POLITECNICO DI MILANO

Facoltà di Ingegneria dell'Informazione

Corso di Laurea Specialistica in Ingegneria Informatica

Automatic recommendation survey system: Movish

Relatore: Prof. Paolo Cremonesi

Correlatore: Prof. Leonardo Bruni

Tesi di Laurea di:

Vincenzo Ampolo

Matricola n. 750336

Anno Accademico 2012–2013

A tutte le persone che hanno contribuito alla mia crescita, in particolare alla mia famiglia, mio padre, mia madre, mia sorella e mio fratello che mi hanno sempre sostenuto soprattutto nei momenti più difficili.

Ringraziamenti

Questa tesi non sarebbe stata possibile se non grazie a tutte le persone che in un modo o in un altro hanno collaborato alla sua realizzazione. In un anno di lavoro sono tante le persone che andrebbero ringraziate.

Grazie al prof. Paolo Cremonesi che ha creduto nelle mie capacità e nella mia voglia di fare ed ha accettato di assistermi nella tesi pur sapendo che per la maggior parte del tempo sarei stato in California per una internship in Riverbed Technology. I suo corso di impianti informatici in laurea triennale e di digital and internet television in specialistica sono stati di ispirazione per la mia carriera lavorativa che ha mosso i primi passi durante la mia carriera universitaria. Rimpiango di non avere più la possibilità di partecipare alle lezioni interattive tipiche del modo di insegnare del prof. Cremonesi. Lo ringrazio anche della pazienza, di tutte le risposte per mail e di tutto il tempo passato su skype per prendere le decisioni più importanti riguardo il sistema da implementare e la ricerca che ne è seguita.

Ringrazio il prof. Leonardo Bruni per avermi seguito costantemente in questo anno di ricerca e sviluppo del sistema aiutandomi con le decisioni quotidiane ed esplorando nuove soluzioni e nuove ottimizzazioni. Le mail celeri e gli incontri informali al NECSTlab del Politecnico di Milano sono stati fondamentali nella realizzazione del sistema. Sarò sempre riconoscente di tutto il tempo che ha dedicato a questo lavoro.

Ringrazio i prof. Marco D. Santambrogio, Stefano Zanero e Federico Maggi per avermi offerto ospitalità al NECSTlab durante il periodo di tesi.

Per avermi fatto sentire a mio agio nel laboratorio che è stata la mia seconda casa durante il mio ultimo anno di permanenza a Milano, per tutte le varie NECST-pizze e per aver offerto spazio ed assistenza per il server che ha ospitato l'applicativo sviluppato durante questo lavoro di tesi. Il NECSTlab sarà di sicuro uno dei luoghi che più ha segnato la mia carriera universitaria.

Ringrazio i miei amici più cari: Daniela Stefan, Andrea Parola, Ali Bahadori, Rodia, Andréia Cha, Luca Galliani, Emil Gavatta e Cristina di Santo che mi sono stati vicino ed hanno reso piacevole la mia permanenza a Milano. Non verranno dimenticati facilmente tutti i momenti di gioia condivisi insieme e spero di condividere molte altre esperienze insieme in futuro.

Infine ringrazio tutte le persone che hanno partecipato alla ricerca compilando il questionario. Anche grazie a voi questo lavoro è stato possibile.

Una volta che abbiate conosciuto il volo, camminerete sulla terra guardando il cielo, perchè là siete stati e là desidererete tornare. Leonardo Da Vinci

Sommario

I sistemi di raccomandazione sono, al giorno d'oggi, componente fondamentale di tutti i nuovi servizi di successo come Facebook, Amazon, Google, Netflix e Hulu. Tutti questi servizi collezionano in maniera piú o meno pervasiva tutte le nostre preferenze, ovvero tutti i *like* con cui cataloghiamo i contenuti che ci vengono proposti. Queste informazioni sono gestite ed utilizzate in maniera differente che dipende dal servizio: per esempio un servizio di ecommerce come Amazon utilizza i dati su che oggetti ci interessano per selezionare altri oggetti di nostro interesse e proporli durante l'acquisto o la visualizzazione del sito in modo di massimizzare le vendite e quindi i profitti.

I sistemi di raccomandazione sono anche utilizzati per diminuire il numero di oggetti visualizzati in caso di sistemi con un vasto catalogo. Infatti, se Amazon dovesse mostrare tutto il suo catalogo in una pagina l'utente sarebbe molto disorientato. Un sistema di raccomandazione è in grado di scegliere gli oggetti piú pertinenti per un utente e quindi visualizzare quelli in primo piano in modo da non disorientare l'utente e suscitare interesse nell'utente.

I sistemi di raccomandazione sono anche utilizzati nel campo del marketing. Aziende come Google utilizzano le preferenze degli utenti per visualizzare o meno una determinata pubblicità in modo da massimizzare, anche in questo caso, le vendite e quindi i profitti.

I vari sistemi di raccomandazione si differenziano principalmente per

gli algoritmi utilizzati. Gli algoritmi di raccomandazione sono il cuore di una raccomandazione e decidono quali elementi selezionare tra quelli in catalogo. Ogni algoritmo ha come input gli utenti, gli elementi e le preferenze o ratings degli utenti sugli elementi. Ogni algoritmo ha come output una lista di elementi consigliati per ogni utente. Il processo di generazione di una raccomandazione è molto oneroso dal punto di vista computazionale per questo gli algoritmi di raccomandazione generano prima un modello e poi riuntilizzano il modello generato con il profilo dell'utente, ovvero la lista di tutti i rating dell'utente, per generare una raccomandazione. La creazione del modello è il task piú oneroso per una raccomandazione e richiede molta Central Processing Unit (CPU) e Random Access Memory (RAM).

Un buon algoritmo è talmente importante per una raccomandazione che il gigante del noleggio di film online, Netflix, ha nel 2006 aperto una competizione per trovare un algoritmo che riuscisse a suggerire vari film in maniera migliore dell'algoritmo usato all'epoca. La competizione è nota come il *NetFlix prize* [1] e aveva un monte premi di un milione di dollari. La competizione è inziata il 2 Ottobre 2006 ed è stata conclusa il 26 Luglio 2009. L'algoritmo vincente migliorava del 10% le raccomandazioni dell'algoritmo ufficialmente utilizzato da NetFlix all'epoca.

Il lavoro di questa tesi si basa in un lavoro precedente mio e di Andréia Coronado Cha di nome Milo che a sua volta si basava su uno studio del software ContentWise [2] di Moviri.

ContentWise è un innovativo sistema di raccomandazione per provider di servizi televisivi. Il sistema é adottato dai maggiori player di telecomunicazioni in Europa e ha ricevuto diversi riconoscimenti come l'IPTV World Series Awards 2008. Nel 2007 Moviri ha vinto la prima edizione del concorso Start Up of the year award, promosso da PNICube (associazione italiana di incubatori universitari) per il business innovativo e crescita potenziale. Recentemente Moviri ha stretto partnerships con HP, BMC, IBM, Oracle e VMware. Nel 2008 Moviri è stata nomitata HP BTO Partner of the year.

Anche se ContentWise si comporta molto bene per l'ambito per il quale è stato pensato il sistema risulta poco flessibile nel campo della ricerca. La necessità di un sistema più flessibile per svolgere ricerche sui sistemi di raccomandazione è stata necessaria. Il primo progetto che ha provato a dar vita ad un sistema simile a ContentWise ma pensato per far ricerca è stato sviluppato durante il corso di **digital and internet television** del Politecnico di Milano. Così è nato il progetto Milo che però è risultato fragile durante i suoi primi utilizzi a causa delle diverse tecnologie utilizzate nello stesso applicativo e a causa della mancanza di varie ottimizzazioni atte a ridurre il consumo di CPU e RAM degli algoritmi di raccomandazione.

Con l'esperienza maturata durante il progetto Milo, questo elaborato presenta un nuovo sistema di raccomandazione per film chiamato Movish che si differenzia dal precedente per le seguenti caratteristiche

- **Tecnologia**. Movish si basa principalmente sul framework web2py [3], un framework open source per lo sviluppo rapido di applicazioni web veloci, scalabili e sicure. Tutta la parte computazionale è basata su Matlab [4] che comunica con il framework web2py e quindi con l'applicazione tramite la libreria pymatlab [5]. Pymatlab è essenzialmente un layer che traduce le strutture dati python in strutture dati matlab e viceversa.

- **Integrazione**. Il sistema è una singola applicazione e non due progetti separati che collaborano tra di loro come in Milo. Questo permette un migliore controllo del sistema ed un singolo punto di intervento per le modifiche. Tuttavia l'applicazione rimane comunque modulare ma non esiste una separazione netta come in Milo per quanto riguarda l'applicazione web che si occupa dell'interfaccia utente con il sistema di raccomandazione che si occupa di generare le raccomandazioni.

- **Flessibilità**. Il sistema è in grado di riconoscere se l'amministratore aggiunge nuovi algoritmi al sistema e automaticamente importa il nuovo algoritmo nel sistema. L'amministratore di sistema è quindi in grado di aggiungere un nuovo algoritmo senza dover modificare il codice dell'applicazione e abilita la possibilità di fare facilmente ricerca su nuovi algoritmi.

- **Indipendenza**. Il sistema ha funzionalità di auto importazione di nuovi film da imdb.com e youtube.com per quanto riguarda i trailers degli stessi. Ogni settimana il sistema importa automaticamente tutti i film che sono stati rilasciati o che verranno rilascitati in un intervallo di 5 anni prima o 5 anni dopo della data in cui avviene l'import. Quando il sistema deve visualizzare un film in cui alcune infomazioni sono incomplete automaticamente prende le informazioni da imdb.com. L'amministratore può ordinare il refresh di tutto il dataset o importare i coming soon attuali o i più popolari selezionando la relativa funzione nel pannello di amministrazione.

- **Scalability**. Il sistema è stato sviluppato per evolversi nel cloud. L'intero sistema è contenuto in una macchina virtuale di tipo KVM che può facilmente essere spostata da infrastruttura ad infrastruttura come Amazon EC2 o Rackspace. Inoltre l'applicazione utilizza in maniera estensiva anche lo scheduler di web2py per permettere l'esecuzione di task asincroni. Ogni operazione che richiede un lavoro computazionale non indifferente è incapsulata in un task che viene eseguito in maniera asincrona rispetto alle richieste web.

Grazie a questi fattori e alla conoscenza sviluppata durante un intero anno nel campo della raccomandazione, Movish risulta essere un sistema flessibile ed efficiente per sviluppare e testare nuovi algoritmi di raccoman-

dazione. Inoltre, affinchè sia possibile testare nuovi algoritmi in maniera efficace un sottosistema di questionari è stato integrato in modo da avere feedback puntuali sulle performances o altri aspetti degli algoritmi di raccomandazione testati.

Il sottosistema di questionari è stato pensato per essere il più modulare possibile in modo da permettere l'inserimento di nuove tipologie di questionario in maniera semplice. La creazione di nuovi questionari la cui tipologia è già definita è invece facilmente gestibile interamente graficamente dal pannello di amministrazione.

Il capitolo 1 illustra una overview della tesi che è una traduzione di questo sommario; il capitolo 2 espone lo stato dell'arte di un sistema di raccomandazione; il capitolo 3 analizza ContentWise, Milo e Movish ed evidenzia le differenze tra i diversi sistemi; nel capitolo 4 Movish è analizzato a livello architetturale e nel dettaglio; nel capitolo 5 il sottosistema dei questionari è analizzato ed infine nel capitolo 6 la prima ricerca utilizzando Movish è esposta.

La ricerca, sempre pubblicata in questo elaborato, che vuole correlare la capacità di una raccomandazione di essere percepita come utile da un utente in relazione al numero di film della raccomandazione stessa e rileva risultati interessanti.

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

ICM	item content matrix
URM	user rating matrix
DEI	Dipartimento di Elettronica e Informazione
GUI	Graphical User Interface
MVC	Model View Controller
DAL	Database Abstraction Layer
SQL	Structured Query Language
ER	Entity Relationship model
RAM	Random Access Memory
nltk	Natural Language Toolkit
HIT	Human Intelligence Tasks
csv	Comma separated values
http	Hypertext Transfer (or Transport) Protocol
https	Hypertext Transfer Protocol Secure

RESTful	Representational State Transfer
html	Hypertext Markup Language
RAM	Random Access Memory
CPU	Central Processing Unit

Chapter 1

Overview

Recommendation systems are the ground in which successful social and e-commerce website are based. The concepts of 'Like' and 'Suggested' heavily used in services like Facebook, Amazon, Google, Netflix, Hulu are based on the concept that users of a system may like or dislike an item. The collection of this preferences or ratings is the basic data for any recommendation algorithms which goal is to find the next item that the user may like. This feature is used in different ways depending from the business model that the service uses: for example on an e-commerce website such as Amazon it is used to let the user think about adding the recommended item in cart to maximize profits, on Netflix the recommendation are used to keep people using the system and thus increase user fidelity.

Recommendation systems are used also to restrict the number of items to display to the user to overcome to information overloading problems. In big e-commerce website, having a big catalogue of thousands of items may disorient the user. Using a recommendation the e-commerce site may still offer a way to navigate through a thousand item catalog, maybe with the support of a search engine, and show the items that the user may look for based on previous purchases of the same user.

Recommendation systems are also used for marketing purposes. Companies like Google use the information of what the user searched to promote some advertisements in respect of others. In this scenario what the user 'liked' is what the user searched for and since the user took an active part in looking for something the advertisement will be much more effective and likely to convince the user in selecting the advertised item.

Recommendation systems differentiate between each other from the used algorithm. The recommendation algorithm is the basic software that makes a recommendation system perform better than another one. It takes three sets of elements: users, items and ratings. It then outputs a list of items for each user. The output is also called recommendation. The process of running an algorithm to generate a recommendation is a computational intensive process and thus complex algorithms elaborate a model and then they use this model to generate the actual recommendation. Creating a model is a very intense process that requires a lot of computation power and Random Access Memory (RAM) for several days. After a model is created it can be used quickly on demand to get the list of suggested items for an user.

Algorithms that do recommendations are very important. It is how the algorithm performs that will make a recommendation system be valid or not. Netflix made an open competition [1] to find the best user profile based algorithm for their needs in suggesting movies to users. The competition had a prize of one million dollars. The competition started on October 2nd 2006 and ended on July 26th 2009. The winning algorithm had an improvement of 10% from the original Netflix algorithm.

The work of this thesis is based on a previous recommendation system developed by me and AndrÃ¨ia Coronado Cha which was previously based on the ContentWise [2] software by Moviri.

ContentWise is and innovative content recommendation engine for IPTV providers. Contentwise has been adopted by a major European triple-player

Telco and shortlisted at the IPTV World Series Awards 2008 at the IPTV World Forum. In 2007 Moviri won the first edition of the Start Up of the Year Award, promoted by PNICube, (the Italian Association of Universities Incubators) for its innovative business and potential growth. More recently, Moviri has partnered with major vendors and service providers as HP, BMC, IBM, Oracle and VMware. In 2008 Moviri has been named HP BTO Partner of the year.

Even if ContentWise has been successfully adopted by a wide range of companies it is not suitable for researches in new recommendation algorithms. To overcome this limitation the first system, called Milo, was developed during the digital and internet television Politecnico di Milano course. Milo was a first attempt in experimenting a new graphical interface for the user and was relaying in too many different technologies that made the overall system really fragile.

Based on the knowledge developed during the coding of Milo, I've developed another recommendation system called Movish which differentiate from the previous one by the following factors:

- **Technology**. Movish is built on top of web2py [3], a free open source full-stack framework for rapid development of fast, scalable, secure and portable database-driven web-based applications under the GPLv3[6] license. All the computational part is managed by Matlab [4] which talks with the web2py powered web application thanks to the pymatlab [5] library. Pymatlab is basically a layer that translates python data structures in Matlab ones.

- **Integration**. The system is a single application and not two separate projects like in Milo. This allows a better control of the system and a single entry point for interventions. The application is still modular but there is not a project level separation between the web application and the recommendation engine.

- **Flexibility**. The system is able to recognize if the administrator adds new algorithms to the source tree and updates the admin panel accordingly. The administrator is thus able of generating a model for a given algorithm and create new survey without modifying the system code. This allows researches to be able to use the system for their tests without caring about the implementation of the system that is going to use the algorithm they implemented. This makes Movish an effective recommendation system for researching new algorithms or improving existing ones.

- **Independence**. The system has an embedded crawler that crawls the information about movies from imdb [7]. Every week the system automatically fetches all the movies that have been released or that are scheduled to be released in an interval of 5 years from the given week. When displaying a movie if some information about the movie is missing the system tries to retrieve the missing information from imdb. The administrator can order to update a movie, the whole dataset or retrieve the most popular movies or the coming soon movies from imdb anytime thanks to the administration interface.

- Scalability. The system has been developed with the cloud in mind. Everything is self contained in a virtualized KVM image which can be easily deployed on popular cloud infrastructure like Amazon EC2 and Rackspace. It uses the web2py scheduler heavily to allow asynchronous tasks. Every time a consuming operation is encapsulated in an asynchronous task which is executed by a list of workers without affecting web users with slow navigation through the movie catalog.

Thanks to this factors and to the knowledge developed during a full years in working with recommendation systems, Movish is an effective and flexible system to do research in the field of algorithms for recommendation systems. In order to test algorithms effectively a flexible survey subsystem was integrated also.

The survey system has been thought to be more modular as possible and allowing easy modifications and adding of new type of surveys. Survey creations is also made trivial thanks to the easy menu in the admin interface.

Chapter 2 exposes the state of the art of a recommendation system; in chapter 3 ContentWise, Milo and Movish are compared between each other; in chapter 4 Movish is being bisected in order to analyze its architecture and features; in chapter 5 the survey subsystem is exposed and discussed and in chapter 6 the first research in Movish is exposed.

The research correlates the number of recommended movies that the user gets displayed with the perception of a useful recommendation with interesting results.

Chapter 2

Recommendation system state of the art

2.1 Introduction

This chapter will simply expose to the reader some backgrounds about recommendation systems. Basic concepts of recommendation systems, its algorithms, explanations and layouts might be of knowledge of the one in hand of this project description, but a brief review could be useful to understand the whole application design. The state of the art of a recommendation system is also analyzed in order to give to the reader a meaningful idea of that a recommendation system is supposed to work and reach its goals in the optimal way.

With the emerging of hardware resources and computational power recommendation system have grown and diversified. Just decades ago a recommendation would have taken hundreds of mainframes to be completed. Nowadays with hardware as a commodity and all the emerging cloud solutions that allows to have the amount of desired computational power, RAM and storage for just the time requested, deploying recommendation system has never been faster and cheaper. Also big companies like Netflix completely run on cloud systems like the one by Amazon or Rackspace. As soon as internet and computers evolved and more information was available and interconnected, search engine started to arise and being adopted by the newborn internet community. These early search engines were basically of two types [8]:

- **Information retrieval**. A keyword is the base concept of this type. Keyword or a set of keywords are used to look for the desired content. The points of failure of this kind of search may be the chosen keyword. In fact the content could have been categorized used a set of different keywords that the one the user is looking for, thus the content is not selected even if it is relevant to the user.
- **Information filtering**. Preferences established by the user are used as a background knowledge to select the relevant items for the user or, alternatively, properties of a item or a subset of items can be used to lead to another set of relevant items.

Recommendation systems are basically of the second type or information retrieval and they evolved heavily under all the 90′. Nowadays search engine giants like Google or Yahoo use both types in integrated system to lead to stronger and valuable results. Lately with the adoption of smart phones and tablets which have a small screen size and different human-computer interaction than a traditional personal computer with a mouse and a keyboard, recommendation systems to surf content on those devices are critical applications and under heavily development.

Recommendation system are widely adopted in all the most known website and web applications. They may be based on:

- **User profile**: the recommendation is based on information gathered from the user. The information can be explicit or implicit. An example of explicit information is the set of ratings. Supposing we are in the case of a movie web application the user profile is the list of all the ratings that the user made for each movie in the database. An example of implicit is the set of pages or items that the user viewed, in the case of a movie web application this can be just the list of items that the user viewed during the continuous navigation of the website.

- **Item content**: the recommendation is based on the current item or set of item displayed. In this case no information about the user is used by the algorithm. As and example of this scenario for a web application abut movies, the item content for the recommendation can be the name of the actors, the genre, the release date or the plot description.

The reader should see a recommendation system as a complementary system of a search engine and not a different implementation of a concept of a search engine. They solve two different problems. Search engines are really powerful when the user knows or has an idea of what he or she is looking for. The user types a keyword or a set of keyword in a text field and the algorithms involved select the contents relevant for that search.

Recommendation system are useful when the user actually does not know what she or he is looking for. Their added value is in suggesting an item that may be interesting from the user perspective. The user has not a keyword to start his/her navigation with. The user will just receive the new item, maybe without even asking about it. Since search engines and recommendation systems solve different problems they are often implemented one aside the other to keep the user in the web application or to satisfy user's needs. From a research [9] at RecSys [10], 45% of users will likely shop in a web application that employs recommendation technology and 69% of users in the highest spending category are more likely to desire the support of recommendation technology during their web application navigation or shopping. So recommendation systems are valuable from both user and web application. It offers the benefit of an easy way to navigate and find items for an user and a better service and/or a better shopping experience for the web application.

2.2 Analysis

Recommendation systems, like search engines, are very complex systems with a strong time constraints. From a recent study [11] users often leave web pages in 10-20 seconds but pages with a clear value proposition can hold people's attention for much longer because visit-durations follow a negative Weibull distribution. Due to this nature of the web user pages should load as quick as possible and thus search results or recommendation results should be displayed in no more than few seconds or the user will leave the page without even looking at the results. These kind of constraints have influenced the way recommendation systems have been developed and evolved through time. From a black box point of view a recommendation system is just a tool that given the proper input returns an output. No matter which algorithm, programming language, or added feature the recommendation system applies, all recommendation system analyzed respect this constraint. Given that we will explore the input of a recommendation system and its stack. The algorithms can be categorized in two families depending from the requested input:

- **Content based**. The elaboration of the suggested items listed are based on the content of the available object options [12]. Therefore the only needed input for this family of algorithms is the item content matrix (ICM). Due to the fact that these algorithms do not know anything about the user they are exposed to some limitations:
  - *the list of metadata influences the recommendation directly*. If the list

  of metadata is not selected properly the recommendation can lead to very poor results for the user perspective.

  - *cold start problem* [13]. without knowing anything about the user the system will may suggest items that are completely not relevant for the user. A solution for this kind of problem may be to shuffle the recommendation with the top popular items of the system. The top popular items of the system may be the most rated or the ones with best ratings.
  - *User will always take suggestion based on a set of preferences that he/she set on the system even if a totally not related item may be interesting for the user*. Usually this problem is overcome by 'Surprise me' or 'I feel lucky' buttons that the user can click that use different ways to select items that are not strictly dependent from the user preferences.
  - *Item overlapping*. the algorithm can treat two different items from the database as if they were the same one. This may happen if the metadata associated to the two movies are similar. In this case the algorithm may hide one of the two items to the user.

- **Collaborative**. This kind of algorithm resemble one of the best algorithm for recommendation ever designed: the word of mouth [14]. If one of your friends suggests to see a movie, maybe with him/her, we will likely do that. No matter if we already know that we may not like the genre or the actor involved, we will still go to see that movies. This continuously happen, even nowadays, every week. In fact movie producers in particular, invest a lot of money and effort in trying to have a good critics. Having a good critics influences the number of people that will watch a movie and thus the income of that movie. The same concept of 'word of mount' is applied on these family of algorithms: based on the ratings, the user is inserted in a cluster of users with similar tastes, movies with good ratings from users in the same cluster are thus selected for the recommendation [15]. Two main concepts behind these algorithms must be clarified [16]:

– *Proximity between users*. Two users with similar tastes and interests tend to rate items in the same patterns. This property is one of the most exploited by the recommendation systems. It seems quite straightforward that if two users like horror movies and at both of them a good horror movie is proposed they will likely rate it the same way with small differences.

– *Proximity between items*. Two related items are usually rated by the same cluster of users at the same way. As in the previous case it seems quite straightforward that if two movies have the same actors and the same genres two similar users will rate the two movies almost in the same way.

As for the content based algorithms, collaborative algorithms have the following problems:

– *Cold start problem* [13]. This problem is much more evident here than in content based algorithms. When a user is inserted in the system there is no way to determine in which cluster the user belongs. There are multiple ways to overcome this problem: the user, at first login, may be forced in rating some movies, this action will define an initial cluster for the users; another solution may be the one of using the top popular items to populate an initial recommendation.

– *Recommendation freshness*. New items that are added to the database and thus have no ratings will not be suggested in any recommendation in the system. This problem can be solved shuffling the recommendation with some new items that have not recommendation.

  - *Incomplete user profile*. If the user rated few items, they are not enough to mold a profile and thus the user cannot be associated to any cluster. This happens because the ratings are not enough for the algorithm to decide to which cluster the user belongs. This can be avoided forcing the user, at first login, to rate a fixed amount of items, maybe from the most popular ones. This is the approach taken from Milo and then Movish to overcome this problem. When the user is going to obtain the first recommendation he/she is forced to rate five movies.
  - *Singular taste*. If the user has a very singular taste the algorithm may be not able to create a cluster if not a single user cluster. This will make really hard for any recommendation system to suggest a relevant item for the user.

Problems related to the algorithms can be overcome thanks to smart decisions taken by recommendation system in terms of user interface. In Movish the cold start problem, and the incomplete user profile problems are solved by forcing the user in rating some movies before the first recommendation. The recommendation freshness and the item overlapping problems are overcome displaying newest movies in the homepage and letting the user rating them.

Collaborative algorithms have another categorization:

- **User based**. The main focus of the algorithm is the user itself. In this kind of algorithms the model is usually created using a "user x user" matrix in which both rows and columns are composed by users. Therefore, each user from the URM will be represented as a vector in a space with N dimensions, being N the number of items present on the database. The proximity between users is then found by the angle between vectors: small angle means that the two user profiles are similar.

- **Item based**. The main focus of the algorithm is the item. In this kind of algorithms the model is usually created using a "item x items" matrix in which both rows and columns are composed by items and each cell represents the similarity level between the items that intersect. After this, the generation proceeding of suggestions are easily done, multiplying the user profile and the model just created which will result in the rating list to all database items to a certain profile.

2.2.1 Comparison

Collaborative algorithms are able to advice an user to see a totally unrelated item in relation to his/her previous ratings enabling a *surprise effect* for the user [17]. As a side effect collaborative algorithms tend to mess the user ability to understand the relation between his or her tastes and the suggested item. Content based algorithms tends to be more clear from the user's perspective, shrinking the probability of confusion by the user. Thus collaborative algorithms tend to be more "aggressive" when compared to content based ones.

2.2.2 Input

There are two kind of inputs for a recommendation system [18]:

- **Item content matrix**. It is used in content based algorithms. It is a matrix which has all the items as columns and all the metadata as rows. A metadata is a specific characteristic or feature which may be present or may not be present for the given item. In case of the metadata being present in the given item the corresponding cell is set

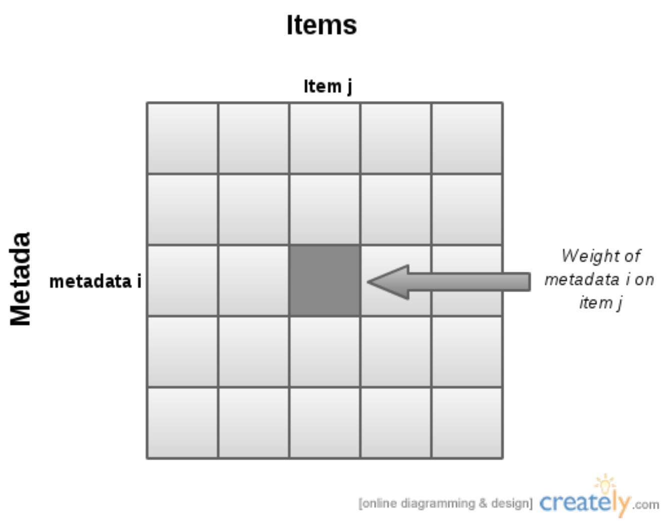

Items
Item j
Metada
metadata i
Weight of metadata i on item j
[online diagramming & design] creately.com

Figure 2.1: ICM

to 1, 0 otherwise. In a case of a recommendation system for movies the metadata could be the presence of an actor, the set of genres and keywords extracted from the plot. Figure 2.1 shows how items and metadata correlate in the composition of the item content matrix.

- **User rating matrix**. It's used for collaborative algorithms. It is a matrix which has all the items as columns and all the users as rows. The cell identified by a item and a user contains the rating that the user gave to that movie. It has the value of 0 if there is not any rating for the given item. To better clarify the composition of an user rating matrix, the figure 2.2 explains how it is composed.

  The rating can be collected in two different ways:

  - Explicitly. The user is aware that he/she is rating an item. The rating can be provided in different scales. The most common scales are: binary, 3-scale or 5-scale. Binary is a two value rating usually referred as 'like' or 'dislike'. 3-scale is usually referred as 3 stars or 3 thumbs up. 5-scale is usually referred as 5 stars or a

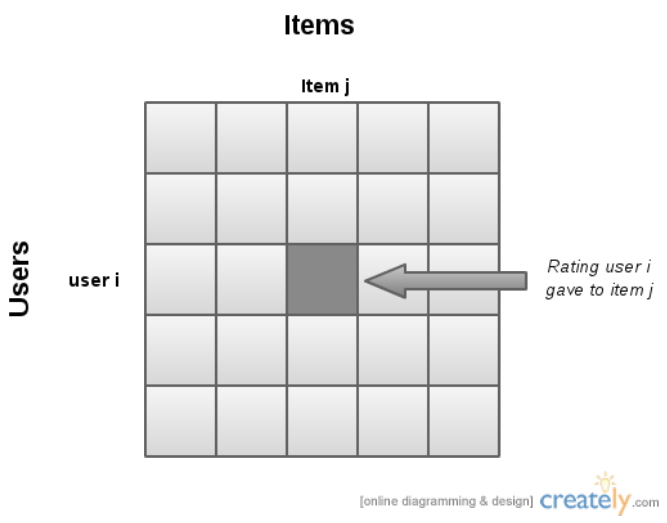

Items
Item j
Users
user i
Rating user i gave to item j
[online diagramming & design] creately.com

Figure 2.2: URM

slicer in the 0-5 range. In all the scales, low rating means that the user disliked the item, high values mean that the user liked the item.

– Implicit. The user is not aware that he/she is rating an item. While browsing the system collect navigation information like clicks on links or displayed pages and based on this information implies ratings for the items. This way of collecting data is very used nowadays with pervasive web applications. Information is key point in business. The more a company know about its customers the better it will be able in satisfy their expectations.

2.2.3 Stack

To make this possible recommendation system evolved in a stack like way that can be summarized in three phases [19]:

- **Batch or model creation**. This is a very computational intensive task. Given an ICM or URM it will create a model that will be used later during the recommendation. In this phase most of the computation is performed in order to reduce the computational resources of the next step. Due to the nature of this phase it is seldom run. Usually this kind of operations are run once a week or once a month.

- **Real time or recommendation generation**. Given the model elaborated in the previous step and a user or a set of users this task produces the recommendation or a list of items for the user or set of users. The main requirement in this phase is a fast response since it has to run in real-time during page generation. A late response leads in dissatisfaction of the user and a high probability of losing users just because they wait too much the page to be generated. The algorithm should be robust enough to prevent error condition and still gives a valid set of items.

- **Anti reshuffling or recommendation update**. Even after recommendation generation, user can choose a suggested item and thus evaluate it. A good recommendation system is able to react actively under those circumstances and operate in order to generate a new recommendation taking into account the fact that new ratings are added. The new recommendation should thus based on the previous one with small modification. That is why it is called anti reshuffling, to avoid the cycle of the same items.

Figure 2.3 summarize the stack of a recommendation algorithm to better clarify the concept. In the figure there is a clear distinction between what is batch and thus computational intensive and what is not. After that the model is ready the recommendation can be performed which generates the Top N List or the recommendation.

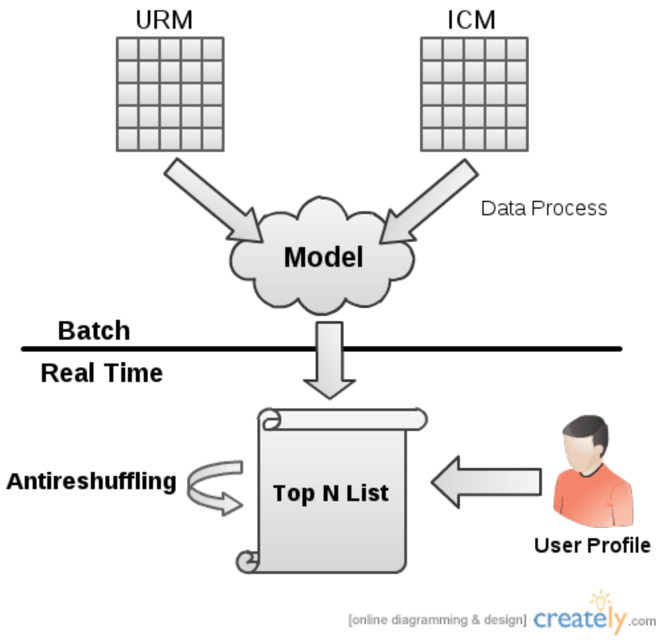


Figure 2.3: Stack

2.3 Output

As anticipated in the previous section the output of a recommendation system can be of two types:

- **Individual scoring**. Given a user the algorithm returns all the rating forecast for this user. This is then turned to a recommendation thanks to the following assumption: recommendation should select items relevant for the user or items that the user will like, since we have user rating forecast we can use the items with the greatest forecasts for our recommendation since the user will probably like them.
- **Top-N recommendation**. The algorithm output a list of the Top-N items most suitable for a certain user. This can also be reproduced starting from the Individual scoring output and ordering that list in descending order of preference..

2.4 Presentation

The recommendation algorithm is essential in selecting a relevant subset of items for the user but it's not enough to create a credible recommendation [20]. For the recommendation to be useful to the user it must be correctly exposed to the user attention. The way the information is displayed is fundamental and contribute in making a recommendation system successful or not. From recent studies it is also confirmed that the way a recommendation is displayed may affect the items selected by the user [21].

Usually recommendation come in a list like form which is a natural way to display a vector of items. A study shows that a text structured overview with concise text and graphics not only facilitates the creation of a trustful relationship between the system and the user but increases user efficiency in selecting a recommended object [22].

All these studies have been taken in consideration while designing both ContentWise and Milo. They have also been considered in Movish too but the style has been modified taking into account that almost 10 years are passed from that studies and that web application such as Facebook [23], Twitter [24] and Pinterest [25] have been gained popularity among users. They provide new ways to explore and display a very big amount of data using a list like, in case of Facebook and Twitter or a catalog like in case of Pinterest, styles. Since movies can be easily represented by their poster which resembles the movie atmosphere, Movish uses a Pinterest like layout to display movie poster, title and genres.

2.5 Conclusions

Recommendation systems are currently a very active research field. The integration with social networks like Facebook, Pinterest, Twitter and Google+ are now opening new ways for them to be more pervasive and be able to match the user tastes better than ever. Also new algorithms are under development that are able to use this new amount of data that was unavailable just five years ago. Not only algorithms and datasets are important, as we demonstrated in this chapter also presentation and usability determine the success or the failure of a recommendation system, especially on e-commerce websites.

Recommendation systems are definitively a kind of technology that will evolve in the future with search engines and other systems in order to increase user satisfaction. In the following chapter we will continue analyzing how Movish has been implemented and how all the challenges that implementing a recommendation system implies have been solved.

Chapter 3

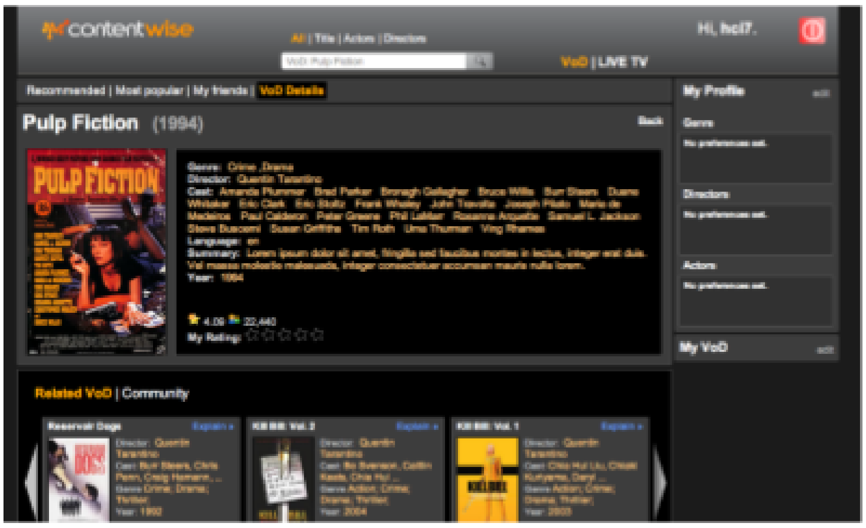
Movish system background

3.1 Introduction

In this chapter the reader can understand the basic architecture of Movish. Since implementing a recommendation system is a challenging process the reader will be guided through all the problems encountered and will receive explanations in how those issues have been addressed either by design or by system architecture in Movish. This chapter is thus fundamental to all the readers that want to understand the basic functions and architecture of the system. We will start from the basic systems that inspired Movish: ContentWise and Milo. We will analyze Movish itself later on.

3.2 ContentWise

ContentWise is a well known recommendation system developer by the Dipartimento di Elettronica e Informazione (DEI) of Politecnico di Milano university. This product basically exposed a movie catalog and makes the use able to perform ratings and obtain recommendations using different algorithms. The user is able to change the recommendation system by itself and the interface is designed to be used by recommendation system



Genre: Crime ,Drama
Director: Quentin Tarantino
Cast: Amanda Plummer Brad Parker Bronagh Gallagher Bruce Willis Burr Steers Duane Whitaker Eric Clark Eric Stoltz Frank Whaley John Travolta Joseph Pilato Maria de Medeiros Paul Calderon Peter Greene Phil LaMarr Rosanna Arquette Samuel L. Jackson Steve Buscemi Susan Griffiths Tim Roth Uma Thurman Ving Rhames
Language: en
Summary: Lorem ipsum dolor sit amet, fringilla sed faucibus montes in lectus, integer erat duis. Vel massa molestie malesuada, integer consectetuer accumsan mauris nulla lorem.
Year: 1994

4.09 22,440
My Rating:

Figure 3.1: ContentWise

professionals. The figure 3.1 shows ContentWise home page.

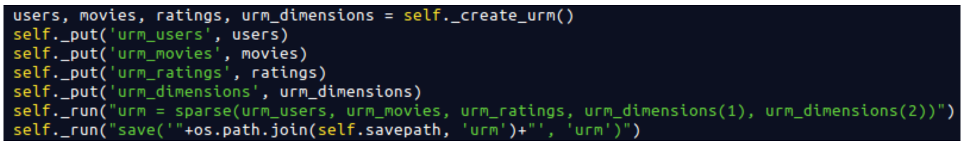
From a first look it is clear that the Graphical User Interface (GUI) has a late 90′ style. The architecture of ContentWise is also very rigid: the algorithms are embedded in the GUI and this makes the action of adding a new algorithm really troublesome. The application is mainly written in Java programming language.

The inability of adding a new algorithm easily and the overcomplicated user interface of ContentWise made the system not suitable for a system that is able to research on new algorithms on unaware users.

3.3 Milo

During the 2011/2012 course of digital and internet television a new recommendation system for movies that wanted to be similar to ContentWise but more modular and flexible such that it was easy to add new algorithms was implemented.

Since there wanted to be clear distinction between the engine and the presentation a Model View Controller (MVC) [26] paradigm has been used.



```
users, movies, ratings, urm_dimensions = self._create_urm()
self._put('urm_users', users)
self._put('urm_movies', movies)
self._put('urm_ratings', ratings)
self._put('urm_dimensions', urm_dimensions)
self._run("urm = sparse(urm_users, urm_movies, urm_ratings, urm_dimensions(1), urm_dimensions(2))")
self._run("save('"+os.path.join(self.savepath, 'urm')+"', 'urm')")
```

Figure 3.2: URM creation code

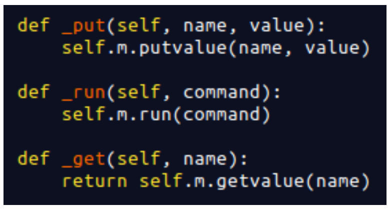
Due to the python programming language popularity and emerging web frameworks, it has been chosen as base language. Among all the available frameworks, Pyramid [27] has been selected thanks to its flexibility and easy to use.

In fact Pyramid allows to have a base MVC framework skeleton and plug it with all the libraries you need for your application. Also modifying the base components it is easy and suggested by developers to fully satisfy the application needs. In particular Milo had to communicate with a matlab [4] based set of algorithms. Thanks to pymatlab [5], python is able to talk directly to matlab in a way that reminds a server-client socket communication.

As stated in chapter 2, recommendation systems perform recommendations using a model. This model requires a lot of computational power and time. Web applications have a better user experience if they have a lower page load delay instead. To overcome this conflicting needs a asynchronous task manager framework has been integrated to allow the web application to trigger some events that add new tasks. These tasks are thus run on a separate process.

In order to make the administrator able to add unmodified Matlab [4] algorithms the pymatlab [5] library has been used. Pymatlab makes python be able to communicate with matlab via the matlab shell using accessible via a socket like interface. Figure 3.2 shows a piece of code of the creation of the user rating matrix (URM) from the python side.

The *self.create_urm()* method generates the urm picking the data from



```
def _put(self, name, value):
    self.m.putvalue(name, value)

def _run(self, command):
    self.m.run(command)

def _get(self, name):
    return self.m.getvalue(name)
```

Figure 3.3: Code for the pythonmatlab communication

the database. That python function returns a numpy array of users, movies, rating e and array indicating the dimension of the the matrix. The dimension is needed to create a sparse matrix of the right size. In fact the *self.create_urm()* function only returns actual data, skipping all the zeros. This avoid not necessary usage of ram due to too big matrices. Line from 2 to 5 put the variables in the matlab environment. Line 6 creates the sparse matrix into the matlab environment. The matrix created so is then stored as a .mat file for later use during the creation of the model.

As you can see all the communication is handled by running string like commands. Variables are stored and retrieved from the matlab environment thanks to the *get* and *put* functions of Figure 3.3.

This allows a clean separation of the python environment and the matlab environment. All the variables are transferred as strings or as floats. This is a limitation of the pymatlab library which is not able to handle integers variable correctly. Also all variables from python to matlab must be numpy [28] arrays.

NumPy is the fundamental package for scientific computing with Python. It contains useful function and data structures for easy management of matrices and numerical data arrays. Together with scipy [29], numpy wants to be an open source alternative to matlab providing software for mathematics, science and engineering. Scipy is built on top of numpy.

Milo was developed in a modular way and was the union of two subsystems [18]: a graphical engine and a recommendation engine called Whis-

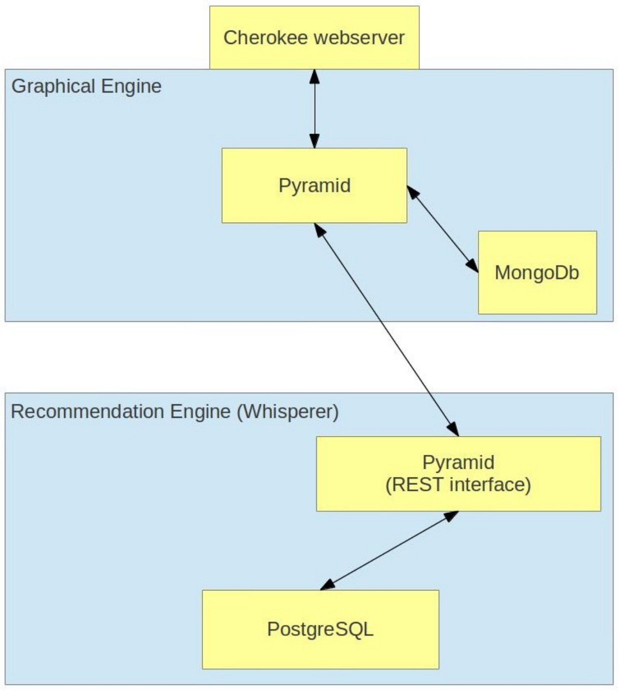


Figure 3.4: Milo architecture

perer. Whisperer was in charge of communicating with matlab through the just exposed pymatlab interface. The graphical engine was able to communicate with Whisperer thanks to a rest interface. In order to clarify reader understanding of Milo architecture the user can refer to Figure 3.4.

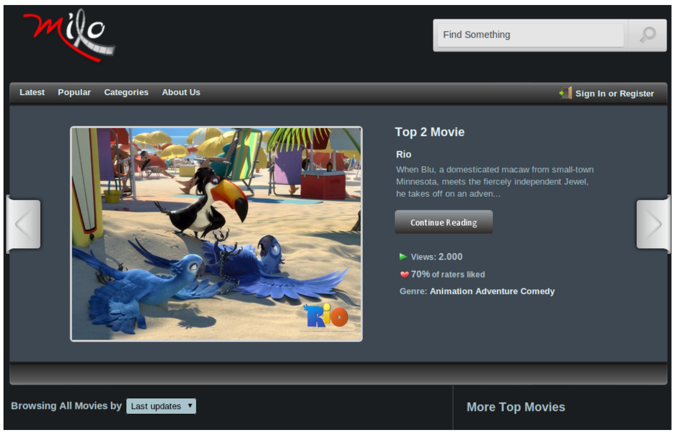
The system is composed by a cherokee webserver. Each requests that arrives to the webserver spawns a new worker or is assigned to an existing one. Each worker is as instance of pyramid web framework with the application running. The frontend database is based on the NoSQL [30] MongoDB [31] databases.

NoSQL is a broad class of database management systems identified by non-adherence to the widely used relational database management system model. NoSQL databases are not built primarily on tables, and generally do not use structured query language for data manipulation. NoSQL database systems are often highly optimized for retrieve and append operations and often offer little functionality beyond record storage (e.g. key, value stores). The reduced run-time flexibility compared to full SQL systems is compensated by marked gains in scalability and performance for certain data models.

In short, NoSQL database management systems are useful when working with a huge quantity of data when the data nature does not require a relational model. The data can be structured, but NoSQL is used when what really matters is the ability to store and retrieve great quantities of data, not the relationships between the elements. Usage examples might be to store millions of key, value pairs in one or a few associative arrays or to store millions of data records. This organization is particularly useful for statistical or real time analyses of growing lists of elements (such as Twitter posts or the Internet server logs from a large group of users).

When the graphical engine needs to do a recommendation it makes a Hypertext Transfer (or Transport) Protocol (http) Representational State Transfer (RESTful) request to another web application, whisperer, that cares about creating the item content matrix (ICM), URM and all the data structures needed for the recommendation. The recommendation engine relays on a PostgreSQL [32] database for storing the data for the users and the items. Notice that the recommendation engine has only the concept of users and items, this means that it flexible enough to recommend any kind of item to a user. The fact that it was used for movies is thus just a particular case. This was one of the biggest improvements from ContentWise [2]. The recommendation system developed could be reused for any kind of recommendation.

The recommendation was exposed to the user through a slider placed on top of the page such that the user was likely to watch the slider for first. This has been accomplished using animations in the slider that capture the



Top 2 Movie

Rio

When Blu, a domesticated macaw from small-town Minnesota, meets the fiercely independent Jewel, he takes off on an adven...

Views: 2.000

70% of raters liked

Genre: Animation Adventure Comedy

Figure 3.5: Milo recommendation slider

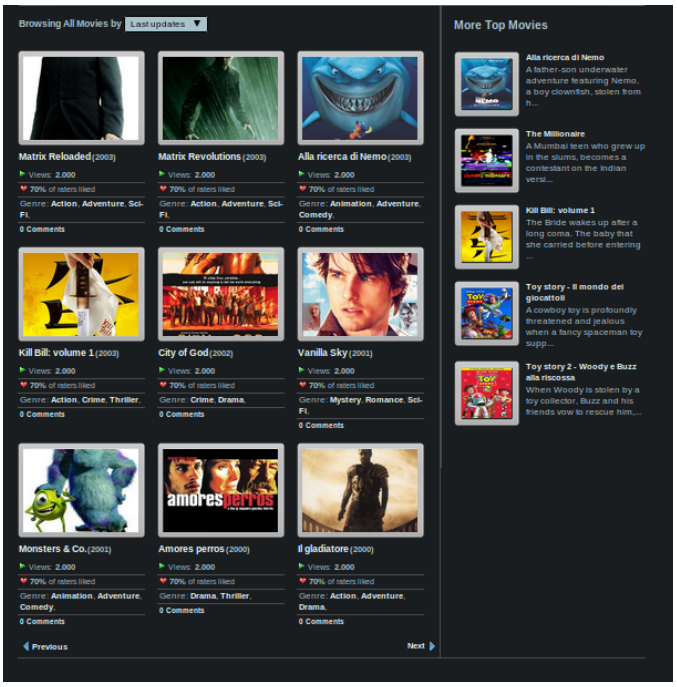
user attention. The result can be see in Figure 3.5

The rest of the homepage of Milo can be seen in Figure 3.6. As studies recall [20], the best way to expose items is an image and a brief description of the item.

This has been accomplished in milo displaying the cover of the movie, its title, and its genres. Doing so in very few words the user has a clue about the movie and can thus decide if he/she is interested or not in the movie. If the user is interested, he/she can click on the movie cover to have see the movie description shown in Figure 3.7

The movie description page is thought to show mainly the poster of the movie in full dimension and its trailer. The user then gets two kinds of recommendation: the content dependent one and a user recommended one using a content based or a user based algorithm that the administrator decided. The user can then read the comments about a movie.

In order to simplify the usage and the administration task, Milo integrated an admin interface to generate the algorithm models, download the relative model files, download the ICM and URM matrices and manage the surveys. For the first time the admin was able of placing the right matlab file in a directory, load the admin page, create a model for the newly



Matrix Reloaded (2003)
Views: 2.000
70% of raters liked
Genre: Action, Adventure, Sci-Fi,
0 Comments

Matrix Revolutions (2003)
Views: 2.000
70% of raters liked
Genre: Action, Adventure, Sci-Fi,
0 Comments

Alla ricerca di Nemo (2003)
Views: 2.000
70% of raters liked
Genre: Animation, Adventure, Comedy,
0 Comments

Kill Bill: volume 1 (2003)
Views: 2.000
70% of raters liked
Genre: Action, Crime, Thriller,
0 Comments

City of God (2002)
Views: 2.000
70% of raters liked
Genre: Crime, Drama,
0 Comments

Vanilla Sky (2001)
Views: 2.000
70% of raters liked
Genre: Mystery, Romance, Sci-Fi,
0 Comments

Monsters & Co. (2001)
Views: 2.000
70% of raters liked
Genre: Animation, Adventure, Comedy,
0 Comments

Amores perros (2000)
Views: 2.000
70% of raters liked
Genre: Drama, Thriller,
0 Comments

Il gladiatore (2000)
Views: 2.000
70% of raters liked
Genre: Action, Adventure, Drama,
0 Comments

Alla ricerca di Nemo
A father-son underwater adventure featuring Nemo, a boy clownfish, stolen from h...

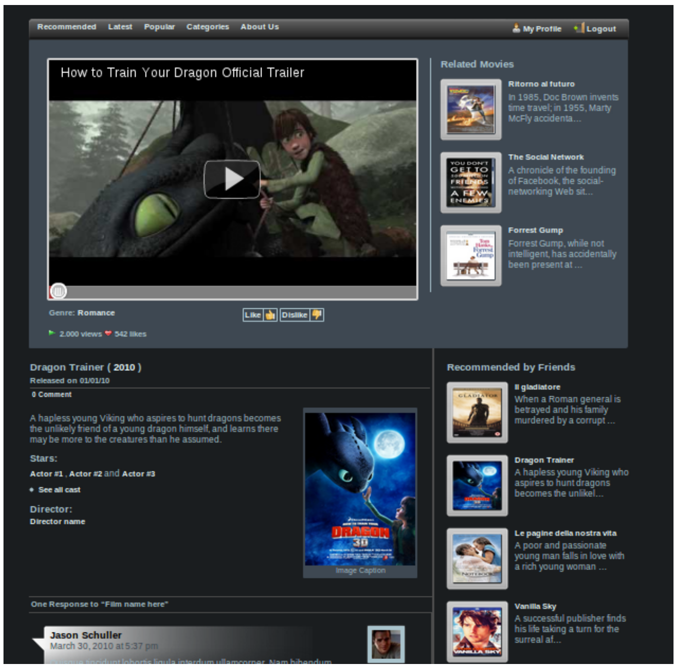
The Millionaire
A Mumbai teen who grew up in the slums, becomes a contestant on the Indian versi...

Kill Bill: volume 1
The Bride wakes up after a long coma. The baby that she carried before entering ...

Toy story - Il mondo dei giocattoli
A cowboy toy is profoundly threatened and jealous when a fancy spaceman toy supp...

Toy story 2 - Woody e Buzz alla riscossa
When Woody is stolen by a toy collector, Buzz and his friends vow to rescue him,...

Figure 3.6: Milo homepage



How to Train Your Dragon Official Trailer

Genre: Romance
2.000 views 542 likes

Ritorno al futuro
In 1985, Doc Brown invents time travel; in 1955, Marty McFly accidenta...

The Social Network
A chronicle of the founding of Facebook, the social-networking Web sit...

Forrest Gump
Forrest Gump, while not intelligent, has accidentally been present at ...

Dragon Trainer (2010)
Released on 01/01/10
0 Comment

A hapless young Viking who aspires to hunt dragons becomes the unlikely friend of a young dragon himself, and learns there may be more to the creatures than he assumed.

Stars:
Actor #1 , Actor #2 and Actor #3
See all cast

Director:
Director name

Image Caption

One Response to "Film name here"

Jason Schuller
March 30, 2010 at 5:37 pm

Il gladiatore
When a Roman general is betrayed and his family murdered by a corrupt ...

Dragon Trainer
A hapless young Viking who aspires to hunt dragons becomes the unlikel...

Le pagine della nostra vita
A poor and passionate young man falls in love with a rich young woman ...

Vanilla Sky
A successful publisher finds his life taking a turn for the surreal af...

Figure 3.7: Milo movie description page

added algorithm and create a survey for that algorithm without modifying one line of code in the system. This was the major improvement of all the previous systems of this kind.

Milo had a two main problems: the first problem was that it had not a clear idea of what it was going to be, that resulted in a hard to maintain code and set of hacks to make things work as fast as possible. This also made the system weak due to many software bottlenecks caused by different databases and mainly by the available hardware for the project that was way limited. Also the matlab part was not optimized in order to reduce memory and free up space as soon as possible. The second problem was mainly the framework chosen, pyramid, that resulted hard to maintain and over complicated in the template section.

Beside those limitations, Milo was an effective and useful system that was able to correctly manage users and generate new surveys.

3.4 Movish

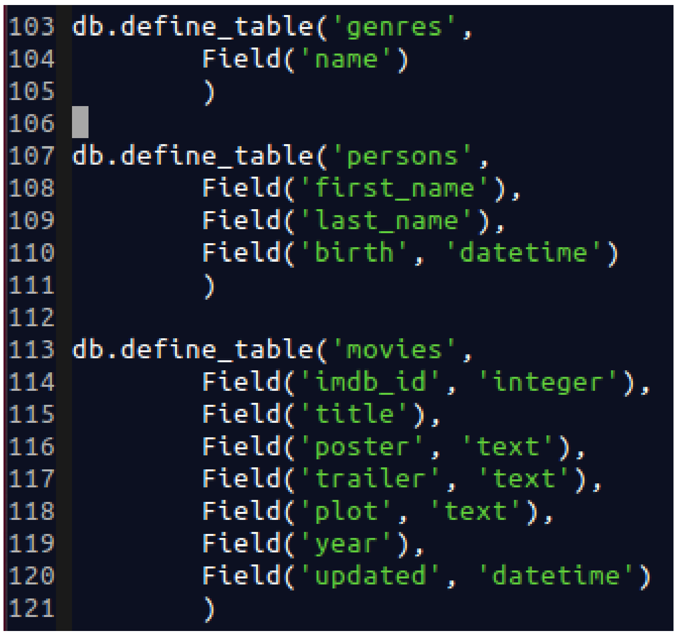
Movish is a step forward from Milo in the fact that it has a clear vision of the fact that it's a platform for research on new recommendation system algorithms. The system is developed in python and uses web2py [3] as main framework. Web2py has been chosen over pyramid because it included all the feature Movish needed:

- MVC. Web2py has a strong MVC concept. The structure of the project defines the paths to reach each single function of the application. Models are easy to define thanks to the Database Abstraction Layer (DAL) which has a direct mapping to Structured Query Language (SQL). Web2py is also capable of doing automatic migrations: as soon as the code that defines a database table is changed, the application recognizes that and runs a SQL ALTER TABLE query to update the database to have the same schema defined by the DAL. The DAL code of the database can be seen in Figures 3.8, 3.9, 3.10, 3.11, 3.12. The Entity Relationship model (ER) diagram can be seen on Figure 3.13. This diagram shows all the relations between the various data structure. The real database is much bigger though. All the data structures used by the web2py framework have been disabled in the graph in order to allow an easy reading of the most important ones.

- Scheduler. Web2py has a power scheduler built in that has been heavily used in Movish. The use of that component is so deep that I have also found some bugs while coding the various tasks. All bugs promptly reported have been fixed in few days after submission. The scheduler allows the requests to be dispatched as fast as they can while allowing longer tasks to be scheduled. After registering the different tasks the framework itself can also trigger them if some conditions are satisfied. This is heavily used to update movies that have incomplete information.

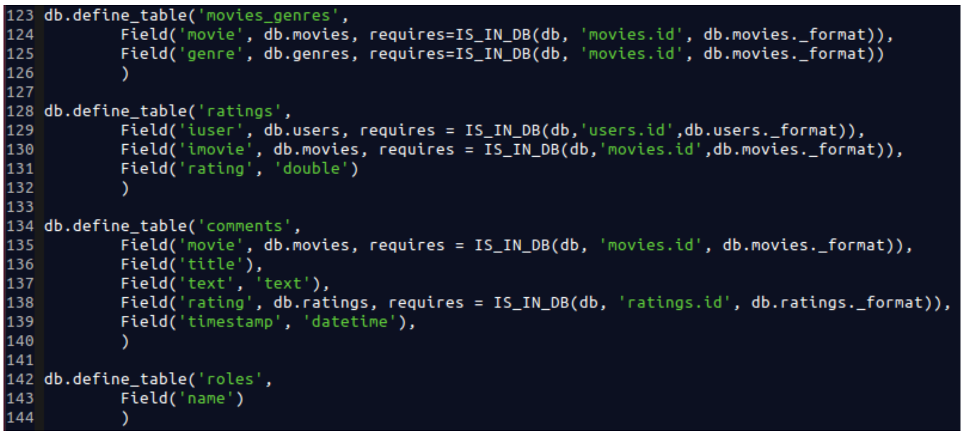
The main data structures are the one for storing information about the user called *auth_user* and *users*, the one for storing the movies called *movies* and the one for the *surveys*. *auth_user* is web2py specific while *users* is an extension of *auth_user* in order to support additional field while keeping it separated from the web system. This is so because there are two kind of users in the system. The ones that have registered from the web interface and the ones that have been added from the crawler that gets data from imdb.com [7]. We will talk about this later. For now just mind that we don't want to pollute the *auth_user* table with crawled user that will likely never use the system.

The *movies* data structure stores all the information about a movie and it is i a many-to-many relationship with either *features*, *genres* and *persons_in_movies*. *ratings* are also a central data structure that stores all the rating in a relative



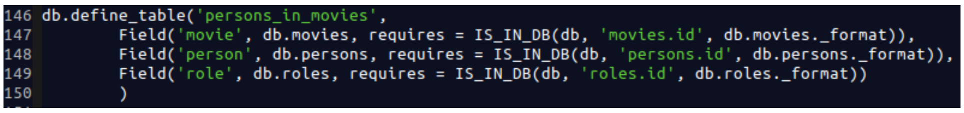
```
db.define_table('genres',
        Field('name')
        )

db.define_table('persons',
        Field('first_name'),
        Field('last_name'),
        Field('birth', 'datetime')
        )

db.define_table('movies',
        Field('imdb_id', 'integer'),
        Field('title'),
        Field('poster', 'text'),
        Field('trailer', 'text'),
        Field('plot', 'text'),
        Field('year'),
        Field('updated', 'datetime')
        )
```

Figure 3.8: Database definition (part 1)



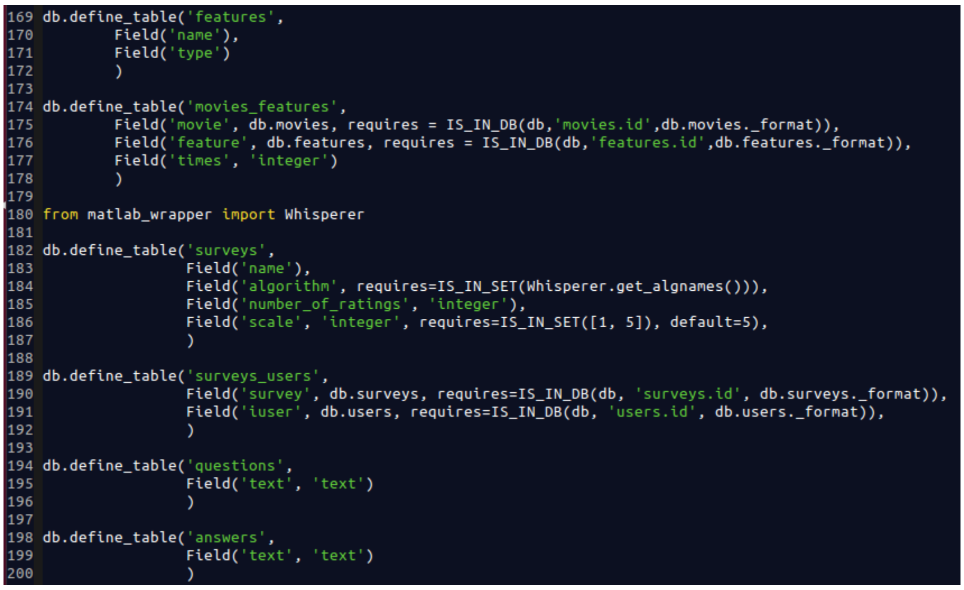
```
db.define_table('movies_genres',
        Field('movie', db.movies, requires=IS_IN_DB(db, 'movies.id', db.movies._format)),
        Field('genre', db.genres, requires=IS_IN_DB(db, 'movies.id', db.movies._format))
        )

db.define_table('ratings',
        Field('iuser', db.users, requires = IS_IN_DB(db,'users.id',db.users._format)),
        Field('imovie', db.movies, requires = IS_IN_DB(db,'movies.id',db.movies._format)),
        Field('rating', 'double')
        )

db.define_table('comments',
        Field('movie', db.movies, requires = IS_IN_DB(db, 'movies.id', db.movies._format)),
        Field('title'),
        Field('text', 'text'),
        Field('rating', db.ratings, requires = IS_IN_DB(db, 'ratings.id', db.ratings._format)),
        Field('timestamp', 'datetime'),
        )

db.define_table('roles',
        Field('name')
        )
```

Figure 3.9: Database definition (part 2)



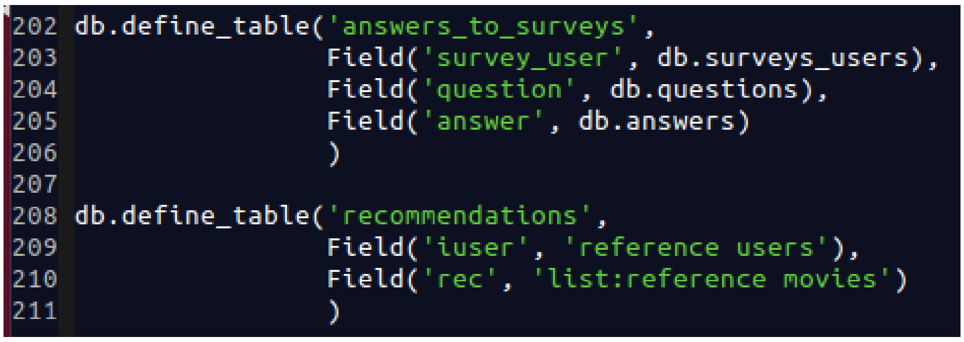
```
db.define_table('persons_in_movies',
        Field('movie', db.movies, requires = IS_IN_DB(db, 'movies.id', db.movies._format)),
        Field('person', db.persons, requires = IS_IN_DB(db, 'persons.id', db.persons._format)),
        Field('role', db.roles, requires = IS_IN_DB(db, 'roles.id', db.roles._format))
        )
```

Figure 3.10: Database definition (part 3)



```
db.define_table('features',
        Field('name'),
        Field('type')
        )

db.define_table('movies_features',
        Field('movie', db.movies, requires = IS_IN_DB(db,'movies.id',db.movies._format)),
        Field('feature', db.features, requires = IS_IN_DB(db,'features.id',db.features._format)),
        Field('times', 'integer')
        )

from matlab_wrapper import Whisperer

db.define_table('surveys',
                Field('name'),
                Field('algorithm', requires=IS_IN_SET(Whisperer.get_algnames())),
                Field('number_of_ratings', 'integer'),
                Field('scale', 'integer', requires=IS_IN_SET([1, 5]), default=5),
                )

db.define_table('surveys_users',
                Field('survey', db.surveys, requires=IS_IN_DB(db, 'surveys.id', db.surveys._format)),
                Field('iuser', db.users, requires=IS_IN_DB(db, 'users.id', db.users._format)),
                )

db.define_table('questions',
                Field('text', 'text')
                )

db.define_table('answers',
                Field('text', 'text')
                )
```

Figure 3.11: Database definition (part 4)



```
db.define_table('answers_to_surveys',
                Field('survey_user', db.surveys_users),
                Field('question', db.questions),
                Field('answer', db.answers)
                )

db.define_table('recommendations',
                Field('iuser', 'reference users'),
                Field('rec', 'list:reference movies')
                )
```

Figure 3.12: Database definition (part 5)

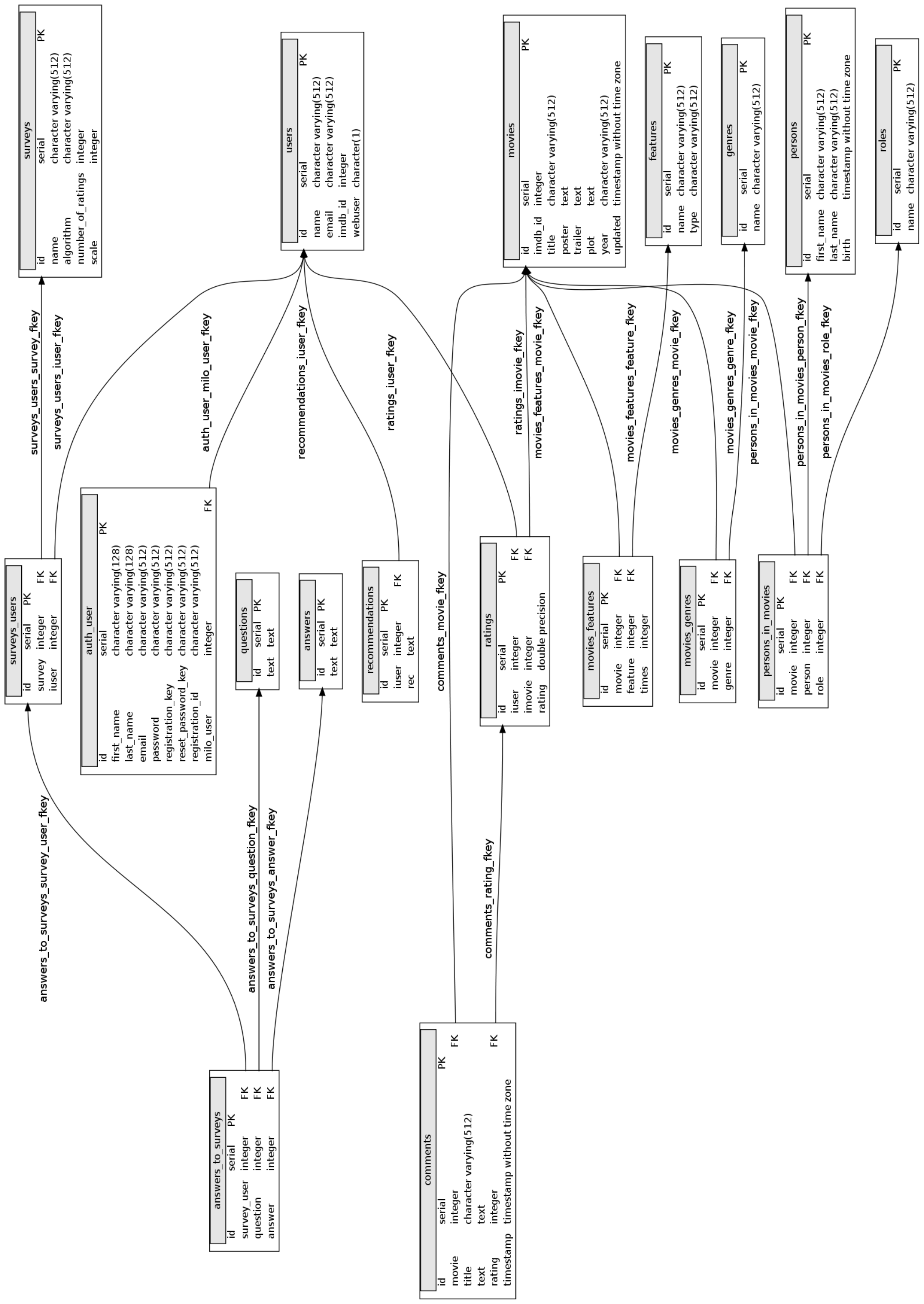


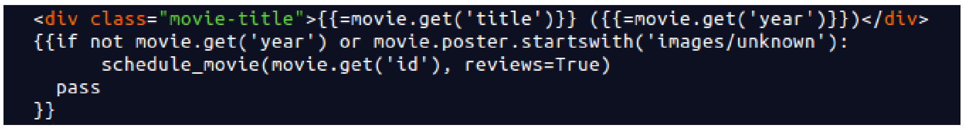
Figure 3.13: Database ER diagram

0 to 1 float number where 0 is not rated, and 1 is rated as the most awesome movie. Since various algorithms use different scales, this representation allows to use a common scale and then do the required conversion then the URM is generated for a particular algorithm. The *surveys* data structure takes into account the users allowed to perform a survey, the user answers and the algorithm used. The *comments* data structure stores the information about the comments that are associated to a movie and a rating. The *persons_in_movies* in Figure 3.10 links the persons that can be an actor or a director to a movie. The *recommendations* data structure stores the latest recommendation for a given user and it is the only table that uses the embedded serialization of fields available from DAL. In particular storing a list of references in a table like on Figure 3.12 it is possible to compress a set of many-to-many entries in a single entry. It is the DAL that will care of serializing and deserializing the objects.

As shown in the previous figures the DAL makes really easy to define new tables using the *db.define_table* function which accept the table name as first parameter and then a variable number of Field objects. The Field object take at least one parameter which is the name of the field and optionally a type which describes the kind of data. The data can be of the following types: string, text, blob, boolean, integer, double, decimal, date, time, datetime, password, upload, reference, list:string, list:integer, list:reference, bigint, big-id, big-reference.

As stated before, one of the main advantages of Movish is the heavy use of the scheduler in order to:

- **Update the dataset**. It can be updated in two ways: by administrator action or automatically. The administrator can perform the update of all the dataset or import the most popular movies of imdb.com, the coming soon movies or in cinema now movies by clicking the corresponding button in the admin panel. The click will add a task to



```
<div class="movie-title">{{=movie.get('title')}} ({{=movie.get('year')}})</div>
{{if not movie.get('year') or movie.poster.startswith('images/unknown'):
      schedule_movie(movie.get('id'), reviews=True)
  pass
}}
```

Figure 3.14: Schedule movie update code

the scheduler that as soon as it detects a free worker will assign the task. While a movie is displayed, if there are not enough information about that movie, the system automatically generates a task to update that movie.

The relevant code about the auto update of a movie is in Figure 3.14. It is part of the view which is the last piece of code executed before sending the output back to the user browser.

Web2py allows to put pure python code into the Hypertext Markup Language (html) pages in order to perform any kind of operation and also embed small logic related to the view. This flexibility is used to detect that if the movie has no year set and the poster is set to the default unknown poster which is *images/unknown* then schedule to retrieve a movie by its id and parse all the reviews of the movie in imdb.com.

- *Create model*. As explained in Chapter 2, the creation of a model is an expensive operation. This characteristic allows it to be suitable for a scheduled task. In fact the administrator can create the model for each supported algorithm clicking the associated button in the administrator interface. Alternatively, the model for a single algorithm is generated as soon as a survey is generated to include the new users that may have been added for the survey. In fact, doing a recommendation for an user which was not in the URM at the moment of model creation will cause an error. This is why it is mandatory to schedule a new model for every newly submitted survey that adds new users to the system.

- **Create of the ICM and URM**. The creation of the two matrices is also an expensive query for the database this is why there is also a task for creating and storing them into matlab as a sparse matrices for easy retrieval.
- **Features and titles vectors**. For conformity also the two features and title vectors of all the items in the database are created using a task. The titles vector is a simple vector with the id and all the titles of the movies in the database. The features vector lists, for each item in the database all the metadata, or features, for that item.
- **Survey creation**. The creation of a survey is composed by different steps: adding of the new users to the system with an auto generated password, the creation of the model, and an informational mail to each user of the survey telling that the survey is ready and that they can login and perform the survey.

Another main component of Movish that was missing in previous systems is the presence of a full featured crawler that uses many sources to retrieve all the needed information for a movie. The crawler is build in a modular and extensible way in order to allow the adding of different sources in the future. As from the time this thesis is being written, the crawler supports the following sources: imdb.com [7] via imdbpy [33] library or by raw html parsing via the lxml [34] library and youtube.

Imdbpy is a python package useful to retrieve and manage the data of the imdb movie database about movies, people, characters and companies. It is well maintained an documented and saves the duty to parse the imdb pages by hand. Unfortunately it does support imdb users and reviews so in order to get the reviews of the users from imdb also a raw html parser has been implemented using the lxml library.

lxml is the most feature-rich and easy-to-use library for processing XML and HTML in python. Benchmarks available on their website shows that the library performs better than any known library in generating and parsing xml streams. It is also the library suggested by the python community. Youtube is accessed via the http RESTful api in order to search movie trailers. It simply searches for the title of the movie appending the word *trailer* to it and picks up the first non sponsored result of the search if any.

3.4.1 State of the art issues

In section 2.2 we discussed about the issues that may affect a recommendation system. Every real world recommendation system has to address those problems. Movish tries to solve them in this way:

- **list of metadata**. Movish uses a tagger on the whole plot to retrieve the maximum number of metadata. It only removes all the conjunctions and stopwords.
- **cold start**. In Movish the user is forced, during survey mode that will be exposed in chapter 5, to perform a number of ratings that the administrator set at survey creation.
- **recommendation freshness**, thanks to the importer we will analyze on chapter 4.2.4, Movish answer to this issue being able to automatically fetch new movies from imdb.com including the ratings and thus the system will have new movies with also ratings. This makes the algorithms be able to use the new movies also at model creation even if the movies have never being displayed on Movish.

Other issues are algorithm specific and unable to be solved from Movish. Movish goal is to provide the best up to date environment of URM, ICM, titles and features vector for an algorithm that has to perform a recommendation.

3.5 Conclusions

Movish is the first fully automatic recommendation system survey platform. It allows to test different algorithms via surveys without editing any source file of the system. It is inspired by ContentWise and Milo but it introduces a completely new flexible architecture to minimize the administrator workload in keeping the system up to date and ready for testing new algorithms. Movish represent a innovative and fresh approach to the problem of an up to date dataset and optimized environment for recommendation generation.

Chapter 4

Movish system

4.1 Introduction

In this chapter we will analyze the Movish automatic recommendation system deeply in all its aspects and we will show how the various challenges of implementing a real world recommendation system can be solved by either architecture or smart intuitions.

The reader will be guided on a tour that will include the project architecture, the project directory structure, the mechanism for automatic discovery of algorithms, the item crawler and the mechanism to hide the generated traffic in order to do not cause traffic throttling by the contacted servers.

4.2 Architecture

Differently from ContentWise, a monolithic application, and from Milo, a fully separated and modular application, Movish sits in the middle: it is a modular application. The Graphical User Interface (GUI) and the recommendation engine are in the same application but are two different modules that cooperate in order to limit the point of failures (that were too spread on Milo) and allow flexibility.

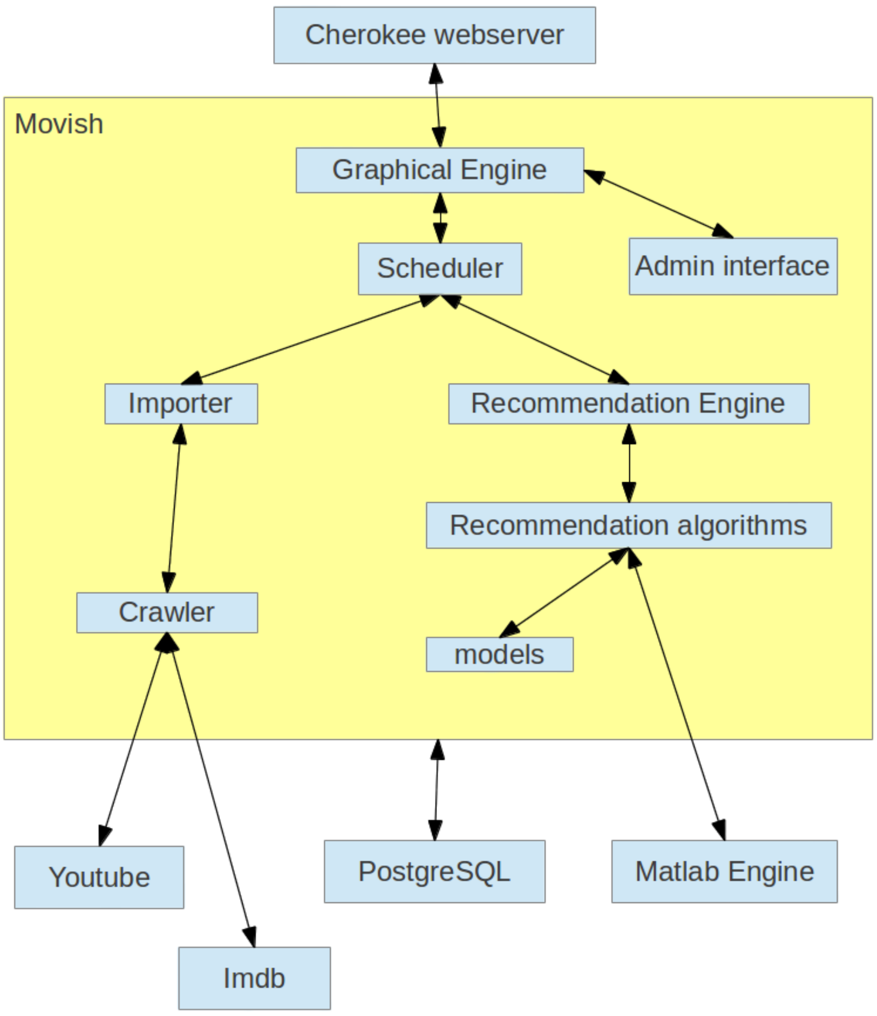


Figure 4.1: Movish architecture

The architecture of the system can be see on Figure 4.1. The system is exposed to internet(working) thanks to a cherokee web server [35]. Cherokee is an innovative, feature rich, and yet easy to configure open source web server. Its goal is to be as fast as it can and serve as many user as the hardware can hold. It is also C10K problem [36] aware. The C10k problem is the problem of serving 10.000 concurrent connections with commodity hardware, it has been well studied and only few webservers have been designed to address this problem.

Cherokee has been chosen for this project because it resulted to be both flexible and fast. Other webserver have been evaluated like nginx [37] and apache werbserver [38]. The first one was very good in benchmarks but lacked in flexibility, the latter was very modular but less likely to scale as soon users grow or the system is under heavy load, which is the case while a recommendation is performed while a model is being created.

Movish core modules are the graphical engine and the scheduler which have their basis on the web2py [3] framework. The graphical engine speaks to the webserver thank to an application handler called uWSGI [39]. uWSGI is an extremely advanced, sysadmin-friendly, highly-modular application container server written in POSIX-compatible C. It is the glue that connect the webserver with the application. As for the webserver, uWSGI has been chosen for its benchmarks and its lightweight.

The application relays in two other software and two services to work:

- A matlab [4] engine executable installed in the local appliance. This is needed to perform the various operations with the algorithms that are written in matlab.
- A PostgreSQL [32] database either remote or local. The database is the primary source of information for various parts of the executable. It is mainly used by the graphical engine, to get information to display about movies and users, and from the scheduler to coordinate all the

workers for all the tasks. The importer also uses the database in order to store the crawled data.

- Imdb.com. An internet access is required to access imdb.com as a primary source for new movies and useful information like release dates, plot, reviews and thus ratings. In fact the whole basic dataset has been built from the ground up crawling information about popular e latest movies of Imdb.com
- Youtube.com or youtu.be. As for imdb.com, an internet connection is required to reach this service. The youtube source is used to search and retrieve the correct trailer to bind to the movie at database level.

Movish is a complex application that can be divided in different areas:

- **Graphical engine**. This is the core of the Model View Controller (MVC) structure. It is responsible of answering the requests from the user, elaborate data and display the output.
- **Admin interface**. This is the administrator specific part that is used to administrate the whole application from model creation, data retrieval, configuration of the algorithms, importing of new movies, administrate the scheduler, create new surveys and retrieve system information and status.
- **Scheduler**. This component has the main task of taking computational intensive operations and execute in a separate process, then populate the database with the result of the computation. Since most of the operations with matrices are very computational intensive this module is heavy used by all the parts of the application.
- **Importer**. It is the component that calls the crawler and manages the information from the crawler in an ordered way for storing into the database.

- **Crawler**. It is responsible for getting all the information it can from imdb.com and youtube.com based on a movie or a set of movies. It is also able to parse imdb user pages and their ratings. It is most used by the task that update a movie given a movie id.

- **Recommendation engine**. This component takes care of managing the communication with the Matlab engine. Since most of the operations of this module are very computationally expensive, this module is very used by the scheduler.

- **Recommendation algorithms**. Those are all the matlab algorithms placed in a specific folder of the project. There is a logic to detect all the algorithms in a sub tree of directories.

- **Models**. As described in section 2.2.3, models need to be created to perform the recommendation. This component manages all the created models for each algorithm in the system.

4.2.1 Graphical Engine

The graphical engine takes advantage of the web2py [3] framework. A request is taken by the controller, the correct function to be executed is selected thank to the automatic route that find the function based on the project directory structure. When the function is executed it retrieves data from the database and populates a html template that is thus sent back to the user browser.

The graphical engine has two modes: normal and admin mode. Only users that have been promoted to admin are allowed to see the admin mode. To switch from one mode another the user can click on the admin link right after the login. The link is visible in figure 4.2.

In Figure 4.3 the reader can see the movish.co homepage as seen by a normal logged in or not logged in user which resembles the normal mode.

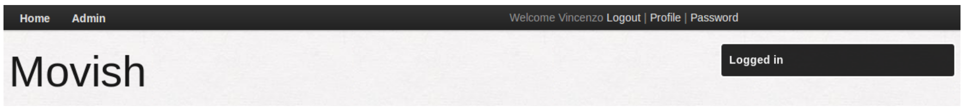


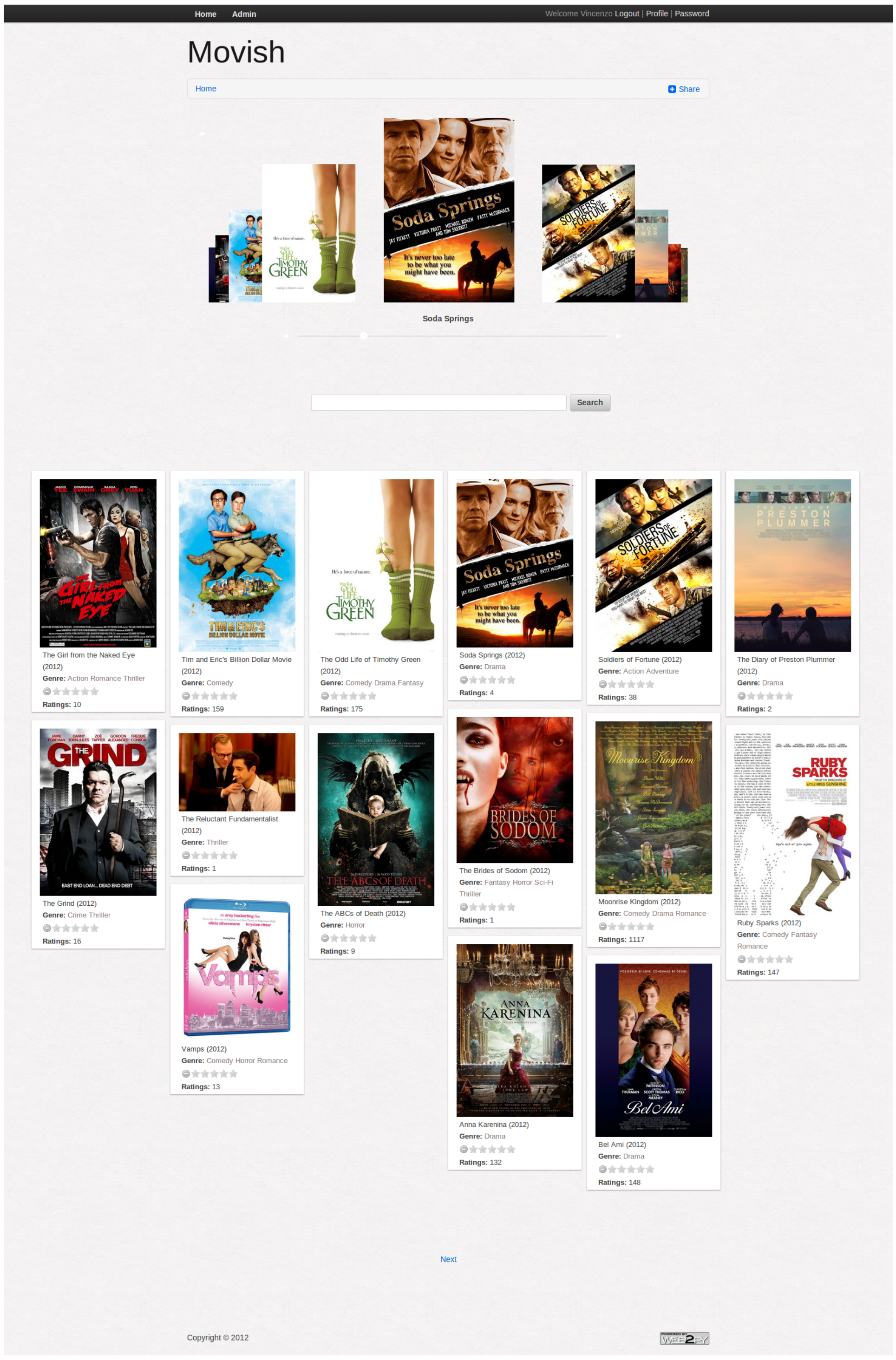
Figure 4.2: Admin and normal mode

The layout has been inspired by pinterest, a popular social network for images but it has some peculiarities that makes it different for different reasons. The main difference from pinterest is the presence of a slider placed exactly at line of sight of the user, in the right position after the header and the Movish logo. The slider displays two sets of data: if the user is not logged in, it displays the most popular movies, if the user is logged in, it displays a recommendation generated by a random algorithm or a administrator defined one. So the user is forced, at first, to have a look at the recommendation or at the most popular movies. Scrolling down the page the user can start browsing the catalog via a search field or by seeing the different pages of the newly added items. In fact, by default, the list of movies that appears in the homepage represent the recently updated or inserted movies.

The fact that having a picture and a small text is the best way to display items [15] has been kept displaying the movie cover, the title, the release year and the genres. This allows to fit a movie in a small portion of the page but display enough amount of information to the user.

The user can then click on a single cover of a movie or a title to open a second page that displays the details about a movie like on figure 4.4.

The movie detail page will display the title, the imdb reference the genres, the plot, the members of the cast and the directors. On the side there is a big version of the movie cover that is used also in the listing of the catalog page. If there are comments of the users about the movie those are shown at the bottom of the page. If there is a trailer of the movie it will be also displayed but not played by default.



The Girl from the Naked Eye (2012)
Genre: Action Romance Thriller
Ratings: 10

Tim and Eric's Billion Dollar Movie (2012)
Genre: Comedy
Ratings: 159

The Odd Life of Timothy Green (2012)
Genre: Comedy Drama Fantasy
Ratings: 175

Soda Springs (2012)
Genre: Drama
Ratings: 4

Soldiers of Fortune (2012)
Genre: Action Adventure
Ratings: 38

The Diary of Preston Plummer (2012)
Genre: Drama
Ratings: 2

The Grind (2012)
Genre: Crime Thriller
Ratings: 16

The Reluctant Fundamentalist (2012)
Genre: Thriller
Ratings: 1

The ABCs of Death (2012)
Genre: Horror
Ratings: 9

The Brides of Sodom (2012)
Genre: Fantasy Horror Sci-Fi Thriller
Ratings: 1

Moonrise Kingdom (2012)
Genre: Comedy Drama Romance
Ratings: 1117

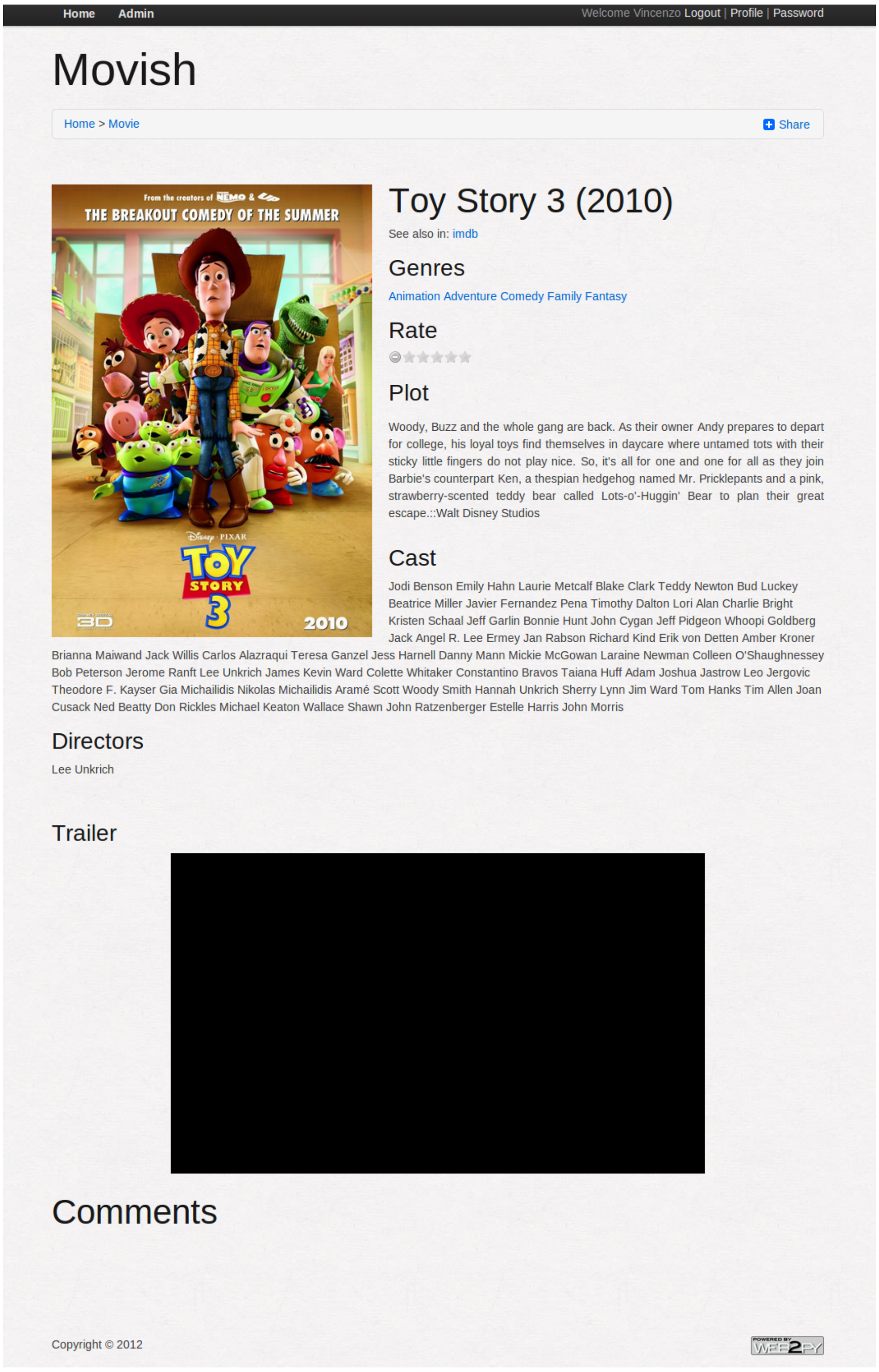
Ruby Sparks (2012)
Genre: Comedy Fantasy Romance
Ratings: 147

Vamps (2012)
Genre: Comedy Horror Romance
Ratings: 13

Anna Karenina (2012)
Genre: Drama
Ratings: 132

Bel Ami (2012)
Genre: Drama
Ratings: 148

Figure 4.3: Movish homepage


Home
Admin
Welcome Vincenzo Logout | Profile | Password
Share
From the creators of NEMO &
THE BREAKOUT COMEDY OF THE SUMMER
Disney · PIXAR
TOY STORY 3
3D
2010

Movish

Home > Movie

Toy Story 3 (2010)

See also in: imdb

Genres

Animation Adventure Comedy Family Fantasy

Rate

Plot

Woody, Buzz and the whole gang are back. As their owner Andy prepares to depart for college, his loyal toys find themselves in daycare where untamed tots with their sticky little fingers do not play nice. So, it's all for one and one for all as they join Barbie's counterpart Ken, a thespian hedgehog named Mr. Pricklepants and a pink, strawberry-scented teddy bear called Lots-o'-Huggin' Bear to plan their great escape.::Walt Disney Studios

Cast

Jodi Benson Emily Hahn Laurie Metcalf Blake Clark Teddy Newton Bud Luckey Beatrice Miller Javier Fernandez Pena Timothy Dalton Lori Alan Charlie Bright Kristen Schaal Jeff Garlin Bonnie Hunt John Cygan Jeff Pidgeon Whoopi Goldberg Jack Angel R. Lee Ermey Jan Rabson Richard Kind Erik von Detten Amber Kroner Brianna Maiwand Jack Willis Carlos Alazraqui Teresa Ganzel Jess Harnell Danny Mann Mickie McGowan Laraine Newman Colleen O'Shaughnessey Bob Peterson Jerome Ranft Lee Unkrich James Kevin Ward Colette Whitaker Constantino Bravos Taiana Huff Adam Joshua Jastrow Leo Jergovic Theodore F. Kayser Gia Michailidis Nikolas Michailidis Aramé Scott Woody Smith Hannah Unkrich Sherry Lynn Jim Ward Tom Hanks Tim Allen Joan Cusack Ned Beatty Don Rickles Michael Keaton Wallace Shawn John Ratzenberger Estelle Harris John Morris

Directors

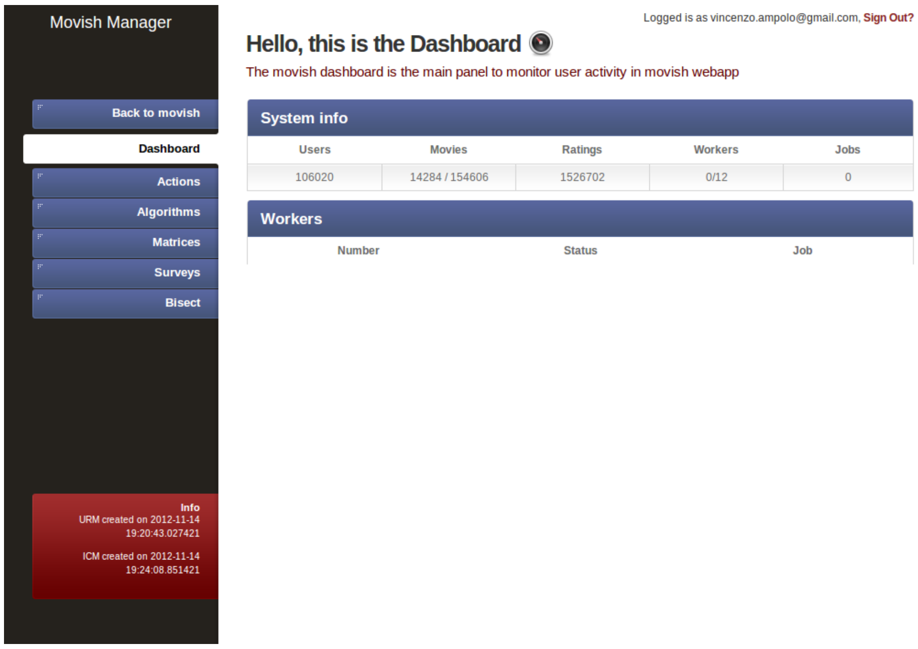
Lee Unkrich

Trailer

Comments

Copyright © 2012

Figure 4.4: Movie detail



Logged is as vincenzo.ampolo@gmail.com, Sign Out?

Hello, this is the Dashboard

The movish dashboard is the main panel to monitor user activity in movish webapp

System info

Users	Movies	Ratings	Workers	Jobs
106020	14284 / 154606	1526702	0/12	0

Workers

Number	Status	Job

Info
URM created on 2012-11-14 19:20:43.027421

ICM created on 2012-11-14 19:24:08.851421

Figure 4.5: Movish admin mode homepage

The detail page has been designed to be as clear and simple as possible displaying all the information about the movie in a single page. There is also a share button to share the link of a specific movie to a social network of your choice including Facebook, Twitter and Google plus.

4.2.2 Admin interface

The admin mode is shown in figure 4.5. The admin has a different layout with a big menu on the left and a white space on the right in order to display information and content. One of the main features of Movish is that the whole system can be managed by this admin interface.

The main page, or the dashboard, shows useful information of the status of the system. It displays the number of users in the system, the number of movies with full information, the number of movies that lack some information, the number of ratings in the system, the number of active and total workers and the number of pending tasks/jobs. In the dashboard there is

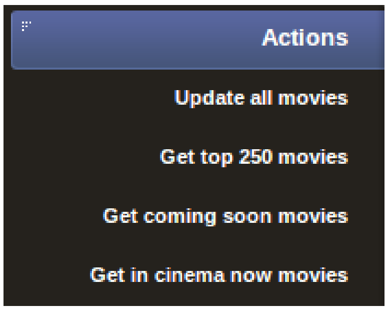


Figure 4.6: Actions menu

also a table that is automatically generated with all the running worker and the task that they are working on. So the admin can see in which state is the system just loading a single page.

Every menu self expands as soon as the user clicks on it using a javascript function that enables a light animation.

The actions menu in figure 4.6 shows all the possible actions that the admin can do to the system. The admin can launch an update all movies task, a get top 250 movies on imdb task, a get coming soon movies from imdb task or get in cinema now movies task. Each click will generate and Ajax [40] call that will add the respective task to the scheduler. For each action the user receives a message saying that the operation has been added to the system. Each task will then be picked up by a worker selected by the scheduler and will execute it. The output of the task is directly written in the database and it is basically an insert or update operation on the database.

When the Algorithms menu in figure 4.7 is clicked, a list of all the algorithms located in the *modules/algorithms* sub-directory of the project is created and displayed. Each algorithm can be clicked. As soon as it is, the information about that specific algorithm is displayed. Figure 4.7 displays all the algorithms that are available in the system as for now.

Adding a new algorithms is easy: the administrator has to add all the

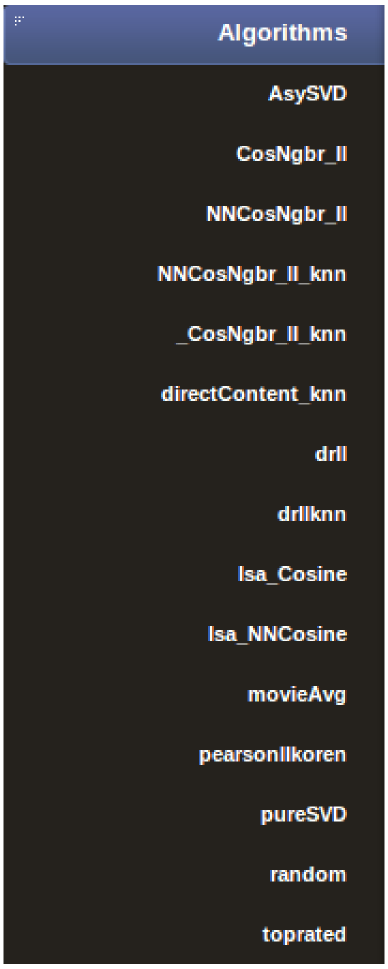
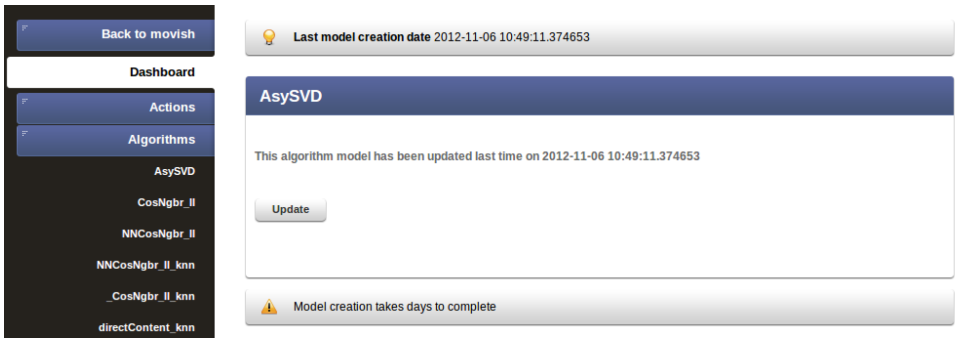


Figure 4.7: Algorithms menu

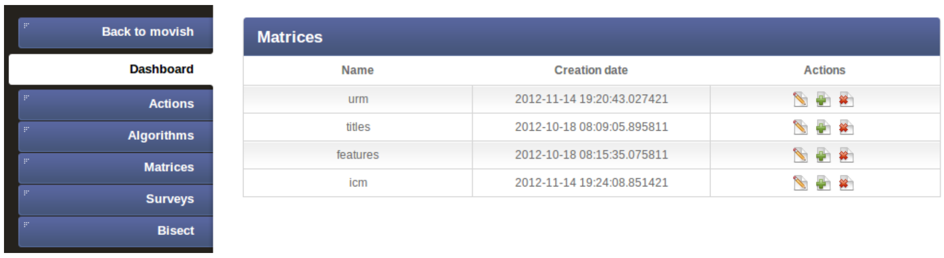


Last model creation date 2012-11-06 10:49:11.374653

This algorithm model has been updated last time on 2012-11-06 10:49:11.374653

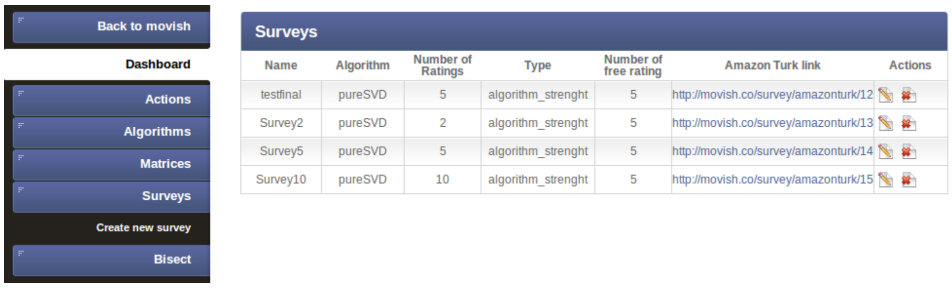
Model creation takes days to complete

Figure 4.8: Algorithm detail



Name	Creation date	Actions
urm	2012-11-14 19:20:43.027421	
titles	2012-10-18 08:09:05.895811	
features	2012-10-18 08:15:35.075811	
icm	2012-11-14 19:24:08.851421	

Figure 4.9: Matrices menu detail



Name	Algorithm	Number of Ratings	Type	Number of free rating	Amazon Turk link	Actions
testfinal	pureSVD	5	algorithm_strenght	5	http://movish.co/survey/amazonturk/12	
Survey2	pureSVD	2	algorithm_strenght	5	http://movish.co/survey/amazonturk/13	
Survey5	pureSVD	5	algorithm_strenght	5	http://movish.co/survey/amazonturk/14	
Survey10	pureSVD	10	algorithm_strenght	5	http://movish.co/survey/amazonturk/15	

Figure 4.10: Surveys menu

relevant files about an algorithm to the *modules/algorithms* directory. The system in fact has automatic discovering facilities to detect all the available algorithms. The algorithms should respect a common set of input and a common set of output. The reader can discover more about how to add an algorithm on section 4.3.

Figure 4.9 shows the detail of the matrices in the system. It is displayed if the administrator clicks on the **Matrices** menu item. It shows the presence of the item content matrix (ICM), user rating matrix (URM), titles an features vector. From the **action** column the user can select to download the matrice or vector, to schedule the creation of a new one or to delete it.

Clicking on the **Surveys** button in the menu will open a sub menu for creating a new survey, called **Create new survey**, and will show all the surveys in the system. As for the **Matrices** menu, there is a *Actions* column to download the data about the survey, or to delete it from the system. Clicking on **Create new survey** will cause the content portion of the page to load a form for submitting a new survey. The reader can find more about the survey subsystem including survey creation or adding a new survey type on chapter 5.

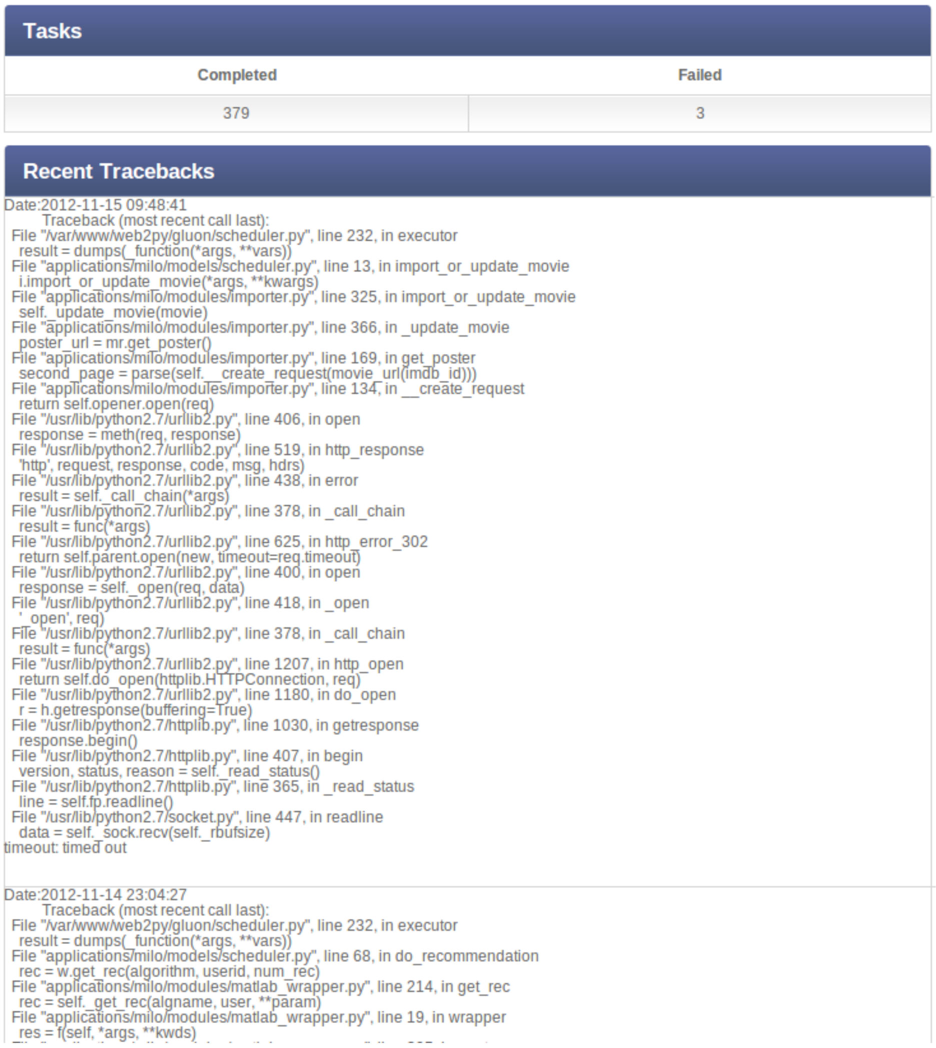
Figure 4.11 shows the last menu item, **Bisect**. This is for meant for debugging purposes. In Bisect the administrator can see the total complete tasks and the failed ones. For the last ten failed tasks there is a detailed traceback with useful information about the error and hopefully in how to fix it. In the figure there is a problem with a timed out connection for example. Of course the **Bisect** section is not meant to do be a full debugger but it may be really helpful to discover bad uses of the software without even opening an code editor.

4.2.3 Scheduler

The scheduler is a fundamental part of the system. It is embedded in the web2py [3] framework and is formed by workers that implement a distributed way to assign task to each worker and solve it. It is perfectly integrated with the underlying database which allows easy monitoring and modifying like exposed in section 4.2.2 in the admin dashboard.

The scheduler is mainly configured in file *models/scheduler.py* which is shown in Figure 4.12 and Figure 4.13. It mainly imports the importer module of which we will talk in section 4.2.4 and encapsulates the most important functions in scheduler tasks. The defined functions are:

- **create_model**. This function takes a name of an algorithm as its input and creates a model for the given algorithm. If the URM and the ICM are not in the system, they are created too.
- **import_or_update_movie**. This is one of the most important scheduler tasks. It gets scheduled as soon as a movie with no title, no cover or no year information is displayed to the user. It fetches the infor-



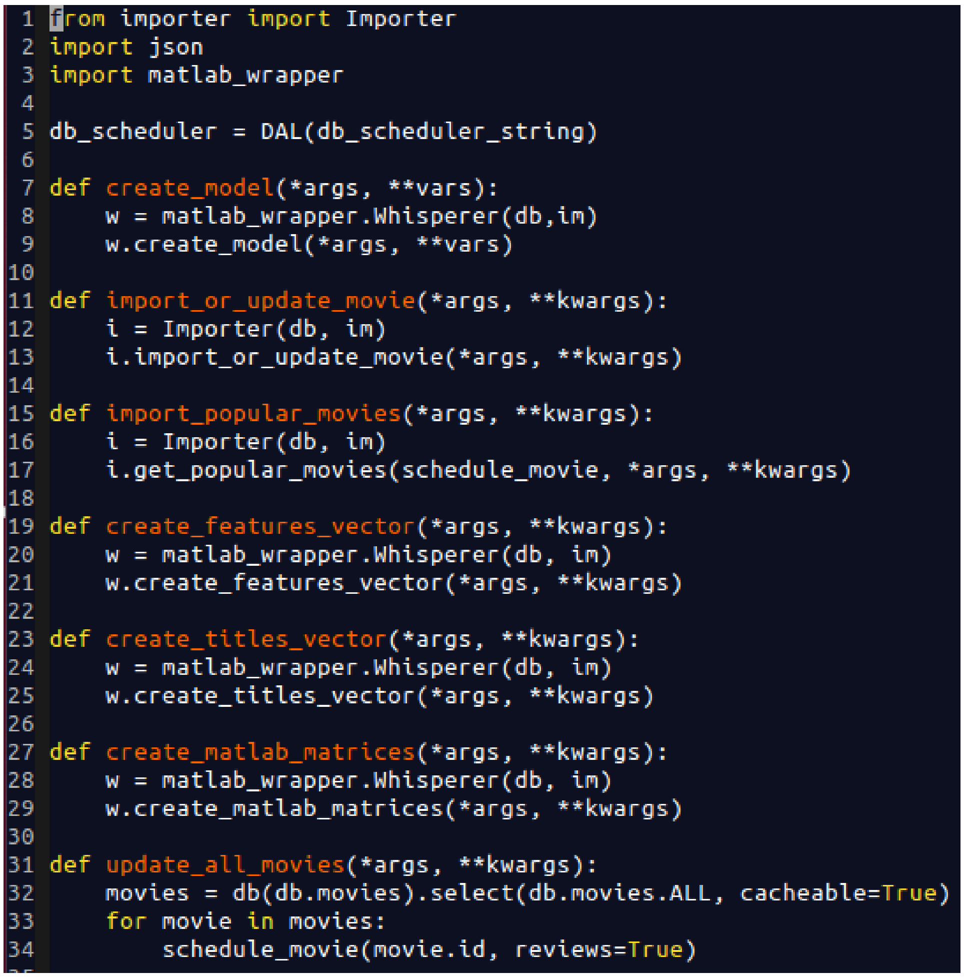
Tasks

Completed	Failed
379	3

Recent Tracebacks

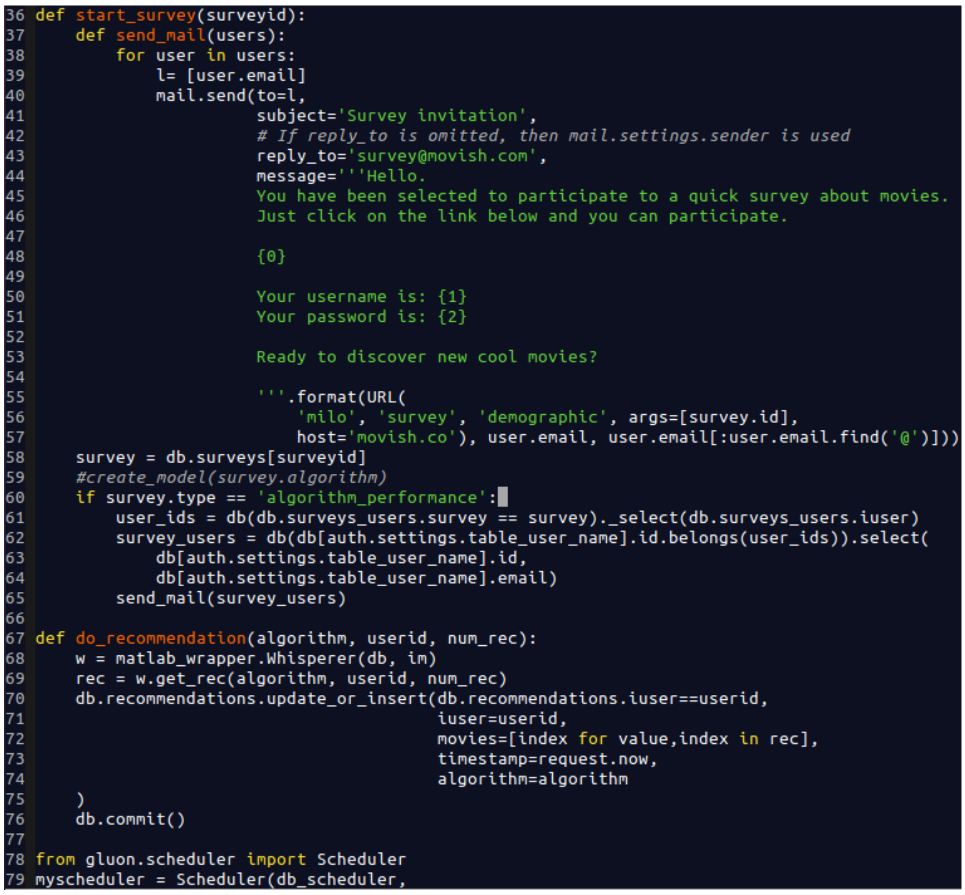
```
Date:2012-11-15 09:48:41
        Traceback (most recent call last):
 File "/var/www/web2py/gluon/scheduler.py", line 232, in executor
  result = dumps(_function(*args, **vars))
 File "applications/milo/models/scheduler.py", line 13, in import_or_update_movie
  i.import_or_update_movie(*args, **kwargs)
 File "applications/milo/modules/importer.py", line 325, in import_or_update_movie
  self._update_movie(movie)
 File "applications/milo/modules/importer.py", line 366, in _update_movie
  poster_url = mr.get_poster()
 File "applications/milo/modules/importer.py", line 169, in get_poster
  second_page = parse(self.__create_request(movie_url(imdb_id)))
 File "applications/milo/modules/importer.py", line 134, in __create_request
  return self.opener.open(req)
 File "/usr/lib/python2.7/urllib2.py", line 406, in open
  response = meth(req, response)
 File "/usr/lib/python2.7/urllib2.py", line 519, in http_response
  'http', request, response, code, msg, hdrs)
 File "/usr/lib/python2.7/urllib2.py", line 438, in error
  result = self._call_chain(*args)
 File "/usr/lib/python2.7/urllib2.py", line 378, in _call_chain
  result = func(*args)
 File "/usr/lib/python2.7/urllib2.py", line 625, in http_error_302
  return self.parent.open(new, timeout=req.timeout)
 File "/usr/lib/python2.7/urllib2.py", line 400, in open
  response = self._open(req, data)
 File "/usr/lib/python2.7/urllib2.py", line 418, in _open
  '_open', req)
 File "/usr/lib/python2.7/urllib2.py", line 378, in _call_chain
  result = func(*args)
 File "/usr/lib/python2.7/urllib2.py", line 1207, in http_open
  return self.do_open(httplib.HTTPConnection, req)
 File "/usr/lib/python2.7/urllib2.py", line 1180, in do_open
  r = h.getresponse(buffering=True)
 File "/usr/lib/python2.7/httplib.py", line 1030, in getresponse
  response.begin()
 File "/usr/lib/python2.7/httplib.py", line 407, in begin
  version, status, reason = self._read_status()
 File "/usr/lib/python2.7/httplib.py", line 365, in _read_status
  line = self.fp.readline()
 File "/usr/lib/python2.7/socket.py", line 447, in readline
  data = self._sock.recv(self._rbufsize)
timeout: timed out


Date:2012-11-14 23:04:27
        Traceback (most recent call last):
 File "/var/www/web2py/gluon/scheduler.py", line 232, in executor
  result = dumps(_function(*args, **vars))
 File "applications/milo/models/scheduler.py", line 68, in do_recommendation
  rec = w.get_rec(algorithm, userid, num_rec)
 File "applications/milo/modules/matlab_wrapper.py", line 214, in get_rec
  rec = self._get_rec(algname, user, **param)
 File "applications/milo/modules/matlab_wrapper.py", line 19, in wrapper
  res = f(self, *args, **kwds)
```

Figure 4.11: Bisect menu



```
from importer import Importer
import json
import matlab_wrapper

db_scheduler = DAL(db_scheduler_string)

def create_model(*args, **vars):
    w = matlab_wrapper.Whisperer(db,im)
    w.create_model(*args, **vars)

def import_or_update_movie(*args, **kwargs):
    i = Importer(db, im)
    i.import_or_update_movie(*args, **kwargs)

def import_popular_movies(*args, **kwargs):
    i = Importer(db, im)
    i.get_popular_movies(schedule_movie, *args, **kwargs)

def create_features_vector(*args, **kwargs):
    w = matlab_wrapper.Whisperer(db, im)
    w.create_features_vector(*args, **kwargs)

def create_titles_vector(*args, **kwargs):
    w = matlab_wrapper.Whisperer(db, im)
    w.create_titles_vector(*args, **kwargs)

def create_matlab_matrices(*args, **kwargs):
    w = matlab_wrapper.Whisperer(db, im)
    w.create_matlab_matrices(*args, **kwargs)

def update_all_movies(*args, **kwargs):
    movies = db(db.movies).select(db.movies.ALL, cacheable=True)
    for movie in movies:
        schedule_movie(movie.id, reviews=True)
```

Figure 4.12: Scheduler 1/2

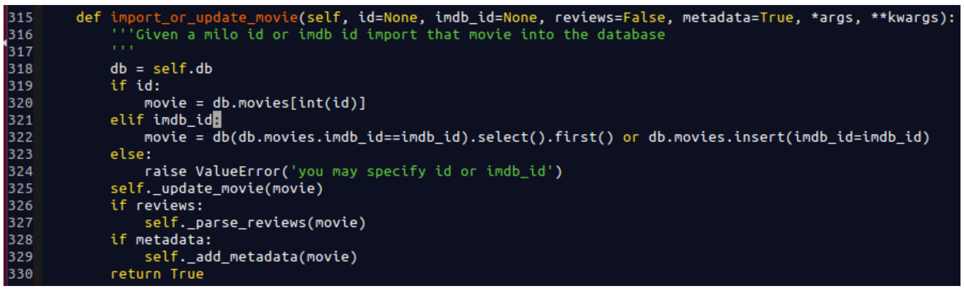


```
def start_survey(surveyid):
    def send_mail(users):
        for user in users:
            l= [user.email]
            mail.send(to=l,
                      subject='Survey invitation',
                      # If reply_to is omitted, then mail.settings.sender is used
                      reply_to='survey@movish.com',
                      message='''Hello.
                      You have been selected to participate to a quick survey about movies.
                      Just click on the link below and you can participate.

                      {0}

                      Your username is: {1}
                      Your password is: {2}

                      Ready to discover new cool movies?

                      '''.format(URL(
                          'milo', 'survey', 'demographic', args=[survey.id],
                          host='movish.co'), user.email, user.email[:user.email.find('@')]))
    survey = db.surveys[surveyid]
    #create_model(survey.algorithm)
    if survey.type == 'algorithm_performance':
        user_ids = db(db.surveys_users.survey == survey)._select(db.surveys_users.iuser)
        survey_users = db(db[auth.settings.table_user_name].id.belongs(user_ids)).select(
            db[auth.settings.table_user_name].id,
            db[auth.settings.table_user_name].email)
        send_mail(survey_users)

def do_recommendation(algorithm, userid, num_rec):
    w = matlab_wrapper.Whisperer(db, im)
    rec = w.get_rec(algorithm, userid, num_rec)
    db.recommendations.update_or_insert(db.recommendations.iuser==userid,
                                        iuser=userid,
                                        movies=[index for value,index in rec],
                                        timestamp=request.now,
                                        algorithm=algorithm
    )
    db.commit()

from gluon.scheduler import Scheduler
myscheduler = Scheduler(db_scheduler,
```

Figure 4.13: Scheduler 2/2

mation about a movie from imdb, surfs between all the reviews and collects them, also includes all the imdb users that gave a review to the movish system. It thus queries youtube in order to find a trailer for the movie and it if finds it, updates the movie info. Given all the operations that this function is capable of it resembles the core part of the auto updating feature of the system.

- **import_popular_movies**. This function surf the imdb popular movies page and add schedules all the movies in that page for an update thanks to *import_popular_movies*.
- **create_features_vector**. This function simply creates the feature vector. It's is scheduled from the admin mode under the action menu as described in section 4.2.2.
- **create_title_vectors**. Like the previous function, this creates the titles vector and it's scheduled from the admin mode.
- **create_matlab_matrices**. This task creates the URM and ICM to be used for the model creation.
- **update_all_movies**. Using *import_or_update_movie* this task cycles in all the movies in the system and schedule them for a update.
- **start_survey**. When a survey is added to the system this function is responsible of setting up the survey and send a mail to each user that has been indicated to perform the survey to let them know that the survey is ready.
- **do_recommendation**. Given an algorithm, a userid and a integer N, this task stores on database a recommendation for the given algorithm of N movies. This function is the core function for the recommendation engine. It has been performed as a task because it can take



```python
    def import_or_update_movie(self, id=None, imdb_id=None, reviews=False, metadata=True, *args, **kwargs):
        '''Given a milo id or imdb id import that movie into the database
        '''
        db = self.db
        if id:
            movie = db.movies[int(id)]
        elif imdb_id:
            movie = db(db.movies.imdb_id==imdb_id).select().first() or db.movies.insert(imdb_id=imdb_id)
        else:
            raise ValueError('you may specify id or imdb_id')
        self._update_movie(movie)
        if reviews:
            self._parse_reviews(movie)
        if metadata:
            self._add_metadata(movie)
        return True
```

Figure 4.14: Importer core function

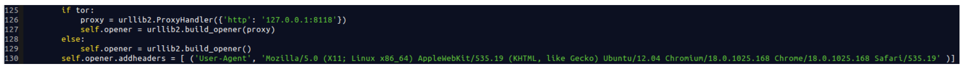
more than 40 seconds in the given hardware which is a Xenon quad core at 3Ghz with 16gb of ram. 40 seconds are not feasible for a web application that has to be as fast as it can. Thus this task has been added.

4.2.4 Importer

The importer is the part of the system that implements all the automatism and makes the system able to auto update with new movies. The importer is defined in file *modules/importer.py*. This file defines a **Importer** and **MediaRetriever** class. The MediaRetriever class is able to perform parsing from pages of imdb.com and youtube.com that are fetched by the crawler of section 4.2.5. The Importer class uses the MediaRetriever to get the trailer and the cover for a movie, it also uses the imdbpy [33] library to retrieve information about the movie and store into database. The Importer is capable to parse all the imdb.com reviews, to generate metadata for each movie and add them to database.

The core function of the importer is shown in Figure 4.14. It is basically the code called by the **import_or_update_movie** of the scheduler exposed in secion 4.2.3.

The importer connects to the crawler via a proxy service installed in the same server at port 8118. The crawler talks with the importer via a transparent Hypertext Transfer (or Transport) Protocol (http) proxy such that the



```
125         if tor:
126             proxy = urllib2.ProxyHandler({'http': '127.0.0.1:8118'})
127             self.opener = urllib2.build_opener(proxy)
128         else:
129             self.opener = urllib2.build_opener()
130         self.opener.addheaders = [ ('User-Agent', 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/535.19 (KHTML, like Gecko) Ubuntu/12.04 Chromium/18.0.1025.168 Chrome/18.0.1025.168 Safari/535.19' )]
```

Figure 4.15: Importer setting the crawler as a gateway code

crawler can be disabled at any time without affecting the overall system.

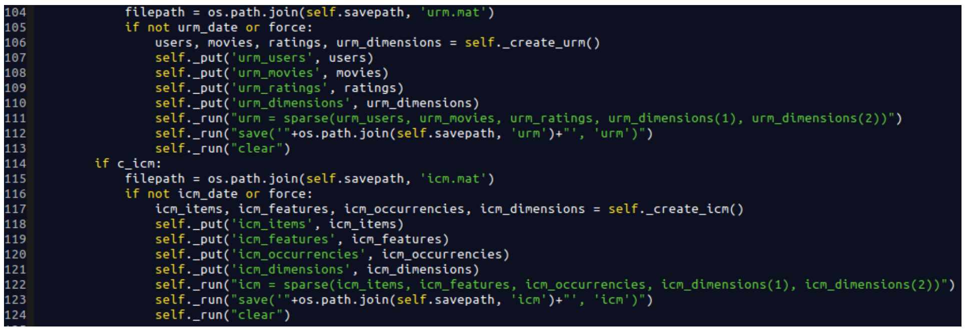
4.2.5 Crawler

As soon as the system started getting information from both youtube and imdb the incoming traffic to these application have been throttled. In order to be able to access those sources a Tor based proxy has been set up.

Tor [41] is a network of virtual tunnels that allows people and groups to improve their privacy and security on the Internet. It also enables software developers to create new communication tools with built-in privacy features. Tor provides the foundation for a range of applications that allow organizations and individuals to share information over public networks without compromising their privacy.

In movish, tor represent the crawler thanks to whom the importer can communicate to imdb.com and youtube.com in an anonymous way using different ip addresses every 10 minute in order to overcome to the bandwidth limitation that those two website impose to their users. Without this proxy the required time to create a useful dataset would require almost a year.

Figure 4.15 shows a small snippet of code from the **Importer** that sets up the **Crawler** as a proxy to reach both youtube and imdb. The code exposed is located at file named *modules/importer.py* at line 125. Thanks to the urllib2 python library flexibility it is possible to configure each request to use a proxy creating an *opener* object. Last line adds an user agent signature to each request to act like a real user opening the requested page.



```python
            filepath = os.path.join(self.savepath, 'urm.mat')
            if not urm_date or force:
                users, movies, ratings, urm_dimensions = self._create_urm()
                self._put('urm_users', users)
                self._put('urm_movies', movies)
                self._put('urm_ratings', ratings)
                self._put('urm_dimensions', urm_dimensions)
                self._run("urm = sparse(urm_users, urm_movies, urm_ratings, urm_dimensions(1), urm_dimensions(2))")
                self._run("save('"+os.path.join(self.savepath, 'urm')+"', 'urm')")
                self._run("clear")
        if c_icm:
            filepath = os.path.join(self.savepath, 'icm.mat')
            if not icm_date or force:
                icm_items, icm_features, icm_occurrencies, icm_dimensions = self._create_icm()
                self._put('icm_items', icm_items)
                self._put('icm_features', icm_features)
                self._put('icm_occurrencies', icm_occurrencies)
                self._put('icm_dimensions', icm_dimensions)
                self._run("icm = sparse(icm_items, icm_features, icm_occurrencies, icm_dimensions(1), icm_dimensions(2))")
                self._run("save('"+os.path.join(self.savepath, 'icm')+"', 'icm')")
                self._run("clear")
```

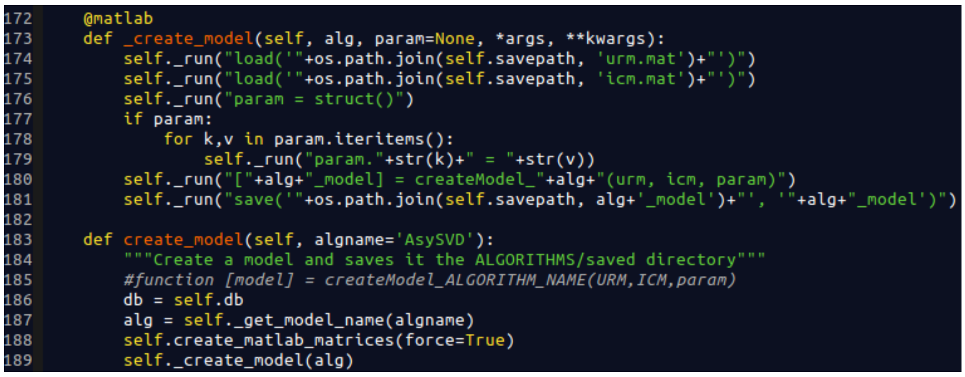
Figure 4.16: URM and ICM creation

4.2.6 Recommendation engine

The recommendation engine is the section of the application responsible of making and retrieve recommendations using a Matlab [4] engine. It was one of the constraints of the system since Milo because all the algorithms developed by the research group at Politecnico di Milano are implemented using that program. Matlab is heavily used in many courses at Politecnico di Milano and many other universities around the world, the Netflix competition [1] was also released using Matlab .mat files.

There is one single file in which matlab is imported and used in the entire system and it is in the *modules/matlab_wrapper.py*. The file contains mainly a object called **Whisperer**. This object exports all the function to create the matlab matrices ICM and URM, the titles and features vectors, the model for each supported algorithm and the recommendation for a given user. The object is also able to talk with the database in order to retrieve the correct data.

Figure 4.16 shows a snippet used for the URM and ICM generation. The variables are inserted in the matlab environment thanks to the *self._put()* function while the matlab commands are run thanks to the *self._run()* function. One of the big improvements of this piece of the application compared to Milo is the optimization performed at memory level by replacing matrices with sparse matrices. This change imposed a change in the whole way



```
    @matlab
    def _create_model(self, alg, param=None, *args, **kwargs):
        self._run("load('"+os.path.join(self.savepath, 'urm.mat')+"')")
        self._run("load('"+os.path.join(self.savepath, 'icm.mat')+"')")
        self._run("param = struct()")
        if param:
            for k,v in param.iteritems():
                self._run("param."+str(k)+" = "+str(v))
        self._run("["+alg+"_model] = createModel_"+alg+"(urm, icm, param)")
        self._run("save('"+os.path.join(self.savepath, alg+'_model')+"', '"+alg+"_model')")

    def create_model(self, algname='AsySVD'):
        """Create a model and saves it the ALGORITHMS/saved directory"""
        #function [model] = createModel_ALGORITHM_NAME(URM,ICM,param)
        db = self.db
        alg = self._get_model_name(algname)
        self.create_matlab_matrices(force=True)
        self._create_model(alg)
```

Figure 4.17: Create model code

the matrices are managed but it was able to drop memory requirements from 80Gb to 3Gb of Random Access Memory (RAM). After the matrices are computed they are stored in Matlab .mat files for easy retrieval from the admin interface exposed in section 4.2.2.

Each function that deals with matlab uses a **@matlab** python decorator. A decorator in python is a function that is applied to other functions. In this case the @matlab decorator ensures that the function that will be called using matlab will find a clear environment from other variables. This enhancement prevent the system from keep loading and unloading matlab environments for each function to call. The system keeps a single matlab session ready for each function, each function that is called cleans the environment after it exits thanks to the decorator instead.

Figure 4.17 shows the code to create a model. As you can see the action is performed by two functions. The first one called **create_model** ensures that the URM and the ICM are created. It thus calls the **_create_model** function that manages a matlab session, this is why it has the **@matlab** decorator, and performs the actual recommendation calling the right function name for the given algorithm in order to create the model.

The recommendation is done in a similar way of the model creation from the **get_rec** function. It accepts three parameters being the algorithm name and the user id. It then retrieve the user profile and performs the recommendation for the given algorithm if a model exists.

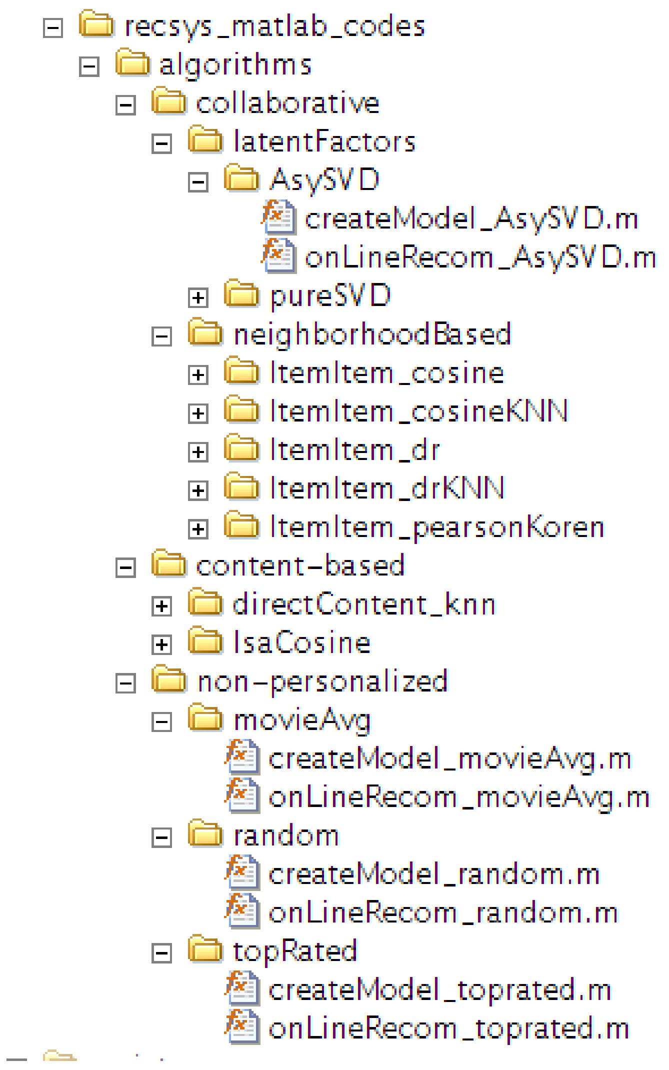
The recommendation engine has also an auto discovery functionality of new algorithms. When generating the lists of algorithms it simply looks in the *modules/algorithms/recsys_matlab_codes/algorithms/* directory and look for files which name that starts with **onLineRecom**. In fact all the algorithm have the form of *onLineRecom_algorithm.m*. For each algorithm found, the relative model creation function is looked up searching in the same directory of the algorithm for a *createModel_algorithm.m*. That file contains the function to generate the model for a given algorithm. Figure 4.18 shows the matlab algorithms tree structure. Each algorithm is categorized to be collaborative, content based or non personalized. For each class of algorithms, there are all the relevant sub classes.

All the computations that result in a creation of a file for matlab store the newly created file in the *modules/elaborated_models/*.

4.3 Structure

The project structure is based on web2py [3] one. The project has been published on GitHub [42]. GitHub defines itself like "the best place to share code with friends, co-workers, classmates, and complete strangers. Over two million people use GitHub to build amazing things together". GitHub is a webapp designed to share code using the git [43] revision control system. The application is composed mainly by the controllers, views, static, modules and models directory. All the other directories are the basic template of a web2y application.

- **controllers** directory contains all the controllers of the application. *admin.py* defines the controllers of the admin interface 4.2.2 with all the functions and the actions that the administrator can perform. *de-*



```
⊟ recsys_matlab_codes
    ⊟ algorithms
        ⊟ collaborative
            ⊟ latentFactors
                ⊟ AsySVD
                        createModel_AsySVD.m
                        onLineRecom_AsySVD.m
                ⊞ pureSVD
            ⊟ neighborhoodBased
                ⊞ ItemItem_cosine
                ⊞ ItemItem_cosineKNN
                ⊞ ItemItem_dr
                ⊞ ItemItem_drKNN
                ⊞ ItemItem_pearsonKoren
        ⊟ content-based
            ⊞ directContent_knn
            ⊞ lsaCosine
        ⊟ non-personalized
            ⊟ movieAvg
                    createModel_movieAvg.m
                    onLineRecom_movieAvg.m
            ⊟ random
                    createModel_random.m
                    onLineRecom_random.m
            ⊟ topRated
                    createModel_toprated.m
                    onLineRecom_toprated.m
```

Figure 4.18: Matlab algorithm tree structure

fault.py is the default controller which is called when the homepage is called. it has the *index* function that perform the basic visualization of the list of movies. *movie.py* is the controller for displaying the details of a movie. *rating.py* is the controller that takes care of performing ratings for a movie. This last controller is different from the other since it is designed to works as a module for other controllers. In fact the rating of a movie is always associate with a movie, so this controller is responsible of generating the correct stars indicating the rating and load the previous rating for a movie if the user already gave a rating for that movie. *survey.py* is the controller that handles all the survey creation.

- **models** directory contains all the data structures used in the application. The *db.py* defines all the database structure the reader can have a look to the ER diagram on section 3.4. *menu.py* defines the context menu that is displayed to the user. It is the menu on top on the black line which never changes and is the landmark for the user. *scheduler.py* this file defines the scheduler tasks. It uses the web2py automatic task discovering features to add the functions to the scheduler. We analyzed the code of the scheduler on secion 4.2.3. *whisperer.py* contains all the function that are called by the application to schedule tasks of the scheduler. Thanks to this file adding a task to the scheduler is easy like calling a function.

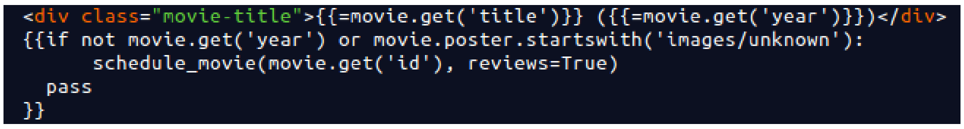
- *modules* directory contains all the matlab algorithm under the *algorithms* directory and all the models under *elaborated_models*. The *countries.py* file defines the list of all countries in the world used by the surveys while asking the demographic data. *importer.py* defines the Importer we talked in section 4.2.4. *matlab_wrapper.py* is the only file that interact with the matlab engine, thus is the only file that imports pymatlab [5]. *metadata.py* is the file used to set up the python Natural Language Toolkit (nltk) [44]. Nltk is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Nltk is used by Movish to perform plot tagging and extract the metadata for the movie. These metadata are useful for the ICM generation as explained in section 2.2.2.

- *static* directory contains all the static content of the application. Mainly all the css the the images.
- **views** directory contains all the html files that are used for outputting the data generated by the controllers to the user.

4.4 Dataset

The initial dataset used for milo was the one used for the netflix competition [1] but since it was released in 2001 it was soon figured out that it was pretty outdated and not useful for real recommendations with users. This is why in milo there were a basic importer to get the information from the imdb.com website. That importer was only able to retrieve basic information. In movish the importer is now able to parse imdb.com directly, to talk to imdb thanks to the imdbpy [33] library and to parse youtube pages directly. It is also able to parse all reviews in different pages of a movie and get the ratings.

Having this powerful importer, during three continuous months of scraping the imdb.com website, the majority of movies from the 1950 until now have been imported in the system. Resulting in 154606 distinct titles and 106266 distinct users as for Nov 23th 2012. In order to keep the datased



```
<div class="movie-title">{{=movie.get('title')}} ({{=movie.get('year')}})</div>
{{if not movie.get('year') or movie.poster.startswith('images/unknown'):
      schedule_movie(movie.get('id'), reviews=True)
  pass
}}
```

Figure 4.19: Automatically schedule a movie information update

up to date, the admin can create a task of updating all the movies via the admin interface explained in secion 4.2.2. The system is also able to automatically detect if a movie has not all the information and schedules a task for updating the movie information as explained in figure 4.19.

When the movie is displayed if the year field is missing or if the poster of the movie is not defined, the movies is then scheduled for an update. The system also provide a cronjob to automatically fetch new reviews and new movies that are going to be released in the next five years or that have been released in the past five years. This ensures that the system has up to date movies an reviews from imdb distributing the load of updating the dataset during time.

This is one of the biggest improvement from ContentWise [2] and enables new scenario and integration with other real time or up to date system such as Facebook and Twitter.

4.5 Conclusions

Movish is an extensible fully automatic recommendation survey system for movies. Thanks to its fundamental in bleeding edge technologies and features it is able of self updating the dataset, perform asynchronous recommendations, and having a good look and feel thanks to the simple interface. It's architecture is composed by many independent pieces that collaborates each other as explained in section 4.2 minimizing the point of failures that were one of the main problems in Milo.

There are some factor that may limit the evolution of Movish in the long term:

- **Matlab dependence**. Having a matlab engine to support existing algorithms was a goal of the project but matlab has been proved, during the movish implementation, that it is not suitable for real time recommendation systems since it requires too many resources and it is too slow compared to native python implementation. In fact if the recommendation algorithms were implemented in python the system would require much less resources and be much faster than it is now. So in the future developing an alternate recommendation engine, maybe in python, would be very helpful and would not impact with the overall system.

- **Improved security**. As for now all the communication with the server is performed via insecure http connections. Switching to Hypertext Transfer Protocol Secure (https) would protect user data transferred over the internet and block password hijacking attempts.

- **Improve sources**. As for now only imdb.com and youtube.com are used as sources. Even if they are the most comprehensive sources available there are many more sources with ratings online that could be imported in the system to perform better recommendations.

In this chapter we didn’t talk about survey management. Since it is another big feature of the web application, chapter 5 will explain how Movish handles surveys, how to create one and how to add a new type of survey.

Chapter 5

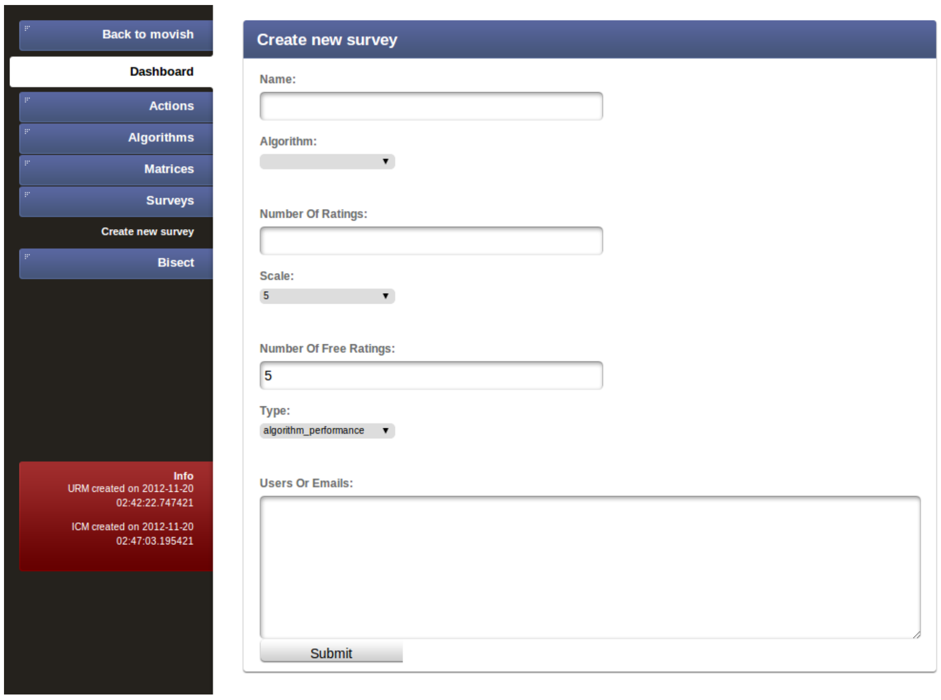
Survey management

5.1 Introduction

In this chapter we will analyze, in depth, how to perform, create and add a new type of survey in the system. Movish is indeed a complex system that can be used as a plain movie catalog or as a survey system. When in survey mode, the system changes smoothly the layout in order to guide the user through the questions and removes many not useful links that would bring the user out of the application or in some undefined states that are only valid when browsing the catalog without performing a survey.

Guiding a user while performing a survey is a challenge. The survey must be designed such that the user doesn't waste time in not useful operation but it is still free to browse, search and rate any content the user wants in order of not alter the recommendation. The user must be guided in order to focus his/her attention in what is the survey goal also. This makes planning a new type of survey a challenge and a task that requires a lot of thinking before being able to implement one.

Movish simplifies the process of adding a new type of survey, that i exposed in secion 5.3 while survey creation is exposed in section 5.2 with the two types of survey supported, the algorithm performance and the algo-



Info
URM created on 2012-11-20 02:42:22.747421
ICM created on 2012-11-20 02:47:03.195421

Create new survey

Name:

Algorithm:

Number Of Ratings:

Scale: 5

Number Of Free Ratings: 5

Type: algorithm_performance

Users Or Emails:

Figure 5.1: Survey creation

rithm strength.

5.2 Survey creation

The creation of a survey is the action of adding a survey to the system and perform all the necessary operations that make the system able to accept new user or existing ones to perform the survey. A user with administration privileges is needed to make a new survey. After logged in the user has to go to the admin interface explained in secion 4.2.2.

Figure 5.1 shows the form to create a new survey. The administrator has to pick up a name, an algorithm name (from the list), a number of ratings representing the number of ratings that will be displayed to the user while being in the recommendation part of a survey; the scale of the rating, the number of free ratings the user is able to perform before receiving the recommendation; the survey type that can be **algorithm_performance** or **algorithm_strength**. We will talk about these two types in the next subsections. The administrator can optionally add a set of comma separated emails of the user that will be notified with a mail that there is a survey for them. Users that are not in the system will be added and a mail will be sent as well. After submitting the form the task of creating a new survey is created.

In general, any user can perform the surveys that are in the system if the user knows the id of the survey. The entry point for each user is in fact *survey/demographic/ID* where ID is the id of the survey. The user must be logged in to access that resource. In the case the user is not registered, it has to register first. To remove this additional step which is mandatory for the recommendation engine to work but may be a problem for a user that is just on the site to perform a survey a special handler that creates a user to the system and start the survey has been created. It can be accessed from *http://movish.co/survey/amazonturk/ID* where ID is the id of the survey. The administrator can see the id of the survey looking at the last number of the *amazon turk link* column in the survey menu in figure 4.10.

Users have not be bounded to a particular survey in order to allow the administrator to select other users while the survey in progress and to allow the Amazon Mechanical Turk [45] cooperation. We will talk more about amazon mechanical turk on secion 5.4. As for now the reader can think of amazon mechanical turk as a service offered by amazon to provide human workforce for tasks that are able to be performed with a computer. Online surveys are a typical case of this kind of work.

Result of the surveys can be retrieved in the survey admin page shown in figure 4.10. Clicking on the small icon with a pencil with a label of *download result* the administrator can get the results of the survey in Comma separated values (csv) format.

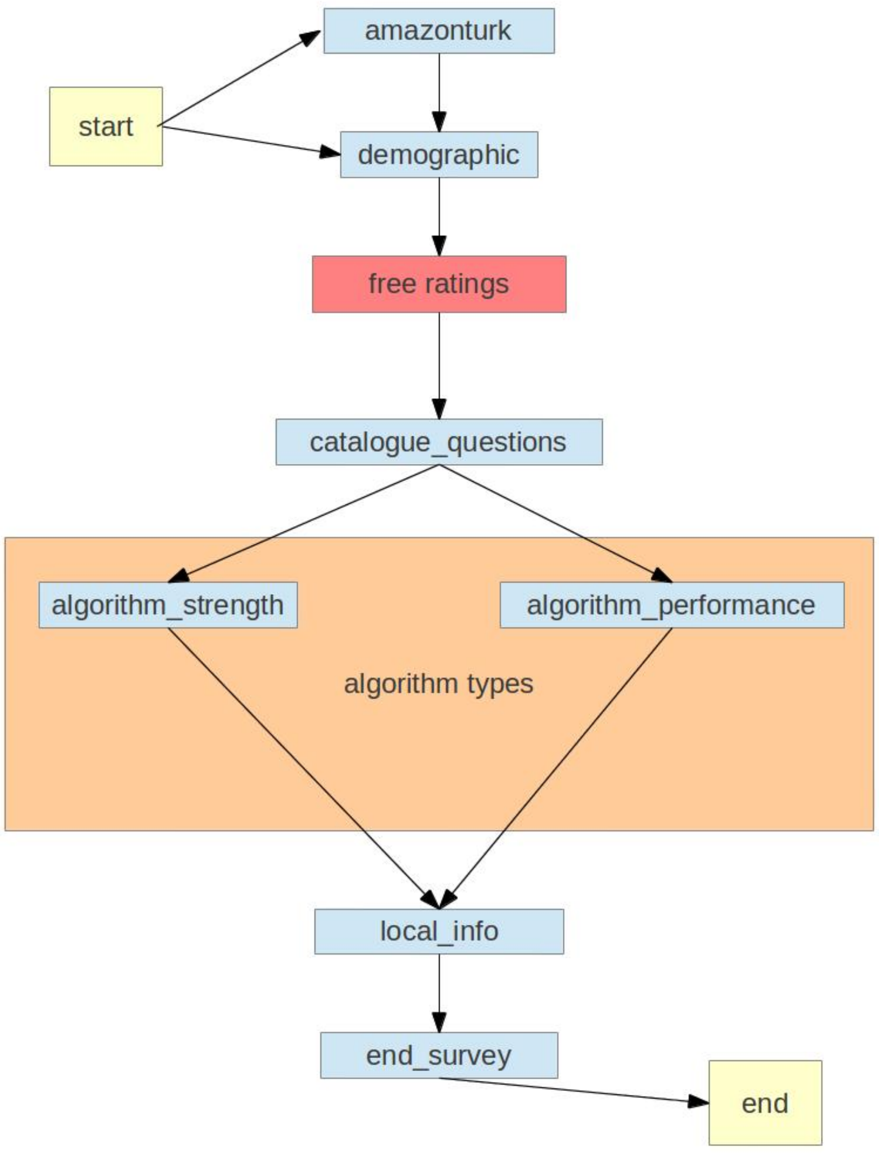


Figure 5.2: Survey architecture: relation between functions

Figure 5.2 shows the survey subsystem architecture. The user starts from the **start** box and ends in the **end** box. The start can happen from the *amazonturk* link ad described earlier or by direct invite at survey creation, in that case the user goes directly to **demographic**. After filling the form the user is redirected in a modified homepage without the possibility to logout and without a the main slider with only the possibility to surf the catalog and search for movies. This step is called **free ratings** and the user has to rate the number of ratings that the admin specified as *number of free ratings* in survey creation. After the user has to answer a series of question regarding the catalogue. This phase is performed by the function **catalogue_questions**. From now the behaviour of the system depends from the type of survey. As for now two types of surveys are implemented and they are performed by **algorithm_strength** and **algorithm_performance** respectively. **catalogue_questions** is able to fetch the survey type and redirect the user to the correct page generated by the correct survey type. All survey types, after performing the recommendation must redirect to the **local_info** function that take cares of asking the last questions to the user and redirect him/her to the **end_survey** page. The end survey page exposes the Amazon Mechanical Turk confirmation code that the user has to submit to the amazon form in order to claim the survey that is just being completed.

5.2.1 Algorithm performance

Algorithm performance is the first type of survey that has been implemented. It shows the recommendation to the user displaying one movie at time and asking him/her subsequent questions. The questions that the user answers are contextual, it means that they vary depending from the previous questions. The main goal is to give to the user all the information in order to make him/her able to rate the movie even if he/she did not actually watched the movie. This was the first type of survey and the only one

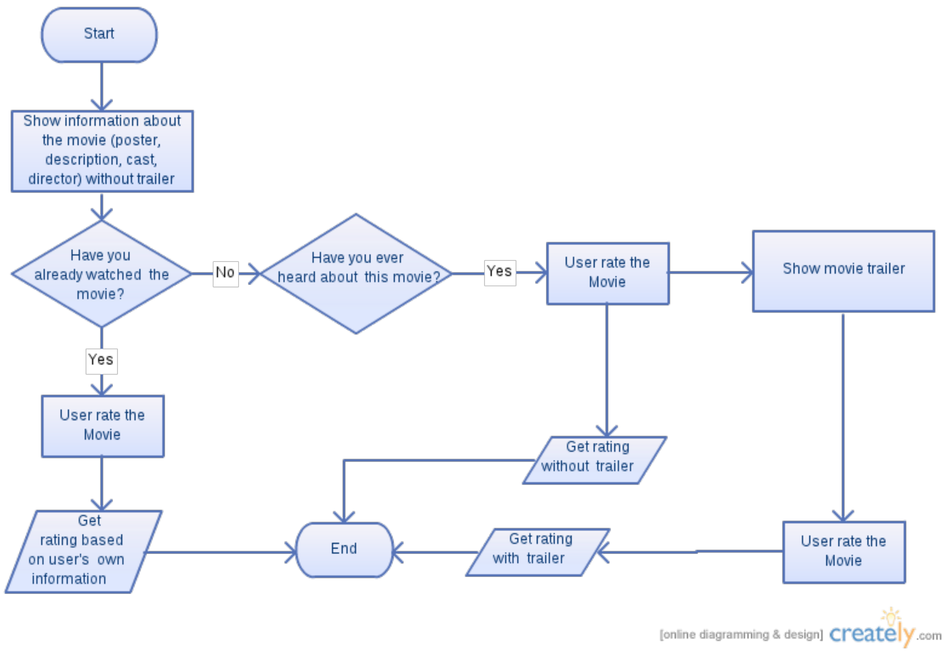

Start
Show information about the movie (poster, description, cast, director) without trailer
Have you already watched the movie?
No
Have you ever heard about this movie?
Yes
User rate the Movie
Show movie trailer
Yes
User rate the Movie
Get rating without trailer
Get rating based on user's own information
End
Get rating with trailer
User rate the Movie
[online diagramming & design] creately.com

Figure 5.3: Algorithm performance flow chart

implemented in Milo [18]. The flow chart in figure 5.3 shows the questions that the user has to answer and their order.

The user starts from the **start** box and ends at the **end** box. As the reader can verify the user may not answer all the questions or get all the information before getting to the **end** box.

5.2.2 Algorithm strength

Algorithm strength is the second type of survey that Movish supports. It has been deployed to perform the research of chapter 6. In this type of survey the user gets a recommendation and the user can see the first Nth elements where N is the *number of ratings* that the administrator set at survey creation. The system then asks some question about the recommendation to the user, the questions are:

- How many movies in this list have you ever watched?

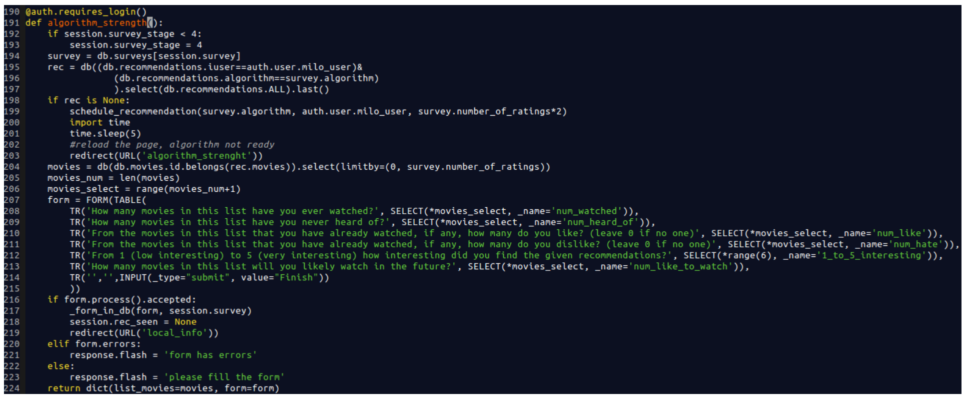
- How many movies in this list have you never heard of?
- From the movies in this list that you have already watched, if any, how many do you like?
- From the movies in this list that you have already watched, if any, how many do you dislike? (leave 0 if no one)
- From 1 (low interesting) to 5 (very interesting) how interesting did you find the given recommendations?
- How many movies in this list will you likely watch in the future?

This kind of survey is used to test how the perception of a good recommendation relates with the number of item shown to the user in order to understand how strong is an algorithm and in general how the number of items displayed to the user influences his/her perception of a good recommendation.

5.3 Adding new survey type to Movish

Adding a new survey to the system is a easy task which needs a bit of understanding in how the movish system works. Movish support a set of predefined questions that are the same for each survey and a variable set of questions and recommendation pages that can be composed and chained easily by the administrator. All the survey subsystem is defined in *controllers/survey.py*. Each survey page is generated by three steps:

- **form definition**. Where the form that the user has to answer is defined using web2py [3] helpers. Web2py helpers are function that produce a html code snippet accordingly. They are used to generate html on the controller or in the view itself.



```
190 @auth.requires_login()
191 def algorithm_strength():
192     if session.survey_stage < 4:
193         session.survey_stage = 4
194     survey = db.surveys[session.survey]
195     rec = db((db.recommendations.iuser==auth.user.milo_user)&
196              (db.recommendations.algorithm==survey.algorithm)
197              ).select(db.recommendations.ALL).last()
198     if rec is None:
199         schedule_recommendation(survey.algorithm, auth.user.milo_user, survey.number_of_ratings*2)
200         import time
201         time.sleep(5)
202         #reload the page, algorithm not ready
203         redirect(URL('algorithm_strenght'))
204     movies = db(db.movies.id.belongs(rec.movies)).select(limitby=(0, survey.number_of_ratings))
205     movies_num = len(movies)
206     movies_select = range(movies_num+1)
207     form = FORM(TABLE(
208         TR('How many movies in this list have you ever watched?', SELECT(*movies_select, _name='num_watched')),
209         TR('How many movies in this list have you never heard of?', SELECT(*movies_select, _name='num_heard_of')),
210         TR('From the movies in this list that you have already watched, if any, how many do you like? (leave 0 if no one)', SELECT(*movies_select, _name='num_like')),
211         TR('From the movies in this list that you have already watched, if any, how many do you dislike? (leave 0 if no one)', SELECT(*movies_select, _name='num_hate')),
212         TR('From 1 (low interesting) to 5 (very interesting) how interesting did you find the given recommendations?', SELECT(*range(6), _name='1_to_5_interesting')),
213         TR('How many movies in this list will you likely watch in the future?', SELECT(*movies_select, _name='num_like_to_watch')),
214         TR('','',INPUT(_type="submit", value="Finish"))
215         ))
216     if form.process().accepted:
217         _form_in_db(form, session.survey)
218         session.rec_seen = None
219         redirect(URL('local_info'))
220     elif form.errors:
221         response.flash = 'form has errors'
222     else:
223         response.flash = 'please fill the form'
224     return dict(list_movies=movies, form=form)
```

Figure 5.4: Adding new algorithm: algorithm_strength example

- **form submission**. Where the form that the user answered get submitted and saved in a database.

- **data retrieval**. Where the data that is shown to the user during a particular survey page is retrieved.

Figure 5.4 shows how the *algorithm_strength* survey type is implemented. It shows clearly where the different parts of the survey page are composed. The first 13 lines, from 192 to 206 are for data retrieval. In those lines the recommendation for the given algorithm of the survey is retrieved. If there is no recommendation available for the current user in the given algorithm a new recommendation is scheduled and the system waits 5 seconds before reloading the same page and check that there is a recommendation. Lines from 207 to 215 are for form definition. The form is mainly formatted with a table in which there are select fields. The *_name* property is very important because it defines the name that will be displayed when the administrator retrieves the survey results.

Last lines, from 216 to 224, are for form submission. If the form is recognized to have valid data (accepted) the form data is added to the database using the special **_form_in_db** function that having a form and a survey id adds the answers that the user provided to the database. Using this function every form created with web2py helpers can be recognized, parsed and stored into database. After saving the data to the database the user is redirected to the last question with the *redirect(URL('local_info'))* line. Each algorithm type, after processing the data must call this redirect in order to let the user finish the survey.

5.4 Crowdsourcing

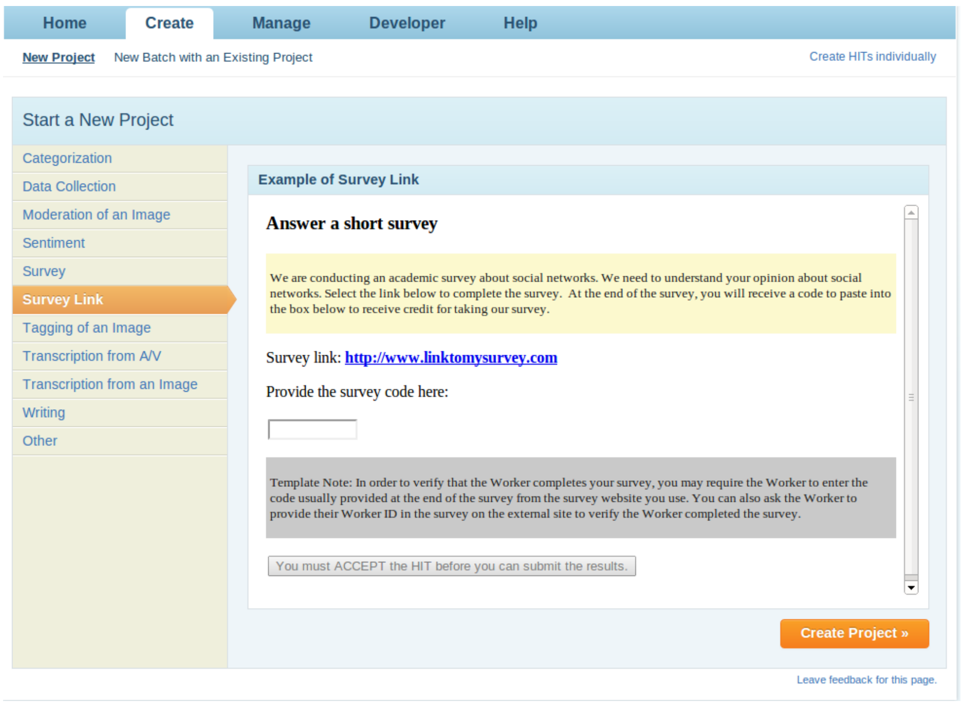
Crowdsourcing [46] is a process that involves outsourcing tasks to a distributed group of people. This process can occur both online and offline. The difference between crowdsourcing and ordinary outsourcing is that a task or problem is outsourced to an undefined public rather than a specific body, such as paid employees.

Amazon Mechanical Turk [45] is a marketplace for work. It gives businesses and developers access to an on-demand, scalable workforce. Workers select from thousands of tasks or Human Intelligence Tasks (HIT) and work whenever it is convenient. The service has two kind of users: workers and requester.

- **Requester**. Asks workers to completer HITs and gets results. Requesters have access to a global 24x7 workforce, get thousands of HITs completed in minutes, and pay only when you are satisfied with the results.

- **Worker**. Are the people that perform the HITs. Workers can work from everywhere with an internet connection, choose their work hours and get paid for doing good work.

Movish has been designed to work with Amazon Mechanical Turk. After the survey has been created, in the survey list page of figure 4.10, the administrator has an *Amazon Turk link* which is the link to provide to the



Answer a short survey

We are conducting an academic survey about social networks. We need to understand your opinion about social networks. Select the link below to complete the survey. At the end of the survey, you will receive a code to paste into the box below to receive credit for taking our survey.

Survey link: http://www.linktomysurvey.com

Provide the survey code here:

Template Note: In order to verify that the Worker completes your survey, you may require the Worker to enter the code usually provided at the end of the survey from the survey website you use. You can also ask the Worker to provide their Worker ID in the survey on the external site to verify the Worker completed the survey.

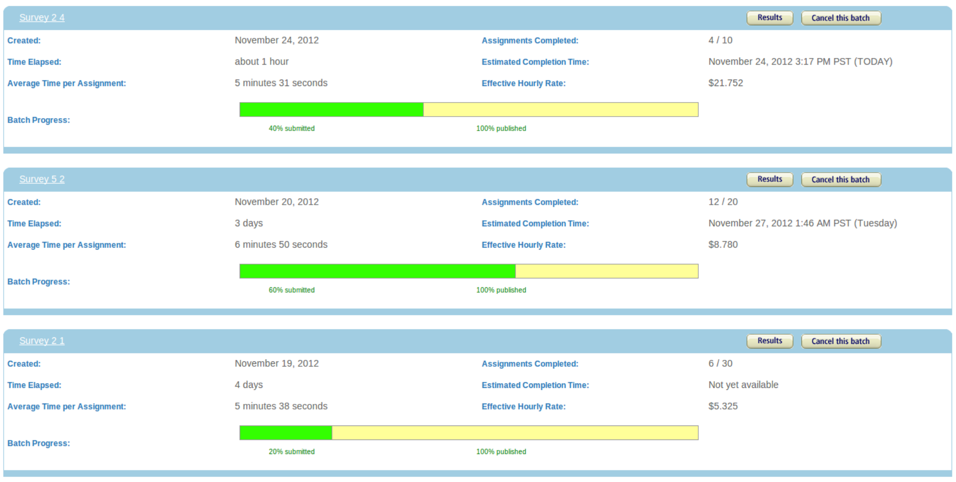
You must ACCEPT the HIT before you can submit the results.

Figure 5.5: Amazon Mechanical Turk survey link type

Amazon Mechanical Turk interface. The suggested way of integrating Movish with the amazon service is by survey link which is described in their webpage as figure 5.5 shows. The administrator has to create a new survey link and use the link provided from the movish admin interface.

After the HIT has been started from Amazon Mechanical Turk, the administrator can monitor the status of all the HITs from the management console shown in figure 5.6. In that picture the status of the three surveys that I performed during this thesis are shown. We will talk more about them on chapter 6.

Thanks to the integration with Amazon Mechanical Turk, Movish is able to be a platform to perform surveys without the admin having to worry in promoting the survey which is usually a time consumptive tasks. Based on the amount of work for a survey and the time spent on it (5 minutes and 31 seconds average) the correct pay for a Movish survey HIT is



Survey 2.4 Results Cancel this batch

Created:	November 24, 2012	Assignments Completed:	4 / 10
Time Elapsed:	about 1 hour	Estimated Completion Time:	November 24, 2012 3:17 PM PST (TODAY)
Average Time per Assignment:	5 minutes 31 seconds	Effective Hourly Rate:	$21.752
Batch Progress:	40% submitted 100% published		

Survey 5.2 Results Cancel this batch

Created:	November 20, 2012	Assignments Completed:	12 / 20
Time Elapsed:	3 days	Estimated Completion Time:	November 27, 2012 1:46 AM PST (Tuesday)
Average Time per Assignment:	6 minutes 50 seconds	Effective Hourly Rate:	$8.780
Batch Progress:	60% submitted 100% published		

Survey 2.1 Results Cancel this batch

Created:	November 19, 2012	Assignments Completed:	6 / 30
Time Elapsed:	4 days	Estimated Completion Time:	Not yet available
Average Time per Assignment:	5 minutes 38 seconds	Effective Hourly Rate:	$5.325
Batch Progress:	20% submitted 100% published		

Figure 5.6: Surveys on Amazon Mechanical Turk

about $1.50. Having set $2 as pay for each survey made the system get 10 surveys in less than 3 hours. This has an effective hourly rate for the worker of $21.687. Having set $1 as pay for each survey made the system collect 10 surveys after three days with an average of 6 minutes and 56 seconds per survey and an effective hourly rate of $8.654.

5.5 Conclusions

Movish implements a flexible and easy to update survey system. It has being designed in order to support continue modification of the survey types and creation of new surveys. The survey creation has been thought to be performed by an administrator in order to do not modify any line of code. New survey types are easily added with a slightly edit of source code. The integration with Amazon Mechanical Turk makes Movish able to also promote itself to reach users thus have completed survey. Using Movish an recommendation system professional can deliver survey without worrying about seeking for users and visibility.

These features make Movish the best system to deploy survey for recommendation system available today.

Chapter 6

Research

6.1 Introduction

The visualization has been proven to be an essential part in the recommendation since it can determine if it is useful or not from the user point of view [22]. There have been studies to determine how an item should be presented to the user in order to gain his/her attention as described in section 2.4 but there are not been much documented studies about the number of items that a recommendation system should display in order to maximize the user perception of a good recommendation.

Thus the proposal of this research is to evaluate how the number of movie visualized alter the user perception of a good or bad recommendation. To be scientific valid only one variable of the system should vary between all the test groups. In this case the number of items displayed has been chosen to vary between 2, 5 and 10 with at least 25 samples (or completed surveys) each. This is also the first research done using the Movish automatic recommendation survey system.

As for algorithm pureSVD has been chosen because it has been proved to have the best performances between all the algorithms supported by the Movish system [47]. PureSVD was the top performer in both Movielens

[48] and Netflix [1] datasets beating more detailed and sophisticated latent factor models even if it is one of the simplest algorithms available.

In section 6.2 the reader will be guided in analyzing the various sources used, in section 6.3 a complete survey for this research is exposed. Then in section 6.4 the analysis of the data is published and section 6.5 exposes the conclusions.

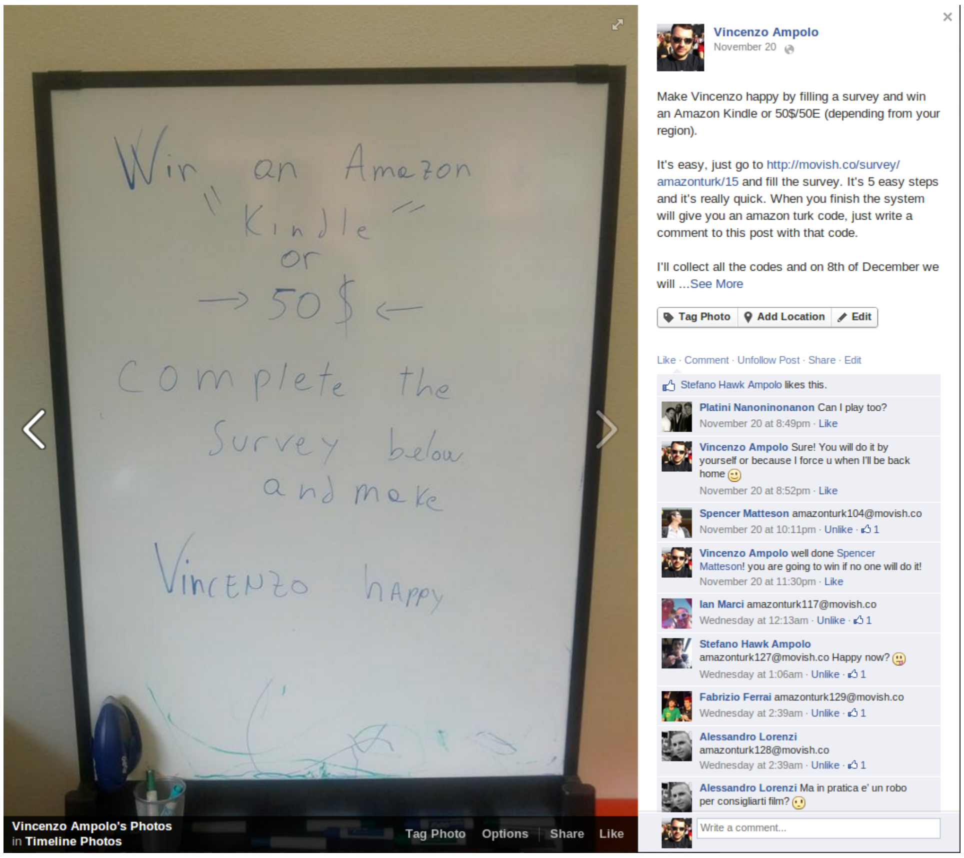
6.2 Sources

For this research a number of different resources has been used in order to collect the maximum number of surveys with the minimum budget. Aside of Amazon Mechanical Turk of which we discussed on section 5.4, also Facebook and Google plus have been used in order to test if a survey can go viral on social networks.

In order to maximize the user attention a whiteboard rewarding an eligibility of prize to the ones that perform the survey has been used. The Facebook post that promoted one of the three surveys that have been performed is displayed in figure 6.1.

The whiteboard message is: *Win an Amazon Kindle or $50, complete the survey below and make Vincenzo happy*. The message had two goals: stimulate all my close friends in taking action in order to help me in my research and to stimulate all the not close friends in performing the survey due to the prize they could win. In order to be eligible for the prize the user/friend had to write the Amazon Mechanical Turk code that is displayed at the end of the survey as comment to the post. The same message has been posted on Google plus and can be seen in figure 6.2.

The facebook post was able to retrieve a total 13 users performing the survey of which 7 of them can be categorized as not close friends. The Google plus post wasn't able to stimulate any user.


Win an Amazon "Kindle" or → 50$ ←
Complete the Survey below and make
Vincenzo happy
Vincenzo Ampolo's Photos in Timeline Photos
Tag Photo
Options
Share
Like
Tag Photo
Add Location
Edit
Like · Comment · Unfollow Post · Share · Edit
Write a comment...

Vincenzo Ampolo
November 20

Make Vincenzo happy by filling a survey and win an Amazon Kindle or 50$/50E (depending from your region).

It's easy, just go to http://movish.co/survey/amazonturk/15 and fill the survey. It's 5 easy steps and it's really quick. When you finish the system will give you an amazon turk code, just write a comment to this post with that code.

I'll collect all the codes and on 8th of December we will ...See More

Stefano Hawk Ampolo likes this.

Platini Nanoninonanon Can I play too?
November 20 at 8:49pm · Like

Vincenzo Ampolo Sure! You will do it by yourself or because I force u when I'll be back home 🙂
November 20 at 8:52pm · Like

Spencer Matteson amazonturk104@movish.co
November 20 at 10:11pm · Unlike · 👍 1

Vincenzo Ampolo well done Spencer Matteson! you are going to win if no one will do it!
November 20 at 11:30pm · Like

Ian Marci amazonturk117@movish.co
Wednesday at 12:13am · Unlike · 👍 1

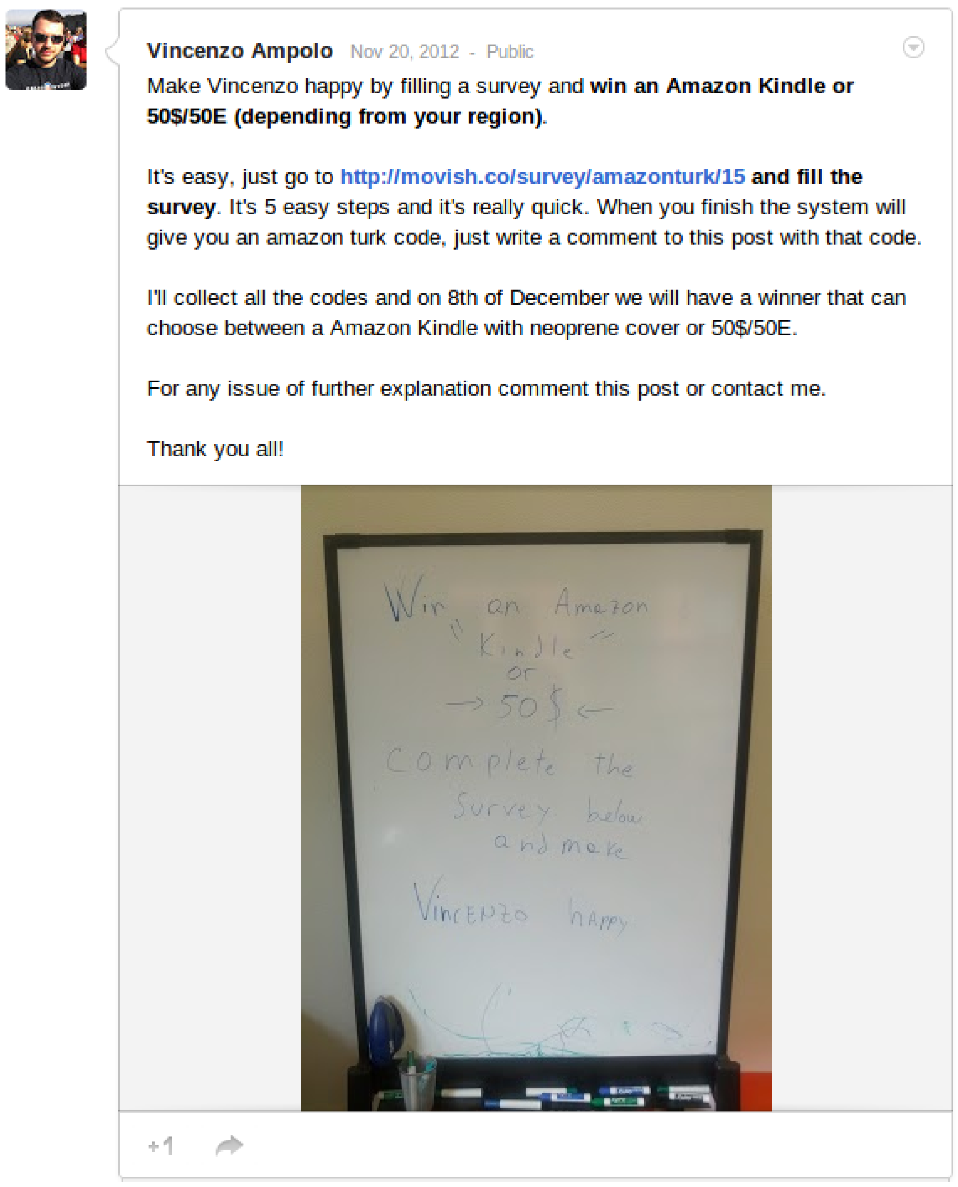
Stefano Hawk Ampolo amazonturk127@movish.co Happy now? 😛
Wednesday at 1:06am · Unlike · 👍 1

Fabrizio Ferrai amazonturk129@movish.co
Wednesday at 2:39am · Unlike · 👍 1

Alessandro Lorenzi amazonturk128@movish.co
Wednesday at 2:39am · Unlike · 👍 1

Alessandro Lorenzi Ma in pratica e' un robo per consigliarti film? 😕

Figure 6.1: Facebook post to participate to a survey



Vincenzo Ampolo Nov 20, 2012 - Public

Make Vincenzo happy by filling a survey and **win an Amazon Kindle or 50$/50E (depending from your region)**.

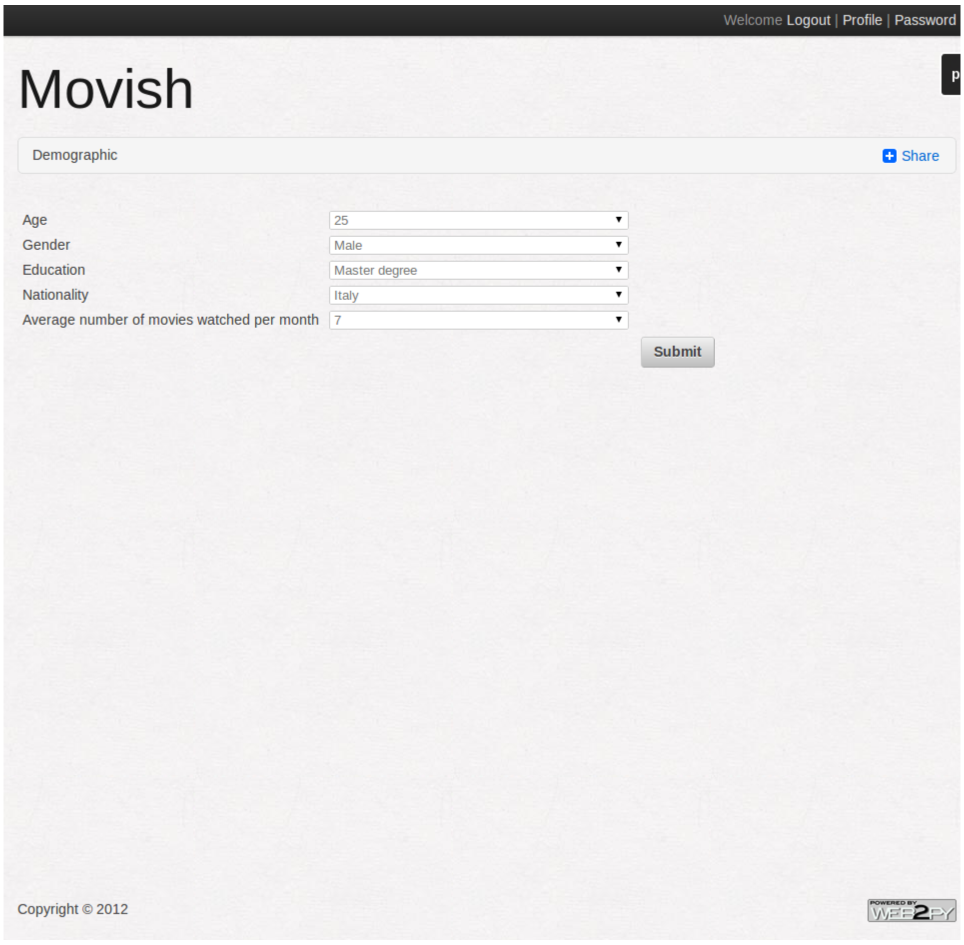
It's easy, just go to http://movish.co/survey/amazonturk/15 **and fill the survey**. It's 5 easy steps and it's really quick. When you finish the system will give you an amazon turk code, just write a comment to this post with that code.

I'll collect all the codes and on 8th of December we will have a winner that can choose between a Amazon Kindle with neoprene cover or 50$/50E.

For any issue of further explanation comment this post or contact me.

Thank you all!

Figure 6.2: Google plus post to participate to a survey



Demographic

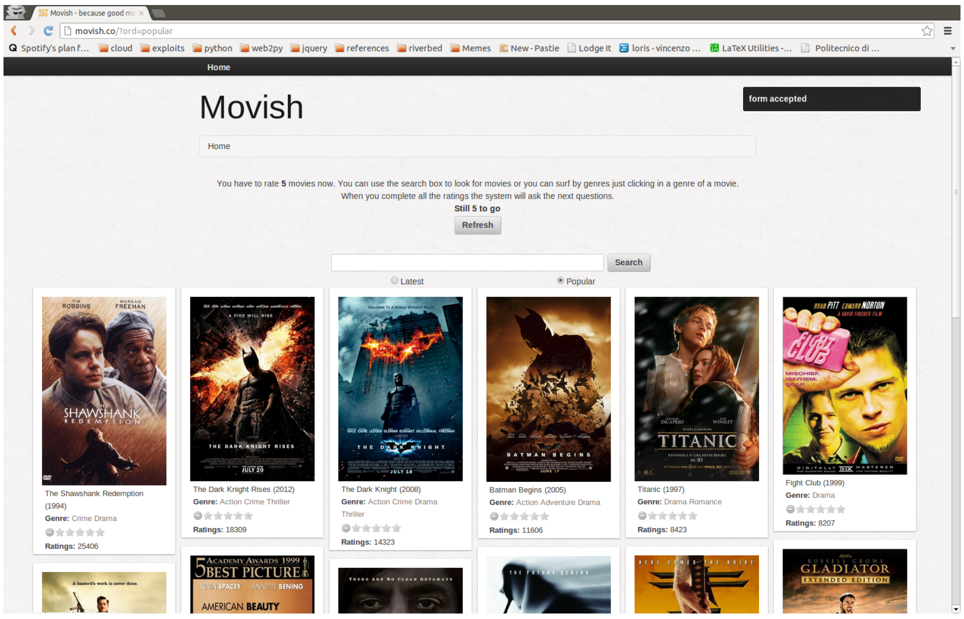
Age	25
Gender	Male
Education	Master degree
Nationality	Italy
Average number of movies watched per month	7

Figure 6.3: Survey phase 1

6.3 Survey

In order to fulfill the goal of this research a survey type of **algorithm_strength** with 5 free ratings using **pureSVD** algorithm has been chosen. The reason of this choice are explained in section 6.1. The survey starts asking general demographic information as shown in figure 6.3.

The user is asked to put the age, the gender, the education type, the nationality and the average numbers of movies watched per month. The goal of those questions is to associate the user with a cluster of users since the preference for a special category of movies change in respect of the age



Movish

form accepted

Home

You have to rate **5** movies now. You can use the search box to look for movies or you can surf by genres just clicking in a genre of a movie.
When you complete all the ratings the system will ask the next questions.

Still 5 to go

Latest [✓ Popular]

The Shawshank Redemption (1994)
Genre: Crime Drama
Ratings: 25406

The Dark Knight Rises (2012)
Genre: Action Crime Thriller
Ratings: 18309

The Dark Knight (2008)
Genre: Action Crime Drama Thriller
Ratings: 14323

Batman Begins (2005)
Genre: Action Adventure Drama
Ratings: 11606

Titanic (1997)
Genre: Drama Romance
Ratings: 8423

Fight Club (1999)
Genre: Drama
Ratings: 8207

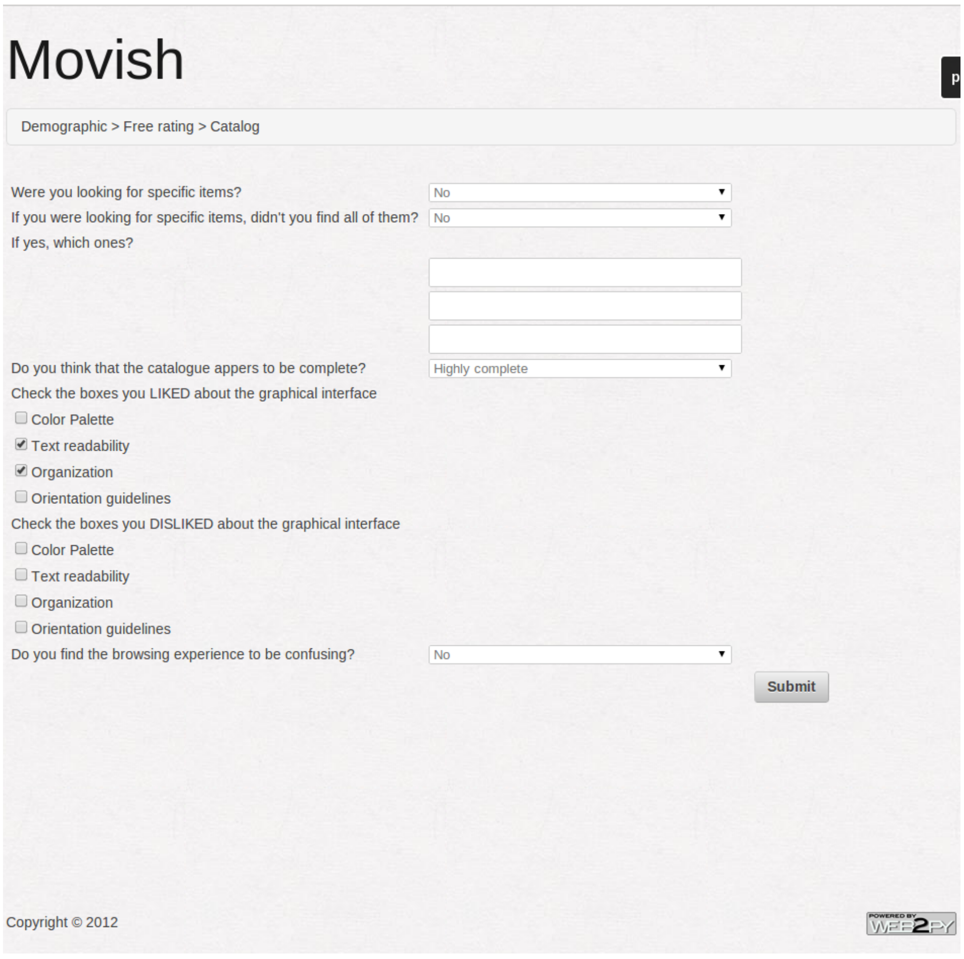
Figure 6.4: Survey phase 2

and the nationality. The average number of movies watched per month is an indicator to test if the user likes, in general, watching movies.

After the demographic questions the user is able to perform the free ratings from the page shown in figure 6.4. From this page the logout link in the top right part of the page disappear: the user which is performing the survey is not supposed to logout anymore. Beside this also all the not useful links that may lead the user out of the Movish system are removed in order to force the user in completing the survey before leaving the system.

To facilitate the user in starting rating movies, the top 15 movies with ratings in the system are chosen and displayed: they represent the top popular movies in the system. The user can rate directly in the homepage thanks to the star system under each movie poster or by going to the movie detail page and rate again from there using the same star system. Optionally the user can also see the trailer and read the plot of a movie in the movie detail page shown in figure 4.4.

User is also allowed to use the search box to look for any movie he/she


Movish

Demographic > Free rating > Catalog

Were you looking for specific items?	No
If you were looking for specific items, didn't you find all of them?	No
If yes, which ones?	
Do you think that the catalogue appers to be complete?	Highly complete

Check the boxes you LIKED about the graphical interface

- ☐ Color Palette
- ☑ Text readability
- ☑ Organization
- ☐ Orientation guidelines

Check the boxes you DISLIKED about the graphical interface

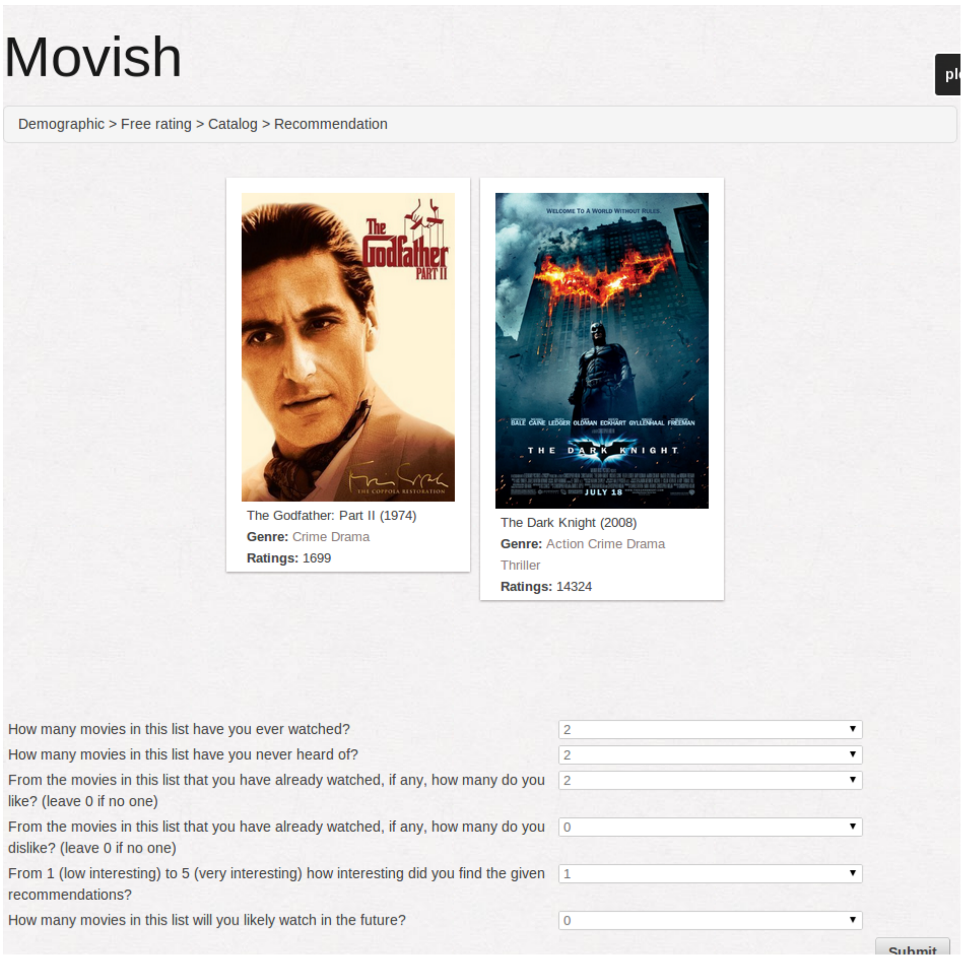
- ☐ Color Palette
- ☐ Text readability
- ☐ Organization
- ☐ Orientation guidelines

Do you find the browsing experience to be confusing?	No

Figure 6.5: Survey phase 3

wants. In fact this way of looking to the items is suggested by the top message of the page. It is so to test the completeness of the catalog. After the user rated the number of free ratings chosen by the administrator at survey creation discussed in section 5.2, the system automatically redirect the user to the next page of the survey, the catalogue questions.

These questions are shown in figure 6.5. The user is asked to answer some questions related to the catalogue he/she just browsed doing the free ratings. He/she has to answer questions about the browsing experience: looking for specific movies or not, if some movies were not found, to


Movish

Demographic > Free rating > Catalog > Recommendation

The Godfather: Part II (1974)
Genre: Crime Drama
Ratings: 1699

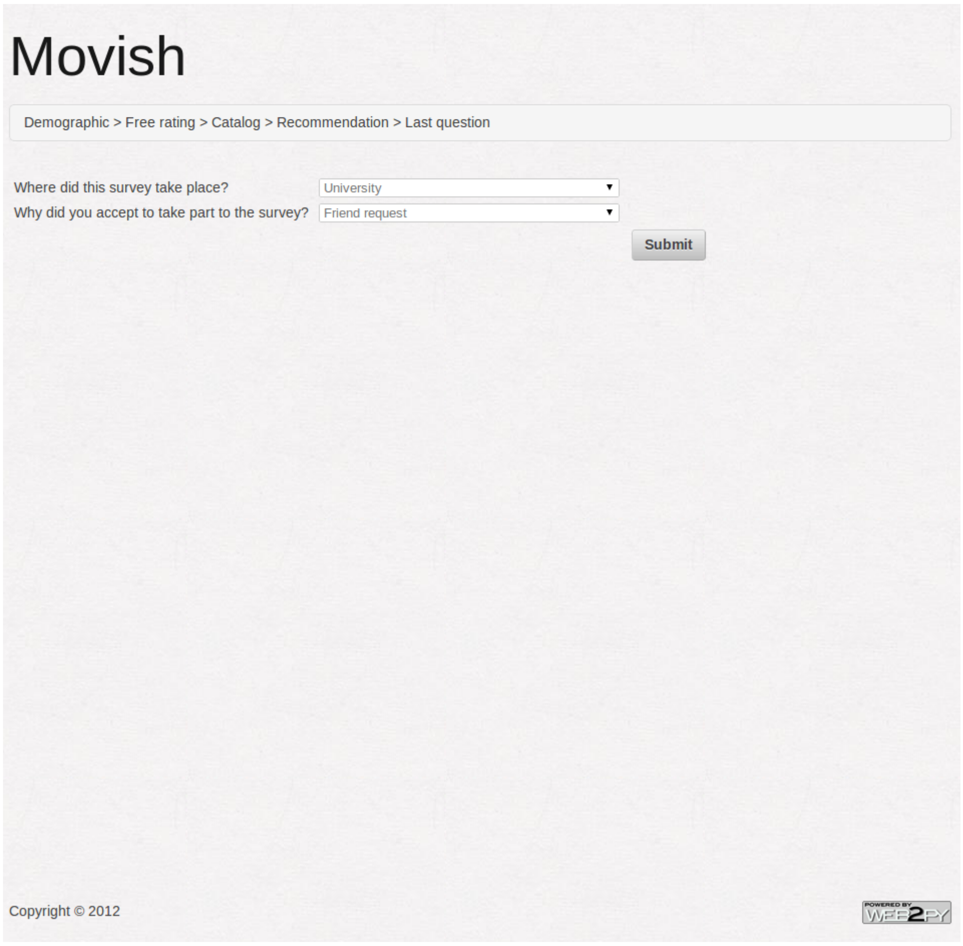
The Dark Knight (2008)
Genre: Action Crime Drama Thriller
Ratings: 14324

How many movies in this list have you ever watched?	2
How many movies in this list have you never heard of?	2
From the movies in this list that you have already watched, if any, how many do you like? (leave 0 if no one)	2
From the movies in this list that you have already watched, if any, how many do you dislike? (leave 0 if no one)	0
From 1 (low interesting) to 5 (very interesting) how interesting did you find the given recommendations?	1
How many movies in this list will you likely watch in the future?	0

Figure 6.6: Survey phase 4

list their names, the completeness of the catalogue and what he/she liked about the website about color palette, text readability, organization and orientation guidelines. After submitting the form the user is redirected to the next phase, the recommendation.

It is shown in figure 6.6. In order to have a better image for this paper, the survey with only two recommendation has been chosen. The number of movies displayed in this phase are 2,5 or 10 depending from which specific survey the user is performing. All the not necessary links are removed. The user cannot see the details of a movie either or click on the genres of a movie


Movish

Demographic > Free rating > Catalog > Recommendation > Last question

Where did this survey take place?	University
Why did you accept to take part to the survey?	Friend request

Figure 6.7: Survey phase 5

to perform a search. Given the poster and the title of the movies in the list only the user has to ask question regarding the number of movies watched, the number of movie the user have heard of, the number of watched movies the user likes or dislikes, the recommendation sentiment or if the recommendation is useful for the user or not, which is the main question leading this research and the number of movies from the recommendation that the user will be likely to watch.

After submitting the recommendation form the user is redirected to the last questions shown in figure 6.7. The user is asked of inserting the place in which the survey is taking place and why the user is performing the

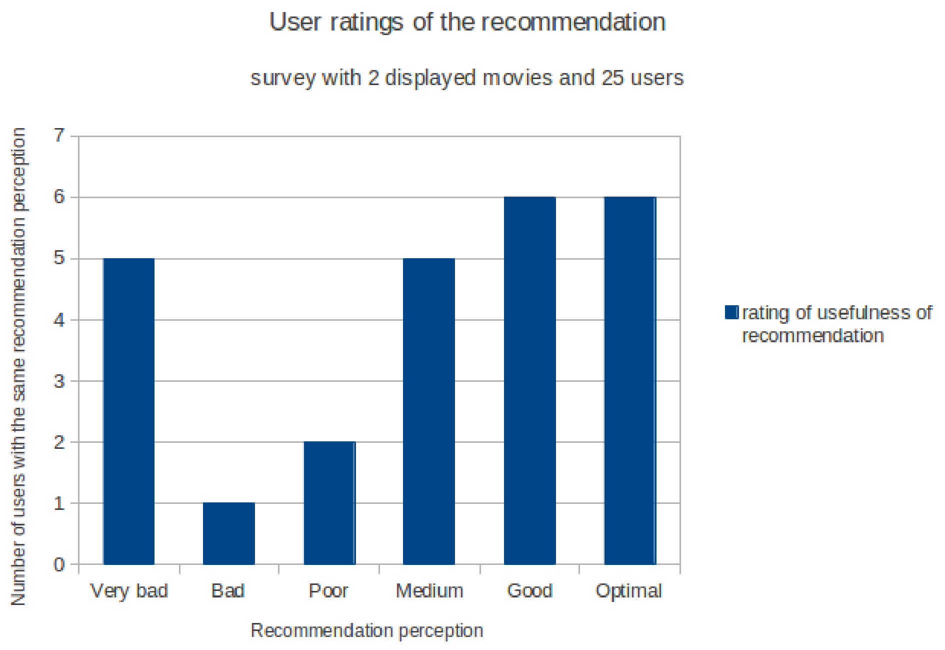


Figure 6.8: Survey having two displayed movies

survey.

6.4 Analysis

The data of the survey can be retrieved from the survey list of the admin interface described in section 4.2.2 by clicking on the *download results* icon after the *Amazon Turk link* column.

For this kind of research the feeling of a good recommendation, such as the rating of the recommendation list, is the driver value of the research. So the first way to look at the data is to count the number of equal ratings and plot them in a barchart, one for each survey. This data is available from the questions of the survey at the question regarding the usefulness of the recommendation.

Figure 6.8 shows the number of users grouped by the rate on the recommendation or the usefulness of the recommendation. The first results

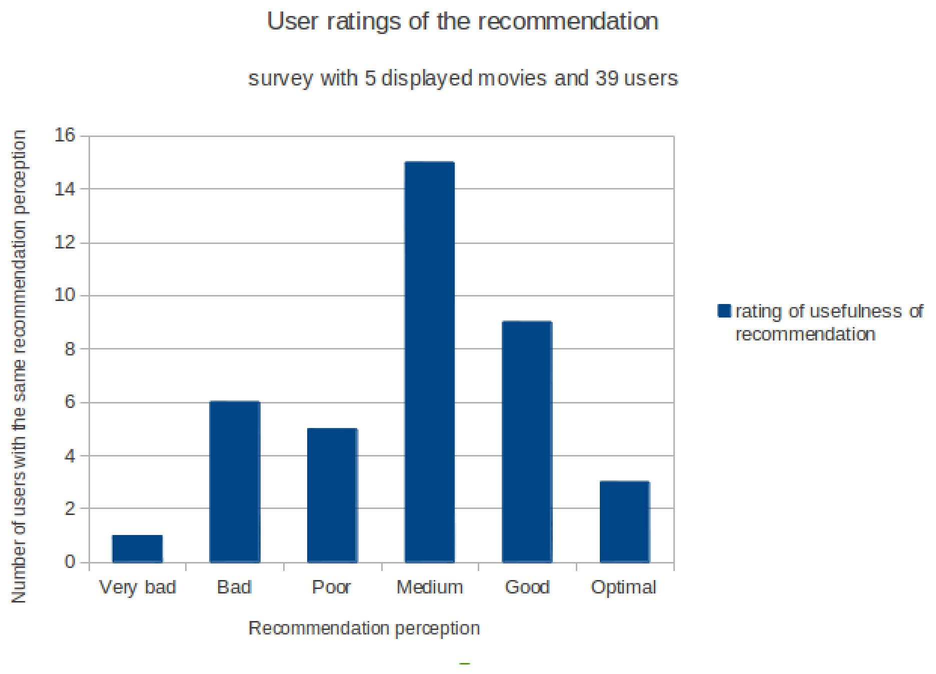


Figure 6.9: Survey having five displayed movies

that highlights in that graph is the fact that over 25 surveys, displaying 2 movies resulted in a very bad recommendation for 5 users. It was also very good for 12 users (summing the *good* and *optimal* columns). The **average** of all the ratings is **2.96**, the **standard deviation** is **1.84**.

Figure 6.9 shows the same graph for the second survey, the one displaying 5 movies at time while the user has to answer the question about the usefulness of the recommendation. A big amount of users thinks that the recommendation has medium usefulness. The big difference with the previous data is that the number of very bad recommendation is down to only one case and that the number of users that declare that the recommendation is optimal (score of 5) is significantly dropped. The **average** of all the ratings is **2.87** and the **standard deviation** is **1.24**.

Figure 6.10 shows the same graph for the third survey instead, the one in which the user sees 10 movies at the time the user has to rate the rec-

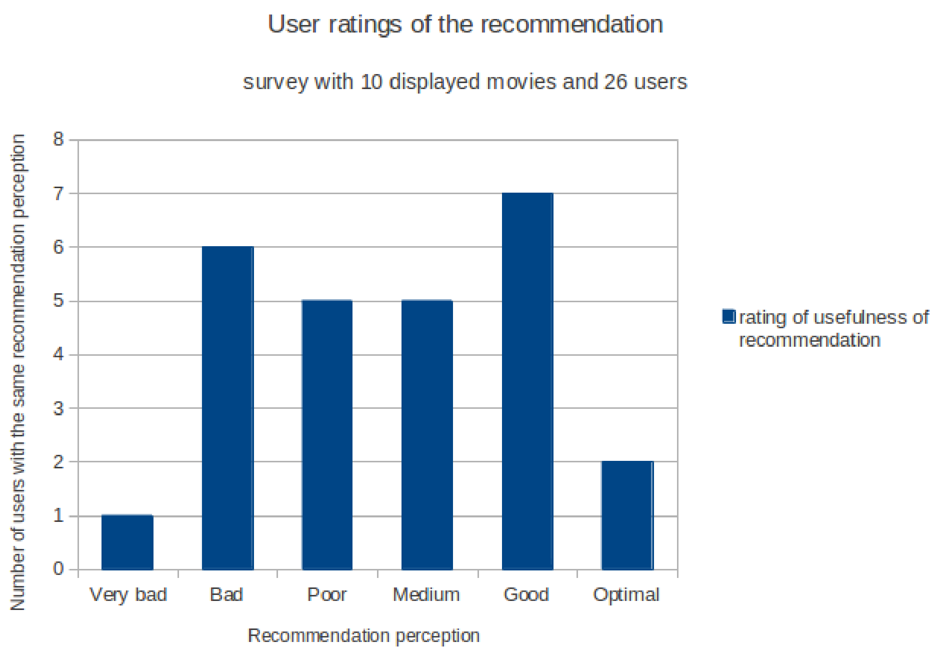


Figure 6.10: Survey having ten displayed movies

Movies in survey	Average	Standard deviation
2	2.96	1.84
5	2.87	1.24
10	2.65	1.41

Table 6.1: Average and standard deviation of the ratings in the three different surveys

ommendation he/she received. In this graph the number of user that rate the recommendation very bad is low like in the survey with 5 movies displayed but the number of people that rated the recommendation as poor is the same as the people that rated the recommendation to be medium. The **average** of all the ratings is **2.65** and the **standard deviation** is **1.41**.

The collected averages and standard deviations are in table 6.1.

As the reader can see, the survey with 10 movies displayed is the one that has been disliked the most. Its standard deviation is also higher of the

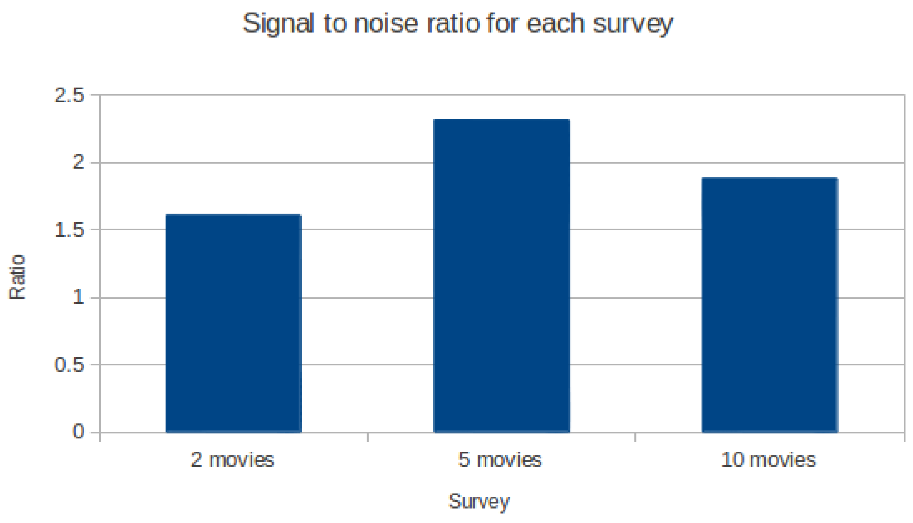


Figure 6.11: Signal to noise ratio for the three surveys

one with 5 movies displayed. So we can exclude the 10 movies recommendation as being the best way, according to the data of this surveys to be the best way to display a recommendation. Analyzing the surveys with 2 and 5 movies displayed is more challenging though since in the 2 movies survey the average is higher of the one of 5 movies survey but the standard deviation is bigger than the one with 5 movies. In order to compare this data to have a singular number to measure the effectiveness of a survey against another the signal to noise ratio can be useful.

We can consider the average as the primary driver or signal and a standard deviation as a noise factor because it indicates how much the ratings of the recommendation deviate from the average.

The signal to noise ratio is defined as $SNR = \mu/\sigma$. Using it, every survey can be reduced to a single scalar indicating which survey that had a different number of displayed movie was the best for the users. This leads to data in figure 6.11

This last graph shows how each survey performs in terms of user likeliness of the recommendation based on the number movies displayed during

Movies in survey	SNR
2	1.61
5	2.31
10	1.88

Table 6.2: Signal to noise of the three surveys

the recommendation phase. The higher is the bar, the better is the survey.

Table 6.2 shows the signal to noise ratio of the three surveys instead. As for the graph, the numeric value highlights the survey with 5 displayed movie to be the best in terms of average and standard deviation.

6.5 Conclusions

This research wanted to prove if the usefulness of a recommendation depends on the number of items that are displayed to the user. In order to perform the research three surveys with the same recommendation algorithm have been submitted to 90 different users using Amazon Mechanical Turk and Facebook. The only difference between the surveys was the different number of movies displayed during the recommendation. The first survey had two displayed movies, the second one had five movies and the third one had ten movies. The collected data showed a relation between the different number of movie displayed and the usefulness of the recommendation itself. After further study, considerations and calculating the mean and the standard deviation the three surveys have been compared using the signal to noise ratio. This lead the survey with five movies displayed to be the one that the user prefer compared to the one with two or ten. If there wouldn't be any relation between the number of displayed movies and usefulness of the recommendation there wouldn't be any significant difference in the average usefulness of the recommendation and its standard deviation.

Bibliography

[1] Wikipedia. Netflix prize. http://en.wikipedia.org/wiki/Netflix_Prize.

[2] Moviri. Contentwise. http://www.moviri.com/.

[3] Massimo Di Pierro. Web2py. http://www.web2py.com/.

[4] Mathworks. Matlab. http://www.mathworks.com/products/matlab/.

[5] Joakim Moller. pymatlab. http://pypi.python.org/pypi/pymatlab.

[6] Free Software Foundation. http://www.gnu.org/licenses/gpl.html.

[7] imdb.com. Internet movie database. http://imdb.com/.

[8] Turrin R. Chiodi L., Cremonesi P. Spiegazioni e confidenza nei sistemi di raccomandazione. Master's thesis, Politecnico di Milano, 2010.

[9] Li Chen and Pearl Pu. A user-centric evaluation framework of recommender systems. *workshop of RecSys'10*, 2010.

[10] Recsys. http://recsys.acm.org/.

[11] Jakob Nielsen's Alertbox. How long do users stay on a webpage. http://www.useit.com/alertbox/page-abandonment-time.html.

[12] M. J. Pazzani and Daniel Bilsus. *Content based recommendation systems*. Springer, 2007.

[13] Paolo Cremonesi and Roberto Turrin. Analysis of cold-start recommendations in iptv systems. *RecSys*, 2009.

[14] P. Maes and Shardanand U. Social information filtering: Algorithms for automating "word of mouth". *ACM CHI*, 1995.

[15] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *ACM*, 1992.

[16] E. Campochiaro, P. Cremonesi, and R. Turrin. Analysis of recommender systems based on implicit datasets. *Technical report*, 2008.

[17] Jonathan L. Herlocker, Konstan Joseph A., Terveen Loren G., and Riedl John T. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 2004.

[18] Andreia Coronado Cha. *Implementation of a subjective evaluatio application for recommender systems*. PhD thesis, Politecnico di Milano, 2011.

[19] Achille, Niccolo, Cremonesi Paolo, and Turrin Roberto. *Realizzazione di un sistema di raccomandazione in Matlab*. PhD thesis, Politecnico di Milano, 2010.

[20] K. Nagaswara Rao and V. G. Talwar. Application domain and functional classification of recommender systems - a survey. *DESIDOC J. Libr. Inf. Technol.*, 2008.

[21] J. Leino and K. Raiha. Case amazon: ratings and reviews as part of recommendations. *RecSys*, 2007.

[22] L. Chen and P. Pu. Trust building in recommender agents. *ICETE*, 2005.

[23] Facebook inc. Facebook. http://www.facebook.com.

[24] Twitter inc. Twitter. http://twitter.com.

[25] Pinterest inc. Pinterest. http://pinterest.com.

[26] Wikipedia. Mvc. http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controlle

[27] Pyramid. http://www.pylonsproject.org/.

[28] Numpy developers. Numpy. http://numpy.scipy.org/.

[29] John Hunter. Scipy. http://www.scipy.org/.

[30] Wikipedia. Nosql. http://en.wikipedia.org/wiki/NoSQL.

[31] 10gen. Mongodb. http://www.mongodb.org/.

[32] The PostgreSQL global development group. Postgresql database. http://www.postgresql.org/.

[33] Davide Alberani. Imdbpy. http://imdbpy.sourceforge.net/.

[34] Stefan Behnel. Lxml. http://lxml.de/.

[35] Alvaro Lopez Ortega. Cherokee. http://www.cherokee-project.com/.

[36] Wikipedia. C10k problem. http://en.wikipedia.org/wiki/C10k_problem.

[37] nginx. Nginx. http://nginx.org/.

[38] Apache Foundation. Apache webserver. http://httpd.apache.org/.

[39] unbit. uwsgi. http://uwsgi-docs.readthedocs.org/en/latest/.

[40] Wikipedia. Ajax. http://en.wikipedia.org/wiki/Ajax_(programming).

[41] TorProject. Tor. https://www.torproject.org/index.html.en.

[42] PJ Hyett Tom Preston-Werner, Chris Wanstrath. Github. https://github.com/.

[43] Junio Hamano Linus Torvalds. Git revision control system. http://git-scm.com/.

[44] NLTK Project. Python natural language toolkit. http://nltk.org/.

[45] Amazon inc. Amazon mechanical turk. https://www.mturk.com/mturk/welcome.

[46] Wikipedia. Crowdsourcing. http://en.wikipedia.org/wiki/Crowdsourcing.

[47] Paolo Cremonesi, Yehuda Korem, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. *recSys2010*, 2010.

[48] University of Minnesota. Movielens. http://movielens.umn.edu.