POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO
Master of Science in Computer Engineering

A Context-Independent Algorithm for Annotation Aggregation in a Crowdsourcing Environment

Supervisor: Prof. Marco Tagliasacchi
Assistant Supervisor: Dr. Luca Galli

Master Graduation Thesis by:
Carlo Bernaschina
id. 798478

Academic Year 2013-2014

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO
Corso di Laura Specialistica in Ingegneria Informatica

A Context-Independent Algorithm for Annotation Aggregation in a Crowdsourcing Environment

Relatore: Prof. Marco Tagliasacchi
Correlatore: Dr. Luca Galli

Tesi di laurea di:
Carlo Bernaschina
matr. 798478

Anno Accademico 2013-2014

Abstract

In this work we present an abstract, reusable and context-independent algorithm for annotation aggregation. This algorithm can be instantiated by defining context-dependent operations on the specific annotation type.
We analyze the state of the art in annotation aggregation and identify the main problems and propose solutions in order to deal with them.
We analyze common annotation types an propose instantiations of the algorithm in this cases.
Tests performed on both synthetic and real datasets revealed a reduction up to 70%, with respect to standard approaches, in the number of required annotations given a predefined accuracy level.

Sommario

In questo lavoro proponiamo un algoritmo astratto, riusabile e indipendente dal contesto per effettuare l'aggregazione di annotazioni, il quale può essere istanziato definendo delle operazioni dipendenti dal contesto e in particolare dal tipo di annotazione in esame.
Viene effettuata un'analisi dello stato dell'arte per quanto riguarda l'aggregazione di annotazioni e un identificazione dei principali problemi legati a questa, proponendo delle soluzione in grado di gestirli.
Verranno analizzate le tipologie di annotazione più comuni e verranno proposte istanze dell'algoritmo relative a questi particolari casi.
I test svolti sia su dataset sintetici che reali hanno rilevato una riduzione fino al 70%, rispetto ad approcci standard, per quanto riguarda il numero di annotazioni necessarie per raggiungere un livello di accuratezza prefissato.

Acknowledgements

First of all, I would like to thank Prof. Marco Tagliasacchi, Prof. Piero Fraternali and Dr. Luca Galli for the opportunity to work with them on this interesting topic.
I would like to thank also:

- Roman Fedorov and Elena Donegani for helping me with the preparation of this document.
- All my friends and colleagues (who are too many to be listed) who have helped and contributed to this work.
- My parents (Luisa and Mario), sister (Cristina) and uncles (Angela e Stefano) for the great support and motivation they have given me, and for their enormous patience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work is mainly related to two topics, human computation and data mining. The border between the two disciplines is becoming more and more blurred. This is due to the fact that more and more data mining works on data sets coming from the human computation field and more and more human computation requires data mining techniques in order to filter and extract knowledge from the growing amount of data collected directly or indirectly from users.
The aim of this Chapter is to introduce and to define these concepts, and to state the problem that the presented work tries to solve.

1.1 Human Computation

The discipline of Human Computation, as stated by Denning at al. in January 1989, is:

> *" The discipline of computing is the systematic study of algorithmic processes that describe and transform information: their theory, analysis, design, efficiency, implementation, and application. The fundamental question underlying all computing is 'What can be (efficiently) automated?' "*
>
> Denning at al. [10]

In my opinion the emphasis should be on the word **efficiently**.
Here is were human computation comes in, there are many problems for which there is not an efficient automated solution, or worse there is not an effective automated solution. Nowadays in many of these situations human beings are really effective and efficient in solving the problem.
Classical examples are **visual features**-related problems.

In the past years a great focus was on improving algorithms in order to reduce the gap between computers and human beings.
Human computation though rely on:

> *" a novel approach: constructively channel human brainpower "*
>
> Luis von Ahn [40]

Many different approaches can be addressed by this definition. In the following Chapters we will follow the taxonomy proposed in [16] that is a combination of the ones proposed by Quinn et al. [32] and Fraternali et al. [19].

- Crowdsourcing *[Active Crowdsourcing]: this approach manages the distributed assignment of tasks to an open, undefined and generally large group of executors. The task to be performed by the executors is split into a large number of microtasks (by the work provider or the crowdsourcing system itself) and each microtask is assigned by the system to a work performer, who executes it (usually for a reward of a small amount of money). The crowdsourcing application (defined usually by two interfaces: for the work providers and the work performers) manages the work life cycle: performer assignment, time and price negotiation, result submission and verification, and payment. In addition to the web interface, some platforms offer Application Programming Interfaces (APIs),whereby third parties can integrate the distributed work management functionality into their custom applications. Examples of crowdsourcing solutions are Amazon Mechanical Turk and Microtask.com [19].*
- Games with a Purpose (GWAPs)*: these are a sort of crowsourcing application but with a fundamental difference in user incentive technique: the process of resolving a task is implemented as a game with an enjoyable user experience. Instead of monetary earning, the user motivation in this approach is the gratification of the playing process. GWAPs, and more generally useful applications where the user solves perceptive or cognitive problems without knowing, address task such as adding descriptive tags and recognising objects in images and checking the output of Optical Character Recognition (OCR) for correctness. [19].*
- Social Computing*: a broad scope concept that includes applications and services that facilitate collective action and*

social interaction online with rich exchange of multimedia information and evolution of aggregate knowledge [31]. Instead of crowdsourcing, the purpose is usually not to perform a task. The key distinction between human computation and social computing is that social computing facilitates relatively natural human behavior that happens to be mediated by technology, whereas participation in a human computation is directed primarily by the human computation system [32].

- Collective Intelligence*: if seen as a process, the term can be defined as groups of individuals doing things collectively that seem intelligent [30]. If it is seen instead as the process result, means the knowledge of any kind that is generated (even non consciously and not in explicit form) by the collective intelligence process. Quinn et al. [32] classifies it as the superset of social computing and crowdsourcing, because both are defined in terms of social behavior. The key distinctions between collective intelligence and human computation are the same as with crowdsourcing, but with the additional distinction that collective intelligence applies only when the process depends on a group of participants. It is conceivable that there could be a human computation system with computations performed by a single worker in isolation. This is why part of human computation protrudes outside collective intelligence [32].*
- Data Mining*: this can be defined broadly as the application of specific algorithms for extracting patterns from data [15]. Speaking about human-created data the approach can be seen as extracting the knowledge from a certain result of a collective intelligence process. Creating this knowledge usually is not the goal of the persons that generate it, in fact often they are completely unaware of it (just think that almost everybody contributes to the knowledge of what are the most popular web sites just by visiting them: they open a web site because they need it, not to add a vote to its popularity). Though it is a very important concept in the field of collective intelligence, machine intelligence applied to social science and passive crowdsourcing [...] is a fully automated process by definition, so it is excluded from the area of human computation.*
- Social Mobilization*: this approach deals with social computation problems where the timing and the efficiency is crucial. Examples of this area are safety critical sectors like*

civil protection and disease control.

- Human Sensors*: exploiting the fact that the mobile devices tend to incorporate more and more sensors, this approach deals with a real-time collection of data (of various natures) treating persons with mobile devices as sensors for the data. Examples of these applications are earthquake and other natural disaster monitoring, traffic condition control and pollution monitoring.*

Roman Fedorov [16]

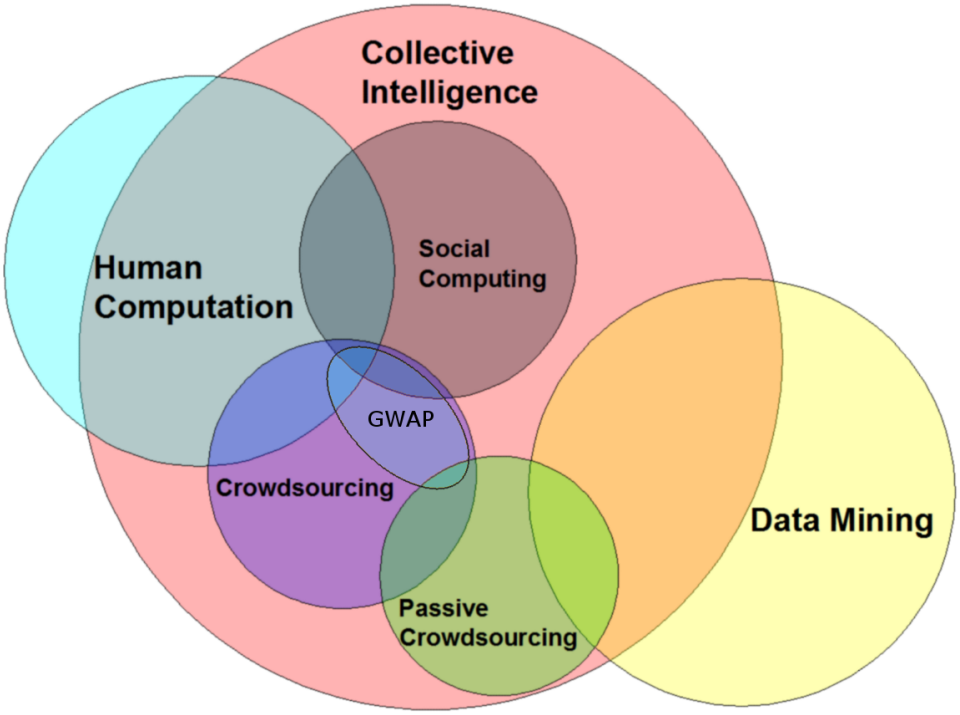

Figure 1.1: Taxonomy of human computation

1.2 Crowdsourcing

Crowdsourcing is one of the methods used to exploit the ability of human beings to solve problems.
In this section we will analyze the main differences between **active** and **passive** crowdsourcing, their similarities and one of the main problems in both of them: the **adversarial behavior**.

1.2.1 Active vs. Passive Crowdsourcing

The main differences between active and passive crowdsourcing systems are related to the engagement of the user.

- **Awareness**

  In **active** systems the user knows what is happening, knows that the tasks that he or she is carrying on are part of a greater problem to solve, even though may or may not be aware of which is the greater problem. An example of this is presented in [12].

  In **passive** systems this information may or may not be known. Passive systems can be hidden inside games (like in the GWAP methodology) or inside other applications or be completely passive and do not require a direct interaction with the user at all. An example of this is presented in [17] [18].

- **Rewarding**

  In **active** systems the user is generally rewarded. In payed systems the reward is a certain amount of money related to the completion of a task (like in Amazon Mechanical Turk [1]), in voluntary systems the user offers its time and knowledge for the benefit of a community (like in Launchpad Translations [2]).

  In **passive** systems the reward depends on the specific case. In GWAP systems the reward is related to gaining points and notoriety in the game community. In systems embedded inside other applications the reward is the possibility to use the application. In other situations there is no reward at all, like in passive systems where the user shares information without knowing that those information will be used by the systems.

- **Controllability**

  Active systems are generally controllable, the task assignment can be scheduled in such a way to obtain better results or exploit users with specific abilities.

  Passive systems are not generally controllable. While in GWAPs it is possible to manage task assignment, even though you cannot force the

user to complete the task, in all the other situations it is not the system to feed the users with tasks but are the users to feed the systems with data. In this situations the flow of information is not controllable and so delays or unavailability of data has to be taken into account.

1.2.2 Adversarial Behavior

As described by Wang et al. [42] one of the great problems related to human computation is the adversarial behavior of some users. In this situations users do not execute the required tasks or feed the system with malicious data. This can be due to many reasons.

- **Maximize Rewards**

  Users of active systems are payed at the completion of the task, while passive systems ones gain points and reputation. In this situation malicious users try to complete the largest number of tasks in the least amount of time, simply by randomly answering or trying to fool the system and complete the task in a fast way.

- **Real Life Advantages**

  In even worse situations users try to poison data in order to gain advantages in the real life. An example are false reviews in recommendation systems.

- **Just For Fun**

  In *Sketchness* [20] (a GWAP designed to assign labels to localized regions in an image), users write obscene phrases, instead of fulfilling the task.

1.3 Problem Statement

The problem addressed in this thesis is the estimation of the accuracy of annotations provided by a set of workers in a crowdsourcing application.
Given a) a set of objects with a specific feature that we require to estimate, b) a set of users called annotators that interact with the system, c) a set of annotations each one of them coming from a known annotator and related to a specific object, our objective is to aggregate the annotations to estimate the real feature of the objects and reduce the impact given by adversarial annotators.
This situation is common to all the systems dealing with users that can not be trusted a priori or users with different accuracy levels.
Some examples are:

- Human assisted object classification (see Section 3.3.1)

- Identification of the regions of interest inside images allowing further analysis in a specific area (see Section 3.3.2)
- Estimation of the bounding box of objects or persons inside and image allowing further investigations e.g. logo detection or person tracking (see Section 3.3.3)
- Ranking of hotel or restaurant in rating systems (see Section 3.3.4)

The proposed algorithm can be also used in the identification of malicious users, by analyzing the consensus between them.

1.4 Document Structure

- In Chapter 2 we will analyze the state of the art in solving the stated problem.
- In Chapter 3 we will first present and analyze a general algorithm for aggregation of annotations regardless to their kind and than we will present instantiations of the algorithm in common situations (annotation types).
- In Chapter 4 we will discuss the algorithm from a practical point of view analyzing its complexity and presenting a reference implementation.
- In Chapter 5 we will discuss experimental results and compare the performance of the algorithm with respect to other approaches.
- Finally in the Chapter 6 we will present many of the interesting research topics still open.

1.5 Contributions of the thesis

The main contribution of this work is a solution for the problem of estimating object features trough the aggregation of crowdsourcing annotations, even with the presence of a high number of malicious users.
Such solution is based on the generalization of the algorithm proposed by Karger et al. [27]. We will analyze in detail the algorithm from both the theoretical and the practical point of view, explaining how it can be used to solve aggregation problems from different domains.
We will show how it allows to reduce up to 70% the number of annotations required to reach a given accuracy level and how it gives acceptable results even with up to 75% of malicious users.

Chapter 2

Related Work and State of the Art

In this Chapter we will analyze various techniques used to solve the adversarial behavior by redundant annotations. This techniques aggregate the annotations in different ways in order to obtain better results.

Many of this techniques are tailored to binary annotations, labeling or classification. The algorithm we will propose in Section 3.2 is not tailored to a specific kind of annotation and though some of the techniques we are going to present in this Chapter can be seen as a particular instantiation of it. In literature we can identify two main classes of methodologies:

1. **Non-iterative**

   uses heuristics to compute a single aggregated value of each question separately [33]. Example of this techniques are **majority voting** (see Section 2.1) and **a priori quality checking** (see Section 2.2).

2. **Iterative**

   performs a series of iterations, each consisting of two updating steps: (i) updates the aggregated value of each question based on the expertise of workers who answer that question, and (ii) adjusts the expertise of each worker based on the answers given by him [33]. Example of this techniques are **expectation maximization** (see Section 2.3) and **iterative learning** (see Section 2.4).

2.1 Majority Voting

One of the most simple techniques used to solve the problem is majority voting. It is also known as **majority decision**.

> *" Majority Decision (MD) is a straightforward method that aggregates each object independently. Given an object o_i, among k received answers for o_i, we count the number of answers for each possible label l_z. The probability $P(X_i = l_z)$ of a label l_z is the percentage of its count over k; i.e. $P(X_i = l_z) = \frac{1}{k} \sum_{k_j=1}^{k} \mathbf{1}_{a_{i,j}=l_z}$. However, MD does not take into account the fact that workers might have different levels of expertise and it is especially problematic if most of them are spammers. "*
>
> Hung at al. [33]

This method is based on mainly two assumptions:

1. The number of cheaters is less than the number of good annotators.
2. A great number of annotations per object is available.

The two assumptions are required to have such a high probability that the consensus of the users is the right one.

Pros

- If the assumptions are respected it generally gives good results.
- Does not require complex aggregation algorithms.
- Does not require any knowledge about the user related to the annotation.
- Does not require any knowledge about the dataset.

Cons

- It requires a strong assumption with respect to the number of good users.

As describe by Sheng at al. [36] this technique is mainly used in binary or classification tasks.
The binary task consists in choosing between two possible answers YES or NO, once gathered the annotations from the users it is just required to choose the answer that has the greatest consensus among them.

The classification task consists in choosing one class from a set of possible classes, once gathered the annotations from the users it is just required to choose the class that has the greatest consensus among them.
As explained in [36], majority voting does perform well when the probability p of obtaining the right answer from a single users is greater that 50%. In this situation the probability of obtaining the right answer using majority voting increases with the number of users, the higher is p the faster it tends to 100%. On the contrary when p is less than 50% majority voting fails. In this situation the probability of obtaining the right answer decreases when the number of users increases, the lower is p the faster it tends to 0.
This was under assumption that all the users has the same quality (probability to give a good answer), in [36] it is even analyzed the situation of users with different quality, reaching more or less the same results.
In [36] it is still presented an extension of the method when used in classification called "soft" labeling that obtains better results due to the multiset nature of the annotation.
Okubo at al. [45] present a small variation of majority voting that exploits information coming from previous answers in order to assign tasks to more thrustfull users. After the assignment the annotations are aggregated in the exact same way as normal majority voting. Even though this version of the algorithm obtains better results it requires more knowledge related to users and the dataset, knowledge that is not always available.
Tsai at al. [43] present a variation of majority voting that requires the users to communicate in order to reach a consensus before assigning the final annotation. It obtains good result when the users engage a profitable debate.

Honeypot

The technique proposed by Lee at al. [28] and extended to the aggregation case by Hung at al [33] is in between majority voting and a priory quality checking.
It uses a technique coming from the computer security field that is commonly used to identify malicious agents and avoid attacks.

> *" In principle, Honeypot (HP) operates as MD, except that untrustworthy workers are filtered in a preprocessing step. In this step, HP merges a set of trapping questions Ω (whose true answer is already known) into original questions randomly. Workers who fail to answer a specified number of trapping questions are neglected as spammers and removed. Then, the probability of a possible label assigned for each object oi is computed by MD among remaining workers. However, this approach has some disadvantages: Ω is not always available or is often constructed sub-*

jectively; i.e truthful workers might be misidentified as spammers if trapping questions are too difficult. "

Hung at al. [33]

2.2 A priori quality check

Another technique used to solve the problem is to do an a priori quality check. Also known as *majority voting with gold standard* or *expert label injected crowd estimation.*

" Expert Label Injected Crowd Estimation (ELICE) is an extension of HP. Similarly, ELICE also uses trapping questions Ω, but to estimate the expertise level of each worker by measuring the ratio of his answers which are identical to true answers of Ω. "

Quoc Viet Hung at al. [33]

Given the expertise level of each worker it is possible to weight differently the different workers. It allows to filter out random annotators (not reliable) and even exploit spammers (always give the wrong answer) by negatively weighting them.

This approach generally gives better results than majority voting as demonstrated by Vuurens at al. [37].

An example can be found in [38] where NLP tasks has been assigned to a crowd of non-experts. In this paper it has been used a gold standard coming from experts in order to evaluate the quality of the crowd.

This method allows to obtains even better result by further analysis.

> *" It estimates the difficulty level of each question by the expected number of workers who correctly answer a specified number of the trapping questions. Finally it computes the object probability* $P(X_i = l_z)$ *by logistic regression that is widely applied in machine learning. In brief, ELICE considers not only the worker expertise* $(\alpha \in [1, 1])$ *but also the question difficulty* $(\beta \in [0, 1])$*. The benefit is that each answer is weighted by the worker expertise and the question difficulty; and thus, the object probability* $P(X_i = l_z)$ *is well-adjusted. However, ELICE also has the same disadvantages about the trapping set* Ω *like HP as previously described. "*
>
> Hung at al. [33]

Pros

- Good performance.
- Robust against random and malicious annotators.

Cons

- Requires a ground-truth with a sufficient size in order to estimate correctly the goodness/expertise of the annotators.
- Requires the ability to inject the ground-truth inside the normal workflow.
- Requires a method to uniquely identify the user that has generated an annotation.
- Requires a greater number of annotations with respect to other methods, because some of them are not directly used in the aggregation, they are just used to estimate the user goodness/expertise.

  This requires more time, and higher costs if it is used with a paid crowdsourcing system.

Ertekin at al. [35] propose a modified version of a priori quality checking that allows to reduce the required annotations. In this version the tasks are assigned to just a subset of the crowd, this subset is identified at runtime.

2.3 Expectation Maximization

Expectation maximization is an approach based on a probabilistic model, as presented by Dempster at al. [11] and Whitehill at al. [44].

> *" The Expectation Maximization (EM) technique iteratively computes object probabilities in two steps: expectation (E) and maximization (M). In the (E) step, object probabilities are estimated by weighting the answers of workers according to the current estimates of their expertise. In the (M) step, EM re-estimates the expertise of workers based on the current probability of each object. This iteration is repeated until all object probabilities are unchanged. Briefly, EM is an iterative algorithm that aggregates many objects at the same time. Since it takes a lot of steps to reach convergence, running time is a critical issue. "*
>
> Hung at al. [33]

This method outperforms a priori quality checking and is more robust to the presence of spammers as demonstrated by Vuurens at al. [41] and Raykar at al. [34] even though it is sensible to the initialization. Different starting points can lead to different solutions.

Pros

- Does not require a ground-truth.
- Robust against random and malicious annotators.

Cons

- Sensible to starting point.
- Requires a method to uniquely identify the user that has generated an annotation.
- Iterative and therefore computational heavy

A similar technique for annotator quality estimation is proposed by Ipeirotis at al. [22]. It has been tailor to multiple choice question and uses "soft" labels instead of hard ones during the estimation of both object probability and worker quality score.

> *" The score separates the intrinsic error rate from the bias of the worker, allowing for more reliable quality estimation. This also leads to more fair treatment of the workers. "*
>
> Ipeirotis at al. [22]

2.4 Iterative Learning

As explained by Kerger at al. [26] [27] Iterative Learning is a belief-propagation-based method for annotation aggregation.
As suggested by Hung at al. [33] it can be even used to estimate question difficulty.

> *" Iterative Learning (ITER) is an iterative technique based on standard belief propagation. It also estimates the question difficulty and the worker expertise, but slightly different in details. While others treat the reliability of all answers of one worker as a single value (i.e. worker expertise), ITER computes the reliability of each answer separately. And the difficulty level of each question is also computed individually for each worker. As a result, the expertise of each worker is estimated as the sum of the reliability of his answers weighted by the difficulty of associated questions. One advantage of ITER is that it does not depend on the initialization of model parameters (answer reliability, question difficulty). Moreover, while other techniques often assume workers must answer all questions, ITER can divide questions into different subsets and the outputs of these subsets are propagated in the end. "*
>
> Hung at al. [33]

As explained in [27] this method obtains performance similar to *expectation maximization* and *belief propagation* with a far more simple underlying model.

Pros

- Does not require a ground-truth.
- Robust against random and malicious annotators.
- Simpler model with respect to *expectation maximization* and *belief propagation*.
- Proven convergence in the binary labeling case [27].

Cons

- Requires a method to uniquely identify the user that has generated an annotation.
- Iterative and therefore computational heavy

The algorithm presented in the Chapter 3 is based on this approach.

Chapter 3

Proposed Approach

The aim of this Chapter is to present and formally define the proposed algorithm.
In the first part we will propose a general framework for annotation aggregation regardless to the kind of annotation. This framework is a generalization of the one proposed by Karger et al. [27] and partially presented in [6], we will go beyond the specific case and try to identify a general version.
In the second one we will analyze common kinds of annotation. For each one of them we will take in account common/naïve algorithms and an instantiation of the proposed algorithm.

3.1 Preliminaries

Let $\mathcal{O}$ be a set of objects, let O_i denote the i-th object in the set, and let $F_i \in \mathcal{F}$ be a feature associated to this object where $\mathcal{F}$ is the space on which the feature is defined.
Let $\mathcal{A}$ be a set of users, called annotators, let a_j denote the j-th annotator in the set and $F_{i,j} \in \mathcal{F}$ the annotation provided by annotator a_j for the feature F_i of the object O_i.
Let $\mathcal{A}_i$ denote the set of the annotators who provided an annotation for the object O_i.
Similarly, let $\mathcal{O}_j$ denote the set of objects annotated by a_j.
Under ideal circumstances, $F_{i,j} = F_i$. However, due to noise intrinsic in the annotation process, $F_{i,j} \neq F_i$, this require to aggregate the annotations coming from more users, in order to reduce/eliminate the noise.
Let $\hat{F}_i \in \mathcal{F}$ denote the aggregated annotation for the feature F_i.
The goal of the algorithm is to minimize the distance (or maximize the similarity) between the real feature F_i and the estimate $\hat{F}_i$.

3.2 General Algorithm

In order to find the estimate $\hat{F}_i$, the available annotations need to be aggregated.

Depending on the specific kind of annotation there are already known algorithms generally based on the computation of an average or a median, like majority voting, which share a common property, they assign the same weight to all the annotations.

Our algorithm goes beyond by assigning different weights to the different annotations. That is:

$$\hat{F}_i = f(\{\langle F_{i,j}, w_{i,j}\rangle | a_j \in \mathcal{A}_i\}) \tag{3.1}$$

The weights $w_{i,j} \in \mathcal{W}$, $\mathcal{W} \equiv [0,1] \subset \mathbb{R}$ capture the quality of annotator a_j to annotate object O_i. The challenging aspect lies in how to automatically determine these weights without any prior knowledge about the quality of the annotators.

To this end, we propose an iterative algorithm that is able to accomplish this task while relying only on the available annotations.

Following an approach similar to [27], the algorithm seeks the solution iteratively by alternating two steps:

- For each object O_i, given the available annotations $F_{i,j}$, $a_j \in \mathcal{A}_i$, and some knowledge about the reliability of each annotator $w_{i,j}^{(k)}$ available at iteration k, compute $|\mathcal{A}_i|$ different estimates $\hat{F}_{i,j}^{(k)}$, $a_j \in \mathcal{A}_i$. Each estimate is obtained by aggregating all annotations but the one given by a_j. That is:

$$\hat{F}_{i,j}^{(k)} = f(\{\langle F_{i,j'}, w_{i,j'}^{(k)}\rangle | a_{j'} \in \mathcal{A}_i \setminus \{a_j\}\}) \tag{3.2}$$

  where $f()$ is an *aggregation function* that computes a weighted consensus among the available annotations.

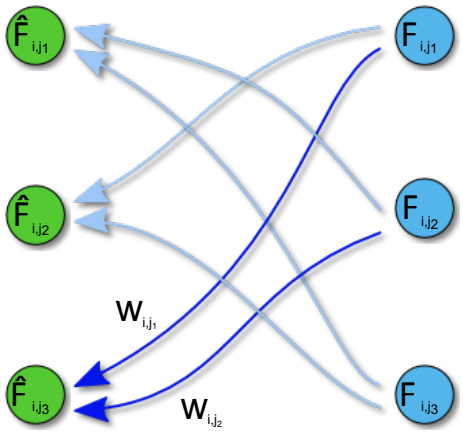

Figure 3.1: Aggregation Step

- For each annotator a_j, given the available annotations $F_{i,j}$, $O_i \in \mathcal{O}_j$, and the current estimate $\hat{F}_{i,j}^{(k)}$, compute $w_{i,j}^{(k+1)}$, i.e., the quality in annotating each object O_i, by measuring the coherence between the annotation $F_{i,j}$, and the current estimate $\hat{F}_{i,j}^{(k)}$ obtained by using all the annotations but the one related to O_i. That is:

$$w_{i,j}^{(k+1)} = g(\{\langle F_{i',j}, \hat{F}_{i',j}^{(k)} \rangle | O_{i'} \in \mathcal{O}_j \setminus \{O_i\}\}) \tag{3.3}$$

  where $g()$ is a *coherence function* that given a set of pairs $\langle F_{i',j}, \hat{F}_{i',j}^{(k)} \rangle$ computes the weight associated to the annotation $F_{i,j}$.

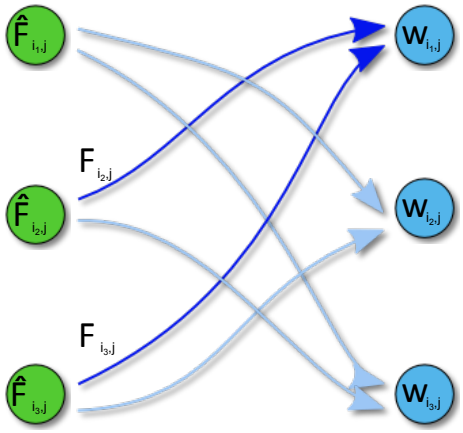

Figure 3.2: Coherence Estimation Step

Note that the description of the algorithm is general and that it does not impose any constraint on the nature of the feature F_i for which the annotations are available. Indeed, the only requirement is the possibility to specify:

i) an aggregation function $f()$ defined in Equation (3.2)

ii) a coherence function $g()$ defined in Equation (3.3)

Therefore, the proposed algorithm significantly extends the original work in [27], which was specifically tailored to work with features associated with a single binary label, whereas we are able to deal with objects that are associated with features of any kind.
The algorithms iteratively executes the two steps until the relative change of the weights falls below a threshold τ, i.e.:

$$\frac{\sum_{i,j} |w_{i,j}^{(k+1)} - w_{i,j}^{(k)}|}{\sum_{i,j} |w_{i,j}^{(k)}|} < \tau \tag{3.4}$$

In our experiments we set $\tau = 10^{-6}$, and the algorithm converged in 6-7 iterations on average. The theoretical analysis of the convergence properties of the algorithm is left to future work. Upon convergence, the final estimate of the feature is computed according to Equation (3.1), in which the weights are set equal to those computed in the last iteration of the algorithm.

3.2.1 Aggregation Function

In Section 3.2 we have presented the aggregation function $f()$ that more rigorously can be defined as:

$$f : (\mathcal{F} \times W)^{|\mathcal{S}_i|} \rightarrow \mathcal{F} \tag{3.5}$$

where $|\mathcal{S}_i|$ is the size of the input set.
In many situations $f()$ can be rewritten in the following way

$$\begin{aligned} f(\mathcal{S}_i) &= m'(\varphi(\mathcal{S}'_i)) \\ \mathcal{S}'_i &= \{\langle x_{i,j}, w_{i,j}\rangle | x_{i,j} = m(F_{i,j}), \langle F_{i,j}, w_{i,j}\rangle \in \mathcal{S}_i\} \\ \mathcal{S}_i &= \{\langle F_{i,j}, w_{i,j}\rangle | a_j \in \mathcal{A}_i\} \end{aligned} \tag{3.6}$$

where:

- $m()$ and $m'()$ are two mapping function that allow to map an item of the feature space to and from an item of a convenient intermediate space $\mathcal{X}$ where linear operations can be defined (ex: $\mathbb{R}^n$). Their rigorously definitions are:

$$m : \mathcal{F} \rightarrow \mathcal{X} \tag{3.7}$$

$$m' : \mathcal{X} \rightarrow \mathcal{F} \tag{3.8}$$

- $\varphi()$ is an aggregation function that works in the intermediate space $\mathcal{X}$. Its rigorously definitions is:

$$\varphi : (\mathcal{X} \times \mathcal{W})^{|\mathcal{S}_i|'} \to \mathcal{X} \tag{3.9}$$

Under these assumptions we can replace $\varphi()$, that is defined over $\mathcal{X}$, where linear operations exist, with a weighted average of the mapped annotations:

$$\varphi(\mathcal{S}'_i) = \frac{\sum_{\langle x_{i,j}, w_{i,j} \rangle \in \mathcal{S}'} w_{i,j} \cdot x_{i,j}}{\sum_{\langle x_{i,j}, w_{i,j} \rangle \in \mathcal{S}'} w_{i,j}} \tag{3.10}$$

3.2.2 Coherence Function

In Section 3.2 we have presented the coherence function $g()$ that more rigorously can be defined as:

$$g : (\mathcal{F} \times \mathcal{F})^{|\mathcal{S}_j|} \to \mathcal{W} \tag{3.11}$$

where $|\mathcal{S}_j|$ is the size of the input set.
In many situations this definition of $g()$ can be too general. Often $g()$ can be defined as the average of the coherence computed on the pairs $\langle F_{i,j}, \hat{F}_{i,j} \rangle$. In this situation $g()$ becomes:

$$g(\mathcal{S}_j) = \frac{1}{|\mathcal{S}_j|} \sum_{s \in \mathcal{S}_j} \sigma(s) \tag{3.12}$$

$$\mathcal{S}_j = \{\langle F_{i',j}, \hat{F}_{i',j} \rangle | O_{i'} \in \mathcal{O}_j \setminus \{O_i\}\}$$

where $\sigma()$ is a function that computes the coherence of a single pair. More rigorously it can be defined as:

$$\sigma : \mathcal{F} \times \mathcal{F} \to \mathcal{W} \tag{3.13}$$

3.3 Example Cases

Here we will present four of the most common kind of annotations.
For each one of them we will present:

- the feature space
- a common aggregation algorithm
- the proposed aggregation function $f()$
- the proposed coherence function $g()$

3.3.1 Binary

Under this kind of annotation can be grouped all the ones related to "on/off" features or classification ones when the available classes are two.

Feature Space

For binary annotations the feature space is simply the set:

$$\mathcal{F} = \{-1, +1\} \tag{3.14}$$

Common Aggregation Algorithm

For binary annotations one of the most common aggregation algorithms is **majority voting**, which formal definition is:

$$\hat{F}_i = \text{sign}\left[\frac{1}{|\mathcal{A}_i|}\sum_{a_j \in \mathcal{A}_i} F_{i,j}\right], \tag{3.15}$$

where $\text{sign}(x) = \pm 1$, depending on the sign of x, and we arbitrarily set $\text{sign}(0) = +1$ to break ties.

Proposed Aggregation Function

For binary annotations we choose to use a modified version of majority voting, the (**thresholded**) **weighted average**, which formal definition is:

$$\hat{F}_i = \text{sign}\left[\frac{\sum_{a_j \in \mathcal{A}_i} w_{i,j} \cdot F_{i,j}}{\sum_{a_j \in \mathcal{A}_i} w_{i,j}}\right], \tag{3.16}$$

Under the framework proposed in Section 3.2.1 this can be defined even in the following way

$$\begin{aligned} \mathcal{X} &= [-1, 1] \subset \mathbb{R} \\ m(F) &= F \\ m'(x) &= \text{sign}(x), x \in \mathcal{X} \end{aligned} \tag{3.17}$$

Proposed Coherence Function

For binary annotations we choose to use the framework proposed in Section 3.2.2 and assign $+1$ when the annotation is equal to the estimate and -1 when it is not:

$$\sigma(\langle F, \hat{F} \rangle) = F \cdot \hat{F} \tag{3.18}$$

3.3.2 Binary Vector

Under this kind of annotation can be grouped all the ones that can be represented as a stream of bits.
We will analyze in the specific case the Regions Of Interest (ROIs) in an image.

Feature Space

For binary vector annotations the feature space is:

$$\mathcal{F} = \{-1, +1\}^N \tag{3.19}$$

In the specific case of the ROIs $N = r \cdot c$ the number of pixels in the image. Every item in the bit stream represents that the corresponding pixel in the image is part of the ROI or not.

Common Aggregation Algorithm

For binary vector annotations the most common aggregation algorithm is **majority voting**, which formal definition is:

$$\hat{F}_i = \text{sign}\left[\frac{1}{|\mathcal{A}_i|}\sum_{a_j \in \mathcal{A}_i} F_{i,j}\right] \tag{3.20}$$

where $\text{sign}(x) = \pm 1$, depending on the sign of x, and we arbitrarily set $\text{sign}(0) = +1$ to break ties.

Proposed Aggregation Function

For binary vector annotations we choose to use a modified version of majority voting, known as (**thresholded**) **weighted average**, executed on each item of the vector, which formal definition is:

$$\hat{F}_i = \text{sign}\left[\frac{\sum_{a_j \in \mathcal{A}_i} w_{i,j} \cdot F_{i,j}}{\sum_{a_j \in \mathcal{A}_i} w_{i,j}}\right] \tag{3.21}$$

Under the framework proposed in Section 3.2.1 this can be defined even in the following way

$$\begin{aligned} \mathcal{X} &= [-1, 1] \subset \mathbb{R} \\ m(F) &= F \\ m'(x) &= \text{sign}(x), x \in \mathcal{X} \end{aligned} \tag{3.22}$$

Proposed Coherence Function

For binary vector annotations and in the specific case ROIs we choose to use the framework presented in Section 3.2.2 and define $\sigma()$ using the **Jaccard's similarity** proposed by Paul Jaccard in [23] [24]:

$$\sigma(F_1, F_2) = \frac{|\{x|F_1(x) = +1 \wedge F_2(x) = +1\}|}{|\{x|F_1(x) = +1 \vee F_2(x) = +1\}|} \tag{3.23}$$

3.3.3 Real Vector

Under this kind of annotation can be grouped all the ones that are based on a vector of real numbers, like a key-point descriptor or a bounding box.

Feature Space

The real vector feature space can be defined as a set of vectors composed by N natural or real numbers:

$$\mathcal{F} = \mathbb{R}^N \tag{3.24}$$

Common Aggregation Algorithm

Two of the most common aggregation algorithms are the **average** and the **median**.

- **Average**

  Which formal definition is:

  $$\hat{F}_i = \frac{\sum_{a_j \in \mathcal{A}_i} F_{i,j}}{|\mathcal{A}_i|} \tag{3.25}$$

  Pros

  - Easy to implement
  - Easy to parallelize
  - Linear complexity in the number of annotations

  Cons

  - High sensitivity to outliers (spammers)

- **Median**

  In the particular we use a median or each component of the vector:

$$\hat{F}_i(x) = \texttt{median}\{F_{i,j}(x) | a_j \in \mathcal{A}_i\} \tag{3.26}$$

  Pros

  - Low sensibility to outliers (spammers)

  Cons

  - Not linear complexity in the number of annotations

Proposed Aggregation Function

As aggregation function we have chosen to use two modified versions of the previously proposed algorithms that take in account the quality of the annotations.

- **Weighted Average**

  The weighted average takes in account the quality of the annotations weighting them in a different way:

$$\hat{F}_i = \frac{\sum_{a_j \in \mathcal{A}_i} w_{i,j} \cdot F_{i,j}}{\sum_{a_j \in \mathcal{A}_i} w_{i,j}} \tag{3.27}$$

  Under the framework proposed in Section 3.2.1 this can be defined even in the following way:

$$\begin{aligned} \mathcal{X} &= \mathbb{R}^N \\ m(F) &= F \\ m'(x) &= x \end{aligned} \tag{3.28}$$

- **Weighted Median**

  As you can see from Figure 3.3 the weighted median is a modified version of the median that weights the elements in a different way as proposed by Edgeworth, F.Y in[13]:

$$\hat{F}_i = \texttt{wmedian}\{\langle F_{i,j}, w_{i,j}\rangle | a_j \in \mathcal{A}_i\} \tag{3.29}$$

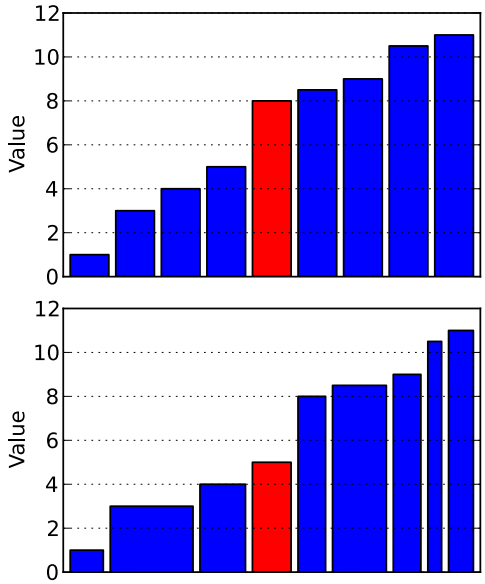

Figure 3.3: Regular Media vs. Weighted Median [3]

Proposed Coherence Function

As coherence function we have chosen to follow the framework proposed in Section 3.2.2 using a $\sigma()$ function based on the **Chebyshev distance** proposed by James Abello et al. [4]:

$$D_{Chebyshev}(p, q) = \max(|p_i - q_i|) \tag{3.30}$$

The Chebyshev distance gives a value in $\{x | x \geq 0 \wedge x \in \mathbb{R}\}$ this value can be mapped in a value of the space used by our framework (see Section 3.2) in the following way:

$$\sigma(\langle F, \hat{F}\rangle) = \frac{1}{1 - D_{Chebyshev}(F, \hat{F})} \tag{3.31}$$

3.3.4 Ranking

Under this kind of annotation can be grouped all the ones that are based on the **ranking**/**sorting** of a finite set of items.

Feature Space

The ranking feature space can be defined as a set of vectors composed by the first N natural numbers without repetition:

$$\mathcal{F} = \{\langle I_1, ...I_N\rangle | N \in \mathbb{N}^+ \wedge \forall_{k\in[1,N]} I_k \leq N \wedge (\forall_{k,l\in[1,N]} k \neq l \iff I_k \neq I_l)\} \subset \mathbb{N}^{+N} \tag{3.32}$$

Common Aggregation Algorithm

The most common ranking aggregation algorithm is the **median rank aggregation** presented by Ronal Fagin et al. [14].
Let $\rho(I_k, F_{i,j})$ be the location given by annotator a_j to item I_k in $F_{i,j}$.
We compute $\mu'_i(I_k)$ as the median over all the $\rho(I_k, F_{i,j})$:

$$\mu'_i(I_k) = \texttt{median}(\{\rho(I_k, F_{i,j}) | a_j \in \mathcal{A}_i\}), k \in [1, N] \tag{3.33}$$

Ordering the $\mu'_i(I_k)$ we can obtain a permutation μ_i that can be used to create the estimate $\hat{F}_i$

$$\begin{aligned} \mu &=< \mu'_i(I_k), ..., \mu'_i(I_l) > | \forall_{k,l\in[1,N]} k < l \iff \mu'_i(I_k) \leq \mu'_i(I_l) \\ \hat{F}_i &=< I_k, ..., I_l > | \forall_{k,l\in[1,N]} k < l \iff \mu'_i(I_k) \leq \mu'_i(I_l) \end{aligned} \tag{3.34}$$

Proposed Aggregation Function

As aggregation function we have chosen to use a modified version of the **median rank aggregation**, that instead of using a simple median use a **weighted median**. The only step in the algorithm that changes is Equation (3.33) that becomes:

$$\mu'_i(I_k) = \texttt{wmedian}(\{\langle \rho(I_k, F_{i,j}), w_{i,j}\rangle | a_j \in \mathcal{A}_i\}), k \in [1, N] \tag{3.35}$$

Proposed Coherence Function

As coherence function we have chosen to follow the framework proposed in Section 3.2.2 using a modified version of the **Spearman's rank correlation coefficient** to compare the ranking pairs, as proposed by Charles Spearman [39].

$$s(\langle F,\hat{F}\rangle) = 1 - \frac{6\sum_{i\in[1,N]} d_i(\langle F,\hat{F}\rangle)^2}{N(N^2-1)} \qquad (3.36)$$
$$d_i(\langle F,\hat{F}\rangle) = \rho(I_i,F) - \rho(I_i,\hat{F})$$

Where d_i is the distance between the position of the i-th item in the two rankings. The coefficient is a value in $[-1,+1]$ where $+1$ means that the two rankings are exactly the same (maximum correlation), 0 no correlation and -1 the rankings are exactly one the opposite of the other. In order to map this in a value in the range $[0,+1]$ we have decided to compute a saturation to 0 of the negative values.
The $\sigma()$ function becomes formally:

$$\sigma(\langle F,\hat{F}\rangle) = \begin{cases} s(\langle F,\hat{F}\rangle) & s(\langle F,\hat{F}\rangle) \geq 0 \\ 0 & elsewhere \end{cases} \qquad (3.37)$$

3.4 Convergence

While for the binary version of the algorithm we can be proven that converges in a finite number of steps as in [27] for the others we have not analyzed the convergence property of the algorithm in detail. We can state that in all our tests the algorithm converges after a small number of iterations. The number of required iterations grows when the number of bad users grows or in case they follow a common pattern.

Chapter 4

Implementation Details

In this chapter we will analyze the algorithm proposed in Chapter Section 3 from the implementation point of view. We will discuss its computational complexity, its scalability and finally present a reference implementation that we have used during our tests.

4.1 Computational Complexity

During this chapter let n be the total number of annotations that the system is going to deal with.

$$n = |\{F_{i,j} | O_i \in \mathcal{O} \wedge a_j \in \mathcal{A}\}| \tag{4.1}$$

Defining the computational complexity of the algorithm is not a simple task, due to the following reasons:

- The algorithm is iterative and does not have a predefined number of iterations.

  The problem can be overcome by making an assumption. The algorithm will converge in a finite number of steps (independent of the size of the input set) or will be interrupted automatically after a predefined number of iterations.

  Let $f_w(n)$ be an hypothetical function that computes the **w**hole algorithm and let $f_i(n)$ be the hypothetical function that computes a single **i**teration.

  Under these assumptions the complexity of the whole algorithm $O(f_w(n))$ can be seen as the computational complexity of the single iteration multiplied by a constant $O(f_w(n)) = O(k \cdot f_i(n)) = O(f_i(n))$. For this reason in order to analyze the complexity of the whole algorithm we can just consider a single step.

- Since the algorithm does not define the aggregation function $f()$ and the coherence function $g()$, it is not possible to find a unique solution, but it will depend on the computational complexity of the specific aggregation or coherence estimation function.

  The complexity of the single iteration will be the sum of the complexities of the aggregation step $f_a(n)$ (function of $O(f(n))$) and the coherence estimation step $f_c(n)$ (function of $O(g(n))$).

$$O(f_i(n)) = O(f_a(n)) + O(f_c(n)) = O(\mathcal{F}(n, f())) + O(\mathcal{G}(n, g())) \quad (4.2)$$

4.1.1 Aggregation Step

As suggested in Equation (4.2) the complexity of the aggregation step $O(f_a(n))$ can be seen as function of the complexity of the aggregation function $O(\mathcal{F}(n, f()))$.
During this step for each annotation $F_{i,j}$ we need to compute its estimate $\hat{F}_{i,j}$, this require to apply the aggregation function $f()$ on the set of all the annotations of the i-th object. The complexity of $f()$ is not independent of the object we are estimating and the number of annotators $|\mathcal{A}_i|$ that has annotated that object.
For these reasons:

$$O(f_a(n)) = \sum_{O_i \in \mathcal{O}} |\mathcal{A}_i| \cdot O(f(F_i, |\mathcal{A}_i| - 1)) \quad (4.3)$$

Let now make some assumptions.
The complexity of $f()$ is independent of the feature we are estimating or that the difference is a reasonably small constant.
We cannot assume that the complexity of $f()$ is independent of the size of $\mathcal{A}_i$ (the set of all the annotators that has given an annotation on O_i). This would be a really strong assumption. We can relax this by stating that the complexity is dependent on the size of $\mathcal{A}_i$, but we can assume that the size of $\mathcal{A}_i$ is independent of i. We are though assuming the graph that defines the problem to be regular. Let so define n_a as the number of annotators (and though annotations) related to each object.

$$\begin{aligned} n &= n_a \cdot |\mathcal{O}| \\ \forall_{O_i \in \mathcal{O}} |\mathcal{A}_i| &= n_a \end{aligned} \quad (4.4)$$

Under these assumptions we can simplify Equation (4.3) as follow:

$$\begin{aligned} O(f_a(n)) &= \sum_{O_i \in \mathcal{O}} |\mathcal{A}_i| \cdot O(f(F_i, |\mathcal{A}_i| - 1)) \\ &= \sum_{O_i \in \mathcal{O}} n_a \cdot O(f(n_a - 1)) \\ &= n \cdot O(f(n_a - 1)) \\ &= O(n \cdot f(n_a)) \end{aligned}$$

Finally

$$O(f_a(n)) = O(n \cdot f(n_a)) \tag{4.5}$$

4.1.2 Coherence Step

As suggested in Equation (4.2) the complexity of the coherence estimation step $O(f_c(n))$ can be seen as function of the complexity of the coherence function $O(\mathcal{G}(n, g()))$.
During this step for each annotation $F_{i,j}$ we need to estimate its quality coefficient $w_{i,j}$, this requires to apply the coherence function $g()$ on the set of all the annotations of the j-th annotator. The complexity of $g()$ is not independent of the feature and the number of annotators $|\mathcal{A}_i|$ that has annotated that objects.
For these reasons:

$$O(f_c(n)) = \sum_{a_j \in \mathcal{A}} \sum_{O_i \in \mathcal{O}_j} O(g(F_i, |\mathcal{O}_i| - 1)) \tag{4.6}$$

Let now make some assumptions.
The complexity of $g()$ is independent of the feature we are estimating or that the difference is a reasonably small constant.
We cannot assume that the complexity of $f()$ is independent of the size of the $\mathcal{O}_j$ (the objects annotated by a_j). This would be a really strong assumption. But we can relax this by stating that the complexity is dependent on the size of $\mathcal{O}_j$, but we can assume that the size of $\mathcal{O}_j$ is independent of j. We are though assuming the graph that defines the problem to be regular. Let so define n_o as the number of objects (and though annotations) related to each annotator.

$$\begin{aligned} n &= n_o \cdot |\mathcal{A}| \\ \forall_{a_j \in \mathcal{A}} |\mathcal{O}_j| &= n_o \end{aligned} \tag{4.7}$$

Under these assumptions we can simplify Equation (4.6) as follow:

$$\begin{aligned} O(f_c(n)) &= \sum_{a_j \in \mathcal{A}} \sum_{O_i \in \mathcal{O}_j} O(g(F_i, |\mathcal{O}_i| - 1)) \\ &= \sum_{a_j \in \mathcal{A}} \sum_{O_i \in \mathcal{O}_j} O(g(n_o - 1)) \\ &= \sum_{a_j \in \mathcal{A}} n_o \cdot O(g(F_i, n_o - 1)) \\ &= n \cdot O(g(n_o - 1)) \\ &= O(n \cdot g(n_o)) \end{aligned}$$

Finally

$$O(f_c(n)) = O(n \cdot g(n_o)) \tag{4.8}$$

4.1.3 Iteration Complexity

We can finally compute the complexity of one iteration by substituting Equation (4.5) and Equation (4.8) in Equation (4.2).

$$\begin{aligned} O(f_i()) &= O(n \cdot f(n_a)) + O(n \cdot g(n_o)) \\ &= O(n \cdot (f(n_a) + g(n_o))) \end{aligned}$$

Finally

$$O(f_i()) = O(n \cdot (f(n_a) + g(n_o))) \tag{4.9}$$

Furthermore we can see that the component $(f(n_a) + g(n_o))$ depends on the aggregation function $f()$, on the coherence function $g()$ and on the structure of the graph (the in-degree of the feature nodes n_a and the out-degree of the user nodes n_o). Being $f()$ and $g()$ predefined and n_a and n_o generally controllable we can state that $(f(n_a) + g(n_o))$ becomes a constant independent of n. Due to these considerations we can see that the complexity of one iteration is:

$$O(f_i(n)) = O(n) \tag{4.10}$$

4.1.4 Algorithm Complexity

Given the consideration presented at the beginning of the chapter and Equation (4.10) we can state that the whole algorithm have a linear complexity in the number of annotations.

$$O(f_w(n)) = O(n) \tag{4.11}$$

This property leads to interesting considerations from the scalability point of view.

4.2 Scalability

One of the possible definition of scalability is the one proposed by André B. Bondi

> *" Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth "*
>
> André B. Bondi [8]

As stated by Mark D. Hill

> *" An intuitive notion of scalability is that it implies a favorable comparison between a larger version of some parallel system with either a sequential version of that same system or a theoretical parallel machine. "*
>
> Mark D. Hill [21]

This definition implies the notion of speedup.
Let $time(n, x)$ be the time required by an n-processor system to execute a program of size x.

$$speedup(n, x) = \frac{time(1, x)}{time(n, x)} \tag{4.12}$$

We define scalable a system with nearly linear speedup $speedup(n, x) \sim n$

> Can the proposed algorithm be implemented in a scalable way?

In order to address the question we need to analyze the parallelizability of the algorithm.
We need to identify the presence of tasks that can be executed with no interdependency and with minimum communication efforts.
The algorithm per se is not fully parallelizable due to inter-iteration data dependencies, like the weights $w_{i,j}^{(k)}$ and infra-iteration data dependencies like the annotation estimates $\hat{F}_{i,j}^{(k)}$ produced by the aggregation step and required by the coherence estimation step.
We need to analyze the parallelizability of both the aggregation step and the coherence estimation step.

4.2.1 Aggregation

Analizing Equation (3.1) and Equation (3.2) we can see that the aggregation step is highly parallelizable due to the fact that the only dependencies are on the input data, the annotations $F_{i,j}$ and the weights $w_{i,j}^{(k)}$ that during this step are immutable so it is not required any synchronization or communication between the parallel tasks.

4.2.2 Coherence Estimation

Analizing Equation (3.3) we can see that the coherence estimation step is highly parallelizable due to the fact that the only dependencies are on the input data, the annotations $F_{i,j}$ and the estimates $\hat{F}_{i,j}^{(k)}$, that during this step are immutable so it is not required any synchronization or communication between the parallel tasks.

4.2.3 Final Considerations

Due to the considerations presented in Section 4.2.1 and Section 4.2.2 we can state that the algorithm can scale if the communication efforts required for data dependency are negligible with respect to the efforts required by the actual computations.

4.3 The Library

In this section we will present a java reference implementation for the proposed algorithm (CrowdAnnotationAggregator - GitHub [5]).
In order to maintain the generic nature of the algorithm we have based the library on templating, in this way only a generic version of the algorithm has been implemented, thus the application of the algorithm become easier for developers.

4.3.1 Asynchronicity

The library has been built from the ground up on the concept of asynchronicity in order to exploit the parallelizability of the underlying algorithms.
To accomplish this all the the APIs are non blocking and will signal their completion via callbacks.

4.3.2 Data Containers

The library is based on three data container classes: the **Annotator**, the **Content** and the **Annotation**.

- The **Annotator** represents the homonym concept a_j in the algorithm.

  While the base version of it simply contains an identifier, it can be extended in order to contain information valuable for both the aggregation and the coherence estimation algorithms.

- The **Content** represents the object O_i in the algorithm.

  While the base version of it simply contains an identifier, it can be extended in order to contain information valuable for both the aggregation and the coherence estimation algorithms.

- The **Annotation** represents the homonym concept $F_{i,j}$ in the algorithm.

  It contains the reference to the **Annotator** from which the annotation comes from and the **Content** it refers to. It must be extended in order to contain information, related to the specific kind of annotation, that will be used by both the aggregation and the coherence estimation algorithms.

4.3.3 Step Abstraction

In order to maintain the generic nature of the algorithm the library base its behavior on two classes the **Aggregator** and the **CoherenceEstimator**.

- The **Aggregator** is an abstraction over the aggregation function $f()$.

  Given the nature of the aggregation step we have chosen to group the annotations by **Content**. Each **Aggregator** is responbile of the aggregation of all the **Annotation**s related to the same **Content**. This allows to exploit the linear version of the aggregation function proposed in Section 3.2.1.

  This class must be extended in order to implement the aggregation function chosen for the specific kind of annotation.

- The **CoherenceEstimator** is an abstraction over the coherence estimation function $g()$.

  Given the nature of the coherence estimation step we have chosen to group the annotations by **Annotator**. Each **CoherenceEstimator** is responbile of the estimation of the weights assigned to all the **Annotation**s given by the same **Annotator**. This allows to exploit the linear version of the coherence estimation function proposed in Section 3.2.2.

  This class must be extended in order to implement the aggregation function chosen for the specific kind of annotation.

4.3.4 Algorithm Abstraction

The whole algorithm is orchestrated by the **AggregationManager** a class responsible for instantiating the **Aggregator**s and **CoherenceEstimator**s, to start the aggregation and coherence estimation tasks and to manage the inter-iteration and infra-iteration data dependencies.
This is the heart of the library and does not need any intervention from the developer, that is just responsible to configure it with the right **Annotation**s, **Aggregator**s and **CoherenceEstimator**s.

4.4 Akka

In order to reach an high scalability we have chosen to implement our tests using Akka.

> *" Akka is a toolkit and runtime for building highly concurrent, distributed, and fault tolerant event-driven applications on the JVM. "*
>
> www.akka.io

4.4.1 Actors

Akka is based on Actors

> *" Actors are very lightweight concurrent entities. They process messages asynchronously using an event-driven receive loop. "*
>
> www.akka.io

In our implementation we have developed a central actor that is responsible to initialize the **AggregationManager** and spawn new actors related to each **Aggregator** and **CoherenceEstimator**. Communications between the **AggregationManager** and the workers are wrapped inside special messages.
This allows us to use the full potential of the underline machine. This is due to the number of active workers that is surely greater than the number of cores of the machine and though, thanks to Akka's automatic load balancing, we can use 100% of the underline computational power.

Chapter 5

Experimental Study

We have carried out a large set of experiments in order to evaluate the performance of the proposed algorithm.

5.1 Image Segmentation

Concept-based image retrieval is based on the analysis of the image content by means of computer vision algorithms to automatically assign labels. In some cases, labels are assigned to the image as a whole. In other cases, image regions are identified and given distinct labels. This sort of fine-grained tagging enables more flexible querying paradigms. As an example, several works have recently addressed the problem of annotating body parts [9] and/or garments [25]. Despite the tremendous improvement in accuracy brought by the latest computer vision algorithms, the problem has not been solved yet.
We solve the problem using human-computation. We ask users to draw the contour of the garments in a set of pictures $r \times c$. This contour can be converted into a binary matrix $r \times c$. This problem can be solved with the methods described in Section 3.3.2 by converting the matrix into a vector simply by serializing rows one after the other.
We have carried two kind of tests, synthetic ones and a real world scenario with data coming from *Sketchness* [20], a GWAP designed to assign labels to localized regions in an image.

5.1.1 Synthetic Case

Table 5.1 reports the parameters used in the generation of the synthetic datasets, which are illustrated in the following. We simulated the gameplay of a total of M players annotating N images. The ground-truth regions of interest were generated by synthesizing one out of many possible geometrical

shapes (rectangles, ellipses, stars, etc.) in a random location within each image. Each image was used in n games ($|\mathcal{A}_i| = n$, $\forall i$). The images were randomly assigned to the players. Hence, the number of played games per player varies, and on average it is equal to nN/M. Alternatively, we also considered a regular topology for the bi-partite assignment graph, in which all players annotate the same number of images. We considered two classes of players: good players and cheaters. The fraction of cheaters over the total number of players is denoted as q. The gaming tracks of good players were generated by adding independent and identically distributed Gaussian noise (zero mean, standard deviation σ) to the ground truth contours of the regions of interest and smoothing the track with an averaging filter. Instead, cheaters added their annotations by drawing a geometrical shape in a random location. We also considered the possibility that a small fraction g of images were available with the corresponding ground truth, to evaluate the impact of the availability of a gold standard.

Parameter	**symbol**	**values**
N. games / image	n	**3**, 5, 10
Number of players	M	10, **100**
Number of images	N	**100**, 1000
Topology	-	regular, **random**
Pr. cheating	q	0, 0.25, **0.50**, 0.75
Noise	σ	10, **40**, 80
Gold standard	g	**0.0**, 0.01, 0.02, 0.10

*Table 5.1: Image Segmentation - Parameters of synthetic dataset (**default**).*

For each configuration of the parameters used in our experiments, we generated 10 instances in order to compute the average results.

Methods

As a baseline, we considered aggregating annotations based on majority voting (`MV`) as in Equation (3.20). We investigated both the proposed algorithm (`wMV`). In addition, we also compare with a modified version of majority voting ($\texttt{MV}_{goodonly}$) that is assumed to know in advance who the good players are, and thus aggregates only their annotations.

Measures

We carried out an evaluation by comparing the regions of interest produced by the different aggregation methods with the ground-truth. The true positive rate (TP) is the fraction of pixels within the region of interest, which are correctly labeled. The false positive rate (FP) is the fraction of pixels

outside the region of interest, which are deemed to belong to the region instead. All algorithms produce as output a binary estimate of the region of interest, due to the *sign* operator in Equation (3.20). However, it is possible to trace complete TP rate vs. FP rate ROC curves by replacing *sign* with a thresholded version of it, and then computing TP rate and FP rate with different thresholds. Hence, using this information we estimate the TP rate at a target FP rate equal to 1% (TP@1%).

Results

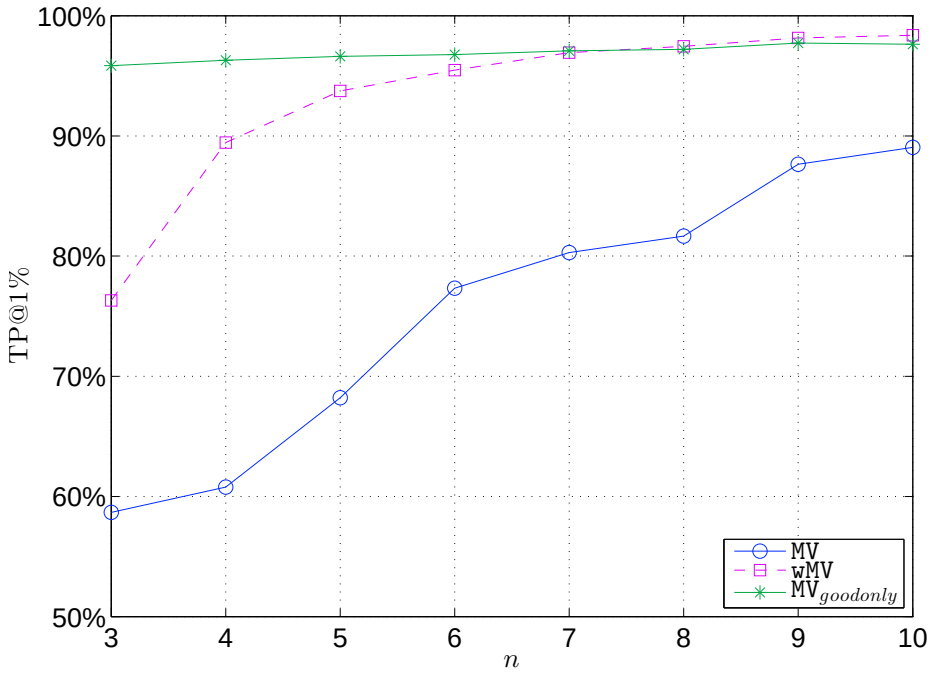

Figure 5.1: Image Segmentation - TP@1% vs. number of games per image n

Figure 5.1 illustrates the performance of the different methods when varying the number n of games per image and setting all other parameters to their default values. The proposed algorithm achieved TP@1% $\approx$ 80% with as few as $n = 3$ games per image, whereas aggregating by majority voting required $n = 6$ games per image. Notice that when $n \geq 6$, the proposed algorithm slightly outperformed even $\texttt{MV}_{goodonly}$. This is due to the fact that, besides assigning a low weight to cheaters (like $\texttt{MV}_{goodonly}$ does), `wMV` assigns unequal weights to good players, depending on their potentially different skills, thus resulting in a more accurate aggregated region of interest. This allows us to trust more the more accurate players and less the other ones, thereby reducing the effect of the imprecise borders of the regions.

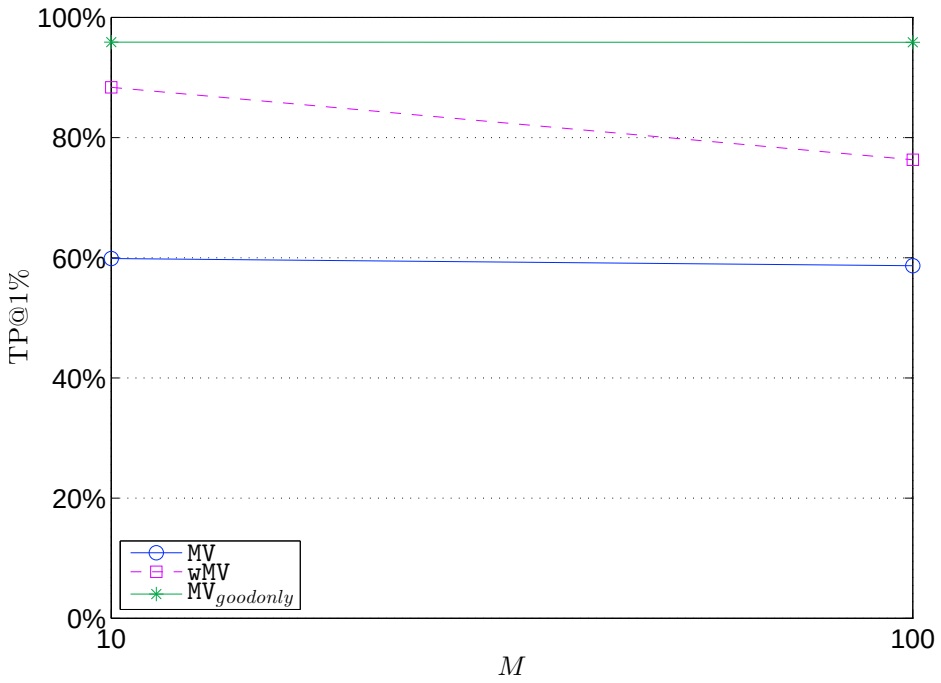

Figure 5.2: Image Segmentation - $TP@1\%$ vs. number of players M

Figure 5.2 illustrates the performance of the different methods when varying the number M of players and setting all other parameters to their default values. It should be noted that the proposed algorithm work better with a small number of players. This can seem strange, but keeping all the other parameters at the default value it means that each player annotates in average 3 images, this reduces the ability of the algorithm to find consensus among the data.

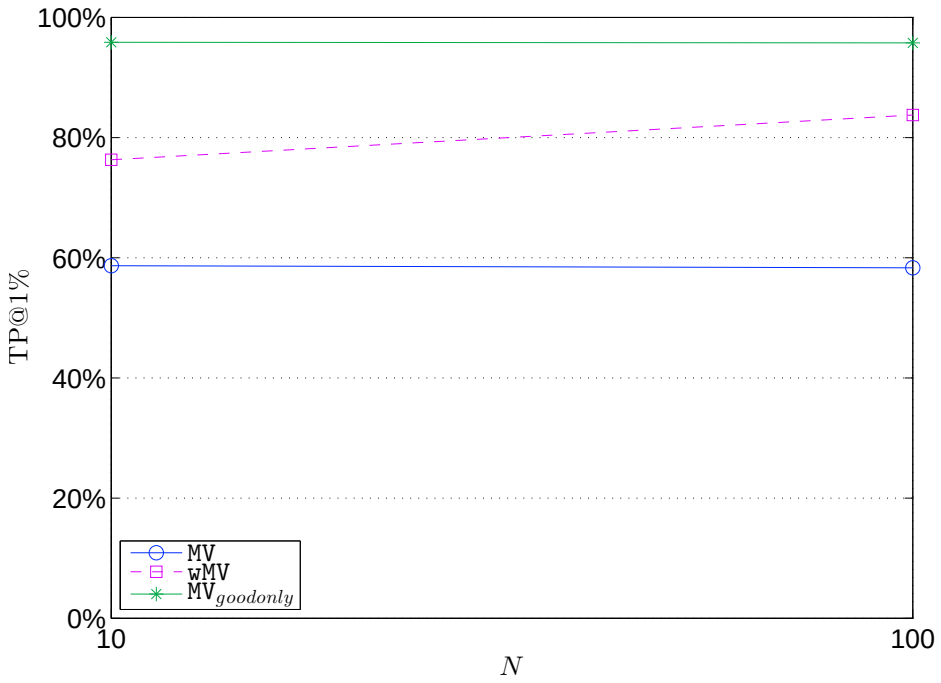

Figure 5.3: Image Segmentation - $TP@1\%$ vs. number of images N

Figure 5.3 illustrates the performance of the different methods when varying the number N of images and setting all other parameters to their default values. It should be noted that the proposed algorithm work better with a great number of images. This can be explained because keeping all the other parameters to default values forces the players to annotate a greater amount of images giving more chances to the algorithm to find consensus among the data.

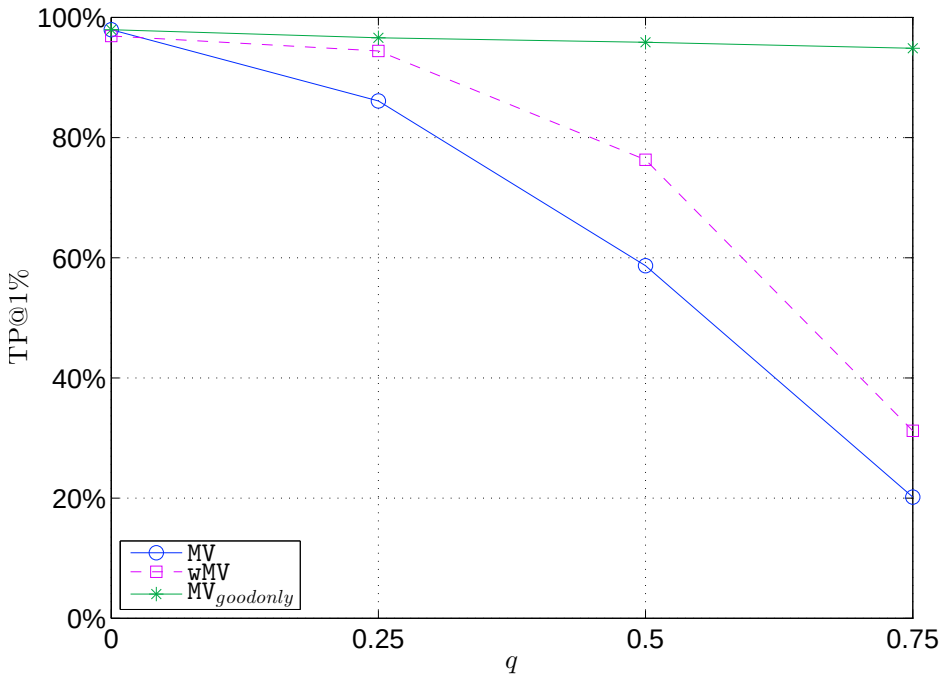

Figure 5.4: Image Segmentation - $TP@1\%$ vs. probability of cheating q

Figure 5.4 illustrates the performance of the different methods when varying the probability q of cheating. For $q = 0$, all methods achieved the same results ($TP@1\% \approx 99\%$). For $q = 0.25, 0.50$, `wMV` significantly outperformed `MV`. When the number of cheaters is greater than the number of good players ($q = 0.75$), the proposed method fails to identify the quality of the players.

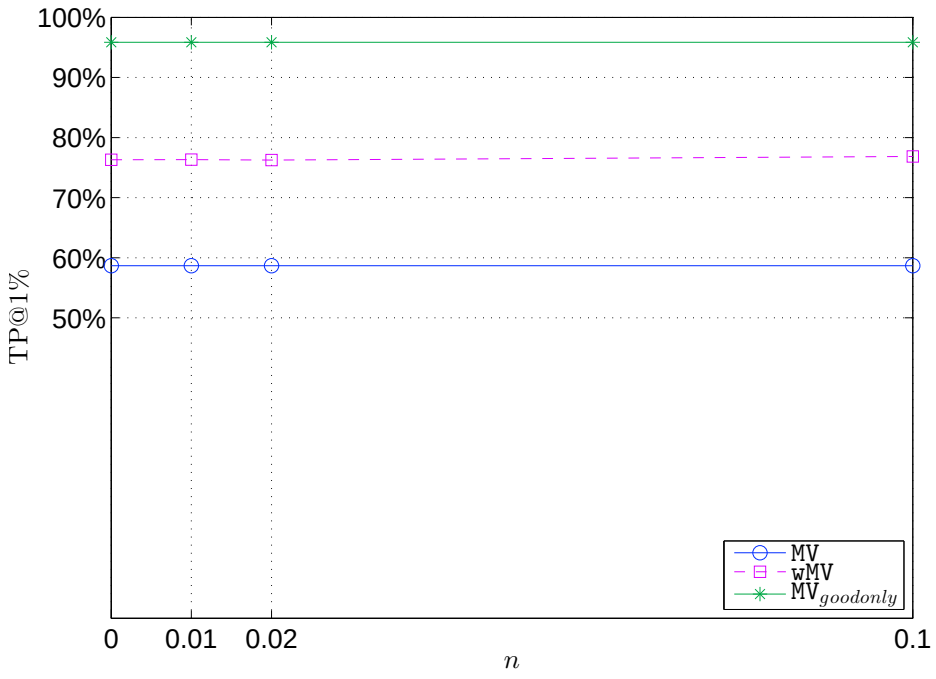

Figure 5.5: Image Segmentation - $TP@1\%$ vs. ground-truth q

Figure 5.5 illustrates the performance of the different methods when varying the ground-truth size g. The availability of the gold standard did not change the results, demonstrating the fact that the proposed algorithm does not need additional source of information to reliably estimate the player's quality.

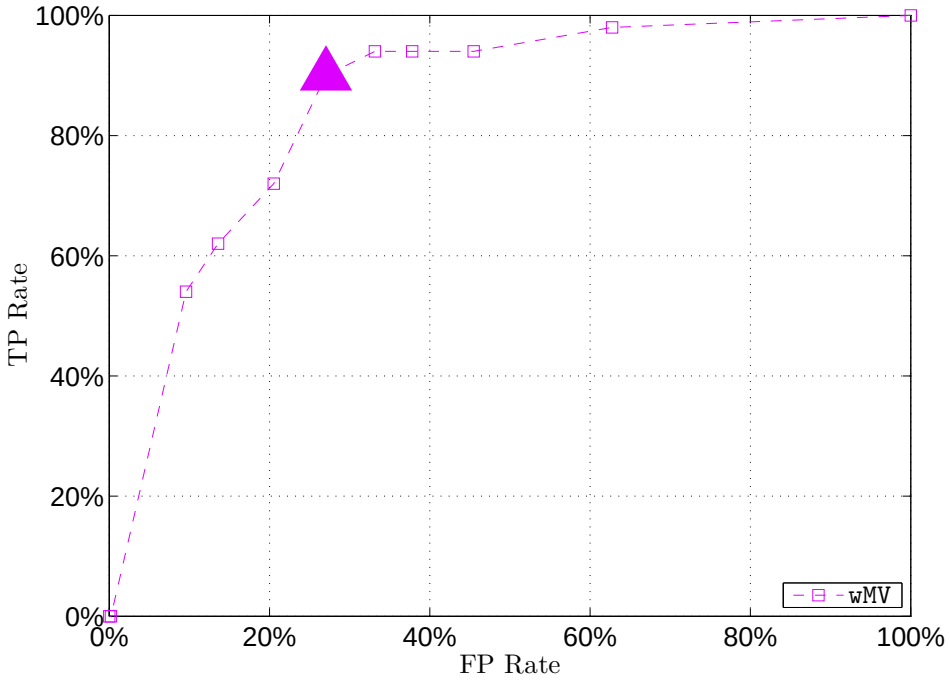

Figure 5.6: Image Segmentation - TP Rate vs. FP Rate in user goodness identification

Figure 5.6 illustrates how the algorithm is able to identify correctly good and bad annotators, thresholding the average final weight of the annotations coming from the same user.

It should be noted that the algorithm was able to identify 90% of the good annotators misidentifying only less than the 30% of the bad ones.

5.1.2 Sketchness

As for the real dataset, we collected 551 gaming tracks from *Sketchness* [20]. The images were obtained from the *Fashion-focused creative commons social dataset* [29], which consists of user-generated content images related to fashion. On average, each image was used in 2.8 games (standard deviation 1.7), and each player participated in 12.8 games (standard deviation 39.4).

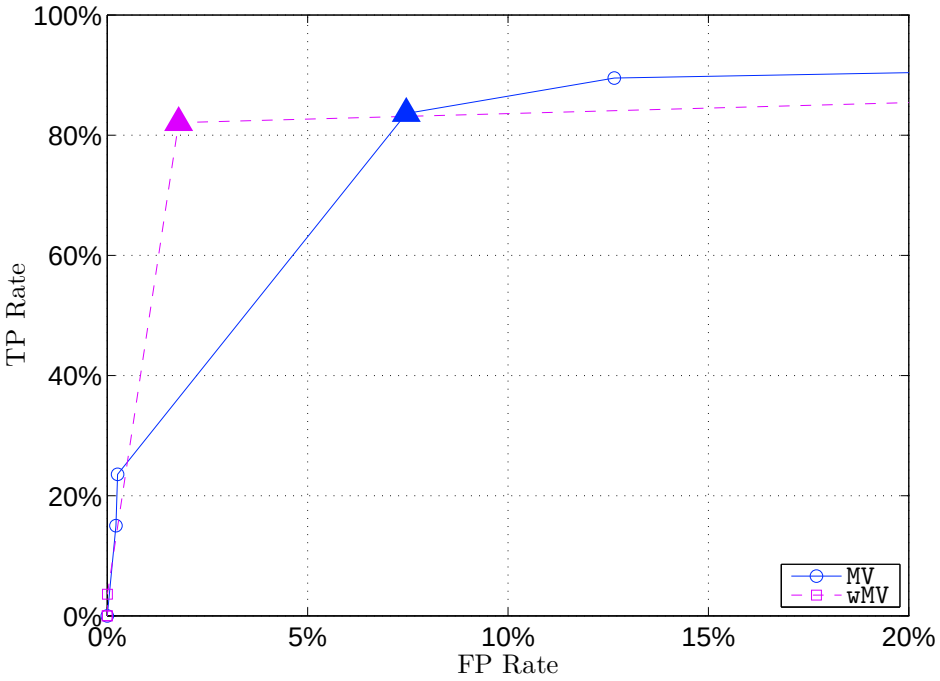

Figure 5.7: Image Segmentation - ROC of pixel identification in the real dataset

Figure 5.7 illustrates the ROC curve obtained using the real dataset which consists of gaming tracks from *Sketchness* [20]. The solid marker (filled triangle) in each curve indicates the point at which the threshold is set equal to zero, as in Equation (3.20). The results confirm those obtained with synthetic data. In this case `wMV` and `MV` attained $TP@1\%$ equal to 55% and 44%, respectively. Increasing the target FP rate to 5%, we obtained 83% and 68%, respectively. In this experiment, we omit $\texttt{MV}_{goodonly}$ as it would require prior knowledge about the good players and the cheaters in a real crowd.

5.2 Bounding Box - Real Vector

Another common task is the identification of objects (or faces) in images. Specifically the interest is on identifying the bounding box of the object. This allows to identify product placements in commercials or identification of persons. Despite the tremendous improvement in accuracy brought by the latest computer vision algorithms, still there are situations in which automatic procedures fail, like in case of non-perspective projections or object occlusions.

We solved the problem using human-computation. We ask users to draw the bounding box of the object (face) in the image. The bounding box is composed of four integer numbers x_{min}, x_{max}, y_{min}, and y_{max}. This problem can be solved by building the vector $\langle x_{min}, y_{min}, x_{max}, y_{max} \rangle$ and applying the methods described in Section 3.3.3.

5.2.1 Synthetic Case

Table 5.2 reports the parameters used in the generation of the synthetic datasets, which are illustrated in the following. We simulated the annotations of a total of M annotators related to N images. The ground-truth bounding box were generated in random location within each image with a random size. Each image was used in n tasks ($|\mathcal{A}_i| = n$, $\forall i$). The images were randomly assigned to the annotators. Hence, the number of task per annotator varies, and on average it is equal to nN/M.

We considered two classes of annotators: good annotators and cheaters. The fraction of cheaters over the total number of annotators is denoted as q. The annotations of good annotators were generated by adding independent and identically distributed Gaussian noise (zero mean, standard deviation σ) to the ground truth coordinates. Instead, cheaters annotations were generated as a random bounding box in a random location and with a random size.

Parameter	**symbol**	**values**
Number of annotations	n	**3**, 4, 5, 6, 7, 8, 9
Number of annotators	M	10, **100**
Number of images	N	**100**, 1000
Pr. cheating	q	0, 0.25, **0.5**, 0.75
Noise	σ	10, **20**

*Table 5.2: Bounding Box - Parameters of synthetic dataset (**default**).*

For each configuration of the parameters used in our experiments, we generated 100 instances in order to compute the average results.

Methods

As baseline, we considered aggregating annotations based on average (`AVG`) as in Equation (3.25) and on median (`Median`) as in Equation (3.3.3). We investigated the proposed algorithms (`wAVG`) and (`wMedian`).

Measures

We carried out an evaluation by comparing the bounding box produced by the different aggregation methods with the ground-truth. We had analyzed the maximum error on one coordinate.

Results

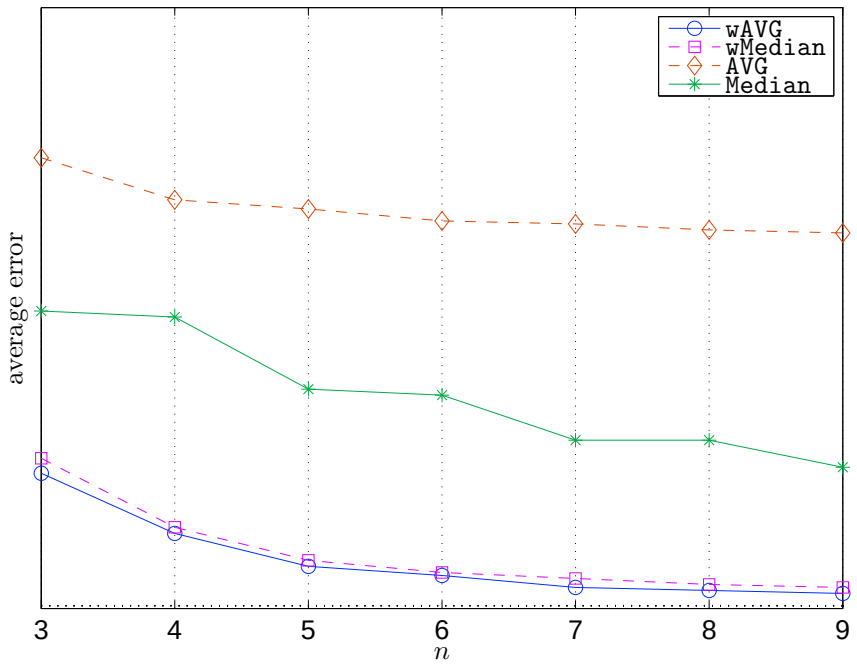

Figure 5.8: Bounding Box - Average Error vs. Number of annotations per image n

Figure 5.8illustrates the performance of the different methods when varying the number of annotations per image size n. It should be noted that the proposed algorithms outperform the others in all configurations and converge faster to a near zero error.

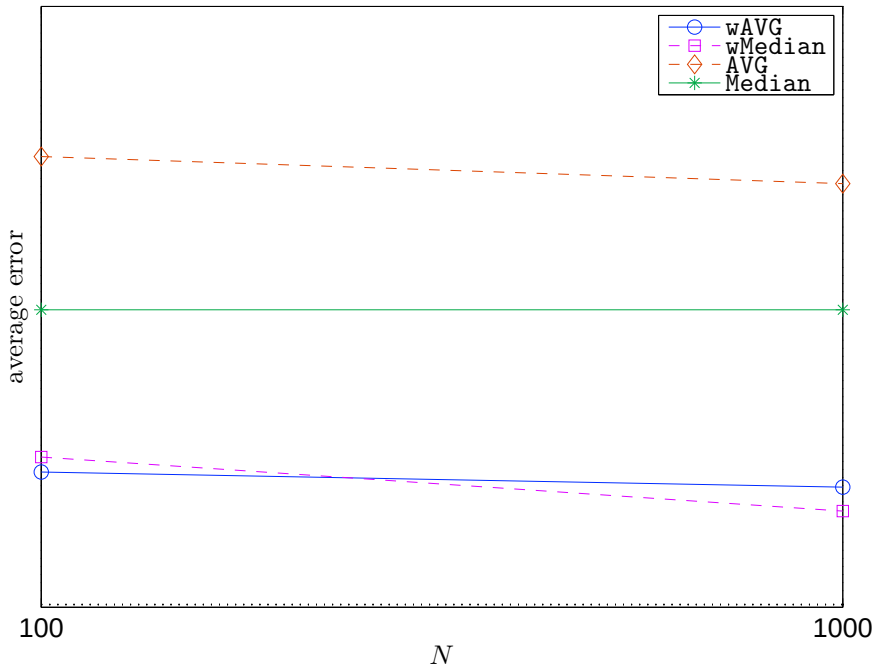

Figure 5.9: Bounding Box - Average Error vs. Number of images N

Figure 5.9 illustrates the performance of the different methods when varying the number of images N. The proposed algorithms outperform the others. It should be noted that increasing the number of images the proposed algorithms reduce the average error (increases in accuracy), because the annotators have to annotate a greater number of images allowing the algorithm to exploit more information about the annotator reliability. In this situation `wMedian` obtain better results with respect to `wAVG`. As expected the `Median` is not affected by N because the actual number of annotations per image does not change and so the performance.

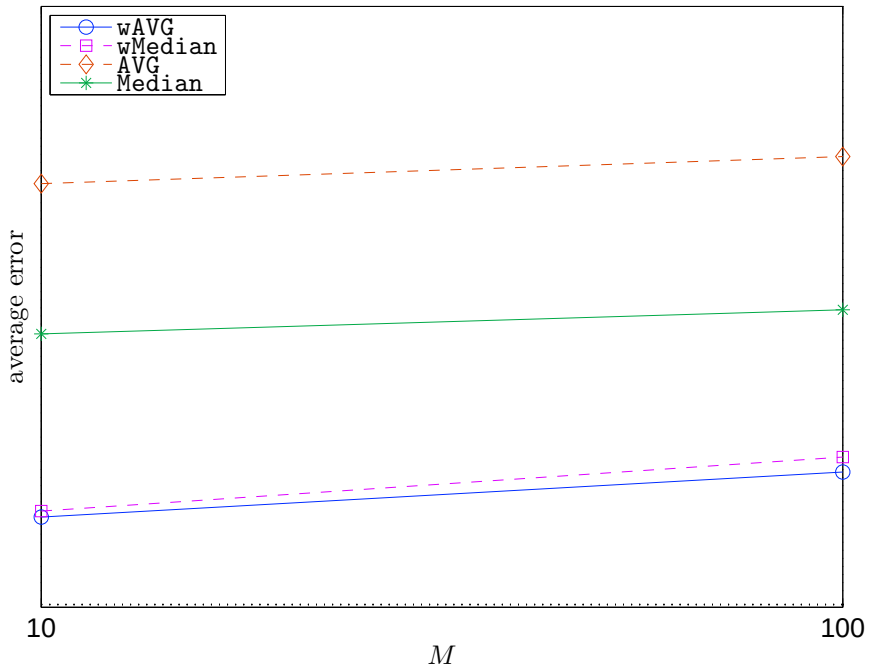

Figure 5.10: Bounding Box - Average Error vs. Number of annotators M

Figure 5.10 illustrates the performance of the different methods when varying the number of annotators M. As expected even the proposed algorithms loose in performance because in this situation the average number of annotations per annotator decrease so the algorithms have less information from which obtain the annotator quality.

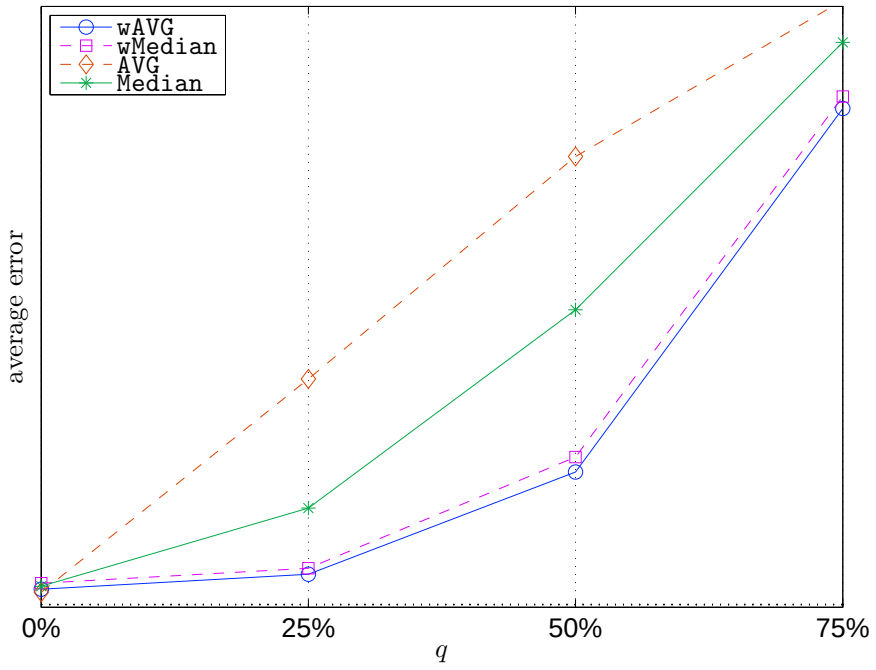

Figure 5.11: Bounding Box - Average Error vs. Probability of cheating q

Figure 5.11 illustrates the performance of the different methods when varying the probability of cheating q. It should be noted that the proposed algorithms are more robust because they do have a very small performance drop increasing q to the 25%, they do have a moderately performance drop increasing q to the 50% while they obtain a performance near to the other algorithms when q reaches 75%.

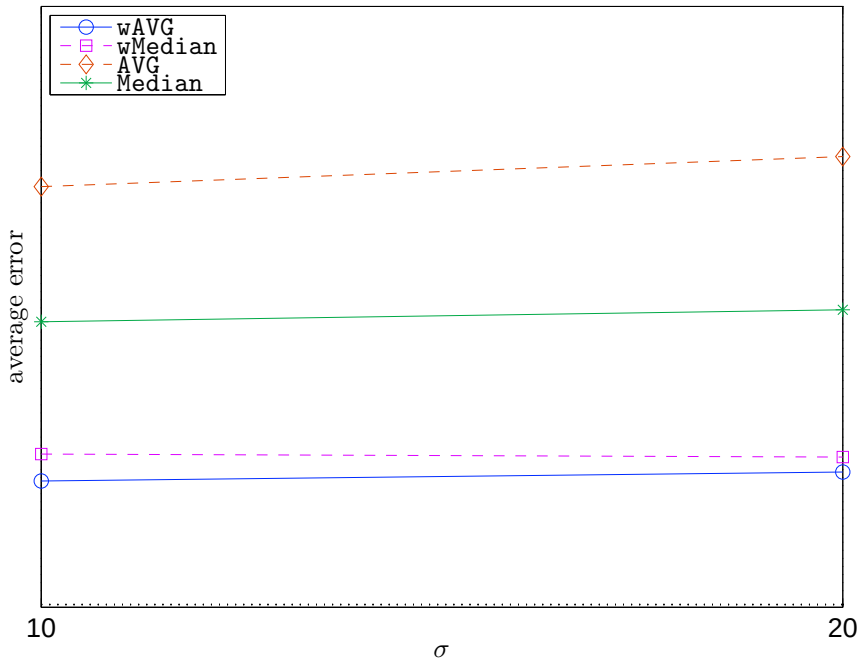

Figure 5.12: Bounding Box - Average Error vs. Noise σ

Figure 5.12 illustrates the performance of the different methods when varying the noise present in the good annotations σ. It should be noted that increasing the noise the error increase, but the proposed algorithms loose in performance less than the other ones. This is due to fact that while the proposed algorithms have to deal with only the noise of the good annotators (the cheaters are weighted less and so are removed) the other ones have to deal even with the cheaters.

5.3 Ranking

Another common task is the ranking of objects. Common examples are hotel or restaurant rating or market surveys. This allows to enhance advertising or recommendation systems.
While already known algorithms lead to valuable solutions we would like to prove that the proposed one gives better results or at least as good as the state of the art.
We will use the same framework proposed in Section 3.3.4 using as baseline Median Rank Aggregation `MRA` and test our modified version Weighted Median Rank Aggregation `wMRA`.

5.3.1 Synthetic Case

Table 5.3 reports the parameters used in the generation of the synthetic datasets, which are illustrated in the following. We simulated the tasks of a total of M annotators related to N groups of objects. The ground-truth ranking were generated by a random permutations of the numbers between 1 and the number of objects. Each group of objects was used in n tasks ($|\mathcal{A}_i| = n, \forall i$). The tasks were randomly assigned to the annotators. Hence, the number of annotations per annotator varies, and on average it is equal to nN/M. We considered two classes of annotators: good annotators and cheaters. The fraction of cheaters over the total number of annotators is denoted as q. The gaming tracks of good players were generated by randomly swapping between 0 and σ pairs of neighbors in the ground truth. Instead, cheaters generate a totally random ranking.

Parameter	**symbol**	**values**
Number of annotations	n	3, **4**, 5, 6, 7, 8, 9
Number of annotators	M	10, **100**
Number of rankings	N	**100**, 1000
Pr. cheating	q	0, 0.25, **0.5**, 0.75
Swaps	σ	0, **1**, 2, 3

*Table 5.3: Ranking - Parameters of synthetic dataset (**default**).*

For each configuration of the parameters used in our experiments, we generated 10 instances in order to compute the average results.

Methods

As a baseline, we considered aggregating annotations based on Median Rank Aggregation (`MRA`) as in Equation (3.31). We investigated the proposed algorithm (`wMRA`).

Measures

We carried out an evaluation by comparing the ranking produced by the different aggregation methods with the ground-truth. We had analyzed the coherence between the estimated ranking and the ground-truth using the Spearman's rank correlation coefficient Equation (3.36).

Results

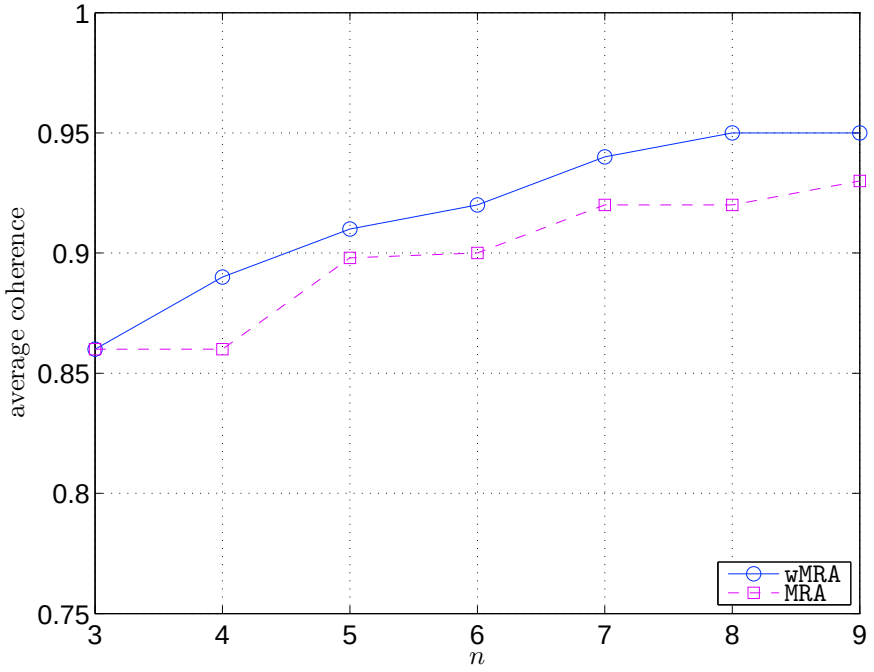

Figure 5.13: Ranking - Coherence vs. Number of annotations per ranking n

Figure 5.13 illustrates the performance of the different methods when varying the number of annotations per image size n. It should be noted that the proposed algorithm `wMRA` gives the same performance of the of the art `MRA` with $n = 3$ while outperforms it when n increases. This shows that while `MRA` has a low sensitivity to bad annotations the proposed algorithm exploit more information and obtains better results.

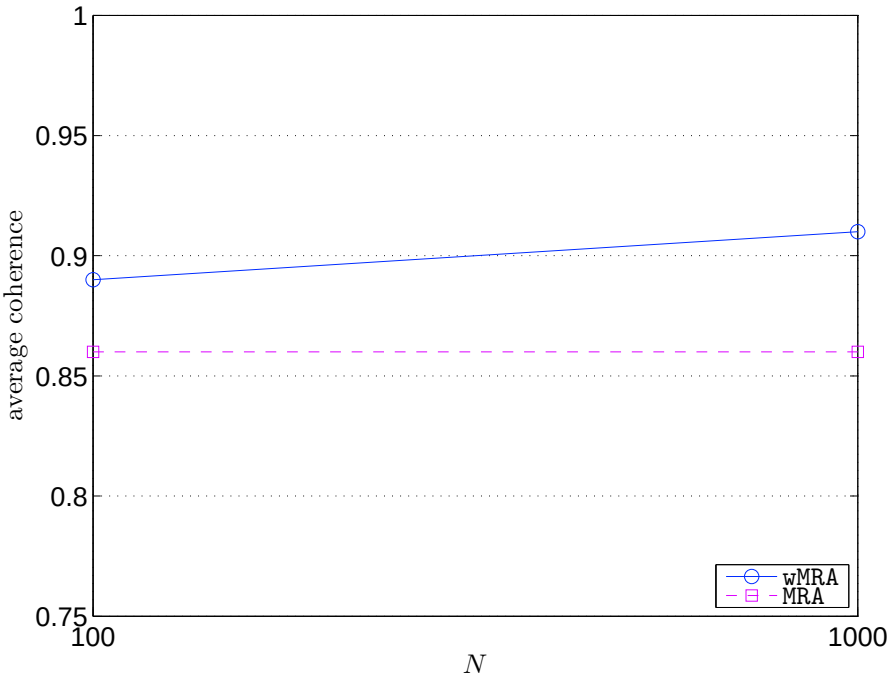

Figure 5.14: Ranking - Coherence vs. Number of rankings N

Figure 5.14 illustrates the performance of the different methods when varying the number of annotated rankings N. It should be noted that `wMRA` obtains better performance increasing N while `MRA` does not. This is due to the fact that increasing N the average number of annotations per annotator increases giving more chances to the algorithm to identify cheaters.

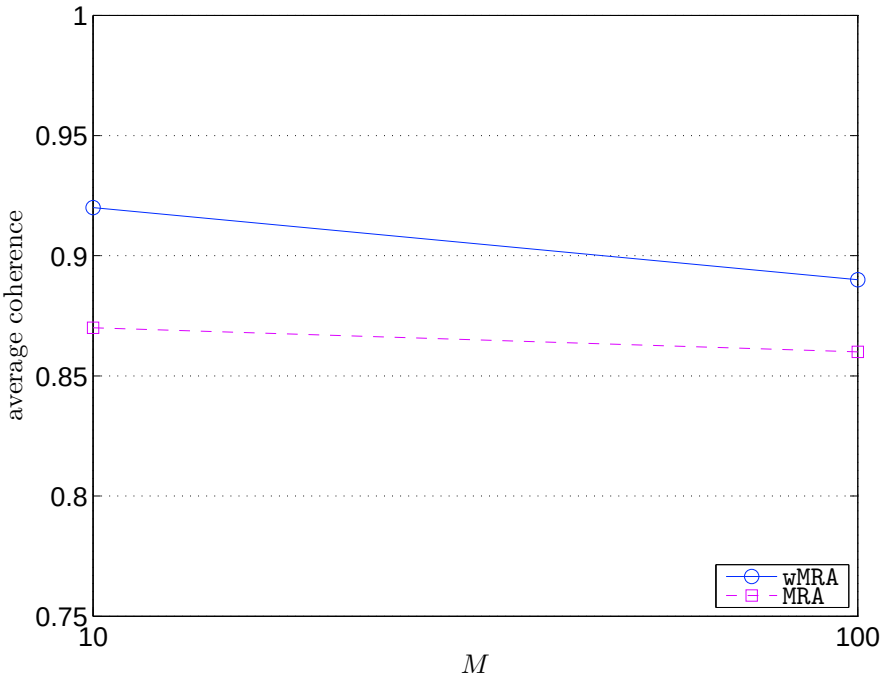

Figure 5.15: Ranking - Coherence vs. Number of annotators M

Figure 5.15 illustrates the performance of the different methods when varying the number of annotators M. It should be noted that `wMRA` obtains worst performance increasing M while `MRA` does not. This is due to the fact that increasing M the average number of annotations per annotator decreases giving lower chances to the algorithm to identify cheaters.

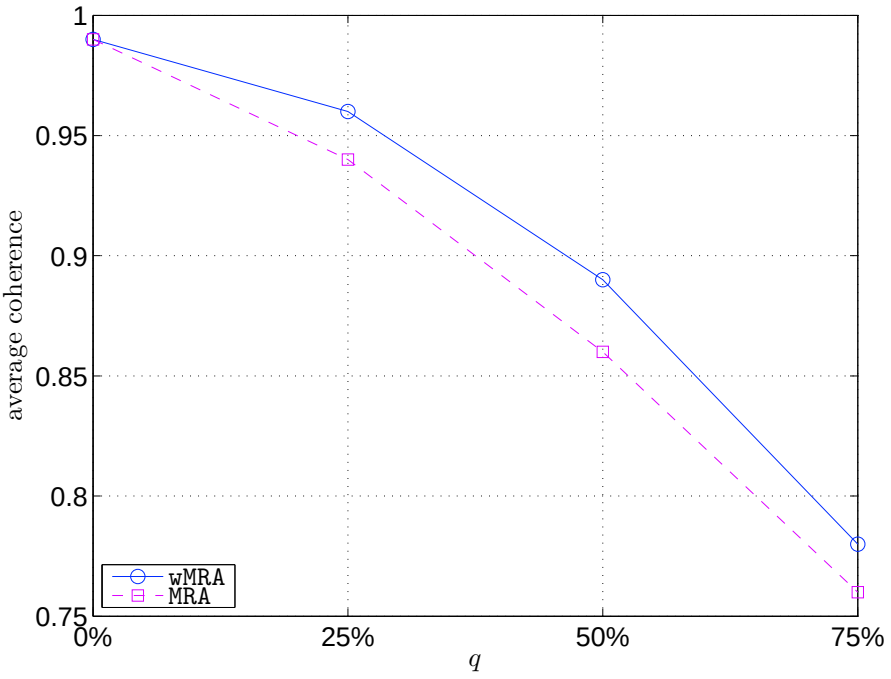

Figure 5.16: Ranking - Coherence vs. Probability of cheating q

Figure 5.16 illustrates the performance of the different methods when varying the cheating probability q. As expected both the algorithms obtains great performance when $q = 0$. Both of them loose in performance increasing q, by the way the `wMRA` gives better performance with respect to `MRA` when q increases, being less sensible to cheaters.

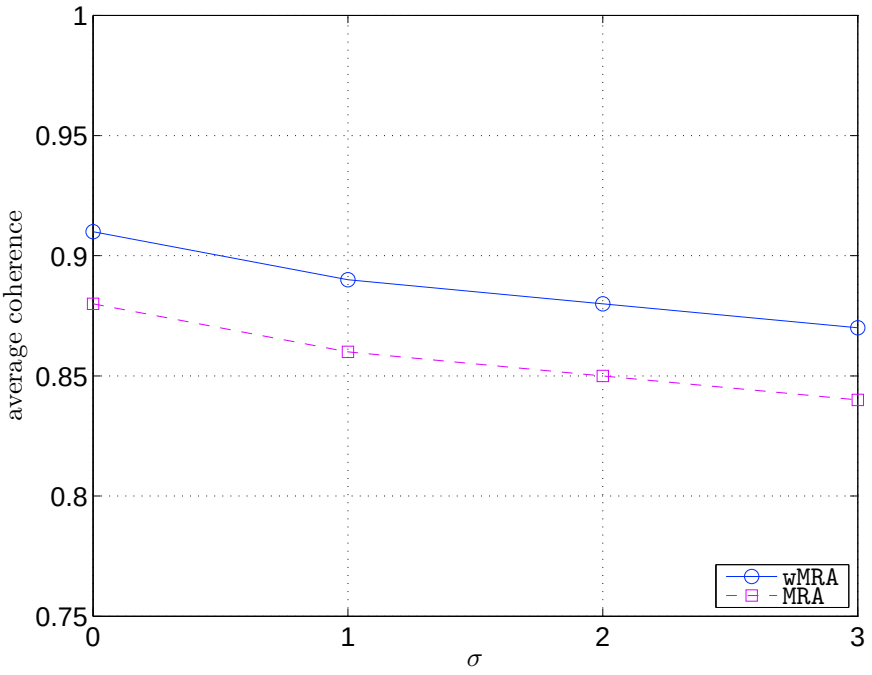

Figure 5.17: Ranking - Coherence vs. Noise σ

Figure 5.17 illustrates the performance of the different methods when varying the the noise (number of random swaps) present in the good annotations σ. As expected increasing σ we obtain worst performance. Both the algorithms loose in coherence in the same way. It should be noted that `wMRA` still has a better performance with respect to `MRA`.

5.4 Final Considerations

As proven by the carried out tests the proposed algorithm always outperforms the state of the art. In the worst situations it gives the same results. Due to this it represents a valuable alternative even if requires a greater effort from the computational point of view.

Chapter 6

Conclusions and Future Work

In this work we have presented a generic algorithm for annotation aggregation in a crowdsourcing environment regardless of the annotation kind. We have analyzed many of the most common annotation types used nowadays and proposed instantiations of the algorithm in that specific cases. The algorithm has generally given good results outperforming naïve approaches. The algorithm gives really good performance when used in batch processes, having a linear complexity in the number of annotations.
The algorithm though does not performs so well in real time applications, especially when the number of annotations grows and when the annotation representation grows in size and complexity (like in the image segmentation case. see Section 5.1).

6.1 Future Studies

As stated in Section 3.4 one of the future studies will be a deep analysis of the convergence property of the algorithm. As for other algorithms, while it is preferable to prove the convergence it is not a deep requirement to still give good results in real world scenarios. If the convergence cannot be proved one of the interesting fields of study will be the analysis of input set in order to identify a property that can guarantee the convergence or not in that specific case.
In Section 5 we have compared the proposed algorithm with naïve approaches that still give acceptable results. Future works will focus on comparing the proposed algorithm with more robust methods like *expectation maximization*, standard *belief propagation* and RANSAC, in order to identify similarities, strengths and weakness of the proposed approach with respect to them.
As suggested in [7] the output of the algorithm are not just the aggregated annotations, but even the final weights can be exploited in order to obtain information on the quality of both the original annotations and the aggregate

ones. While an intuitive relation is that the more good is the annotation the higher is the weight, an interesting study can be carried on the correlation between the final weights and the goodness of the annotator.

6.2 Future Enhancements

Identify the possibility to compute an incremental update of the estimates when new annotations are available. This will allow the algorithm to be used in real time scenarios. Even though this may not be possible an approximate solution is still acceptable. In this situation a complete execution of the algorithm can be scheduled periodically and in the meantime incremental updates will be used.

Bibliography

[1] Amazon mechanical turk. https://www.mturk.com/.

[2] Launchpad translations. https://translations.launchpad.net.

[3] “weighted median" by bscan - own work. licensed under creative commons zero, public domain dedication via wikimedia commons. http://commons.wikimedia.org/wiki/File:Weighted_median.svg#mediaviewer/File:Weighted_median.svg.

[4] James Abello, Panos M. Pardalos, and Mauricio G. C. Resende, editors. *Handbook of Massive Data Sets*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[5] Carlo Bernaschina. Crowdannotationaggregator. https://github.com/B3rn475/CrowdAnnotationAggregator, 2014.

[6] Carlo Bernaschina, Piero Fraternali, Luca Galli, Davide Martinenghi, and Marco Tagliasacchi. Robust aggregation of gwap tracks for local image annotation. In *Proceedings of International Conference on Multimedia Retrieval*, ICMR '14, pages 403:403–403:406, New York, NY, USA, 2014. ACM.

[7] Giorgia Beroffio. Designing bots in gwap. Master's thesis, Politecnico di Milano - Polo di Como, Como, Italy, September 2014.

[8] André B. Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2Nd International Workshop on Software and Performance*, WOSP '00, pages 195–203, New York, NY, USA, 2000. ACM.

[9] Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1365–1372, Sept 2009.

[10] D. E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a discipline. *Commun. ACM*, 32(1):9–23, January 1989.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[12] Marcello Dionisio, Piero Fraternali, Erik Harloff, Davide Martinenghi, Isabel Micheel, Jasminko Novak, Chiara Pasini, M Tagliasacchi, and Srđan Zagorac. Building social graphs from images through expert-based crowdsourcing. In *Proceedings of the International Workshop on Social Media for Crowdsourcing and Human Computation, Paris*, 2013.

[13] F.Y. Edgeworth. *On a New Method of Reducing Observations Relating to Several Quantities.* 1888.

[14] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 301–312, New York, NY, USA, 2003. ACM.

[15] Usama Fayyad, Gregory Piatetsky-shapiro, and Padhraic Smyth. Knowledge discovery and data mining: Towards a unifying framework. pages 82–88. AAAI Press, 1996.

[16] Roman Fedorov. Mountain peak detection in online social media. Master's thesis, Politecnico di Milano - Polo di Como, Como, Italy, September 2013.

[17] Roman Fedorov, Piero Fraternali, and Marco Tagliasacchi. Mountain peak identification in visual content based on coarse digital elevation models. In *Proceedings of the 3rd ACM International Workshop on Multimedia Analysis for Ecological Data*, MAED '14. ACM, 2014.

[18] Roman Fedorov, Piero Fraternali, and Marco Tagliasacchi. Snow phenomena modeling through online public media. In *Image Processing, 2014 IEEE International Conference on*, pages 2179–2181. IEEE, 2014.

[19] Piero Fraternali, Andre Castelletti, Rodolfo Soncini-Sessa, Carmen Vaca Ruiz, and Andrea Emilio Rizzoli. Putting humans in the loop: Social computing for water resources management. *Environ. Model. Softw.*, 37:68–77, November 2012.

[20] Luca Galli, Piero Fraternali, Davide Martinenghi, Marco Tagliasacchi, and Jasminko Novak. 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, Sept 2012.

[21] Mark D. Hill. What is scalability? *SIGARCH Comput. Archit. News*, 18(4):18–21, December 1990.

[22] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, HCOMP '10, pages 64–67, New York, NY, USA, 2010. ACM.

[23] Paul Jaccard. *Lois de distribution florale dans la zone alpine*. Bulletin de la Société Vaudoise des Sciences Naturelles. 1902.

[24] Paul Jaccard. The distribution of the flora in the alpine zone.1. *New Phytologist*, 11(2):37–50, 1912.

[25] Yannis Kalantidis, Lyndon Kennedy, and Li-Jia Li. Getting the look: Clothing recognition and segmentation for automatic product suggestions in everyday photos. In *Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*, ICMR '13, pages 105–112, New York, NY, USA, 2013. ACM.

[26] David R. Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems.

[27] David R. Karger, Sewoong Oh, and Devavrat Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564, 2011.

[28] Kyumin Lee, James Caverlee, and Steve Webb. The social honeypot project: Protecting online communities from spammers. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1139–1140, New York, NY, USA, 2010. ACM.

[29] Babak Loni, Maria Menendez, Mihai Georgescu, Luca Galli, Claudio Massari, Ismail Sengor Altingovde, Davide Martinenghi, Mark Melenhorst, Raynor Vliegendhart, and Martha Larson. Fashion-focused creative commons social dataset. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, pages 72–77, New York, NY, USA, 2013. ACM.

[30] T.W. Malone, R. Laubacher, and C. Dellarocas. Harnessing crowds: Mapping the genome of collective intelligence. Research Paper No. 4732-09, MIT, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA, February 2009. Sloan Research Paper No. 4732-09.

[31] Manoj Parameswaran and Andrew B. Whinston. Social computing: An overview. *Communications of the Association for Information Systems*, 19, 2007.

[32] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM.

[33] Nguyen Quoc Viet Hung, NguyenThanh Tam, LamNgoc Tran, and Karl Aberer. An evaluation of aggregation techniques in crowdsourcing. In Xuemin Lin, Yannis Manolopoulos, Divesh Srivastava, and Guangyan Huang, editors, *Web Information Systems Engineering âĂŞ WISE 2013*, volume 8181 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2013.

[34] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. *J. Mach. Learn. Res.*, 11:1297–1322, August 2010.

[35] Ertekin Şeyda, Hirsh Haym, and Rudin Cynthia. Approximating the wisdom of the crowd. In *Proceedings of the Second Workshop on Computational Social Science and the Wisdom of Crowds (NIPS 2011)*, 2011.

[36] Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. 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*, KDD '08, pages 614–622, New York, NY, USA, 2008. ACM.

[37] Mark D. Smucker. Crowdsourcing with a crowd of one and other trec 2011 crowdsourcing and web track experiments. In *In Proceedings of the Text REtrieval Conference (TREC*, 2011.

[38] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. 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*, EMNLP '08, pages 254–263, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[39] Charles Spearman. The proof and measurement of association between two things. By C. Spearman, 1904. *The American journal of psychology*, 100(3-4):441–471, 1987.

[40] Luis Von Ahn. *Human Computation*. PhD thesis, Pittsburgh, PA, USA, 2005. AAI3205378.

[41] Jeroen Vuurens, Arjen P. De Vries, and Carsten Eickhoff. How Much Spam Can You Take? An Analysis of Crowdsourcing Results to Increase Accuracy. In Matthew Lease, Vaughn Hester, Alexander Sorokin, and

Emine Yilmaz, editors, *Proceedings of the ACM SIGIR 2011 Workshop on Crowdsourcing for Information Retrieval (CIR 2011)*, pages 48–55, Beijing, China, July 2011.

[42] Gang Wang, Tianyi Wang, Haitao Zheng, and Ben Y Zhao. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *23rd USENIX Security Symposium, USENIX Association, CA*, 2014.

[43] Tsai Wei-Tek, Wu Wenjun, and M.N. Huhns. Cloud-based software crowdsourcing. *Internet Computing, IEEE*, 18(3):78–83, May 2014.

[44] Jacob Whitehill, Paul Ruvolo, Ting fan Wu, Jacob Bergsma, and Javier Movellan. *Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise*, page 2035–2043. December 2009.

[45] Okubo Yuki, Kitasuka Teruaki, and Aritsugi Masayoshi. A preliminary study of the number of votes under majority rule in crowdsourcing. *Procedia Computer Science*, 22(0):537 – 543, 2013. 17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems - {KES2013}.