interesting for a large part of users, independently from the topic of interest one is tracking. However, this falls on the million follower fallacy, where a user is influential if she is a celebrity.

In the state of the art, patterns of temporal variation of popularity have been investigated too, mostly focusing on the attention received by pieces of content. Previous works include for instance the study of video popularity saturation on YouTube in relation to content visibility [Figueiredo et al., 2011] and the classification of bursty Twitter hashtags in relation to the volume of related tweets before and after the peak [Lehmann et al., 2012]. Time series have been used to predict popularity in blogs, where the reaction time of the crowd is strongly correlated to the expected overall popularity [Lerman and Hogg, 2010]. However, a few works focus on the mining of temporal patterns in content diffusion and people activity on social media, which could help in tracing the dynamics of influence.

Another aspect regards multimedia search. Multimedia search before the social media era aimed at answering multimedia queries (using e.g., query by example approach) on static databases. Social media change the scene by generating and sharing huge volumes of ephemeral content. The volumes of image/video shared through the social media every day and the ephemeral nature of the postings require new indexing and searching methodologies. It is thus necessary to design and develop a NoSQL-based time-aware multimedia search service that is

[1] `http://www.blogmeter.eu`
[2] `https://klout.com/home`
[3] `http://www.peerindex.com`
[4] `http://www.proskore.com`

able to answer complex time related and multimedia queries, supporting large but ephemeral multimedia content processing.

Syria's socially mediated civil war

A real scenario where the recognition of influential users and relevant content is difficult to be performed can be found in the study of the socially mediated civil war in Syria [Lynch et al., 2014]. An exceptional amount of what the outside world knows about Syria's nearly three-year-old conflict has come from videos, analysis and commentary circulated through social networks. In the state of the art, several remarkable number of creative and important efforts to exploit the vast quantities of information about Syria can be found. Journalists use videos and social media accounts to report on an exceptionally difficult and dangerous conflict, while analysts inside and outside governments have also used online information to paint detailed accounts of the factions of the Syrian insurgency.

However, English-language conversation about Syria is particularly insular and interacts only with itself, creating a badly skewed impression of the broader arabic discourse. Arabic-language conversation, on the other hand, quickly came to dominate the online discourse, requiring dozens of experts to analyze manually the produced content to extrapolate interesting and non-polarized information. Thus, it is becoming more and more difficult to guarantee a balanced analysis between different sources. More specifically, the deluge of information made it difficult for even specialists to evaluate the credibility and significance of videos, images or information circulating online.

Some online hubs (i.e., famous users in this scenario) have been selected as curators for the online content, sorting through, interpreting and synthesizing the online materials. An interesting insight is that a large number of individual hubs that are little known to the wider public have massive influence within discrete communities, while large audiences tend instead to rely on a relatively small number of such individuals. Such hubs played an important role in newspapers and television, although no assumption could be made about the independence or neutrality of these hubs: many (if not most) hubs were activists on one side or the other.

The primary use of Internet by researchers has been to search online sources to extract otherwise unknown information about the conflict. These efforts range from the very simple (i.e., watching YouTube videos for evidence of jihadist involvement or foreign weapons) to the more complex (i.e., estimating deaths and casualties). Several research teams are thus created so as to contain a mixture of competences: researchers with Arabic-language skills, developers of automatic text analyzers, analysts that watch videos to look for specific content (e.g., new types of weaponry). Countless hours are thus spent by people to manually validate available online content and spot the most relevant published videos, images and texts. Moreover, these studies depend heavily on the validity of the underlying data. Several of the datasets employed are produced by organizations that belong to the Syrian opposition.

Nevertheless, requiring humans to process manually the data to assess its quality is dangerous and costly. First of all, humans are not able to process a large corpus of data, which usually is collected by analyzing large social networks, with consequent loss of quality. Moreover, this requires to pay additional resources to perform manual validation.

4.3 Retrieving relevant content from Twitter: our solution

To solve the problems presented in Section 4.2, we now introduce our topic-related content retrieval pipeline, which, given a topic T, automatically identifies the most relevant terms describing T, uses them as search criteria to download topic-related content from Twitter and analyzes it to extract influencers.

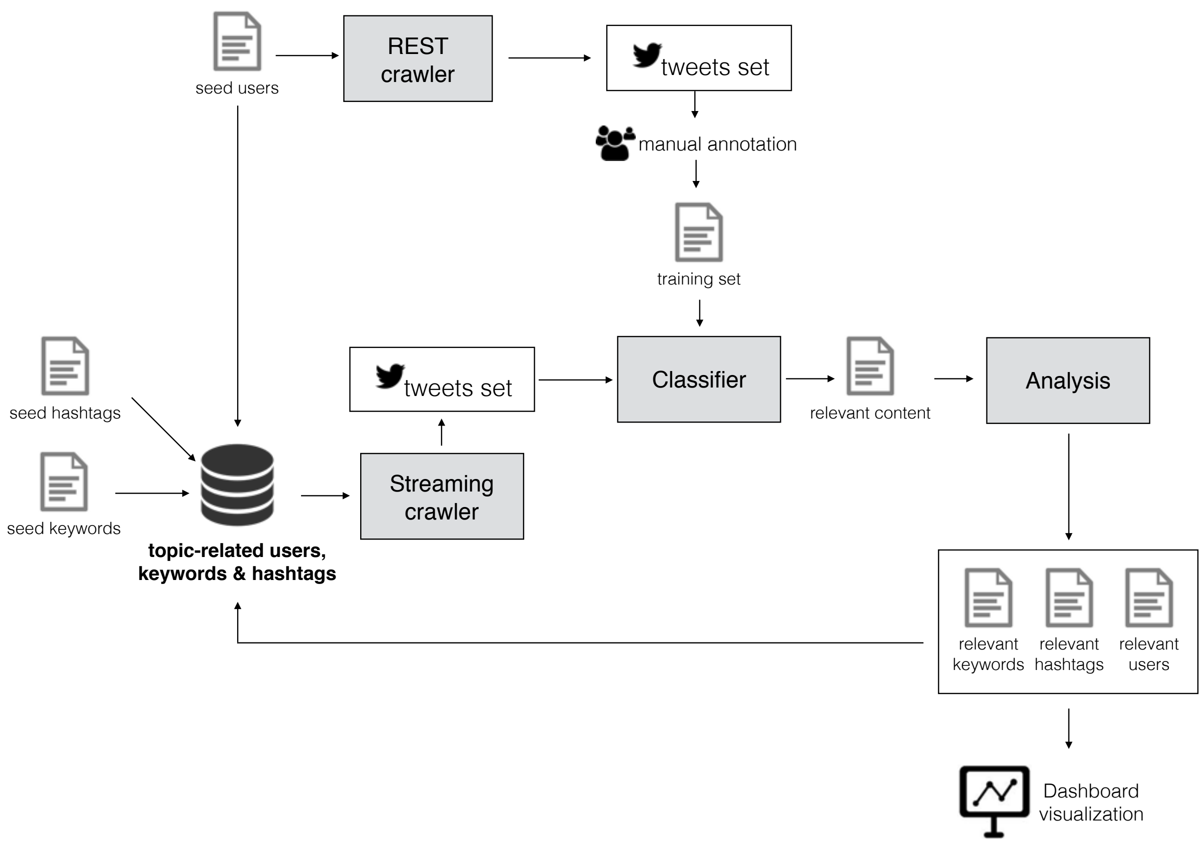


Figure 4.2: Topic-related content retrieval from Twitter: general architecture

4.3.1 Terminology

The *Twitter firehose* is the massive, real-time tweets production by users that flow from Twitter every day [5].

A *keyword* is a term that begins with a letter or a number and appears in topic-related content with statistically unusual (and typically high) frequency. A *hashtag* is a metadata tag characterized by a word or an un-spaced phrase prefixed with the sign #.

An *influencer* is a user with influence. Social influence is the ability of affecting others' emotions, opinions and behaviors.

4.3.2 Topic-related content retrieval pipeline

Figure 4.2 shows the general architecture of the topic-related content retrieval pipeline, which solves problems related to the *Topic classification*, *Key words extraction*, *Expert mining* and *Report visualization* in Figure 1.2. Given a specific topic T, the pipeline crawls the Twitter firehose, finds relevant content (i.e., content focused on topic T) and extract useful information and statistics such as: *i)* who are the influencers for the topic T; *ii)* which are the most popular keywords and hashtags for the topic T.

The pipeline is structured as follows.

At first, three lists $\mathcal{U}$, $\mathcal{K}$ and $\mathcal{H}$ (containing respectively an initial seed of relevant users, keywords and hashtags for the topic T) are built (step *Key words extraction* in Figure 1.2). The list $\mathcal{U}$ contains references to users that mainly publish topic-related content, while the lists $\mathcal{K}$ and $\mathcal{H}$ contain respectively keywords and hashtags that are typical of T. For instance, if T is

[5] `http://apivoice.com/2012/07/12/the-twitter-firehose/`

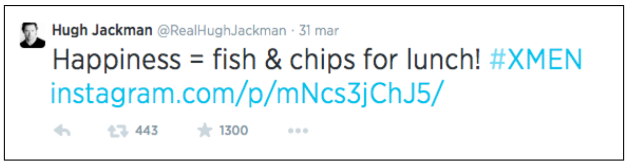

Hugh Jackman @RealHughJackman · 31 mar
Happiness = fish & chips for lunch! #XMEN
instagram.com/p/mNcs3jChJ5/
443 1300

Figure 4.3: During the release of the first trailers for the X-Men movie, many people used the hashtag `#xmen` (trending topic) to make their own tweet gain visibility, although the tweet content was not related to the movie itself

`cooking`, $\mathcal{K}$ may contain the terms `cake`, `pasta` and `apple`, while the list $\mathcal{H}$ may contain the hashtags `#RecipeOfTheDay` and `#EasyRecipes`. Section 4.7 will suggest how to build such lists.

Before the main crawling process is started, a crawler (see Section 4.4.2) accesses to the stream of users contained in the list $\mathcal{U}$ and reads tweets they published in the past days. These tweets are manually annotated as either relevant or not relevant for the topic T, and then fed as an input to a classifier (see Section 4.6), which is in this way trained to recognize relevant content for topic T (step *Topic classification* in Figure 1.2).

At this point, the lists $\mathcal{U}$, $\mathcal{K}$ and $\mathcal{H}$ are fed as an input to the real-time crawling process. A streaming crawler, i.e., a real-time crawler drinking tweets directly from the Twitter firehose (see Section 4.4.3), retrieves the tweets that either are produced by a user in $\mathcal{U}$, or contain at least one keyword from $\mathcal{K}$ or a hashtag from $\mathcal{H}$.

Notice that the retrieved content may not be relevant, although it contains one of the specified search criteria. This may happen for different reasons: *i)* users in $\mathcal{U}$ may publish something that is not related to T; *ii)* words in $\mathcal{K}$ may have multiple meanings, not all related to T (e.g., `apple` is both a fruit and a brand); *iii)* hashtags in $\mathcal{H}$ that are popular may be used to make their tweets gain visibility, although those tweets are not related to T (see an example in Figure 4.3). Hence, in the following step the trained classifier is used to filter out those tweets that are not relevant for the topic T.

Then, the relevant tweets are analyzed to extract knowledge about new keywords, hashtags and users that were not included in the lists $\mathcal{U}$, $\mathcal{K}$ and $\mathcal{H}$ (steps *Expert mining* and *Key words extraction* in Figure 1.2). This information is used to enrich those lists, and keep updated the set of search criteria. This update has several advantages:

- *Influencers*: influence changes over time [Cha et al., 2010]. Thus, keeping updated $\mathcal{U}$ allows us to discover people that are gaining influence
- *Keywords*: by updating $\mathcal{K}$ we are able to track how communication moves between the sub-topics of T
- *Hashtags*: hashtags on Twitter have a short life, and after a while people stop using them [Kywe et al., 2012]. Thus, if $\mathcal{H}$ is not updated, after a while it will not serve as an entry point for filtering the firehose, since there will not be any tweet (or a few of them) containing the hashtags in $\mathcal{H}$

Finally, a visual dashboard shows the up-to-date sets of influencers, keywords and hashtags, ordered by relevance (step *Report visualization* in Figure 1.2).

In the following sections we will describe each component of the architecture in Figure 4.2. Section 4.4 shows how to build a REST crawler and a streaming crawler for Twitter. Then, Section 4.6 illustrates how to model and train a classifier for topic-related content. Section 4.7 explains how to populate an initial version of the lists $\mathcal{K}$ and $\mathcal{H}$, and how to keep them updated.

4.4 Retrieval of topic-related content from Twitter

In this Section we illustrate the process used to retrieve topic-related content from Twitter.

4.4.1 Twitter APIs

The Twitter platform offers access to the tweets produced every day by users, via a family of APIs. Each API represents a different facet of Twitter. In the following, we present the idea behind the main APIs.

REST API. This API allows developers to access the main information behind every Twitter profile, i.e., timelines, status updates and profile information. The retrieved content includes also profile avatars and information about the graph of the people that the analyzed user is following. With this API, it is possible to specify a set of users to be followed, so that during the crawling phase the past tweets that were produced by those users are retrieved.

Streaming API. This API allows to retrieve a real-time sample of the Twitter firehose. With this API, it is possible to specify a set of keywords, hashtags and users to be tracked, so that during the crawling phase the entire firehose is filtered and only the tweets matching with one of the search criteria are retrieved. This requires to create a stable HTTP connection and maintain it as long as the crawling is needed.

These APIs come with some restrictions [6].

- When using the current REST API version (i.e., `1.1`), applications are allowed to make 150 logged out (i.e., unauthenticated) requests per hour, or 350 authenticated requests

- When using the current Streaming API version, applications are allowed to download 1% of the firehose. Whenever this limit is reached, the firehose access is momentarily stopped, and restored after some seconds (causing the loss of some data, which are passing through the firehose but not captured by the crawler)

4.4.2 Implementation of the Twitter REST crawler

A Twitter REST crawler is a crawler that reads past tweets generated by a set of specified users.

Given a list of filtering user IDs (denoted by F_u), the crawler accesses to the related profiles, reads all the tweets those users generated and stores them without any further filtering process: tweets are generally not filtered by keyword and/or hashtag in this crawling procedure.

4.4.3 Implementation of the Twitter streaming crawler

A Twitter streaming crawler is a crawler that monitors the firehose, looking for all those tweets that match some search criteria.

The search criteria for a Twitter streaming crawler, according to the Twitter APIs, are of two types: user IDs (denoted by list F_u) and keywords (denoted by list F_k). The list F_u is filled with those user IDs contained in the list $\mathcal{U}$, i.e., $F_u = \mathcal{U}$. The list F_k is instead filled with the keywords and hashtags contained in the lists $\mathcal{K}$ and $\mathcal{H}$.

The filtering lists F_u and F_k are fed as an input to the crawler, that constantly accesses the firehose and returns those tweets that contain *at least* one of the specified criteria. This means that either the tweet is produced by one of the specified users, or it contains a topic-related keyword or hashtag, or a combination of these factors.

The streaming crawler is meant to run as a long-term running process. Thus, it requires to open an HTTP connection to the Twitter API services, that remains open as long as we need to access to the firehose. Once the connection is interrupted or lost, the crawling stops and the real-time tweets that are passing through the firehose while the crawler is not working are lost forever (since there is no way of accessing to a past time frame of the firehose).

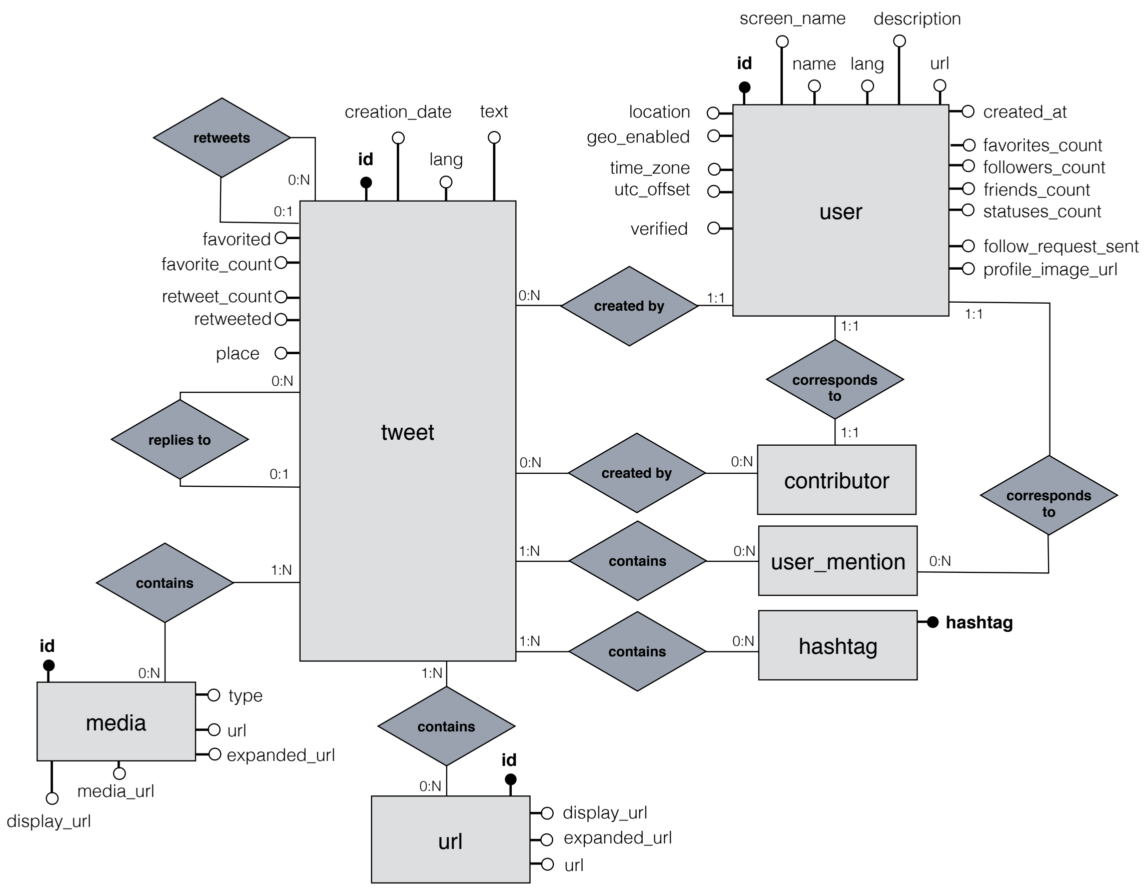


Figure 4.4: Tweet data model

4.5 Tweet data model

Figure 4.4 shows the data model used to store tweets in the database. In the following, we illustrate its details.

Tweet entity This entity represents a tweet in the database. Every tweet that is retrieved by the crawler and considered as relevant by the classifier is stored in the database according to this schema.

Every tweet is created by a Twitter user (see Section 4.5) and may contain extra content, in the form of hashtags, user mentions, media files and URLs (see Sections 4.5-4.5). Moreover, other users may contribute to its writing, and thus the tweet can be associated with a contributor (see Section 4.5).

The tweet is mainly identified by its textual content and its creation date. Moreover, it is associated with a language, which is generally the one spoken by its creator.

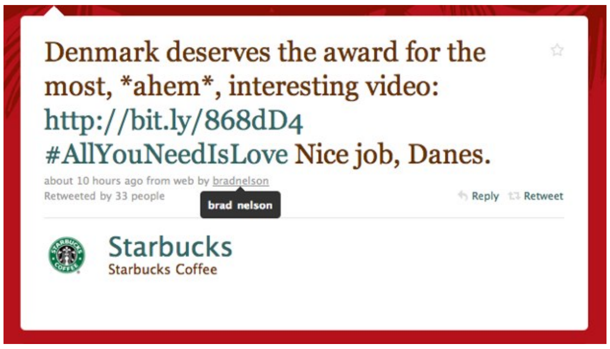
Furthermore, the Tweet entity registers if the tweet was retweeted and/or favorited, together with the number of retweets and likes the tweet received.

Finally, the entity registers also the place in which the tweet was written, which will have a non-null value in case the user allowed its device to register the geolocation while posting.

User entity This entity represents a Twitter user in the database.

A Twitter user is a subject that either creates relevant tweets (see Section 4.5), or contributes in writing tweets for other accounts (see Section 4.5), or is mentioned in other users' tweets (see

[6]`https://dev.twitter.com/docs/rate-limiting/1`


Denmark deserves the award for the most, *ahem*, interesting video: http://bit.ly/868dD4 #AllYouNeedIsLove Nice job, Danes.

about 10 hours ago from web by bradnelson

Retweeted by 33 people

Figure 4.5: Example of contribution: here the user `@bradnelson` contributes to a tweet for the `@Starbucks` profile. Source: `http://cdn.mashable.com/wp-content/uploads/2009/12/Fullscreen.jpg`

Section 4.5).

We identify a user by specifying her name, her screen name (i.e., pseudonym), her description, the URL pointing to her profile page and her language. Moreover, the entity registers some statistics about her activity on Twitter (i.e., number of followers and followees, number of published tweets, number of favorites), her location, her time zone and her profile image.

Finally, it is possible to tag a profile as verified, meaning that it represents an authentic identity. This is mainly used by public figures which would like to guarantee the authenticity of their profile.

Hashtag entity This entity represents a hashtag cited in the tweet.

User mention entity This entity represents a user mentioned in the tweet. Generally the tweet contains the screen name, represented as `@screenname`. From this, we retrieve the whole user profile and link it to the tweet citing the user.

Url entity This entity represents a URL reported in the tweet. Usually Twitter users shorten the links they post on Twitter, since tweets are limited in length. Thus, the entity contains also the expanded version of the URL.

Media entity This entity represents a multimedia file that is linked in the tweet. The entity specifies the file type (e.g., image), the URL used to retrieve the image and its expanded version.

Contributor entity A contributor is a user that generates a tweet for another Twitter profile (see an example in Figure 4.5).

This feature was created by Twitter to help companies to manage their profile: multiple contributors can publish content to the company account, so that all other contributors can see it and answer to it as in a real conversation. The feature appends the contributor's username to the tweet, so that who reads knows more about the people behind organizations.

4.6 Identification of relevant content

The Twitter streaming crawler introduced in Section 4.4.3 is used to retrieve topic-related tweets. Given a topic T and a set of search criteria for T, the crawler retrieves all the tweets that are produced in real-time and match at least one of the specified search criteria (i.e., users, hashtags, keywords). However, this does not suffice to state that the content is relevant: keywords may have multiple meanings, hashtags may be used incoherently and users are not always talking

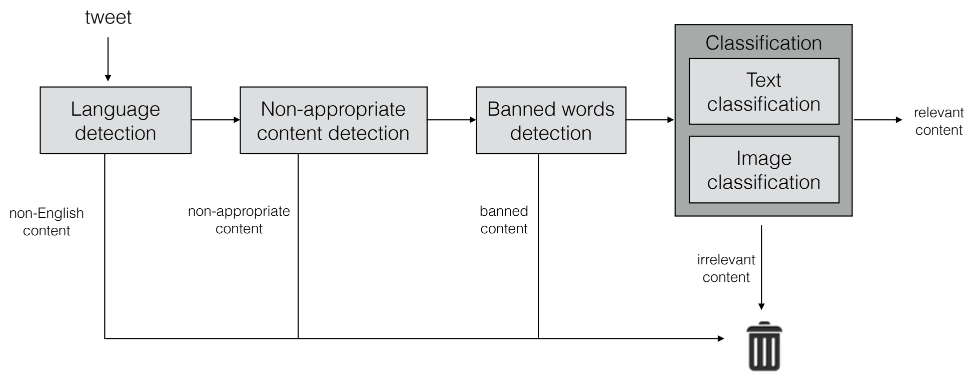


Figure 4.6: Identification of relevant content

about a specific topic. Thus, in this Section we will introduce some classification methodologies used to filter out irrelevant tweets from the result set. The identification pipeline is shown in Figure 4.6.

4.6.1 Text-based approach

In this section we illustrate how to filter out irrelevant tweets by analyzing their textual content.

The text filtering process is divided in the following phases: *i)* filter out non-English content; *ii)* filter out non-appropriate content; *iii)* classify text to state whether the tweet content is relevant for the topic T. If a tweet survives to these three stages, then it is considered relevant and thus stored in the database according to the schema in Section 4.5.

Filter out non-English content

Our target is English content. Thus, it is necessary to filter out all the tweets that are written in other languages.

As a first, simple filter one could detect the profile language of the user: if it differs from English, then the content produced by that user is automatically discarded.

However, several users write content in other languages, although their Twitter account is registered as an English profile, and viceversa. Consequently, filtering out content based on the user profile is not sufficient: the proposed simple filter ignores the language of the tweet itself, assuming that a user writes all her tweets always in the declared language.

Thus, we developed a language detector, based on the Language Detection library[7]. This component, given a tweet text, recognizes its language by comparing its content with a set of predefined language profiles. When a tweet is retrieved, if the detector recognizes that its language is not English, we discard it before performing any other classification step.

Non-appropriate content filtering

Non-appropriate content is intended as content including references to sexually explicit content and violence. All the tweets containing references to these topics should be discarded a priori, without any chance of being evaluated by the text classifier as positive samples.

To do so, we retrieved the Google list of blocked terms [Google, 2014], commonly used by the search engine to filter out inappropriate content.

When a new tweet is received by the streaming crawler, it is divided in words. Then, if the tweet contains at list one word that is present in the Google list of blocked terms, it is discarded.

[7]`https://code.google.com/p/language-detection/`

Text classification

Text classification is performed via an SVM classifier (see Section 2.3.1). In the following, we introduce the procedure used to setup the classifier.

Training set building. Given a topic T, we manually select a set of relevant users (i.e., users that are mainly writing content about topic T) and a set of non-relevant users (i.e., users who normally do not talk about topic T). Then, using a REST crawler (see Section 4.4.2), we download the tweets those users produced during a fixed time span. The first set will produce mainly positive samples containing words that are typical of T, while the second set will produce unrelated tweets (which we call *hard negatives*). All these tweets are collected in the set of training samples $\mathcal{T}$. Each training sample $(x^{(i)}, y^{(i)}) \in \mathcal{T}$ is manually annotated as either relevant (i.e., positive class with label `Y`) or non-relevant (i.e., negative class with label `N`). Finally, only in case the number of positive and negative samples in $\mathcal{T}$ differs, we rebalance the dataset by downloading other tweets, annotating them and selecting only those ones that are needed (i.e., positive samples if the number of negatives is larger than the number of positives, or viceversa).

Tweet processing. Every tweet is subdivided into words. A *word* is a piece of text surrounded by either spaces or punctuation. From the list of words we delete the user mentions (typically written as `@username`), since they are not useful for understanding the topic of the analyzed text. Moreover, we remove the stop words. Hashtags, on the other hand, could be used as discriminative features, and thus are kept in the list (although without the # symbol). Then, we normalize all the words, by lowering all the letters and applying Porter stemming.

Classifier training. We subdivide the set of collected samples $\mathcal{T}$ in training set $\mathcal{T}_{\text{train}}$ (60% of dataset), cross validation set $\mathcal{T}_{\text{CV}}$ (20% of dataset) and test set $\mathcal{T}_{\text{test}}$ (20% of dataset). Then, we process the tweets in $\mathcal{T}_{\text{train}}$ with the procedure explained above. All the stemmed words that survive the processing are collected in a unique dictionary D, which constitutes the feature set. The feature vector for each tweet in the training set $\mathcal{T}_{\text{train}}$ is thus built according to a TF-IDF approach. In this vector, the j-th component for the i-th tweet is computed as:

$$v_{ij} = TF(f_j, d_i) IDF(f_j) \tag{4.1}$$

where $TF(f_j, d_i)$ is the term-frequency of the feature f_j in the tweet d_i:

$$TF(f_j, d_i) = \texttt{frequency}(f_j, d_i)$$

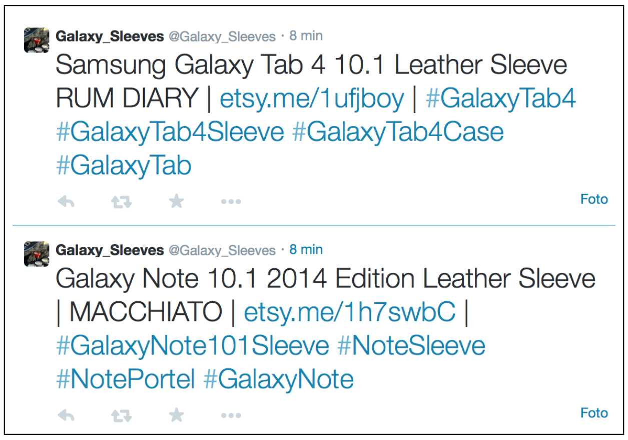
while $IDF(f_j)$ is the inverse document frequency:

$$IDF(f_j) = \log \frac{|\mathcal{T}_{\text{train}}|}{|\{d \in \mathcal{T}_{\text{train}} | f_j \in d\}|}$$

The SVM classifier is finally trained on the available training set $\mathcal{T}_{\text{train}}$, by varying the SVM parameters C and σ. The combination of parameters that guarantees the best performance on the cross validation set $\mathcal{T}_{\text{CV}}$ is selected, and then the classifier performance is computed on the test set $\mathcal{T}_{\text{test}}$ using the selected parameters.

Banned words

In this Section, we introduce a new filter that allows one to filter out tweets containing specific unwanted terms. As an example, consider Figure 4.7. Here, we report two non-relevant tweets produced by the user @`Galaxy_Sleeves` and classified as relevant by the textual classifier for the


Galaxy_Sleeves @Galaxy_Sleeves · 8 min
Samsung Galaxy Tab 4 10.1 Leather Sleeve RUM DIARY | etsy.me/1ufjboy | #GalaxyTab4 #GalaxyTab4Sleeve #GalaxyTab4Case #GalaxyTab
Foto

Galaxy_Sleeves @Galaxy_Sleeves · 8 min
Galaxy Note 10.1 2014 Edition Leather Sleeve | MACCHIATO | etsy.me/1h7swbC | #GalaxyNote101Sleeve #NoteSleeve #NotePortel #GalaxyNote
Foto

Figure 4.7: An example of misclassified tweets for the topic *food*

topic *food*. As one may notice, the misclassification occurred because the tweet text, if cleaned from all the extra content (URLs, hashtags), reports either terms that are not known by the classifier (since they are very specific for another topic, e.g., Samsung for the topic *smartphones*) or terms that are apparently related to food. As a result, our pipeline continuously crawled content from this account, detecting it as influencer for the selected topic. However, this user is a bot that automatically publishes advertisements for food-shaped smartphones sleeves and pouches. Thus, it cannot be considered a relevant user for the topic *food*.

As a solution, one could think of retraining the classifier every time these situations occur, by introducing some hard negative samples containing the unwanted terms. Unfortunately, this operation is costly, since it requires to download other samples (which are in part containing the content to be filtered), annotating them as relevant/non-relevant, and re-parameterize the classifier. On the other hand, if we could have the possibility of filtering some keywords from the crawled one (e.g., in the example above, Samsung and Galaxy), one could avoid to retrieve (and misclassify) the unwanted content.

Consequently, we introduced a new filtering level in the pipeline, that allows one to filter out content containing some manually filtered words. This filter works exactly as the non-appropriate content filter, but instead of working on a fixed taxonomy, it allows us to manually populate the list of filtered words. Thus, every time we detect terms that deterministically happen to be misclassified (e.g., Galaxy in the example above), we add those terms in the filtered words list: from that point on, the pipeline will automatically discard tweets containing any of the specified filtered terms.

4.6.2 Image-based approach

In this Section we illustrate how to filter out irrelevant tweets by analyzing their multimedia content.

The image classification process is divided in the following phases: *i)* extraction of a common vocabulary for both negative and positive images; *ii)* training set building; *iii)* classification of images to state whether they are relevant for the topic T. In the following, we operate on a dataset of images $\mathcal{T}^I$ extracted from tweets in the set $\mathcal{T}$ (both relevant and non-relevant), which were downloaded in a fixed time span.

Extraction of a common vocabulary

We manually annotate the images in $\mathcal{T}^I$ as relevant for the topic T (i.e., belonging to the positive class) or non-relevant (i.e., belonging to the negative class). Then, we extract an equal (and small) number of positive samples and negative samples from $\mathcal{T}^I$ and analyze them: firstly, we extract their key-points, and then we compute the related SIFT descriptors [Lowe, 1999]. These descriptors, also known as *visual words*, characterize the set of candidates to be considered as features in the feature set. By applying the well-known k-means clustering technique, we aggregate the extracted descriptors in W clusters, and extract the center of the learned clusters as representative terms. These W extracted visual terms characterize the visual dictionary, and thus will be used as feature set.

Training set building

We subdivide the set of images $\mathcal{T}^I$ in three sets: the training set $\mathcal{T}^I_{\text{train}}$, the cross validation set $\mathcal{T}^I_{\text{CV}}$ and the test set $\mathcal{T}^I_{\text{test}}$. Each sample $(x^{(i)}, y^{(i)})$ in these sets is then analyzed to extract its feature vector, as follows. At first, we extract the key-points of $(x^{(i)}, y^{(i)})$ (and related descriptors). Then, for each key-point we find the three nearest neighbor in the dictionary of visual terms (i.e., the three terms whose descriptors are the most similar to the query key-point). Finally, we build a histogram of occurrences of the found nearest neighbors and normalize it. The output of this procedure produces the feature vector for the sample $(x^{(i)}, y^{(i)})$.

Classifier training

An SVM classifier is finally trained on the available training set $\mathcal{T}^I_{\text{train}}$, as in Section 4.6.1.

4.6.3 Combine multiple classifiers into a multi-modal classifier

Let $\Gamma = \{\mathtt{Y}, \mathtt{N}\}$ be the set of relevance classes for a topic T. A tweet t belongs to class $\gamma = \mathtt{Y}$ if it is relevant for T, and to class $\gamma = \mathtt{N}$ otherwise. We will denote with $l^T(t)$ the true label of t. This label takes values in the set Γ, i.e., $l^T(t) = \mathtt{Y}$ if t is relevant for T, and $l^T(t) = \mathtt{N}$ otherwise.

Let $\mathcal{C}$ be the set of classifiers we use to classify each tweet as either relevant or non-relevant for topic T (e.g., text classifier, image classifier). Each classifier $c \in \mathcal{C}$, when a tweet t is provided, returns the label $l^c(t)$, which again takes values in the set Γ, i.e., $l^c(t) = \mathtt{Y}$ if the classifier classifies t as relevant, and $l^c(t) = \mathtt{N}$ otherwise.

Our purpose is to combine the set of labels $l^c(t)$ provided by the classifiers $c \in \mathcal{C}$ so as to find the aggregated label $L(t)$. In the following, we introduce a method based on the work [Xu et al., 1992], which applies Bayesian formalism and belief functions to estimate the aggregated label $L(t)$.

First, let $\mathtt{CM}(c)$ denote the confusion matrix for the classifier $c \in \mathcal{C}$, defined as:

$$\mathtt{CM}(c) = \begin{bmatrix} n_{\mathtt{Y},\mathtt{Y}} & n_{\mathtt{Y},\mathtt{N}} \\ n_{\mathtt{N},\mathtt{Y}} & n_{\mathtt{N},\mathtt{N}} \end{bmatrix}$$

where each row corresponds to a true label $l^T(t)$ (i.e., either Y or N), each column corresponds to a classifier label $l^c(t)$ (i.e., either Y or N), and $n_{i,j}$ is the number of samples (in a test set) with true label $l^T(t) = i$ and classifier label $l^c(t) = j$.

Suppose the classifier c suggested the label $l^c(t)$ for the sample t. Hence, the probability that t is of class $\gamma \in \Gamma$ (i.e., $l^T(t) = \gamma$), given the label $l^c(t)$, can be computed as follows:

$$\Pr(l^T(t) = \gamma | l^c(t)) = \frac{n_{\gamma, l^c(t)}}{\sum_{\bar{\gamma} \in \{\mathtt{Y},\mathtt{N}\}} n_{\bar{\gamma}, l^c(t)}} \tag{4.2}$$

We can aggregate such probabilities so as to build the belief function $bel(t, \gamma)$. This function represents our belief that a tweet t belongs to a class $\gamma \in \Gamma$ given the set of opinions $\{l^c(t)\}$ by the classifiers $c \in \mathcal{C}$. The belief function is expressed as:

$$bel(t, \gamma) = \eta \prod_{c \in \mathcal{C}} \Pr(l^T(t) = \gamma | l^c(t)) \tag{4.3}$$

where η is a constant defined as:

$$\eta = \frac{1}{\sum_{\gamma \in \{\mathtt{Y},\mathtt{N}\}} \prod_{c \in \mathcal{C}} \Pr(l^T(t) = \gamma | l^c(t))} \tag{4.4}$$

Finally, the aggregated label $L(t)$ is computed as:

$$L(t) = \begin{cases} \mathtt{Y} & bel(t, \mathtt{Y}) > bel(t, \mathtt{N}) \\ \mathtt{N} & \text{otherwise} \end{cases} \tag{4.5}$$

where $bel(t, \mathtt{Y})$ and $bel(t, \mathtt{N})$ are the belief that the tweet t belongs to class Y and N, respectively.

4.7 Definition of search keywords and hashtags set

The Twitter streaming crawler shown in Section 4.4.3 retrieves tweets that either are produced by users in the list $\mathcal{U}$, or match keywords and hashtags in the lists $\mathcal{K}$ and $\mathcal{H}$.

In this Section, we explain how to populate conveniently the lists $\mathcal{K}$ and $\mathcal{H}$, so that they are dynamically modified to contain keywords and hashtags that describe the real-time communication trends on Twitter.

4.7.1 Static selection of filtering terms

Keyword and hashtag lists $\mathcal{K}$ and $\mathcal{H}$ could be initially populated by manually selecting the most relevant keywords and hashtags in a predefined time span, and keeping them unchanged along the whole crawling period.

Let $\mathcal{T}$ be a set of samples that were downloaded from Twitter and manually annotated to confirm whether they are relevant for a specific topic T or not. The static selection of keywords and hashtags requires to do an analysis of the content of the samples in $\mathcal{T}$ to find the elements (both keywords and hashtags) that are very frequent in the relevant samples and almost absent from the non-relevant samples. These elements are the most representative terms for the topic T in the monitored sample.

Let $\mathcal{W}$ be the set of words (either a set of keywords $\mathcal{W}_K$ or a set of hashtags $\mathcal{W}_H$) that are extracted from the set of samples $\mathcal{T}$. These words are pre-filtered so as to remove stop words; duplicates are not removed. Each word $w \in \mathcal{W}$ is associated with a score $s(w)$, computed as follows:

$$s(w) = \log \frac{p(1-r)}{(1-p)r} \tag{4.6}$$

where:

$$p = \frac{\Pr(\text{relevant}|w)\Pr(w)}{\Pr(\text{relevant})} \tag{4.7}$$

and:

$$r = \frac{\Pr(\text{non-relevant}|w)\Pr(w)}{\Pr(\text{non-relevant})} \tag{4.8}$$

This score takes large values in case the word w appears very frequently in the relevant documents (i.e., documents $t \in \mathcal{T}$ such that $\text{class}(t) = \text{relevant}$) and almost never in the non-relevant documents (i.e., documents $t \in \mathcal{T}$ such that $\text{class}(t) = \text{non-relevant}$).

$\Pr(\text{relevant}|w)$ is the probability that, once a word w is selected, it is relevant for topic T:

$$\Pr(\text{relevant}|w) = \frac{|\{t \in \mathcal{T} : \text{class}(t) = \text{relevant} \wedge w \in t\}|}{|\{t \in \mathcal{T} : w \in t\}|}$$

Equivalently, $\Pr(\text{non-relevant}|w)$ is the probability that, once a word w is selected, it is not relevant for topic T:

$$\Pr(\text{non-relevant}|w) = \frac{|\{t \in \mathcal{T} : \text{class}(t) = \text{non-relevant} \wedge w \in t\}|}{|\{t \in \mathcal{T} : w \in t\}|} = 1 - \Pr(\text{relevant}|w)$$

Algorithm 6: EXTRACTSEARCHCRITERIA keeps updated the list of search criteria (i.e., keywords and hashtags) by analyzing the incoming tweets

```
Input: Time period $P$, Annotated samples $\mathcal{T}$
 1. $\mathcal{W} = \texttt{terms}(\mathcal{T})$
 2.
 3. while true
 4.     $\bar{\mathcal{W}} = \texttt{unique}(\mathcal{W})$
 5.
 6.     for each $w \in \bar{\mathcal{W}}$
 7.         Compute $s(w)$
 8.     Order $\bar{\mathcal{W}}$ in descending order of $s(w)$
 9.     $\mathcal{W}^* =$ top-$K$ elements of $\bar{\mathcal{W}}$
10.
11.     $\mathcal{T} =$ tweets retrieved by pipeline during period $P$
12.     $\mathcal{W} = \mathcal{W} \cup \texttt{terms}(\mathcal{T})$
```

Pr(relevant) is the probability of a sample $t \in \mathcal{T}$ of being relevant for topic T:

$$\Pr(\text{relevant}) = \frac{|\{t \in \mathcal{T} : \text{class}(t) = \text{relevant}\}|}{|\mathcal{T}|}$$

Equivalently, Pr(non-relevant) is the probability of a sample $t \in \mathcal{T}$ of being not relevant for topic T:

$$\Pr(\text{non-relevant}) = \frac{|\{t \in \mathcal{T} : \text{class}(t) = \text{non-relevant}\}|}{|\mathcal{T}|} = 1 - \Pr(\text{relevant})$$

Finally, $\Pr(w)$ is the probability that, when a word is sampled from $\mathcal{W}$, it coincides with w:

$$\Pr(w) = \frac{|\{v \in \mathcal{W} : v = w\}|}{|\mathcal{W}|}$$

When the scores for all the words $w \in \mathcal{W}$ are computed, we order the words in descending order of $s(w)$. Then, the top-K words ($K = 50$) are selected as the most relevant words for the topic T.

Notice that this procedure does not depend on the fact that the list $\mathcal{W}$ contains hashtags or keywords. Thus, the procedure is run two times (once for hashtags and once for keywords), so that the top-K selected hashtags populate the list $\mathcal{H}$, while the top-K selected keywords populate the list $\mathcal{K}$.

4.7.2 Automatic expansion of filtering terms

A static list of filtering terms (such as keywords and hashtags) is not the best solution one could adopt: it does not reflect the current discussions and topics, and thus it could include outdated terms that are not used frequently by users. A typical example is the one of hashtags, which have a limited life: after a while, Twitter users stop using them and create new hashtags. If we select hashtags statically in a predefined time span, these will cease to exist very soon: thus, if we use them to crawl content, we will not retrieve any tweet, since there will not be any tweet containing them.

Thus, we propose a way to automatically keep updated the keyword list $\mathcal{K}$ and hashtag list $\mathcal{H}$. Since the procedure does not depend on the fact that we are extracting hashtags or keywords, in the following we indicate the set of candidates with $\mathcal{W}$ and the set of selected search criteria with $\mathcal{W}^*$, which can correspond either to sets of hashtags or to sets of keywords.

We are given with a set of annotated samples $\mathcal{T}$. At first, we extract a set of candidate terms $\mathcal{W}$ from $\mathcal{T}$, which is constituted by all the (non-stop words) terms contained in each sample $t \in \mathcal{T}$

(line 1). This set will be periodically updated, with period P (set to 10 minutes). Every time an update is requested, we extract the set of unique words $\bar{\mathcal{W}}$ contained in $\mathcal{W}$ (line 4) and we compute their score $s(w)$, according to Equation (4.6) (line 6). The list $\bar{\mathcal{W}}$ is then ordered in descending order of $s(w)$, and the top-K words are extracted as new search criteria $\mathcal{W}^*$ (lines 8-9). Then, new tweets are collected until the next update, that will happen after a period P. During the next update, the terms extracted from the newly collected tweets are added to $\mathcal{W}$ (line 12), and $\mathcal{W}^*$ is recomputed accordingly.

4.7.3 Filter out non-relevant keywords

Unfortunately, a completely automatic solution for the expansion of the keywords and hashtags set is not accurate. In fact, if for some reason (e.g., a misclassification error of the downloaded content) a non-relevant keyword happens to be in the list $\mathcal{K}$, then the crawler will start downloading non-relevant content and will introduce noise in the data set.

Thus, we decided to introduce a new filtering level on the keyword selection process. Hashtags, on the other hand, are not filtered, since they are not necessarily attributable to a meaningful expression, and thus it is difficult to analyze them automatically to understand whether they are relevant for topic T or not.

In order to filter out non-relevant keywords, we built a *topic taxonomy* out of already existing taxonomies. In particular, since in our use case we considered the topic *food*, we merged the following taxonomies: *i)* NDSR 2014 Foods in the NCC Food and Nutrient Database[8]; *ii)* the Google product taxonomy, kitchen and food sections[9]. The obtained set of terms are typical of the selected topic, and thus useful to filter out all the words that do not match with topic T. Consequently, we added a new step in Algorithm 6, so that during the keyword selection process, if a keyword with high score is not included in the topic taxonomy, then it is automatically discarded, since it is not relevant for topic T.

4.8 Influencers retrieval

In this Section we are going to introduce the influence metrics we will use to compute the influence degree of users in social media.

4.8.1 Influence metrics

Preliminaries

Let $\mathcal{R}^t$ be the set of relevant tweets, i.e., the set of tweets that were retrieved from the firehose and classified as relevant for topic T by the classifier. Moreover, let $\mathcal{R}^u$ be the set of relevant users, i.e., the users that tweeted at least one of the tweets in $\mathcal{R}^t$.

Let $\texttt{creator}(t)$ be the user that originally created the tweet t.

Let $\mathcal{P}(u)$ be the set of tweets published by the user $u \in \mathcal{R}^u$, defined as follows:

$$\mathcal{P}(u) = \{t \in \mathcal{R}^t : \texttt{publisher}(t) = u\}$$

where $\texttt{publisher}(t)$ is the user that published the tweet t.

Let $\mathcal{O}(u)$ be the set of original tweets published by the user $u \in \mathcal{R}^u$, defined as follows:

$$\mathcal{O}(u) = \{t \in \mathcal{R}^t : \texttt{publisher}(t) = u \wedge \texttt{retweeted}(t) = \texttt{false}\}$$

where $\texttt{retweeted}(t)$ states whether t is a retweet of another tweet or not.

[8] http://www.ncc.umn.edu/products/database.html
[9] https://support.google.com/merchants/answer/1705911?hl=en

Let $\texttt{retweet}(t)$ denote a retweet of the original tweet t. Moreover, let $\texttt{RT}(t)$ be the set of retweets for the tweet t as follows:

$$\texttt{RT}(t) = \{t' \in \mathcal{R}^t : t' = \texttt{retweet}(t)\}$$

Let $\texttt{RT}(u)$ be the set of retweets made by u for tweets created by others.

Let $\texttt{mention}(u, u')$ denote a user mention of the user u made by u'. Moreover, let $\texttt{UM}(u)$ be the set of users mentioning u.

Metrics

The metrics proposed in this Section is based on the work [Pal and Counts, 2011], where the components are adapted to the real-time case (i.e., the case in which content is downloaded in real-time). In this case, in fact, APIs present some limitations on the retrieval of information.

Let $u \in \mathcal{R}^u$ be a user who either published or retweeted a relevant tweet.

The *creativity* $C(u)$ of u expresses her ability of generating topic-related original content:

$$C(u) = \frac{|\mathcal{O}(u)|}{|\mathcal{P}(u)|} \tag{4.9}$$

The *retweet impact* of a user u measures the impact of retweets of a Twitter user, expressed as the difference between how much u is retweeted by others and how much u retweets others.

$$\begin{aligned} RI(u) = \;& \textstyle\sum_{t \in \mathcal{O}(u)} |\texttt{RT}(t)| \cdot \log(\sum_{t \in \mathcal{O}(u)} |\texttt{publisher}(\texttt{RT}(t))|) + \\ & -|\texttt{RT}(u)| \cdot \log(|\texttt{creator}(\texttt{RT}(u))|) \end{aligned} \tag{4.10}$$

The *mention impact* of a user u measures the impact of mentions of a Twitter user, expressed as the difference between how much u is mentioned by others and how much u mentions others.

$$\begin{aligned} UMI(u) = \;& \textstyle\sum_{u' \in \mathcal{R}^u \setminus u} |\texttt{mention}(u, u')| \cdot \log(|\texttt{UM}(u)|) + \\ & -\textstyle\sum_{u' \in \mathcal{R}^u \setminus u} |\texttt{mention}(u', u)| \cdot \log(\sum_{u' \in \mathcal{R}^u \setminus u} |\texttt{UM}(u')|) \end{aligned} \tag{4.11}$$

The *activity* $A(u)$ of u measures the activity level of u on Twitter. Let τ be the time that passed after the last published tweet of user u. Then:

$$A(u) = e^{-\alpha\tau} \tag{4.12}$$

The *multimedia enrichment* $M(u)$ measures the rate of publication of multimedia content:

$$M(u) = \frac{|\{t \in \mathcal{O}(u) : \texttt{containsImage}(t) = \texttt{true}\}|}{|\mathcal{O}(u)|} \tag{4.13}$$

The *external content enrichment* $E(u)$ measures the rate of publication of external content, via URLs included in the tweet:

$$E(u) = \frac{|\{t \in \mathcal{O}(u) : \texttt{containsURLs}(t) = \texttt{true}\}|}{|\mathcal{O}(u)|} \tag{4.14}$$

4.9 Visualization of results

In the following, we illustrate the Web application used to visualize reports about influencers analysis. We will refer to this application as *dashboard*.

Two kinds of analysis can be shown to users via the dashboard: real-time analysis on the data that are currently being read from the firehose, and past data analysis on the data that were stored in the database in past moments.

The Web application architecture is shown in Figure 4.8. The main components are: *i)* the *interface*, which constitutes the interface that shows users a detailed report of the retrieved information; *ii)* the *REST API*, which connects to the database containing the crawled data and retrieves them.

In the following, we introduce the design of these two components.

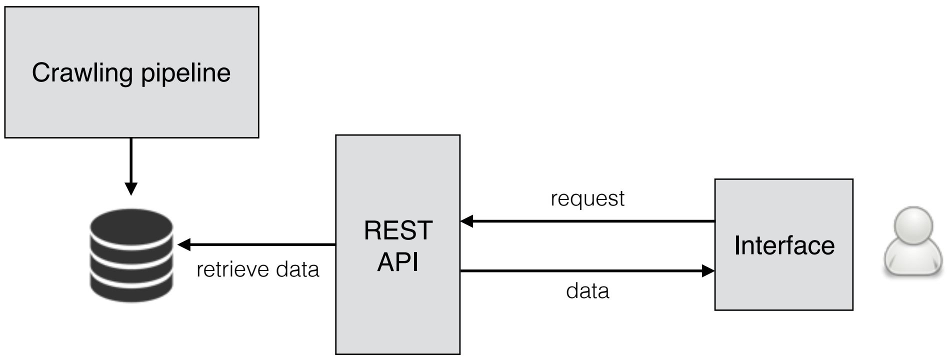


Figure 4.8: Architecture of the Web application used to show reports on influencers analysis

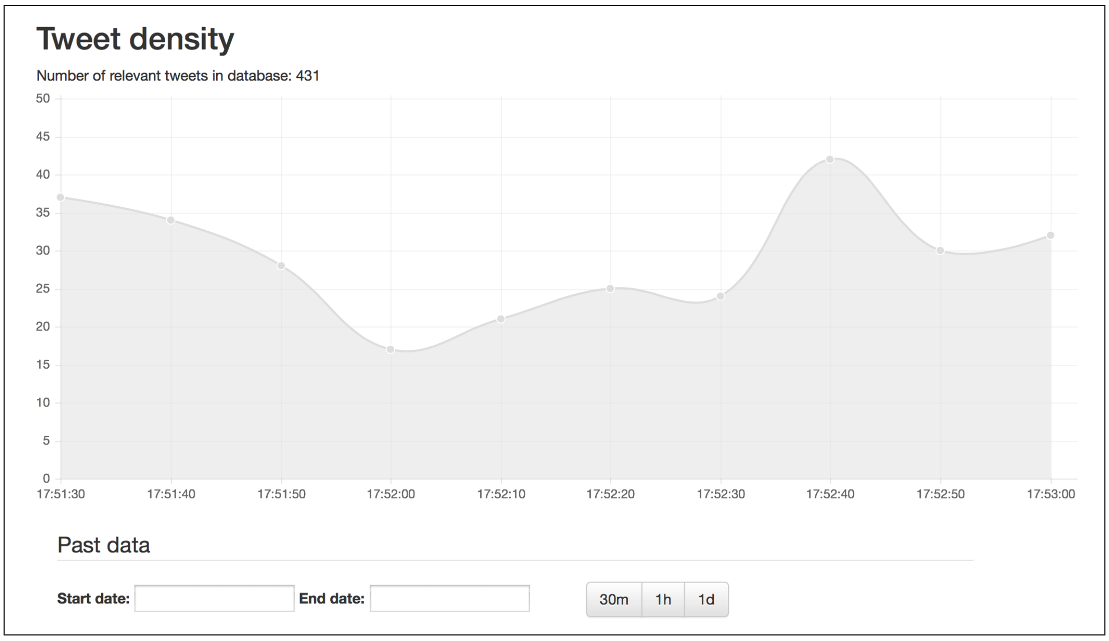


Figure 4.9: Real-time tweet density, showing the number of incoming tweets

4.9.1 Design of visual interface

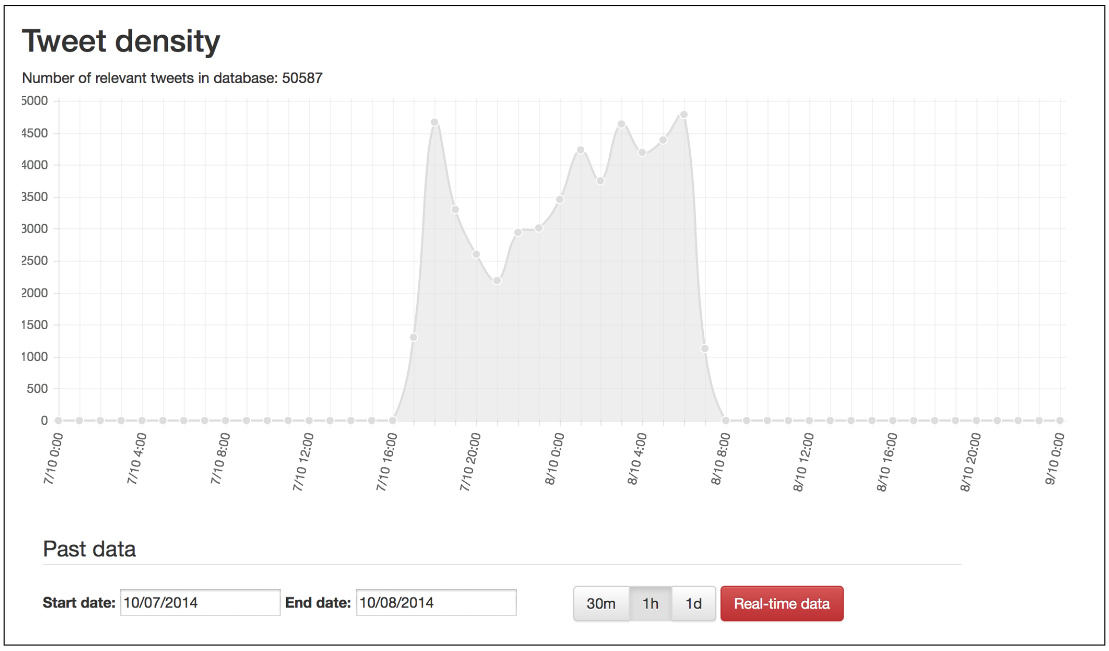
The interface displays the following reports: a tweet density graph, a feed of incoming tweets, a list of influencers, a keywords tag cloud and a hashtags tag cloud.

Tweet density graph

The tweet density graph shows the number of incoming tweets (either current or past data).

The real-time data view is shown in Figure 4.9. With this view, the user is provided with the density of incoming tweets that are read in real-time from the firehose. This view is automatically updated every 10 seconds, so as to dynamically change its content as tweets are stored in the database. From this view, it is possible to move to the past data view of the same graph.

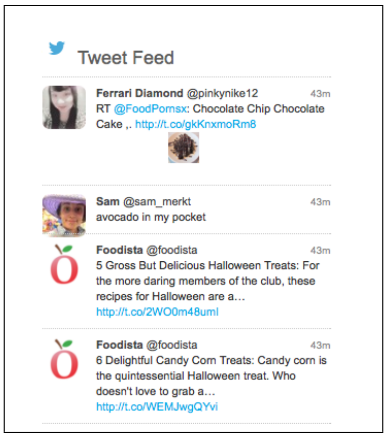
The past data view is shown in Figure 4.10. With this view, the user is provided with the density of tweets that were collected in the past. This view is not periodically updated: the user is required to specify the period (in the form of start date and end date) for which she would like to visualize the collected data. The user can specify the visualization period for the data (30 minutes, 1 hour or 1 day). From this view, it is possible to move to the real-time data view of the same graph.



Number of relevant tweets in database: 50587

Start date: 10/07/2014 **End date:** 10/08/2014

Figure 4.10: Past tweet density, showing the number of tweets collected in the past



Ferrari Diamond @pinkynike12 43m
RT @FoodPornsx: Chocolate Chip Chocolate Cake ,. http://t.co/gkKnxmoRm8

Sam @sam_merkt 43m
avocado in my pocket

Foodista @foodista 43m
5 Gross But Delicious Halloween Treats: For the more daring members of the club, these recipes for Halloween are a… http://t.co/2WO0m48uml

Foodista @foodista 43m
6 Delightful Candy Corn Treats: Candy corn is the quintessential Halloween treat. Who doesn't love to grab a… http://t.co/WEMJwgQYvi

Figure 4.11: Incoming tweets feed

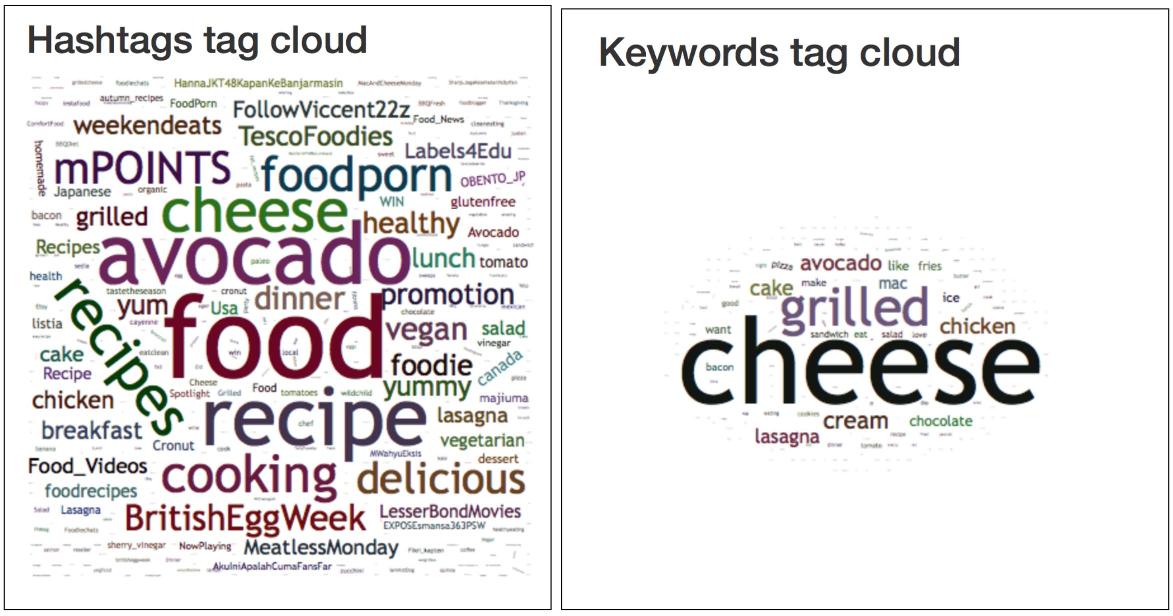
Incoming tweets feed

The incoming tweets feed shows the last 20 received tweets, and is updated dynamically as tweets are stored in the database. The most recent tweets are shown in the upper part of the list. If the tweet contains an image, a clickable thumbnail is shown together with the text: if the user clicks on it, the original image is displayed. Moreover, by clicking on the user name, the original user profile is shown on Twitter.

Influencers list

The influencers list is a ranked list showing the top-20 current influencers. Each name, if clicked, brings to the influencer's Twitter user profile. The users are ordered according to the influence score (see Section 4.8).

It is possible to specify a fixed time period in the past (in the form of start date and end date) and visualize the influencers in that period.



(a) Hashtags (b) Keywords

Figure 4.12: Keywords and hashtags tag clouds

Keywords and hashtags tag clouds

The keywords and hashtags tag clouds show the most used keywords and hashtags in a fixed period of time (either in real-time or in past data).

These views are shown in Figure 4.12. The interface rendering is the same for past data and real-time data, although the source changes: when past data visualization is required, the user specifies the period (in the form of start date and end date) for which she would like to visualize the collected data, while if real-time data visualization is required, the most frequent terms in the last 12 hours are shown.

4.9.2 Visualizing the map of influential users

Every user on Twitter, when she registers herself to the platform, can specify where she lives by inserting a textual description of her permanent address. Unfortunately, this field is neither mandatory nor constrained to any rules. Thus, while a large percentage of users specify their location, others either do not specify it, or provide imaginary locations (e.g., *Neverland* and *Wonderland*), inaccurate descriptions and text containing typos.

In the following, we illustrate how we extracted the location for each influential user, so as to visualize the top-25 list of influencers in a map.

Extraction of user location

Our objective is to extract geographical coordinates (i.e., latitude and longitude) from a textual location description.

Several GIS tools that extract latitude and longitude from a text are available. These services are usually queried via a REST interface, and go through the available maps to look for the place whose name is the most similar to the provided text. Then, they extract the coordinates of the selected place and return them to the user that queried the service.

In this work, we selected the ARCGIS service[10]. Given a location textual description (provided by the user), we query the service to retrieve the related coordinates. The system looks

[10] `https://geocode.arcgis.com`

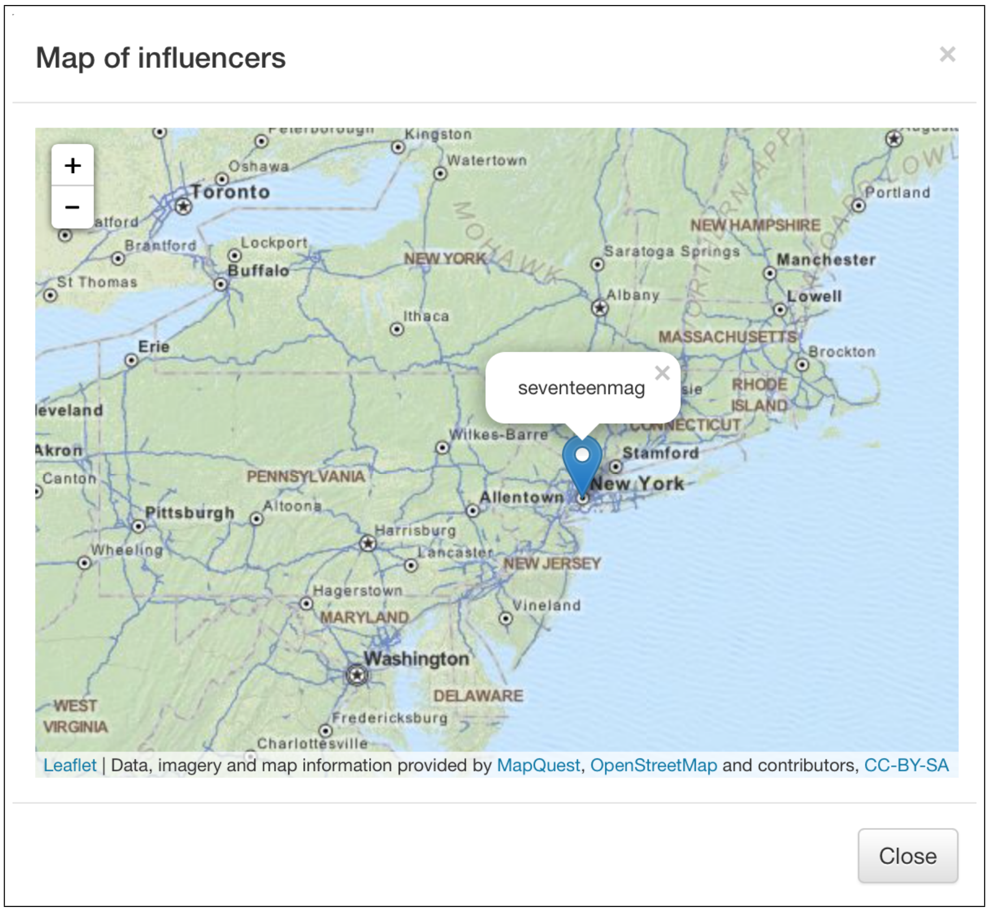

Map of influencers
seventeenmag
Leaflet | Data, imagery and map information provided by MapQuest, OpenStreetMap and contributors, CC-BY-SA
Close

Figure 4.13: Influencers map

for the provided location in the maps, and if it does not find it (e.g., in case of imaginary or wrong locations) it returns empty coordinates. Otherwise, the correct coordinates are retrieved.

Visualization on a map

We modified the visual interface introduced in Section 4.9 so as to visualize the list of influencers in a map.

We used the JavaScript Leaflet library[11], which, given as an input a list of points (described by latitude and longitude), displays them in the world map, as shown in Figure 4.13. Each pin pointing to a location of an influencer is decorated with the influencer username.

4.10 Implementation details

4.10.1 Pipeline for content retrieval

Parallelization

The Twitter throughput can be really large in case the selected search criteria are used frequently by users. Consequently, it may happen that the crawler retrieves a really large number of tweets, that are stored in memory waiting to be processed. However, the crawling process is faster than the tweet analysis process, and thus as a result the tweets are continuously accumulated in the memory, until it saturates and the process is killed.

[11] `http://leafletjs.com`

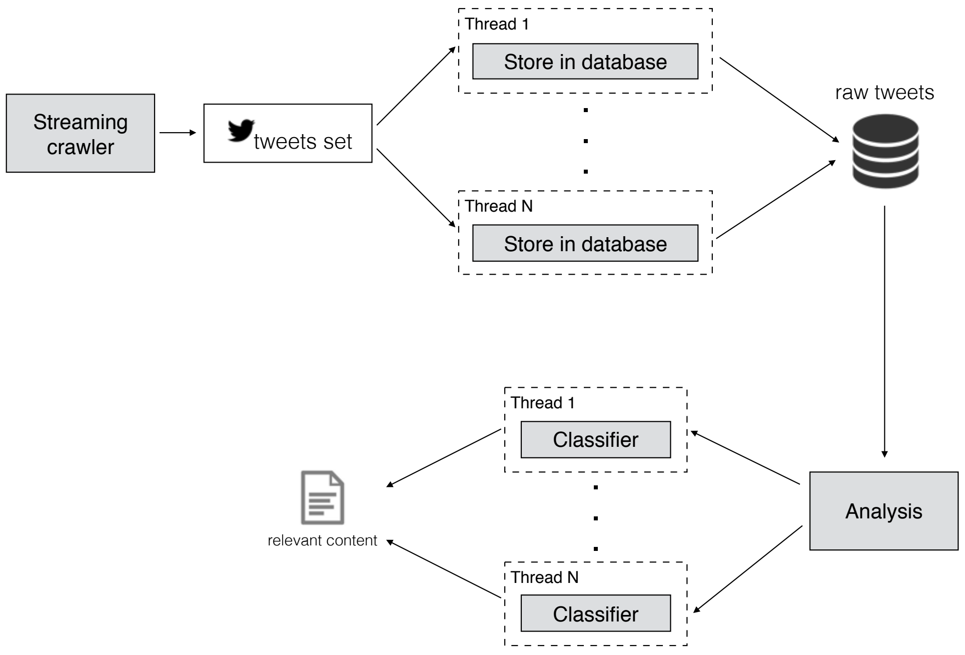


Figure 4.14: Parallelization of topic-related content retrieval pipeline

Hence, we modified the pipeline so as to parallelize the crawling and analysis process. Figure 4.14 shows the result of the parallelization of the architecture presented in Figure 4.2. The idea is to store momentarily in the database those tweets that are still not processed. The architecture is consequently divided in two parts: the first segment reads tweets from the incoming queue and stores them in the database; the second segment extracts the non-processed tweets from the database and analyzes them to understand whether they are relevant or not (in case throwing away the non-relevant ones). Moreover, to speed up the process, we designed the two segments so that the processing is done in parallel by N threads. Consequently, N different threads extract tweets from the queue to store them in the database, and N other threads read from database to analyze them. As a result, the incoming tweets do not saturate the memory, and the process can run continuously without being stopped due to low performance.

4.10.2 Dashboard

REST API for data statistics retrieval

The following calls are used by the interface to retrieve real-time and past data via the REST API for the tweet density graph:

- `rest/tweetPastCount?startDate=&endDate=&mode=` retrieves past data between `startDate` and `endDate`. The parameter `mode` specifies the data visualization mode:
  - `mode=m` requires a visualization period of 30 minutes
  - `mode=h` requires a visualization period of 1 hour
  - `mode=d` requires a visualization period of 1 day
- `rest/tweetCount` retrieves the number of relevant tweets currently stored in the database
- `rest/periodicTweetCount` retrieves the last 10 tweets count of the relevant tweets collection, to be displayed as points in the real-time tweet density graph

The following calls are used to retrieve data to populate keywords and hashtags tag clouds:

- `rest/keywords?startDate=&endDate=` retrieves the keywords contained in tweets collected from `startDate` and `endDate`
- `rest/hashtags?startDate=&endDate=` retrieves the hashtags contained in tweets collected from `startDate` and `endDate`

The following call retrieves influencers:

- `rest/influencers?startDate=&endDate=` retrieves the top-20 influencers in the period between `startDate` and `endDate`

Finally, the following call retrieves tweets for the tweet feed:

- `rest/tweetFeed?startDate=&endDate=` retrieves the 20 most recent tweets in the period between `startDate` and `endDate`

Indexing

To speed up the retrieval of statistics that compose these reports, we precompute indexes for the data in the database, so as to avoid content processing at each request.

The tweet density graph index is composed of: *i)* a real-time frequency histogram, maintaining a cache of the last 10 database counts to be displayed in a real-time graph fashion (see Figure 4.9); *ii)* a past frequency histogram, where each bin has a 5 minutes depth and maintains the count of tweets collected in those 5 minutes.

As for the keywords, hashtags and influencers, we maintain a collection of tuples, each one having a 5 minutes depth, where we store the top influencers, top keywords and top hashtags we collected in those 5 minutes.

4.11 Summary

In this Chapter we proposed a methodology for the application of passive crowdsourcing techniques so as to extract relevant content and influential users from social media. The extraction is based on the automatic multimodal analysis of user-generated content, based on the assumption that relevant content is produced by influential users. The automatic pipeline reads in real-time tweets from the Twitter firehose, and classifies them as either relevant or non-relevant for the selected topic. Then, the most relevant keywords and hashtags are extracted from the topic-related content and fed as an input to the pipeline, as new keywords for the keyword-based tweet search. This assures to follow dynamically changes in communication topics.

The proposed work answers to the following research questions:

- **Research question 5:** ***What seed queries can be used to initialize topic-related information retrieval?*** In this Chapter we proposed a probabilistic method that identifies the most relevant keywords and hashtags in a set of topic-related tweets. These terms, if used to initialize the content retrieval procedure, allow us to retrieve a large quantity of topic-related information. Unfortunately, conversations are not always focused on the same sub-topics, and thus relevant keywords and hashtags change day by day. Consequently, in architectures like the one we proposed in this Thesis, which monitor user-generated content in real time, it is necessary to periodically update the set of keywords used to retrieve content. Thus, in our solution we analyze periodically the retrieved relevant content, we extract keywords and hashtags, and we feed them as an input to the crawler, so as to retrieve fresh and updated conversations.

- **Research question 6: *What type of data can be analyzed to improve the accuracy of topic-related information retrieval?*** In this Chapter we proposed a multimodal approach that analyzes both textual and multimedia content in order to assess whether content is topic-related. We show that this is a good approach, as demonstrated in the experimental section, since images may contain hints on the relevance of content, which are not captured by text.

- **Research question 7: *How are influential content producers defined and identified?*** In this Chapter we proposed a metrics for the identification of influential users, which takes into account several factors such as the activity, originality and communicativeness of a user. In the experimental section we show that this metrics allows us to retrieve a large number of relevant and influential users for the selected topic.

- **Research question 8: *What is the impact of considering content producers' influence level on the accuracy of topic-related information retrieval?*** In the experimental section we show that the retrieved influential users produce a large quantity of topic-related information, which is not retrieved by using other (baseline) influence metrics.

Chapter 5

Optimizing passive crowdsourcing: Detection of video ancestry relationships

In this Chapter, we introduce a second use case for the passive crowdsourcing application. Here, we exploit the multimedia content users publish on the YouTube social media to detect any ancestry relationships between multimedia files.

User-generated content is easily distributed on the Web, and it is often replicated in more copies which are spread through different channels. Duplications sometimes involve modifications of the content. For instance, images and videos may be modified by transformations such as color corrections, insertion of artifacts, compressions, scaling and rotations, and then republished. This generates multiple copies of the same content which are similar (but not necessarily identical).

Sometimes it is useful to track down all the copies of a given multimedia content, for instance in case of copyright enforcement, in which the copied content was subjected to copyright laws. The family of problems in which we need to detect all the copies of a given document cohabiting on the Web is named NDDR, i.e., Near Duplicate Detection and Recognition.

A more ambitious task is the one of identifying among a set of multimedia documents the ones that are original and the structure of generation of each duplicate. That is, given a set of near duplicates, we would like to trace the history of transformations that generated the duplicates.

The evolution process of images and videos is similar to a *phylogenetic tree*, or *phylogeny*, i.e., the branching process whereby populations are altered over time and may split into separate branches, hybridize together or terminate by extinction. In case of video content: i) a branch corresponds to the duplication of some parts of a video into another video; ii) a hybrid is the product obtained by mixing clips deriving from multiple videos; and iii) an extinction is the absence of further duplications of a given clip.

We would like to exploit the information about similarities among the individuals (i.e., videos in a video collection) to identify the evolving history of the original multimedia content.

Previous works [Dias et al., 2010] [Dias et al., 2011] deal with the problem of identifying the root in a collection of near duplicates (either images or videos). However, this process is limited to a set of documents in which the root is unique and all the other documents duplicate its content. This is obviously a very special case in which the phylogeny is a tree, i.e., a structure in which the content of each child is a duplicate of the content of a single parent, and does not replicate the content of any other document. However, cases in which a document includes content which was duplicated from two or more other documents are very diffused. Consider for instance the creation of news video summaries: they are usually composed of many clips which may be duplicated from many sources. This suggests that a video may inherit content from several videos, and thus many roots (i.e., videos whose content is entirely original) may be present.

In this Chapter we propose a method for constructing the video phylogeny of a set of near duplicate videos, without any limitation on number of roots and graph structure. We start from a simple matching phase whose output is a preliminary phylogeny. Since the algorithms used for segmenting videos and performing the matching are subjected to uncertainties, the produced phylogeny will contain noise, i.e., some non-existing similarity relationship will be detected and some existing relationship will not. Thus, we run an adapted version of the well-known DFS algorithm so as to remove noise and restore the real phylogeny.

5.1 Video similarity graph computation

Suppose a user queries a video sharing platform (e.g., YouTube) requiring all the videos related to a specific topic. As a running example consider the case in which we require all the videos whose topic is the resignation of pope Benedict XVI in 2012. Let $\mathcal{V}$ denote the retrieved collection of videos.

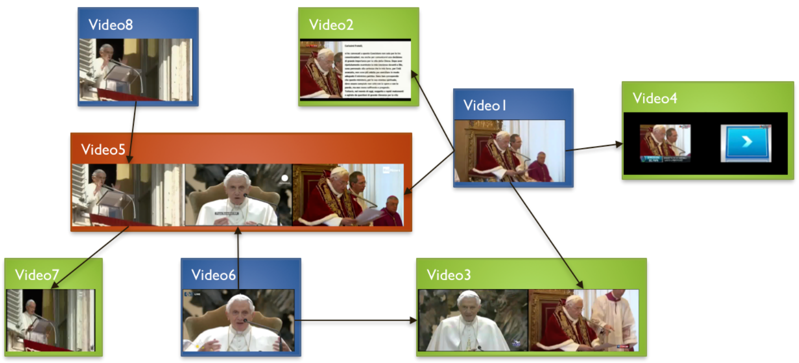


Figure 5.1: An example of a collection of videos in which content is replicated. Each time the original content is copied, it could be subjected to transformations (e.g.: clipping, rotation, rescaling)

Each video $v \in \mathcal{V}$ could include either *original content* (i.e., content that was not duplicated from any other document in $\mathcal{V}$) or *replicated content* (i.e., content that was copied from other documents in $\mathcal{V}$). This usually happens in presence of viral content, which is copied in a huge amount of multimedia files and then spread over the Web in new versions. In our running example, as represented in Figure 5.1, many clips are duplicated in more videos, since either TV shows or users on the Web copied part of the content of the original videos and split it in multiple versions. In the represented example, the videos colored in blue contain only original content, while all the others contain replicated content. The arrows illustrate the flow of duplications that created the copies.

Figure 5.2 illustrates a possible duplication process for a sample of 6 videos. Notice that each duplicate may contain either an exact copy the original content or a modified version of it. These modifications are usually caused by superimposition of artifacts (logos, text) or by transformations (e.g., cropping, scaling, color corrections).

In the following, we propose a method for producing a similarity graph for the videos $v \in \mathcal{V}$, i.e., a graph in which each node is a video v in the video collection and each edge (v_i, v_j) specifies the degree of similarity of v_i and v_j.

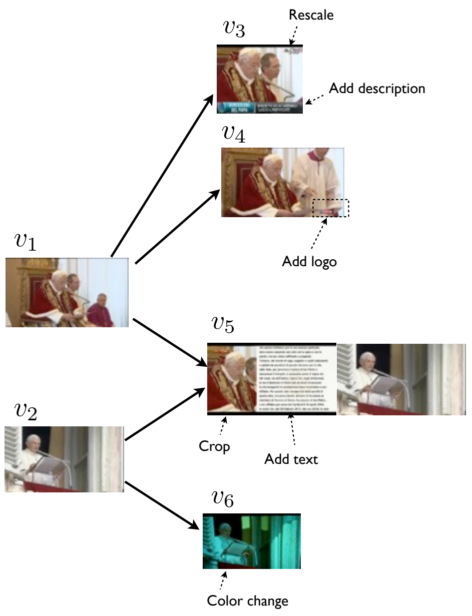


Figure 5.2: Video similarity graph on a real dataset. Notice that the duplicated clips are modified and then combined together in new videos

5.1.1 Similarity graph: definition

In this Section, a definition of video similarity graph is provided.

Let $\mathcal{V}$ be a collection of videos. Moreover, suppose that each video $v \in \mathcal{V}$ is composed of a set of video segments $\mathcal{S}_v = \{s_1, \ldots, s_N\}$, i.e., portions of video which shoot the same scene in their entire duration.

We select two video segments: s_k is a segment in $\mathcal{S}_{v_i}$ and s_l is a segment in $\mathcal{S}_{v_j}$. These segments may be either duplicates or distinct segments. In order to establish whether s_l is a duplicate of s_k, the similarity degree $\sigma(s_k, s_l)$ can be computed:

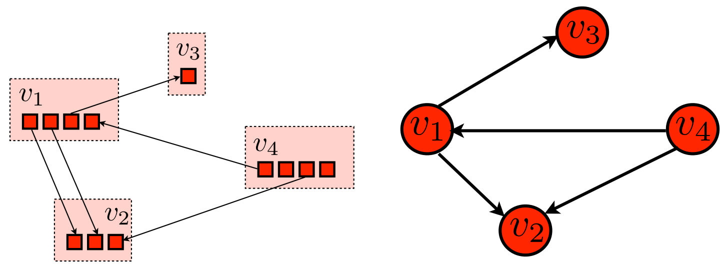
$$\sigma(s_k, s_l) = \begin{cases} M(s_k, s_l) & M(s_k, s_l) \geq \tau \\ 0 & M(s_k, s_l) < \tau \end{cases} \tag{5.1}$$

where $M(s_k, s_l)$ returns the confidence score of the matching between s_k and s_l and τ is a threshold on the confidence value. That is, whenever the confidence value is lower than τ the two segments do not match and their similarity is 0.

A *video similarity graph* is defined as a graph in which: i) each node represents a video; ii) each node contains a number of sub-nodes, representing its segments; iii) edges are traced between pair of segments; iv) an edge $e = (s_k, s_l)$ whose starting node is s_k and whose destination node is s_l is labeled with their similarity degree $\sigma(s_k, s_l)$. An example of video similarity graph is shown in Figure 5.3(a). We call *roots* the videos whose indegree is 0, and *leaves* the videos whose outdegree is 0.

The similarity relationships between videos can be synthesized by analyzing the similarity relationships between segments. For instance, the graph in Figure 5.3(a) suggests that some segments of v_1 are copied in v_2 and v_3 and that some segments of v_4 are copied in v_1 and v_2. The synthesized relationships can be represented in a graph in which segments are not visualized, as shown in Figure 5.3(b). The weight of the edge (v_i, v_j) represents the similarity degree of the two connected videos, computed as:

$$\sigma(v_i, v_j) = \frac{\sum_{s_k \in \mathcal{S}_{v_i}} \sum_{s_l \in \mathcal{S}_{v_j}} \{M(s_k, s_l) > \tau\}}{|\mathcal{S}_{v_i}| \cdot |\mathcal{S}_{v_j}|} \tag{5.2}$$



(a) Segment similarity representation (b) Summarized relationships

Figure 5.3: Video similarity graph

5.1.2 Video Processing Pipeline

We suppose that the video collection $\mathcal{V}$ is available as an input. Several steps are needed to process it and extract the video similarity graph. In this Section we propose a video processing pipeline, where $\mathcal{V}$ is fed as an input and the similarity graph is produced as an output.

General overview

The video processing pipeline is divided into two main phases: the indexing phase and the matching phase.

In the *indexing phase*, we are provided with a collection of videos $\mathcal{V}$ and our aim is to extract keyframes and descriptors from them. These descriptors will be then used as an input for the matching phase. In order to identify keyframes, each video $v \in \mathcal{V}$ is segmented into shots. Then, for each shot one or more keyframes are selected, extracted from the video and processed so as to compute their descriptors.

In the *matching phase*, the descriptors that were previously extracted from the video segments are used to perform the matching between pairs of segments (s_k, s_l) (i.e., by computing the quantity $M(s_k, s_l)$). For each pair s_k, s_l in the collection the similarity $\sigma(s_i, s_j)$ is computed according to Equation 5.1. Finally, the video similarity graph is built, as described in Section 5.1.1.

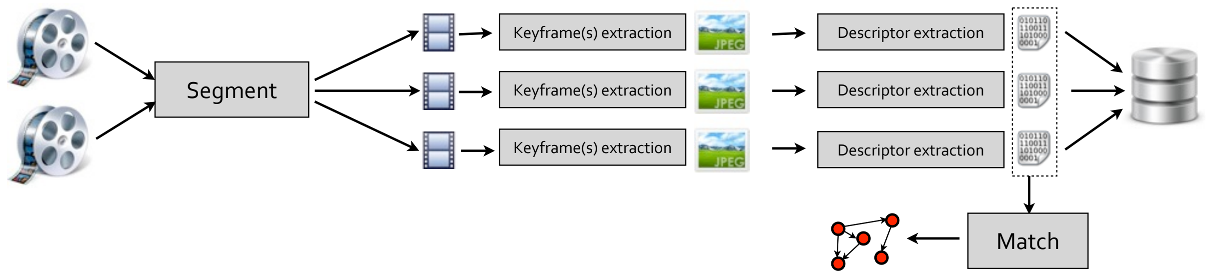


Figure 5.4: Video processing pipeline

Figure 5.5 represents the component diagram of the application. The application structure is divided into two main components, i.e., the `Indexer` component and the `Similarity graph extraction` component. The first one is in charge of indexing the video collection, thus its subcomponents are the `Video segmentation` component, the `Keyframe extractor` component and the `Visual feature extractor` component. The `Indexer` component produced output is then used by the `Similarity graph extraction` component, which is in charge of computing the similarities between the videos in the video collection and returns the video similarity graph.

Finally, the `Browser visualization application` component is the one employed by users for visualizing the video similarity graph. Here, the result is read by the `Similarity graph extraction` component, which checks whether the graph was already indexed by the `Storage` component, and if it is not the case, asks to the `Matcher` component to compute the video similarity graph and then stores the result using the `Storage` component.

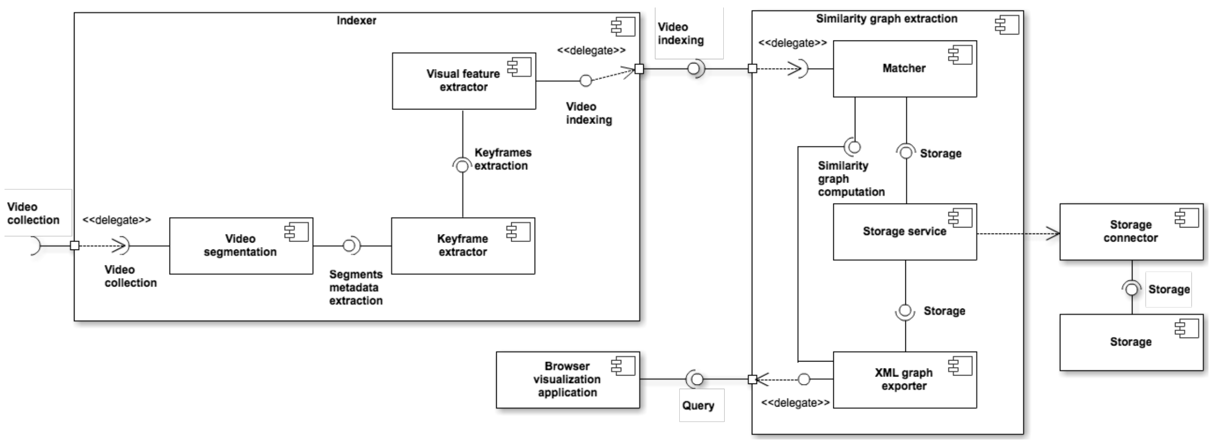


Figure 5.5: Component diagram

Phases

Video segmentation The *video segmentation* is the process of partitioning a video $v \in \mathcal{V}$ into sets of consecutive frames that are homogeneous according to some defined rule.

When a video v is segmented, it is partitioned into *camera takes* (i.e., subsets of consecutive frames that are captured by a single camera from the moment it starts shooting to the moment it stops), *shots* (i.e., subsets of consecutive frames belonging to the same camera take in an edited video) and *scenes* (i.e., a group of continuous shots that are semantically connected) [Vid, 2013].

For each shot one or more keyframes are extracted. The number of extracted keyframes depends on the camera motion in the scene, i.e., the more the camera moves, the more the captured frames content changes, the larger is the number of required frames to represent the shot.

In this work, the video segmentation was performed using either one of two modules: the human segmentation module and the automatic segmentation module.

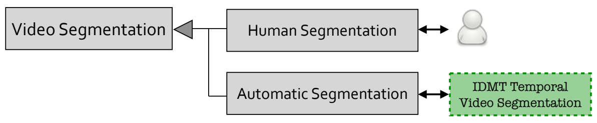


Figure 5.6: Video segmentation components

The *human segmentation module* requires a human to segment the set of videos into shots and extract the related keyframes. This prevents to over-segment (or under-segment) the videos, which may happen when an automatic component is used to perform the segmentation. As a result, a manual segmentation improves the performance, although it cannot be applied in large datasets, since the operation is time-consuming.

On the other hand, the *automatic segmentation module* automatically detects shots, representative keyframes and scenes. In this work we adopted a component called IDMT Temporal Video Segmentation [FRH, 2012], developed by Fraunhofer IDMT. With this component, keyframes are detected depending on changes between adjacent frames in a shot. As stated earlier, this solution may be less accurate than manual segmentation, although it is strongly suggested with large datasets.

Descriptors extraction The descriptor extraction phase extracts a global (a set of local) descriptor(s) for each extracted keyframe. In this work we extract SIFT-based descriptors [Lowe, 1999] and color descriptors [van de Sande et al., 2010].

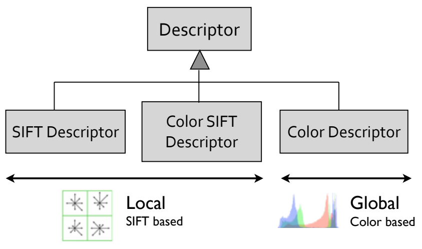


Figure 5.7: Descriptors

The SIFT Descriptor and the Color Descriptor are based on custom implementations (in OpenCV and Java, respectively).

The Color SIFT Descriptor implementation is based on the ColorDescriptor component [RGB, 2012] developed by Koen van de Sande. The derived descriptors increase illumination invariance and discriminative power of descriptors, so that the obtained description is robust to light intensity changes/shifts and to light color changes/shifts [van de Sande et al., 2010].

Matching During the matching phase, the similarity degree between videos and video segments is computed. Let $s_k \in \mathcal{S}_{v_i}$ and $s_l \in \mathcal{S}_{v_j}$ denote two video segments. Their similarity degree is computed as stated in Equation 5.1. The matching is performed:

- *symmetrically* in the case of color descriptors, i.e., if $M(s_k, s_l) \geq \tau$ then s_k matches with s_l and viceversa;
- *asymmetrically* in the case of SIFT-based descriptors, i.e., if $M(s_k, s_l) \geq \tau$ then s_k matches with s_l, but it is not sure that s_l matches with s_k ($M(s_l, s_k)$ is computed so as to verify is this is the case).

In case of SIFT-based descriptors, given two keyframes $\mathcal{K}_k \in s_k$ and $\mathcal{K}_l \in s_l$, the confidence score of their matching is computed as the percentage of the key-points of $\mathcal{K}_k$ that match with the key-points of $\mathcal{K}_l$:

$$M(\mathcal{K}_k, \mathcal{K}_l) = \frac{|\mathrm{KP}(\mathcal{K}_k) \cap \mathrm{KP}(\mathcal{K}_l)|}{|\mathrm{KP}(\mathcal{K}_k)|} \tag{5.3}$$

where $\mathrm{KP}(\mathcal{K})$ is the set of key-points of the keyframe $\mathcal{K}$. Thus, the confidence score of the matching of s_k and s_l is computed as:

$$M(s_k, s_l) = \max_{\mathcal{K}_k \in s_k, \mathcal{K}_l \in s_l} M(\mathcal{K}_k, \mathcal{K}_l) \tag{5.4}$$

In case of color descriptors, given two keyframes $\mathcal{K}_k \in s_k$ and $\mathcal{K}_l \in s_l$, the confidence score of their matching is computed according to the Bhattacharyya distance, as follows:

$$M(\mathcal{K}_k, \mathcal{K}_l) = \sum_{b \in B} \sqrt{H_{\mathcal{K}_k}(b) \cdot H_{\mathcal{K}_l}(b)} \tag{5.5}$$

where $H_{\mathcal{K}_k}$ and $H_{\mathcal{K}_l}$ are the color histograms of the keyframes $\mathcal{K}_k$ and $\mathcal{K}_l$, respectively, and B is the set of bins of the two histograms. Thus, the confidence score of the matching of s_k and s_l is computed as stated in Equation 5.4.

In order to build the video similarity graph, the S segments in the video collection $\mathcal{V}$ are inserted in a unique array (keeping the reference to the videos they belong to). Then, an $S \times S$ matrix is built, where the entry on the i-th row and j-th column represents the similarity degree $\sigma(s_i, s_j)$ between the i-th and j-th segments in the array. This matrix is used as the adjacency matrix of the video similarity graph.

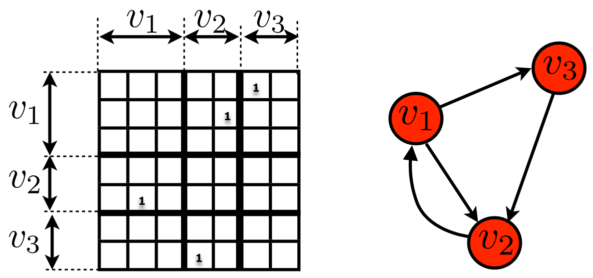


Figure 5.8: Translate a similarity matrix into a graph

5.2 Video phylogeny reconstruction

In this section the algorithms used for removing noise from a corrupted phylogeny are introduced.

5.2.1 Phylogeny structure analysis

A video phylogeny is a graph structure describing the evolution of multimedia content over time. Some videos contain the original content, which is copied, in case modified and reshared over the Web.

The duplication process follows a unique direction, i.e., from past copies to new copies. Indeed, since new copies are created by mixing new content and existing video clips, it never happens that parts of a newer video are copied in an older video. That is, if a video v is created at time $t(v)$ and a (complete or partial) duplicate v_c of v is created at time $t(v_c)$, then it is always true that:

$$t(v_c) > t(v) \tag{5.6}$$

In other words, if the phylogeny contains a path from v to v_c, there does not exist a path that starts from v_c and goes to v, otherwise the constraint in Equation 5.6 would be violated. Consequently, the phylogenetic tree is a directed acyclic graph.

5.2.2 Problems related to phylogeny construction

When a set of videos $\mathcal{V}$ is given as an input, we build a preliminary phylogeny by computing the video similarity graph (see Section 5.1.1).

In an ideal situation, the similarity graph that is returned as the output of our pipeline is a directed acyclic graph. This means that: i) *roots*, i.e., nodes whose indegree is 0, represent those videos whose content is original, i.e., not duplicated from other videos; ii) *leaves*, i.e., nodes whose outdegree is 0, represent those videos whose content is at least partially duplicated (unless the node is both a root and a leaf, meaning that it is disconnected from the graph and its content is original); iii) *paths* represent the content duplication process, i.e., the steps through which the original content was copied in other videos.

However, the algorithms that are used to extract descriptors and perform matching are usually uncertain, meaning that false positives (i.e., edges that are in the graph but that do not exist in reality) and false negatives (i.e., edges that exist in reality but that are not detected) may be introduced.

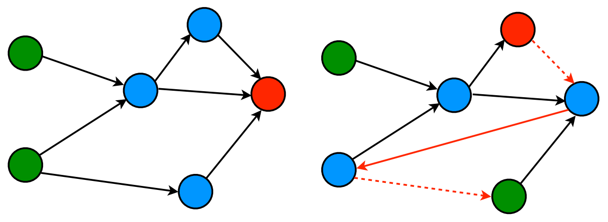

(a) Original graph (b) Corrupted graph

Figure 5.9: Original phylogeny (a) and output of the video processing phase (b)

This results in a graph that does not reflect the real phylogeny of the video collection $\mathcal{V}$:

- when a *false positive* is introduced, the duplication process represented in the graph is enriched with respect to the original one, reporting that the content of a video v_i was copied from the content of a video v_j (which is not true);
- when a *false negative* is introduced, the edge between two videos v_i and v_j is not detected and the real duplication process that was going from a root v_r to a node v is broken in two pieces, from v_r to v_i and from v_j to v. This means that v_j and v_i may be reported as a root and a leaf in the graph, respectively.

As a consequence, new roots may be introduced and the real roots may be incorporated in the graph as either internal or leaf nodes. This means that the information about which videos carry the original content may be corrupted. The same applies for leaves.

Moreover, due to the insertion of false positives, cases in which the new edges violate the constraint in Equation 5.6 are frequent. This means that cycles are introduced in the video similarity graph, and this result in a structure which is not compliant with the video phylogeny features.

An example of phylogeny corruption is shown in Figure 5.9. The dashed red lines and the red line show the introduced false negatives and false positives, respectively. Notice that both roots (in green) and leaves (in red) change between the two graphs. Moreover, a cycle is created, invalidating the phylogeny structure.

5.2.3 DFS algorithm for phylogeny reconstruction

Suppose a phylogeny corrupted by noise is given as an input. Our aim is to reconstruct the original phylogeny by removing cycles and restoring the original roots, leaves and edges.

The Depth-First Search algorithm can be used so as to remove cycles from the graph structure and reconstruct the original phylogeny. The idea behind this procedure is that if the search starts from the roots, whenever an edge that violates the constraint in Equation 5.6 is found, it is deleted. Thus, by following the duplication process directionality (from root to leaves), DFS easily identifies the false positives and eliminates them. To this end, we devised two different approaches for this strategy.

DFS: single iteration

Algorithms 7 and 8 illustrate the procedure used for deleting cycles from the video similarity graph using the DFS algorithm.

The algorithm navigates the edges connecting pairs of *video segments* and deletes them if a cycles between *videos* is detected. Since there may be more than one connected component in the graph, the cycle removal is started from each video $v \in \mathcal{V}$ (line 2 of Algorithm 7), so as to explore the entire graph and detect all the cycles.

For each starting point v, Algorithm 8 is used to navigate the set of *videos* in the graph that can be reached from v. At each step, we check whether the current video v was visited in previous steps (line 1), and if it is the case, a cycle is detected and broken by deleting the last visited edge. Otherwise (line 3) the video is tagged as visited by queuing it in the stack of visited nodes. Then, all the edges e_i that are outgoing from any *segment* $s \in \mathcal{S}_v$ are selected and analyzed (lines 5 and 6): the destination *video* d of each edge e_i is selected and used as starting point for the next step. Eventually, the acyclic graph is returned.

Figure 5.10 shows a run of the algorithm when v_1 is used as start point. At step 1 the two outgoing edges of v_1 are selected and the first one (from v_1 to v_3) is crossed. Then, at step 3 v_3 is analyzed and its outgoing edge (from v_3 to v_4 is crossed). Finally, the outgoing edge of v_4 is selected so as to be crossed, but it brings to v_1 (which was already visited), and thus this last edge is deleted from the graph. Finally, the edge between v_1 and v_2 is crossed at step 5, reaching a leaf and terminating the algorithm. The outcome of this procedure is a phylogeny with v_1 as a root node and v_2, v_4 as leaves.

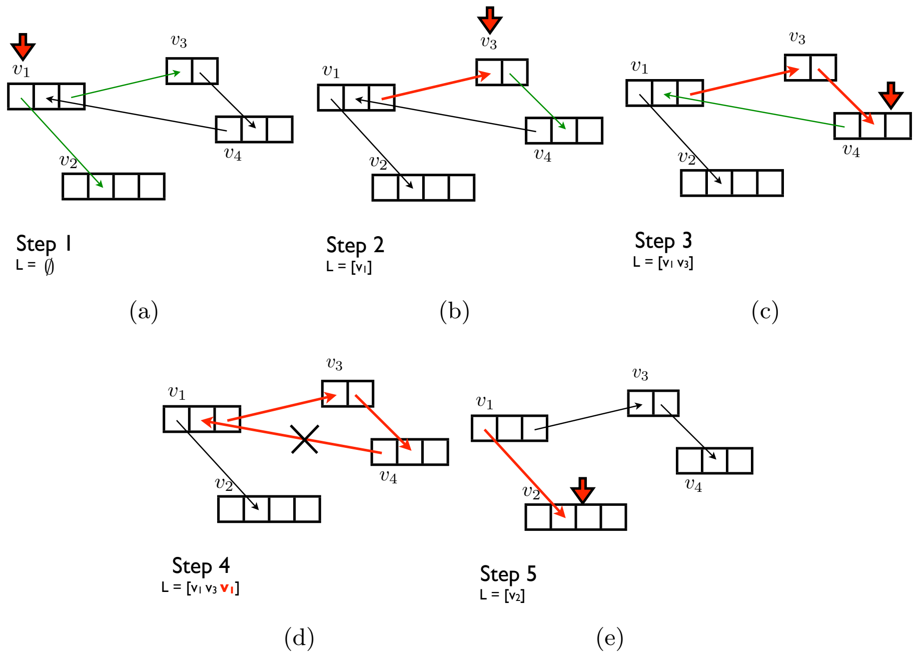


Figure 5.10: Steps of a DFS run for cycle removal

DFS: multiple iterations

The quality of the output provided by of the algorithm proposed in Section 5.2.3 depends on the order with which the videos in $\mathcal{V}$ are used as starting point for the depth-first search. For instance, consider the case in Figure 5.11. If the cycle is entered from different nodes, different edges will be deleted by the algorithm. However, deleting randomly one of the edges in a cycle

Algorithm 7: RemoveCycles uses the DFS algorithm so as to remove cycles from the video similarity graph

```
Input: Corrupted phylogeny $\mathcal{P}^C$
Output: Reconstructed phylogeny $\mathcal{P}$
1. $\mathcal{P} = \mathcal{P}^C$;
2. for each $v \in \mathcal{V}$
3.     $\mathcal{P}$ = DFSCyclesRemoval($\mathcal{P}$, $v$, $\emptyset$, $\emptyset$);
```

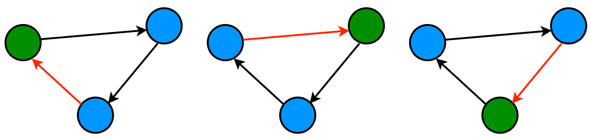


Figure 5.11: Different starting points for the DFS algorithm lead to the deletion of different edges

could bring either to the right solution or to the wrong solution. Thus, the cycle access point cannot be defined randomly.

In the following, we define a method for initializing the algorithm multiple times and then aggregate the results achieved by different iterations. The main idea is that each time the DFS algorithm is executed, it will retrieve a graph in which edges are either existing or not existing. The edges that are reported as existing will receive one vote. If we start DFS multiple times changing the starting nodes at each iteration, the graph exploration will be different each time. Consequently, some iterations will vote some edges as existing, while others will not. We call **strong edges** the ones which are annotated as existing by a large number of DFS iterations, and **weak edges** the ones that receive a small number of votes. Eventually, strong edges will be preferred to weak edges. This makes the procedure more robust, since if the number of iteration is large, most of them will foresee the correct phylogeny direction, and weak edges will correspond to the ones violating the constraint in Equation 5.6.

Algorithm 9 illustrates the procedure. At first, the DFS algorithm is instantiated N times, and the resulting N graphs are collected in an array $\mathcal{P}$ (lines 1-5). In each iteration, the DFS starting nodes are redefined by randomizing the order of the videos $v \in \mathcal{V}$. Consequently, each graph in $\mathcal{P}$ corresponds to an *annotation* made by the DFS algorithm on the presence/absence of the edges in the phylogeny. In order to aggregate the N annotations, for each edge $e = (s_i, s_j)$ the number of received votes is computed (lines 8-10). Finally, we evaluate whether each edge e will be inserted in the final phylogeny. To do so, we start integrating the strong edges and then we move to the weak ones. At each iteration, we select the set E of edges which received exactly a votes (line 13), where a is varying between N and 1. Then, we start integrating the edges $e \in E$ in a new phylogeny whose set of edges is initially empty (line 15). For each edge e we check whether inserting it into the phylogeny generates a cycle (line 16): if this is not the case, e is inserted in the final phylogeny.

Algorithm 8: DFSCyclesRemoval applies the DFS algorithm to a graph G in order to remove cycles. The search starts at v

```
Input: Graph $G$, Current node $v$, Last visited edge $e_l$, List of visited videos $L$
Output: Graph $G$
 1. if $v \in L$ then
 2.     Remove $e_l$ from $G$
 3. else
 4.     Add $v$ to $L$
 5. $E$ = getOutgoingEdges($v$, $G$);
 6. for $e_i \in E$
 7.     $d$ = getDestinationVideo($e_i$);
 8.     $G$ = DFSCyclesRemoval($G$, $d$, $e_i$, $L$);
 9. return $G$;
```

Algorithm 9: N-RemoveCycles applies N times the DFS algorithm to a graph G in order to remove cycles

```
Input: Corrupted phylogeny $\mathcal{P}^C$, Number of iterations $N$
Output: Reconstructed phylogeny $\mathcal{P}$
 1. for $i = 1$ to $N$
 2.     $\mathcal{P}[i] = \mathcal{P}^C$;
 3.     Set $\mathcal{V}_i$ as a random sequence of the videos in $\mathcal{V}$;
 4.     for $v \in \mathcal{V}_i$
 5.         $\mathcal{P}[i] =$ DFSCyclesRemoval($\mathcal{P}[i]$, $v$, $\emptyset$, $\emptyset$);
 6.
 7. Set $\mathcal{S}$ as the collection of all the segments in $\mathcal{V}$;
 8. for $s_i \in \mathcal{S}$
 9.     for $s_j \in \mathcal{S}$
10.         votes[$i$][$j$] = computeEdgeVotes($\mathcal{P}$, $s_i$, $s_j$);
11.
12. $\mathcal{P}_o = (\mathcal{V}, \emptyset)$;
13. for $a = N$ to 1
14.     Set $E = \{(s_i, s_j)\}$ as the set of edges corresponding to the cells votes[$i$][$j$] $= a$;
15.     for $e \in E$
16.         if $\mathcal{P}_o \cup e$ is not cyclic
17.             Add $e$ to $\mathcal{P}_o$;
18. return $\mathcal{P}_o$;
```

5.2.4 Integration in the software architecture

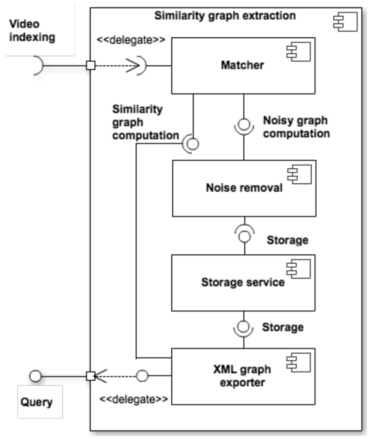


Figure 5.12: Integration of the DFS module in the software architecture

Figure 5.12 shows the integration of the DFS module in the software architecture presented in Section 5.1.2. The `Noise removal` component is the module in charge of applying the DFS algorithm so as to remove noise from the phylogeny. Once the computation ends, the result is stored in memory using the `Storage service` component.

5.2.5 Problems that DFS cannot solve

In this work, we exploit the DFS algorithm so as to remove cycles and delete those edges which violate the constraint in Equation 5.6. However, there are problems which DFS cannot solve.

First of all, let e be a false positive, i.e., an edge that is not present in the real phylogeny, which does not generate a cycle in the corrupted phylogeny. The DFS algorithm will not identify it as a wrong edge, and thus it will be present in the output.

Secondly, the DFS algorithm is used to remove edges from the corrupted graph. However, it happens sometimes that two segments are not recognized as similar in the matching phase.

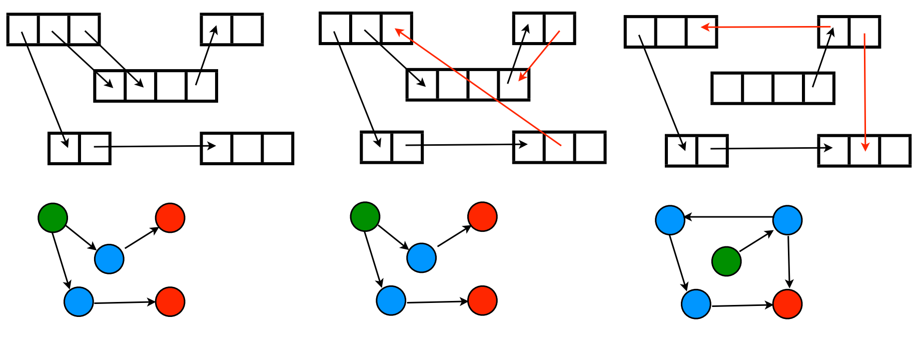

(a) Original (b) Small amount of noise (c) Large amount of noise

Figure 5.13: Problems related to DFS

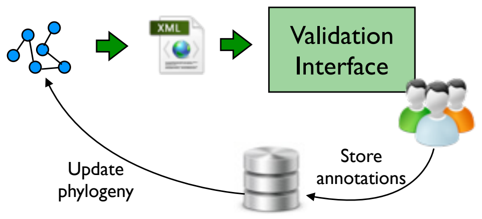


Figure 5.14: Web application architecture for the visualization and validation of results

Thus, false negatives are introduced, i.e., some edges are missing in the output. DFS is not capable of restoring those edges, and thus some relationships will be missing.

In an ideal situation in which a small amount of false positives and false negatives are present, these problems do not impact seriously on the algorithm performance, since errors are introduced at segment similarity level and are globally masked by the video similarity level. However, in all those cases in which the introduced noise is high, the outcome could differ from the real phylogeny, affecting the performance. Nevertheless, we will show how in real cases the performance is still good.

An example is shown in Figure 5.13. Notice that if the amount of noise is limited (e.g., in Figure 5.13(b) an edge is missing, two are introduced), then the phylogeny is perfectly reconstructed, since the introduced edges are recognized as false positives and removed, and the remaining edges replace the false negatives. However, when the amount of noise is larger (e.g., in Figure 5.13(c)), the reconstructed phylogeny differs from the original one: notice that both the root set (in green) and the

5.3 Visualization and validation

In the previous chapters we proposed algorithms for building the video similarity graph and reconstructing the phylogeny. However, a visual representation of the result would be useful to validate the correctness of the reported video similarity relationships. Moreover, since the result could be subjected to uncertainty (as stated in Section 5.2.5), manual annotation could be used to remove false positives and restoring false negatives.

In this Chapter the application used for visualizing and validating the output of the algorithm is presented.

5.3.1 Web application architecture

Since the application is asked to collect annotations from a crowd, it needs to be accessible through the Web. Thus, we built a Web application and deployed it on a server, so that it can be accessed from different workstations.

The architecture is described in Figure 5.14. The phylogeny, which has a graph structure, is fed as an input to the Web application. Firstly, it is translated in an XML document. The XML document reports the structure of each video and the list of all the detected relationships among video segments.

```
<VideoDuplicateDescription>
       <Content>
              [...]
       </Content>
       <Relationship>
              [...]
       </Relationship>
</VideoDuplicateDescription>
```

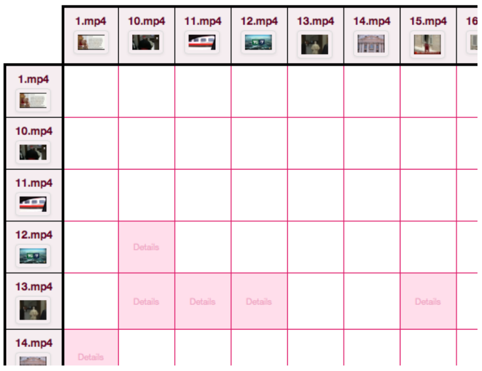
The `Content` tag contains information related to the structure of the video collection. Each child `ContentObject` contains the description of a single video in the video collection (reporting the MIME type of the video, its ID, its name and the location). Each video contains also a `MediaSegments` tag, whose children represent the video segments.

```
<ContentObject MIMEType="videoMIMEType"
                        ID="videoID" MediaLocator="URL">
       <Descriptions ID="videoID" Name="videoName">
              <ItemAnnotations Duration="durationInSeconds"
                     CreationTS="creationTimeStamp"/>
              <MediaSegments>
                     <SceneSegmentDescription ID="segmentID" <!-- a segment -->
                            StartTS="segmentStartTimeStamp"
                            EndTS="segmentEndTimeStamp"/>
              </MediaSegments>
       </Descriptions>
</ContentObject>
```

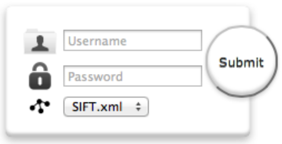
The `Relationship` tag contains the list of all the detected relationships between video segments. Each relationship (listed as `TemporalRelationship`) refers to two video segments (the start node and the destination node, respectively), together with the relationship confidence value.

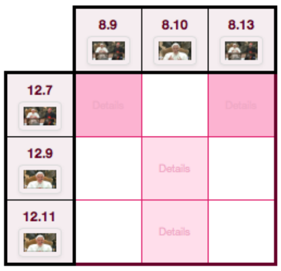
```
<TemporalRelationship refID1="segmentID1" refID2="segmentID2">
       <SegmentRelationshipAnnotation>
              <Confidences>
                     <Value>
                            <ConfidenceValue Value="value"/>
                     </Value>
              </Confidences>
       </SegmentRelationshipAnnotation>
</TemporalRelationship>
```

The interface displays the phylogeny structure described in the XML file. Users can visualize the underlying relationships (together with the confidence value), and if a relationship is recognized to be a false positive, it is possible to mark it as a candidate for the deletion. Each time a



(a) Visualization of videos and relationships





(b) Login form (c) Visualization of video segments and relationships

Figure 5.15: Validation interface

user confirms the relationship or discards it, her annotation is stored in a database. Finally, it is possible to merge all the annotations and update the phylogeny by deleting those relationships that were marked as non existing by most of the users.

5.3.2 Visualization

We suppose the users to be registered in our system.

The entry point of the application is a login form, shown in Figure 5.15(b). Since we supposed that many descriptors can be used for indexing the video content, we suppose here that one graph for each descriptor type is created. This means that many XML files are available, one for each loaded graph. Thus, using the drop-down list, the user selects the XML he is interested in and visualizes the result. In the figure, we are loading the `SIFT.xml` file.

Once the user submits her data, the required graph is visualized (see Figure 5.15(a)). The visualized structure is a matrix in which each row and each column represent a video. A colored matrix cell represents the presence of a relationship between the corresponding pair of videos, while a blank cell states that there doesn't exist any pair of segments in those videos which are similar. By clicking on the row (column) title, a preview of one of the keyframes for the corresponding video is visualized in a popup.

Finally, if the user selects the colored cell corresponding to the videos v_i and v_j, he accesses to a second matrix reporting the relationships between the segments of v_i, v_j (see Figure 5.15(c)). Again, by clicking on the row (column) title the user can visualize a preview for the segment, composed of its keyframes. Notice that here the colored cells have different intensities: the higher is the saturation, the higher is the confidence value.

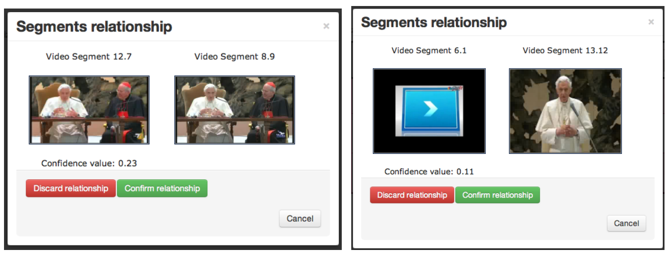


Figure 5.16: Examples of validation popups

5.3.3 Validation

Once the relationships between the segments of two videos v_i, v_j are visualized (Figure 5.15(c)), the user could decide to annotate them as true positives or false positives. To do so, he has to click on one of the colored cells. A popup is visualized, reporting the keyframes of the two segments and the matching score. Two examples are shown in Figure 5.16.

As shown in the Figure, the user could decide to click either on the Confirm button or on the Discard button. In either case, the specified annotation is stored in the database and the relationship is tagged as 'annotated' by that user, so as to prevent the insertion of two (or more) annotations for the relationship from the same user.

5.4 Summary

In this Chapter we proved how an automatic analysis of multimedia content can help in the identification of the original content among a set of duplicate copies. The analysis is based on computer vision techniques, that firstly identify copies in a pool of multimedia object, and then try to infer relationships between them. Each relationship is a copy relationship, indicating that one item is copied from another, and that it may introduce some variation on the content (e.g., color corrections, rescaling).

The proposed work answers to the following research question:

- **Research question 7:** ***How are influential content producers defined and identified?*** Users that are mainly focused on a single topic and that produce original content are probably experts in that topic, and experts produce relevant content. Thus, the identification of duplicated content allows us to discard users that do not usually produce relevant and topic-related content, hence improving the accuracy of the retrieval process.

Chapter 6

Experimental evaluation

In this Chapter we will discuss the experiments that support the claims introduced in Chapters 3, 4 and 5.

6.1 Active crowdsourcing optimization: Fighting uncertainty on structured data

In this section we evaluate the proposed offline and online uncertainty reduction methods on several synthetic and two real datasets, and collect answers through a real crowdsourcing platform.

First, we exploit the synthetic datasets to investigate the impact of uncertainty; the study shows that even small sizes of the dataset ($N < 100$) might lead to an extremely large number of possible orderings. This justifies the need for considering top-K query results, which dramatically reduce the number of orderings by restricting the analysis to the tuples occurring in the first K levels of the tree (Figure 6.5).

Then, we compare the online and offline methods described in Section 3.7 on uncertain datasets characterized by thousands of possible orderings of top-K tuples (Figures 6.7 and 6.10).

6.1.1 Baselines and Oracle algorithm

For completeness, we include in our analysis the comparison with an omniscient algorithm (`Oracle`), plus two simple algorithms used as baselines: `Random` and `Naive`.

Oracle algorithm

The `Oracle` solution is obtained by: *i)* generating all the possible sets of B questions; *ii)* testing them with a hypothetical omniscient and fair worker, who knows the real ordering and works at no cost; *iii)* evaluating the uncertainty reduction derived from each answer, and then *iv)* returning the set providing the highest uncertainty reduction.

First baseline: Random algorithm

The `Random` algorithm returns a sequence of B different questions chosen completely at random among all possible tuple comparisons in $\mathcal{T}_K$, i.e., drawing questions from

$$\mathcal{Q}_{\texttt{all}} = \{t_i \overset{?}{\prec} t_j | t_i, t_j \in \mathcal{T}_K \wedge i < j\}. \tag{6.1}$$

Full name	Parameter	Tested values
Size of dataset	N	100, 500, **1000**, 10000, 100000, 1000000
Number of results	K	1, 3, 5, 7, **10**
Question budget	B	1, **3**, 5, 7, 10
Score probability distribution spread	δ	1e-6, 1e-5, 1e-4, **1e-3**
Uncertainty estimation approach	`appr`	**`avg`**, `opt`, `pess`
Measure of uncertainty	U	entropy (U_H), weighted entropy (U_W), ORA (U_{ORA}), **MPO** (U_{MPO})
Score interval distribution	-	**uniform**, Zipfian
Accuracy of annotators	p	0.8, 0.9, **1**

Table 6.1: Operating parameters (defaults in bold)

Second baseline: Naive algorithm

The `Naive` algorithm avoids irrelevant questions by returning a sequence of B different questions chosen randomly from $\mathcal{Q}_K$ (see Equation (3.24)), i.e., from all the possible comparisons between tuples in $\mathcal{T}_K$ *that have overlapping pdf's.*

Other baselines

Two other baselines were considered: these are relaxed versions of the `Naive` and `Random` algorithms, in which the set of questions are defined, respectively, as:

$$\mathcal{Q}^N = \{t_i \overset{?}{\prec} t_j | t_i, t_j \in \mathcal{T} \wedge \texttt{overlap}(f_i, f_j) \wedge i < j\}$$

and

$$\mathcal{Q}^N_{\texttt{all}} = \{t_i \overset{?}{\prec} t_j | t_i, t_j \in \mathcal{T} \wedge i < j\}$$

However, the number of questions in $\mathcal{Q}^N$ and $\mathcal{Q}^N_{\texttt{all}}$ is huge, since every pair of objects in the complete tree $\mathcal{T}$ is used to define a question. Thus, it is very difficult for the algorithm to select randomly a relevant question (i.e., a question that, if asked, reduces at least a bit of uncertainty), and as a result, these baselines achieve a null uncertainty reduction.

Consequently, for fairness we decided to drop these baselines from our experiments.

6.1.2 Datasets

The experimental phase exploits one synthetic and two real datasets.

Synthetic datasets

The synthetic datasets consist of collections of tuples with uncertain scores. The score of each tuple $s(t_i)$ is uniformly distributed within the interval $[l_i, u_i]$.

We used $\delta = u_i - l_i$ as a parameter to tune the level of uncertainty: the larger is δ, the larger is the uncertainty on the score distribution.

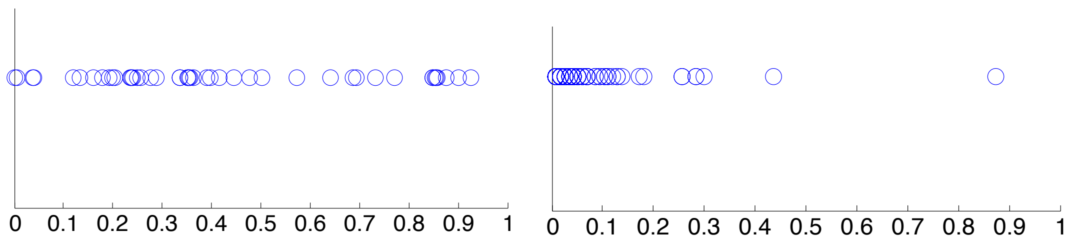
For each tuple, the value l_i is sampled at random in the interval $[0, 1-\delta]$ from either a uniform (Figure 6.1(a)) or a Zipfian (Figure 6.1(b)) distribution, such that $0 \le l_i < u_i \le 1$.

For each configuration of the parameters, we generate 10 instances in order to compute the average performance.

The real ordering ω_r of the tuples in $\mathcal{T}$ is simulated by sampling, for each tuple, a value of the score from the corresponding pdf, and sorting tuples in decreasing value of score. Such an ordering corresponds to one of the paths of the tree of possible orderings $\mathcal{T}$.

YouTube dataset

Real datasets are often characterized by tuples whose score uncertainty cannot be represented with a uniform distribution.



(a) Uniform distribution (b) Zipfian distribution

Figure 6.1: Uniform distribution (a) and Zipfian distribution (b) for a set of 500 objects. Notice how, while in the first case the objects are equally distributed in the entire score space, using the Zipfian distribution the number of objects with high score is reduced

Event	**Event date**	**Upload date**
Curiosity landing on Mars	6-aug-12 (1st)	6-aug-12 (1st)
King Richard III body found	12-sep-12 (2nd)	25-oct-12 (3rd)
Felix Baumgartner freefall	14-oct-12 (3rd)	15-oct-12 (2nd)
Obama election	6-nov-12 (4th)	7-nov-12 (4th)
Palestine entering UN	29-nov-12 (5th)	19-dec-12 (6th)
S.H. school shooting	14-dec-12 (6th)	15-dec-12 (5th)
Los Roques lost airplane	4-jan-13 (7th)	7-jan-13 (7th)
Solomon Islands tsunami	6-feb-13 (8th)	22-feb-13 (9th)
Pope Benedict XVI resignation	11-feb-13 (9th)	11-feb-13 (8th)
Città della Scienza fire	4-mar-13 (10th)	10-mar-13 (10th)

Table 6.2: Event occurrence date and upload date of one set of videos in the YouTube dataset.

As a concrete example, we considered the case of videos downloaded from YouTube and their associated upload timestamps. The upload time is an uncertain indication of the occurrence time of the event captured by a video, since, given a YouTube video, it is not sure whether the user uploaded it as soon it was shot or not.

Therefore, we estimated the distribution $f_{\texttt{offset}}$ describing the temporal offset between the upload time and the event time from a training dataset. The dataset was obtained by downloading 3000 YouTube videos in response to keyword queries referring to events happened in October-November 2012, either unexpected (e.g., Canada earthquake) or announced (e.g., Obama election, Hurricane Sandy in New Jersey, Felix Baumgartner's freefall). The events that were taken into account in this fase are enlisted in Table 6.2.

At first, we manually processed this dataset so as to remove the irrelevant videos returned in response to the keyword queries, i.e., video that, although being part of the result set proposed by YouTube, were not related to the events we were considering. After preprocessing, the training dataset contained 939 videos, whose timestamps were used to estimate $f_{\texttt{offset}}$. In order to do this, we shifted the timestamps of the videos by setting the origin ($t = 0$) to the actual occurrence time of the event, i.e., we moved each timestamp T of the event happened at time T_e to the timestamp $T - T_e$.

Figure 6.2 shows the estimated cumulative density function (crosses), as well as the parametric model obtained by fitting a piecewise polynomial with degree $n = 3$ and 12 intervals (solid line). This approximation is needed to compute the probabilities of the orderings using the algorithm in [Li and Deshpande, 2010], as discussed in Section 3.3.1.

Then, we considered a separate test dataset, which contains 10 sets of $N = 10$ videos each. Within each set, each video refers to a separate event. As an example, Table 6.2 reports the video upload date for one of the sets used in the experiments and the corresponding event date. For each set of videos it is possible to build a tree of possible orderings, by assigning to each video an uncertain occurrence time, modeled with a non-uniform distribution $f_{\texttt{offset}}$ centered in the video upload time. On average, each tree of possible orderings contains approximately

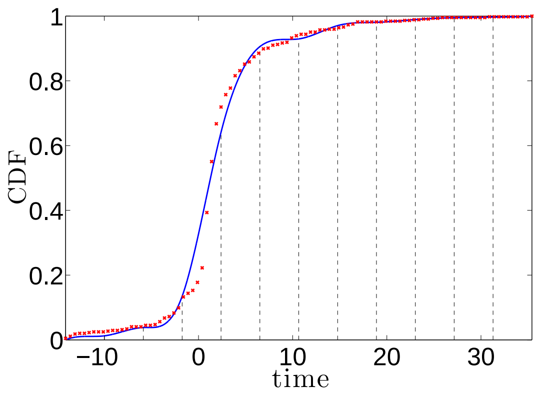


Figure 6.2: CDF estimate for the YouTube training dataset

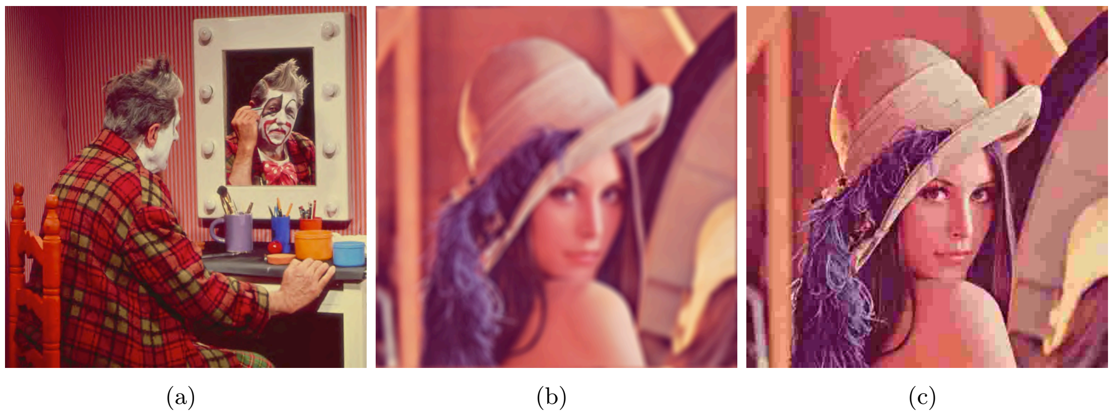


Figure 6.3: A sample of images from the IVC dataset. Some of the images are distortion-free (a), while others are corrupted by blurring (b), JPEG compression (c) or other types of distortion

450 orderings. Note that, in this case, $K = N$, since the ordering is based on time rather than on the relevance score computed with respect to a user query.

The test dataset was used to evaluate the proposed algorithms, with the goal of measuring the number of questions needed to determine the correct ordering ω_r of events in each set. Each question asks the worker to indicate the correct temporal ordering of two events.

Image quality dataset

The second real dataset consists of a collection of images affected by different types of distortion (e.g., blur, Gaussian noise, JPEG compression, etc.) extracted from the IVC dataset [Le Callet and Autrusseau, 2005]. A sample is shown in Figure 6.3.

Each image comes with a Mean Opinion Score (MOS) value (a number in the range $[0, 100]$), which can be used to determine the real ordering ω_r of the images based on their quality. MOS values are difficult to obtain, since they are the result of the aggregation of scores provided by individuals that take part in time-consuming subjective evaluation campaigns. For this reason, objective image quality metrics have been proposed in the literature to automatically assign quality scores. For example, the SSIM metrics introduced in [Wang et al., 2004] evaluates the difference between an image I_d and its original (distortion-free) version I_o, and produces as output an objective quality metrics for I_d.

Due to the different kinds of distortion affecting digital images and the complex task of modeling visual perception, objective metrics (including SSIM) provide an approximate (uncertain) indication of image quality. In [Wang et al., 2004], a mapping between the objective/subjective

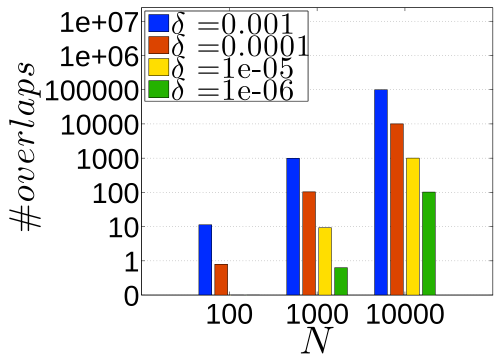


Figure 6.4: Number of overlaps in $\mathcal{T}$ as dataset size N and pdf's spread δ vary

scores is provided, showing that a single SSIM value may correspond to a range of MOS values, comprised on average within an interval of 10 MOS units. This finding allows modeling uncertainty in image quality scores with a uniform probability distribution f_q centered on the SSIM value, with a spread δ of 10 MOS units.

We considered 10 different sets of $N = 15$ images each, with the goal of determining the top-K ($K = 3$) images in each set based on their quality. The proposed methods were used to determine which questions to ask, i.e., which pairs of images need to be compared to sort images according to their visual quality.

This dataset is characterized by a very high level of uncertainty due to difficulties in devising objective quality metrics mimicking the human visual system. Increasing N would lead to compare pairs of images with too similar a quality, thus being adversely affected by the subjective nature of the task.

6.1.3 Evaluation measures

In the experiments, we assess performance of the various algorithms by comparing the average distance from the real ordering in the obtained tree of possible orderings. Note that, if uncertainty is measured as a distance from a representative ordering (such as ORA or MPO), which in turn is a probabilistic proxy for the real ordering, then minimizing the residual uncertainty indeed amounts to minimizing an expectation of the distance from the real ordering.

The average distance $D(\omega_r, \mathcal{T}_K)$ between the real ordering ω_r and the orderings in the tree $\mathcal{T}_K$ is computed as follows:

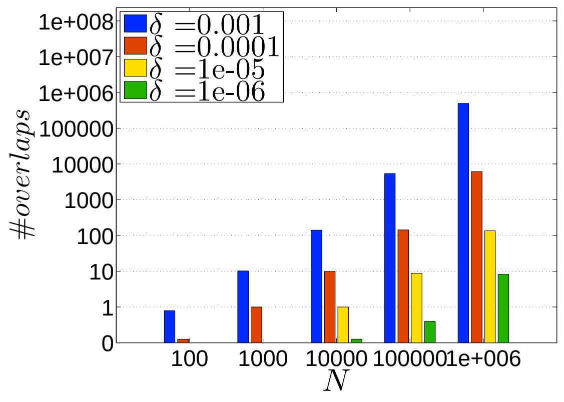
$$D(\omega_r, \mathcal{T}_K) = \sum_{\omega \in \mathcal{T}_K} \Pr(\omega) d(\omega_r, \omega), \tag{6.2}$$

where $d(\omega_r, \omega)$ is a weighted Kendall-Tau distance (see (3.4)).

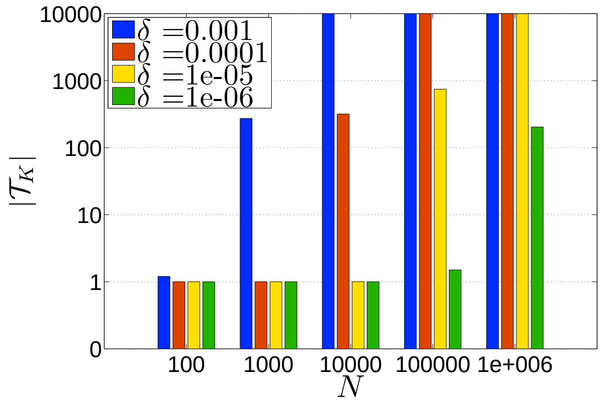
6.1.4 Uncertainty and problem size

We start by examining the impact of uncertainty on full datasets containing all the N tuples. Note that the amount of uncertainty depends on the combination of N and δ (i.e., the spread of the score distribution). Indeed, given a fixed value of δ, the number of overlaps among the pdf's increases with N, since the score distributions are more densely spaced in the $[0, 1]$ interval. Figure 6.4 shows this phenomenon for limited sizes of dataset N.

The size of the problem, measured in terms of the number of orderings in $\mathcal{T}$, grows exponentially in both N and δ. Thus, even with relatively small values of N, the number of orderings quickly becomes intractable when increasing δ. For example, when $N = 100$ and $\delta = 0.001$, there are 12 overlapping pairs of tuples (see Figure 6.4), which generate as many as $|\mathcal{T}| = 4000$



(a) # overlaps at depth K



(b) # orderings at depth K

Figure 6.5: Number of overlaps and orderings in $\mathcal{T}_K$ (K=10) as dataset size N and pdf's spread δ vary

possible orderings. Note that increasing N to just 110 tuples makes the number of orderings grow by an order of magnitude.

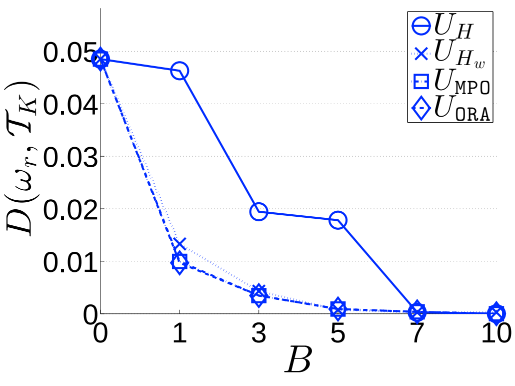
However, users are mostly interested in the top-K results only. Indeed, regardless of N, we observed that the number of tuples in the first K levels of $\mathcal{T}$ is only slightly larger than K, growing slowly when increasing δ. Figure 6.5 shows the impact of uncertainty when one focuses on $\mathcal{T}_K$, for $K = 10$ and several different values of N and δ. Both the number of overlaps of tuples and the number of orderings grow with N, although in a much more tractable manner: for $N = 100$ and $\delta = 0.001$ the average number of orderings is $|\mathcal{T}_K| = 1.2$, while for $N = 1000$ and $\delta = 0.001$ the average number of overlaps is 10, leading to $|\mathcal{T}_K| = 273$. A similar problem size is obtained, e.g., when $N = 10^6$ and $\delta = 10^{-6}$ ($|\mathcal{T}_K| = 203$).

Although there are combinations of the parameters N and δ for which the number of orderings is exceedingly large, they correspond to cases of little practical interest. Indeed, in such cases, the pdf's of the tuples in the first K levels are nearly all overlapping with each other, indicating an extremely large amount of uncertainty in the data. In those cases, very little is known about the ordering among tuples, and human intervention would not be useful in reducing uncertainty, since in those circumstances it would be quite impossible to identify the real ordering. Thus, other means to reduce uncertainty (e.g., aggregating/fusing data from additional sources) should be performed beforehand [Zhang et al., 2006].

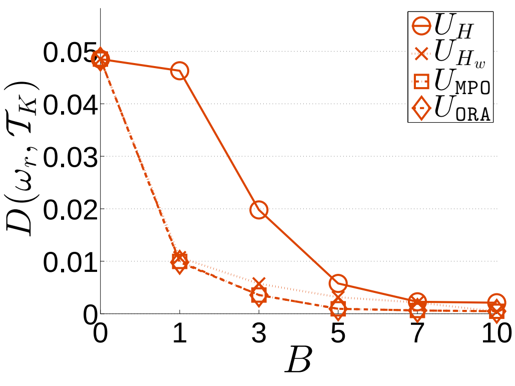
6.1.5 Performance with different uncertainty measures

Figure 6.6 shows the performance of `T1−on` and `C−off` with the four different uncertainty measures (U_H, U_W, U_{MPO}, and U_{ORA}). In particular, while Figures 6.6(a) and 6.6(b) show the results obtained when the weights $\pi(\cdot,\cdot)$ used for the extended weighted Kendall-Tau distance in Equation (3.4) vary with a logarithmic trend (adapted from [Kumar and Vassilvitskii, 2010] to the top-K context), as shown in Equation (3.5). On the other hand, Figures 6.6(c) and 6.6(d) show the results with a linear trend, as shown in Equation (3.6).

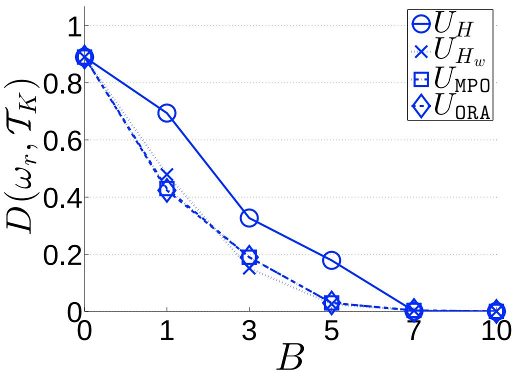
The choice of $\pi(\cdot,\cdot)$ does not change the relative strength of the various algorithms: the results obtained with entropy (U_H) are consistently worse than with all the other measures, which make the algorithms converge to the real ordering much more slowly. Indeed, U_H only takes into account the probabilities of the orderings, thus neglecting the structure of the tree of possible orderings and inadequately quantifying its uncertainty. For instance, with U_H, a tree of possible orderings with branching only at level $K-1$ may easily be considered more uncertain than a tree of possible orderings with branching close to the root.



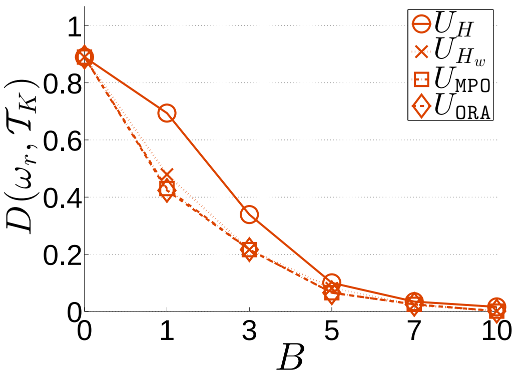
(a) T1−on, weights as in (3.5)



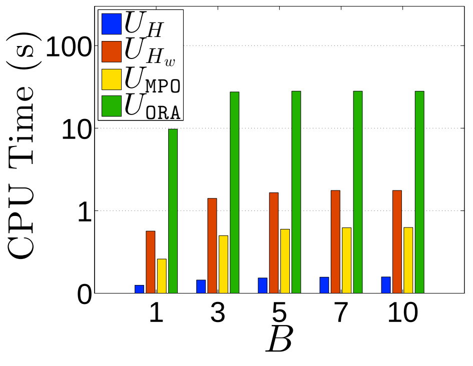
(b) C−off, weights as in (3.5)



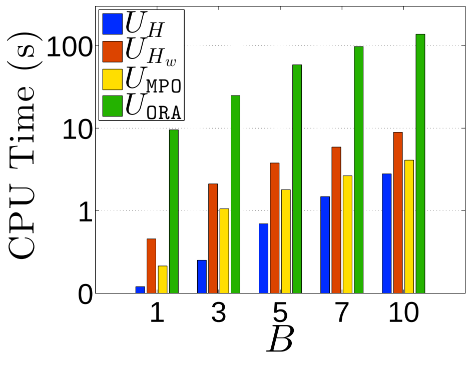
(c) T1−on, weights as in (3.6)



(d) C−off, weights as in (3.6)



(e) T1−on, CPU time



(f) C−off, CPU time

Figure 6.6: Performance of different measures when the T1−on and C−off algorithms are used

The three other measures achieve better performance (Figures 6.6(a)–6.6(c)–6.6(b)–6.6(d)), since they all consider the tree structure in the evaluation of the uncertainty: trees that contain less orderings and/or whose branching happens in the lowest levels are considered as more certain. Although they all achieve similar performance, U_{MPO} requires the lowest CPU time (Figures 6.6(e) and 6.6(f)); thus, we adopt U_{MPO} as the default measure.

Moreover, since the choice of $\pi(\cdot,\cdot)$ is orthogonal to the choice of algorithm, for fairness we use Equation (3.6), which is less convenient in terms of performance (i.e., it causes the algorithms to converge more slowly to the real ordering).

Note that Figures 6.6(e) and 6.6(f) do not include the time needed to compute the tree of possible orderings, which is the same for all the measures and algorithms; full execution times are shown in Figure 6.10(b), along with various optimizations.

6.1.6 Comparing the methods on synthetic data

In this section we compare the effectiveness of the proposed offline and online methods in reducing the number of possible orderings, using the `Random` and `Naive` algorithms as baselines. We adopt the default values of the parameters indicated in Table 6.1, unless stated otherwise.

In this Section, we assume that each crowd worker's answers $ans(q)$ always reflect the real ordering of the tuples. Thus, we prune from the tree $\mathcal{T}_K$ only the paths that do not agree with the relative order of objects in the real ordering ω_r.

Number of questions B

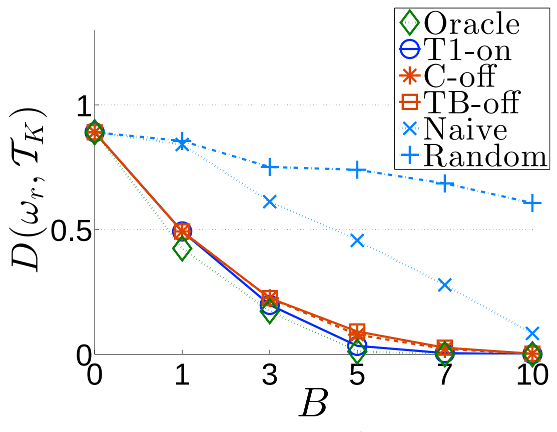
Increasing the number of questions B to ask reduces the uncertainty while converging to the underlying real ordering, as shown in Figure 6.7(a). This is because the collected answers allow discarding incompatible orderings from the tree of all possible orderings.

The `T1−on` algorithm achieves nearly the same performance as the `Oracle` solution, both in terms of uncertainty reduction and distance from the real ordering (Figure 6.7(a)). In particular, both converge to the real ordering with $B = 10$ questions, when considering the default parameters. The offline algorithms, namely `TB−off` and `C−off`, obtain the same result as the `T1−on` strategy when $B = 1$ (since in this case the strategies coincide) and eliminate all uncertainty when $B = 10$. However, since none of the answers are available when choosing the questions, `TB−off` and `C−off` are outperformed by `T1−on`, which has knowledge of previous answers. `C−off` performs better than `TB−off`, because at each step a new question is selected by taking into account the past choices. As for the baselines, selecting questions at random (`Random`) yields very poor results. With respect to the simple `Naive` heuristics, the online algorithm attains the same performance with 50% to 80% fewer questions.

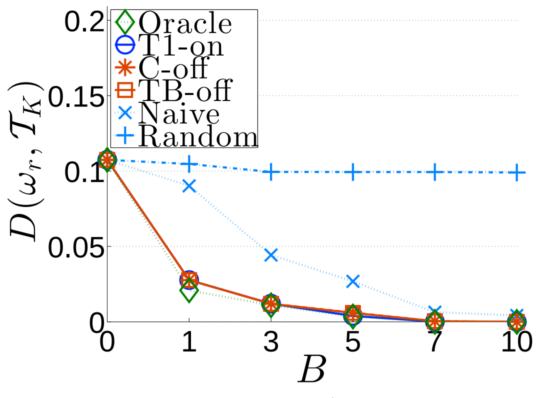
Similar results are shown in Figure 6.9(a), where the uncertainty degree is higher. Here, `Incr − Hyb` and `Incr − On` algorithms are shown too. It is shown how they have lower performance with respect to non-naive algorithms that require the full materialization of the tree. However, these algorithms perform better than baselines that choose questions at random.

Figure 6.7(a) does not include the two algorithms based on A^*, namely `A*−on` and `A*−off`. This is due to their high computational complexity, which makes them impractical for a problem size using the default parameters. Hence, we repeated the same experiment with $N = 500$ and $K = 7$, whose result is visible in Figure 6.8. In this case, we observed that `T1−on` achieved nearly the same performance as `A*−on`, and `C−off` did the same with `A*−off` (in both cases less than 1% difference in the required budget). Therefore, `T1−on` and `C−off` attain the best tradeoff between execution time and result quality.

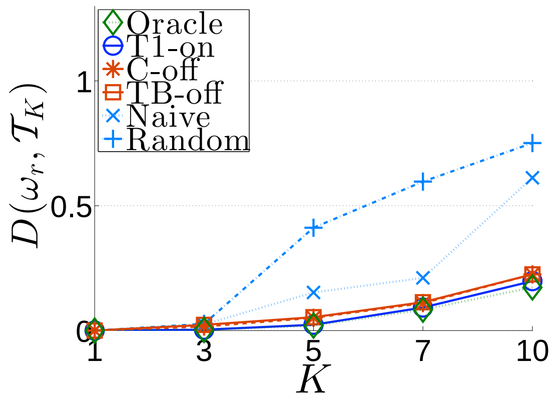
Figure 6.7(a) does not include the algorithm based on the decision trees, namely `DT`, since its results proved to be worse than the other algorithms (about 10% worse). This happens because the choice of a path in the decision tree (that consequently brings to the choice of a set of questions to be asked to the crowd) depends on both questions and answers. Thus, every



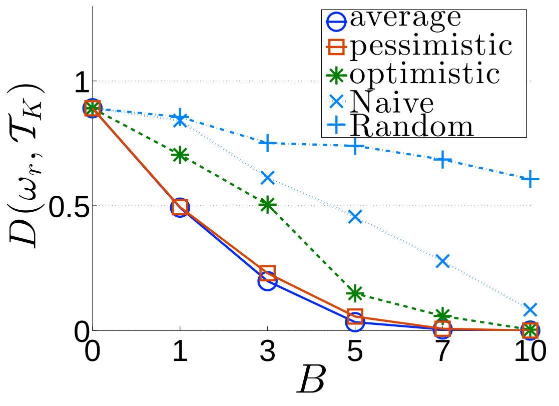
(a) Uniform, N=10^3, K=10



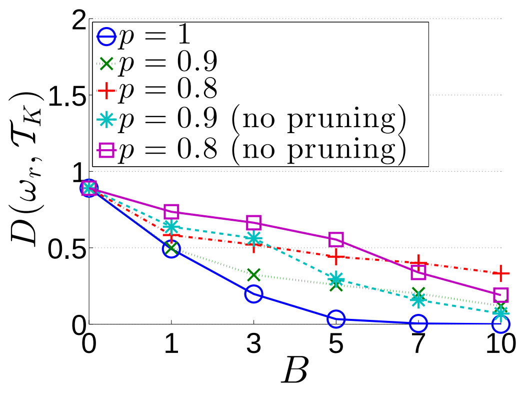
(b) Zipf's, N=10^4, K=10



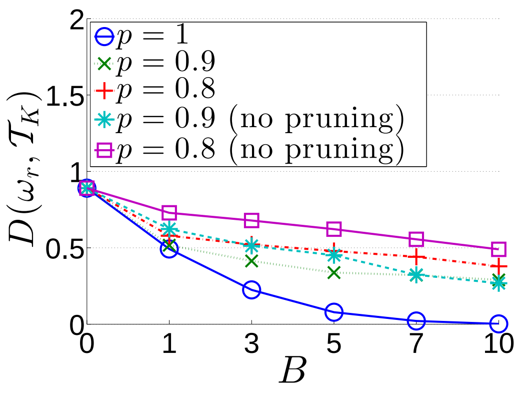
(c) Uniform, N=10^3, B=3



(d) `T1−on` wrt. approaches



(e) `T1−on` wrt. accuracy



(f) `C−off` wrt. accuracy

Figure 6.7: Distances to real ordering as B, K, approaches and worker's accuracy vary ($\delta = 0.001$)

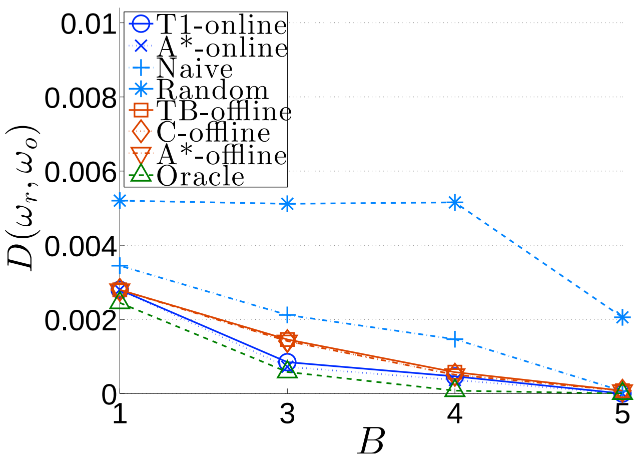


Figure 6.8: Distances to real ordering as B varies ($N = 500$, $\delta = 0.001$, $K = 7$)

time we choose a path, we base our choice on the assumption that users will always answer to questions with the associated most probable answer. This is obviously not true in all cases, and thus, when users' answers deviate from the expected behavior, the best path may not be the one that we selected. In order to adjust this problem we should modify the choice every time the user returns us an answer, but since this means to transform an offline algorithm to an online one, and `DT` in an online version equals to `T1−on`, we did not include the results in the analysis.

Figure 6.7(b) shows the results obtained when $N = 10^4$ tuples are sampled according to Zipf's distribution and $K = 10$. The results are similar to those in Figure 6.7(a). Yet, `T1−on` outperforms `Naive` by an even larger margin when B is small. The `Random` algorithm reaches very poor results, as it is more likely to select irrelevant questions.

Number of results K

Increasing the number of results K increases the uncertainty, as a larger number of possible orderings can be generated. Indeed, Figure 6.7(c) shows that, for a fixed budget of $B = 3$ questions, the distance from the real ordering increases when increasing K. The differences between algorithms become virtually more significant, due to the increase in the number of questions that can be potentially selected, but their difference is smoothed out by the use of the weighted Kendall-Tau distance.

A similar behavior can be seen in Figure 6.9(b), where the uncertainty degree is larger. Here, the difference between algorithms is more visible, due to the larger number of questions needed to remove uncertainty.

Uncertainty reduction estimation approach

Figure 6.7(d) compares the average, optimistic, and pessimistic approaches for the `T1−on` algorithm. The average and pessimistic approaches give the best results and achieve good uncertainty reduction even with very small budgets.

Conversely, the optimistic approach performs poorly, as it expects workers to always answer in the most convenient way, i.e., with the answer that guarantees the highest uncertainty reduction. However, the answer provided by a worker could differ from the expected one; when this happens, this approach essentially amounts to randomly selecting pertinent questions (as the `Naive` algorithm).

A similar behavior can be seen in Figure 6.9(c), where the uncertainty degree is larger.

NDCG metrics

Figure 6.9(d) shows the performance of the proposed algorithms when the NDCG metrics [Järvelin and Kekäläinen, 2002] is used to measure the quality of the output. Since NDCG measures the quality of a ranking, the performance of the algorithms increase while B increases. This is due to the fact that the larger is the number of posed questions, the lower is the uncertainty in the tree.

Number of questions in a batch n

Figure 6.9(e) shows the performance of the $\texttt{Incr} - \texttt{Hyb}$ algorithm when the number of questions n in each batch varies. The results show that the larger the number of questions in a batch, the higher the performance, since larger n require to build a larger part of $\mathcal{T}_K$, thus bringing to a larger question choice.

Accuracy of the workers

Figures 6.7(e) and 6.7(f) show the distance from the real ordering when a worker's accuracy p varies for $\texttt{T1}-\texttt{on}$ and $\texttt{C}-\texttt{off}$. When $p = 1$ the worker always answers correctly, while when $p = 0.5$ answers are random.

For $p < 1$, we compare the effect of pruning the tree of possible orderings as in Section 3.6.1 vs. keeping the entire tree of possible orderings and adjusting its probabilities as in Section 3.8 ("no pruning"). For low p values, pruning is likely to remove the correct ordering, especially when many answers are collected (B high). Conversely, "no pruning" always keeps the entire tree of possible orderings, which may be overly cautious for high p and low B.

It turns out that pruning is more effective when B is low ($B < 5$ with default values), while "no pruning" is preferable when B is high. Yet, both approaches reduce the distance from the real ordering as B increases if p is reasonably high ($p \geq 0.8$).

Note that majority voting or more sophisticated approaches can be conveniently employed to aggregate the answers of noisy workers, so as to attain a sufficiently high quality of the answers. Aggregating answers from multiple workers requires scaling up the budget by a non-negligible factor [Sheng et al., 2008], thus making the budget savings obtained with our algorithms yet more significant.

A similar behavior can be seen in Figure 6.9(f), where the uncertainty degree is higher.

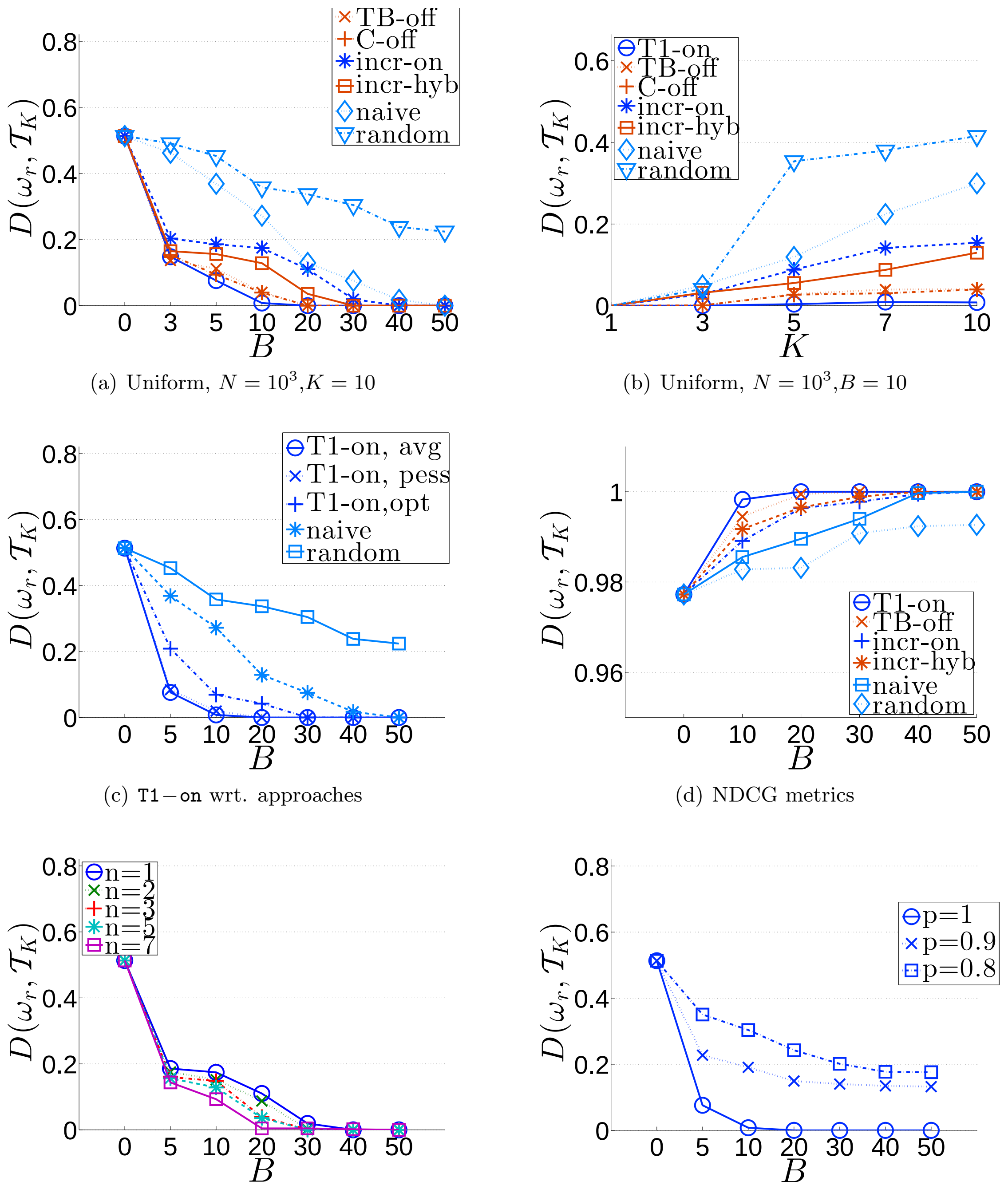
CPU time and optimizations

Figure 6.10(a) shows the required CPU time for selecting the questions when B varies. The results show that the CPU time increases with B.

A fixed amount of time, not shown in the bars, is needed to build the tree of possible orderings; this is a cost paid only once before asking any question to the workers, and required nearly 13 seconds in our experiments with the default parameter values if tree of possible orderings's probabilities are computed exactly.

$\texttt{A}^*-\texttt{off}$ is computationally intractable even for a small B. Conversely, $\texttt{T1}-\texttt{on}$ is the fastest, requiring a negligible amount of time. Among the offline algorithms, $\texttt{C}-\texttt{off}$ has the highest overhead, because at each iteration it analyzes all the questions in $\mathcal{Q}_K$ to select a question; on the other hand, $\texttt{TB}-\texttt{off}$ exhibits the lowest overhead, because it analyzes the questions in $\mathcal{Q}_K$ only once, on the first iteration.

Figures 6.10(b) and 6.10(c) show the results obtained with the strategies of Section 3.7.4. We compared the standard $\texttt{T1}-\texttt{on}$ (with MPO and exact probabilities) against $\texttt{T1}-\texttt{on}$ with MPO and probabilities approximated by sampling ("$\texttt{T1}-\texttt{on}$, sampling"), as well as $\texttt{T1}-\texttt{on}$ with the effectiveness criterion ($\texttt{T1}-\texttt{on} + \epsilon$) and the incremental algorithm ($\texttt{Incr} - \texttt{On}$).



(a) Uniform, $N = 10^3$,$K = 10$

(b) Uniform, $N = 10^3$,$B = 10$

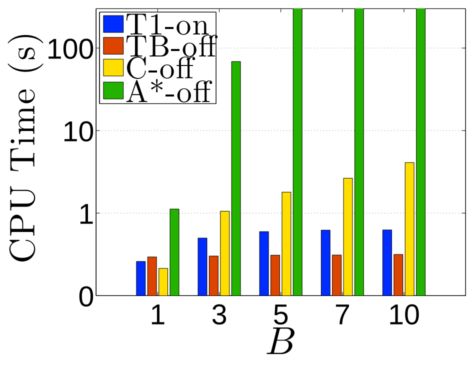
(c) `T1−on` wrt. approaches

(d) NDCG metrics

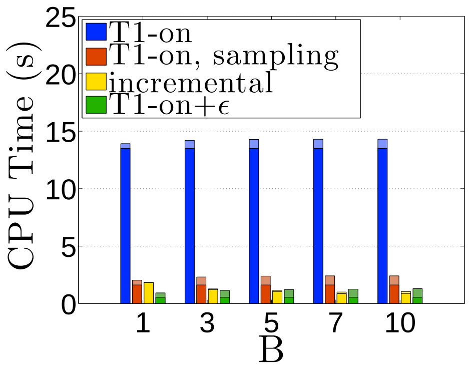
(e) `Incr − Hyb` wrt. n

(f) `T1−on` wrt. accuracy

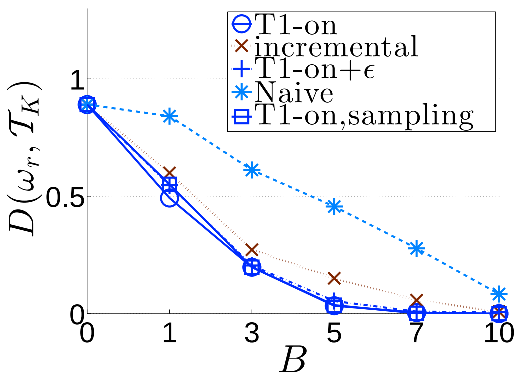
Figure 6.9: Distances to real ordering as B, K, approaches and worker's accuracy vary ($\delta = 0.003$, $N = 1000$)



(a) CPU time vs. algorithms



(b) CPU time vs. quick selection



(c) Distance vs. quick selection

Figure 6.10: CPU time ($N = 1000$, $\delta = 0.001$, $K = 10$)

The results obtained with sampling (100, 000 samples) are nearly as good as those with exact probabilities, at a fraction of the time for building the tree of possible orderings (1.6s vs 13s, as shown in the lower part of the bars). The quality of the results obtained with $\texttt{T1}-\texttt{on}+\epsilon$ and $\texttt{T1}-\texttt{on}$ is comparable, but the former is much quicker, since computing the orderings probabilities is not required.

Although $\texttt{Incr}-\texttt{On}$ still achieves good results, its quality is slightly lower, as $\mathcal{T}_K$ is not completely built, and thus only a subset of questions can be selected. Note that $\texttt{Incr}-\texttt{On}$'s CPU time decreases as B increases, since $\texttt{Incr}-\texttt{On}$ prunes orderings early in the process, thus alleviating the burden of computing many of the ordering probabilities, and thus becoming the least costly. Moreover, since $\texttt{Incr}-\texttt{On}$ considers a small number of questions at a time, the algorithm execution time is small too.

6.1.7 Comparing the methods on real data

In the following, we compare the effectiveness of the proposed online and offline methods on real datasets (as described in Section 6.1.2). Then, we test our methodologies on a real crowdsourcing platform.

YouTube dataset

Figure 6.11(a) shows the uncertainty reduction obtained by varying B between 1 and 10 and asking workers to sort pairs of videos based on temporal order of the captured event. $\texttt{C}-\texttt{off}$ has almost the same performance as $\texttt{T1}-\texttt{on}$ when B is low, while `Naive` and `Random` require many additional questions to find the real ordering. This confirms that a random question selection, even if restricted to relevant questions, is highly inefficient , since it disregards the underlying distribution capturing the uncertainty.

The same experiment was repeated adopting the ORA measure. In this case, when $B = 1$, both $\texttt{T1}-\texttt{on}$ and $\texttt{C}-\texttt{off}$ reduce uncertainty by 68%, gaining 30% with respect to the case in which the entropy measure is adopted. However, after $B \geq 7$ questions, the uncertainty reduction achieved by using both measures is the same in case of the $\texttt{T1}-\texttt{on}$ algorithm, while $B = 10$ questions are needed to achieve the same performance in the case of the $\texttt{C}-\texttt{off}$ algorithm.

Image quality dataset

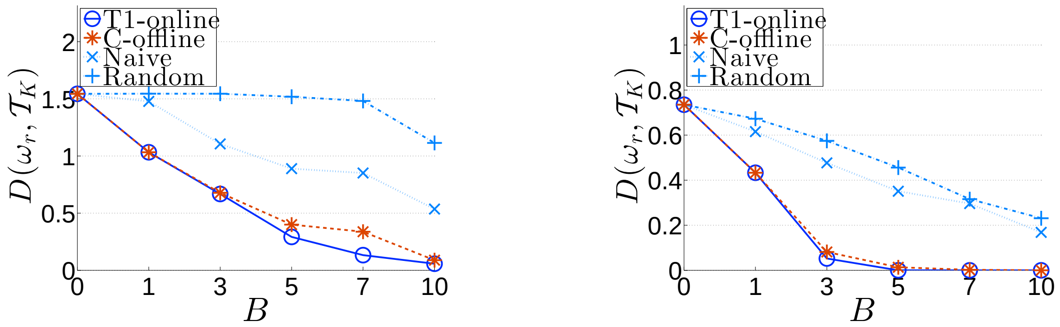
Figure 6.11(b) shows the uncertainty reduction when asking workers to compare pairs of images in terms of their visual quality. The results confirm the findings obtained on the synthetic datasets: $\texttt{T1}-\texttt{on}$ and $\texttt{C}-\texttt{off}$ always dominate `Naive`. However, here `Naive` achieves almost the same results as `Random`, since the number of overlaps between the score pdf's is very high. Thus, it is very likely that a question picked at random is relevant.

The same experiment was repeated adopting the ORA measure. In this case, when $B = 1$ the uncertainty is reduced by 52% by both $\texttt{T1}-\texttt{on}$ and $\texttt{C}-\texttt{off}$, a 30% gain with respect to the entropy measure. However, when $B \geq 5$, the same performance is attained by using both measures.

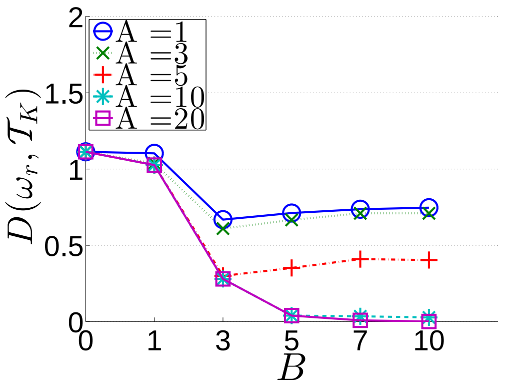
6.1.8 Tests on Microtask

In this section we analyze the effectiveness of $\texttt{C}-\texttt{off}$ when questions are asked to noisy workers on the Microtask[1] crowdsourcing platform. Ten videos from the YouTube dataset describing ten events are selected and the tree of possible orderings $\mathcal{T}_K$ is created. Then, $\texttt{C}-\texttt{off}$ is run to get the best $B = 10$ questions to ask to 20 annotators.

[1]`http://www.microtask.com/`



(a) "YouTube", $N = 10$, $K = 10$ (b) "Image quality", $N = 15$, $K = 3$



(c) "YouTube", varying # of noisy workers

Figure 6.11: Distance to solution achieved on real datasets, when B varies (`T1−on` and `C−off`)

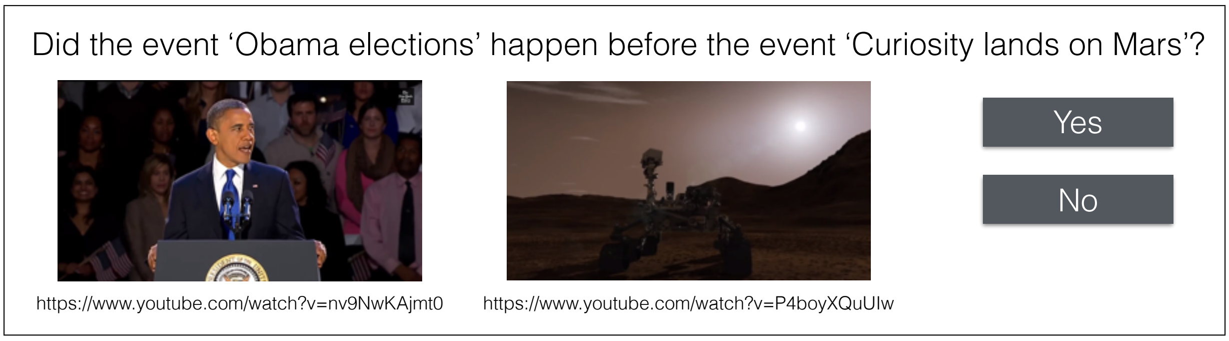
Each question is of the form *"Did A happen before B?"* (where A and B are labels describing each event, e.g., 'Obama elections '). Moreover, the interface presents a screenshot of the related videos and links the YouTube videos describing A and B. Users can decide their answer either by heart (i.e., remembering the order of the events by reading the labels / looking at the screenshots) or open the videos to look for some hint in their content. A sample of the interface design is shown in Figure 6.12

After the answers are collected, the aggregated annotation is computed via majority voting. Figure 6.11(c) shows the results obtained when the number of annotators A varies. When A is small, although uncertainty is anyhow reduced, erroneous answers will increase the distance to the real ordering. Conversely, higher values of A make majority voting more likely to remove incorrect annotations.

6.1.9 Tests on Crowdflower

In this Section we analyze the effectiveness of `Incr − Hyb` when questions are asked to noisy workers on the Crowdflower[2] crowdsourcing platform. In order to assess the quality of workers, we built 279 questions from the YouTube dataset. Each question is of the form *"Did A happen before B?"* and links the YouTube videos describing A and B. Only questions involving videos whose PDFs overlap were considered. We replicated each question 4 times, and selected 5 questions as test questions, in order to filter out workers with low quality. When a worker fails to answer correctly to more than 70% of test questions, she is considered untrusted and her answers are discarded. We paid each answer 0.01$. In this experiment, 158 workers provided answers, and their estimated accuracy on the proposed set of questions was 69.5%. Then, we ran

[2]`https://crowdflower.com`



Did the event ‘Obama elections’ happen before the event ‘Curiosity lands on Mars’?

https://www.youtube.com/watch?v=nv9NwKAjmt0

https://www.youtube.com/watch?v=P4boyXQuUlw

Figure 6.12: Interface for a task on the real crowdsourcing platform Microtask. The interface shows a snapshot of the related videos, along with their links. The user can visualize the videos before answering the question

a second experiments on a larger set of questions, where 419 questions were replicated 5 times and a more selective test question set of 10 items was used to filter out low quality workers. In this second experiment, 142 workers provided answers, and their estimated accuracy was 91.2%. The performance on the second experiment was higher, since the selected test question set was larger and more selective, and thus answers provided by low quality workers were automatically excluded.

Figure 6.13 reports the results we conducted on the Crowdflower platform. Specifically, Figures 6.13(a)-6.13(c) show the results obtained on the first experiment, while Figures 6.13(d)-6.13(f) show the results obtained on the second experiment.

The results show the latency of answers, the amount of provided correct answers and the participation of workers during the experiments.

6.2 Passive crowdsourcing optimization: Influencers retrieval on multimodal dynamic data

In the following, we show the experiments conducted for the Twitter influencers retrieval use case. The reference topic for these experiment is *food*.

6.2.1 Initial seed of users, keywords and hashtags

An initial set of users, keywords and hashtags was selected manually from Twitter, as shown in Table 6.3.

As for keywords and hashtags, the procedure shown in Section 4.7.1 was used to select the top 50 keywords and 50 hashtags, as shown in Table 6.3. These terms were sampled from the annotated data set that was used to train the text classifier (see Section 6.2.2 for further details).

As for the users, we manually crawled Twitter to select 50 users whose focus is the topic *food*. An initial seed of 10 users was provided by a field expert; the other profiles were found by navigating in the Twitter social graph and selecting relevant friends and followers of the already stored relevant users. We did not order users according to any principle.

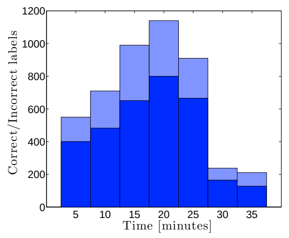
6.2.2 Classifiers

Text classifier

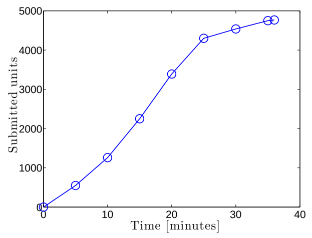
In the following, we illustrate the procedure used to train the text classifier and the obtained results.

Keywords	**Hashtags**	**Users**
banana	#recipeoftheday	@smittenkitchen
pasta	#vegan	@wichcraft
coco	#healthy	@foodgawker
asparagus	#chicken	@northernspyfood
brownie	#yum	@foodfest2014
berries	#meat	@TASTOUR
toast	#mexican	@bombfood
coconut	#cupcake	@Fooddotcom
cinnamon	#gfree	@TOCEatOut
grilled	#cupcakes	@ItsTheFoodPorn
soup	#fdbloggers	@Melbfoodandwine
avocado	#gfreefoodie	@LAMagFood
salmon	#grill	@GoVeggieFoods
shrimp	#bacon	@rawgurl
onion	#meatlessmonday	@QuadrilleFood
pancake	#mexicanfood	@sortedfood
yogurt	#pizza	@PBSFood
basil	#apple	@ColorsFood
caramel	#beer	@BuzzFeedFood
peas	#breakfast	@LAistFood
strawberries	#cheese	@HealthyEats
blueberry	#chefmovie	@EatClean
fries	#delicious	@HealthyFood
muffins	#dessert	@ArtofEating
nutella	#lunch	@Food_And_Think
mushroom	#macaron	@ImCravingFood
brownies	#vegetarian	@FoodChannel
crisp	#brunch	@DailyFoodPix
pancakes	#burrito	@JamiesFoodTube
waffles	#butter	@FoodPornMenu
rosti	#cookies	@HeraldSunFood
meatless	#easy	@OnlyFoodPorn
artichoke	#eatclean	@YourFoodPorn
creamy	#kosher	@kraftfoods
spinach	#lemon	@TescoFood
stir	#macarons	@BeFitFoods
mozzarella	#pasta	@olivemagazine
pretzel	#sliceofcomfort	@foodrepublic
biscuit	#sugarfree	@cookbooks365
frosting	#sundaysupper	@WOWFoodPics
juice	#yummy	@YahooFood
layer	#baking	@SBS_Food
melon	#biscoff	@DIYfoods
pastry	#blueberrypie	@HealthFood
pesto	#britishsandwichweek	@thedailymeal
pickle	#culinarystudent	@TestKitchen
pineapple	#deliciouslinks	@FoodHealth
quinoa	#diy	@FoodPornsx
carrot	#eatseasonal	@TastingTable
casserole	#eggs	@lafood

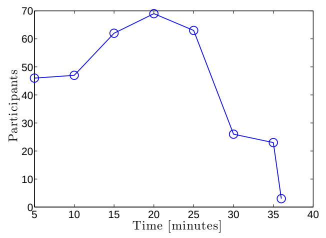
Table 6.3: Initial users, keywords and hashtags



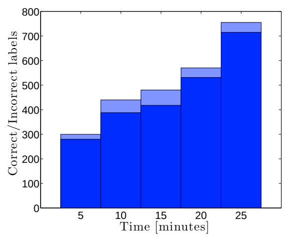
(a) Bottom: correct answers, Top: wrong answers (experiment 1)



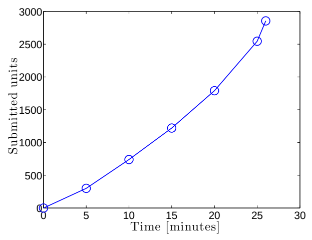
(b) Latency (experiment 1)



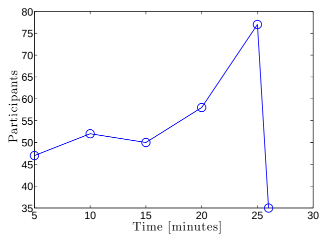
(c) Workers (experiment 1)



(d) Bottom: correct answers, Top: wrong answers (experiment 2)



(e) Latency (experiment 2)



(f) Workers (experiment 2)

Figure 6.13: Correctness of answers (a-d), Latency (b-e) and Number of involved workers (c-f)

Samples	28452
Positive samples	14234
Negative samples	14218

Table 6.4: Annotated samples which will constitute training, cross validation and test set for text classification

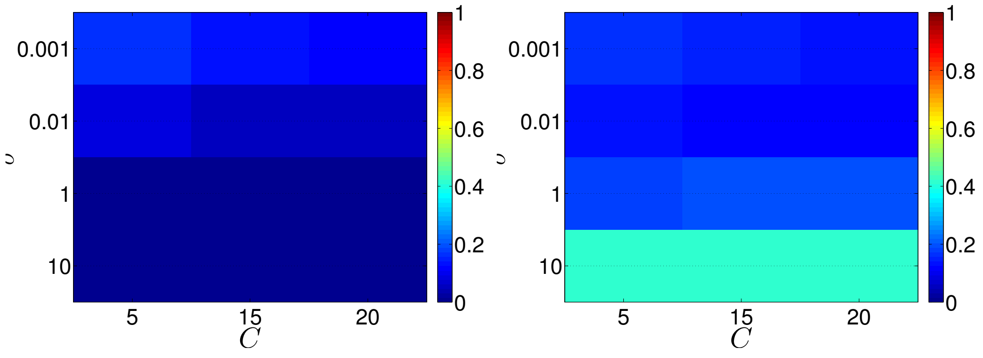
Training set. We used a REST crawler to download a large set of tweets, which were then used to constitute the training, cross validation and test sets. A large part of the collected sample was retrieved by crawling the feeds of the seed users. However, crawling from these topic-focused users provided us with a large set of positive samples and a more restricted set of negative samples. Thus, to make the data set symmetrical (i.e., so that it would contain the same number of positive and negative samples), we downloaded additional tweets from topic-unrelated profiles (e.g., actors, singers, producers, sportsmen). As a positive side effect, these additional samples enriched the dictionary with words that are not related to the topic *food*, thus enlarging the classifier's known set of terms.

At the end, the set of downloaded samples contained 28452 tweets, which were manually annotated as either relevant or not relevant for the topic *food*. The positive population contains 14234 tweets, while the negative population contains 14218 tweets (see Table 6.4). Then, we divided the set in training set $\mathcal{T}_{\text{train}}$ (60% of data), cross validation set $\mathcal{T}_{\text{CV}}$ (20% of data) and test set $\mathcal{T}_{\text{test}}$ (20% of data).

Each tweet was processed as follows: *i)* we removed URLs and user mentions from the tweet content; *ii)* we removed the hash signs from the hashtags; *iii)* we divided the text in words, removing the punctuation; *iv)* we removed the stop words; *v)* we normalized the words by lowering the characters and stemming them. The dictionary (i.e., feature set) was finally built out of the 12988 (stemmed) words in the training set.

Algorithm 10: Inter-annotator agreement computation

```
Input: Annotation set $\Theta$, Number of annotations for each annotator $K$, Annotator set $\mathcal{A}$
Output: Inter-annotator agreement $I$
  1. $\epsilon = 0$
  2. for each $a_i \in \mathcal{A}$
  3.     for each $a_j \in \mathcal{A} \setminus a_i$
  4.         for $k = 1 : K$
  5.             $\epsilon = \epsilon + \Theta_i(k) \oplus \Theta_j(k)$
  6. $I = (K \cdot |\mathcal{A}|! - \epsilon)/(K \cdot |\mathcal{A}|!)$
```



(a) Training error (b) Cross validation error

Figure 6.14: Training error and cross validation error for the text classifier

Inter-annotator agreement. Some tweets are characterized by ambiguous content, and thus annotating them as relevant or not relevant is difficult for a human annotator too. Thus, we estimated the inter-annotator agreement on a subset of the training set, to understand which is the percentage of ambiguous tweets. The selected sample, of $K = 1000$ tweets, has been provided to three human annotators, that independently annotated each tweet as either relevant or non-relevant for the topic *food*. These annotation were fed as an input to Algorithm 10, to finally compute the inter-annotator agreement. The procedure is very simple: it simply iterates over the annotators, every time setting their annotations as ground truth, and then compute the number of errors the other annotators do with respect to the defined ground truth. Here, Θ_i is the set of K annotations provided by i-th annotator, while $\Theta_i(k)$ is the k-th annotation from annotator a_i. At the end, we found that the inter-annotator agreement is 93.86%.

Performance. The classifier was tested on training set and cross validation set, and the training error and cross validation error were evaluated, as shown in Figure 6.14. The figure shows training error and cross validation error when the parameters C and σ varies. The colder the color, the smaller the errors. We would like to choose a combination of parameters in which both the errors (i.e., training error and cross validation error) are low: this combination guarantees that the classifier is neither in overfitting (i.e., low training error and high cross validation error) nor in underfitting (i.e., high training error and high cross validation error). The chosen parameter combination is $C = 15$ and $\sigma = 0.01$, since it avoids both overfitting (i.e., low training error and high cross validation error) and underfitting (i.e., high training error and high cross validation error). With this parameterization, the training error is 5.88% and the cross validation error is 11.99%. This brought to a test error equal to 11.75%.

Figure 6.15 shows the learning curves for the textual classifier. On the x axis, we indicate the percentage of available training samples used to train the classifier. It is clear how increasing the number of samples, the classifier learns the model, since the cross validation error decreases and the training error increases (symptom of a classifier leaving the state of overfitting).

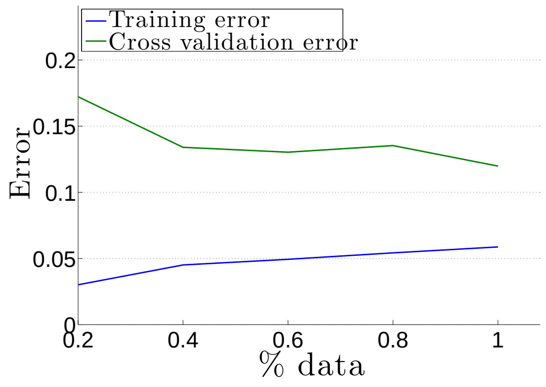


Figure 6.15: Learning curves for the textual classifier

Samples	23505
Positive samples	11759
Negative samples	11746

Table 6.5: Annotated samples which will constitute training, cross validation and test set for image classification

Image classifier

Training set. In order to build the training sample set for the image classifier, we let the pipeline run for an hour, so that it could retrieve tweets and images contained in those tweets. Then, we selected 20000 images from the retrieved sample and manually annotated them as either relevant or non-relevant for the topic *food*. Finally, to balance the data set (so that it would contain the same amount of positive and negative samples) we downloaded 3505 hard negative images, by activating a streaming crawler on random keywords (e.g., cat, sport, school, phone). At the end of the process, we obtained 11759 positive samples and 11746 negative samples.

Initially, we built a visual vocabulary by: *i)* selecting 50 positive samples and 50 negative samples at random from the sample set; *ii)* extracting their SIFT features; *iii)* clustering them in 5000 clusters. Each cluster, represented by its centroid, corresponds to a word in the visual dictionary. Each image was then processed so as to extract its feature vector, as follows: *i)* extract its SIFT features; *ii)* associate each extracted feature with a word in the visual dictionary; *iii)* build a histogram of occurrences of the words in the dictionary and normalize it.

Performance. The classifier was tested on training set and cross validation set, and the training error and cross validation error were evaluated, as shown in Figure 6.16. The figure shows training error and cross validation error when the parameters C and σ varies. The colder the color, the smaller the errors. We would like to choose a combination of parameters in which both the errors (i.e., training error and cross validation error) are low: this combination guarantees that the classifier is neither in overfitting (i.e., low training error and high cross validation error) nor in underfitting (i.e., high training error and high cross validation error). The chosen parameter combination is $C = 1$ and $\sigma = 1000$, since it avoids both overfitting (i.e., low training error and high cross validation error) and underfitting (i.e., high training error and high cross validation error). With this parameterization, the training error is 5.87% and the cross validation error is 20.17%. This brought to a test error equal to 19.95%.

Figure 6.17 shows the learning curves for the visual classifier. On the x axis, we indicate the percentage of available training samples used to train the classifier. It is clear how when the number of samples in the training set is low, the classifier shows overfitting (i.e., low training

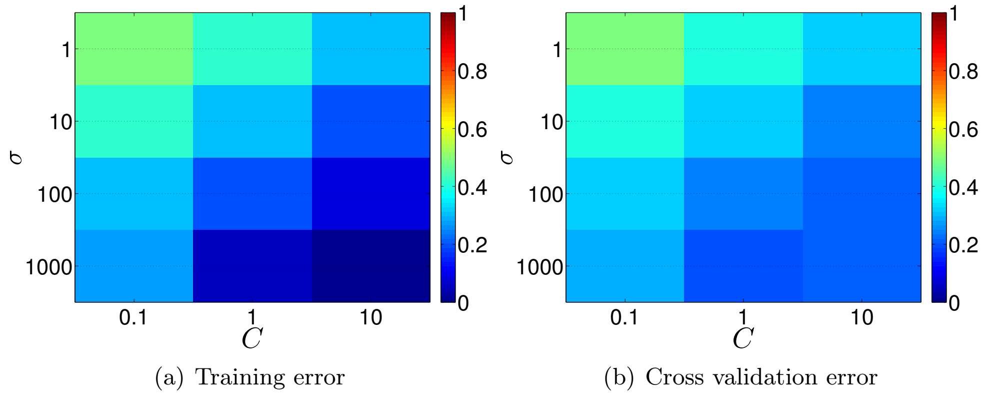


Figure 6.16: Training error and cross validation error for the visual classifier

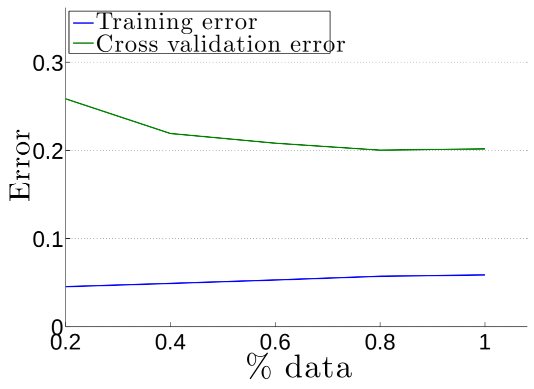


Figure 6.17: Learning curves for the visual classifier

error, large cross validation error). Instead, when the number of samples in the training set is increased, the training error increases and the cross validation decreases.

Aggregate classifiers

We collected via a streaming crawler 886 tweet samples composed of both text and image and containing at least one of the seed keywords/hashtags. These tweets were manually annotated as either relevant or non-relevant for the topic *food*, by considering both text and visual content.

Then, we ran the text classifier, the image classifier and their aggregation on the collected dataset. With this analysis, we discovered that using a single classifier allows us to achieve lower performance with respect to the case in which multiple classifier opinions are integrated. In particular: *i)* 87.92% of tweets were classified correctly by the text classifier; *ii)* 81.72% of tweets were classified correctly by the image classifier; and *iii)* 89.28% of tweets were classified correctly by the aggregation of classifiers.

Evaluation on real-time data

Previous Sections evaluate the classifiers performance in a controlled environment, where duplicates, empty tweets and ambiguous samples (where abbreviations and typos are introduced) were manually removed. In this Section we want to prove that our methodology guarantees an improvement in performance also when non-filtered tweets from the real-time feed are classified.

To prove our claim, we instantiated two pipelines for the automatic download and classification of real-time feed tweets: *i)* the first one classified tweets with text classification; *ii)* the

	Text tweets	**Text+Image tweets**
Text classification	$\texttt{Accuracy} = 0.75218$ $\texttt{Precision} = 0.65665$ $\texttt{Recall} = 0.80538$ $\texttt{F1} - \texttt{measure} = 0.72345$	$\texttt{Accuracy} = 0.73467$ $\texttt{Precision} = 0.63612$ $\texttt{Recall} = 0.80299$ $\texttt{F1} - \texttt{measure} = 0.70988$
Multimodal classification		$\texttt{Accuracy} = 0.8223$ $\texttt{Precision} = 0.79125$ $\texttt{Recall} = 0.97215$ $\texttt{F1} - \texttt{measure} = 0.87239$

Table 6.6: Performance of text classification and multimodal classification over real-time data

second one classified tweets with multimodal classification. The pipeline ran from 1^{st} to 15^{th} January 2015. Then, we extracted and manually annotated a random sample of 1900 tweets: *i)* among the ones containing only text and processed via text classification; *ii)* among the ones containing both text and images and processed via text classification; *iii)* among the ones containing both text and images and processed via multimodal classification. Annotators were required to state if each tweet was cooking-related. Finally, we computed accuracy, precision, recall and $F1$-measure in all three cases. The results are reported in Table 6.6. It is shown that text classification performance degrade when images are involved, since text classification is not able to interpret multimedia content and may misinterpret the text associated with the image. On the other hand, multimodal classification improves performance with respect to text classification only, increasing both precision and recall.

Throughput and processing time

The estimated throughput of tweets processed by the classification pipeline is 323 each minute. Each tweet is processed in 0.19 seconds.

The percentage of relevant tweets is 37.14%, and the percentage of irrelevant tweets is $62,86\%$.

The processing time for the text classifier is 0.366 seconds for each tweet that passes all the filtering phases (i.e., non-appropriate content filtering, bad words filtering). The processing time for the image classifier is 0.420 seconds for each image.

6.2.3 Influence evaluation

In this Section, we demonstrate how the proposed influence metrics is able to retrieve users that are topic-focused and relevant.

Baselines

In this Section we illustrate the influence computation baselines we used in this work to assess the quality of our influence metrics.

Tweet count metrics. This metrics ranks users according to their tweet post frequency: the larger the number of posted tweets, the higher the influence. This generates from the assumption that the more active is a user (i.e., the more content she publishes), the more influential she is.

Number of followers metrics. This metrics ranks users according to the number of their followers: the larger the number of followers, the higher the influence. This generates from the assumption that a user with a large number of followers is considered an interesting user by other users in the network, and thus a potential influencer for the field. Several works base their influence score computation on this baseline (see Section 7.2.1 for references).

Results

We collected daily the top-25 influencers list over the period 1st December 2014-15th December 2014, for the following three metrics: *i)* tweet count metrics, *ii)* number of followers metrics, and *iii)* our custom metrics. Figure 6.18 shows a classification of influential users retrieved using these metrics.

The retrieved influential users were manually assigned to classes, which describe their profile over Twitter. The results show the following:

1. The *tweet count* baseline metrics retrieves mainly common users and spammers, which are neither well known over the network, nor focused on the topic. These users are very active and publish a large quantity of content, which is not necessarily relevant for the analyzed topic, and often generated automatically by bots. Other topic-related users are retrieved as influential too: however, their percentage is still very limited.

2. The *number of followers* baseline metrics retrieves mainly real-life celebrities (e.g., actors, singers, sportsmen) and social media celebrities (e.g., quote feeds, users who publish viral videos). These users are very well known over the network, but publish a really small amount of topic-related content. Food celebrities are retrieved as influencers too: however, these users are usually involved in TV shows and/or write books, and their activity over Twitter is focused on advertising their products rather than publishing topic-related content. Other topic-unrelated sources (e.g., media channels, magazines, news feeds, generic blogs) are retrieved too, because they are read by a large number of users.

3. Our metrics retrieves mainly topic-related accounts (e.g., *food porn* accounts, recipe feeds, food magazines). This happens since the creativity, activity and communicativeness of users are taken into account while computing their influence degree over the network, excluding those users that either publish mostly topic-unrelated content (as for the tweet count metrics) or are generic celebrities.

6.2.4 Content relevance

In this Section, we demonstrate that the retrieval of influencers via an appropriate metrics allows us to retrieve also large quantities of relevant, topic-related content.

We collected the lists of top-25 users in the period 1st-15th December, and extracted the set of unique users that in this period were reported as influencers. Then, we retrieved 10 tweets for each user in this set via REST API. These tweets were manually annotated as either relevant or non-relevant for the considered topic, for each influence metrics. The results show that:

1. the *tweet count* metrics retrieved 23.07% of relevant content

2. the *number of followers* metrics retrieved 26.60% of relevant content

3. our influence metrics retrieved 76.77% of relevant content

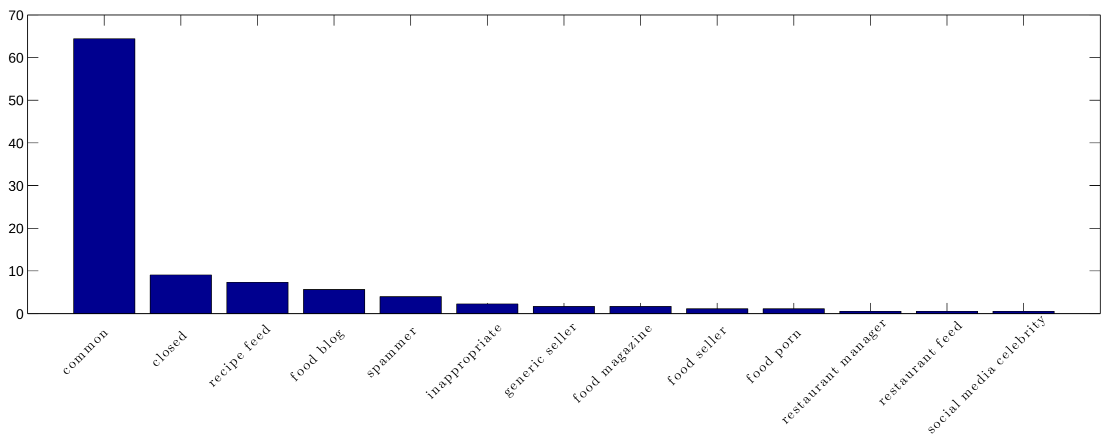
6.3 Passive crowdsourcing optimization: Detection of video ancestry relationships

6.3.1 Methodology

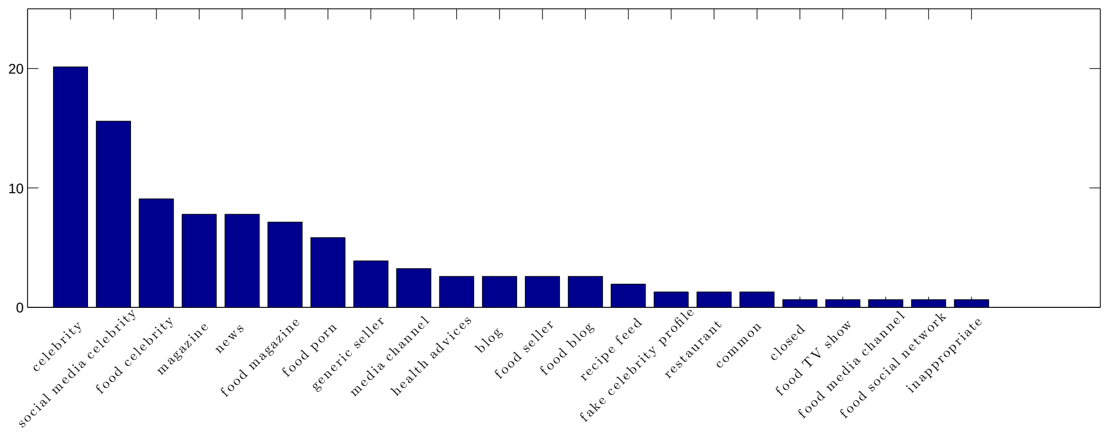
In the following, the datasets used in the experimental evaluation are described.

Synthetic dataset.

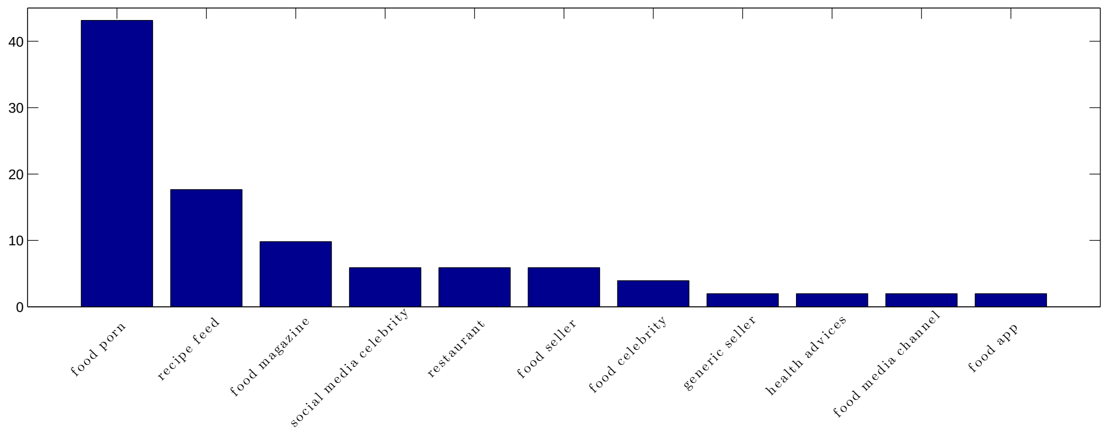
The synthetic dataset consists of $V = 30$ videos, whose number of segments is comprised in the interval $[1, 7]$.



(a) Tweet count



(b) Number of followers



(c) Custom metrics

Figure 6.18: Comparison between influence metrics.

Minimum video length	00:55
Maximum video length	54:33
Minimum number of segments per video	1
Maximum number of segments per video	129
Average number of segments per video	24

Table 6.7: Parameters for `pope − dataset`

Non-corrupted phylogeny. We suppose the original content to be incorporated in a set $\mathcal{R}$ of root nodes (containing a maximum of 5 videos). The similarity graph $G = (\mathcal{V}, E)$ initially contains the roots: $\mathcal{V} \equiv \mathcal{R}$, $E \equiv \emptyset$. Then, it is expanded by copying the content of $\mathcal{R}$ in other videos. To do so, we generate new videos $\bar{\mathcal{V}} = \{v_1, \dots, v_k\}$ $(k < V - |\mathcal{V}|)$, we simulate the duplication by connecting the segments in $\mathcal{V}$ with the segments of some $v_i \in \bar{\mathcal{V}}$ and we add $\bar{\mathcal{V}}$ to $\mathcal{V}$. Then, process is repeated. The construction of the graph terminates when the graph contains V nodes.

Corrupting the phylogeny. The graph G corresponds to a non-corrupted phylogeny, in which the weight of non-existing edges is $w = 0$, while the weight of existing edges is $w = 1$. In the following, we describe the procedure used for integrating noise in the phylogeny, so that false positives and false negatives are added, i.e., existing edges may become non-existing and viceversa. To do so, we created two Normal distributions $\mathcal{N}_0 = (0, \sigma_0)$ and $\mathcal{N}_1 = (1, \sigma_1)$, and we used them to resample weights for non-existing edges and existing edges, according to the following methods:

- **Gaussian-only**: the weights of non-existing edges are sampled from $\mathcal{N}_0$ (introducing false positives); the weights of existing edges are sampled from $\mathcal{N}_1$ (introducing false negatives)
- **Binary-Gaussian**: the weights of non-existing edges are sampled from $\mathcal{N}_0$ (introducing false positives); for each existing edge $e = (s_i, s_j)$ a weighted coin is tossed (q is the probability of getting head): if we get head the weight of $\bar{e} = (s_j, s_i)$ is sampled from $\mathcal{N}_1$ (introducing false positives), otherwise it is sampled from $\mathcal{N}_0$; moreover, the weights of existing edges are sampled from $\mathcal{N}_1$ (introducing false negatives)

Obviously, the larger is σ_0, the larger is the number of introduced false positives, while the larger is σ_1, the larger is the number of introduced false negatives. Then, all the weights are thresholded ($\tau = 0.5$), so that if a weight w is such that $w < \tau$, we set it to 0.

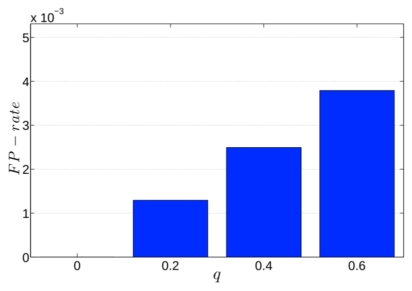
Real dataset

The real dataset (`pope − dataset`) consists of a collection of 15 videos, which where downloaded from Youtube. All the videos in the dataset are related to a specific topic ('pope Benedict resignation'). The videos were downloaded by specifying the following queries: `pope Benedict resignation`, `pope Benedict last Angelus`, `papa Benedetto ultimo incontro parroci`. The length of the videos varies between one minute and one hour.

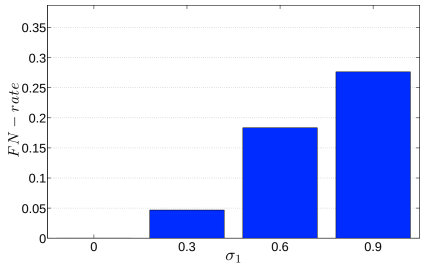
In total, the collection contains 367 segments, which were manually segmented so as to avoid over-segmentation or under-segmentation problems (using an automatic segmentation process). The features of the dataset are summarized in Table 6.7. Then, we generated the video similarity graph by matching the SIFT descriptors of the segment keyframes.

6.3.2 Metrics

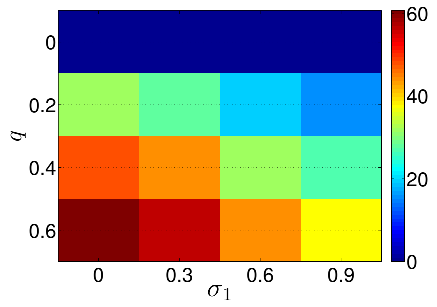
In this section the metrics used to test the performance of the algorithms are presented.



(a) FP rate



(b) FN rate



(c) Number of cycles

Figure 6.19: False positive rate (a), False negative rate (b) and number of cycles(c) (binary-gaussian noise)

The `roots` metrics measures the percentage of root nodes that are correctly restored by the algorithm. If $\mathcal{P}$ is the non-corrupted phylogeny (having $\mathcal{R}$ as root nodes set) and $\mathcal{P}^C$ is a corrupted version of $\mathcal{P}$ (having $\mathcal{R}^C$ as root nodes set), then:

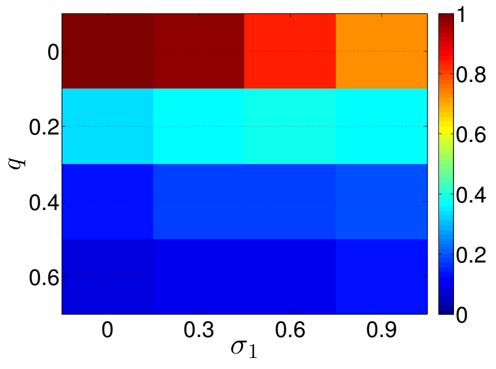
$$\texttt{roots} = \frac{|\mathcal{R} \cap \mathcal{R}^C|}{|\mathcal{R} \cup \mathcal{R}^C|} \tag{6.3}$$

Similarly, the `leaves` metrics measures the percentage of leaf nodes that are correctly restored by the algorithm. If $\mathcal{L}$ is the set of leaves for $\mathcal{P}$ and $\mathcal{L}^C$ is the one for $\mathcal{P}^C$ then:

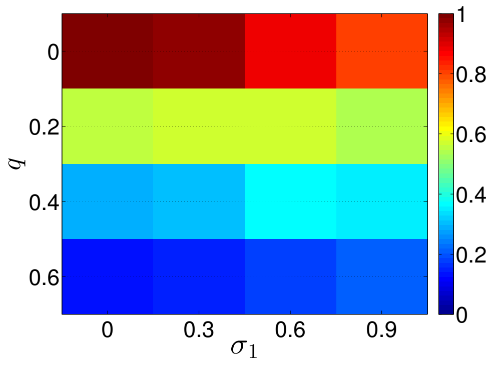
$$\texttt{leaves} = \frac{|\mathcal{L} \cap \mathcal{L}^C|}{|\mathcal{L} \cup \mathcal{L}^C|} \tag{6.4}$$

6.3.3 Results

Binary-gaussian noise



(a) `roots` metrics



(b) `leaves` metrics

Figure 6.20: Features of the corrupted phylogeny $\mathcal{P}^C$ (binary-gaussian noise)

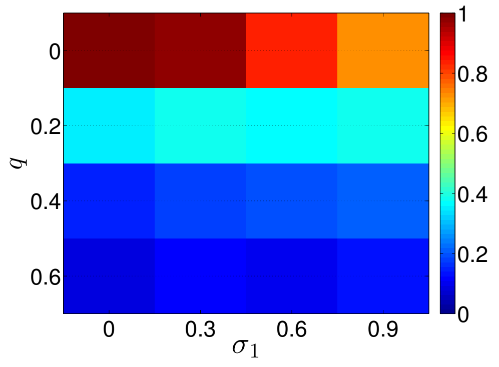
The first experiment was conducted on the synthetic dataset when $\sigma_0 = 0$, σ_1 is varying between 0 and 0.9 and q varies between 0 and 0.6. The produced false positive rate and false negative rate are shown in Figure 6.19.

The phylogeny $\mathcal{P}$ was firstly built and then corrupted, producing $\mathcal{P}^C$. Figure 6.20 shows the `roots` and `leaves` measures when no algorithms were executed to restore the original phylogeny. The results show that the larger q, the lower the coherence of $\mathcal{P}^C$ with respect to $\mathcal{P}$, since the larger is q, the higher is the probability of adding the edge $\bar{e} = (s_j, s_i)$ as false positive when $e = (s_i, s_j)$ already exists. However, when both σ_1 and q are large, the similarity of $\mathcal{P}$ and $\mathcal{P}^C$ is larger, since the left tail of $\mathcal{N}_1$ is under the threshold τ and there could be the possibility for the weight of $\bar{e}$ of being thresholded (without introducing any false positive). This result is confirmed when looking at the number of cycles that are introduced in the phylogeny $\mathcal{P}$ (see Figure 6.19(c)), which is very large when σ_1 is small and q is large.

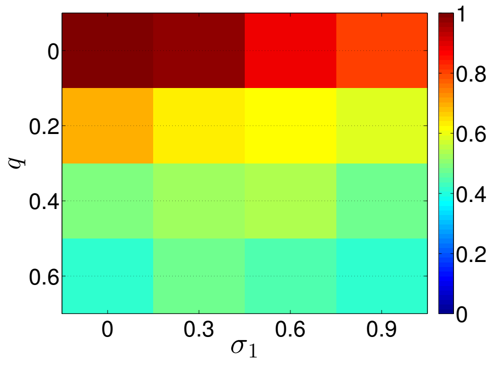
Figure 6.21 shows the performance of the DFS algorithm when 1, 30 and 80 iterations are performed. Notice that a single iteration is sufficient to increase the similarity between $\mathcal{P}$ and $\mathcal{P}^C$ with respect to Figure 6.20. Moreover, the larger is the number of iterations, the higher is the performance. However, we are not able to restore completely the original phylogeny $\mathcal{P}$, due to the problems exposed in Section 5.2.5.

Gaussian noise

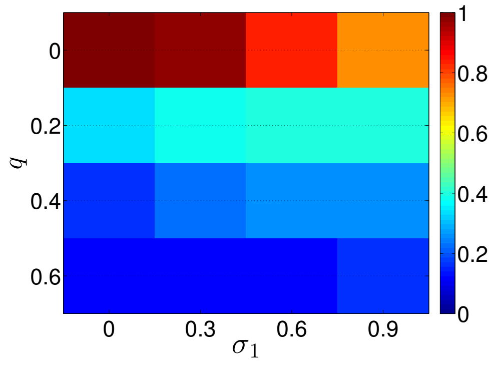
The second experiment was conducted on the synthetic dataset when $q = 0.6$ and σ_0, σ_1 are varying. The produced false positive rate and false negative rate are shown in Figure 6.22.



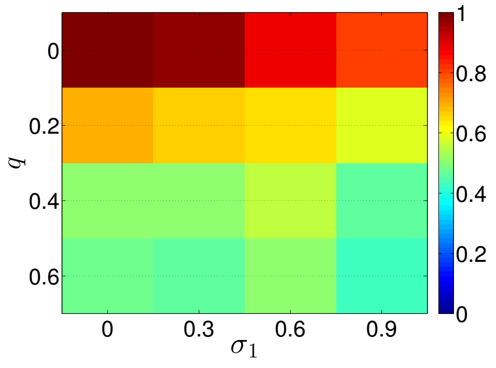
(a) `roots` metrics, one iteration



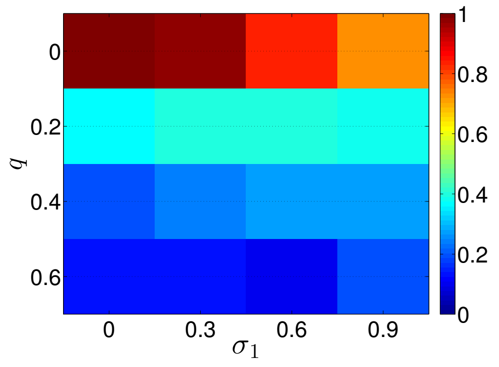
(b) `leaves` metrics, one iteration



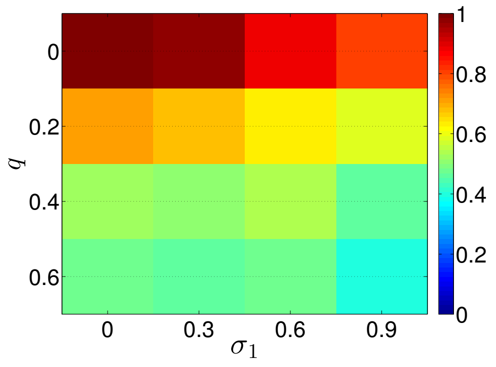
(c) `roots` metrics, 30 iterations



(d) `leaves` metrics, 30 iterations

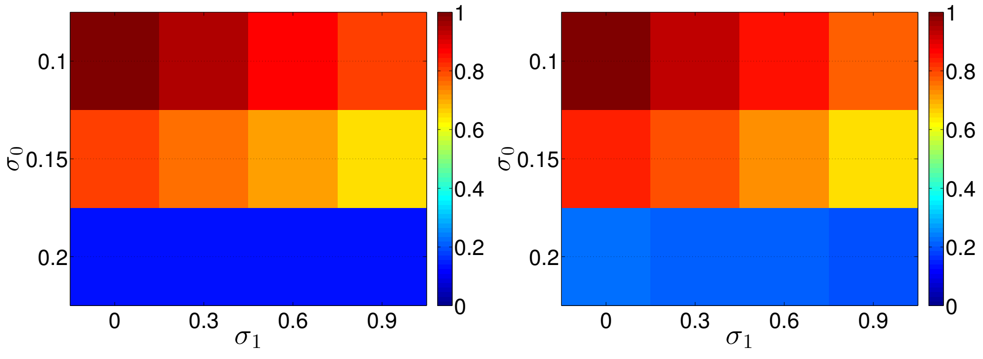


(e) `roots` metrics, 80 iterations



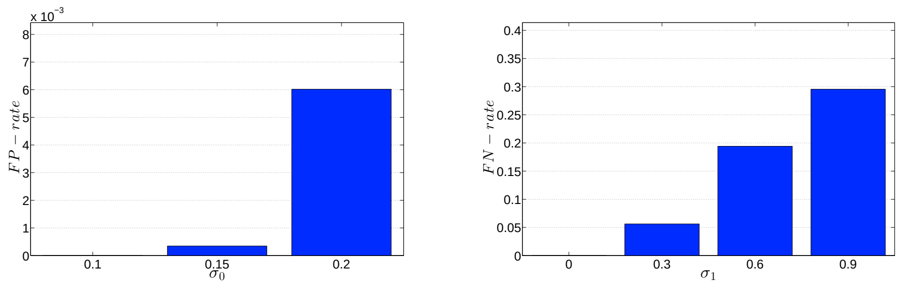
(f) `leaves` metrics, 80 iterations

Figure 6.21: Performance of the DFS algorithm (binary-gaussian noise)



(a) `roots` metrics (b) `leaves` metrics

Figure 6.23: Features of the corrupted phylogeny $\mathcal{P}^C$ (gaussian noise)



(a) FP rate (b) FN rate

Figure 6.22: False positive rate (a) and False negative rate (b) (gaussian noise)

Again, the phylogeny $\mathcal{P}$ was firstly built and then corrupted, producing $\mathcal{P}^C$. Figure 6.23 shows the `roots` and `leaves` measures when no algorithms were executed to restore the original phylogeny. The results shown that the larger σ_1 and σ_0, the lower the similarity between $\mathcal{P}$ and $\mathcal{P}^C$.

Figure 6.24 shows the performance of the DFS algorithm when 1, 30 and 80 iterations are performed. It is shown that the larger is the number of iterations, the higher the performance. However, due to the problems introduced in Section 5.2.5, the DFS algorithm is not sufficient for restoring completely the phylogeny $\mathcal{P}$.

Experiments on the real dataset

Figure 6.25 shows the performance of the DFS algorithm applied on `pope − dataset`. In particular, we considered two versions of the graph:

- `notAnnotated` **graph**: the actual graph, as returned after the SIFT matching phase;
- `annotated` **graph**: a modified version of the `notAnnotated` graph, which was annotated by the crowd, with the goal of deleting some of the relationships that are incorrect. A total of 488 annotations were collected.

The results show that the performance are quite low on the `notAnnotated` graph, since videos were all similar between each other and thus a huge amount of noise was introduced in the graph (115 cycles were present). However, when the `annotated` graph is considered, the performance of the DFS algorithm are increased, although part of the noise still remains in the graph (60 cycles are still present). This is because by deleting part of the noise from the graph, the directionality of the duplication process is easily identified, together with roots and leaves.

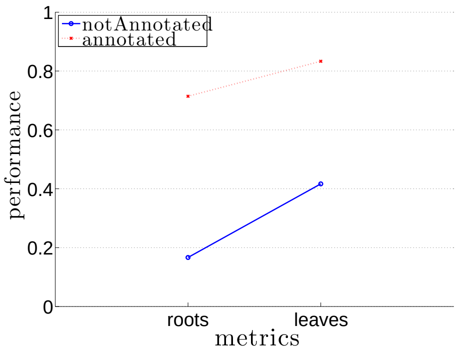


Figure 6.25: Performance on the real dataset

6.4 Discussion of results

In this Section we discuss about the findings of this Thesis, as proved by the experiments presented in this Chapter.

6.4.1 Active crowdsourcing optimization: Fighting uncertainty on structured data

Objective. In this Section we devised strategies for the minimization of uncertainty in a space of possible orderings via crowdsourcing. The experiments were run in a context in which the maximum budget (i.e., the maximum amount of crowd tasks that can be asked to the crowd) was fixed to B questions. The objective was to find the set of B questions that guaranteed the maximum uncertainty reduction over the set of possible orderings.

Approach. The designed strategies are classified in two families: the first family of algorithms require to materialize the full space of possible orderings, with consequent delay at indexing time, while the second family (called *incremental*) alternates tree construction steps with question selection steps. Moreover, each algorithm involve different selection strategies: with the *online strategies*, questions are asked one at a time and question selection takes into account already collected answers, while with *offline strategies*, B questions are selected in batch before asking them to the crowd. The selection is based on the expected uncertainty reduction a question can provide: the larger the expected uncertainty reduction, the higher the probability the question will be selected. To this extent, we defined several metrics that measure the underlying data uncertainty. Experiments were conducted on both synthetic and real data.

Results. The experiments were divided in two slots:

- choice of the best uncertainty metrics, to quantify uncertainty over the set of uncertain data;
- performance measurement for the set of proposed uncertainty reduction algorithms.

Selection of uncertainty metrics. The selection of an appropriate uncertainty metrics is crucial: given a set of unstructured and incomplete data, it is necessary to quantify how a crowd task reduces uncertainty when posed to workers. In the context of top-K queries, where the nature of the problem is strictly related to the structure of orderings and rankings, using a simple metrics such as entropy proved to be a poor choice: this metrics looks only to ordering probabilities and not to their structure, and thus does not give different value to different orderings with similar probabilities. Thus, we devised a set of metrics that take into account both orderings structure and probabilities, namely, the ORA metrics, the MPO metrics and the weighted entropy metrics. These metrics outperformed entropy: when used to measure uncertainty over the space of possible orderings, allowed to save budget and reduce quickly uncertainty. However, some of them are complex and thus require large execution time. As a consequence, we selected MPO as the best tradeoff between performance and complexity.

This section answers to the research question 1 (*How can uncertainty of structured data be modeled?*).

Performance of algorithms: full materialization of tree. Some of the proposed algorithms (namely, `T1−on`, `TB−off`, `C−off`, `Naive` and `Random`) required a full materialization of the tree in order to run. That is, before performing any selection strategy, the space of possible orderings $\mathcal{T}_K$ needs to be computed, along with its ordering probabilities. This may require a long execution time, specifically with large instances where the degree of uncertainty is high. However, the full materialization of tree allows the algorithm to be aware of all the valuable questions that may be asked to the crowd: having a large choice of questions allows one to pick up the best question possible, and leave others for later stages. Consequently, these algorithms are suggested for cases in which instances are small and the uncertainty degree is limited.

Among all the algorithms, the `T1−on` algorithm performed better, since it has the possibility of reshaping the tree of possible orderings at every received answer. Thus, its selection strategy provides the best reply at each stage. With respect to the simple `Naive` heuristic, the online algorithm attains the same performance with 50% to 80% fewer questions. Offline algorithms such as `TB−off` and `C−off`, instead, are outperformed by the `T1−on` ones, since they select blindly questions at the first stage, and are not aware of the answers that workers provide. `C−off`, however, has good performance, since it tries to foresee which answer a (good quality) worker will provide at each stage, and acts consequently. In all cases, all these algorithms outperform naive algorithms (i.e., `Naive` and `Random`) that select randomly questions. These algorithms do not apply any logic in question choice, and thus it is very probable that the selected questions are redundant or useless.

This section answers to the research questions 2 (*How do crowd task answers impact on data uncertainty?*), 3 (*How do task selection and budget constraint affect uncertainty on structured data?*) and 4 (*How does worker quality impact on crowd task effectiveness?*).

Performance of algorithms: incremental algorithms. We devised a family of algorithms, called incremental, that do not require the full materialization of tree, but rather alternate a tree construction step (level by level) and a question selection step. These algorithms act blindly with respect to what will appear in the non-built levels of the tree, and thus are outperformed by all the algorithms (but not the baselines) that require the full construction of trees. This is obviously related to the fact that the smaller the number of built levels, the more limited the question choice, and thus the best reply may be hidden behind those non-built levels that will come in the following steps. However, it happens often that the budget is completely consumed before reaching those best replies, and thus performance is lowered. However, since it is not required to build the full tree, these algorithms are performing better in terms of execution time. Consequently, these algorithms are suggested for cases in which instances are large and the uncertainty degree is large.

Specifically, the `Incr − Hyb` algorithm allows one to pose batches of questions on a crowdsourcing platform, and get batches of answers that reshape at the same time the tree. If the number n of questions in the batch is varied, it is clear how performance changes: a larger n requires a larger construction of the tree, enlarging the pool of questions among which one can choose, and thus the execution time increases, but performance increases too; a smaller n requires a limited construction of the tree, shrinking the pool of questions among which one can choose, and thus the execution time shrinks and performance worsens too.

This section answers to the research questions 2 (*How do crowd task answers impact on data uncertainty?*), 3 (*How do task selection and budget constraint affect uncertainty on structured data?*) and 4 (*How does worker quality impact on crowd task effectiveness?*).

Analysis on real data. Real data turn out to be very uncertain in some contexts. For instance, recommendation systems for items such as movies, books and hotels (e.g., IMDB, Tripadvisor, Goodreads) collect hundreds of thousands of opinions from users, which are all diversified: it happens often that the best movies are not liked by all users, and that the worse books are appreciated by at least some people. Thus, when we try to build the score PDFs for those items, we find out that all of them are overlapping, and that there is not a predominant, most probable ordering: probabilities are distributed uniformly and question selection is almost random. As an example, we report in Figure 6.26 five score distributions of books users evaluated on the Goodreads[3] website. Different colors are related to different objects. It is clear how all the PMFs are overlapping, leading to a large uncertainty degree. Moreover, assuming to be able to select at least one question to be asked to the crowd, users will answer quite randomly, since the score here is subjective. Thus, in this Thesis we did not focus on subjective scores and cases in which the uncertainty degree is too large.

Instead, we focused on cases in which the groundtruth is non-subjective, as in the case of queries on real-world events. However, also in these cases the effect of uncertainty can be difficult to be handled, and thus when the number of involved objects or the overlap between PDFs increase, some of the proposed algorithms take a very long time to run (specifically the ones that require a whole materialization of the tree). In these cases, the incremental algorithm characterizes a good tradeoff: since it does not require to materialize the full space, it allows to reduce uncertainty with acceptable costs.

6.4.2 Passive crowdsourcing optimization: Influencers retrieval on multimodal dynamic data

Objective. In this Section we devised strategies for the automatic processing of user-generated content so as to retrieve topic-related content and influential users with the minimal cost (time and storage).

Approach. We created a pipeline for the automatic analysis of content created in real-time on the Twitter microblogging platform. When the objective is to download content for a given topic, the content is processed via the following phases: *i)* exclusion of non-appropriate content; *ii)* exclusion of non-English content; *iii)* multimodal (i.e., image and text) classification of content to retain only topic-related posts; *iv)* dynamic recognition of topic-related keywords, hashtags and users so as to track dynamically communications.

Results. The conducted experiments purpose is twofold:

- assess the performance of the multimodal classification;
- assess the performance of the proposed influence metrics.

Change of topics in social communications. In this Thesis we proposed a method to track changes of topics in communications. Keywords and hashtags are always changing and reflect the interests of users that are talking on social media.

[3]`http://www.goodreads.com`

This section answers to research question 5 (*What seed queries can be used to initialize topic-related information retrieval?*).

Multimodal classification performance. In this Thesis we proved that a multimodal classification that merges the opinions of text classifier and image classifier performs better than a simple text classifier (that ignores multimedia content while assessing content relevance). Specifically, the performance of text classification and multimodal classification on a filtered test set are, respectively, 87.92% and 89.28%. On tweets collected in real-time from the Twitter firehose, instead, the gain is much more evident: the performance of text classification alone is 73.45%, while multimodal classification increases performance up to 82.23%. This result suggests that images (and in general multimedia content) carry important information, that text alone is not able to convey. Some accounts in social media publish posts that are mainly focused on multimedia content, where text is masked in images, written on photos, used as caption. This content would not be correctly classified by the most common machine learning techniques that just implement a text classifier. In this work, we proved that multimodal classification may help in increasing accuracy also in these cases.

This section answers to research question 6 (*What type of data can be analyzed to improve the accuracy of topic-related information retrieval?*).

Influence metrics performance. Influential users are commonly identified in the state of the art by looking for those people that are very well connected and that publish a lot of content. However, in this Thesis we proved that these simple influence metrics do not perform well.

When we retrieve well-connected people as influencers, we may wrongly retrieve social media celebrities or real-life celebrities. These people are in general very famous either for their public profile or for the content that they publish, but are rarely very focused on the topic we are interested in. Real-life celebrities often mix posts about the topic with posts about their personal life, while social media celebrities often talk about random topics, publish quotes of various authors and share random news. We show that the amount of topic-related content these people publish amounts to 26%.

When we retrieve active users as influencers, we may wrongly retrieve spammers or common people taking about random topics. These people are in general very active, since they want to gain visibility by posting as many tweets as possible, and are not necessarily real people: often behind a very active account hides a spam bot (whose accounts are often closed due to excessive spam). Once in a while it happens also that these people are really focused on the topic and very active, and thus deserve to be considered as influential; however, in most cases very active users produce a large quantity of topic-unrelated posts. We show that the amount of topic-related content these people publish amounts to 23%.

Our metrics, instead, proved to be very accurate, retrieving influencers that produce a large amount of topic-related content. This is favored by the fact that the metrics considers different aspects when establishing the score of a person: her activity, her connection with other users, her originality, her ability of generating interesting communications. We show that the amount of topic-related content these people publish amounts to 77%.

This section answers to research question 7 (*How are influential content producers defined and identified?*) and 8 (*What is the impact of considering content producers' influence level on the accuracy of topic-related information retrieval?*).

6.4.3 Passive crowdsourcing optimization: Detection of video ancestry relationships

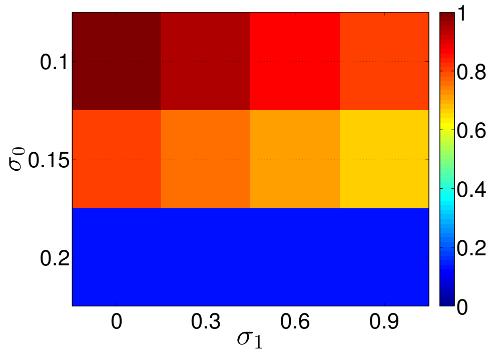
Objective. In this Section we devised strategies for reconstruction of a video phylogeny starting from a set of near duplicates.

Approach. We defined a similarity graph as a graph in which nodes represents videos and (directed) edges indicate the direction of information duplication. Then, we defined an approach for the elimination of false positive relationships from the graph. This strategy is iterative, and removes cycles from the similarity graph, which are not consistent with a phylogeny structure, thus restoring the replication chain. The approach was evaluated on both synthetic and real data.

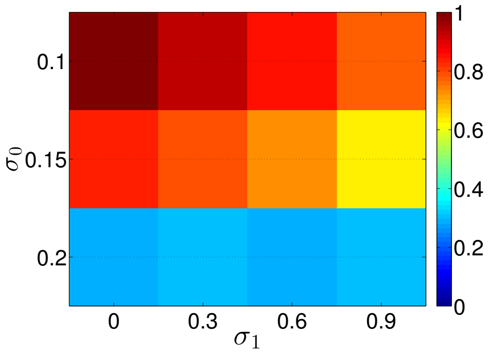
Results. The devised approach resulted effective in case the similarity graph presents a reduced level of noise, i.e., when the number of copies is limited and few near-duplicate relationships are established between nodes in the graph. Obviously, since the algorithm is iterative and each iteration removes more noise with respect to previous iterations, the larger the number of iterations, the larger the amount of noise (expressed as number of cycles in the graph) that can be removed from the similarity graph, and the higher the performance.

Unfortunately, in case of large noise (as the one presented in the real dataset), the performance of the algorithm is limited. The noise is too large, and thus it is difficult to understand which is the copy direction in the graph and which one among the near duplicates is the original copy. However, in this case active crowdsourcing may be helpful: workers are better than machines in analyzing visual content and understanding its content, and may be more accurate in recognizing superimposed logos, modification of colors and shapes, resizing etc. Thus, we performed some test using an internal crowd, where workers were required to annotate some of the relationships in the similarity graph as verified/non-verified: a verified relationship is a directed annotation between nodes A and B that the worker verified as existing, while a non-verified relationship is a relationship the worker recognized as non-existing. This increased the performance of 50% on the real dataset.

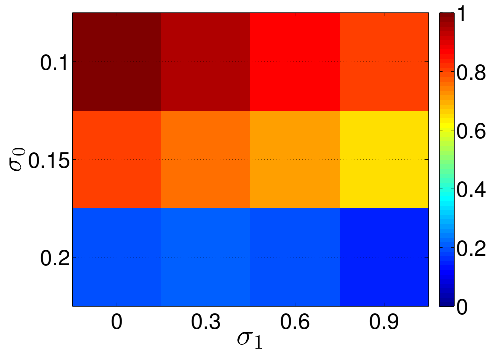
This section answers to research question 7 (*How are influential content producers defined and identified?*).



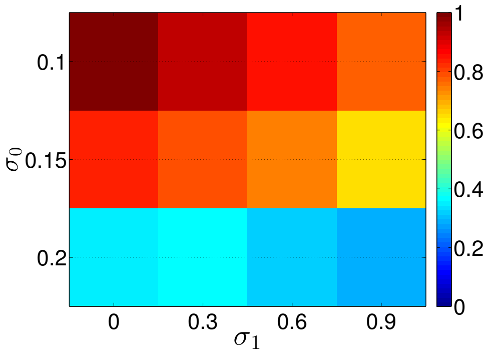
(a) `roots` metrics, one iteration



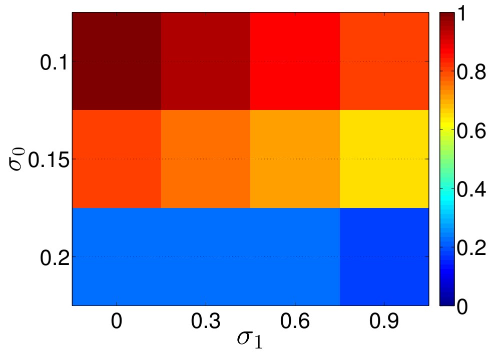
(b) `leaves` metrics, one iteration



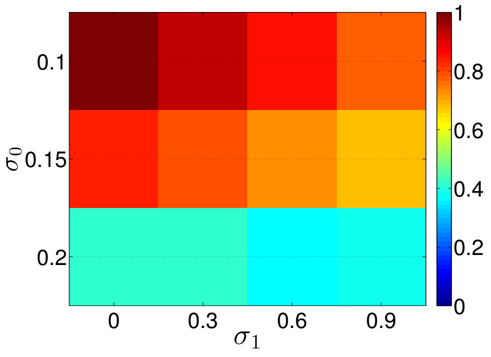
(c) `roots` metrics, 30 iterations



(d) `leaves` metrics, 30 iterations



(e) `roots` metrics, 80 iterations



(f) `leaves` metrics, 80 iterations

Figure 6.24: Performance of the DFS algorithm (gaussian noise)

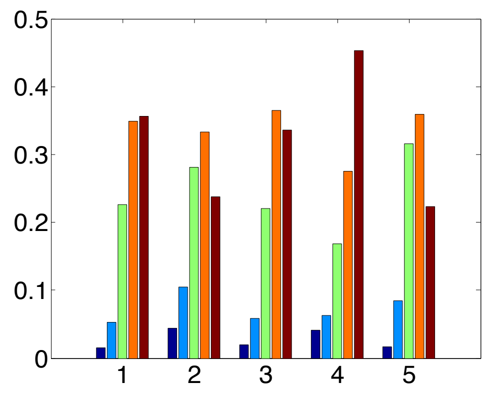


Figure 6.26: Goodreads: PMFs of 5 objects. On the x axis: scores. Different colors are related to different objects. It is clear how all the PMFs are overlapping

Chapter 7

Related work

In this Section, we contextualize and assess the work discussed in the previous Chapters with respect to academic literature.

7.1 Active crowdsourcing

7.1.1 Fighting uncertainty on top-K queries

Tree building

The problem of ranking tuples in the presence of uncertainty has been addressed in several works [Soliman and Ilyas, 2009, Li et al., 2009, Li and Deshpande, 2010]. The main idea in most proposals is to compute the space of possible orderings and its probability distribution in an efficient way. The information contained in this space is then exploited in order to retrieve the answers to different types of queries.

The work in [Soliman and Ilyas, 2009] presents a probabilistic model based on partial orders, in which the possible orderings are derived by exploiting the available information about the relative order of tuples. The space of possible orderings is thus encapsulated in a compact data structure, and techniques for processing the probability distribution over the space in an efficient way are presented. However, the probabilities attached to each ordering in the space are computed by means of a Monte Carlo simulation, which is very expensive.

The works in [Li et al., 2009] and [Li and Deshpande, 2010] propose a unified approach to ranking in probabilistic databases, providing parameterized ranking functions that generalize or approximate many of the previously proposed ranking functions. These functions are exploited so as to rank multiple tuples having uncertain scores. However, these techniques are only used to compute the probability that a tuple is ranked at a certain position, with no emphasis on the structure of the space of possible orderings.

Crowdsourcing application to resolve uncertainty

Many works in the crowdsourcing area have studied how to exploit a crowd to obtain reliable results in uncertain scenarios.

In a crowdsourcing setting, tasks are materialized as a set of simple questions (e.g., in the form `yes/no`), so that they can be easily tackled by humans. However, since these tasks are small, the gathered information is small too, and thus the requester is often forced to create a large number of tasks (and invest a large amount of money in their resolution by the crowd).

On the other hand, there may be tasks providing the same information, and thus performing all of them may be redundant. Consider for instance the case in which we would like to cluster objects in a collection so that each cluster contains instances of the same object. If we find out that object A differs from object B, and that object B is equal to object C, then asking to the crowd if A is equal to C is useless, since this information can be extracted by transitivity.

Thus, recent works focused on finding a way of *optimizing* the selection of questions, so as to minimize some cost metrics.

In [Parameswaran et al., 2011], the authors provide one of the first studies on the optimization of human computation. In the proposed scenario, binary questions are used to label nodes in a directed acyclic graph, showing that an accurate question selection improves upon a random question selection.

Similarly, [Parameswaran et al., 2012] and [Marcus et al., 2011a] aim to reduce the time and budget used for labeling objects in a dataset by means of an appropriate question selection.

Instead, [Guo et al., 2012] proposes an online question selection approach for finding the next most convenient question so as to identify the highest ranked object in a set.

A query language where questions are asked to either humans or algorithms is described in [Parameswaran and Polyzotis, 2011]; humans are assumed to always answer correctly, and thus each question can be asked once.

All these works do not apply to a top-K setting and thus cannot be directly compared to our work.

Fighting uncertainty on top-K context with crowdsourcing

We now discuss recent works on uncertain top-K scenarios where questions comparing tuples in a set are asked to a crowd.

In [Davidson et al., 2013], the authors consider a crowd of noisy workers and tuples whose scores are totally uncertain. This approach does not lend itself well to our scenarios, where prior knowledge on the score pdf's is assumed: for instance, when $N = 1000$, $\delta = 0.001$ and workers answer correctly with probability 0.8, their approach would require 999 questions to determine the top-1 tuple, while 2.7 are in average sufficient with our `T1−on`.

The work in [Marcus et al., 2011b] proposes a query interface that can be used to post tasks to a crowdsourcing platform such as Amazon MTurk. When addressing a top-K query, their method first disambiguates the order of *all* the tuples by asking questions to the crowd, and then extracts the top-K items. This amounts to asking many questions that are irrelevant for the top-K prefix, since they could involve tuples that are ranked in lower positions. The wasted effort grows exponentially as the dataset cardinality grows. Instead, our work only considers questions that involve tuples comprised in the first K levels of the tree.

A more recent work in [Polychronopoulos et al., 2013] builds the top-K (top-1 in [Venetis et al., 2012]) list by asking workers to sort small sets of s tuples whose scores are, again, totally uncertain. The top-K tuples are determined via a voting mechanism that refines the set of top-K candidates after each "roundtrip" of tasks, until only K tuples are left. Although when $s = 2$ the tasks are a comparison of two tuples like in our approach, their question selection is completely agnostic of any prior knowledge on the tuples, thus resulting in a much higher overall amount of questions in scenarios like those considered in this paper.

In [Das Sarma et al., 2014] the authors propose procedures for the extraction of k objects that have a specified property. The proposed algorithms extract sets of n objects that are analyzed in parallel by humans. At each round, n tasks are submitted to the crowd, and the objects that are recognized as relevant (i.e., objects having the specified property) are retrieved. Rounds are continuously created, until exactly k objects are retrieved. The work takes into account both the cases in which humans are oracles or noisy workers. However, the work just proposes way of extracting a set of objects, which is not guaranteed to contain the k most relevant objects, and does not order the retrieved instances.

In [Nieke et al., 2014] the authors build the top-K list of a set of objects described by several attributes. At first, all the objects with certain score are extracted and ordered by descending score, producing a tentative top-K list. Then, objects with uncertain scores are ordered in descending order of their probability of being relevant, and one after another are posted to a crowdsourcing platform to disambiguate the missing attributes. The answers are then used to update the resulting list of objects, in an online fashion. However, the authors handle only those cases in which the score distributions are uniform. Moreover, the proposed crowd tasks in which users are required to fill in missing attributes are far too complicated, so that in most of the data set it would be really difficult to obtain a significant answer.

Fighting uncertainty with crowdsourcing in object matching

In [Zhang et al., 2013], uncertainty in schema matching is tackled by posing questions to workers. The same authors propose a demonstrator of the same techniques in [Zhang et al., 2014]. Uncertainty is measured via entropy, and two algorithms (online and offline) are proposed to select the questions reducing uncertainty the most. A similar approach is proposed in [Fan et al., 2014] for the context of web tables schema matching, although only an online scenario is considered in this case. We have shown that, in top-K contexts, the results obtained by measuring uncertainty via entropy are largely outperformed by the use of other criteria (e.g., U_{MPO} and ϵ).

Noisy workers are used to validate schema matchings also in [Hung et al., 2013], with emphasis on the design of questions, so as to maximize their informativeness and reduce the noise in validations. Yet, [Hung et al., 2013] does not present any question selection strategy, which we have shown to be a useful means to obtain good results even with a noisy crowd and simple boolean questions.

There are several other works about object matching in the state of the art. An example is [Wang et al., 2013b], where the objective is to identify all pairs of matching objects between two collections of objects. The authors propose a mixed online and offline approach, where the selected sequence of question is annotated partially by machines and partially by users, and the sequence is selected so as to minimize the number of questions answered by humans. However, the authors of [Vesdapunt et al., 2014] proved that the claims by [Wang et al., 2013b] are wrongs and the solutions are improvable. Thus, they propose two alternative algorithms for entity resolution. Another work is [Wang et al., 2012], where entity resolution (i.e., find different records that refer to the same entity) is performed via a hybrid approach, where machines identify the most uncertain pairs (i.e., those pair of objects with high probability of being instances of the same entity) and users disambiguate those pairs. Tasks are of two types: the first one is about validating the equivalence of pairs of objects; the second one is about labeling single objects in different categories.

Applications of crowdsourcing techniques in other contexts

A very recent work [Parameswaran et al., 2014], which extends another work from the same authors [Parameswaran et al., 2012], mixes human computation and machine computation so as to perform the task of filtering, i.e., using humans to evaluate or rate items such as images, videos or text. In this work, possible answer states are enumerated and the optimal state minimizing the cost and the error is found via linear programming. The updates with respect to the previous work include distinct worker abilities (i.e., workers may have different performance on the same set of tasks) and prior information (i.e., some knowledge about the fact that some items are more likely to pass the filter than others), which is computed automatically.

The work [Marcus et al., 2012] treats the problem of counting occurrences of an object in a set of uncertain objects. The use case is the one of finding all the photos of a red-headed person in a collection of images. People are asked to evaluate either some or all the images in the set, and the analysis of the provided answers produces an estimate for the number of instances of the query object. The work proposes two approaches: *i)* the first one requires users in the crowd to label manually images as relevant/non-relevant, until a large sample is annotated and the confidence of the estimate is high; *ii)* the second one provides a set of randomly selected images, and requires the user to provide an estimate of the number of instances of the query object in the set. The results show that the first approach is suited for every type of data sets, although it is costly, while the second one is suited for multimedia data sets only, since humans are not able to count the number of text instances without reading all the samples.

The work [Franklin et al., 2011] proposes SQL schema and query extensions that enable the integration of crowdsourced data and processing in the DBMS. With this approach, people help the DBMS in answering queries, by performing tasks that complete missing data or validate possibly wrong tuples. Thus, developers can write SQL queries without having to focus on which operations will be done in the database and which will be done by the crowd.

7.2 Passive crowdsourcing

7.2.1 Influencers retrieval from multimodal activity data

Characteristics of influencers

Influencers in Twitter are mainly celebrities, popular bloggers and organizations [Zhai et al., 2014, Cataldi and Aufaure, 2014, Bi et al., 2014], and result related to similar arguments of discussion (i.e., fashion, music, etc. that could be labeled as entertainment) [Cataldi and Aufaure, 2014]. Moreover, Twitter users tend to strengthen the relationships with users in the same areas of interest. Thus, their retweet connections with similar users allows to identify and discriminate their main area of influence [Cataldi and Aufaure, 2014].

It is evident that each estimated influence value is strictly dependent from the considered topic-based community: National Geographic has a higher influence value on the scientific domain than the one of Barack Obama on political news when considering number of retweets, since a larger number of authoritative information sources retweeted National Geographic [Cataldi and Aufaure, 2014]. Highly connected users and/or community can easily result in higher estimated influence values in their domains of interest [Cataldi and Aufaure, 2014].

On the other hand, passive users (i.e., people who follow many people but retweet a small percentage of the information they consume) are robot accounts (which automatically aggregate keywords or specific content from any user on the network), suspended accounts (which are likely to be spammers) and users who post extremely often. Moreover, the amount of attention a person gets may not be a good indicator of the influence they have in spreading the message, and users with very low number of followers often have high influence [Romero et al., 2011].

There is evidence that influence changes over time, so that the group of top-10 influencers change frequently, leaving space for other people to become influencers for the same topics [Cha et al., 2010].

Influence metrics

Several metrics were proposed in the state of the art for identifying an influencers on a social network. These metrics try to infer the degree of influence of a given user, and are all mainly based on: *i)* some descriptor of the text of the user posts; *ii)* some descriptor of the social network of the user. Generally, the multimedia content (e.g., photos contained in the posts) is not considered.

In the following description, we consider the use case of Twitter. Similar metrics may be found for the other social media, but since Twitter is the most analyzed source in this field of research, we can focus on it.

Text metrics. In this Subsection, we introduce some possible descriptors of the text of the user tweets.

Analysis of original tweets of a user.

The most used descriptors of the posts text are be the number of original tweets, the number of shared URLs, the number of used keywords and hashtags [Pal and Counts, 2011, Jabeur et al., 2012], which come natural. Moreover, when the analysis focuses on a specific topic, one could decide to measure the self-similarity score of a user, which measures how similar are the author's recent tweets with respect to her previous tweets: if the author focuses himself on a topic, then the self-similarity of her posts is expected to be high [Pal and Counts, 2011].

Furthermore, some works [Weng et al., 2010] take into account the *homophily* of different authors: users that talk about similar topics are easily involved in what the other is saying, while low homophily profiles do not share topics of interest.

Finally, since users could produce text containing typos, some works [Cataldi and Aufaure, 2014] compute all the possible n-grams (i.e., all the possible combinations of characters of every word in the original tweet text) and use them as a descriptor of the produced text. This introduces more robustness on the typos in the text, since at least one of the produced n-grams will contain the correct, typo-free term.

However, there are some works which deviate from the typical approach. An example can be found in a quite recent work [Quercia et al., 2011] that evaluates the influence of an author by estimating her behavioral traits. This work uses the LIWC dictionary [1] to extract language categories which are typical of some personality traits (e.g., self-esteem, self-confidence). Then, for each tweet one can extract the percentage of words that describe each language category: if the most shown traits are typical of an influential person, then the user is considered an influencer.

User involvement.

A *mention* in the form of `@user` captures a user attention to follow the content published by the author. Some works [Pal and Counts, 2011, Jabeur et al., 2012, Cha et al., 2010, Lian et al., 2012] use this factor as a metrics for stating how much a user is able to involve others in the topic. The degree of involvement is measured by taking into account, for instance, the number of mentions of others by the authors and the number of users mentioned by the authors.

Conversational degree of a user.

Some works in the state of the art measure the conversational degree of a user on Twitter, based on the presence of mentions in the tweet text [Pal and Counts, 2011, Lian et al., 2012].

A *conversational tweet* is a tweet directed to other user. To create such tweets the authors put the mention `@user` before the tweet text, meaning that the tweet is directed to the user `@user`. The conversational degree of a user is thus generally computed depending on the number of conversational tweets produced by the user, the number of conversations started by the user (i.e., those conversations whose first conversational tweet was produced by the user) and the number of conversational tweets that involve the user.

Content replication on Twitter.

A *retweet* on Twitter is the copy of forwarding of a user's posts by other users. To create such tweets the authors put the string `RT @user`, meaning that the tweet is copied from the user `@user` (i.e., the original author of the post is `@user`).

Several works consider the content replication degree (i.e., the capability of diffusing the content on Twitter) as a possible descriptor of the influence of a user [Pal and Counts, 2011, Jabeur et al., 2012, Lian et al., 2012, Cataldi and Aufaure, 2014, Cha et al., 2010, Kong and Feng, 2011]. This degree is measured in terms of the number of tweets the user copies from others, the number of tweets others copy from the user, the number of users that were involved in retweeting operations with the user and the number of topic-related retweets.

Aggregated metrics.

Some works [Pal and Counts, 2011, Zhai et al., 2014] propose ways of aggregating the factors explained so far, which, by combining simple statistics on the tweets (such as number of retweets,

[1] `http://www.liwc.net`

number of conversational tweets, etc.), measure the topical focus of an author, her retweet impact, her ability of diffusing content etc.

Graph metrics. In this Subsection, we introduce some possible descriptors of the social network of a user's social graph.

The influence of a user depends on the structure of her social network: the larger it is, the higher is the probability that the information is diffused and shared. Several works [Zhai et al., 2014, Pal and Counts, 2011] captures the extension of a user's social network by considering the number of her followers (i.e., the users reading her contents) and friends (i.e., the users whose content is read by her). However, not all the followers and friends are interested in the content the user publishes, since their focus could be other topics. Thus, a more refined analysis [Cha et al., 2010, Agarwal et al., 2008] filters the number of followers and friends so as to consider just the ones that talk about the topic on which the user in analysis is focused.

When talking about influence on the social network of a user, one could consider two important factors: *homophily* and *reciprocity* [Kwak et al., 2010]. Homophily is the similarity between a user and her followers and friends: it states how much the topics, the geographic position and the popularity degree of the friends/followers in are similar to the ones of the user in analysis. Reciprocity, on the other hand, is the property for which a user in the social network follows another user just because that user followed her. While homophily is a good descriptor of how much two users in a social network are close (i.e., are similar), reciprocity is not a good descriptor of the relevance of a user for another, since it is just an expression of politeness towards one's followers.

Other works use other mathematical properties of graph structures to measure the level of influence of users. For instance, the works [Shetty and Adibi, 2005, Sun and Ng, 2013] measure the influence of a user u by removing her from the graph $G(U)$: if the difference between the graph entropy of $G(U)$ and the graph entropy of $G(\{U \setminus u\})$ is high, then this means that u has high influence on the graph structure. In other cases, instead, the influence of a user on another user is measured by computing their distance on the network, which can be expressed either as a sum of the weights connecting the two nodes in the network (where the weight depends on the relationship type, i.e., friend, follower, not in relation) [Weitzel et al., 2012], or simply by the number of edges separating the two nodes [Cataldi and Aufaure, 2014].

Finally, several works [Pal and Counts, 2011, Zhai et al., 2014, Kazienko and Musial, 2007, Sun and Ng, 2013, Agarwal et al., 2008] build aggregated metrics that compute the information diffusion, the topical follower signal, the social position, the relationship strength and the topological influence of a user in her network.

Other metrics. Other works build descriptors based on other factors. For instance, the work in [Chen et al., 2014] checks if the profile of a user is verified, meaning that she corresponds with high probability to a celebrity and thus has a high influence. Moreover, the same work applies sentiment analysis techniques to state whether the user is talking good or bad about a specific topic. Indeed, usually fans talk well about a subject, while experts criticize it. Detecting a high expertise level of a user is a suggestion of the fact that he is an influencer on the topic.

Datasets

In the state of the art, most of the works focused on influence evaluation on social networks build their own datasets by crawling the target social network for a certain period of time. Thus, it happens rarely that the proposed machinery analyzes the social network in real time to find out which are the changes in influence. However, the work in [Cha et al., 2010] states that the influence degree changes over time and attention of the most moves to other topics and users, so that the top-10 influencers are renewed from time to time.

Algorithms

Score computation. The simplest used influencers retrieval approach used in the state of the art is the one of computing an influence score for each user, and then return the top-K users with the largest influence score [Agarwal et al., 2008, Kong and Feng, 2011, Bi et al., 2014].

Clustering and classification. Some works perform user clustering based on user influence characteristics, to find groups of users having similar influence on the network [Pal and Counts, 2011, Chen et al., 2014]. In other cases, users are classified in influence classes (e.g., influential, popular, listener, highly read) according to their features (both text-based and graph-based) [Quercia et al., 2011].

Graph structure analysis. In the state of the art, some works that classify the importance of a node with respect to the graph topology can be found too [Shetty and Adibi, 2005]. Here, each node n_i is temporarily removed from the graph G, and the graph entropy gap between G and $G \setminus n_i$ is computed. Then, the node n_i with the largest entropy gap is selected as the most influential, since it causes the largest impact on the graph. Similar works analyze the network to find those users with the largest ability of spreading news and content over the network [Saez-Trumper et al., 2012]. Other works create algorithms inspired to PageRank [Jabeur et al., 2012, Cataldi and Aufaure, 2014, Weng et al., 2010].

Baselines

In the following, we list some algorithms used by other authors as baselines for influencers retrieval.

The work [Pal and Counts, 2011] uses the same retrieval algorithm, although the influence metrics is downgraded to a simpler version (e.g., when the metrics considers both graph-based and text-based metrics, baselines could be defined as the same metrics, in which either the graph-based or the text-based dependence is relaxed).

Several works use the PageRank algorithm [Page et al., 1999] as baseline [Romero et al., 2011, Saez-Trumper et al., 2012, Kong and Feng, 2011, Huang et al., 2013, Kwak et al., 2010, Weng et al., 2010] or a variant of the same algorithm [Jabeur et al., 2012]. Other works [Kong and Feng, 2011] consider the HITS algorithm as baseline.

Centrality metrics [Huang et al., 2013] (e.g., degree, closeness) and graph-based characteristics [Jabeur et al., 2012, Kwak et al., 2010] (e.g., number of followers [Weng et al., 2010], number of retweets) can be used as baselines to estimate the user importance.

Finally, works in which the topic of the tweets is estimated from the data use other (simpler) classification algorithms (e.g., Naive Bayes classifiers, K-nearest neighbors classifiers) as baselines [Cataldi and Aufaure, 2014].

Results evaluation

Influence is a subjective measure, and thus multiple way of assessing the performance of the proposed metrics and algorithms are proposed in the state of the art.

Trivial metrics. Some works use some trivial metrics to state the accuracy of their work. For instance, the work in [Weng et al., 2010] computes the accuracy measure by intersecting the obtained result set (i.e., set of influencers) with the set of most active users in the dataset. The implicit assumption here is that an active user is also an influencer, which in some cases can be true. However, since this does not hold always, this metrics is too simple to capture an intricate measure such as the influence (which depends on several factors).

Manual evaluation. Several works propose evaluation metrics based on manual evaluation of the result set [Pal and Counts, 2011, Shetty and Adibi, 2005, Cha et al., 2010, Huang et al., 2013, Zhai et al., 2014, Bi et al., 2014, Cataldi and Aufaure, 2014, Weng et al., 2010]. This evaluation procedure is based on the manual check of the influencer profile, to see whether it can be considered as an influencer by humans. For instance, if the topic is the movie 'Toy Story 3 ', then the director of the same movie can be considered as an influencer for the topic.

User study. Other works [Pal and Counts, 2011, Hannon et al., 2010, Jabeur et al., 2012, Chen et al., 2014] perform users studies with numerical evaluation of the topic relevance and influence metrics. In these studies an annotator (which usually is a specific expert in the field) or a set of annotators are required to go through the set of results, to assess its quality. Evaluations can be either anonymous (e.g., only the text is shown to the annotator, and thus the influencer profile is not visible) or non-anonymous (e.g., the name of the user is known and thus the profile is visible). In case of multiple annotators, the opinions are aggregated in a unique evaluation, using classical methodologies (e.g., majority voting).

Manual ground truth. When the focus is the one of classifying tweets in topics [Cataldi and Aufaure, 2014], manual ground truth construction is performed on the collected dataset, so as to compute the accuracy at the end of the classification process. This can be generally done when a fixed dataset or a fixed training set is crawled from the social network.

Top-K posts on other services. Some work use other services to evaluate their topic classification quality. These services are usually news aggregators whose aim is to select viral Internet issues. Consequently, the top-K posts one can find on those services correspond to the K posts with the highest visibility on the network. For instance, the work in [Cataldi and Aufaure, 2014] compares its results with the top-K posts on Digg[2]. Another example can be found in [Kwak et al., 2010], which compares its result set with the one of Google Trends[3] (i.e., a collector of trending topics by Google) and CNN Headlines[4].

Other works use the same principle, but exploiting information that is already present in the analyzed social network. For instance, [Bi et al., 2014] computes the accuracy of its results by counting the number of verified profiles that are present in the retrieved result set, since the verified users are considered celebrities (and thus relevant users).

Content relevance

The growth of social media caused the growth of the volume of user-generated content. Showing the whole set of published content to users brings to a poor user experience, where users are not able to visualize only the most relevant content: spam and replicated content can be shown to the users too if not filtering techniques is applied.

Mixing content relevance and user influence. Some works mix influence metrics and content relevance metrics to identify relevant content. The idea behind these works is that if a user is influent for a specified topic, then she is also producing relevant content for the topic. Thus, selecting her content as the most relevant guarantees a high quality in the retrieved results.

The work [Silva et al., 2013] builds a bipartite graph with users and posts, and unleashes a random tweeter that randomly selects a tweet from a Twitter feed, navigates to the related user page, and restarts with the tweet random selection step. The most visited users are the ones considered as most influent, and their posts are considered the most relevant.

[2]`www.digg.com`
[3]`http://www.google.com/trends/`
[4]`www.cnn.com/QUICKNEWS/`

The work [Schenkel et al., 2008] queries the social graph to retrieve the most relevant documents, by checking two factors: *i)* the similarity between the query keywords and the documents, and *ii)* the social context around the user that performs the query. The idea is that the documents created by the user's friends will be appreciated, since people that are close in the social network tend to share the same interests.

A similar work is [Gou et al., 2010], where user geolocation, social context, topics of interest and bookmarks of the user that is performing a query are used to assign a document a first relevance score, which is then refined by computing the similarity between the document and the query keywords.

Another work exploiting the same idea is [Yin et al., 2010], where social influence and document similarity with the query are combined to compute the relevance score of a document.

Graph characteristics. The work [Bakshy et al., 2011] investigates the influence of Twitter users by estimating the diffusion depth and visibility of their content. The authors build cascades, i.e., phylogenies in which the flow of reposts of the original content is shown. The larger the cascade, the larger the visibility of that content, the larger the influence of the user.

Diversity of content. The works [De Choudhury et al., 2011b] and [De Choudhury et al., 2011a] (extension) does not look for content created by influencers, stating that this approach would bring to the selection of tweets from celebrities, which are not necessarily relevant. Thus, the work looks for relevant tweets by selecting a diversity level, and retrieving the most relevant tweets with that degree of diversity. Diversity guarantees that content is always new and not replicated.

Content relevance via sampling. Some past works propose to select at random content and show it to users, so as to reduce the amount of shown information and thus help users in focusing on what they are presented with. However, this obviously brings to low performance, since the sampled content is not guaranteed to be relevant. Thus, more refined techniques can be found in the state of the art for the selection of relevant content.

Given a topic, [Ghosh et al., 2013] proposes to select at first experts for that topic, and then retrieve a fixed number of sample from their tweet feeds. This helps in increase the content quality, since it is supposed that expert users publish topic-related content.

Content relevance prediction. The work [Lerman and Hogg, 2010] predicts which Digg articles will become relevant (and come up in the first page) in short time. This is done by modeling the way in which news enter and exit from the top-K list of relevant news, and applying it to current, real-time data to predict future outcomes.

Similarly, [Khosla et al., 2014] applies various techniques from computer vision and machine learning to predict the view count for each image in a dataset.

Content relevance on community question answering. Several works analyze community question answering (such as Yahoo! Answers[5] or Stackexchange[6]) so as to recognize the most relevant answers and questions. As an example, [Agichtein et al., 2008] builds a graph in which users, answers and questions characterize nodes in a graph, and with a PageRank-based solution recognizes the most important nodes.

[5]`http://answers.yahoo.com`
[6]`http://stackexchange.com`

Assessment via active crowdsourcing. Some works design crowd tasks in which workers are asked to establish the content quality of some (usually multimedia) objects. As an example, [Keimel et al., 2012] design tasks to assess video quality. Here, two videos (the original video and its distorted video) are shown to the workers, with the request of evaluating the quality difference between the two versions. This study shows how the comparison of multimedia complex objects (such as videos) may bring to bad crowdsourced results, since users tend to leave the task unfinished, due to fatigue. Moreover, this type of evaluation cannot be applied to large data sets, since the effort needed to manually evaluate thousands of objects is too large.

The work [Vonikakis et al., 2014] asks workers to select a representative set of relevant images from a pool of portrait photos. The crowdsourcing study reveals identifiable patterns in the photo selection process, with more appealing photos securing more HITs than less appealing photos. Then, a classifier is trained so as to identify the most relevant photos automatically, using the manually annotated photos as training set.

The work [Van Kleek et al., 2012] uses a game with a purpose called Twiage to find relevant content on Twitter. The game shows the player a set of tweets, and consists in the recognition of the most relevant tweet in the pool. For each annotation, users get scores, so as to motivate them.

Classification of tweets in topics

The work [Lee et al., 2011] proposes a strategy for the classification of trending topics (i.e., a popular topic on Twitter) based on a text classifier and an influence metrics. The idea is that the textual content can help in the classification of a tweet in a predefined category (e.g., sports, cooking, art, fashion), and consequently in the classification of the contained trending topics. Moreover, given two topics T_1 and T_2, if the set of influencers for those topics have strong overlaps, then the topics are similar and should be classified as belonging to the same class.

The work [Yerva et al., 2011] classifies tweets containing a given keyword, to state whether it is related or not to a given company. A company profile is traced by collecting keywords from the related website. These keywords are used as feature set, and the relevance of the tweet for that company is computed by classifying its content with a text classifier on the selected feature set.

The work [Sriram et al., 2010] proposes an approach for the classification of tweets into predefined set of generic categories such as News, events, Opinions, Deals and Private Messages. The features are extracted from the tweets and user's profile, while the feature set for each category is learned from a manually annotated training set.

The work [Genc et al., 2011] introduces a Wikipedia-based classification technique, where tweets are classifies by mapping message them into their most similar Wikipedia pages, and calculating semantic distances between messages based on the distances between their closest Wikipedia pages.

The work [Becker et al., 2011] explores approaches for the distinction of tweets between real-world events and non-event messages. The authors use an online clustering technique to group topically similar tweets together, and compute features that can be used to train a classifier to distinguish between event and non-event clusters.

Focused Web crawling algorithms

The information available on the Web can be exploited to collect more on-topic data by intelligently choosing what links to follow and what pages to discard. This process is called *focused crawling* [Novak, 2004]. With this procedure, a crawler is started with a set of seed pages that indicate the type of content the user is interested in, and provide the initial links. These pages are put in a priority queue and are subsequently downloaded. Retrieved pages are then evaluated for topic relevance. Hyperlinks found on pages are extracted and ran through a filter, so as to filter pages that are specified as black-listed pages.

In some early works [De Bra et al., 1994] Web crawling was simulated by a group of fish migrating on the Web, where each URL corresponds to a fish and its survivability depends on the visited page relevance. Only when a fish traverses a specified amount of irrelevant pages it dies. [Hersovici et al., 1998] extends this concept, so that URLs of pages are prioritized by taking into account a linear combination of source page relevance, anchor text and neighborhood of the link on the source page. Later, some works based on PageRank were proposed [Cho et al., 1998].

[Chakrabarti et al., 1999] uses an existing document taxonomy and seed documents to build a model for classification of retrieved pages into categories. In [Chakrabarti et al., 2002] page relevance and URL priorities are decided by separate models. [Ehrig and Maedche, 2003] considers an ontology-based algorithm for page relevance computation. Entities are extracted from the page and counted, and page relevance depends on the number of entities it contains.

[Aggarwal et al., 2001] introduces a concept of intelligent crawling where the user can specify an arbitrary predicate (e.g., keywords, documents similarity) and the system adapts itself in order to maximize the harvest rate.

Multi-modal classifiers

In the state of the art, several works that suggest how to merge different classifiers opinions can be found.

The most used technique is the *majority voting*, which can be applied in different ways [Woźniak et al., 2014]: *i)* unanimous voting, in which a sample is classified as positive only if all the classifiers agree; *ii)* simple majority, in which a sample is classified as positive only if the percentage of consensus the sample had is greater than half the pool of classifiers; and *iii)* majority voting, in which a sample is classified as positive if the majority of received votes is positive. An alternative way of collecting votes requires to weigh differently the decisions coming from the classifier pool [Woźniak et al., 2014]. Similar techniques are the max rule, min rule and median rule [Kittler et al., 1998].

In case of multi-classifiers systems in which all the classifiers work together on the same feature set, approaches like boosting (e.g., AdaBoost) and bagging (i.e., bootstrap aggregating) can be used [Briem et al., 2002]. With the boosting technique the classifier is iteratively trained on a training set where weights for easy samples (i.e., the ones that the classifier classifies correctly) decrease and weights for hard samples (i.e., the ones that the classifiers classifies wrongly) increase. With the bagging technique the classifier is trained several times on re-sampled subsets of training set, and the opinions are then aggregated so that every opinion has equal weight during the voting. This helps in reducing the error on the test set, since multiple training configurations are used and this increases the generalization.

In other cases, support function fusion can be used to aggregate different classifiers opinions [Woźniak et al., 2014]. A support function provides a score for the decision taken by an individual classifier. The value of a support function is the estimated likelihood of a class, computed either as a neural network output, a posteriori probability, or fuzzy membership function.

Finally, some trainable fusers can be applied to aggregate probabilistically classifier opinions [Woźniak et al., 2014]. As an example, Dempster and Shafer's theory allows to reach a consensus on the weights to combine several decisions. Similar results can be obtained by applying Bayesian formalism [Xu et al., 1992].

7.2.2 Video phylogeny and ancestry relationships detection for multimedia objects

The work [Kender et al., 2010] characterizes a seed work in this field. Here, a YouTube dataset is analyzed to characterize near duplicates between videos. Videos, in this perspective, are like organisms within an ecology, competing for survivals. Each organism (i.e., video) breed generating new organisms, which contain DNA samples (i.e., video segments) inherited from their ancestors. The work builds a graph in which near duplicate relationships are specified, and a plot of a possible phylogeny is provided.

The work [Dias et al., 2010] supposes that, when an image is manipulated by users, it is possible to track down the structure of transformation the image was subjected to. Thus, it introduces the definition of an Image Phylogeny Tree (IPT), i.e., a tree where the root represents an image, and each branch represents a manipulation of the original image that brings to a new version of it. Obviously, each manipulated version of the image can be further manipulated, branching again the tree in a new level. All the manipulations of the same image are called *near duplicates* of the original image, in the sense that their content is similar and that the duplicate can be traced back to the original version. Hence, the authors propose a method for the construction of an IPT starting from a set of images potentially containing some duplicates. The tree, when constructed, is able to suggest which is the original content, which images have been duplicated from it, and which kind of transformations were applied to the image to bring to those modified versions. A similar approach is proposed in [Dias et al., 2011], where a phylogeny tree is built for a set of videos instead of a set of images. However, all these works suppose that there can be only a single root for a phylogeny, and that all other duplicates are generated from a single source. Nevertheless, multimedia objects (specifically videos) can be generated by pasting together multiple pieces of videos coming from different sources.

Chapter 8

Conclusions

In this dissertation we have shown how to optimize the application of active and passive crowdsourcing techniques to minimize costs and maximize the quality of the output.

In the context of active crowdsourcing, we proved that data uncertainty can deeply affect the quality of the result, and that via crowdsourcing humans could help in reducing that uncertainty and improve the outcome. However, the application of crowdsourcing has its own cost: the larger the number of tasks workers have to solve, the larger the cost the requester has to bear. Thus, we developed a framework for measuring the uncertainty on the underlying data and identifying the most promising tasks to be posed to the crowd, with particular focus on top-K queries, so as to minimize the costs and achieve the largest expected uncertainty reduction. Results show that an accurate selection of tasks allows us to save budget and converge to an uncertainty-free result in short times.

In the context of passive crowdsourcing, we developed a framework for the automatic analysis of user-generated content, so as to identify topic-related content in short times and focused crawling. The analysis is conducted both on textual data and images, with the objective of identifying relevant content and influencers (i.e., celebrities for the topic). Results show that an accurate, real-time analysis of content brings to a low cost identification of relevant entities (i.e., interesting content and people).

8.1 Summary of the work

In the following, we summarize the research work carried out in this Thesis.

8.1.1 Active crowdsourcing

The research process in the field of active crowdsourcing reported in this Thesis was focused on the top-K query application. Some works in the state of the art proposed to model data uncertainty via a tree of possible orderings (i.e., a set of orderings which are all potentially valid for the underlying set of tuples), while others proposed how to compute the probability of an ordering. We started with the aggregation of several contribution in this field, and adapted them to the top-K context.

Then, we designed a crowd task type that could help in reducing the uncertainty in the tree of possible orderings, and we defined how an answer to that task could modify the tree of possible orderings (i.e., by deleting some of the orderings that are not compliant with the collected answer).

After that, we defined four uncertainty metrics that measure the amount of uncertainty in the tree of possible orderings. When a specific crowd task is provided as an input, these metrics allow us to compute the expected uncertainty reduction that task would produce. Consequently, given a set of possible crowd tasks, we order them according to the expected uncertainty reduction they would bring, and select the task (or the set of tasks) that are most promising, i.e., that bring to the largest uncertainty reduction over the tree of possible orderings.

The selection of tasks can be performed in different ways: a first type of selection methods requires the full materialization of the tree of possible orderings (which can be huge when the uncertainty on the data is large), while a second type requires to materialize the tree incrementally. Among the first type, we can list the online methods (i.e., methods that select the i-th question by considering all the answers that were collected with the previous $(i-1)$ tasks) and the offline methods (i.e., methods that select batches of B questions without requiring to collect any answer in the meanwhile). Among the second type, we can list the incremental online method (i.e., a method in which the tree is built incrementally one level after another, and each construction step is alternated with a question selection step) and the incremental hybrid method (i.e., a method in which each construction step is alternated with the selection of a batch of n questions). Results show that methods that require the full materialization of the tree are slower, but more precise (since the question selection horizon is more extensive), while incremental methods lose precision (due to the limits during question selection phases) but are faster.

8.1.2 Passive crowdsourcing

The research process in the field of passive crowdsourcing started with the creation of a past data crawler for the microblogging platform Twitter, which retrieves tweets generated by a set of manually selected topic-related set of users. The retrieved tweets (containing both text and images) were manually annotated as either relevant or non-relevant, and used for two purposes: *i)* train two classifiers (a textual one and an image one) so as to recognize automatically topic-related content; *ii)* select an initial set of topic-related keywords and hashtags.

Then, a real-time crawler collecting currently created tweets was started so as retrieve any content that contained at least one of the identified keywords, hashtags and users. In order to follow dynamically the conversation changes, the crawler is set so as to change the monitored keywords/users periodically. Moreover, tweets are automatically classified as relevant/non-relevant by merging the opinions of the classifiers.

After that, we defined an influence metrics that judges user influence by considering characteristics that depend on both the social graph and the produced content. The more relevant and original is the produced content and the more the user is connected with others, the larger is the influence degree.

Results prove that our metrics retrieves users that are more topic-focused with respect to users retrieved by baseline metrics, and that these users produce large quantities of relevant content. Moreover, we created a dashboard that allows one to manually inspect past and real-time data, trends in communication and influencers profiles.

8.2 Contributions

The most significant contributions of this work are:

1. A framework for the quantification of data uncertainty. The main innovation of the proposed approach consists in the definition of uncertainty measures that depends not only on the probabilities on the orderings, which is the classical approach, but also on the structure of the tree of possible orderings, which makes the uncertainty measure more adherent to the structure of the problem.

2. A framework for the reduction of data uncertainty. In this Thesis, we proposed several algorithms for the selection of the best questions to be asked to the crowd (i.e., the one with the highest expected uncertainty reduction). Online and offline algorithms that require to

materialize the whole space of possible orderings have high performance, but are best suited with small instances, where the uncertainty of data is small and it is possible to materialize the space of possible orderings in little time. The incremental algorithms characterize instead a good tradeoff between latency (i.e., computational time and crowdsourcing latency) and performance, fitting the nature of the problem more than other classical online and offline algorithms, and work well also in case the degree of data uncertainty is high.

3. A multimodal classification of user-generated content, which considers not only the textual information, but also multimedia content. Images are indeed often published as a support to tweet text, and may carry fundamental information needed to understand if the published content is relevant or not. Thus, we merge the opinion of the two classifiers to have a general opinion on the relevance of the published content.

4. A pipeline for the automatic identification of topic-related content, so as to easily identify relevant content and users for a topic in an efficient and low-cost way. The main innovation of the proposed approach consists in the introduction of a scoring function that gives more credits to words appearing mainly in positively classified (i.e., topic-related) tweets. With this approach, words that describe currently produced content are periodically extracted from the analyzed tweets and fed as an input to the pipeline, so as to track dynamically changes in communication topics.

5. A metrics for the identification of influential users in social media. The metrics, which comes natural, scores as the most influential users those users that produce original and topic-related content, that are very active on the topic, and that generate large interest and communications.

6. A methodology for the automatic identification of replicated content on social media. The approach allows one to recognize among a set of near duplicates (i.e., a set of copies of the same original multimedia object) which was the original content, and how transformations (e.g., color corrections, content alteration) were spread over the network.

8.3 Open problems

In this Section, we discuss the open problems that will be investigated in the future.

In the context of active crowdsourcing, we did not treat cases in which scores are subjective and the degree of overlap between score distributions is too large. In these situations, the amount of uncertainty is too large to be treated with our algorithms, which would require long time to reduce uncertainty without eliminating it completely. Thus, additional techniques that remove a portion of uncertainty before applying our algorithms are needed. Some possible solutions could be found in: asking users to reduce the spread of an uncertain score distribution by excluding some score values, reshaping automatically score distribution so as to cut tails, mixing multiple sources of information in order to build score distributions.

In the context of passive crowdsourcing, we limited our study to microblog content and users. Here, posts are short and mainly focused on a specific topic, since there is no room for focusing on different subjects. However, moving our interest on richer content (e.g., the one extracted from blogs or news feeds) could make topic-specific content extraction more difficult to be performed, since content is often not structured and talks about several entities and subjects. Moreover, the influence metrics, which is now very focused on the content a user produces, can still be improved by adding new factors related to user profile and connection between users. Influence, as shown by works in the state of the art, is contagious, and thus if a person is very influent, it is probable that some of the users that are connected to her will be influent too for the same topic. Finally, further investigation on the relationships between content and users are needed to understand the nature of influence itself: is interesting content generating user influence, or are charismatic people creating interesting content?

8.4 Future work

The described work covers important steps of a broader research program that will address several open issues such as:

1. Generalization of the uncertainty reduction problem in the field of active crowdsourcing, so as to work on other uncertain data and queries. For example, in skill-based expert search, queries may be desired skills and result sets may contain sequences of people sorted based on their uncertain topical expertise.
2. Application of automatic user-generated content analysis and influencers/relevant content extraction to other dynamic contexts where the geospatial component is very relevant. For instances, in cases such as floods of earthquakes, content should move as the event moves in time and space, and thus following its evolution over the network could help in understanding and predicting its evolution in the real world.
3. Analysis of trends and patterns in influence change and topic focus change in communication, so as to track how influence change, how influencers stop being the most influential users to give space to other users, and how certain sub-topics (represented by keywords and hashtags) come back in vogue after being absent from communication for a certain time.
4. Analysis of sentiment and emotions of produced content, so as to understand if there is a correlation between the emotion one shows in her content and her degree of influence on the network.

Bibliography

[RGB, 2012] (2012). Colordescriptor software. http://koen.me/research/colordescriptors/readme.

[FRH, 2012] (2012). Frh video segmentation component. http://www.idmt.fraunhofer.de/en/Service_Offerings/technologies/a_d/av_analyzing_toolbox.html.

[Vid, 2013] (2013). Video segmentation. http://www.springerreference.com/docs/html/chapterdbid/63556.html.

[Agarwal et al., 2008] Agarwal, N., Liu, H., Tang, L., and Yu, P. S. (2008). Identifying the influential bloggers in a community. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 207–218. ACM.

[Aggarwal and Yu, 2009] Aggarwal, C. and Yu, P. (2009). A survey of uncertain data algorithms and applications. *Knowledge and Data Engineering, IEEE Transactions on*, 21(5):609 –623.

[Aggarwal et al., 2001] Aggarwal, C. C., Al-Garawi, F., and Yu, P. S. (2001). Intelligent crawling on the world wide web with arbitrary predicates. In *Proceedings of the 10th international conference on World Wide Web*, pages 96–105. ACM.

[Agichtein et al., 2008] Agichtein, E., Castillo, C., Donato, D., Gionis, A., and Mishne, G. (2008). Finding high-quality content in social media. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 183–194. ACM.

[Akkaya et al., 2010] Akkaya, C., Conrad, A., Wiebe, J., and Mihalcea, R. (2010). Amazon mechanical turk for subjectivity word sense disambiguation. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 195–203. Association for Computational Linguistics.

[Allahbakhsh et al., 2013] Allahbakhsh, M., Benatallah, B., Ignjatovic, A., Nezhad, H. R. M., Bertino, E., and Dustdar, S. (2013). Quality control in crowdsourcing systems: Issues and directions. *IEEE Internet Computing*, 17(2):76–81.

[Allahbakhsh et al., 2012] Allahbakhsh, M., Ignjatovic, A., Benatallah, B., Beheshti, S.-M.-R., Bertino, E., and Foo, N. (2012). Reputation management in crowdsourcing systems. In *CollaborateCom*, pages 664–671.

[Alonso and Lease, 2011] Alonso, O. and Lease, M. (2011). Crowdsourcing for information retrieval: principles, methods, and applications. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 1299–1300. ACM.

[Alonso et al., 2008] Alonso, O., Rose, D. E., and Stewart, B. (2008). Crowdsourcing for relevance evaluation. In *ACM SigIR Forum*, volume 42, pages 9–15. ACM.

[Bakshy et al., 2011] Bakshy, E., Hofman, J. M., Mason, W. A., and Watts, D. J. (2011). Everyone's an influencer: quantifying influence on twitter. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 65–74. ACM.

[Barbier et al., 2012] Barbier, G., Zafarani, R., Gao, H., Fung, G., and Liu, H. (2012). Maximizing benefits from crowdsourced data. *Computational and Mathematical Organization Theory*, 18(3):257–279.

[Becker et al., 2011] Becker, H., Naaman, M., and Gravano, L. (2011). Beyond trending topics: Real-world event identification on twitter. *ICWSM*, 11:438–441.

[Bernstein et al., 2011] Bernstein, M. S., Brandt, J., Miller, R. C., and Karger, D. R. (2011). Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 33–42. ACM.

[Bernstein et al., 2012] Bernstein, M. S., Teevan, J., Dumais, S., Liebling, D., and Horvitz, E. (2012). Direct answers for search queries in the long tail. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 237–246. ACM.

[Bi et al., 2014] Bi, B., Tian, Y., Sismanis, Y., Balmin, A., and Cho, J. (2014). Scalable topic-specific influence analysis on microblogs. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 513–522. ACM.

[Bi and Zhang, 2004] Bi, J. and Zhang, T. (2004). Support vector classification with input data uncertainty. nips.

[Bozzon et al., 2012] Bozzon, A., Catallo, I., Ciceri, E., Fraternali, P., Martinenghi, D., Tagliasacchi, M., and Tagliasacchi, M. (2012). A framework for crowdsourced multimedia processing and querying. In *CrowdSearch*, pages 42–47.

[Brabham, 2008] Brabham, D. C. (2008). Crowdsourcing as a model for problem solving an introduction and cases. *Convergence: the international journal of research into new media technologies*, 14(1):75–90.

[Briem et al., 2002] Briem, G. J., Benediktsson, J. A., and Sveinsson, J. R. (2002). Multiple classifiers applied to multisource remote sensing data. *Geoscience and Remote Sensing, IEEE Transactions on*, 40(10):2291–2299.

[Cataldi and Aufaure, 2014] Cataldi, M. and Aufaure, M.-A. (2014). The 10 million follower fallacy: audience size does not prove domain-influence on twitter. *Knowledge and Information Systems*, pages 1–22.

[Cha et al., 2010] Cha, M., Haddadi, H., Benevenuto, F., and Gummadi, P. K. (2010). Measuring user influence in twitter: The million follower fallacy. *ICWSM*, 10:10–17.

[Cha et al., 2009] Cha, M., Kwak, H., Rodriguez, P., Ahn, Y.-Y., and Moon, S. (2009). Analyzing the video popularity characteristics of large-scale user generated content systems. *IEEE/ACM Trans. Netw.*, 17(5):1357–1370.

[Chakrabarti et al., 2002] Chakrabarti, S., Punera, K., and Subramanyam, M. (2002). Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th international conference on World Wide Web*, pages 148–159. ACM.

[Chakrabarti et al., 1999] Chakrabarti, S., Van den Berg, M., and Dom, B. (1999). Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623–1640.

[Chan and Misra, 1990] Chan, K. K. and Misra, S. (1990). Characteristics of the opinion leader: A new dimension. *Journal of advertising*, 19(3):53–60.

[Charalabidis et al., 2013] Charalabidis, Y., Karkaletsis, V., Triantafillou, A., Androutsopoulou, A., Loukis, E., and Grigoriou, P. (2013). Requirements and architecture of a passive crowdsourcing environment. In *EGOV/ePart Ongoing Research*, pages 208–217.

[Chen et al., 2014] Chen, C., Gao, D., Li, W., and Hou, Y. (2014). Inferring topic-dependent influence roles of twitter users. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1203–1206. ACM.

[Chen et al., 2010] Chen, W., Wang, C., and Wang, Y. (2010). Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1029–1038. ACM.

[Chicco et al., 2014] Chicco, D., Ciceri, E., and Masseroli, M. (2014). Correlation of gene function annotation lists through enhanced spearman and kendall measures. *Proceedings of CIBB*, 2:1.

[Chklovski, 2003] Chklovski, T. (2003). Learner: a system for acquiring commonsense knowledge by analogy. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 4–12. ACM.

[Cho et al., 1998] Cho, J., Garcia-Molina, H., and Page, L. (1998). Efficient crawling through url ordering. *Computer Networks and ISDN Systems*, 30(1):161–172.

[Consortium, 2001] Consortium (2001). The gene ontology consortium: Creating the gene ontology resource: Design and implementation. 11:1425–1433.

[Cook et al., 2005] Cook, K. S., Yamagishi, T., Cheshire, C., Cooper, R., Matsuda, M., and Mashima, R. (2005). Trust building via risk taking: A cross-societal experiment. *Social Psychology Quarterly*, 68(2):121–142.

[Cooper et al., 2010] Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., Popović, Z., et al. (2010). Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760.

[Crandall et al., 2009] Crandall, D. J., Backstrom, L., Huttenlocher, D., and Kleinberg, J. (2009). Mapping the world's photos. In *Proceedings of the 18th international conference on World wide web*, pages 761–770. ACM.

[Das Sarma et al., 2014] Das Sarma, A., Parameswaran, A., Garcia-Molina, H., and Halevy, A. (2014). Crowd-powered find algorithms. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 964–975. IEEE.

[Davidson et al., 2013] Davidson, S. B. et al. (2013). Using the crowd for top-k and group-by queries. In *ICDT '13*, pages 225–236.

[De Bra et al., 1994] De Bra, P., Houben, G.-J., Kornatzky, Y., and Post, R. (1994). Information retrieval in distributed hypertexts. In *RIAO*, pages 481–493.

[De Choudhury et al., 2011a] De Choudhury, M., Counts, S., and Czerwinski, M. (2011a). Find me the right content! diversity-based sampling of social media spaces for topic-centric search. In *ICWSM*.

[De Choudhury et al., 2011b] De Choudhury, M., Counts, S., and Czerwinski, M. (2011b). Identifying relevant social media content: leveraging information diversity and user cognition. In *Proceedings of the 22nd ACM conference on Hypertext and hypermedia*, pages 161–170. ACM.

[Dias et al., 2010] Dias, Z., Rocha, A., and Goldenstein, S. (2010). First steps toward image phylogeny. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6.

[Dias et al., 2011] Dias, Z., Rocha, A., and Goldenstein, S. (2011). Video phylogeny: Recovering near-duplicate video relationships. In *Information Forensics and Security (WIFS), 2011 IEEE International Workshop on*, pages 1–6.

[Doan et al., 2011] Doan, A., Ramakrishnan, R., and Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96.

[Domingos and Richardson, 2001] Domingos, P. and Richardson, M. (2001). Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM.

[Downs et al., 2010] Downs, J. S., Holbrook, M. B., Sheng, S., and Cranor, L. F. (2010). Are your participants gaming the system?: screening mechanical turk workers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2399–2402. ACM.

[Ehrig and Maedche, 2003] Ehrig, M. and Maedche, A. (2003). Ontology-focused crawling of web documents. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 1174–1178. ACM.

[Eickhoff and de Vries, 2011] Eickhoff, C. and de Vries, A. (2011). How crowdsourcable is your task. In *Proceedings of the workshop on crowdsourcing for search and data mining (CSDM) at the fourth ACM international conference on web search and data mining (WSDM)*, pages 11–14.

[Ellison et al., 2007] Ellison, N. B. et al. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230.

[Estellés-Arolas and González-Ladrón-de Guevara, 2012] Estellés-Arolas, E. and González-Ladrón-de Guevara, F. (2012). Towards an integrated crowdsourcing definition. *Journal of Information science*, 38(2):189–200.

[Fan et al., 2014] Fan, J. et al. (2014). A hybrid machine-crowdsourcing system for matching web tables. *ICDE*.

[Faradani et al., 2011] Faradani, S., Hartmann, B., and Ipeirotis, P. G. (2011). What's the right price? pricing tasks for finishing on time. *Human Computation*, 11.

[Fayyad et al., 1996] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., et al. (1996). Knowledge discovery and data mining: Towards a unifying framework. In *KDD*, volume 96, pages 82–88.

[Fedorov et al., 2014] Fedorov, R., Fraternali, P., and Tagliasacchi, M. (2014). Snow phenomena modeling through online public media. In *Image Processing, 2014 IEEE International Conference on*, pages 2179–2181. IEEE.

[Fedorov et al., 2013] Fedorov, R., Martinenghi, D., Tagliasacchi, M., and Castelletti, A. (2013). Exploiting user generated content for mountain peak detection.

[Figueiredo et al., 2011] Figueiredo, F., Benevenuto, F., and Almeida, J. M. (2011). The tube over time: characterizing popularity growth of youtube videos. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 745–754. ACM.

[Finin et al., 2010] Finin, T., Murnane, W., Karandikar, A., Keller, N., Martineau, J., and Dredze, M. (2010). Annotating named entities in twitter data with crowdsourcing. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 80–88. Association for Computational Linguistics.

[Franklin et al., 2011] Franklin, M. J., Kossmann, D., Kraska, T., Ramesh, S., and Xin, R. (2011). Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 61–72. ACM.

[Gallagher et al., 2009] Gallagher, A., Joshi, D., Yu, J., and Luo, J. (2009). Geo-location inference from image content and user tags. In *Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on*, pages 55–62. IEEE.

[Galli et al., 2012] Galli, L., Fraternali, P., Martinenghi, D., Tagliasacchi, M., and Novak, J. (2012). A draw-and-guess game to segment images. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 914–917.

[Gao et al., 2011] Gao, H., Barbier, G., Goolsby, R., and Zeng, D. (2011). Harnessing the crowdsourcing power of social media for disaster relief. Technical report, DTIC Document.

[Genc et al., 2011] Genc, Y., Sakamoto, Y., and Nickerson, J. V. (2011). Discovering context: classifying tweets through a semantic transform based on wikipedia. In *Foundations of Augmented Cognition. Directing the Future of Adaptive Systems*, pages 484–492. Springer.

[Ghiassi et al., 2013] Ghiassi, M., Skinner, J., and Zimbra, D. (2013). Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Systems with Applications: An International Journal*, 40(16):6266–6282.

[Ghosh et al., 2013] Ghosh, S., Zafar, M. B., Bhattacharya, P., Sharma, N., Ganguly, N., and Gummadi, K. (2013). On sampling the wisdom of crowds: Random vs. expert sampling of the twitter stream. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 1739–1744. ACM.

[Gomez Rodriguez et al., 2010] Gomez Rodriguez, M., Leskovec, J., and Krause, A. (2010). Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1019–1028. ACM.

[Goodchild and Glennon, 2010] Goodchild, M. F. and Glennon, J. A. (2010). Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, 3(3):231–241.

[Google, 2014] Google (2014). Google list of banned words. `https://gist.github.com/jamiew/1112488`.

[Goolsby, 2010] Goolsby, R. (2010). Social media as crisis platform: The future of community maps/crisis maps. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 1(1):7.

[Gou et al., 2010] Gou, L., Zhang, X. L., Chen, H.-H., Kim, J.-H., and Giles, C. L. (2010). Social network document ranking. In *Proceedings of the 10th annual joint conference on Digital libraries*, pages 313–322. ACM.

[Goyal et al., 2010] Goyal, A., Bonchi, F., and Lakshmanan, L. V. (2010). Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM.

[Grady and Lease, 2010] Grady, C. and Lease, M. (2010). Crowdsourcing document relevance assessment with mechanical turk. In *Proceedings of the NAACL HLT 2010 workshop on creating speech and language data with Amazon's mechanical turk*, pages 172–179. Association for Computational Linguistics.

[Gruber, 2008] Gruber, T. (2008). Collective knowledge systems: Where the social web meets the semantic web. *Web semantics: science, services and agents on the World Wide Web*, 6(1):4–13.

[Guo et al., 2012] Guo, S. et al. (2012). So who won?: Dynamic max discovery with the crowd. In *SIGMOD '12*, pages 385–396.

[Gupta et al., 2014] Gupta, M., Li, R., and Chang, K. C.-C. (2014). Towards a social media analytics platform: Event detection and user profiling for twitter. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web Companion*, WWW Companion '14, pages 193–194, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Hannon et al., 2010] Hannon, J., Bennett, M., and Smyth, B. (2010). Recommending twitter users to follow using content and collaborative filtering approaches. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 199–206. ACM.

[Harris, 2011a] Harris, C. (2011a). You're hired! an examination of crowdsourcing incentive models in human resource tasks. In *Proceedings of the Workshop on Crowdsourcing for Search and Data Mining (CSDM) at the Fourth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 15–18.

[Harris, 2011b] Harris, C. G. (2011b). Dirty deeds done dirt cheap: a darker side to crowdsourcing. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 1314–1317. IEEE.

[Hart et al., 1968] Hart, P. et al. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE TSSC*, 4(2):100–107.

[Hartigan, 1975] Hartigan, J. A. (1975). Clustering algorithms.

[Hays and Efros, 2008] Hays, J. and Efros, A. A. (2008). Im2gps: estimating geographic information from a single image. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE.

[Heer and Bostock, 2010] Heer, J. and Bostock, M. (2010). Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 203–212. ACM.

[Heimerl et al., 2012] Heimerl, K., Gawalt, B., Chen, K., Parikh, T., and Hartmann, B. (2012). Communitysourcing: engaging local crowds to perform expert work via physical kiosks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1539–1548. ACM.

[Hersovici et al., 1998] Hersovici, M., Jacovi, M., Maarek, Y. S., Pelleg, D., Shtalhaim, M., and Ur, S. (1998). The shark-search algorithm. an application: tailored web site mapping. *Computer Networks and ISDN Systems*, 30(1):317–326.

[Howe, 2008] Howe, J. (2008). *Crowdsourcing: How the power of the crowd is driving the future of business.* Random House.

[Howe and Robinson, 2006] Howe, J. and Robinson, M. (2006). Crowdsourcing: A definition.

[Huang, 2007] Huang, F. (2007). Building social trust: A human-capital approach. *Journal of Institutional and Theoretical Economics (JITE)/Zeitschrift für die gesamte Staatswissenschaft*, pages 552–573.

[Huang et al., 2013] Huang, P.-Y., Liu, H.-Y., Chen, C.-H., and Cheng, P.-J. (2013). The impact of social diversity and dynamic influence propagation for identifying influencers in social networks. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2013 IEEE/WIC/ACM International Joint Conferences on*, volume 1, pages 410–416. IEEE.

[Huberman et al., 2009] Huberman, B. A., Romero, D. M., and Wu, F. (2009). Crowdsourcing, attention and productivity. *Journal of Information Science.*

[Hung et al., 2013] Hung, N. et al. (2013). On leveraging crowdsourcing techniques for schema matching networks. In *Database Systems for Advanced Applications*, LNCS 7826, pages 139–154.

[Ikeda et al., 2013] Ikeda, K., Hattori, G., Ono, C., Asoh, H., and Higashino, T. (2013). Twitter user profiling based on text and community mining for market analysis. *Knowledge-Based Systems*, 51:35–47.

[Ilyas et al., 2008] Ilyas, I. F., Beskales, G., and Soliman, M. A. (2008). A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58.

[Ipeirotis, 2010] Ipeirotis, P. G. (2010). Demographics of mechanical turk.

[Ipeirotis and Gabrilovich, 2014] Ipeirotis, P. G. and Gabrilovich, E. (2014). Quizz: Targeted crowdsourcing with a billion (potential) users. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 143–154, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Jabeur et al., 2012] Jabeur, L. B., Tamine, L., and Boughanem, M. (2012). Active microbloggers: identifying influencers, leaders and discussers in microblogging networks. In *String Processing and Information Retrieval*, pages 111–117. Springer.

[Järvelin and Kekäläinen, 2002] Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

[Joglekar et al., 2013] Joglekar, M., Garcia-Molina, H., and Parameswaran, A. (2013). Evaluating the crowd with confidence. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 686–694. ACM.

[Kaplan and Haenlein, 2010] Kaplan, A. M. and Haenlein, M. (2010). Users of the world, unite! the challenges and opportunities of social media. *Business horizons*, 53(1):59–68.

[Karp, 1998] Karp, P. D. (1998). What we do not know about sequence analysis and sequence databases. *Bioinformatics*, 4(9):753–754.

[Katz and Lazarsfeld, 1970] Katz, E. and Lazarsfeld, P. F. (1970). *Personal Influence, The part played by people in the flow of mass communications.* Transaction Publishers.

[Kazai, 2010] Kazai, G. (2010). An exploration of the influence that task parameters have on the performance of crowds. *Proceedings of the CrowdConf*, 2010.

[Kazienko and Musial, 2007] Kazienko, P. and Musial, K. (2007). On utilising social networks to discover representatives of human communities. *Int. J. Intell. Inf. Database Syst.*, 1(3/4):293–310.

[Keimel et al., 2012] Keimel, C., Habigt, J., and Diepold, K. (2012). Challenges in crowd-based video quality assessment. In *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*, pages 13–18. IEEE.

[Kempe et al., 2003] Kempe, D., Kleinberg, J., and Tardos, É. (2003). Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM.

[Kendall, 1938] Kendall, M. G. (1938). A new measure of rank correlation. *Biometrika*, pages 81–93.

[Kender et al., 2010] Kender, J. R., Hill, M. L., Natsev, A. P., Smith, J. R., and Xie, L. (2010). Video genetics: a case study from youtube. In *Proceedings of the international conference on Multimedia*, pages 1253–1258. ACM.

[Kennedy and Naaman, 2008] Kennedy, L. S. and Naaman, M. (2008). Generating diverse and representative image search results for landmarks. In *Proceedings of the 17th international conference on World Wide Web*, pages 297–306. ACM.

[Khosla et al., 2014] Khosla, A., Das Sarma, A., and Hamid, R. (2014). What makes an image popular? In *Proceedings of the 23rd international conference on World wide web*, pages 867–876. International World Wide Web Conferences Steering Committee.

[Kirkels and Post, 2013] Kirkels, Y. and Post, G. (2013). Crowd voting, a method tested in favour of entrepreneurship. In *XXIV ISPIM Conference—Innovating in Global Markets: Challenges for Sustainable Growth Conference, Helsinki. Lappeenranta: Lappeenranta University of Technology Press.*

[Kiss and Bichler, 2008] Kiss, C. and Bichler, M. (2008). Identification of influencers—measuring influence in customer networks. *Decision Support Systems*, 46(1):233–253.

[Kittler et al., 1998] Kittler, J., Hatef, M., Duin, R. P., and Matas, J. (1998). On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239.

[Kong and Feng, 2011] Kong, S. and Feng, L. (2011). A tweet-centric approach for topic-specific author ranking in micro-blog. In *Advanced Data Mining and Applications*, pages 138–151. Springer.

[Kontopoulos et al., 2013] Kontopoulos, E., Berberidis, C., Dergiades, T., and Bassiliades, N. (2013). Ontology-based sentiment analysis of twitter posts. *Expert systems with applications*, 40(10):4065–4074.

[Krumm et al., 2008] Krumm, J., Davies, N., and Narayanaswami, C. (2008). User-generated content. *IEEE Pervasive Computing*, 7(4):10–11.

[Kumar and Vassilvitskii, 2010] Kumar, R. and Vassilvitskii, S. (2010). Generalized distances between rankings. In *WWW '10*, pages 571–580.

[Kwak et al., 2010] Kwak, H., Lee, C., Park, H., and Moon, S. (2010). What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM.

[Kywe et al., 2012] Kywe, S., Hoang, T.-A., Lim, E.-P., and Zhu, F. (2012). On recommending hashtags in twitter networks. In Aberer, K., Flache, A., Jager, W., Liu, L., Tang, J., and Guéret, C., editors, *Social Informatics*, volume 7710 of *Lecture Notes in Computer Science*, pages 337–350. Springer Berlin Heidelberg.

[Le et al., 2010] Le, J., Edmonds, A., Hester, V., and Biewald, L. (2010). Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *SIGIR 2010 workshop on crowdsourcing for search evaluation*, pages 21–26.

[Le Callet and Autrusseau, 2005] Le Callet, P. and Autrusseau, F. (2005). Subjective quality assessment IRCCyN/IVC database. `http://www.irccyn.ec-nantes.fr/ivcdb/`.

[Lee et al., 2011] Lee, K., Palsetia, D., Narayanan, R., Patwary, M. M. A., Agrawal, A., and Choudhary, A. (2011). Twitter trending topic classification. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 251–258. IEEE.

[Lehmann et al., 2012] Lehmann, J., Gonçalves, B., Ramasco, J. J., and Cattuto, C. (2012). Dynamical classes of collective attention in twitter. In *Proceedings of the 21st international conference on World Wide Web*, pages 251–260. ACM.

[Lerman and Hogg, 2010] Lerman, K. and Hogg, T. (2010). Using a model of social dynamics to predict popularity of news. In *Proceedings of the 19th international conference on World wide web*, pages 621–630. ACM.

[Leskovec et al., 2007] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., and Glance, N. (2007). Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM.

[Levine and Kurzban, 2006] Levine, S. S. and Kurzban, R. (2006). Explaining clustering in social networks: Towards an evolutionary theory of cascading benefits. *Managerial and Decision Economics*, 27(2-3):173–187.

[Li and Deshpande, 2010] Li, J. and Deshpande, A. (2010). Ranking continuous probabilistic datasets. *PVLDB*, 3(1-2):638–649.

[Li et al., 2009] Li, J. et al. (2009). A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513.

[Li et al., 2011] Li, J., Saha, B., and Deshpande, A. (2011). A unified approach to ranking in probabilistic databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 20(2):249–275.

[Lian et al., 2012] Lian, J., Liu, Y., Zhang, Z.-J., Cheng, J.-J., and Xiong, F. (2012). Analysis of user's weight in microblog network based on user influence and active degree. *Journal of Electronic Science and Technology*, 10(4).

[Licklider, 1960] Licklider, J. C. R. (1960). Man-computer symbiosis. *Human Factors in Electronics, IRE Transactions on*, (1):4–11.

[Little et al., 2010] Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. (2010). Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 57–66. ACM.

[Liu et al., 2012] Liu, X., Lu, M., Ooi, B. C., Shen, Y., Wu, S., and Zhang, M. (2012). Cdas: a crowdsourcing data analytics system. *Proceedings of the VLDB Endowment*, 5(10):1040–1051.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee.

[Lu et al., 2012] Lu, D., Li, Q., and Liao, S. S. (2012). A graph-based action network framework to identify prestigious members through member's prestige evolution. *Decision Support Systems*, 53(1):44–54.

[Luz et al., 2014] Luz, N., Silva, N., and Novais, P. (2014). A survey of task-oriented crowdsourcing. *Artificial Intelligence Review*, pages 1–27.

[Lynch et al., 2014] Lynch, M., Freelon, D., and Aday, S. (2014). Syria's socially mediated civil war. *United States Institute Of Peace.*

[Ma et al., 2011] Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011). Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296. ACM.

[Maheswaran et al., 2007] Maheswaran, M., Tang, H. C., and Ghunaim, A. (2007). Towards a gravity-based trust model for social networking systems. In *Distributed Computing Systems Workshops, 2007. ICDCSW'07. 27th International Conference on*, pages 24–24. IEEE.

[Malone et al., 2009] Malone, T. W., Laubacher, R., and Dellarocas, C. (2009). Harnessing crowds: Mapping the genome of collective intelligence.

[Marcus et al., 2011a] Marcus, A. et al. (2011a). Crowdsourced databases: Query pro- cessing with people. In *CIDR '11*, pages 211–214.

[Marcus et al., 2011b] Marcus, A. et al. (2011b). Human-powered sorts and joins. *PVLDB*, 5(1):13–24.

[Marcus et al., 2012] Marcus, A., Karger, D., Madden, S., Miller, R., and Oh, S. (2012). Counting with the crowd. *Proceedings of the VLDB Endowment*, 6(2):109–120.

[Markowsky, 2013] Markowsky, G. (2013). Crowdsourcing, big data and homeland security. In *Technologies for Homeland Security (HST), 2013 IEEE International Conference on*, pages 772–778. IEEE.

[Mason and Watts, 2010] Mason, W. and Watts, D. J. (2010). Financial incentives and the performance of crowds. *ACM SigKDD Explorations Newsletter*, 11(2):100–108.

[Mellebeek et al., 2010] Mellebeek, B., Benavent, F., Grivolla, J., Codina, J., Costa-Jussa, M. R., and Banchs, R. (2010). Opinion mining of spanish customer comments with non-expert annotations on mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 114–121. Association for Computational Linguistics.

[Merton, 1957] Merton, R. K. (1957). Patterns of influence: Local and cosmopolitan influentials. *Social theory and social structure*, pages 387–420.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Molm et al., 2000] Molm, L. D., Takahashi, N., and Peterson, G. (2000). Risk and trust in social exchange: An experimental test of a classical proposition. *American Journal of Sociology*, pages 1396–1427.

[Moreno et al., 2009] Moreno, A., de la Rosa, J. L., Szymanski, B. K., and Barcenas, J. M. (2009). Reward system for completing faqs. In *CCIA*, pages 361–370.

[Nam et al., 2009] Nam, K. K., Ackerman, M. S., and Adamic, L. A. (2009). Questions in, knowledge in?: a study of naver's question answering community. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 779–788. ACM.

[Ngo et al., 2013] Ngo, C.-W., Xu, C., Kraaij, W., and Saddik, A. E. (2013). Web-scale near-duplicate search: Techniques and applications. *IEEE MultiMedia*, 20(3):10–12.

[Nieke et al., 2014] Nieke, C., Güntzer, U., and Balke, W.-T. (2014). Topcrowd-efficient crowd-enabled top-k retrieval on incomplete data.

[Nitta et al., 2014] Nitta, N., Kumihashi, Y., Kato, T., and Babaguchi, N. (2014). Real-world event detection using flickr images. In *MultiMedia Modeling*, pages 307–314. Springer.

[Novak, 2004] Novak, B. (2004). A survey of focused web crawling algorithms.

[Nowak and Rüger, 2010] Nowak, S. and Rüger, S. (2010). How reliable are annotations via crowdsourcing: a study about inter-annotator agreement for multi-label image annotation. In *Proceedings of the international conference on Multimedia information retrieval*, pages 557–566. ACM.

[Nucci et al., 2013] Nucci, M., Tagliasacchi, M., and Tubaro, S. (2013). A phylogenetic analysis of near-duplicate audio tracks. In *Multimedia Signal Processing (MMSP), 2013 IEEE 15th International Workshop on*, pages 099–104.

[O'reilly, 2009] O'reilly, T. (2009). *What is web 2.0.* " O'Reilly Media, Inc.".

[Page et al., 1999] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web.

[Pal and Counts, 2011] Pal, A. and Counts, S. (2011). Identifying topical authorities in microblogs. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 45–54. ACM.

[Paolacci et al., 2010] Paolacci, G., Chandler, J., and Ipeirotis, P. G. (2010). Running experiments on amazon mechanical turk. *Judgment and Decision making*, 5(5):411–419.

[Parameswaran et al., 2014] Parameswaran, A., Boyd, S., Garcia-Molina, H., Gupta, A., Polyzotis, N., and Widom, J. (2014). Optimal crowd-powered rating and filtering algorithms. *Proceedings Very Large Data Bases (VLDB).*

[Parameswaran et al., 2011] Parameswaran, A. et al. (2011). Human-assisted graph search: It's okay to ask questions. *PVLDB*, 4(5):267–278.

[Parameswaran and Polyzotis, 2011] Parameswaran, A. and Polyzotis, N. (2011). Answering queries using humans, algorithms and databases. In *CIDR '11.*

[Parameswaran et al., 2012] Parameswaran, A. G., Garcia-Molina, H., Park, H., Polyzotis, N., Ramesh, A., and Widom, J. (2012). Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 361–372. ACM.

[Parameswaran and Whinston, 2007] Parameswaran, M. and Whinston, A. B. (2007). Social computing: An overview. *Communications of the Association for Information Systems*, 19(1):37.

[Polychronopoulos et al., 2013] Polychronopoulos, V. et al. (2013). Human-powered top-k lists. In *WebDB*, pages 25–30.

[Porter, 2010] Porter, J. (2010). *Designing for the social web.* Peachpit Press.

[Quercia et al., 2011] Quercia, D., Ellis, J., Capra, L., and Crowcroft, J. (2011). In the mood for being influential on twitter. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 307–314. IEEE.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106. 10.1007/BF00116251.

[Quinn and Bederson, 2011] Quinn, A. J. and Bederson, B. B. (2011). Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1403–1412. ACM.

[Rashtchian et al., 2010] Rashtchian, C., Young, P., Hodosh, M., and Hockenmaier, J. (2010). Collecting image annotations using amazon's mechanical turk. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, pages 139–147. Association for Computational Linguistics.

[Richardson and Domingos, 2002] Richardson, M. and Domingos, P. (2002). Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70. ACM.

[Rogers Everett, 1995] Rogers Everett, M. (1995). Diffusion of innovations. *New York*.

[Romero et al., 2011] Romero, D. M., Galuba, W., Asur, S., and Huberman, B. A. (2011). Influence and passivity in social media. In *Machine learning and knowledge discovery in databases*, pages 18–33. Springer.

[Ross et al., 2010] Ross, J., Irani, L., Silberman, M., Zaldivar, A., and Tomlinson, B. (2010). Who are the crowdworkers?: shifting demographics in mechanical turk. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 2863–2872. ACM.

[Saez-Trumper et al., 2012] Saez-Trumper, D., Comarela, G., Almeida, V., Baeza-Yates, R., and Benevenuto, F. (2012). Finding trendsetters in information networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1014–1022. ACM.

[Sakaki et al., 2010] Sakaki, T., Okazaki, M., and Matsuo, Y. (2010). Earthquake shakes twitter users: Real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 851–860, New York, NY, USA. ACM.

[Sakaki et al., 2013] Sakaki, T., Okazaki, M., and Matsuo, Y. (2013). Tweet analysis for real-time event detection and earthquake reporting system development. *Knowledge and Data Engineering, IEEE Transactions on*, 25(4):919–931.

[Schenkel et al., 2008] Schenkel, R., Crecelius, T., Kacimi, M., Michel, S., Neumann, T., Parreira, J. X., and Weikum, G. (2008). Efficient top-k querying over social-tagging networks. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 523–530. ACM.

[Scripps et al., 2009] Scripps, J., Tan, P.-N., and Esfahanian, A.-H. (2009). Measuring the effects of preprocessing decisions and network forces in dynamic network analysis. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 747–756. ACM.

[Serdyukov et al., 2009] Serdyukov, P., Murdock, V., and Van Zwol, R. (2009). Placing flickr photos on a map. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 484–491. ACM.

[Sheikh et al., 2006] Sheikh, H. R., Sabir, M. F., and Bovik, A. C. (2006). A statistical evaluation of recent full reference image quality assessment algorithms. *Image Processing, IEEE Transactions on*, 15(11):3440–3451.

[Sheng et al., 2008] Sheng, V. S., Provost, F., and Ipeirotis, P. G. (2008). Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 614–622. ACM.

[Shetty and Adibi, 2005] Shetty, J. and Adibi, J. (2005). Discovering important nodes through graph entropy the case of enron email database. In *Proceedings of the 3rd international workshop on Link discovery*, pages 74–81. ACM.

[Silva et al., 2013] Silva, A., Guimarães, S., Meira Jr, W., and Zaki, M. (2013). Profilerank: finding relevant content and influential users based on information diffusion. In *Proceedings of the 7th Workshop on Social Network Mining and Analysis*, page 2. ACM.

[Singh et al., 2002] Singh, P., Lin, T., Mueller, E. T., Lim, G., Perkins, T., and Zhu, W. L. (2002). Open mind common sense: Knowledge acquisition from the general public. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, pages 1223–1237. Springer.

[Smith, 2009] Smith, T. (2009). The social media revolution. *International journal of market research*, 51(4):559–561.

[Snow et al., 2008] Snow, R., O'Connor, B., Jurafsky, D., and Ng, A. Y. (2008). Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 254–263. Association for Computational Linguistics.

[Soliman et al., 2011] Soliman, M. et al. (2011). Ranking with uncertain scoring functions: semantics and sensitivity measures. In *SIGMOD '11*, pages 805–816.

[Soliman and Ilyas, 2009] Soliman, M. and Ilyas, I. (2009). Ranking with uncertain scores. In *ICDE '09.*, pages 317 –328.

[Soliman et al., 2010] Soliman, M., Ilyas, I., and Ben-David, S. (2010). Supporting ranking queries on uncertain and incomplete data. *The VLDB Journal*, 19:477–501. 10.1007/s00778-009-0176-8.

[Soliman et al., 2007] Soliman, M. A., Ilyas, I. F., and Chang, K. C.-C. (2007). Urank: formulation and efficient evaluation of top-k queries in uncertain databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 1082–1084, New York, NY, USA. ACM.

[Soliman et al., 2008] Soliman, M. A., Ilyas, I. F., and Chang, K. C.-C. (2008). Probabilistic top-k and ranking-aggregate queries. *ACM Trans. Database Syst.*, 33(3):13:1–13:54.

[Sriram et al., 2010] Sriram, B., Fuhry, D., Demir, E., Ferhatosmanoglu, H., and Demirbas, M. (2010). Short text classification in twitter to improve information filtering. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 841–842. ACM.

[Sun and Ng, 2013] Sun, B. and Ng, V. T. (2013). Identifying influential users by their postings in social networks. In *Ubiquitous Social Media Analysis*, pages 128–151. Springer.

[Tan et al., 2010] Tan, C., Tang, J., Sun, J., Lin, Q., and Wang, F. (2010). Social action tracking via noise tolerant time-varying factor graphs. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1049–1058. ACM.

[The Washington Post, 2011] The Washington Post (2011). Tweets move faster than earthquakes.

[Tsang et al., 2011] Tsang, S., Kao, B., Yip, K., Ho, W.-S., and Lee, S. D. (2011). Decision trees for uncertain data. *Knowledge and Data Engineering, IEEE Transactions on*, 23(1):64 –78.

[van de Sande et al., 2010] van de Sande, K. E. A., Gevers, T., and Snoek, C. G. M. (2010). Evaluating color descriptors for object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1582–1596.

[Van Kleek et al., 2012] Van Kleek, M., Smith, D., Stranders, R., et al. (2012). Twiage: a game for finding good advice on twitter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 889–898. ACM.

[Venetis et al., 2012] Venetis, P. et al. (2012). Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998.

[Venetis and Garcia-Molina, 2012] Venetis, P. and Garcia-Molina, H. (2012). Quality control for comparison microtasks. In *Proceedings of the First International Workshop on Crowdsourcing and Data Mining*, pages 15–21. ACM.

[Vesdapunt et al., 2014] Vesdapunt, N., Bellare, K., and Dalvi, N. (2014). Crowdsourcing algorithms for entity resolution. *Proceedings of the VLDB Endowment*, 7(12).

[von Ahn, 2006] von Ahn, L. (2006). Games with a purpose. *Computer*, 39(6):92–94.

[Von Ahn, 2009] Von Ahn, L. (2009). Human computation. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 418–419. IEEE.

[Von Ahn and Dabbish, 2008] Von Ahn, L. and Dabbish, L. (2008). Designing games with a purpose. *Communications of the ACM*, 51(8):58–67.

[Von Ahn et al., 2008] Von Ahn, L., Maurer, B., McMillen, C., Abraham, D., and Blum, M. (2008). recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468.

[Vonikakis et al., 2014] Vonikakis, V., Subramanian, R., Arnfred, J., and Winkler, S. (2014). Modeling image appeal based on crowd preferences for automated person-centric collage creation.

[Wang et al., 2013a] Wang, J., Korayem, M., and Crandall, D. (2013a). Observing the natural world with flickr. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 452–459.

[Wang et al., 2012] Wang, J., Kraska, T., Franklin, M. J., and Feng, J. (2012). Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494.

[Wang et al., 2013b] Wang, J., Li, G., Kraska, T., Franklin, M. J., and Feng, J. (2013b). Leveraging transitive relations for crowdsourced joins. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 229–240, New York, NY, USA. ACM.

[Wang et al., 2013c] Wang, Y., Li, X., Li, X., and Wang, Y. (2013c). A survey of queries over uncertain data. *Knowledge and information systems*, 37(3):485–530.

[Wang et al., 2004] Wang, Z. et al. (2004). Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612.

[Watts and Dodds, 2007] Watts, D. J. and Dodds, P. S. (2007). Influentials, networks, and public opinion formation. *Journal of consumer research*, 34(4):441–458.

[Weimann, 1994] Weimann, G. (1994). *The influentials: People who influence people.* SUNY Press.

[Weitzel et al., 2012] Weitzel, L., Quaresma, P., and de Oliveira, J. P. M. (2012). Measuring node importance on twitter microblogging. In *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, page 11. ACM.

[Weng et al., 2010] Weng, J., Lim, E.-P., Jiang, J., and He, Q. (2010). Twitterrank: finding topic-sensitive influential twitterers. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 261–270. ACM.

[Woźniak et al., 2014] Woźniak, M., Graña, M., and Corchado, E. (2014). A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3–17.

[Xintong et al., 2014] Xintong, G., Hongzhi, W., Song, Y., and Hong, G. (2014). Brief survey of crowdsourcing for data mining. *Expert Systems with Applications*, 41(17):7987–7994.

[Xu et al., 1992] Xu, L., Krzyzak, A., and Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(3):418–435.

[Yang et al., 2008] Yang, J., Adamic, L. A., and Ackerman, M. S. (2008). Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *Proceedings of the 9th ACM conference on Electronic commerce*, pages 246–255. ACM.

[Yang et al., 2007] Yang, J., Jiang, Y.-G., Hauptmann, A. G., and Ngo, C.-W. (2007). Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the International Workshop on Workshop on Multimedia Information Retrieval*, MIR '07, pages 197–206, New York, NY, USA. ACM.

[Yerva et al., 2011] Yerva, S. R., Miklós, Z., and Aberer, K. (2011). What have fruits to do with technology?: the case of orange, blackberry and apple. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, page 48. ACM.

[Yin et al., 2010] Yin, P., Lee, W.-C., and Lee, K. C. (2010). On top-k social web search. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1313–1316. ACM.

[Yu et al., 2005] Yu, H. et al. (2005). Enabling ad-hoc ranking for data retrieval. *ICDE.*

[Yuen et al., 2011] Yuen, M.-C., King, I., and Leung, K.-S. (2011). A survey of crowdsourcing systems. In *Privacy, security, risk and trust (passat), 2011 ieee third international conference on and 2011 ieee third international conference on social computing (socialcom)*, pages 766–773. IEEE.

[Zhai et al., 2014] Zhai, Y., Li, X., Chen, J., Fan, X., and Cheung, W. K. (2014). A novel topical authority-based microblog ranking. In *Web Technologies and Applications*, pages 105–116. Springer.

[Zhang et al., 2013] Zhang, C. J. et al. (2013). Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768.

[Zhang et al., 2014] Zhang, C. J., Zhao, Z., Chen, L., Jagadish, H., and Cao, C. C. (2014). Crowdmatcher: crowd-assisted schema matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 721–724. ACM.

[Zhang et al., 2012] Zhang, H., Korayem, M., Crandall, D. J., and LeBuhn, G. (2012). Mining photo-sharing websites to study ecological phenomena. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 749–758, New York, NY, USA. ACM.

[Zhang, 2005] Zhang, J. B. T. (2005). Support vector classification with input data uncertainty. *Advances in neural information processing systems*, 17:161–169.

[Zhang et al., 2006] Zhang, W. et al. (2006). A trust based framework for secure data aggregation in wireless sensor networks. In *SECON '06*, pages 60–69.

[Zhou and Chen, 2014] Zhou, X. and Chen, L. (2014). Event detection over twitter social media streams. *The VLDB Journal*, 23(3):381–400.