Evaluating Collaborative Filtering Over Time

Neal Kiritkumar Lathia

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University of London.

Department of Computer Science

University College London

June 14, 2010

To my parents

and their fervent passion for education

Abstract

Recommender systems have become essential tools for users to navigate the plethora of content in the online world. Collaborative filtering—a broad term referring to the use of a variety, or combination, of machine learning algorithms operating on user ratings—lies at the heart of recommender systems' success. These algorithms have been traditionally studied from the point of view of how well they can *predict* users' ratings and how *precisely* they rank content; state of the art approaches are continuously improved in these respects. However, a rift has grown between how filtering algorithms are investigated and how they will operate when deployed in real systems. Deployed systems will continuously be queried for personalised recommendations; in practice, this implies that system administrators will iteratively retrain their algorithms in order to include the latest ratings. Collaborative filtering research does not take this into account: algorithms are improved and compared to each other from a *static* viewpoint, while they will be ultimately deployed in a *dynamic* setting. Given this scenario, two new problems emerge: current filtering algorithms are neither (a) designed nor (b) evaluated as algorithms that must account for time. This thesis addresses the divergence between research and practice by examining how collaborative filtering algorithms behave over time. Our contributions include:

1. A fine grained **analysis** of temporal changes in rating data and user/item similarity graphs that clearly demonstrates how recommender system data is dynamic and constantly changing.
2. A **novel methodology** and time-based **metrics** for evaluating collaborative filtering over time, both in terms of accuracy and the diversity of top-N recommendations.
3. A set of **hybrid algorithms** that improve collaborative filtering in a range of different scenarios. These include temporal-switching algorithms that aim to promote either accuracy or diversity; parameter update methods to improve temporal accuracy; and re-ranking a subset of users' recommendations in order to increase diversity.
4. A set of **temporal monitors** that secure collaborative filtering from a wide range of different temporal attacks by flagging anomalous rating patterns.

We have implemented and extensively evaluated the above using large-scale sets of user ratings; we further discuss how this novel methodology provides insight into dimensions of recommender systems that were previously unexplored. We conclude that investigating collaborative filtering from a temporal perspective is not only more suitable to the context in which recommender systems are deployed, but also opens a number of future research opportunities.

Acknowledgements

Over the past years, I have been very lucky: I have been surrounded by brilliant, intelligent and inspiring people. They contributed to this thesis with their questions, insights, encouragement, and support; I am much indebted to them all. I will never be able to thank my supervisors, Steve Hailes and Licia Capra, enough: being mentored by researchers of this calibre was often all the motivation I needed. Thanks to Cecilia Mascolo, who was the first to suggest that I apply for a PhD (would I be writing this had it not been for that suggestion?); Daniele Quercia, with his unrivalled and contagious passion for research (and blogging); and all of the members of the MobiSys group. Thanks to EPSRC Utiforo, for the financial support, and thanks to all the project partners for the colourful meetings. A special thanks to Torsten Ackemann: the experiments I ran over the past few years would still be running had it not been for his invaluable help with the department's Condor cluster.

A highlight of the recent years is the time I spent in Telefonica I+D's Multimedia Group in Barcelona. A big thanks to Xavier Amatriain, Josep M. Pujol and Jon Froehlich; I not only learned a lot during these summer months, but made some great friends and thoroughly enjoyed my time there. I hope to one day finally manage to go hiking with Jon.

While all those with whom I worked with deserve my utmost thanks, I am even more indebted to my family and friends, who were there to take my mind off of my PhD. Thanks to Paul, Usha, Fergal and Preeya; to Pavle and Justin (we await your return to London), and Viktor (who always turned up at my doorstep at the right time). Thanks to my sisters, Sheila and Anna (who has put up with living with me). A special thanks to Yasmin, who has always been there for me. Lastly, thanks to the bands I have been a part of over these years (The Hartes; Pavle, and The Jukebox Leans; Sean and Duncan), for allowing me to keep nurturing my love for music.

This thesis is dedicated to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The birth and proliferation of the Internet has revolutionised the way people interact with and consume information. Resources that were previously difficult and expensive to find are now instantly available; where communication used to be slow, it is now instantaneous and effortless, and tantalising projects such as group-edited encyclopedias are now easily crowdsourced to millions. However, the evolving nature of the web—including the rise of social media—means that the paradigm shift in how people interact online continues to this day. A recent example is the explosive growth of microblogging: the content of web (and, in particular, social network) sites is constantly growing. The single constant aspect of this landscape is that it is ever-changing.

Two themes have emerged from this setting. The first is *cooperation*: many online tasks now take full advantage of interactions between web users in order to help each of them accomplish their own goals. Wikis, blogs, product reviews, and question/answer sites are a handful of examples where users can both contribute (produce) and gain (consume) information: the Internet has blurred the lines between those who create and enjoy content. The second is a battle for users' *attention*. There is now a plethora of sources where content is available; keeping up to date with the latest (movies, news, music, etc) is an established daily challenge.

Recommender systems have emerged in the last decade as powerful tools that people can use to navigate large databases according to their own interests. The engine that underlies these tools is a Collaborative Filtering (CF) algorithm, which is an automated means of ranking content "based on the premise that people looking for information should be able to make use of what others have already found and evaluated" [ME95]. In doing so, these systems capture both of the above themes: users implicitly *cooperate* as the CF algorithm uses each of their ratings and can focus their *attention* on the content that is likely to be of most interest to them. The key insight here is that, while retrieving information from the web tends to be a solitary task, the collective set of experiences can be used to help each individual: recommender systems can evaluate information quality based on the preferences of others with a similar point of view.

In this chapter, we introduce the motivations that seed recommender system research (Section 1.1) and provide a brief overview of the history of recommender system research (Section 1.2); we then define the scope and problem space that we will address in this thesis and outline the contributions that all subsequent chapters will make (Section 1.3). We close this chapter by listing a set of publications that are related to this thesis.

1.1 Motivating Information Filtering

The most widely discussed motivation for researching recommender systems has remained largely unmodified throughout the 15 years that these systems have been actively studied. It can be described as follows. People's ability to make meaningful use of information (for example, finding and reading interesting news reports) is rapidly falling behind the *rate* at which information is growing (i.e., the number of news reports that are becoming available). The web itself is not only saturated with content, but constantly growing and evolving; people now suffer from the effects of *information overload*. Although this term was first coined by Alvin Toffler in 1970 [Tof70], it continues to describe the difficulty people have when trying to navigate large information repositories—regardless of whether they contain web documents or research articles, e-commerce catalogue products, musicians, movies (and so forth). Recommender systems come to the rescue by suggesting new content to users based on what they have liked in the past [PM97].

Clay Shirky has recently provided an alternative perspective which also substantially motivates research into information filtering [Shi09]. He claims that "we are to information overload as fish are to water," and argues that information abundance has become the norm rather than the problem. Instead, he argues that our focus should be on identifying how we previously filtered information and why those filters, in the face of the information age, are no longer appropriate. For example, publishers—who have the means to produce any book of their choosing—filter their output based on an economic incentive: they are unlikely to sell books of low quality. The advent of the web, however, removed the physical cost of binding a book. In doing so, it eliminated the incentive to publish a narrower range of titles. Furthermore, individuals can now circumvent the publishing houses altogether and directly publish their content online at little to no cost. In other words, we used to filter publications based on (a) money (the economic incentive to sell what is published) and (b) convenience (it was simply too difficult to self-publish). The web has broken the filters we used to rely on by removing these constraints: we now need new ways of filtering published media, and using automated recommender systems to do so may be the tool we are looking for.

The final motivation that we discuss here also takes a different stance to the general context of information filtering. As above, a vast amount of available content is assumed to already exist; the motivation to filter it is that doing so results in heightened user activity (which often translates to increased revenue for web-based businesses). Building tools like recommender systems, that offer personalised views of a web site's content to visiting users, encourages people to actively engage with the site: two thirds of the movies rented by Netflix.com were recommended, Google news recommendations result in 38% more clickthroughs, and 35% of the product sales on Amazon.com were recommended items [CL07]. The motivation to filter information is therefore the fact that tailoring what each user sees to their own needs has been the secret to success of a number of online business.

1.2 Brief History of Recommender Systems

Over the last decade, research into recommender systems has evolved: the particular target scenarios that have been explored have mirrored changes to the way people use the Internet. In the early 1990s, the first filtering system, Tapestry, was developed at the Xerox Palo Alto Research Center [GNOT92]. This system, recognizing that simple mailing lists do not ensure that all users interested in an e-mail's content receive the message, allowed users to annotate e-mail messages so that others could filter them by building queries. This was the first system to capture the power of combining human judgments (expressed as message annotations) with automated filtering, in order to benefit all of the system's users. Similar concepts were later applied to Usenet news by the GroupLens research project, which extended previous work by applying the same principles to the Internet discussion forum, which had become too big for any single user to manage [KMM$^+$97]. The GroupLens project subsequently implemented the MovieLens movie recommender system; the valuable rating data from it was then made available to the wider research community[1], which subsequently shifted focus from news boards toward filtering movies.

The initial success that recommender systems experienced is reflected in the surge of e-commerce businesses that implement them; Schafer *et al.* review and describe a number of mainstream examples [SKR99, SKR01]. The cited sites, like Amazon.com and CDNow.com, implement recommenders to build customer loyalty, increase profits, and boost item-cross selling. More recently, web sites like Last.fm have reaped the benefits of collecting user-music listening habits, in order to provide customized radio stations and music recommendations to their subscribers. The influence, presence, and importance of the recommender system is not only well established, but continues to grow over time.

The widespread commercial applicability of recommender systems is mirrored in the research domain by the extensive breadth of fields that these systems have a presence in. Recommender systems are researched in the context of statistics [AW97], machine learning [CS01], human-computer interaction [PC06, HKR00], social network analysis [MY07], distributed and mobile systems [MKR05, LHC07], agent-based artificial societies [WBS07], computational trust [LSE08, ARH97], and more: it is becoming impossible to capture all of the contributions that are being made to recommender system research. More recently, researchers have explored how content-annotations (tags) can be used to compute recommendations [ZC08], how mobility can be used to recommend social network connections [QC09], disseminate content over mobile networks [QHC07], filter online news [DDGR07] and improve search engine performance [SBCO09].

The most significant recent event related to recommender system research was the announcement of the Netflix prize in late 2006. Netflix—an online DVD rental company from the U.S.—released a dataset of user-movie ratings which, to date, remains the largest publicly available set of user-ratings. They challenged the broader community to outpredict their own system by at least 10% and offered a million dollar reward to the team that was best able to do so. The competition's award itself shows the extent to which web-based businesses value their recommender systems; over 20,000 teams spent 3 years tackling the prediction problem before the winners were announced [Kor09b, TJB09, PC09]. A variety

[1] http://www.grouplens.org/node/73

of lessons were learned throughout the course of the competition; we highlight three here:

- **Matrix Factorisation** emerged early in the competition as a powerful prediction algorithm for CF [Pia07]; it subsequently was consistently used throughout all the leading solutions.
- **Ensemble Methods**. The winning teams did not invest their time in designing accurate predictors; instead, they combined hundreds of individual prediction methods that *together* achieved the target prediction accuracy.
- **Temporal Dynamics**. The Netflix dataset included the date when users input each rating; this data was soon found to be very useful when predicting user tastes, since it reflects the changing bias that users may have [Pot08] or can be incorporated into a wider set of classifiers [Kor09a].

The competition also raised a number of questions, which motivate the work in this thesis.

- **Competition Structure**. Does the structure of the competition (i.e., predicting a hidden set of user ratings) reflect how recommender systems are used in practice? In this thesis, we discuss and propose a novel methodology that more closely reflects the reality of deployed recommender systems.
- **Metrics**. The focus of the competition was accuracy: is this the best way to measure the performance of a recommender system? More importantly, while the leading solutions certainly *predict* ratings well, do they provide better *recommendations*? In this thesis, we examine these points by evaluating collaborative filtering across a number of different dimensions (accuracy, diversity, and robustness).

In the following section, we examine these questions by defining the scope of the research presented in this thesis.

1.3 Problem Statement and Contributions

Recommender systems are built as navigational tools and widely deployed online. In practice, this means that a CF algorithm is implemented and then trained with all the available ratings that the system has for the current content; the algorithm can then be queried to produce recommendations for each user. This process is repeated in a cyclical manner. Why? CF algorithms tend to suffer from very high latency; training an algorithm with the ratings of (potentially millions of) users is a very expensive operation, often requiring exponential space and time, and can thus not be repeated at will (there are, however, a few exceptions [GRGP00]). Recommender systems therefore tend to perform iterative, regular updates (e.g., weekly [Mul06]). Users will not be consistently offered newly computed recommendations, and will have to wait for a system update for their latest ratings to be included in the CF training phase. Since recommendations often elicit further ratings, CF algorithms are iteratively retrained in order for them to have learned from all the data (including any that may have been input since they were last trained).

The traditional research methodology used to design and evaluate CF algorithms, instead, is a two-phase process: researchers measure the performance of an algorithm by first training a given algorithm with a set of ratings and then querying it for recommendations. The problem here is that the research methodology is static, while deployed systems operate in a cyclical manner: there is a rift between how CF algorithms are studied and how they will be used in practice.

In the following chapters, we address problems that revolve around the central theme of *temporal updates to a recommender system*. The research in this thesis constitutes both *methodological* and *algorithmic* contributions; the former being supported by analysis of large scale rating datasets, and the latter validated with empirical experiments. We decompose problems related to this into three groups: those pertaining to the rating data, evaluating a system that is updated, and securing the robustness of an updating system:

- **Temporal Features of Rating Data**. We report the results of an extensive temporal analysis of three rating datasets (Chapter 3). We draw two main conclusions: (a) CF datasets are subject to a variety of changes over time that are not accounted for when computing recommendations (ranging from dataset growth to customer preference drift) and (b) state of the art similarity measures violate the CF assumption that like mindedness persists between people: they do not produce values that consistently reflect similar people.
- **Evaluating Recommender Systems Over Time**. We define a novel methodology for evaluating the temporal performance of CF algorithms (Chapter 4), based on simulating a number of iterative updates. We accompany this methodology with a number of novel metrics that we use to visualise two facets of CF performance:
    - **Accuracy**. We show how temporal accuracy results provide insight into a facet of performance that would otherwise go unnoticed: accuracy does not improve over time, or with additional ratings. We then propose and evaluate a set of hybrid-switching CF algorithms that keep track of and improve upon their own temporal performance. We show how the same switching strategy can be applied to a mixed set of CF algorithms, or to select and update the k-Nearest Neighbour or Singular Value Decomposition parameters.
    - **Diversity**. We define a novel metric for temporal diversity and show how diversity relates to accuracy. Based on the observations we make when analysing CF temporal diversity, we propose and evaluate a set of algorithms that promote diversity over time (Chapter 5).
- **Securing Recommender System Robustness**. We examine the threats that CF algorithms face from a temporal perspective, and show how attackers who do not factor time into their attack can easily be defeated. We then define how temporal attacks may be conducted and show their effects on a large scale dataset of user ratings. We design and evaluate a series of monitors that can identify when a variety of attacks are taking place. We finally show how attackers may modify their attacks in order to circumvent these defences, and discuss the additional difficulty they will face when trying to do so (Chapter 6).

1.3.1 Timeliness of Research

With the completion of the Netflix prize, collaborative filtering research is reaching an interesting juncture. Striving for accuracy, which has long been the focal point of CF evaluation, has now been pushed to extreme limits [APO09]. New themes, such as context-aware [ASST05] and mobile [QC09] recommender systems are beginning to emerge, and recommender system methods are now being applied to new domains (such as large-scale software projects [LQF10]). However, the temporal aspect of recommender systems has not been addressed, and *all* of the new application domains assume systems that will be deployed over time. Furthermore:

- While recommender systems are widely used online, there is no insight into how these systems perform as they are updated and users continue rating content.
- The importance of *time* has emerged from the Netflix prize. However, work to date only considers the importance of time in terms of drifting customer preferences [Kor09a]. In this thesis, we consider an alternative (though not mutually exclusive) perspective: how the system performs over time.

We believe that the work in this thesis is timely since it proposes novel methods and metrics to evaluate CF algorithms' temporal performance and provides insights based on empirical evidence that cannot be investigated with current research methods.

1.4 Publications Related To This Thesis

The following publications (and submissions) are related to this thesis:

1. [Lat08a] N. Lathia. Computing Recommendations With Collaborative Filtering. Chapter 2 in *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*. pp. 23–41. September 2008. IGI Global.
2. [LHC08c] N. Lathia, S. Hailes, L. Capra. Trust-Based Collaborative Filtering. In *Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM)*. pp 119–134. July 2008. Trondheim, Norway.
3. N. Lathia, S. Hailes, L. Capra. The Role of Trust in Collaborative Filtering. Under Submission.
4. [LHC08b] N. Lathia, S. Hailes, L. Capra. The Effect of Correlation Coefficients on Communities of Recommenders. In *ACM SAC TRECK*. pp. 2000–2005. March 2008. Fortaleza, Brazil.
5. [LHC08a] N. Lathia, S. Hailes, L. Capra. kNN CF: A Temporal Social Network. In *ACM Recommender Systems (RecSys)*. pp. 227–234. October 2008. Lausanne, Switzerland.
6. [LHC09b] N. Lathia, S. Hailes, L. Capra. Temporal Collaborative Filtering With Adaptive Neighbourhoods. In *ACM SIGIR*. pp. 796–797. July 2009. Boston, Massachusetts, USA.
7. [LHC09a] N. Lathia, S. Hailes, and L. Capra. Evaluating Collaborative Filtering Over Time. In *ACM SIGIR Workshop on the Future of IR Evaluation*. pp. 41–42. July 2009. Boston, Massachusetts, USA.
8. [LHC10b] N. Lathia, S. Hailes and L. Capra. Temporal Diversity in Recommender Systems. In *ACM SIGIR*. July 2010. Geneva, Switzerland.
9. [LHC10a] N. Lathia, S. Hailes, L. Capra. Temporal Defenses for Robust Recommendations. ECML/PKDD Workshop on Privacy and Security Issues in Data Mining and Machine Learning September 2010. Barcelona, Spain.

These appear in this thesis as follows: [Lat08a, LHC08c] and [3] review state of the art approaches to CF (Chapter 2), [LHC08b, LHC08a] investigate the temporal qualities of CF algorithms (Chapter

3), [LHC09b, LHC09a] propose novel metrics and algorithms for predicting ratings over time (Chapter 4), [LHC10b] analyses the diversity of recommendations and evaluates methods to augment temporal diversity (Chapter 5), and [LHC10a] addresses the problem of system robustness (Chapter 6).

There are also a number of other publications that were completed during this time period; while relevant to the broader topics of trust and collaborative filtering, they are not directly within the scope of this thesis:

1. [LHC07] N. Lathia, S. Hailes, L. Capra. Private Distributed Collaborative Filtering Using Estimated Concordance Measures. ACM RecSys 2007. Minneapolis, USA.
2. [Lat08b] N. Lathia. Learning to Trust on the Move. In Joint TIME-SPACE Workshops (IFIPTM). June 2008. Trondheim, Norway.
3. [ALP$^+$09] X. Amatriain, N. Lathia, J.M. Pujol, H. Kwak, N. Oliver. The Wisdom of the Few: A Collaborative Filtering Approach Based on Expert Opinions From the Web. In ACM SIGIR. July 2009. Boston, Massachusetts, USA.
4. [LAP09] N. Lathia, X. Amatriain, J.M. Pujol. Collaborative Filtering With Adaptive Information Sources. In *IJCAI Workshop on Intelligent Techniques for Web Personalization and Recommender Systems*. July 2009. Pasadena, California, USA.

1.5 Summary

In this chapter, we have introduced the generic scenario that recommender systems are best suited to: settings where the *volume* of available information is so great that it exceeds the *ability* that users have to find what they are looking for. Recommender systems are useful since they push content to users without requiring them to formulate explicit queries. Instead, they use the implicit *cooperation* between users in order to rank content for each one of them. We discussed the various motivations that may lie behind building such systems: (a) coping with information overload, (b) replacing filters that no longer work, and (c) making money by encouraging users to interact with their recommendations. We then briefly reviewed the 15 years that recommender systems have been researched and the variety of fields that contribute to it—ranging from statistics to human computer interaction. We placed a particular emphasis on the Netflix prize, since questions that arise from the competition's *structure* and *metric* of choice motivated the work in the following chapters.

In this thesis, we focus on the temporal performance of CF algorithms: we aim to analyse and measure how CF operates over time. In doing so, we endeavour to make both methodological and algorithmic contributions, ranging from a variety of temporal analyses (ratings, similarity, prediction and diversity performance) to a wide range of algorithms to address the accuracy, diversity, and robustness of these algorithms over time. We begin in the following chapter by reviewing state of the art collaborative filtering algorithms.

Chapter 2

Computing Recommendations With Collaborative Filtering

Recommender systems, based on Collaborative Filtering (CF) algorithms, generate personalised content for their users by relying on a simple assumption: those who have had similar opinions in the past will continue to share similar tastes in the future. This chapter serves as a review of the state of the art in collaborative filtering. We first introduce the rating data, detailing how the ratings are collected (Section 2.1). We then introduce the various approaches that have been adopted when designing CF algorithms. In particular, we differentiate between *data mining* (Section 2.2) and *user modelling* (Section 2.3) approaches: the focus of the former is on designing algorithms that augment the predictive power of CF when applied to a set of ratings; the latter, on the other hand, builds methods based on preconceived models of trust between system users. We then explore how these systems are evaluated (Section 2.4), including the methodology and metrics related to CF evaluation, and enumerate a number of open problems (Section 2.5) that we will address in this thesis.

2.1 Ratings And User Profiles

The focal point of recommender systems is the *user-rating data* upon which the system operates. Before introducing the underlying algorithms of recommender systems, we define the terms related to this data that will be used throughout this thesis.

- **User:** the end user of the system, or the person we wish to provide with recommendations. The entire set of users is referred to as the community.
- **Item:** a generic term used to denote the system's content, which may be a song, movie, product, web page, etc.
- **Rating:** a numerical representation of a user's preference for an item; these are considered in more depth in Section 2.1.1.
- **Profile:** the set of ratings that a particular user has provided to the system. Equivalently, the profile of an *item* is the set of ratings that have been input for that item by the users.
- **User-Item Matrix:** The ratings input by all users are often represented as a matrix, where columns are individual items, and rows are users. Each matrix entry $r_{u,i}$ is the rating (an numerical value

on a given range, e.g. 1-5 stars) input by user u for item i.

Recommender systems revolve around the set of user profiles; by containing a collection of ratings of the available content, this set is the focal source of information used when providing each user with recommendations. Ratings can be collected either *explicitly* or *implicitly*; in the following section we describe and compare each of these methods.

2.1.1 Implicit and Explicit Ratings

Ratings (human opinions of the system's content) can come from two separate sources. On the one hand, the opinions could be in the form of *explicit* ratings: users may be asked to rate items on a N-point (e.g., five-star) Likert scale. In this case, the rating is a numeric value that is input directly by the user. On the other hand, opinions can be extracted from the user's *implicit* behaviour. These include time spent reading a web page, number of times a particular song or artist was listened to, or the items viewed or purchased when browsing an online catalogue; logging implicit behaviour is an attempt to capture taste by reasoning on how users interact with the content.

The most significant difference between explicit and implicit ratings is that, in the latter scenario, users cannot tell the system that they *dislike* content: while an explicit scale allows users to input a low score (or negative feedback), implicit ratings can only infer positive behaviour. Hu *et al* [HKV08] further this notion by describing how, in the implicit case, the system does not determine a user's *preference* for an item, but rather reasons on the *confidence* it has in a user's affinity to the item, based on measured behaviour. A consequence of this is the possibility of making noisy measurements. For example, Hu *et al* [HKV08] mention that it is impossible to differentiate between a user who watches a television program for a long period of time or is asleep in front of the television; this is a problem where simple thresholding is not a sufficient solution. However, Amatriain *et al* [APO09] show that explicit ratings are also prone to noise: users may be careless and irregular when they manually input ratings.

The art of collecting ratings thus seems to be context-specific: furthermore, Herlocker *et al* identified that the act of rating itself is motivated by different reasons, including self-expression and helping or influencing others' decisions [HKTR04]. For example, movie-recommender systems often prefer to let users explicitly rate movies, since users may dislike a particular movie they have watched. Music recommender systems tend to construct user profiles based on listening habits, by collecting metadata of the songs each user has listened to; these systems favour implicit ratings by assuming that users will only listen to music they like. Implicit ratings can be converted to a numeric value with an appropriate transpose function (the algorithms we describe next are hence equally applicable to both types of data).

Both implicit and explicit ratings share a common characteristic: the set of available judgments for each user, compared to the total number of items that can be rated, will be very small. In particular, it is impossible to determine whether an item remains *unrated* because it is disliked or because the user has simply not yet encountered the content to date. The lack of information is known as the problem of *data sparsity*, and has a strong effect on the efficacy of any algorithms that base their recommendations on this data. However, rating sparsity is a natural consequence of the problem that recommender systems address: if it were feasible for users to find and rate all the content they were interested in, recommender systems would no longer be needed.

Rating data is related to the broader category of relevance feedback from the information retrieval community [RL03, FO95]; in fact, recommender systems can be broadly considered to be query-less retrieval systems, that operate solely using user feedback, in order to both find and rank content. The main goal of recommender systems is to filter content in order to provide relevant and useful suggestions to each user of the system. There are two methods to do so: the *content-based* and *collaborative* approaches. Content-based algorithms formulate recommendations by matching descriptions of the system's items to those items in each user profile [PB07]. These systems, however, are not appropriately or readily applied to the entire range of scenarios where users may benefit from recommendations: they require content that can be described in terms of its attributes, which may not always be the case. Furthermore, simply matching on attributes disregards what users think about the content, as the preference data remains untouched. In the following section, we review the alternative approach: collaborative filtering algorithms.

2.2 Collaborative Filtering Algorithms

The task of a recommender system algorithm is to take as input a set of user ratings and output personalised recommendations for each user. To do so, recommender systems use a collaborative filtering (CF) algorithm, which mines patterns within the ratings in order to forecast each user's preference for unrated items. At the broadest level, the recommendations are generated by:

- **Collecting Ratings**. The system collects ratings from each user.
- **Predicting Missing Values**. Collected ratings are input to a CF algorithm: the algorithm is trained with the available ratings, and asked to *predict* the values of the missing ratings.
- **Ranking and Recommending**. The predictions are used to create a personalised ranking of unrated items for each user; this tailored list is served to each user as a ranked list of recommendations.

Users can continue rating items, and the process of rate-predict-recommend continues. This process highlights a number of features of CF. The founding assumption is that *like-mindedness is persistent*: if users have shown similar interests in the past, they are likely to enjoy similar items in the future. In other words, the ratings contain useful information to train learning algorithms. CF algorithms tend to disregard any descriptive attributes of the items (or what the items actually are) in favour of the ratings, and focus on generating recommendations based solely on the opinions that have been input. Embedded in this is a fine-grained notion of similarity between items: two items are similar if they are enjoyed by the same users, regardless of what they actually are.

The problem of generating recommendations, and the use of the data that is available to tackle this task, has been approached from a very wide range of perspectives: in this section, we elaborate on a number of widely-used algorithms. Each perspective applies different heuristics and methodologies in order to create recommendations. Historically, the two broadest categories of collaborative filters were the memory and model based approaches. In the following section, we define these categories and discuss whether this grouping of approaches reflects state of the art research.

2.2.1 Grouping the Algorithms

A range of literature partitions the CF algorithms into two groups: memory based and model based approaches. Memory-based CF has often been referred to as the dominant method of generating recommendations due to both its clear structure and successful results, making it an easy choice for system developers. In essence, this group includes the k-Nearest Neighbour (kNN) algorithm and the range of variations it has been subject to (we explore these further in Section 2.2.3). Model based approaches to CF, instead, aim to apply any of a number of other classifiers developed in the field of machine learning, including the use of singular value decomposition, neural net classifiers, Bayesian networks, support vector machines, perceptrons, induction rule learning, and latent semantic analysis [BHK98, YST$^+$04, CS01] to the problem of information filtering. A complete introduction to all available machine learning algorithms is beyond the scope of this thesis, and we point the reader to appropriate introductory texts [Alp04]. Each differs in the method applied to learn how to generate recommendations, but they all share a similar high-level solution: they are based on inferring rules and patterns from the available rating data.

The kNN algorithm differentiates itself from members of the model based group by being a *lazy* or *instance-based learning algorithm*; it operates on a subset of ratings (or instances) when computing predicted values. As we explore below, the kNN training phase consists of forming neighbourhoods for each user (or item). In other words, this phase entails selecting a *subset* of the user-item matrix for each of the predictions that the algorithm will subsequently be asked to make. On the other hand, model based approaches adopt *eager learning* characteristics, and need not return to the training data when computing predictions. Instead, they formulate a *model*, or higher level abstraction, of the underlying relationships between user preferences. For example, matrix factorisation (Section 2.2.4) approaches decompose the user-item matrix into a set of user and item factors and compute recommendations with these, rather than the ratings themselves.

These differences have motivated the grouping of CF algorithms into memory and model based approaches, and lead to studies comparing the relative performance of each group in different CF domains [GFM05]. However, this partitioning is not clear-cut. For example, the kNN algorithm *models* user preferences with user (or item) neighbourhoods; a user's taste is represented by a sample of peers who have exhibited similar preferences. Similarly, model based approaches learn from the available instances (or *memory*) of user ratings; their efficacy is also strictly related to the training data.

Furthermore, grouping algorithms in this way fails to acknowledge the diversity of research approaches used when designing and evaluating CF algorithms. Broadly speaking, these approaches tend to be based on *data mining*, where the focus is on extrapolating predictive power from the available rating data, or *user modelling*, that designs algorithms using preconceived notions of how the system's users will interact with (or, for example, trust) each other. Should a user-modelling perspective, which uses a kNN algorithm, be a memory or model based approach? Is trust a memory or model based characteristic? Due to these shortcomings, in the following sections we review a set of CF algorithms (baseline, k-Nearest Neighbour, matrix factorisation, and hybrid methods) that reflect the contributions of both the mining and modelling approaches without strictly adhering to the memory/model dichotomy.

2.2.2 Baselines

The first set of approaches we examine are *baselines*. These are the simplest predictions that can be made, and include the user mean (or mean of items i rated by user u). Given a user u, with profile R_u, predictions $\hat{r}_{u,t}$ for each target item t are assigned the same value:

$$\hat{r}_{u,t} = \frac{1}{|R_u|} \times \sum_{i \in R_u} r_{u,i} \tag{2.1}$$

Similarly, given a target item t with profile R_i, item mean values are computed as:

$$\hat{r}_{u,t} = \frac{1}{|R_i|} \times \sum_{u \in R_i} r_{u,i} \tag{2.2}$$

In general, the item-mean method is preferred to the user-mean, since the latter is not conducive to ranking any of the content (since all predictions for a user will have the same value). Similarly, if the item-mean method is coupled with the *frequency* of ratings for an item, this method corresponds to recommending a non-personalised popularity-sorted ranking of items.

The *bias model*, as described by Potter [Pot08], builds on the above by predicting user u's rating for an item i using u's mean rating (bias, b_u) and the average deviation from each user's mean for item i (preference, p_i). Overall, this method is highly dependent on each user's mean to predict the ratings; we elected to investigate it since user means will change over time. For a set of users U and items I, the biases and preferences are initialised as zeroes and then computed by iterating over the following:

$$\forall i \in I : p_i = \frac{1}{|R_u|} \times \sum_{u \in R_i} (r_{u,i} - b_u) \tag{2.3}$$

$$\forall u \in U : b_u = \frac{1}{|R_i|} \times \sum_{i \in R_u} (r_{u,i} - p_i) \tag{2.4}$$

The iteration continues until the difference in the Root Mean Squared Error (RMSE) achieved on the training set is less than a pre-defined value δ. The predicted rating $\hat{r}_{u,i}$ for a user-item pair is computed as:

$$\hat{r}_{u,i} = b_u + p_i$$

The main role of these and similar [BK07] baseline methods (sometimes referred to as "global effects") has been to normalise data prior to applying other classifiers. Further amendment to the method scales each rating by the user's variance, if that variance is non-zero [Pot08].

2.2.3 k-Nearest Neighbours

The k-Nearest Neighbour (kNN) algorithm has enjoyed enourmous popularity in the domain of recommender systems; it appeared in early research efforts [HKBR99] and continues to be applied in state of the art solutions [BK07]. Two flavours of the algorithm exist: the user-based [HKBR99] and item-based [SKKR01, LSY03] approaches. The two approaches differ in how they view the underlying rating data: the user-based approach views the data as a collection of *users* who have rated items, while the item-based approach views the same data as a collection of *items* that have been rated by users. It is important to note, however, that the techniques described here can be equally applied to both user or item profiles. In the interest of consistency, we adopt the terminology of the user-based approach throughout this section.

In general, it is impossible to claim with any authority that one method will always outperform the other. However, the item-based approach is often preferred since available CF datasets tend to have many more users than items. In this section, we elaborate on how the algorithm works by decomposing it into two stages: neighbourhood formation and opinion aggregation.

Neighbourhood Formation

This first step aims to find a unique subset of the community for each user by identifying others with similar interests to act as recommenders. To do so, every pair of user profiles is compared, in order to measure the degree of similarity $w_{a,b}$ shared between all user pairs a and b. In general, similarity values range from 1 (perfect similarity) to -1 (perfect dissimilarity), although different measures may only return values on a limited range. If a pair of users have no profile overlap, there is no means of comparing how similar they are, and thus the similarity is set to zero.

Similarity can be measured in a number of ways, but the main goal of this measure remains that of modelling the potential relationship between users with a numeric value. The simplest means of measuring the strength of this relationship is to count the proportion of co-rated items, or Jaccard similarity, shared by the pair of users [Cha02]:

$$w_{a,b} = \frac{|R_{a,i} \cap_i R_{b,i}|}{|R_{a,i} \cup_i R_{b,i}|} \tag{2.5}$$

This similarity measure disregards the values of the ratings input by each user, and instead only considers what each user has rated; it is the size of the intersection of the two users' profiles over the size of the union. The underlying assumption is that two users who continuously rate the same items share a common characteristic: their choice to rate those items.

The most cited method of measuring similarity is the Pearson Correlation Coefficient (PCC), which aims at measuring the degree of linearity that exists on the intersection of the pair of users' profiles [BHK98, HKBR99]:

$$w_{a,b} = \frac{\Sigma_{i=1}^{N}(r_{a,i} - \bar{r}_a)(r_{b,i} - \bar{r}_b)}{\sqrt{\Sigma_{i=1}^{N}(r_{a,i} - \bar{r}_a)^2 \Sigma_{i=1}^{N}(r_{b,i} - \bar{r}_b)^2}} \tag{2.6}$$

Each rating above is normalised by subtracting the user's mean rating (e.g., $\bar{r}_a$), computed using Equation 2.3. The PCC similarity measure has been subject to a number of improvements. For example, if the intersection between the pair of user's profiles is very small, the resulting similarity measure is highly unreliable, as it may indicate a very strong relationship between the two users (who have only co-rated very few items). To address this, Herlocker *et al* [HKBR99] introduced significance weighting: if the number of co-rated items n is less than a threshold value x, the similarity measure is multiplied by $\frac{n}{x}$. This modification reflects the fact that similarity measures become more reliable as the number of co-rated items increases, and has positive effects on the predictive power of the filtering algorithm. The same researchers also cite the constrained Pearson correlation coefficient, which replaces the user means in the above equation with the rating scale midpoint.

Similarity measures are also often coupled with other heuristics that aim at improving the reliability and power of the derived measures. For example, Yu *et al* [YWXE01] introduced variance weighting; when comparing user profiles, items that have been rated by the community with greater variance receive a higher weight. The aim here is to capture the content that, being measurably more “controversial” (and eliciting greater disagreement amongst community members) is a better descriptor of taste. Measuring similarity, however, remains an open issue; to date, there is little that can be done other than comparing prediction accuracy in order to demonstrate that one similarity measure outperforms another on a particular dataset.

There are a number of other ways of measuring similarity that have been applied in the past. These include the Spearman Rank correlation, the Vector Similarity (or cosine angle between the two user profiles), Euclidean and Manhattan distance, and other methods aimed at capturing the proportion of agreement between users, such as those explored by Agresti and Winner [AW97]. Each method differs in the operations it applies in order to derive similarity, and may have a strong effect on the power the algorithm has to generate predicted ratings.

Opinion Aggregation

Once comparisons between the user and the rest of the community of recommenders (regardless of the method applied) are complete, predicted ratings of unrated content can be computed. As above, there are a number of means of computing these predictions. Here we present two [HKBR99, BK07]. Both equations share a common characteristic: a predicted rating $p_{a,i}$ of item i for user a is computed as a weighted average of neighbour ratings $r_{b,i}$. The first is a weighted average of neighbour ratings:

$$p_{a,i} = \frac{\Sigma_b r_{b,i} \times w_{a,b}}{\Sigma w_{a,b}} \tag{2.7}$$

The second subtracts each recommender’s mean from the relative rating; the aim of this normalisation step is to minimize the differences between different recommenders’ rating styles, by considering how much ratings deviate from each recommender’s mean rather than the rating itself.

$$p_{a,i} = \bar{r}_a + \frac{\Sigma_b (r_{b,i} - \bar{r}_b) \times w_{a,b}}{\Sigma w_{a,b}} \tag{2.8}$$

The weights $w_{a,b}$ may come from one of two sources. They may be the similarity measures we found in the first step; neighbours who are more similar will have greater influence on the prediction. On the other hand, recent work [BK07] uses similarity weights to *select* neighbours, and then re-weights the selected group simultaneously via a linear set of equations in order to maximise the accuracy of the selected group on the user’s profile.

The natural question to ask at this step is: which recommender ratings are chosen to contribute to the predicted rating? A variety of choices is once again available, and has a direct impact on the performance that can be achieved. In some cases, only the top-k most similar neighbours are allowed to contribute ratings, thus guaranteeing that only the closest ratings create the prediction. However, it is often the case that none of the top-k neighbours have rated the item in question, and thus the prediction coverage, or the number of items that can be successfully predicted, is negatively impacted. A straightforward alternative, therefore, is to consider the top-k recommenders who can give rating information about the item in question. This method guarantees that all predictions will be made; on the other hand, predictions may now be made according to ratings provided by only modestly-similar users, and may thus be less accurate. A last alternative is to only select users above a pre-determined similarity threshold. Given that different similarity measures will produce different similarity values, generating predictions this way may also prevent predictions from being covered. All methods, however, share a common decision: what should the threshold value, or value of k, be? This question remains unanswered and dependent on the available dataset; research in the area tends to publish results for a wide range of values.

In general, while the process of selecting neighbours is a focal point of kNN approaches, selecting the best neighbours is an inexact science. Rafter *et al* [ROHS09] examined the effect that neighbours (selected with the methods described above) have on prediction accuracy, and found that neighbours are often *detrimental* in this process. Instead, [LAP09] reports on the potential massive gains in accuracy if an optimal set of neighbours is selected. The problem of neighbour selection, and approaches designed according to user models, will be further addressed in the Section 2.3.

Up to this point, we have considered the process of generating recommendations strictly from the so-called memory based, nearest-neighbour approach. In the following section, we review another of the most prominent CF algorithms, which is based on matrix factorisation.

2.2.4 Matrix Factorisation

Recommender systems connect *large* communities of users to *large* repositories of content; the rating data that they contain is thus invariably sparse. A powerful technique to address this problem is that of matrix factorisation, based on Principle Component Analysis (PCA) or Singular Value Decomposition (SVD) [Pat06, MKL07, KBC07]; in this section we focus on SVD.

As detailed by Amatriain *et al* [AJOP09], the core function of an SVD is to use the sparse rating data in order to generate two *descriptive* matrices that can be used to approximate the original matrix. In other words, given a matrix of user-item ratings R, with n users and m items, the task of an SVD is to compute matrices U and V such that:

$$R = U\lambda V^T \tag{2.9}$$

U is an $(n \times r)$ matrix, and V is an $(r \times m)$ matrix, for a given number of required features r, and λ is a diagonal matrix containing the singular values. Each matrix contains important information. For example, V describes the system content in terms of affinity to each feature: in fact, this information can be used to describe the implicit relations among the system's content by showing how each item relates to others in the computed feature space.

Once the decomposed matrices have been computed, the rating for a user-item pair is approximated as the dot product between the user's feature vector (from U) and the item's feature vector (from V). In other words, for a user u and item i, the predicted rating $\hat{r}_{u,i}$ is:

$$\hat{r}_{u,i} = \sum_{f=0}^{r} U_{u,f} \times V_{f,i} \tag{2.10}$$

Based on the above, the U and V matrices themselves can be approximated iteratively: after initialising each matrix, the features can be updated in order to minimise the squared error between the computed predictions and ratings for the available data instances. This process is bounded by using an appropriate learning rate (to avoid overfitting) and a number of rounds that should be dedicated to each feature [Pia07].

Although factorisation addresses data sparsity, it does not overcome it: computing the SVD itself is challenging when data is missing. Sarwar *et al* [SKKR00] address this by replacing the missing values with item mean ratings. Factorisation serves many purposes: it may be used as both a preprocessing or prediction mechanism; in the next section, we explore how it has been used in the context of *hybrid* algorithms.

2.2.5 Hybrid Algorithms

As we have seen above, there is a wide range of algorithms suitable for the CF context. Each algorithm offers a different method of reasoning on the rating data, produces different results, and has different shortcomings. This range of differences between these methods motivates using more than one recommendation algorithm in unison, in order to reap the benefits of each method (and, hopefully, overcome the limitation that each one suffers when used alone). In [Bur02], Burke provides a comprehensive review of hybrid algorithms: in this section, we review popular approaches, which we decompose into two groups: *preprocessing* and *ensemble methods*. Each group need not be implemented alone. In fact, there is no limit as to how hybrid algorithms may be designed; it is simply the case that CF methods become *hybrid* when they are designed using more than a single classifier.

Preprocessing

The purpose of preprocessing is either to *modify* or *partition* the raw data, in order to apply one of the above classification algorithms. Two widely used approaches are *normalising* user ratings (as discussed in Section 2.2.2), and *clustering* the data into smaller groups. The former tends to transform the integer ratings into a set of residuals, by subtracting user biases [Pot08, BK07]; these account for the different ways that users interpret the rating scale (e.g., some users consistently rate higher than others). There is a variety of techniques available for the latter purpose, including k-means, hierarchical, and density based clustering [AJOP09, JMF99]. For example, Rashid *et al* [RLKR06] proposed a filtering algorithm suitable for extremely large datasets that combines a clustering algorithm with the kNN prediction method. The aim was to cluster similar users together first, in order to overcome the costly operation of measuring the similarity between all user pairs in the system, and then apply a nearest-neighbour technique to make predictions. Much like the work presented by Li and Kim [LK03], clustering methods can be implemented to replace the “neighbourhood formation” step of the kNN approach. The Yoda system, designed by Shahabi *et al* [SBKCM01], is an example of a system that performs similar functions: clustering is implemented to address the scalability issues that arise as the community of users and available items grows.

Similarly, a sequence of classification algorithms can be designed, where the output of each method is the input to the next; each algorithm preprocesses the data for the subsequent method. For example, the rating matrix can be factorised first and each user's k-nearest neighbours can then be computed using the feature matrix, rather than the raw data itself [Kor08]. The aim of this is to compute neighbours using the *dense* feature matrix, rather than the extremely sparse rating data—-the hope being that more reliable neighbours can be found this way.

Ensemble Methods

The second set of hybrid techniques we review differentiate themselves from the above by focusing on combining output rather than refining it over sequential steps [Die00, Pol06]. In other words, classification algorithms are run independently of one another, and the output of each is then collapsed into a unified set of predictions and recommendations. There are two options here: hybrid-*switching* or *weighting*. Switching entails selecting individual predictions from each classifier, based on the assumption that some classifiers will be more accurate on a subset of instances than others [LAP09]. Weighting, on the other hand, combines the predictions of each classifier as a set of linear equations [ZWSP08]; weights are typically found by means of regression. First, the training data is further split into *training* and *probe* subsets. The algorithms are then given the training subset and queried with the probe instances.

Given a set $P = [a, b, ..., z]$ of predictors weighted by a vector $w = [w_a...w_z]$, a vector of probe ratings $r \in [r_1...r_n]$, and predictions $\hat{r}_{x,n}$ by classifier x for item n, predictions of each probe rating can be formulated as a linear combination of each classifier's prediction:

$$\hat{r}_n = \sum_{c \in P} (\hat{r}_{c,n} \times w_c) \tag{2.11}$$

If the set of predictions from each classifier is combined into a single matrix X, the problem of finding a classifier weight vector can be expressed in matrix form:

$$\begin{bmatrix} \hat{r}_{a,1} & \hat{r}_{b,1} & ... & \hat{r}_{z,1} \\ \hat{r}_{a,2} & \hat{r}_{b,2} & ... & \hat{r}_{z,2} \\ ... & ... & ... & ... \\ \hat{r}_{a,n} & \hat{r}_{b,n} & ... & \hat{r}_{z,n} \end{bmatrix} \begin{bmatrix} w_a \\ w_b \\ ... \\ w_z \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ ... \\ r_n \end{bmatrix} \tag{2.12}$$

The idea is to minimise the error between the weighted predictions matrix X and the vector of ratings r by simultaneously solving a set of linear equations. To solve for the weights w, each side of the equation must be multiplied by the inverse X^T of X:

$$\left(X^T X\right) w = \left(X^T r\right) \tag{2.13}$$

The same weights that have been learned are then applied to the full training set in order to predict any unrated items.

There are other techniques available for blending, based on gradient boosted decision trees [Kor09b], neural networks [PC09], and kernel ridge regression [TJB09]; it is beyond the scope of this thesis to cover all of these in detail. In the context of the Netflix prize, hundreds of predictors were blended together to produce the final solution. However, this is where the border between using CF to solve a *prediction* problem or build a recommender system lies. In fact, the chief product officer at Netflix stated[1] in an interview that:

[1] $http://www.appscout.com/2009/09/netflix\_1m\_prize\_winners\_inclu.php$

> “There are several hundred algorithms that contribute to the overall 10 percent improvement - all blended together,” Hunt said.“In order to make the computation feasible to generate the kinds of volumes of predictions that we needed for a real system we’ve selected just a small number, two or three, of those algorithms for direct implementation.”

2.2.6 Online Algorithms

All of the above algorithms share a common trait: they are all trained and queried offline; thus reinforcing why system administrators need to iteratively retrain their system in order to include the latest ratings. However, there is a separate class of techniques—online algorithms—that dynamically update as ratings are introduced into the system. Online algorithms are usually decomposed into four steps: (a) receiving an instance to predict (e.g., a user-movie pair), (b) predicting the value (rating) of that instance, (c) receiving the true rating input by the user, and finally (d) updating itself according to some predetermined loss function. There are a number of examples where online algorithms have been applied to collaborative filtering contexts. These range from slope one approaches [LM05] to online matrix factorisation feature update [RST08].

One of the main challenges facing online algorithms is that the way they work is not a perfect match with how recommender systems work. First, algorithms are not given a *single* instance to predict at any time; they are given a (large) *set* of instances—all the items that a particular user has not rated. Predictions of this set will then be used to generate a ranked list of recommendations. They will also only receive feedback on a subset of these predictions; there are no guarantees governing if and when they will receive the true rating for an item, and how the update will be biased by the predictions that it has made itself. Lastly, since items need to be ranked, the algorithm will need to be queried prior to the user interacting with the system; it may also be excessive to re-predict all instances when the user inputs a single rating. In fact, at this point it becomes difficult to distinguish between how online algorithms will be used differently to the offline ones explored above.

2.2.7 From Prediction to Recommendation

Once predicted ratings have been generated for the items, and sorted according to predicted value, the top-N items can be proposed to the end user as recommendations. This step completes the process followed by recommender systems, which can now elicit feedback from the user. User profiles will grow, and the recommender system can begin cycling through the process again: re-computing user similarity measures, predicting ratings, and offering recommendations.

It is important to note that the user interface of the system plays a vital role in this last step. The interface not only determines the ability the system has to present generated recommendations to the end user in a clear, transparent way, but will also have an effect on the response that the user gives to received recommendations. Wu and Huberman [WH07b] conducted a study investigating the temporal evolution of opinions of products posted on the web. They concluded that if the aggregate rating of an item is visible to users and the cost of expressing opinions for users is low (e.g. one click of a mouse), users will tend to express either neutral ratings or reinforce the view set by previous ratings. On the other hand, if the cost is high (such as requiring users to write a full review), users tended to offer opinions when they felt they could offset the current trend. Changing the visibility of information and the cost imposed on users to express their opinions, both determined by the interface provided to end users, will thus change the rating trend of the content, and the data that feeds into the filtering algorithm.

2.3 Trust and User Modelling

The previous section highlighted the data mining approaches to CF; what follows is a review of state of the art research that aims to augment these techniques by incorporating *user models* into collaborative filtering. In particular, we focus on the use of *trust* in recommender systems. However, before we proceed, we explore trust itself: what is trust? How has it been formalised as a computational concept?

A wide range of research [ARH98, JIB07, AH08] stems from sociologist Gambetta's definition of trust [Gam90]. Gambetta states:

> "trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action"

Trust is described as the level of belief established between two entities in a given context. Discussing trust as a probability paved the way for computational models of trust to be developed, as first explored by Marsh [Mar94] and subsequently by a wide range of researchers [Gol08]. The underlying assumption of trust models is that users' (or agents', peers', etc) historical behaviour is representative of how they will act in the future: much like CF, the common theme is one of *learning*. The differences between the two emerges from the stance they adopt toward their target scenarios; unlike CF, trust models are often adopted as a control mechanism (by, for example, rewarding good behaviour in commerce sites with reputation credit) and are user-centred techniques that are both aware and responsive to the particular characteristics desired of the system (such as, in the previous example, reliable online trade).

Trust models have been applied to a wide range of contexts, ranging from online reputation systems (e.g. eBay.com) to dynamic networks [CNS03] and mobile environments [QHC06]; a survey of trust in online service provision can be found in [JIB07]. Due to its widespread use, trust modelling may draw strong criticism with regards to its name: it is arguable that, in many of these contexts, "trust" is a vague synonym of "reliability," "competence," "predictability," or "security." However, encapsulating these scenarios under the guise of trust emphasises the common themes that flow between them; namely, that researchers are developing mechanisms for users to operate in computational environments that mimic the way humans interact with each other outside of the realm of information technology.

2.3.1 Motivating Trust in Recommender Systems

The motivations for using a notion of trust in collaborative filtering can be grouped into three categories:

1. In order to accommodate the **explainability** required by system users. Both Tintarev [TM07] and Herlocker *et al* [HKR00] discuss the effect explainability has on users' perceptions of the recommendations that they receive, especially those recommendations that are significantly irrelevant or disliked by the users. Chen and Pu [PC06] further investigate this issue by building explanation interfaces that are linked to, and based on, a formal model of trust. Although a major compo-

nent of these works revolve around presenting information to the end users, they recognise that building an explainable algorithm is a key component of transparency: it converts a “black-box” recommendation engine into something to which users can relate.

2. To address **data sparsity**. The volume of missing data has a two-fold implication. First, new users cannot be recommended items until the system has elicited preferences from them [RAC$^+$02]. Even when ratings are present, each of a pair of users who may actually share common *interests* will never be cited in each other’s neighbourhood unless they share *ratings* for items: information cannot be propagated beyond each user’s neighbourhood. Second, computed similarity will be incomplete, uncertain, and potentially unreliable.
3. To **improve the robustness** of CF to malicious attacks. Since recommender systems are often deployed in an e-commerce environment, there are many parties who may be interested in trying to exploit the system for their benefit, using what are known as *shilling* attacks [MBW07]. From the point of view of the ratings themselves, it is difficult to differentiate between what was input by honest users and the ratings that have been added in order to perform an attack. Trust models come to the rescue: by augmenting traditional collaborative filtering with a notion of how users interact, the robustness of recommender systems can be improved.

A consequence of incorporating trust models into CF is also often a measurable benefit in terms of prediction accuracy; however, state of the art algorithms that are *only* tuned for accuracy [BK07] do not mention trust models at all.

2.3.2 Using Trust For Neighbour Selection

One of the central roles that trust modelling has served in CF is to address the problem of *neighbour selection*, by replacing the kNN *neighbourhood formation* step above. Traditional approaches to CF are based on populating users’ kNN neighbourhood with others who share the highest measurable amount of similarity with them [HKBR99]. However, these methods do not guarantee that the right neighbours will be selected; the aim of using trust is thus to capture information that resides outside of each user’s local *similarity* neighbourhood in a transparent, robust and accurate way.

Two main approaches have been adopted: *implicit* methods, which aim to infer trust values between users based on item ratings, and *explicit* methods, that draw trust values from pre-established (or manually input) social links between users. Both methods share a common vision: the underlying relationships (whether inferred or pre-existing) can be described and reasoned upon in a web of trust, a graph where users are nodes and the links are weighted according to the extent that users trust each other.

Computing Implicit Trust

The first perspective of trust in CF considers values that can be inferred from the rating data: a web of trust between users is built from how each user has rated the system’s content. In these cases, trust is used to denote *predictability* and to allow the different ways that users interact with the recommender system; in fact, many of these measures build upon error measures, such as the mean absolute error.

For example, Pitsilis and Marshall focus on deriving trust by measuring the uncertainty that similarity computations include [PM04, PM05]. To do so, they quantify the uncertainty $u(a,b)$ between users a and b, which is computed as the average absolute difference of the ratings in the intersection of the two user's profiles. The authors scale each difference by dividing it by the maximum possible rating, $max(r)$:

$$u(a,b) = \frac{1}{|R_a \cap R_b|} \sum_{i \in (R_a \cap R_b)} \left(\frac{|r_{a,i} - r_{b,i}|}{max(r)} \right) \tag{2.14}$$

The authors then use this uncertainty measure in conjunction with the Pearson correlation coefficient to quantify how much a user should *believe* another. In other words, trust is used to scale similarity, rather than replace it. Similarly, O'Donovan and Smyth define a trust metric based on the recommendation error generated if a single user were to predict the ratings of another [OS05, OS06]. The authors first define a rating's correctness as a binary function. A rating $r_{b,i}$ is *correct* relative to a target user's rating $r_{a,i}$ if the absolute difference between the two falls below a threshold ϵ:

$$correct(r_{a,i}, r_{b,i}) \iff |r_{a,i} - r_{b,i}| \leq \epsilon \tag{2.15}$$

The notion of correctness has two applications. The first is at the *profile* level, $Trust^P$: the amount of trust that user a bestows on another user b is equivalent to the proportion of times that b generates correct recommendations. Formally, if $RecSet(b)$ represents the set of b's ratings used to generate recommendations, and $CorrectSet(b)$ is the number of those ratings that are *correct*, then profile-level trust is computed as:

$$Trust^P(b) = \frac{|CorrectSet(b)|}{|RecSet(b)|} \tag{2.16}$$

The second application of Equation 2.15 is item-level trust, $Trust^I$; this maps to the reputation a user carries as being a good predictor for item i, and is a finer-grained form of Equation 2.16, as discussed in [OS05]. Both applications rely on an appropriate value of ϵ: setting it too low hinders the formation of trust, while setting it too high will give the same amount of trust to neighbours who co-rate items with the current user, regardless of how the items are rated (since *correct* is a binary function). Similar to Pitsilis and Marshall, this metric also operates on the intersection of user profiles, and does not consider what has not been rated when computing trust.

In [LHC08c], the authors approach trust inference from a similar perspective, but extend it from a binary to continuous scale and include ratings that fall outside of the profile intersection of a user pair. Rather than quantifying the correctness of a neighbour's rating, the *value* that b's rating of item i would have provided to a's prediction, based on a's rating:

$$value(a,b,i) = 1 - \rho|r_{a,i} - r_{b,i}| \tag{2.17}$$

This equation returns 1 if the two ratings are the same, and 0 if user b has not rated item i; otherwise, its value depends on the *penalising* factor $\rho \in [0,1]$. The role of the penalising factor is to moderate the extent to which large differences between input ratings are punished; even though the two ratings may diverge, they share the common feature of having been input to the system, which is nevertheless relevant in sparse environments such as CF. A low penalising factor will therefore have the effect of populating neighbourhoods with profiles that are very similar in terms of what was rated, whereas a high penalising factor places the emphasis on how items are rated. In [LHC08c], the authors use $\rho = \frac{1}{5}$. The trust between two users is computed as the average value b's ratings provide to a:

$$trust(a,b) = \frac{1}{|R_a|}\left(\sum_{i \in R_a} value(a,b,i)\right) \tag{2.18}$$

This trust metric differs from that of O'Donovan and Smyth by being a pairwise measure, focusing on the value that user b gives to user a. Unlike the measures explored above, the value sum is divided by the size of the target user's profile, $|R_a|$, which is greater than or equal to the size of the pair's profile intersection, $|R_a \cap R_b|$, depending on whether a has rated more or fewer items than b. This affects the trust that can be awarded to those who have the sparsest profiles: it becomes impossible for a user who rates a lot of content to trust highly those who do not, while not preventing the inverse from happening.

The three methods we have presented here are not the only proposals for trust inference between users in CF contexts. For example, Weng *et al* [WMG06] liken the CF web of trust structure to a distributed peer-to-peer network overlay and describe a model that updates trust accordingly. Hwang and Chen [HC07] proposed another model that again marries trust and similarity values, taking advantage of both trust propagation and local similarity neighbourhoods. Papagelis *et al* [PPK05] do not differentiate between similarity and trust, by defining the trust between a pair of users as the correlation their profiles share; they then apply a propagation scheme in order to extend user neighbourhoods.

Many of the problems of computed trust values are akin to those of similarity; for example, it is difficult to set a neighbourhood for a new user who has not rated any items [RAC$^+$02]. However, the characteristics of trust modelling allow for solutions that would not emerge from similarity-centric CF. For example, [LHC08c] includes a constant *bootstrapping* value β for users who have rated no items, which translates to initial recommendations that are based on popularity, and would become more personalised as the user inputs ratings.

All of the methods we have explored share the common theme of using error between profiles as an indication of trust. Similarly, there is a broad literature on similarity estimation that does not adopt the language of trust modelling, such as the "horting" approach by Aggarwal *et al* [AWWY99] and the probabilistic approach by Blanzieri and Ricci [BR99]. In all of the above, each user pair is evaluated independently; the significant differences appear in how each method reflects an underlying user model of trust.

Selecting Neighbours Explicitly

The alternative to computing trust values between users is to transfer pre-existing social ties to the recommender system. There are two approaches that have been followed here: on the one hand, users may be asked explicitly to select trustworthy neighbours. On the other hand, social ties may be drawn from online social networks where it is possible to identify each user's friends.

Massa and Avesani describe trust-*aware* recommender systems [MB04, MA07]. In this scenario, users are asked to rate both items and *other users*. Doing so paves the way to the construction of an explicit web of trust between the system users. Since users cannot rate a significant portion of the other users, the problem of sparsity remains. However, assuming that user-input trust ratings for other users are more reliable than computed values, trust can then be propagated to broaden each user's neighbourhood.

Trust propagation is a highly *explainable* process: if a trusts b, and b trusts c, then it is likely that a will trust c. However, this transparency is obscured as the propagation extends beyond a two-hop relationship. The validity of propagation rests on the assumption that trust is transitive, an assumption that can be challenged once the propagation extends beyond "reasonable" limits. In small-world scenarios (such as social networks), this limit is likely to be less than the famed six-degrees of separation, since it is apparent that people do not trust everyone else in an entire social network. Much like similarity and computed trust, the efficiency of trust propagation is therefore dependent on the method used and the characteristics of the underlying data.

A range of other works centre their focus on the social aspect of recommendations. For example, Bonhard and Sasse [Bon04, PBH07] perform a series of experiments that analyse users' perception of recommendations: they conclude that participants overwhelmingly prefer recommendations from familiar (as opposed to similar) recommenders. The experiments reflect the ongoing asymmetry between algorithmic approaches to CF, which tend to focus on predictive accuracy, and user studies that mainly consider recommender system interfaces. It is difficult to evaluate one independently of the other, and Bonhard's motivations for the use of social networks echo those used to motivate the use of trust models in Section 2.3.1: in order to reconcile the end users' mental model of the system and the system's model of the users.

Golbeck explored the power of social networking in the FilmTrust system [Gol06], showing that these systems produce comparable accuracy to similarity-based CF. The application of social networks can also be beneficial to CF since relationships in the web of trust can be augmented from simple weighted links to annotated, contextual relationships (i.e., b is my sister, c is my friend). Context-aware recommender systems is a nascent research area; Amodavicius *et al* [ASST05] provide a first view into this subject by looking at multi dimensional rating models. Full coverage of this falls beyond the scope of this chapter; however, it is apparent how network ties can be fed into mechanisms that include who and where the users are before providing recommendations.

The main criticism of many of these approaches is that they require additional explicit input from the end user; in effect, they move against the fully automated view of recommender systems that original collaborative filtering proposed. However, social networks are on the rise, and users proactively dedicate a significant portion of time to social networking. The implementation of these methods therefore aims to harness the information that users input in order to serve them better.

It is important to note that both the computed and explicit methods of finding trustworthy neighbours are not in conflict; in fact, they can be implemented side by side. Both require users to be rating items in order to provide recommendations, while the latter also requires social structure. Popular social networking sites, such as Facebook[2] include a plethora of applications for which users are requested to rate items, making the conjunction of the two methods ever easier.

[2] http://www.facebook.com/apps/

2.3.3 Trust-Based Collaborative Filtering

Once neighbours have been chosen, content can be filtered. However, there are a range of choices available to do so; in this section we outline the methods implemented by the researchers we discussed in the previous section. The approaches revolve around rating aggregations; in other words, taking a set of neighbour ratings for an item and predicting the user's rating using Equation 2.8. The difference between each method is (a) what neighbours are selected, and (b) how the ratings from each neighbour are weighted. We split the methods into three strategies, trust-based *filtering*, *weighting*, and social filtering.

1. **Trust-Based Filtering**. In this case, neighbours are selected (filtered) using computed trust values. The ratings they contribute are then weighted according to how similar they are with the target user.
2. **Trust-Based Weighting** departs fully from similarity-based CF: neighbours are both selected and their contributions weighted according to the trust they share with the target user.
3. **Social Filtering**. Neighbours are selected based on the social ties they share with the target user. Ratings can then be weighted according to either their shared similarity or trust with the target user.

All of these methods assume that users will be using the rating scales symmetrically, i.e. if two users predict each other perfectly, then the difference $(r_{a,i} - \bar{r}_a)$ will be the same as $(r_{b,i} - \bar{r}_b)$, regardless of what each user's mean rating actually is. In practice, this is not always the case: predictions often need to be changed to fit the rating scale, since users each use this scale differently. This notion was first explored in the aforementioned work by Aggarwal *et al* [AWWY99], who aimed to find a linear mapping between different users' ratings. However, [LHC08c] extends this notion to encompass what they refer to as *semantic distance*, by learning a non-linear mapping between user profiles based on the rating contingency table between the two profiles. The results offer accuracy benefits in the MovieLens dataset, but do not hold in all cases: translating from one rating scheme to another is thus another research area that has yet to be fully explored.

The above work further assumes that the underlying classifier is a kNN algorithm. Recent work, however, has been moving away from kNN-based recommender systems. In fact, the data derived from users telling the system whom they trust can also be input into other algorithms, such as matrix factorisation techniques [MYLK08, MKL09]. In these works, Ma *et al* describe matrix factorisation models that account for both what users rate (their preferences) and to whom they explicitly connect (who they trust). While certainly beneficial to cold-start users, introducing trust data into factorisation models reignites the problem of *transparency*: how will users understand how their input trust values contribute to their recommendations? A potential avenue for research lies in the effect that hybrid trust models have on users. For example, Koren describes how a neighbourhood and factorisation model can be combined [Kor08], and this work may begin to bridge the chasm between the factorisation-based and kNN-based use of trust in recommender systems.

2.4 Evaluating Recommendations

In order to design and evaluate CF algorithms, researchers require three components: (a) a *dataset* of user ratings to which to apply the algorithm, (b) a *methodology* to conduct repeatable experiments, and (c) appropriate *metrics* to measure system performance. In this section, we review each of these components.

2.4.1 Rating Datasets

The rise of Web 2.0 sites equipped with developer application programming interfaces (APIs) to access user data is improving both the *ease of access* and *wealth of data* available to CF research. Dell'Amico and Capra [DC08] are but one example of researchers who have collected and experimented with a dataset of music preferences from Last.fm[3]. Amatriain *et al* [ALP$^+$09] gathered movie rating datasets from Rotten Tomatoes[4] and Flixster[5]. However, requiring researchers to each crawl the web for a dataset is not only an unnecessary overhead, but does not allow for results to be compared between different researchers' experiments if the dataset is not shared. There is also a number of pre-packaged rating datasets. In this thesis we focus on two; the largest and most widely studied datasets of ratings:

- **Netflix Prize Dataset**: The largest of the available datasets, this set of movie ratings consists of 100 million ratings, by $480,189$ users who have rated one or more of the $17,770$ movies on a $1-5$ star scale.
- **MovieLens Dataset**: There are currently three movie-rating datasets available from the GroupLens website[6]. The first includes $100,000$ ratings of $1,682$ movies by 943 users; the second has 1 million ratings for $3,900$ movies by $6,040$ users; the last includes 10 million ratings and $100,000$ tags for $10,681$ movies by $71,567$ users. As with the Netflix dataset, items are rated on a $1-5$ star scale. In this thesis, we have used the first two datasets since the third dataset was only released in 2009.

Other available datasets include the *Jester* joke-rating dataset, and the *Book-Crossing* dataset of book ratings. This is by no means an exhaustive list of datasets; for example, Github[7] (an online collection of source code repositories) released a dataset of the repositories "watched" by each of its members, and ran a competition that also aimed at designing an algorithm to recommend repositories to its users.

2.4.2 Methodology

The aim of an experiment with user data is to manipulate the ratings in a way that (a) reflects assumptions of how the users will interact with the end system, and (b) produces measurable results that mirror the users' experience.

The assumptions that researchers make with regards to the former goal are twofold: first, that users will have already provided *at least* one rating to the system. Given that only the ratings are being

[3] http://www.last.fm

[4] http://www.rottentomatoes.com

[5] http://www.flixster.com

[6] http://www.grouplens.org

[7] http://contest.github.com

used to generate recommendations (demographic or item-attribute data is unavailable), then lack of any rating data does not allow CF algorithms to produce personalised results. Second, if a user's preferences were already known, then ranking items according to the user's ratings would provide the most useful recommendations. Therefore, the main focus of CF evaluation has historically been that of *predicting* the ratings of items that users have not rated.

To evaluate how well an algorithm is accomplishing the task of providing recommendations, researchers use one of the available rating datasets. The dataset is first partitioned into two subsets; the first acts as a *training* set that will be available for the algorithm to learn from. The second subset is the *test* set, with rating values that remain hidden to the algorithm. An evaluation will query the algorithm to make predictions on all the items in the test set. Results are then cross-validated by repeating experiments on multiple partitionings of the data. Hidden in these steps is an assumption that the performance of a *deployed* algorithm (with an online recommender system) will be comparable to that of the selected training and test sets.

However, one of the fundamental problems with this methodology is related to the second goal: the extent to which each user's *qualitative* experience with the system can be translated into measurable, *quantitative* results is questionable. In effect, researchers reduce the problem of generating interesting, serendipitous, and useful recommendations into one of accurate preference prediction. In the following section, we review the metrics used for this task and discuss the controversy surrounding their use.

2.4.3 Metrics

The most widely adopted metrics used to evaluate the efficacy of CF algorithms deal with prediction accuracy. Available measures of statistical accuracy include the mean absolute error (MAE) and the root mean squared error (RMSE):

$$MAE = \frac{\sum_N |r_{a,i} - \hat{r}_{a,i}|}{N} \tag{2.19}$$

$$RMSE = \sqrt{\frac{\sum_N (r_{a,i} - \hat{r}_{a,i})^2}{N}} \tag{2.20}$$

Both of the above measures focus on the difference between a rating of item i by user a, $r_{a,i}$, and the prediction for the same user and item, $\hat{r}_{a,i}$. In general, both metrics measure the same thing and will thus behave similarly; the difference lies in the degree to which different mistakes are penalised. There are a number of modified mean error metrics that aim to introduce fairer representations of the system users. For example, Massa uses the mean absolute *user* error in order to compensate for the fact that more predictions are often made for some users rather than others [MA07] and O'Donovan and Smyth compare algorithm performance by looking at how often one is more accurate than another [OS05].

Coverage metrics complement accuracy results: they aim to quantify the breadth of predictions that are possible using a given method. They compare the proportion of the dataset that is uncovered to the size of the test set, in order to measure the extent that predictions were made possible using the current algorithm and parameters.

While prediction accuracy continues to receive significant attention, due to it being the metric of choice of the Netflix prize, the focus on accuracy in recommender research continues to be disputed.

While we already described the difficulty of translating qualitative experience into quantitative values, there are a number of further reasons for this:

- **Accuracy Metrics Focus on Predictions** that have been produced, regardless of whether the prediction was at all possible or not. Massa and Avesani [MA07] show that prediction accuracy continues to perform well when pre-defined values are returned. For example, if each prediction simply returned the current user mean (thus not allowing content to be ranked and converted into recommendations), accuracy metrics would still not reflect such poor behaviour.
- **Test Set Items Have Already Been Rated**. An evaluation method that only makes predictions on items in the test set (items that the user has rated) may tend to show good performance, especially if there is low variance in the way users rate. Real systems, that have to provide recommendations based on making predictions on *all* unrated items, may have much worse performance.
- **Predictions vs. Recommendations**. McLaughlin and Herlocker [MH04] argue that striving for low mean errors biases recommender systems towards good *predictors* rather than *recommenders*. In other words, an error in a prediction affects the mean error the same way, regardless of whether the prediction enabled the entry to qualify as a recommendation or not.

Mean errors will therefore not tend to reflect the end user experience. Concerns over accuracy-centric research continues; McNee *et al* [MRK06] even argued that striving for accuracy is detrimental to recommender system research, and propose that evaluations should revert to user-centric methods. Accuracy metrics persist, however, due to the need for empirical evaluations of filtering algorithms that can compare the relative performance of different techniques without including the subjective views of a limited (and, more often than not, inaccessible) group of test subjects. Furthermore, while it remains difficult to understand exactly how accurate predictions translate into usefully ranked recommendations, accuracy is a useful metric for understanding the extent to which an algorithm is approximating the preference values input by the system users.

Examining an algorithm from the point of view of top-N recommendations provides an alternative means of evaluation; rather than considering the predictions themselves, information retrieval metrics (i.e., precision and recall) are used on a list of items sorted using the predictions. However, sorting a list of recommendations often relies on more than predicted ratings: for example, how should one sort two items that both have the highest possible rating predicted? How large should the list size N be? These kinds of design decisions affect the items that appear in top-N lists, and has motivated some to change from deterministic recommendation lists to looking at whether items are “recommendable” (i.e., their prediction falls over a predefined threshold) or not [ALP$^+$09].

Other error measures have been applied when analyzing the accuracy of a filtering algorithm, including receiver-operating characteristic (ROC) sensitivity [HKBR99]. This metric aims at measuring how effectively predicted ratings helped a user select high-quality items. Recommendations are therefore reduced to a binary decision: either the user “consumed” the content (i.e., watched the movie, listened to the song, read the article) and rated it, or did not. By comparing the number of false-positives, or items that should have been recommended that were not, and false-negatives, or not recommending an

Source	Problems	Example Resources
Data	Sparsity Noise Cold-Start Privacy Popularity Bias	[MKL07] [APO09, HKV08] [NDB07, PPM$^+$06, RAC$^+$02] [LHC07, Can02, BEKR07] [CC08]
Algorithm	Accuracy Latency Scalability	[BK07] [GRGP00] [BK07]
System	Robustness Distributed	[MBW07, LR04] [Zie05]
User	Explainability Context Implicit Trust Explicit Trust	[HKR00, PC06] [ASST05] [MA07] [Bon04, Gol06]
Evaluation	Metrics Accuracy User Experience	[HKTR04, BHK98] [MRK06] [MH04]

Table 2.1: A Sample of Open Problems in Recommender Systems

item that should have been, this metric aims at measuring the extent to which the recommender system is helping users making good decisions. However, this method relies on a prediction score threshold that determines whether the item was recommended or not, which often does not translate to the way that users are presented with recommendations. A comprehensive review of metrics used when evaluating CF can be found in [HKTR04].

2.5 Open Problems

There is a wide variety of problems that state of the art recommender systems face. These range from *user-centric issues* (poor recommendation explainability, users distrusting recommendations), to *data-related problems* (namely, the adverse consequences of data sparsity), *algorithm-related challenges* (latency, scalability, accuracy), *system-wide weaknesses* (vulnerability to attack), and *evaluation-related issues* (or, how to best evaluate CF). A summary of these open issues, along with pointers to relevant research in each direction, is provided in Table 2.1. Each category that we have enumerated, however, is intertwined with the others: for example, *data* cold-start issues are related to *user* preference elicitation, and affect the *algorithm*'s accuracy. This section enumerates the open research problems pertaining to recommender systems that we addressed in this thesis.

2.5.1 Ratings: Changing Over Time

Given the number of users and items in a typical recommender system, the rating matrix tends to be extremely sparse. The problem of data sparsity highlights the dependence that filtering algorithms have on the altruism of the community of users making use of the recommender system; if users do not rate items then the cyclical process of generating recommendations cannot be completed. Both current state of the art algorithms and evaluation techniques are aware of this shortcoming; however, they both also view the matrix as a *static* data collection. In doing so, they assume that (a) changes in the data over time are not relevant when predicting user preferences, and (b) the current data is sufficient for computing appropriate recommendations (e.g., to find similar kNN neighbourhoods).

The relationship between the assumption of persisting like-mindedness and the reality of neighbourhood patterns over time in recommender systems remains largely unexplored. We thus explore ratings over time: delving into how datasets change over time and the different patterns of behaviour that emerge. We show that observing a snapshot of similarity between all the system users is both subjective to how the similarity is computed, unreliable, and difficult to differentiate from a random process. By modelling neighbourhood-based CF as a graph that is iteratively updated, we highlight a significant break between the assumptions and operation of CF. Analysis of the ratings over time lays the foundation for our proposed methodology that modifies traditional CF evaluation by including a notion of temporal updates.

2.5.2 Methodology & Evaluation

In Section 2.4.2, we recounted how CF algorithms are evaluated: datasets are split and algorithms are queried about a test set, after learning from the training set. Repeating this process provides a cross-validated snapshot of the expected performance of an algorithm when trained with a dataset with those particular characteristics. However, deployed recommender systems do not handle unchanging datasets: they are iteratively updated on *growing* sets of user ratings. Deployed systems need to cope with a growing set of both users and items; unfortunately, though, they are not evaluated as such. In Chapter 4 we propose a novel method to perform CF experiments that includes the notion of temporal updates. We extend current accuracy-based metrics onto the temporal dimension, and evaluate the performance of a series of CF algorithms over time. Lastly, we show that introducing an awareness of temporal updates into the algorithm's operation can offer improved temporal accuracy.

While accuracy remains a crucial focal point of recommender system evaluation, the controversy that surrounds its use reflects the fact that it only highlights a single dimension of performance: how close the predictions are to the user input values. There are a wide range of qualities that may be desired of a recommender system, such as diversity. The problem of diversity has already been explored by Smyth and McLave [SM01]. If a user rates an item (for example, rating an album by The Beatles), loading the user's recommendations with extremely similar items (i.e., all of the other albums by The Beatles) is often not helpful at all; the user has not been pointed towards new information, and is only inundated with recommendations towards content that is probably known already. The question therefore becomes: to what extent do filtering algorithms generate diverse recommendations? The temporal dimension to this problem considers the sequence of recommendations with which users are provided as they interact with the system: to what extent does the system compute the *same* recommendations over and over? In Chapter 5, we explore the temporal diversity of CF algorithms and the relationship they share with temporal accuracy. We then design and evaluate a hybrid algorithm that increases temporal diversity.

2.5.3 System Robustness

The last problem we consider here relates to *system vulnerabilities*. Attackers may aim to modify the recommendations that are output by a system for any number of selfish reasons. They may wish artificially to promote a piece of content, to demote content (perhaps since it competes with the attacker's content), or to target a specific audience of users. These attacks are aided by the near-anonymity of users participating in the recommender system. In some cases, signing up to an e-service that uses recommender system technology only requires an email address. Creating a number of fake accounts is thus not beyond the realms of possibility; furthermore, if each of these fake accounts is treated as an honest user, it becomes possible to change the output of recommender systems at will.

Research in the field of recommender system vulnerabilities can be divided into two categories. On the one hand, system administrators require a means of identifying attacks by being able to recognise both when an attack is occurring and which users are malicious. On the other hand, the vulnerabilities themselves are addressed; how can these attacks be prevented? How can the cost or effect be minimise? A comprehensive review of the vulnerabilities of collaborative recommender systems and their robustness to attack can be found in Mobasher *et al* [MBW07].

The growing body of literature addressing CF attacks takes a binary view of the rating matrix: either it has been attacked, or it has not. The former is created using the latter along with an appropriate attack model, and the objective of attack detection algorithms is to separate the honest and dishonest profiles. However, as recommender systems are updated over time, this scenario is unlikely to appear in deployed systems: sybils' profiles may be built *over time,* and may be able to inflict damage prior to being in the detectable state that they are when evaluated by researchers. In Chapter 6, we address this problem by designing and evaluating a series of monitors that alleviate the impact of these attacks by detecting behavioural shifts in the system that indicate ongoing anomalous activity.

2.6 Summary

In this chapter, we have introduced the data, algorithms, and open problems related to CF recommender systems. CF automates the process of generating recommendations by drawing from its assumption of like-mindedness between its users. In other words, people who have displayed a degree of similarity in the past will continue sharing the same tastes in the future. The model of users held by these systems therefore focuses on the set of preferences that each individual has expressed, and interactions between users can be determined according to values derived by operating on the information available in users' profiles.

The approaches themselves, however, originate from a wide variety of backgrounds, and include content-based methods, which infer recommendations from item attributes and machine-learning inspired solutions; we discussed baseline, kNN, matrix factorisation, and hybrid approaches. Nearest neighbour algorithms follow a three-stage process: finding a set of recommenders for each user based on a pre-defined measure of similarity, computing predicted ratings based on the input of these recommenders, and serving recommendations to the user, hoping that they will be accurate and useful suggestions. The choice of what method to implement relies on a fine balance between accuracy and performance, and is also dependent on the specific context that recommendations need to be made for. Each method has its own strengths and weaknesses, and hybrid methods attempt to reap the best of both worlds by combining a variety of methods.

The general problems faced by recommender systems remain the same, regardless of the approach used to build the filtering algorithm. These problems were grouped into a set of categories: problems originating from the data, algorithm, system, users, and evaluation. In this thesis, we tackle a fundamental issue underlying all approaches that have been proposed before: that is, how to model and evaluate a system that will be deployed over time.

Chapter 3

Temporal Analysis of Rating Datasets

The previous chapter highlighted an important problem with recommender systems: CF evaluation does not take into account that the data used to compute recommendations is subject to change over time. In this chapter, we analyse this phenomenon. We begin by introducing the rating datasets we use for this study in Section 3.1. We then split our analysis into two parts: in Section 3.2, we perform an in-depth analysis of the ratings that recommender systems receive. In Section 3.3, we investigate how these changes affect the *similarity* between users over time (and, consequently, the recommendations that CF computes). These observations lay the foundations of work we present in the following chapters; namely, how to use temporal information to improve the accuracy, augment the diversity, and secure the robustness of recommender systems.

3.1 Rating Datasets

We focus on three explicit-rating datasets: two MovieLens sets (which we refer to as ML-1 and ML-2) and the Netflix prize set. These datasets have been at the fulcrum of CF research for a number of years, and can be described as collections of 4-tuples:

$$[u, i, r_{u,i}, t_{u,i}] \tag{3.1}$$

Each tuple contains: a user id u, a movie id i, the rating $r_{u,i}$ given by the user to the movie, and the time $t_{u,i}$ when this rating was input. The motivation behind comparing these datasets extends beyond their popularity. They all provide users rating the same type of *content* (movies) with the same *scale* (1-5 stars)—thus allowing a direct comparison of each dataset's temporal rating characteristics. However, we still expect to identify temporal differences between the sets: there are significant differences in each set's size in terms of users, items, and ratings. We summarise these differences in Table 3.1. In addition to each set's relative size, there are a number of implicit reasons why temporal differences may emerge, including:

- **Motivation**: The MovieLens data is sampled from a system built for research purposes[1]. Netflix, instead, is a commercial system[2] incentivised by financial targets. The system users themselves

[1] http://www.movielens.org/login

[2] http://www.netflix.com/

Dataset	**Users**	**Movies**	**Ratings**	**Time (Days)**
MovieLens-1	943	1,682	100,000	215
MovieLens-2	6,040	3,706	1,000,209	1,036
Netflix	480,189	17,770	100,480,507	2,243

Table 3.1: Users, Items, Ratings in Each Dataset

have different relationships with each system: in the former, they are *contributing* to research by rating; in the latter, they are *customers* who are requesting and receiving DVDs when subscribed.

- **Interface**: We assume that users are more likely to rate content to which they are exposed; how and what people rate may thus be dependent on the design and usability of each system's interface.
- **Algorithm**: Similarly, we assume that there may be a relationship between what users are *recommended* and what they *rate*. The datasets therefore become subject to the CF algorithm that was in operation when the data was collected.

The rating data alone is not sufficient to understand which of these forces is at play; we are also unaware of any changes to which each system's interface or algorithm may have been subject to during the time span of ratings available. We cannot therefore explicitly discuss the causality of changes we observe in the data. This point is aggravated by the uncertainty as to whether time was taken into account when sampling each system's ratings. The ML-1 documentation states that the dataset has been "cleaned-up:" users with fewer than 20 ratings or incomplete demographic information were pruned from the set. However, there is no further mention of the subsampling technique used.

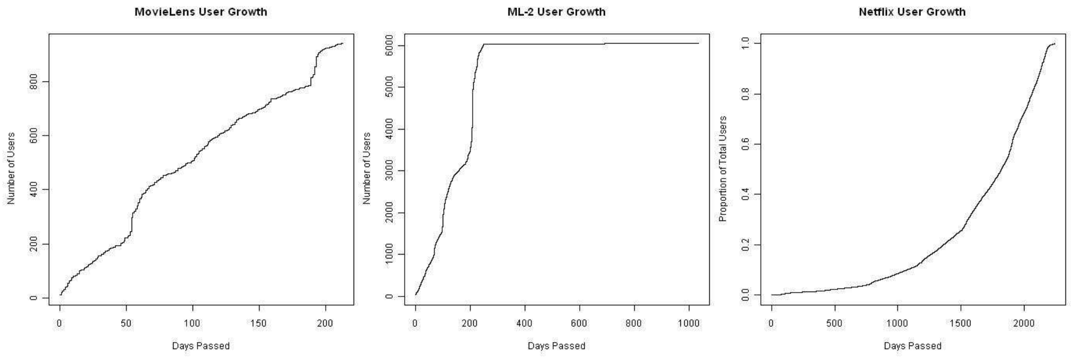
These uncertainties challenge the accuracy of hypotheses that have been verified using these datasets; in particular, it is difficult to claim that algorithms that yield improved accuracy on one of these sets will produce similar results once deployed. However, one point remains: recommender systems are *subject to change over time*, as new users join the system, new ratings are input, and new movies are released. The purpose of the following sections is to show that these changes occur, see how they are visible (in the available data), and examine their impact on conclusions drawn using current evaluative techniques that do not take them into account.

3.2 Ratings Over Time

We divide our analysis into four groups: we look at the growth of the number of users, items and ratings over time (Section 3.2.1), how this growth affects summary statistics derived from the ratings (Section 3.2.2), how user rating behaviour changes with time (Section 3.2.3), and the seasonal trends that emerge when users rate content (Section 3.2.4).

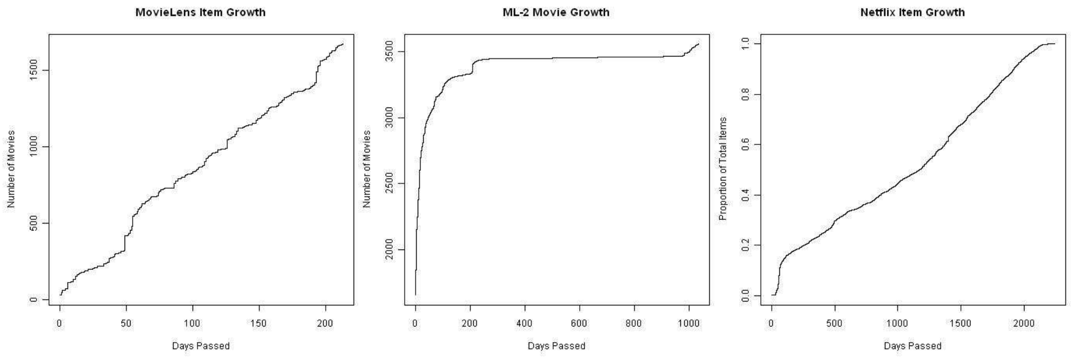
3.2.1 Dataset Growth

In Figures 3.1, 3.2 and 3.3 we visualise the cumulative growth of the number of users, movies, and total ratings over time for each dataset. In these plots we measure *daily* changes, since the Netflix timestamp data only reports the date that users input ratings. The MovieLens datasets' timestamps would allow for a finer grained analysis; however, we opt for daily views in order to consider all three sets simultaneously.



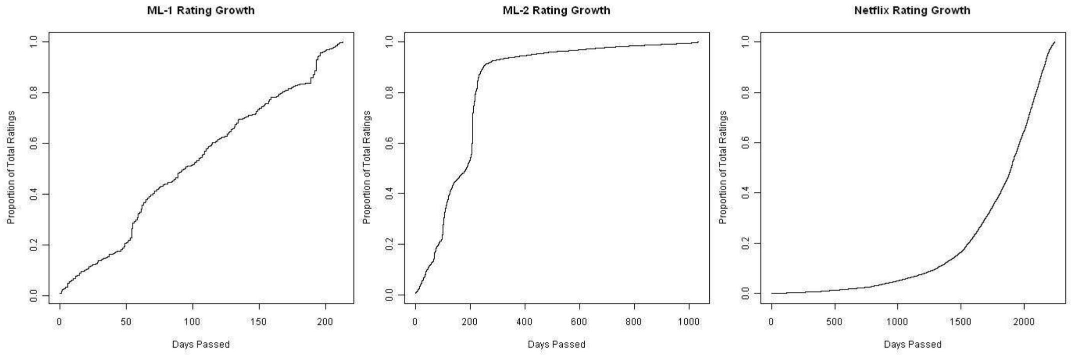
(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

Figure 3.1: Number of Users Over Time (ML-1, ML-2, Netflix)



(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

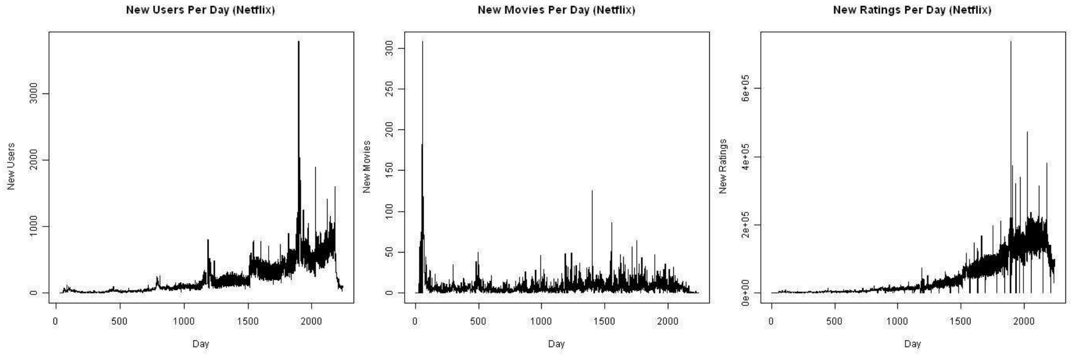
Figure 3.2: Number of Movies Over Time (ML-1, ML-2, Netflix)



(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

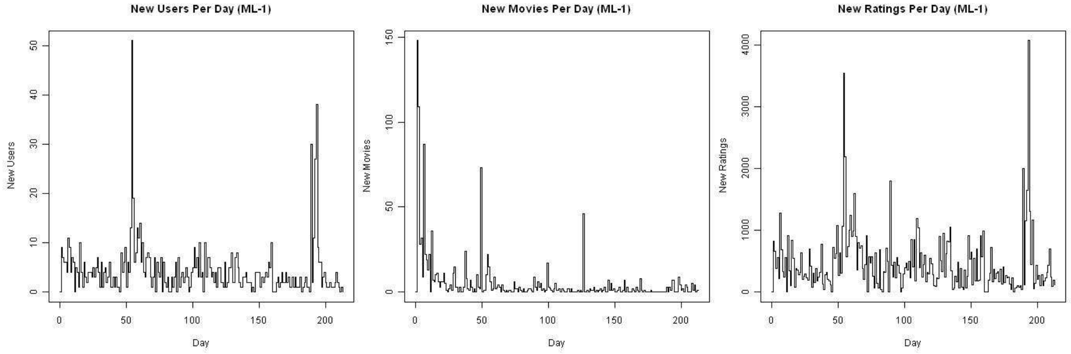
Figure 3.3: Number of Total Ratings Over Time (ML-1, ML-2, Netflix)

Since we do not have sign-up data, we consider that users “join” the system the moment they make their first rating. Similarly, a movie appears in the system when it is first rated, since we do not know when it was actually added to the movie database. We assume this to be a justifiable measure of dataset growth since CF algorithms (that do not include content information) can only compute predictions for movies that have been rated and users that have rated at least once.



(a) New Users Per Day (b) New Movies Per Day (c) New Ratings Per Day

Figure 3.4: Non-Cumulative Netflix Daily Growth: the spikes represent days when a lot of users/movies/ratings were added



(a) New Users Per Day (b) New Movies Per Day (c) New Ratings Per Day

Figure 3.5: Non-Cumulative ML-1 Daily Growth

Each dataset shows varying rates of growth. The number of Netflix users and ratings grow exponentially, while the movies appear in the system at a near-linear pace. The ML-1 set also displays near-linear growth: the number of users, items, and ratings continues to increase over each time step. The ML-2 dataset distinguishes itself from the other two by being the only one that shows a sharp change in growth over time. In fact, the majority of the users appear within the first half of the dataset; after this phase of accelerated growth, user growth halts and the rate at which new ratings and items are added to the system sharply declines. The ML-1 and Netflix sets, instead, do not exhibit this anomaly and continue to grow over time, but appear to do so at different rates.

One of the reasons for this apparent difference is the time that each dataset covers: the ML-1 set, ranging over 215 days, is less than one tenth of the time that the Netflix set ($2,243$ days) spans. In order to account for this difference, we examined how much each dataset grows per day. In Figures 3.4 and 3.5 we plot how many new users, movies, and ratings appear in each day of each dataset. From this perspective, the two datasets look more similar (differences between them may be explained by the relative size of each set). Both have peaks, where a large volume of users appears in the system. Similarly, both item plots (Figures 3.4(b) and 3.5(b)) spike in the early days of the dataset, when the

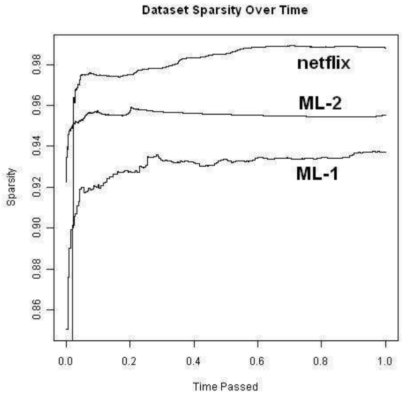
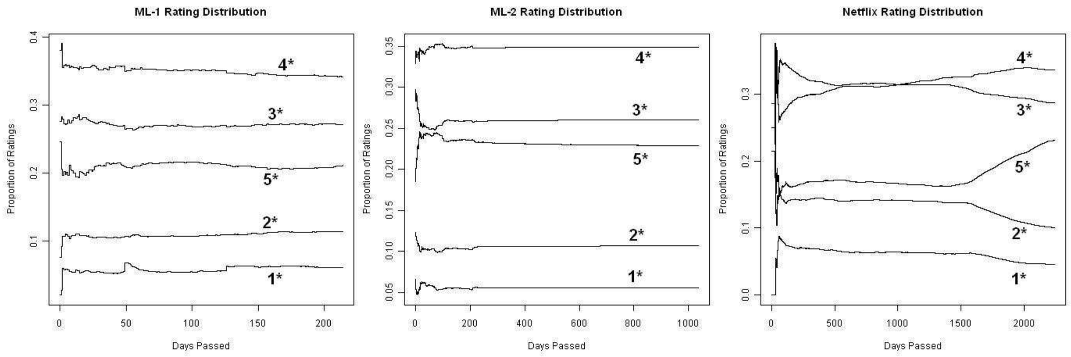


Figure 3.6: Sparsity Over Time For Each Dataset: Netflix is the most sparse dataset

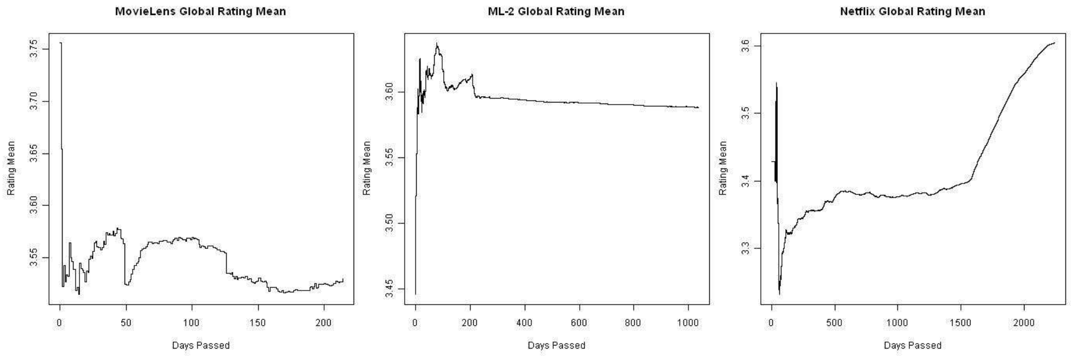
incoming ratings are going to items that have not been rated before. The Netflix data, however, continues to display accelerating growth: Figures 3.4(a) and 3.4(c) shows that the volume of incoming users and ratings tends to increase over time.

A changing volume of users, movies, and ratings will affect each dataset's sparsity and rating distribution. In Figure 3.6, we plot the sparsity over time after normalising the number of days in each dataset. All of the datasets are consistently over 90% sparse—less than 10% of the potential user-movie ratings exist—but the Netflix dataset remains the sparsest, with a maximum value near 99%. The ML-1 set, while being the smallest, is also the least sparse (potentially due to the pruning of users with fewer than 20 ratings). In Figure 3.7, we show how the rating distributions vary with time. If we consider the datasets in their entirety, the absolute ordering of ratings is equal throughout all datasets: there are more 4 star than 3-star ratings, more 3 stars than 5 star ratings, and a very small proportion of 1 and 2 star ratings. This seems to imply that people tend to rate what they already like, but tend to also avoid the "extreme" ratings (1 and 5 stars). However, the Netflix dataset's distribution (in Figure 3.7(c)) changes: in the early days of the dataset, there are more 3 stars than 4 stars. Roughly 1000 days into the dataset, the 4 star rating overtakes the 3 star rating. It seems that, at this point, users are responding more positively to their recommendations; in doing so, they shift the entire distribution of ratings towards the positive end. However, as we do not have data to know what recommendations users were given, we cannot empirically justify this claim. The main conclusion we make from these observations is that viewing rating sets from a static viewpoint does not account for the changes that real systems' data actually undergoes. In particular, user, item, and rating growth over time implies that the amount of information available to create recommendations (and thus the *value* that different users can draw from the system, and potential accuracy) at different times will be quite large. Many users, who rate items that have not been rated before, are not simply responding to recommendations but are proactively seeking to rate items, set rating trends, and respond to rating incentives [HJAK05, WH07a, BLW$^+$04].



(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

Figure 3.7: Rating Distribution Over Time Of Each Dataset: Netflix is the only dataset with no consistent ordering between the rating values



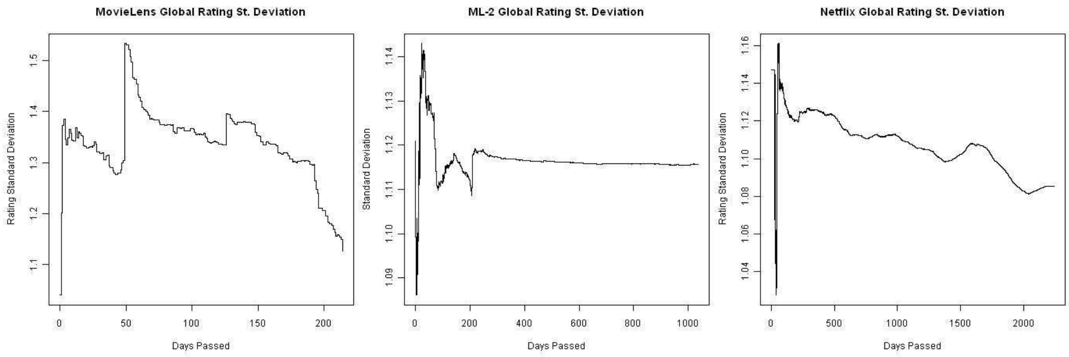
(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

Figure 3.8: Datasets' Global Rating Mean Over Time, Again highlighting the stop in ML-2's growth

3.2.2 Changing Summary Statistics

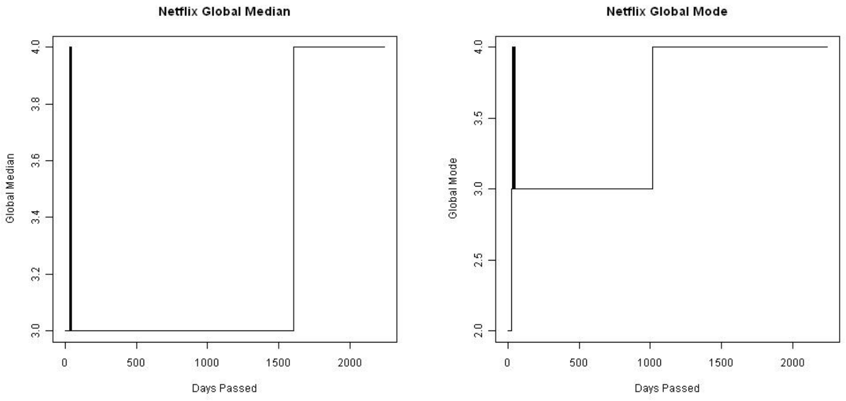
While new ratings are added, any summary statistics computed from the available data may fluctuate. In this section, we consider both *global* and *per-user* or *item* summary statistics. We begin with the global rating means, in Figure 3.8. The means are computed daily using the entire history of available ratings to date; we weight all ratings equally, regardless of when they were input (i.e., there is no time decay). All of the means consistently fall between 3 and 4 stars, but vary quite widely within this range. For the Netflix dataset, the most notable time segments are before the first 500 days, where the global mean rises sharply, falls, and then once again rises, and after the first $1,500$ days have passed, where the mean begins to grow again. The ML-2 set (Figure 3.8(b)) emphasises the relationship between *growth* and *change*: when the dataset stops receiving new users (as we saw in the previous section), its global mean stabilises as well.

The standard deviations are shown in Figure 3.9. The Netflix plot (Figure 3.9(c))—like its global mean—suffers from high fluctuation in the initial days of the dataset, and then decreases from 1.14 to 1.08 in a near-linear fashion. In other words, the ratings become less dispersed around the mean over time. Given that the mean is between 3 and 4 stars, this translates to a tendency to rate more



(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

Figure 3.9: Datasets' Global Rating Variance Over Time



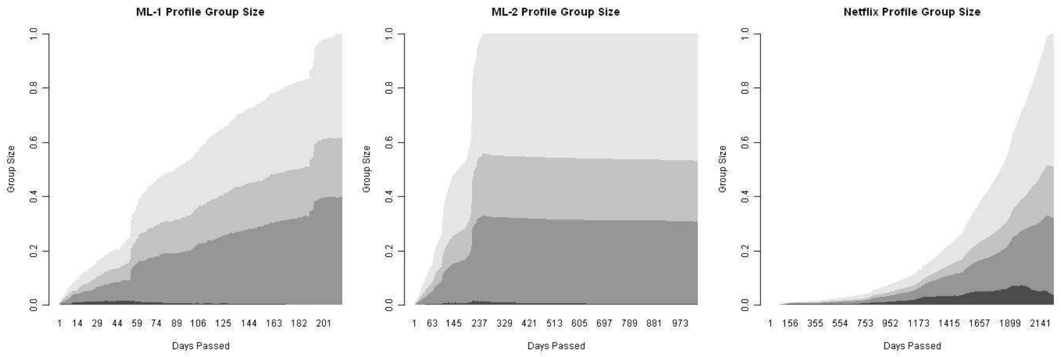
(a) Global Median (b) Global Mode

Figure 3.10: Netflix Rating Median and Mode Over Time

positively. Similarly, the ML-2 standard deviation stops changing when its mean flattens. The ML-1 dataset standard deviation is consistently higher than those we observed in the other datasets; however, excluding the edges of the 215 days, it remains relatively flat. The peak in the plot coincides with the dip in the temporal mean.

The problem here is that state of the art research does not factor in this feature of the data. For example, consider the BellKor solution to the Netflix prize competition [Kor09b]; the foundation of the ensemble of techniques they used successfully to win the competition was a *baseline* predictor, which includes the *global rating average*: a value that, as we have seen, will change over time. Although [Kor09b] does account for temporal changes at the user and item level (by binning the data into sequential windows of varying size), the global baseline prediction is used as a fixed starting value from which to build predictions. If we consider the range of values that this global mean takes over time, it seems that the accuracy of this baseline would vary significantly.

To understand why the mean and variance display such change, consider Figure 3.10, which shows the rating median and mode (i.e. the most frequent rating value) of the Netflix dataset over time. We do not plot the ML-1 and ML-2 temporal medians and modes, since they do not change: they all remain



(a) ML-1 Dataset

(b) ML-2 Dataset

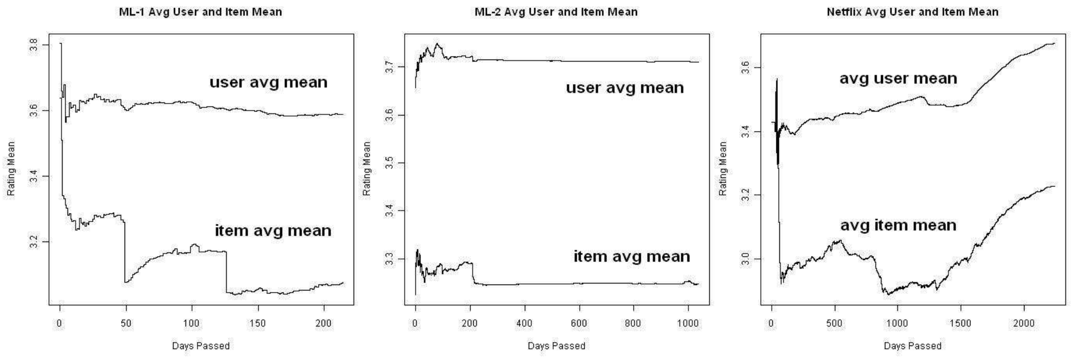
(c) Netflix Dataset

Figure 3.11: Users Binned By Profile Size Over Time

constant at 4 stars. The initial fluctuation in the Netflix mean is mirrored by a change of the rating mode from 2 to 4 stars. The mode then reverts and stabilises at 3 stars, until it again changes, and remains, at 4 stars—accounting for the rise of the rating average. The rating median behaves very similarly to the mode: days after the mode jumps to 4 stars, the median increases from 3 stars to 4 stars, reflecting the surge in the 4 and 5 star ratings that are input in this time, and accounting for the changes observed in both the mean and mode. A median of 4 tells us that *half* of the ratings in the system are 4 and 5 stars; however, more importantly, the *change* the median displays over time reflects that the distribution of ratings does not remain consistent. As above, it is impossible to deduce from the data why the global behaviour changed as we see here; changes to the Netflix interface, recommendation algorithm, user base, or combinations of these may be, but cannot be confirmed to be, the cause.

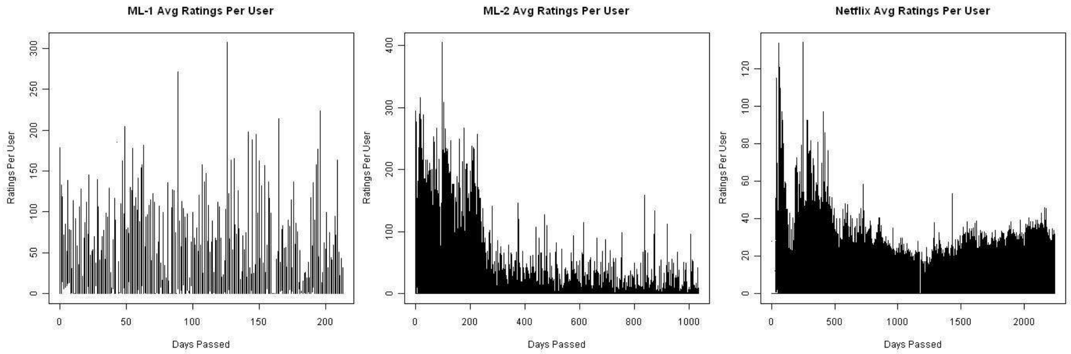
While it is possible to explore CF datasets from a global perspective, it is important to remember that the datasets represent a *collection* of individuals' profiles, and that the global state of the dataset can mask the state and changes that single profiles undergo. For example, as shown in Figure 3.11, if we first split the users into groups according to each user's number of ratings, we can then see how the group sizes fluctuate over time. In Figure 3.11, we bin users into four groups: (black) those with fewer than 10 ratings (excluding those who have yet to rate for the first time), (dark grey) those with $10 - 50$ ratings, (grey) those with $50 - 100$ ratings, and (light grey) those with more than 100 ratings. We then plot the relative group sizes as each dataset grows. These plots highlight the skewed distribution of profile sizes over time. In fact, the group of users with fewer than 10 ratings each may even be under represented in the data, although we do see that the Netflix prize data includes the highest proportion of this group.

The above analysis shows that global summary values fluctuate over time, reflecting how the overall distribution of ratings shifts as more users interact with the system. However, many algorithms that are used for CF do not use global summary statistics, but rather prefer to formulate predictions using either the item or user mean rating values. These values are also subject to change, as we show in Figure 3.12, where we plot the average item and user mean ratings over time. Each perspective (item-based or user-based) of the average means falls into a different range over time. Interestingly, the average *user* mean rating is consistently higher than the average *movie* mean rating; while users tend to rate



(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

Figure 3.12: Average User and Item Mean Rating Over Time



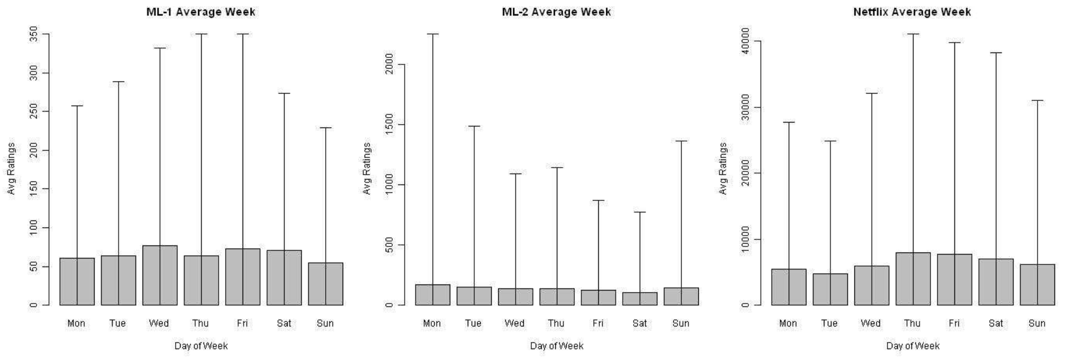
(a) ML-1 Dataset (b) ML-2 Dataset (c) Netflix Dataset

Figure 3.13: Standard Deviation of Ratings Per User Per Day

positively, there are items that are not liked (and thus rated lower), which pulls down the average item mean rating. The datasets not only remain sparse, but also do not stabilise (with the exception of ML-2, which stops growing); recommender systems continuously have to make decisions based on both *incomplete*, *inaccurate*, and *changing* data, and the range of the changes we observe in the Netflix data are likely to have a strong impact on the predictability of ratings.

3.2.3 Temporal User Behaviour

Thus far, our focus has been on the data: how the volume of users, items, and ratings grow and how summary statistics derived from them will change. Since we are dealing with explicit rating datasets, the mere act of rating also reveals how users are interacting with the system. To explore how user behaviour will vary over time, we plotted the standard deviation of the number of ratings input by returning users (i.e., users who have previously visited the sytem and input ratings at least once) per day in Figure 3.13. The plots show the high variability in how users interact with the recommender system. Both the ML-2 (Figure 3.13(b)) and Netflix (Figure 3.13(c)) datasets have high initial fluctuation in average user ratings per week; following this, the mean value flattens out. The ML-1 dataset, instead, has a more steady stream of average ratings per user, with small peaks corresponding to days that users (on average) rated more. Both of the MovieLens datasets have a much higher dispersion—many of the bars are over 100—



(a) ML-1 Dataset

(b) ML-2 Dataset

(c) Netflix Dataset

Figure 3.14: MovieLens: Average Number of Ratings Per Week (With Standard Deviation)

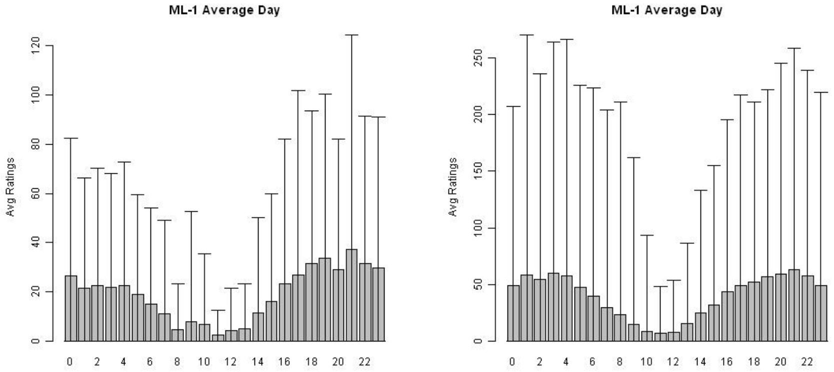
while the Netflix data (after the initial high period) falls below 50: it seems that users are proactively rating more in the MovieLens system.

3.2.4 Daily and Weekly Trends

In the previous section, we observed how user rating behaviour fluctuates over time, by looking at the entire window available for each dataset. However, this same rating behaviour can be further summarised by relating it to the day of the week when the ratings are input. In Figure 3.14 we plot the average number of ratings input per day for each dataset. Netflix sees its highest activity at the end of the week, as more ratings tend to be input on Thursdays, Fridays, and Saturdays than the other days. However, as the two MovieLens datasets show us, these results are again dependent on the subset of ratings available in the dataset: both ML datasets come from the same *system*, yet display very different rating activity. ML-1 rating trends tend to be lower during the weekend, with most ratings being input Wednesdays-Fridays, while the ML-2 dataset shows us the opposite, where more ratings are received on Mondays than any other day of the week.

Since the MovieLens timestamps allow us to know the precise moment when each rating was submitted, we can extract a finer-grained view of user activity over an average day in the system. Instead of binning ratings according to the day they were input, we binned them by hour, and plot the results in Figure 3.15. Unlike Figure 3.14, the two datasets now show very similar activity patterns: users tend to rate movies in the evenings, and the lowest volume of ratings appear roughly between 8am and 3pm; we assume this may be the cause since the majority of the system users would otherwise be occupied at work during these hours.

This analysis reflects an important aspect of recommender systems: the data is being produced by *people*, who tend to exhibit regular patterns of behaviour. The fact that people are behind the data input process also bounds the number of ratings we can expect to be input by a single person in a particular period of time. For example, it seems unlikely for a person to be able to rate 100 movies in less than a minute; moreover, if they were able to input this number of ratings, we could question the extent to which this person is providing honest (and thus, not noisy [APO09]) values. We will revisit this result and use this conclusion when we address the problem of recommender system robustness (Chapter 6).



(a) ML-1 Dataset

(b) ML-2 Dataset

Figure 3.15: MovieLens: Average Number of Ratings Per Hour (With Standard Deviation)

Regardless of how collective behaviour changes over time, the focus of a recommender system is to harvest user ratings in order to then generate personalised recommendations for each user. As we have seen before, this operation relies on the assumption of persisting *like-mindedness* between users. In other words, changes to the data over time are only important if they affect the quality and accuracy of ranked recommendations. We begin to explore this facet in the following sections, where we investigate the extent to which measurable similarity persists over time.

3.3 Similarity Over Time

The various algorithms that have been applied to collaborative filtering contexts operate in different ways, but all focus on capturing the *similarity* between users or items as content is rated. For example, nearest-neighbour algorithms focus on similarity by using explicit similarity metrics and making predictions with the most similar items (or users), and factorisation methods project item pairs into feature spaces where the similar pairs will be near one another. In this section, we explore how measurable similarity changes over time. In Section 3.3.1, we redefine the similarity metrics on which we will focus. We then look at similarity from two perspectives: the *static* case (Section 3.3.2), allowing us to visualise the effects of different similarity weights, and the *temporal* case (Section 3.3.3), which explores how similarity changes over time and the consequences of it doing so.

3.3.1 Similarity Measures

We focus on three metrics: the Pearson Correlation Coefficient (PCC), the Vector (or Cosine) Similarity, and the Jaccard distance. The simplest similarity measure between two user profiles—the Jaccard distance—can be derived using information that disregards the actual ratings themselves, but considers two other factors. The act of rating an item is a conscious decision made by human users, and represents a judgment on a product that has been “consumed” (viewed, listened to, etc.). Therefore, when two users have selected the same product, they already share a common characteristic: their choice to consume and rate that product. This similarity measure disregards each user’s judgment of the item, and weights users according to the proportion of co-rated items:

$$w_{a,b} = \frac{|R_{a,i} \cap_i R_{b,i}|}{|R_{a,i} \cup_i R_{b,i}|} \tag{3.2}$$

The Cosine similarity measure works by comparing the intersection of two users' profiles as vectors of ratings:

$$w_{a,b} = \frac{R_a \bullet R_b}{||R_a||||R_b||} = \frac{\sum_i r_{a,i} \times r_{b,i}}{\sqrt{\sum r_{a,i}^2}\sqrt{\sum r_{b,i}^2}} \tag{3.3}$$

The PCC aims to measure the degree of agreement between two users by measuring the extent to which a linear relationship exists between the two users' historical ratings [HKBR99].

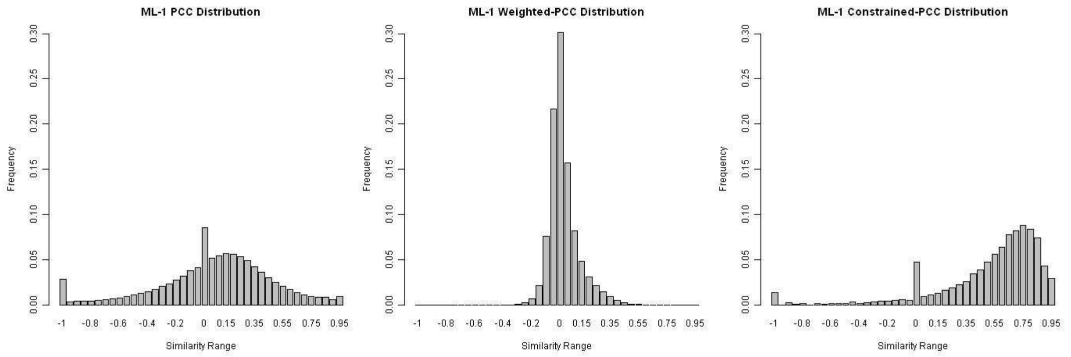
$$w_{a,b} = \frac{\Sigma_{i=1}^{N}(r_{a,i} - \bar{r}_a)(r_{b,i} - \bar{r}_b)}{\sqrt{\Sigma_{i=1}^{N}(r_{a,i} - \bar{r}_a)^2 \Sigma_{i=1}^{N}(r_{b,i} - \bar{r}_b)^2}} \tag{3.4}$$

We also include two variations of the PCC—the *weighted* PCC, where users who have co-rated n items (fewer than a threshold value $x = 50$ [HKBR99]) have their similarity scaled by $\frac{n}{x}$, and the *constrained* PCC, where user ratings are normalised with the rating scale mid point (2.5 stars) rather than each users' mean rating—making a total of five similarity measures.

3.3.2 Static Similarity

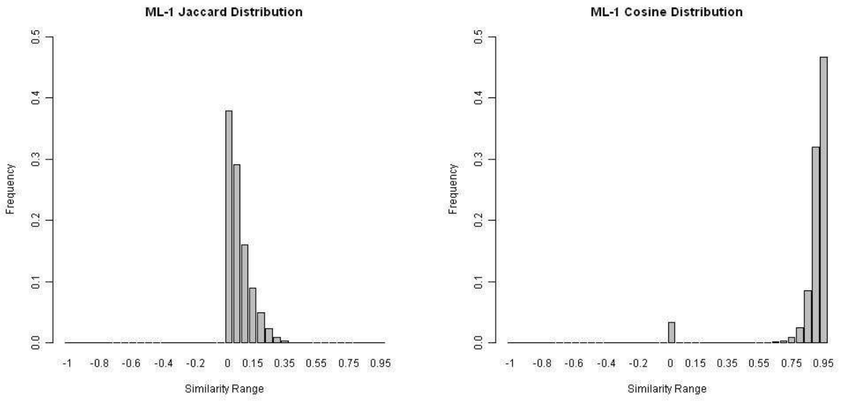
The intuition behind similarity metrics is that if they are well-suited to the problem at hand (i.e., finding good neighbours for users or items) then they will lead to better kNN predictions and, as a consequence, better recommendations. However, there is a problem with similarity measures that is best demonstrated with an example. If Alice's rating history for five items, on a five-point rating scale, is $[2, 3, 1, 5, 3]$, and Bob's rating history for the same items is $[4, 1, 3, 2, 3]$, then the Cosine similarity will be about 0.76. The PCC will return -0.50, while adding significance-weighting will produce -0.05. Other methods will result in equally different values. There is no consensus between the different methods as to how similar Alice and Bob are. Just as the relationship between Alice and Bob will change from good to bad depending on how they compute their similarity, selecting different coefficients will alter the weightings of all the user-pairs in the community. The relative ordering of similarity will also change: given three users (a, b, c), with $w_{a,b} < w_{a,c}$ when using the PCC does *not* imply that $w_{a,b} < w_{a,c}$ will remain true when using the Cosine similarity. The similarity values will, in turn, affect the prediction accuracy and coverage of the CF process.

We investigated the nature of these different similarity measures by looking at their distribution over the full range of available neighbours in the MovieLens-1 dataset. We focus on this dataset since, as we found in Figure 3.6, it is consistently the least sparse; similarity values derived from this dataset are thus assumed to be more reliable. We first computed all the coefficients between every pair of users, using all available profile information. We then plotted the proportion of the total number of coefficients that fell within a given range (in bins of size 0.05) to be able to see how these coefficients are shared out among all the available user pairs in Figure 3.16 and 3.17. The PCC distribution has two interesting peaks: one in the range of $(0, 0.05)$, and the other between $(-1.0, -0.95)$. In other words, a relatively high proportion of coefficients fall between the two ranges covered by these points: many users are either not similar or



(a) PCC Distribution (b) Weighted-PCC Distribution (c) Constrained-PCC Distribution

Figure 3.16: ML-1 PCC, Weighted-PCC & Constrained-PCC Similarity Distribution



(a) Jaccard Distribution (b) Cosine Distribution

Figure 3.17: ML-1 Jaccard & Cosine Similarity Distribution

very dissimilar to one another. Applying significance weighting to the coefficient changes the distribution drastically, by increasing the frequency of neighbours who have very low correlation. Nearly half of the user pairs are valued within $(0, 0.05)$, which implies that a high proportion of recommendations are weighted extremely lightly. The constrained-PCC skews the entire distribution toward the positive end; it seems thus that this variation of the PCC will increase the similarity between pairs of users that may otherwise have been deemed minimally similar with the standard PCC.

On the other hand, the similarity distributions based on the Jaccard distance peaks at 0, for the number of users who do not share any rated items. The rest of the user-pairs all share a positive similarity. Since this coefficient is derived using the number of co-rated items that the user-pair share, this coefficient cannot be negative, and thus a community of recommenders in this scenario will only have positive links. The Cosine distribution had the largest number of coefficients within a very high range: 0.78, or nearly 80%, of the community is weighted between 0.9 and 1.0. This is the result of summing the proportion of coefficients between $(0.9, 0.95)$, 0.32, and $(0.95, 1.0)$, 0.46. In other words, vector-similarity weights will favour neighbour recommendations much higher than, for example, the Jaccard distance. Finding that the majority of the population share similar coefficients may imply that the population is

neighbourhood	Co-Rated	PCC	Weighted-PCC	R(0.5, 1.0)	R(-1.0,1.0)	Constant(1.0)
1	0.9449	1.1150	0.9596	1.0665	1.0341	1.0406
10	0.8498	1.0455	0.8277	0.9595	0.9689	0.9495
30	0.7979	0.9464	0.7847	0.8903	0.8848	0.9108
50	0.7852	0.9007	0.7733	0.8584	0.8498	0.8922
100	0.7759	0.8136	0.7647	0.8222	0.8153	0.8511
153	0.7725	0.7817	0.7638	0.8053	0.8024	0.8243
229	0.7717	0.7716	0.7679	0.7919	0.8058	0.7992
459	0.7718	0.8073	0.8025	0.7773	0.7812	0.7769

Table 3.2: MAE Prediction Error, MovieLens u1 Subset

Dataset	Co-Rated	PCC	Weighted-PCC	R(0.5,1.0)	R(-1.0,1.0)	Constant(1.0)
u1	0.7718	0.8073	0.8025	0.7773	0.7812	0.7769
u2	0.7559	0.7953	0.7903	0.7630	0.7666	0.7628
u3	0.7490	0.7801	0.7775	0.7554	0.7563	0.7551
u4	0.7463	0.7792	0.7747	0.7534	0.7554	0.7531
u5	0.7501	0.7824	0.7784	0.7573	0.7595	0.7573
Average	0.7548	0.7889	0.7847	0.7613	0.7638	0.7610

Table 3.3: MAE Prediction Error For All MovieLens Subsets

full of very similar users, but following this same analysis using the PCC yielded quite opposing results. Once again, we found that the distribution given by each similarity measure does not agree with any of the others. There does not seem to be any unifying behaviour or descriptive characteristics, in terms of coefficient distribution, of the dataset, as the method for computing the coefficients is varied.

Any attempt at finding the “best” user weighting, to date, can only be done by conducting an analysis on comparative results of different techniques applied to the same dataset of user ratings; there is no way of measuring how close these algorithms are to an optimal answer. We can, however, produce a worst-case scenario: we construct a similarity matrix based on *random values*, and observe how accurately this scenario can generate predicted ratings. Random-based similarity does not use any information from the dataset to find like-minded peers; it simply is a set of uniformly distributed random values on a pre-defined range. We thus expected that the error reported on the prediction set would be devastatingly worse than when any similarity measures were used, since use of random numbers does not consider how much users have co-rated items or how much their ratings agree with each other.

In order to see how accurate predictions are with different similarity metrics, we measured the mean absolute error (MAE) of the predicted ratings *only* in the case when a prediction was made. If no information was available, typical experiments will simply return the user mean, and this value is not used when finding the MAE of the predictions. Since MAE measures the mean absolute deviation from the actual ratings, and the MovieLens dataset uses a five-point rating scale, the error measures can be expected to fall between 0, or perfect prediction, and 4.

We experimented with three ranges of random-similarity: $(-1.0, 1.0)$, or randomly assigning relationships so that the distribution of coefficients over all user pairs is uniform over the full similarity scale; $(0.5, 1.0)$, i.e. giving all the user-pairs high similarity relationships; and all 1.0, giving all user pairs perfect correlation.

Table 3.2 shows the prediction error results as k is increased, when using a subset of the MovieLens data (named $u1$). However, as we have seen, prediction results are dependent on the data that is being used. We therefore cross-validate our results by averaging the prediction error across five subsets of the ML-1 dataset (named $u1, u2, u3, u4, u5$). The most accurate results were obtained when predicted ratings were derived using all of the community members' ratings; Table 3.3 shows the prediction results for all subsets, when using this value.

To our surprise, the results of the experiments using random-valued and constant relationships were not only comparable to the performance of the correlation coefficients, but on average they also performed slightly better than the tested similarity measures. Such results would be expected if there were a certain degree of homogeneity amongst the community members, regardless of whether the specific correlation values agreed or not. A simple popularity-based recommender, which returns the average rating of an item (using all available ratings of it) also produces comparable performance. The average MAE over all data subsets, in this case, is 0.8182, which is 0.04 less than the weighted-PCC's accuracy.

The datasets may be to blame for the results; they may be too small, or not representative enough of a heterogeneous set of users. The MovieLens dataset we used does comply with the "long-tailed" characteristic of user-ratings; however, little more is known of what qualifies a rating dataset as appropriate. Repeating the above experiments with the Netflix dataset produced different results. Nearly all predictions were not covered, since randomly assigning neighbours to each user did not produce useful neighbourhoods. However, if we tune the experiment to account for the larger dataset size by selecting neighbours randomly from the pool of users who have rated the item that needs to be predicted, we again see similar results to the above. These results are another sign that the dominant error measures used to compare collaborative filtering algorithms may not be sufficient. Traditional similarity-based kNN cannot be differentiated from the output of random-similarity kNN. The results further highlight the fact that the current similarity measures are not strong enough to select the best neighbours. In the following section we will see that this result persists over time.

3.3.3 Temporal Similarity

An analysis of the distribution of correlation coefficients in the community of recommenders may, at first glance, seem inappropriate, since the coefficient values will change over time, as they are recomputed with growing user profiles. In this section, we examine *how they got there*, by looking at how similarity between users changes over time.

A useful means of analysing how similarity changes over time is to consider the act of computing similarity between all users as a process that generates a graph. In this case, each user is a node. Links to other nodes are weighted according to how similar the user-pair is, and (in the case of kNN prediction) the algorithm imposes the restriction that each node can only link itself to the k most similar neighbours; the out-degree of each node is limited. From this perspective, similarity graphs are a *constrained implicit* *social network* between the users in the system. The network is *implicit* since the users are not actively involved in selecting who they want to link to, and is *constrained* since the k parameter places an upper bound on the number of neighbours each user can have.

Observing similarity computation as a graph-generating process paves the way for a wide range of analysis that can be performed on recommender systems, drawing from methods described in graph theory and previous work on (explicit) social network analysis [BA02, MAA08]. The aim of analysing the graph generated by a filtering algorithm is to understand how the rating data is being manipulated in order to derive predictions. Furthermore, iterative updates of a recommender system can be viewed as re-generating the user graph. Changes in the graph when updates are computed will highlight how these systems perform over time, and give insight into why the parameters and methods that can be used to produce different accuracy results.

In the following sections, we explore the emergent properties of dynamic, temporal user-user similarity graphs, by decomposing the analysis into four separate stages:

- **Node Pairs**: Drawing from the growth of both nodes and rating information, we explore how similarity between a pair of nodes evolves over time. This analysis allows us to classify similarity measures into three groups, based on how they evolve the relationship between a pair of nodes: incremental, corrective, and near-random measures.
- **Node Neighbourhoods**: We have already mentioned that a kNN algorithm imposes restrictions on the graph, by allowing nodes to point to a pre-defined number of neighbours. We project this restriction onto the temporal scale, and observe the volatility of user neighbourhoods as profiles grow and similarities are re-computed.
- **Community Graphs**: The last section of our analysis considers the entire community of users. We computed properties such as connectness, average path length, and the in-degree distribution of links, to find that similarity graphs display the small-world, scale-free characteristic that is common to social networks. In other words, CF algorithms intrinsically favour some users over others; we refer to these as *power users*, and perform experiments that aim to collect the influence they exert on the predictive accuracy of the kNN algorithm.

User Pairs Over Time

Based on the way the datasets change over time, we first turn our attention to how the relationship between a *pair* of nodes will evolve. The primary concern of collaborative filtering, based on the user profiles explored above, is to predict how much users will rate items, in order to offer the top-N of these predictions as recommendations. As reviewed in Chapter 2, predictions are often computed as a weighted average of deviations from neighbour means [HKBR99]:

$$p_{a,i} = \bar{r}_a + \frac{\Sigma(r_{b,i} - \bar{r}_b) \times w_{a,b}}{\Sigma w_{a,b}} \tag{3.5}$$

In other words, a prediction $p_{a,i}$ of item i for user a is an average of the set of deviations $(r_{b,i} - \bar{r}_b)$ from each neighbour's mean rating $\bar{r}_b$, weighted according to the similarity $w_{a,b}$ between the user a, and

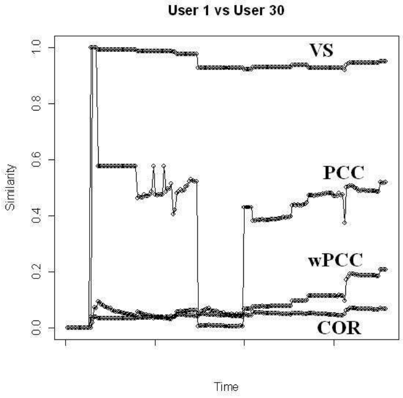


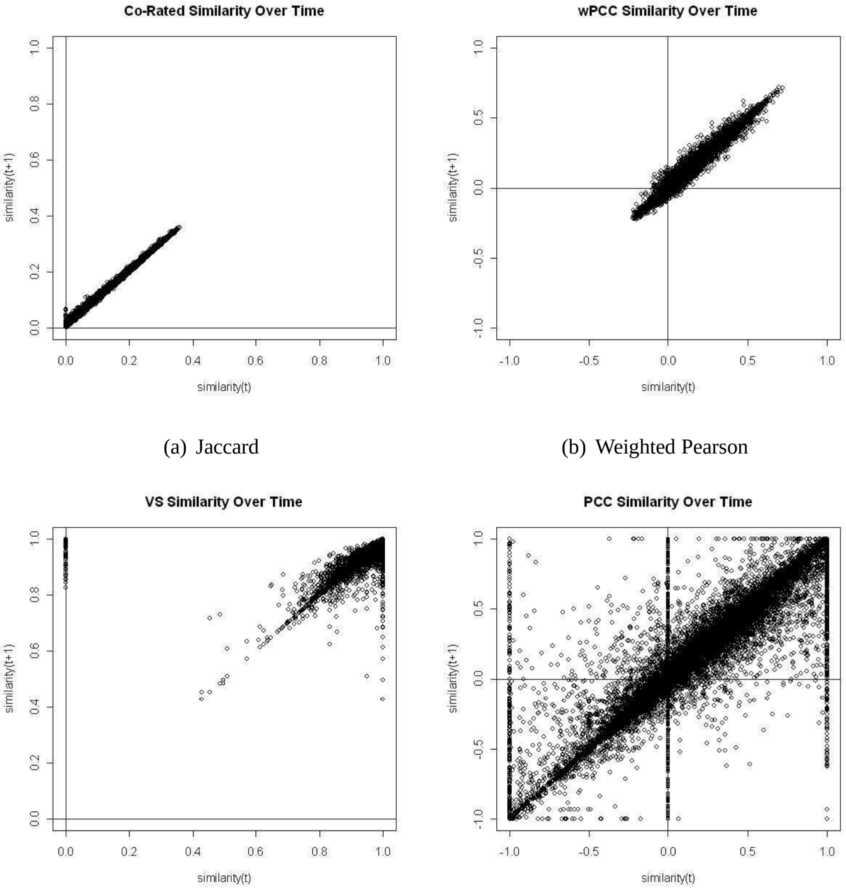
Figure 3.18: Similarity Between User 1 and 30: Similarity depends on how you measure it

neighbour b. All methods share the fact that they weight the contribution of each neighbour according to the degree of similarity shared with the current user: similarity is central to this process.

As we saw in Section 3.3.2, various similarity metrics offer different ways of computing similarity and will equally produce differing values. Despite this disagreement between similarity measures, one would expect the similarity between pairs of users to *converge*. As ratings are added to one of the two profiles, the similarity measure is computed on more information and should become more refined. However, some similarity measures do not display this behaviour.

We can consider a small example: users 1 and 30 from the ML-1 dataset. We chose this pair of users since their profiles have a large overlap over time (126 days), allowing for an extended view of their similarity's progression. If we order their profiles temporally, and then iteratively re-compute the similarity between the two as each user inputs a rating, we can observe how similarity evolves over time. Figure 3.18 shows the results of this experiment; in this case all measures return positive similarity between the users. The similarity for all measures begins at zero, when there is no overlap between the two users' profiles. Once they begin to co-rate items, the Cosine measure skyrockets to near 1.0, or perfect similarity. Over time, it very gradually degrades. The PCC measure also displays a large shift away from zero when the profile overlap begins and then both returns toward zero and jumps back up as the overlap increases. Only the wPCC and Jaccard measures grow slowly, without large shifts in similarity from one measurement to the next.

This example displays how the similarity between this particular pair of users progresses. In order to be able to generalise these results, we next aimed to analyse how the similarity of user 1's profile evolves relative to any other user in the system. There are a number of ways this evolution can be visualised; in this work we plot the similarity at time t, $sim(t)$ against the similarity at the time of the next update, $sim(t+1)$. This way we disregard the *actual* time between one update and the next, and favour focusing on how the similarity itself between a pair of users evolves. This method also allowed us to plot the similarity of one user compared to all others in the dataset, as we have done in Figure 3.19. These four images show the similarity of user 1 in the ML-1 dataset compared to the rest of the community, using different similarity measures. These results are similar to those we observed between the pair of users



(a) Jaccard

(b) Weighted Pearson

(c) Cosine

(d) Pearson

Figure 3.19: Evolution of Similarity for the Jaccard, wPCC, Cosine and PCC Similarity Masures, Comparing User 1 to All Other Users in the System

we examined before.

The first point to notice is that the range of values returned by the different similarity measures is not the same; some measures return values between 0.0 and 1.0, while others report values between -1.0 and 1.0. However, the more important aspect of these plots is the variance the points have from the diagonal, or the line $y = x$. If a point is on the diagonal it means that the similarity between the pair of users at time $(t + 1)$ is the same as it was at time t; nothing has changed. Similarly, if the point is below the diagonal then the pair is less similar that it was before, and a point above the diagonal implies that the measured similarity has grown. Therefore, the distance that these points have from the diagonal represents the extent to which similarity between the pair changed from one update to the next. As is visible in the plots of Figure 3.19, the greatest distance from the diagonal is reported in both the Cosine and PCC measures. These reflect the observations that were made when we compared user 1 and 30. Furthermore, they are representative of plots we created for other members of the community; these are not included here due to lack of space. The way that these methods evolve similarity between a pair of users follows one of three patterns. This allows for similarity measures to be classified according to their temporal behaviour:

k	COR	wPCC	PCC	Cosine
ML-1 Dataset: 943 Users				
1	2.48±2.3	2.54±2.3	4.94±6.5	10.59±16.8
10	22.22±16.3	22.18±16.2	25.15±19.9	35.80±41.4
20	42.06±28.6	42.12±27.9	41.73±26.4	49.88±45.3
100	171.80±87.9	168.99±83.9	156.27±67.4	159.03±69.9
150	237.86±109.4	230.23±104.9	216.94±88.6	221.16±87.1
ML-2 Dataset: 6040 Users				
1	1.69±1.1	1.74±1.2	3.51±3.8	3.16±5.1
10	16.75±9.3	16.85±9.4	22.75±19.0	34.68±36.2
20	33.22±17.7	33.33±18.1	40.08±28.2	60.02±54.6
100	160.26±79.1	160.93±79.9	161.68±77.4	187.02±118.3
150	236.97±113.6	237.99±114.5	231.35±102.6	255.23±142.9

Table 3.4: Average Unique Recommenders in Users' Neighbourhoods

- **Incremental**: In this case, as we observed with the Jaccard and wPCC methods, similarity begins at zero and slowly converges towards the final value. The difference between one step and the next is minimal, and therefore the relationship between a pair of nodes can be described as growing.
- **Corrective**: The Cosine method is noteworthy because similarity "jumps" from zero to near-perfect. However, once it has made this jump, the similarity between the pair tends to degrade, as can be observed by number of datapoints that fall below the diagonal on the graph. Therefore, this measure corrects its result after the initial jump.
- **Near-random**: The last class of similarity measures includes the PCC, and displays an exceeding amount of near-random behaviour. In other words, if similarity at time t is 0.0, or incomparable, and at time $(t + 1)$ there is measurable similarity, the PCC returns values over the entire range of similarity. Once it has made this jump from zero in either direction, it is not guaranteed to be well-behaved; as the plot shows, it may very well make a large jump again.

Dynamic Neighbourhoods

Now that we have observed how similarity evolves between a pair of nodes, we can widen the scope of our analysis and consider *user neighbourhoods*. The importance of measuring similarity of all user pairs is to be able to create a subjective ranking for each user of everyone else, and then to pick the top-k to form the user neighbourhood. The often-cited assumption of collaborative filtering is that users who have been like-minded in the past will continue sharing opinions in the future; this assumption has thus paved the way for learning algorithms to be applied to the problem of predicting ratings. If we project this assumption onto a longer time period, we would expect groups of users to naturally emerge from the data. In particular, when applying user-user kNN CF, as we do in this work, we would expect each user's neighbourhood to converge on a fixed set of neighbours over time.

To measure this property, we ran a modified CF experiment that includes the idea of system updates. The system begins at the time of the first rating in the dataset and is updated daily. While this value

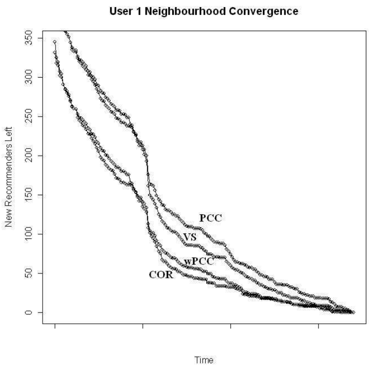


Figure 3.20: ML-1 User 1: New Relationships Left Over Time

perhaps corresponds to more frequent updates than most recommender systems can allow themselves to perform, it gives a finer-grained insight into the behaviour of the CF algorithm. At each update time, all user neighbourhoods are re-computed. In this chapter, we do not consider temporal accuracy, as we are focusing on the dynamic graph properties imposed by the algorithm.

As the users' profiles grow and neighbourhoods are recomputed, the users will be connected to a varying number of other users. The actual number of neighbours that a user will be connected to depends on both the similarity measure and neighbourhood size that is used. If, for example, $k = 1$, the user's profile is updated 10 times, and at each time step a different neighbour becomes the user's top recommender, then the user will meet 10 unique neighbours: the higher the number of unique recommenders, the higher the volatility of the user's neighbourhood. Table 3.4 displays the average unique neighbours for all users in the datasets.

The first point to note is that the number of unique recommenders is not close to k; in most cases it is nearly double the size of the allowed neighbourhood. In other words, even though a particular value of k represents the number of neighbours to use when making a prediction, the fluctuation of neighbours over time will be such that about double this value will be interacted with. For most values of k, the COR and wPCC similarity measures assign fewer unique recommenders to each user, a result that is not immediately visible when using the average number of neighbours across all users that Table 3.4 does.

Figure 3.20 shows the number of unique neighbours that user 1 has yet to meet over time when $k = 150$; it thus visualises how quickly the change within the user's neighbourhood will play out. As with the similarity plots, it is the shape of the plotted lines that gives insight into how neighbourhoods are changing over time: the steeper they are, the faster the user is meeting other recommenders. If the lines were step-shaped, the user would be meeting recommenders and staying connected to them for some time. Steeper lines, however, mean that the user's neighbourhood is converging faster, since the number of unique neighbours that have yet to be seen is decreasing. In fact, the Jaccard and wPCC similarity measures also converge to a fixed set of known recommenders faster.

Nearest-Neighbour Graphs

The last perspective we consider is the broadest view possible: the entire graph of user profiles. We have already seen that the volatility of each user's neighbourhood is quite large: this implies that the entire

k	Edges	Connected?	Max Path	Avg Path	Reciprocity
ML-1 Dataset: 943 Users					
1	1750	No	3	1.78	0.08
10	16654	Yes	4	2.63	0.13
100	148608	Yes	3	1.83	0.27
150	213260	Yes	2	1.76	0.33
200	272970	Yes	2	1.69	0.38
ML-2 Dataset: 6040 Users					
1	11406	No	5	2.58	0.06
10	109438	Yes	5	3.29	0.10
100	1055188	Yes	3	2.01	0.14
150	1568576	Yes	3	1.96	0.16
200	2076112	Yes	3	1.94	0.16

Table 3.5: wPCC-kNN Graph Properties

graph is being “re-wired” each time an update is performed. Therefore, in this section, we mainly focus on non-temporal characteristics of the dataset represented as a graph instead. Since the kNN algorithm determines where the links between users in the graph will be, the link positioning gives us the clearest insight into how the algorithm is manipulating the user-rating dataset. Table 3.5 shows a number of properties of the wPCC-kNN graph, for various values of k; we do not include results for the other similarity measures since they are very similar.

Path Length. Table 3.5 reports the maximum and average path length between any two nodes. These values were computed using Floyd’s algorithm, based on an undirected representation of the kNN graph. In other words, we assume that if a link between the pair exists (regardless of its direction), then so does some measurable quantity of similarity. Another curious characteristic of the values reported in Table 3.5 is that while k increases, the maximum and average path between any pair of nodes remains small, ranging from 1.4 to 2.9 hops; in fact, the graph demonstrates small-world properties that are very similar to those measured in explicit social networks.

Connectedness. An analysis of the entire graph, generated using only positive similarity links, shows that the clusters of users appear depending on the neighbourhood size parameter k that is used. When $k = 1$, a number of small clusters of users emerge, regardless of what similarity measure is used. The different methods only vary on average intra-cluster path length (as explored above); this reflects the way that these small clusters are shaped. In some cases, such as the wPCC graph, the clusters are formed of a group of nodes that all point to the same top-neighbour. In other cases, such as in the COR graph, the clusters form small chains of nodes, which accounts for the longer intra-cluster path length between users. The majority of these characteristics disappear as soon as k is incremented above one. As soon as users are allowed to point to more than their single most similar neighbour, the graph collapses in on itself: clusters are lost and, in most cases, the graph becomes fully connected.

Reciprocity. We counted the number of edges as the number of links between nodes, whether they be directed or not. In fact, when $k = 1$, the number of edges is *less than* the $1\times$ the total number

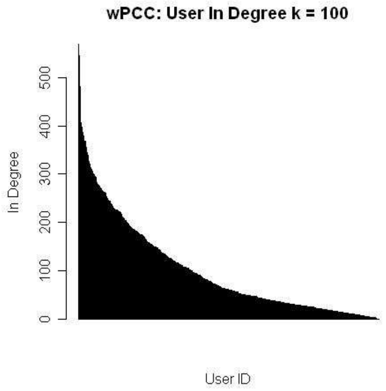


Figure 3.21: In-degree long tail of wPCC-kNN $k = 100$ ML-1 Graph

of nodes. This is due to the fact that in some cases, a pair of nodes point to each other; two directed links turn into a single undirected link, and the pair have a reciprocal relationship. Reciprocity is a characteristic of graphs explored in social network analysis [KNT06]; in our context it translates to the proportion of users who are in each other's top-k. On the one hand, reciprocity may be regarded as a desirable characteristic, since it implies that the generated graph really does pair very similar users together. On the other hand, high reciprocity can have dire consequences, as it will prevent information from being propagated over the similarity graph. The index of reciprocity that we use in Table 3.5 is the number of bi-directional links between nodes over the total number of links. The value ranges from 0, or no reciprocity, to 1, where all nodes pairs have reciprocal relationships. As the table shows, reciprocity grows as the allowed number of neighbours increases, and remains minimal when $k = 1$. However, it does not grow very much: adding a large number of links when k is incremented from 10 to 100 does very little to increase the measured reciprocity between users. This reflects the fact that although measured similarity is symmetric, this does not imply that each user will also have the same rank in the other's top-k; and this will matter when computing recommendations.

In Degree Distribution. We can further observe this phenomenon by considering the in-degree distribution of the nodes in the graph. The in-degree of a particular node n is the number of directed links that end on this node; in the context of collaborative filtering this equates to the number of users who place user n in their top-k. Figure 3.21 shows the in-degree of each user in the wPCC kNN graph, when $k = 100$. The distribution follows a power-law, much like the distribution that compares the number of ratings between different movies [LHC08b].

The in-degree distribution amongst users brings to light a new characteristic of kNN algorithms. Given a CF dataset and a nearest neighbour parameter k, there may be some users who are *not* in any other's top-k. Their ratings are therefore inaccessible and, although they will be considered when estimating the similarity between a pair of users, they will not be used in any prediction. To highlight this factor, we ran kNN prediction algorithms using the four similarity measures we are focusing on in this work on the ML-1 MovieLens subsets. Each rating in the training sets was coupled with a boolean flag, which would be set to true if the rating was used in making any prediction. We were thus able to

ML-1 Dataset				
k	COR	wPCC	PCC	VS
1	0.92	0.91	0.99	0.99
10	0.59	0.59	0.95	0.95
100	0.23	0.25	0.81	0.85
150	0.12	0.16	0.59	0.71
200	0.05	0.05	0.18	0.42

Table 3.6: Unused Proportions of the Dataset

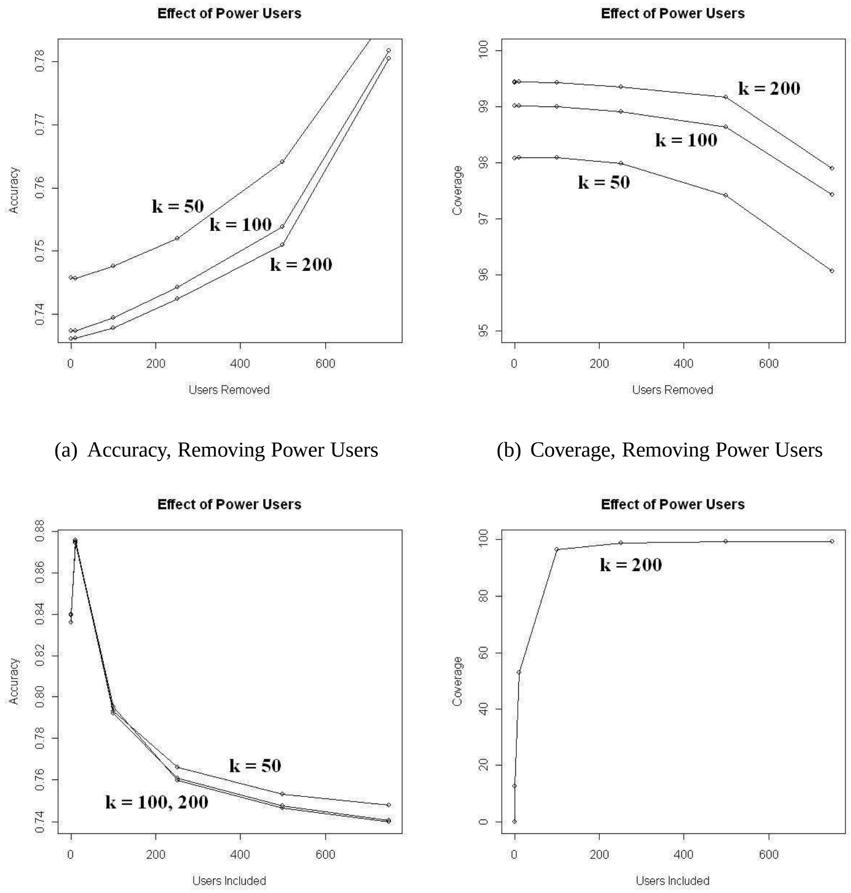
count how much of the training set remained unused after all the predictions had been completed.

Table 3.6 reports the proportions of the ML-1 dataset that are not used for varying values of k. The table does not reflect how many times individual ratings may have been used; it only counts whether the rating has ever been used or not. As the table shows, when k is very low, over 90% of the ratings are not used. In fact, these values of k generate predictions based on a very small subset of the training data, which may thus account for why they suffer from lower accuracy and impoverished coverage. As k increases, so does the use of the training data; if k were set to the total number of users in the system then the only ratings that would not be used would be those of a user who has no measurable similarity to any other in the system. However, a difference between the better-performing COR/wPCC and lower-accuracy PCC/VS similarity measures emerges once again: as k increases the former quickly use more of the dataset in predictions. When $k = 200$, only 5% of the training ratings are not used in predictions, while the VS similarity measure has barely made use of more than half of the ratings. The intuitive benefit of the COR/wPCC similarity measures may very well emerge here: they offer broader access to the ratings in the training set.

The Influence of Power Users

Another observation from Figure 3.21 is that some users will have an exceptionally high in-degree. We call this group *power* users; by being a frequently selected neighbour, they will have a stronger influence on the predictions that are made for others. These users emerge from the use of all the above similarity measures in kNN graphs. This is a characteristic that appears in other networks like the World Wide Web, movie actor collaboration graphs, and cellular networks, and is explained in terms of *preferential attachment* [BA02]. In other words, when a new node connects to the graph, the probability that it connects to another node is proportional to the in-degree of that node. In the context of collaborative filtering, therefore, it it important to understand the effect that generating a nearest-neighbour graph with power users has on the performance of the algorithm. We therefore ran two separate experiments. In the first, we forced all users' similarity with the top-P power users to be 0: in effect, removing their ability to contribute to predictions.

Figures 3.22(a) and 3.22(b) are the 5-fold cross validation mean absolute error and coverage results when removing a varying number of power users, for different values of k. As power users are removed, both accuracy and coverage worsen, although even when 750 (out of 943) profiles are made inaccessible



(a) Accuracy, Removing Power Users

(b) Coverage, Removing Power Users

(c) Accuracy, Only Power Users

(d) Coverage, Only Power Users

Figure 3.22: Results When Excluding or Exclusively Using Power Users

accuracy is still within 0.78. It seems, therefore, that the remaining non-power users can still make significant contributions to each user's predictions. These results reflect the dependency that accuracy has on the number of users in a system, another relationship that remains unexplored.

We followed this experiment by performing the inverse. Figures 3.22(c) and 3.22(d) show the 5-fold cross validation accuracy and coverage results when *only* the top-P power users are allowed to contribute to predicted ratings; if a neighbour is not a power user, a zero similarity value is set between the pair. The early spike in the plot is explained as follows: making predictions by simply returning each users' mean rating outperforms using only the topmost power user alone, but accuracy quickly returns to the same as when no users have been removed from the dataset when P increases; in other words, there are some user profiles in the dataset that do not contribute at all to the overall performance. The coverage plot shows a potential reason why these users are power users: the 10 topmost power users hold access to over 50% of the dataset.

3.4 Summary

In this chapter, we have examined the temporal characteristics of recommender system data, from the perspective of the ratings, users, and items. We have observed how the way people use recommender systems changes over time: new users and items are added, the rating distribution and both global and per-user/item summary statistics change. In other words, *all* features of the data that are used to make predictions in state of the art algorithms will vary with time.

We also performed a graph analysis of inter-user similarity, including the changes that appear throughout these graphs as time passes. The evolution of similarity between any pair of users is dominated by the method that is used to measure similarity, and the four measures we explored can be classified into three categories (*incremental, corrective, near-random*) based on the temporal properties they show. The number of unique neighbours that a particular user will be given over time also depends on both the similarity measured and parameter k used; furthermore, the rate at which they meet these new neighbours will vary for different similarity measures. Measures that are known to perform better display the same behaviour: they are *incremental*, connect each user quicker and to fewer unique neighbours, and offer broader access to the ratings in the training set. The focus here, therefore, is on the emergent *structure* of the graph using the MovieLens dataset.

In the following chapters, we shift our focus toward the temporal *performance* of CF algorithms. Collaborative filtering algorithms have traditionally been evaluated by: (1) splitting a dataset of user ratings into training and test sets, (2) feeding the training set into the learning algorithm, and (3) querying the algorithm for predictions of items in the test set. Evaluations are then conducted by comparing the predictions to the actual ratings that were withheld in the test set. There are two problems with this setup: both the *metrics* and *methodology*, in their current form, are not suited to a context in which a sequence of updates is required. We therefore first define a methodology for performing temporal experiments and examine how the changes to the data observed here affect the accuracy of rating predictions (Chapter 4). We then evaluate the temporal diversity in recommendations produced by changing data using novel metrics (Chapter 5). Lastly, we use the regularity in users' behaviour to construct systems that are robust to attack (Chapter 6).

Chapter 4

Temporal Accuracy of Collaborative Filtering

The primary task of a recommender system is to take user ratings and predict the values that users would attribute to content they have not rated, in order to generate personalised ranked lists of recommendations. Intuitively, the changes in the rating datasets that we have observed in the previous chapter will affect the performance of any learning algorithm that is iteratively retrained with the user ratings. In this chapter, we explore the extent to which this intuition is true: we first define a methodology for performing collaborative filtering temporal experiments and discuss a variety of design decisions that we made by reporting the results of two case study experiments (Section 4.1). We then perform and analyse a set of cross-validated temporal experiments with the Netflix data (Section 4.2). The key observation that we make is that state of the art filtering algorithms that are regularly batch-updated are not aware of their own temporal performance; we thus hypothesise that introducing this feature will improve an algorithm's temporal accuracy. We test this hypothesis in Section 4.3 by designing and evaluating a hybrid-switching CF algorithm that modifies how it predicts user ratings according to its performance to date.

4.1 Measuring Temporal Performance

At the broadest level we consider a scenario where, given a time t, we will train the CF algorithm with all ratings that have been input prior to t and want to predict the ratings (or a subset theoreof) that will be input *after* t. We then require a means of tracking performance over time. In this section, we examine the range of choices available when designing an experiment that mimics recommender systems that are updated.

4.1.1 Simulating Temporal Updates

Our generic description above prescribed a method for iteratively retraining CF algorithms. The simulated recommender system begins at time ϵ and will be updated at different times t. When an update occurs, the CF algorithm is retrained with the currently available ratings and then it derives predicted ratings of unrated items in order to present each user with personalised recommendations. There are a number of challenges that we face:

1. **Starting Point**: When should we begin the train-test cycle? If we begin at the first available date in the dateset, we will be making predictions with no ratings to learn from. In other words, how many ratings are enough to bootstrap a recommender system?

2. **Updates**: how often should the system be updated? Should we retrain the algorithm with all ratings input prior to the one we would like to predict? Or should the system be updated at some predefined regular interval (daily, weekly, monthly)?
3. **Test Sets**: how are they to be defined? By retraining CF algorithms with a growing dataset, we are simply performing a sequence of updates where the training set has been augmented. However, what should we be predicting? The test set could be a *static* set of ratings that will be input in some arbitrary time in the future, or a *changing* set of ratings based on what will be rated after the current update. Unlike the traditional methodology, we may also encounter a situation in which multiple predictions can be made for a user-item pair before the user rates the item. For example, if we are updating the system weekly, predicting all unrated items, and a user will rate an item one month after the current update, then we will make four predictions of the same rating prior to the user rating the item. Should they all be included in error measurements? If not, which one is the most *relevant*?

We explore these questions in Section 4.1.3 by comparing the results of different experiments; however, we first define the options available to measure temporal accuracy.

4.1.2 Metrics: Sequential, Continuous, Windowed

In terms of prediction error, there are three ways that a set of recommender systems' temporal updates can be evaluated. The first is a *sequential* view, where we compute the RMSE on each experiment separately. The alternative is the *continuous* time-averaged RMSE metric. To observe the dependence of prediction error on time, we modified the RMSE calculation, in a manner akin to that of the Time Averaged Rank Loss that is described in [CS01]. If we define R_t as the set of predictions made up to time t, then the time-averaged error is simply the RMSE achieved on the predictions made so far:

$$RMSE_t = \sqrt{\frac{\sum_{\hat{r}_{u,i} \in R_t} (\hat{r}_{u,i} - r_{u,i})^2}{|R_t|}} \tag{4.1}$$

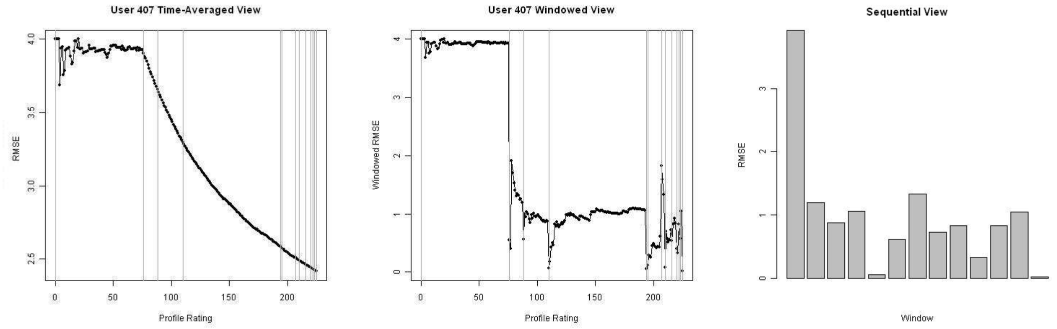
Similarly, we can define the time-averaged mean absolute error (MAE):

$$MAE_t = \frac{\Sigma_{i=0} |\hat{r}_{u,i} - r_{u,i}|}{|R_t|} \tag{4.2}$$

The last possibility is the *windowed* view. Error is accumulated and tracked using the continuous equations above, but once an update has been performed, we reset the error count to zero. This allows us to see how prediction error is distributed within a single update: are predictions more accurate immediately after the update? Do they become less accurate as time passes (since new ratings have not been accounted for in the CF algorithm)?

4.1.3 Case Study

Prior to committing to a particular methodology, we explore the options available by running a number of experiments. In the first, we focus on a single user from the MovieLens dataset; we then expand our analysis to include all of the MovieLens users. These experiments allow us to reason about what choices to make regarding the experimental starting point, update frequency, and predictions. Lastly, we run a



(a) Time-Averaged RMSE (b) Windowed RMSE (c) Sequential RMSE

Figure 4.1: User 407: Three Views of Temporal Error

group of experiments that look at temporal updates with a *static* test set and conclude that, while this form of experiment does not reflect the reality of recommender systems, they provide important insight into the influence of rating data on prediction performance.

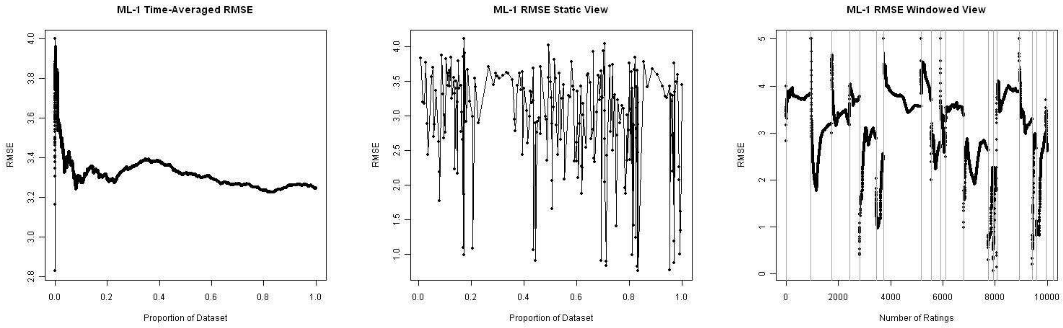
Single User

We begin with a single user; we picked the ratings of user 407 from the ML-1 dataset since they span a relatively lengthy time scale. We also make the following assumptions:

- The system will be updated *daily*; at each update user similarity is recomputed with all ratings input to date and predictions are made for any ratings that will be input before the next update. The first update occurs exactly one day after the first rating is input to the system. This allows us to (a) minimise how far into the future we have to predict (thus minimising any bias that may result from this) and (b) have a very fine-grained view of the system.
- Predictions will be made by a user-based kNN algorithm, where $k = 10$ and user-similarity is computed with the Pearson Correlation Coefficient. We therefore use a simple algorithm which has been extensively studied in the past [HKBR99].

We plot three temporal perspectives of the error in predicting user 407's ratings in Figure 4.1. Vertical gray lines in Figures 4.1(a) and 4.1(b) denote when the system was updated; each point represents the input of a successive rating. From these, we can make a number of observations:

- The user does not rate items consistently; for example, the number of items that were rated before the first update are far greater than those input before the second.
- All ratings input prior to the first update have a distinctly large error. From the algorithm's perspective, there is no data to use to predict this user's interests. The *cold-start* problem is often described as an issue that users with *few* ratings face; what we observe here, where the user has no historical profile, is an extreme version of it (i.e., had no ratings to generate a neighbourhood at the previous update and no mean rating value to provide an appropriate baseline prediction). Figures 4.1(c) and 4.1(b) show that the cold start region lasts until the next update; unfortunately, the time-averaged results in Figure 4.1(a) are skewed throughout the entire duration of the predictions by these initial predictions (the plotted error is on the range $[2.5, 4]$).



(a) Time-Averaged RMSE (b) Sequential RMSE (c) Windowed RMSE

Figure 4.2: ML-1 Dataset: Three Views of Temporal Error

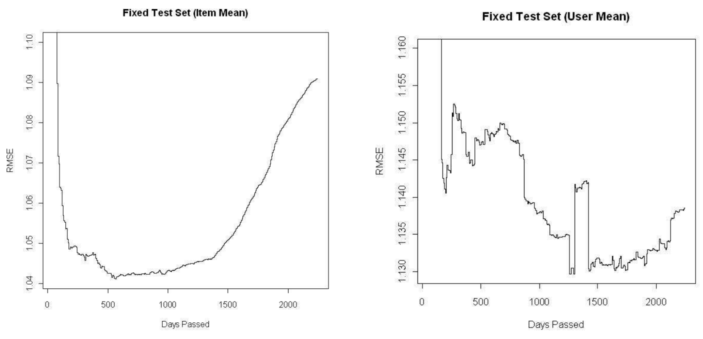
- Prediction error does not improve over time for this user. The time-averaged results seem to imply that predictions are improving; however, as stated above, this is due to the skew from the initial predictions. After the cold-start day, the error of predictions made in any given window range lies between slightly above 0 to just under 2.

We thus find that the time-averaged metric will only be appropriate if we do not include cold-start predictions. The windowed and sequential metrics do not suffer from this problem; in fact, they have already highlighted the large variability in prediction accuracy as time passes.

Groups of Users

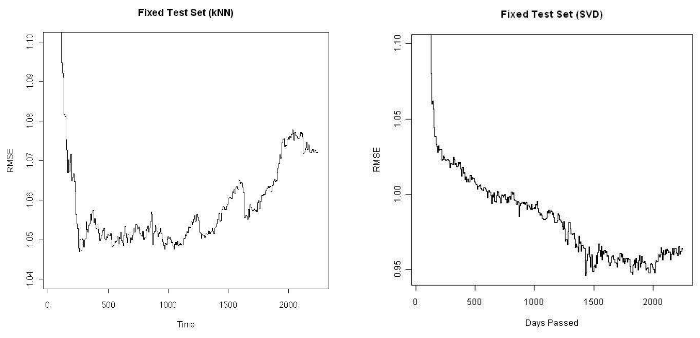
Based on the observations above, we broadened the scope of the experiment and included all the ML-1 users. This way we can view the same results as above for a large group of users: we plot these in Figure 4.2. The results highlight a number of points:

- Again, the time-Averaged RMSE is of little value if we include cold-start predictions. As above, users face the cold-start problem when they have no historical ratings; they thus have no mean rating value or neighbours. Our options here are to (a) modify our prediction algorithm in order to return an appropriate non-zero prediction for cold-start users, or to (b) disregard cold-start predictions. Since the cold-start problem has been approached from a variety of perspectives [NDB07, PPM$^+$06] we opt for the latter in this work rather than limit ourselves to a single available solution.
- The windowed perspective (Figure 4.2(c)) shows that inter-window behaviour does not follow a single pattern. There are some windows that, as they progress, become ever more accurate; there are also windows that become less accurate as time passes. However each window is distinctly different from the others: how many ratings are input, along with what items are being rated, continuously changes.
- The sequential view (Figure 4.2(b)) is a summarised form of the windowed perspective; each point represents the *average* error of each window. In fact, since our prediction model is updated iteratively, the windowed results (Figure 4.2(c)) will be subject to the order that ratings are input. The static and time-averaged views, instead, are both not subject to this limitation and useful for



(a) Item Mean (b) User Mean

Figure 4.3: Temporal Experiments With a Static Test Set (User/Item Mean)



(a) kNN, k = 50 (b) SVD, f = 64

Figure 4.4: Temporal Experiments With a Static Test Set (kNN/SVD)

measuring performance across updates.

Predicting Static Test Sets

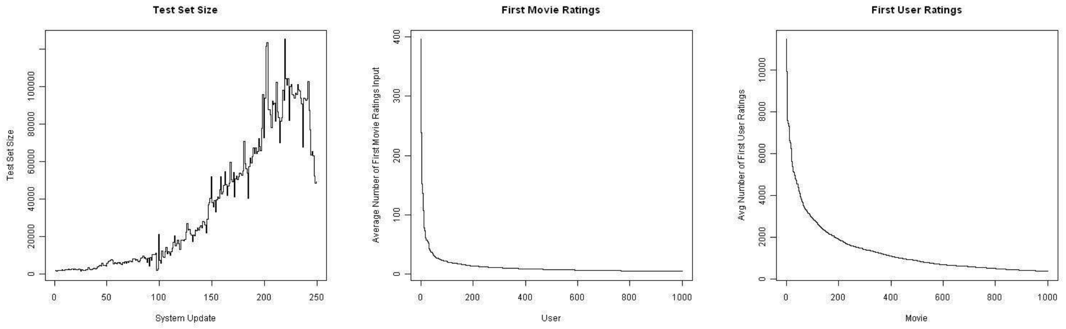
How should we define our test set? We can either keep the test set fixed and only change the training set, or update both as the simulated temporal updates are performed. In the context of a deployed recommender system, we assume that the prediction that has the greatest impact (in terms of the recommendations generated for each user) is the last one made before the rating is input. Why? Changes in the predictions as they are updated will affect each user's recommendations: when users rate items, only their current recommendations (i.e., ranked prediction values) will be influencing their decisions. Therefore, at time t (with update frequency μ), we decided to only make predictions for ratings that will be input in $t + \mu$; no predictions are recomputed or updated for ratings that the users have already input or will input after $t + \mu$. In other words, predictions are only made once. This may differ from deployed recommender systems, that do not know *when* users will rate items, and will therefore not be able to update predictions until the user inputs a rating. However, as described above, this allows us to focus on the predictions that will have the greatest impact on a user when they are rating an item. In the following chapters, we will remove this assumption when we evaluate the top-N lists created over time.

The alternative to the above would be to keep a *static* test set. In other words, we define an (unchanging) set of ratings that we would like to predict, and observe the prediction error as we add more data to the training set. This allows us to explore how prediction quality varies with time from the opposite point of view of the above: we can see the performance trajectory of the system as it approaches the state where the user will rate the predicted item. We tried this setup with the Netflix dataset: we first made a static test set, consisting of all ratings input in the first fifty days of the dataset, and reserved the rest as training data. We selected the ratings from the first (rather than last) fifty days since this guaranteed that the items we are predicting will already be in the system and will continue to be rated in the training data. We selected four CF algorithms: two baseline prediction methods (the user and item mean), an item-based kNN with $k = 50$ neighbours, and a SVD with 64 user and item features; this range of choices both reflects state-of-the-art CF and each manipulate the rating data in different ways. We then iteratively trained each algorithm with a growing dataset, incrementing it with one week's worth of ratings at each round. The choice of one week increments is arbitrary; in our case, it allows for a relatively high number of ratings to be added to the training set. Given that we used the Netflix dataset for these experiments (which spans a longer time frame), this choice also requires fewer iterations of the algorithms to be run. After each training phase, we queried the algorithms for predictions of the test set, and plot the time-averaged RMSE results in Figures 4.3 and 4.4. All of the results share common traits: on the left side of the plots, where very little training data is available, RMSE values are very high. These results hint at the fact that when more training data is available CF algorithms will be able to make better predictions. Each prediction method's results shows different amounts of variability; the most notable is the user mean, which has very clear changes in performance when the test set users add more ratings to their profile. However, all of the methods' best predictions were made prior to the full training data being made available to the algorithms. Even the SVD, with RMSE results that seem to decrease monotonically over time, hits a minimum value before all the data has been given to it. All of the minimum values occur at different times, highlighting how prediction algorithms will each be affected in different ways by the available data, and any noise within it [APO09].

The purpose of these experiments was to see how CF accuracy is affected by a growing training set, from the point of view of a fixed set of ratings that need to be predicted. In practice, these results show us how accuracy will vary as predictions for unrated items are updated. For example, assume that a user does not rate a movie for one month after it becomes available. The system does not know when the user will rate the item and will thus continue updating its prediction until the true rating is input. These results show us that the prediction (which determines whether or not the movie will be recommended) may suffer from high variability and will not necessarily improve as time passes. Deployed recommender systems, however, do not have the luxury of having a closed test set: as we saw in Chapter 3, the available items will continue to grow over time. Real systems will thus have to face a *dynamic* test set: they continuously have to predict the future. In the following sections, we perform experiments reflecting this context; we begin by explicitly defining how we will do so.

4.1.4 Methodology

Based on the exploratory work we reported above, we define here how we conducted temporal experiments. Given a dataset of timestamped user ratings, a start time ϵ and update frequency μ, we define:



(a) Test Set Size Over Time

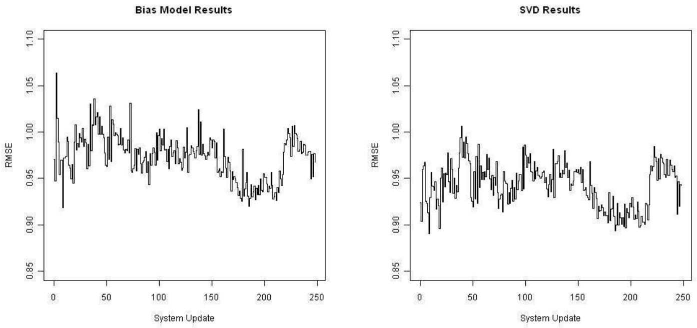
(b) Users Ratings Removed

(c) Items Ratings Removed

Figure 4.5: Temporal Experiment Test Sets' Characteristics: Size, and Distribution of Users Who Rate Items First and Items that Are Rated First

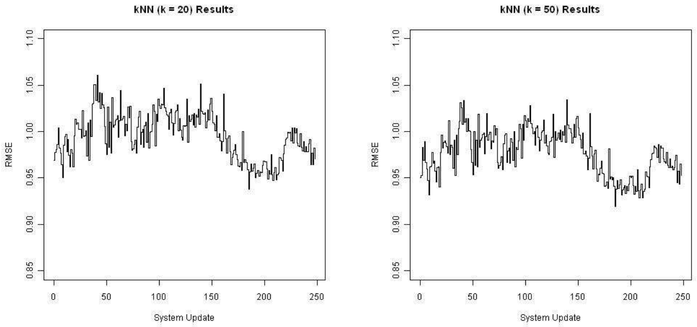
1. **Starting Point**: we define the starting time as ϵ and elected to wait for an arbitrary number of days before beginning the train-test cycle; this allowed us to observe a system that (in terms of number of ratings) is not suffering from system-wide cold start problems. We denoted this number of days as the "edge." In the Netflix experiments below these are any ratings input in the first 500 days of the dataset; our data thus allows for 250 temporal updates.
2. **Updates**: we elected to update the system based on accounts of deployed large-scale recommender systems [Mul06]; the system will be updated weekly. When it is updated, it will train with all ratings input up to the current time.
3. **Test Sets**: After each update, the system will be queried for predictions concerning ratings that will be entered before the next update, only if both the user and item have at least 1 historical rating. We thus still expect to see how our algorithms cope with the cold start problem; however, this assumption will remove the need to define a default prediction to return in the case of no history.

This setup has two implications, due to the temporal structure of the dataset: on the one hand, the number of historical ratings (or training set) will grow as t increases. On the other hand, the number of ratings in $t + \mu$ (the test set) will also increase, as plotted in Figure 4.5. It is interesting to note that pruning the test sets of items and users who have no history tends to exclude some users more than others. Figure 4.5 includes two plots that highlight this feature. Figure 4.5(b) shows the average number of ratings pruned per user; these ratings are pruned from the test set since the user is rating movies that have no historical ratings. Figure 4.5(c) shows the equivalent distribution for the movies; these show ratings excluded from the test set because they are the first ratings input by each user. This highlights an important characteristic of the data set: there are certain users who are consistently rating items that have never been rated before (items that have no data available for them to be recommended), and seem to be exhibiting behaviour that extends beyond merely responding to recommendations [HKTR04]. There are also movies that consistently appear as users' first rating, which may give insight into what recommendations Netflix was offering to new users.



(a) Bias Model (b) SVD

Figure 4.6: Sequential RMSE Results for User Bias Model and SVD



(a) kNN, $k = 20$ (b) kNN, $k = 50$

Figure 4.7: Sequential RMSE Results for kNN Algorithm With $k \in \{20, 50\}$

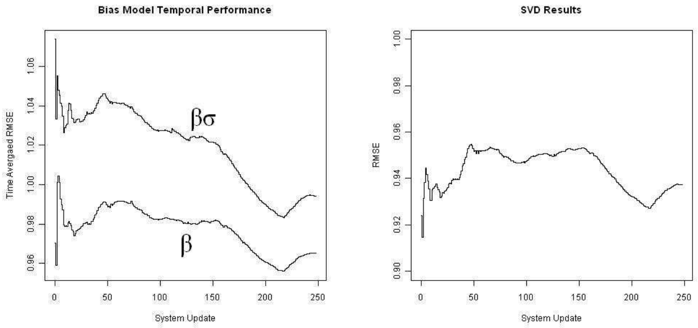
4.2 Results

We now evaluate CF algorithms *over time*, as they are iteratively applied to a growing dataset of ratings. In order to cross-validate our results, we subsampled the Netflix dataset. To do so, we split the users into 50 bins (according to profile size) and randomly selected $1,000$ users from each bin; by repeating this process, we produced five subsets of $50,000$ users. We then selected all ratings belonging to these users and any rating input before time ϵ. Our final subsets have about $60,000$ users: setting the ϵ value as we did is equivalent to bootstrapping a recommender system with $10,000$ users.

We focus on three algorithms: Potter's bias model [Pot08], an item-based kNN (with $k \in \{20, 35, 50\}$), and a SVD with 64 user and item features. In doing so, we cover *baseline* models, the ever-popular *nearest-neighbour* method and a *factorisation*-based approach, which represent three different and important state-of-the-art algorithms.

4.2.1 Sequential Results

The sequential results for the bias model, kNN with $k = 20, 50$, and the SVD are in Figures 4.6 and 4.7. From these we can observe that CF algorithm performance lies on a *range* of values. The $k = 50$ results fall in $[0.9193, 1.034]$, while the range for $k = 20$ is slightly worse, $[0.9383, 1.0608]$; nevertheless, the ranges overlap significantly. However, while the bias model outperformed both kNN methods on the probe [LHC09b], its temporal performance is between 0.9186 and 1.0637: at best, it shows a minor



(a) Bias Model

(b) kNN For Users With $|R_u| \in [0, 10)$

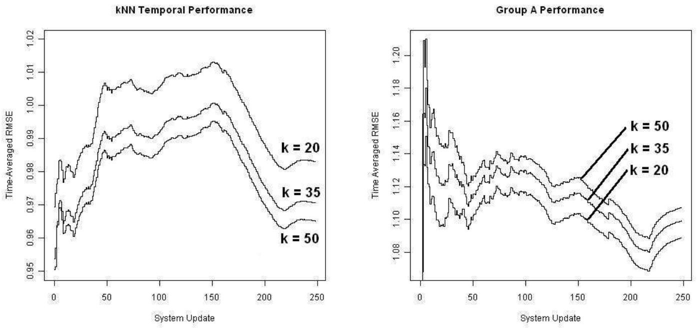
Figure 4.8: Time-Averaged RMSE for User Bias Model and SVD

improvement over $k = 50$, while in other cases is outperformed by $k = 20$. Similarly, the SVD values range between $[0.8907, 1.0061]$; while achieving the best minimum, that values are not consistently lower than the other algorithms. The performance across all methods falls by approximately 0.02 between the 218th and 219th update, highlighting a change in the *data* that results in all methods degrading in performance. While the trend between the different plots is roughly similar, the precise moments that each algorithm performs best (or worst) differs between each method. Both kNN methods achieve their lowest RMSE on the 186th update; however, $k = 50$ yields its worst performance on the 140th update, while the equivalent for $k = 20$ happens at the 42nd update. The bias model achieves both the best and worst performance within the first 10 updates. The SVD, instead, hits its minimum on the 8th update, and maximum at its 39th update.

What do we learn from these results? Viewing the sequence of RMSE results emphasises the difficulty of identifying which algorithm outperforms the others. Ranking the algorithms according to performance is dependent on what snapshot of the data is currently being trained with. However, the $k = 50$ parameter was more accurate than $k = 20$ in 248 (of the 250) iterations. Similarly, the SVD is more accurate than the $k = 50$ kNN for 245 updates. The balance between $k = 50$ and the bias model is not as one-sided: the bias model is more accurate in about two-thirds of the updates (158), while in the other 91 cases the kNN model is more accurate. In other words, while it is possible to deduce relative performance based on a set of results, the best performing method in any *individual* time segment varies.

4.2.2 Time-Averaged Results

The time-averaged results of 5-fold cross validated experiments are shown in Figure 4.8 and 4.9. This visualisation provides a different perspective on the experimental results, and there are a number of observations that can be made. Figure 4.9(a) shows that $k = 50$ tends to outperform $k = 20$ over time. We also tried experiments with $k = 0$; in this case the current item mean is returned. All values of $k \neq 0$ consistently outperformed this baseline; this result persists if the current *user* (rather than item) mean rating is returned. The difference in time-averaged performance of each kNN parameter setting is less than 0.02, and remains approximately constant after the 50th system update. The performance itself



(a) kNN $k = \{20, 35, 50\}$ (b) kNN For Users With $|R_u| \in [0, 10)$

Figure 4.9: Time-Averaged RMSE for kNN Algorithm and Users With Fewer Than 10 Ratings

varies: after the $50th$ update predictive accuracy wanes. However, after the $150th$ update performance once again improves, falling sharply by 3% in the case of $k = 50$. This highlights the dependence that these methods have on the data they train on.

Figure 4.8(a) plots the time-averaged performance of the bias model. The bias model with variance scaling is consistently outperformed by the model that has no variance adjustment. The differences in performance are in the range $[0.03, 0.1]$: scaling user ratings with a *dynamic* variance introduces more error to the predictions. Why do the probe and temporal results differ? One indicative factor is the difference in the rating distribution over time; users with fewer than 10 ratings make up more than half of the dataset for most of the interval we consider. However, only 3% of the users remain in this group when considering the entire dataset. The majority of the user variance values, in the temporal case, are therefore computed with incredibly sparse data.

Comparing Figures 4.9(a) and 4.8(a): although the bias model outperforms kNN when predicting the Netflix probe, it does not consistently outperform kNN on the temporal scale. For example, in the 4th update, the bias model time-averaged result is 1.004, while the kNN result is 0.964. From these results, kNN with $k = 50$ emerges as the most temporally accurate method. However, we also explored how prediction error is distributed across a community of *individuals* by, once again, splitting users into groups according to profile size and plotting the group's 5-fold cross-validated time-averaged performance. As expected, group performance is proportional to the range of ratings that defines the group: the group of users who have fewer than 10 ratings also have the least accurate predictions, compared to the groups with more ratings. However, as shown in Figure 4.9(b), the k value performance in the group with fewer than 10 ratings is the opposite of what we observed when all groups were merged: larger neighbourhoods leads to *less* accurate results.

4.2.3 Discussion

The above results provide insight into a number of characteristics of recommender systems. The foremost observation to be made is that recommender systems are not built to be *aware* of their own temporal performance. Each update is treated independently of the rest: algorithms are retrained with all of the available data, and no changes are made based on the temporal performance to date. The experiments also show the range of results that algorithms produce: a single snapshot of algorithm performance is not sufficient to conclude that one algorithm is indeed more accurate than another. In fact, there is often no consensus between the method that produces the best *global* performance and that which best suits *each user*.

These conclusions led us to formulate the following hypothesis: collaborative filtering algorithms that modify how they predict user ratings (by *switching* algorithm [Bur02] or *updating* parameters) based on their temporal performance will be more accurate than algorithms that do not. In the following sections, we test this hypothesis by designing and evaluating temporal hybrid switching algorithms.

4.3 Adaptive Temporal Collaborative Filtering

Currently, prediction methods are applied iteratively as the data grows; the only change from one step to the next is rating *data* that is input to the algorithm. In particular, no information on the current performance is fed forward to the next iteration of the algorithm. We therefore propose temporally *adaptive* collaborative filtering, which will make use of this information to change the algorithm that is used at each iteration. There are two adaptive methods that we explore and evaluate. The first selects between different algorithms (Section 4.3.1), while the second is based on only adapting kNN (Section 4.3.2) or SVD (Section 4.3.3) parameters.

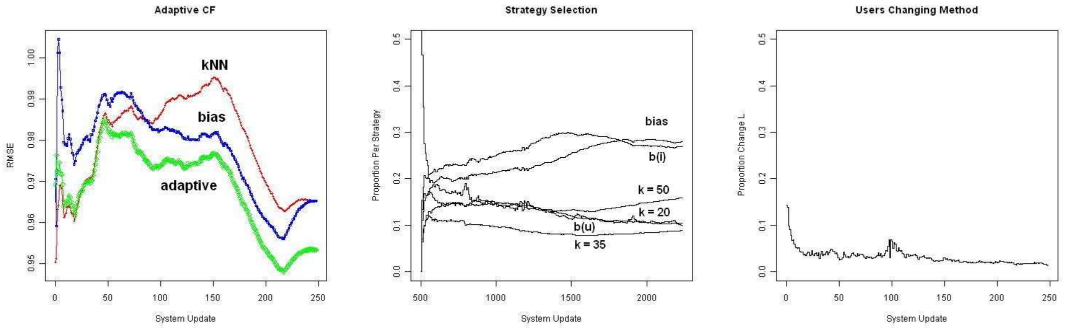
4.3.1 Adaptive CF

To implement temporally adaptive CF, we begin with a pre-defined set P of CF algorithms. In this work, the set includes kNN, with $k = \{0, 20, 35, 50\}$, and the bias model. A $k = 0$ value disregards neighbourhoods completely; in this case we can either return a baseline item ($b(i)$) or user ($b(u)$) mean rating (there are six candidate methods altogether). Each user u is assigned a label $L_{u,t}$ denoting which algorithm L best predicts their preferences at time t. At each time step, each user also has a corresponding error value $e_{u,t}$ denoting the time-averaged RMSE achieved on the predictions made to date on *the individual profile*. We therefore aim to minimise the per-user $e_{u,t}$ value by selecting the $L \in P$ that would have maximised the improvement on the current error:

$$\forall u : L_{u,t+1} = \max_{L \in P} (e_{u,t} - RMSE_{u,t}) \tag{4.3}$$

Although the previous analysis binned users according to profile size, and demonstrated that relative performance varies depending on the group being considered, we did not opt to adapt based on which "group" users belonged to. We did this for two reasons: first, the grouping was done with pre-defined values that could themselves benefit from fine-tuning; secondly, this form of grouping continues to mask the predictive performance on *individual* profiles, and the aim we envisage for adaptive filtering is based on addressing users' profiles individually. In doing so, the CF algorithm that provides personalised recommendations becomes itself personalised.

The five-fold cross-validated time-averaged RMSE results for the adaptive method are plotted in Figure 4.10(a), compared to the two best individual methods: kNN with $k = 50$ and the bias model. As the plot shows, the adaptive method begins by following the same pattern as the kNN curve, but then



(a) Adaptive CF (b) User Proportions Per Algorithm (c) User Change Proportions

Figure 4.10: Time-Averaged RMSE Comparing $k = 50$, the Bias Model, and Adaptive CF; Proportions of Users Who Selected Each Algorithm Over Time, and Proportions of Users Who Changed Method At Each Interval

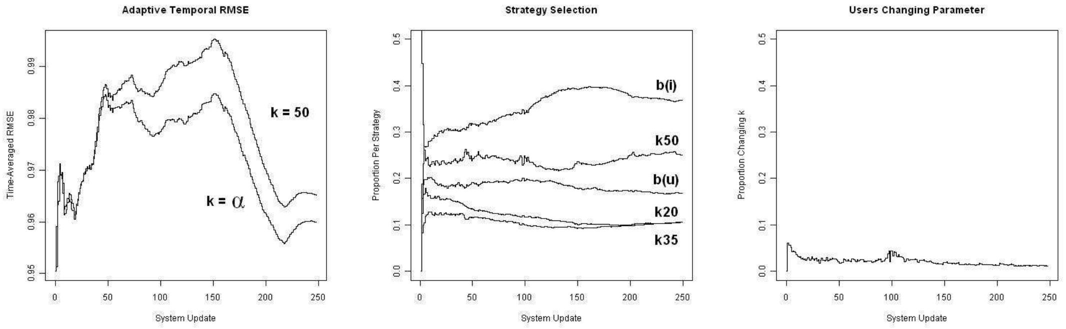
(as the bias model becomes more accurate) departs from this pattern and becomes more accurate than either model alone. In fact, adapting on a per-user basis offers better temporal accuracy than if we simply selected the minimum of the two methods. We also plotted in Figure 4.10(b) the proportion of users who select each method over time. The results show that, while the bias model dominates the others (in terms of the proportion of users that the algorithm selects the bias model for), it is selected for less than 30% of the users: no single model 'best' predicts the majority of end users.

To gain insight into how often the algorithm needs to change a decision it had previously made, we plotted the proportion of users who, during the update, changed algorithm from the one used during the previous window (Figure 4.10(c)). Overall, very few of the growing population of users changes method from one update to the next; the change is consistently between 1.3% and 14.3% of the growing user community, and on average is 3.1 ± 1.6%.

4.3.2 Adaptive kNN

While the above method offers greater accuracy, it has two shortcomings. First, it is *expensive*: multiple CF algorithms must be implemented and independently trained at each update on the growing data. Given the volume of data that large scale recommender systems must handle and the time it takes to train CF algorithms [Mul06], repeating this process with multiple algorithms may be prohitive and difficult to scale. However, if the cost can be incurred, and the goal of doing so is to heighten accuracy, then blending the predictors (rather than switching between them) will offer better results. In fact, one of the first lessons to be learned from the Netflix prize is that greater accuracy can be achieved by blending a wide variety of predictors; the grand prize solutions combined hundreds of predictors in order to surpass the 10% improvement goal [Kor09b, TJB09, PC09].

In the interest of scalability, we therefore also explored a method that only tunes the kNN parameters. To do so, we first select a subset of potential k values $P \subset \mathbb{N}$. In this work, $P = \{0, 20, 35, 50\}$. We then proceed to set a value $k_{u,t} \in P$ for each user u at time t. When new users enter the system, their $k_{u,t}$ value is bootstrapped to a pre-determined member of P. The idea is for each $k_{u,t}$ to be set to



(a) Adaptive kNN (b) User k-Value Proportions (c) User Change Proportions

Figure 4.11: Time-Averaged RMSE Comparing $k = 50$ and Adaptive ($k = \alpha$) kNN, Proportions of Users Who Selected Each k Value Over Time, and Proportions of Users whose k Value Changed At Each Interval

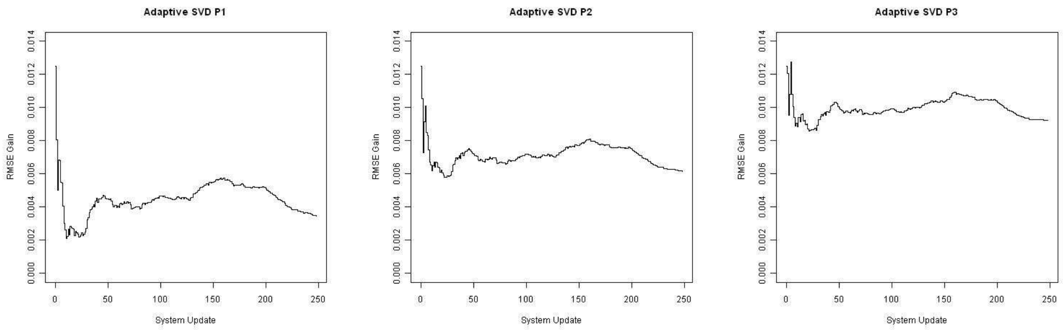
that which would have provided the steepest improvement on the users' $e_{u,t}$ value in the last time step, just as shown in Equation 4.3:

$$\forall u : k_{u,t+1} = \max_{k \in P} \left(e_{u,t} - RMSE_{u,t} \right) \tag{4.4}$$

It is important to note that this parameter update method is independent of the particular flavour of kNN that is implemented. In other words, it is equally applicable to both the user-based and item-based approaches; for example, if the item-based approach is implemented (as we have experimented with above), then a prediction $\hat{r}_{u,i}$ of item i for user u is done by aggregating ratings by k similar *items*. It could also be applied to the user-based approach, where predictions would aggregate ratings from k similar *users*. We still aim to optimise performance on a per-user basis.

The results are plotted in Figure 4.11. Figure 4.11(a) compares the five-fold cross validated time-averaged RMSE results of the best *global* parameter setting ($k = 50$) and the adaptive technique. The results highlight a number of benefits of adaptive CF. In particular, the adaptive strategy at first rivals the performance of $k = 50$, but then improves the overall time-averaged RMSE, without requiring any manual parameter tuning. In these runs we opted for the bootstrapping setting to be $k = 50$, since it performed *worst* when predicting users with very small profiles, as plotted in Figure 4.9(b) (we thus are considering a worst case scenario). It will thus tend to disadvantage new entrants to the system; however, we still can see an improvement in temporal accuracy. Changes to the bootstrapping value affected the first number of updates, but, after a number of updates, all the values we tested differed in performance by less than 0.001; they all outperformed $k = 50$.

To explore how the different parameter settings are distributed amongst members of the system over time, we plotted the proportions of current users who have adopted each setting, shown in Figure 4.11(b). From this, we see that users do not converge to a single parameter and moreover, the dominant strategy (selected by up to 40% of the current users) is the baseline item mean, followed by $k = 50$, the user mean, $k = 20$, and lastly 35. The most selected method, when operating alone, was consistently outperformed by all other k values. However, it plays an important role in providing greater temporal



(a) $P_1 \in (56, 64, 72, 80, 88, 96)$ (b) $P_2 \in (72, 80, 88, 96)$ (c) $P_3 \in (88, 96)$

Figure 4.12: Time-Averaged RMSE Gain of Adaptive-SVD With Different Subsets of Parameters

accuracy to the adaptive case.

The method we have outlined allows the k value for each user to be updated at *every* interval. The k value is changed if a different value would have yielded better predictions at the current time; it is possible, therefore, that this k value would continuously fluctuate without finding a stable value. To explore this possibility, we graphed the proportion of users who *change* neighbourhood size over time in Figure 4.11(c), and found that only a very small proportion of the user neighbourhood sizes are being changed at any given update. On average, only 2% of the current users change neighbourhood size; at most, 6% adopt a new size for the next interval. While this does not imply that users are converging and remaining on the optimal strategy, it highlights the proportion of users with parameters *not* set to the best member of P.

The improved accuracy of adaptive-kNN comes at little cost: the computational overhead is minimal. The cost of computing predictions remains the same, since, for example, the computations for both the $k = 20$ and 35 predictions for a user-item pair are contained within those required to compute $k = 50$. User profiles need to be augmented to include e_i, the error achieved to date, and a set of error values that each k has achieved in the current time step.

While the notion of adaptive-CF has been applied here to *temporal* collaborative filtering, it can also be applied to the static case. In the latter context, the problem is that of determining appropriate k values in a single step. We leave a full analysis of adaptive CF in the static case as a topic of future work; however, here we explore the potential for improvement by reporting the results of the *optimal* case. Given $P = \{0, 20, 35, 50\}$, if we select the optimal parameter setting for each user (assuming full knowledge of the RMSE each method produces for each user), the probe RMSE would be 0.8158. This error lies below the threshold for the Netflix prize, and is achieved by adaptively selecting from 5 techniques that *alone come nowhere close to this mark*. Furthermore, there is no single method that dominates over the others: 22% select $k = 20$, 12% opt for $k = 35$, 14% select $k = 50$, 24% the item mean, and 26% the user mean rating. Interestingly, the two baseline (mean rating) based methods together compose half of the users in the dataset.

4.3.3 Adaptive SVD

In the previous section, we showed that the neighbourhood size parameter k can be selected from a predefined subset of candidates and updated over time in order to improve the system's time-averaged performance. A natural question to ask is whether this technique is bound to how the kNN algorithm works, and whether the general principle of parameter update based on temporal performance can be applied to other CF algorithms as well.

In order to investigate this question, we turned to SVD-based CF. As introduced in Chapter 2 (Section 2.2.4), SVDs are given a parameter f that denotes how many features will be used to describe the users and movies once they are projected to a lower dimensional space. While nonparametric versions of this algorithm have been explored recently [YZLG09], we focus on the family of SVDs that are initialised with a predefined value of f. In our case, we ran an experiment where $f = 96$. We also output all predictions for any f in $P \in \{56, 64, 72, 80, 88, 96\})$; a number of arbitrary parameters, selected so as to be evenly spaced from each other (they are, in fact, all multiples of eight). Note that we do not recompute the user and movie feature values with a new parameter, but simply output a number of predictions, where each uses a varying subset of the features computed with $f = 96$. We then repeat the same update process that we implemented above; this time, instead, we select (for each user) a future f value based on the one that is currently performing best:

$$\forall u : f_{u,t+1} = \max_{f \in P} (e_{u,t} - RMSE_{u,t}) \tag{4.5}$$

In this case, we take our baseline to be the predictions computed using the full (96) user and movie features, since any hybrid switching approach will select to move away from the full feature matrix toward lower valued parameters. We also varied the range of f values we allowed in the full set P, in order to test the effect of excluding the smaller members of P. We tried $P_1 \in \{56, 64, 72, 80, 88, 96\})$, $P_2 \in \{72, 80, 88, 96\})$, $P_3 \in \{88, 96\})$: the results from these three experiments are plotted in Figure 4.12. In order to highlight how much we gain from the baseline, we plotted the difference between the baseline and each method's time-averaged RMSE. As with the kNN in the previous section, all P_i consistently improve the time-averaged RMSE of the baseline. However, we observe in this case that broadening the range of available f values does not always help. In fact, the group that achieves the highest gain from the baseline is the one with only two f candidates.

4.4 Related Work

Adaptive-CF differs from hybrid methods since, rather than focusing on merging different predictive models, individual methods are selected based on current performance. To that extent, adaptive-CF is independent of the particular set of selected classifiers that it alternates between, and falls under the broader category of available meta-learners [VD02], although we strictly consider the temporal scenario. It is therefore also possible to widen the set of choices available in order to further improve accuracy; for example, some users' ratings may be best predicted by performing a SVD with a varying number of features. We have not included this possibility here since doing so may well also introduce the potentially prohibitive cost of computing many models in a deployed system.

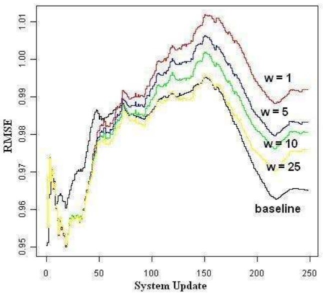


Figure 4.13: Time-Averaged RMSE of kNN With Limited History

Previous work that highlights the importance of time in CF (and in related fields, such as information retrieval [AG06]) tends to focus on the data, rather than the sequential application of an algorithm. For example, Potter [Pot08] (whose bias model we explored above) and Bell & Koren [BK07] also consider the temporal nature of ratings, by looking at the variability of individual user ratings across different days of the week in order to improve predictive performance. Temporality has also been explored from the point of view of changing user tastes [Kor09a, DL05]; in this case, ratings are scaled according to when they were input. The aim is to capture the most *relevant* ratings that represent current user tastes.

Both our adaptive-CF and this method could be merged; in this work we focus on the algorithm rather than modifying the set of ratings we train with. However, there are a number of questions to be addressed in future research. One of them is the influence of the update interval μ. In this section, we highlight a different example: the balance between time-averaged accuracy and how long the ratings that are being trained with have been in the system.

We repeated our temporal kNN experiments, but limited the algorithm to computing item neighbours using only ratings that were input within the last $w \in \{1, 5, 10, 25, 50\}$ updates. In other words, if $w = 1$, then item similarity is computed using only the ratings input in the previous week; a potential majority of the ratings are excluded. Any ratings input before the allowed 'window' were only used to compute item means. The results, along with the baseline (where all historical data is used), are plotted in Figure 4.13. The figure can be roughly divided into three sections: in the beginning, the baseline performance degrades over time. Then, after a period where the baseline is relatively flat, performance improves for majority of the final updates. During the period where performance degrades, all of the limited/windowed kNNs are more accurate. However, when the baseline performance begins improving, the baseline overtakes the windowed kNNs, although the $w = 50$ is remarkably close (considering the difference in data that each method has available). Just as we found a relation between the *accuracy* and the CF *algorithm*, in order to design our hybrid method, there is also a relationship between *accuracy* and the *data*.

4.5 Summary

This chapter departs from traditional CF research by extending the analysis of prediction performance to incorporate a sequence of classification iterations that learn from a growing (and changing) set of ratings.

The contributions we made can be summarised as follows:

- **Methodology and Metrics**. We defined a novel approach with which to examine CF's temporal predictions, using three variations (continuous, sequential, windowed) of accuracy metrics.
- **Evaluation of State-of-the-Art Algorithms**. We ran a variety of experiments that evaluated CF algorithms' temporal accuracy from two perspectives, and highlighted the *variability* of both static and dynamic sets of predictions as training sets are augmented with new ratings.
- **Adaptive Algorithms for Improved Temporal Accuracy**. We implemented and evaluated two adaptive algorithms that improve temporal accuracy over time: a method to switch between CF algorithms and a computationally cheap technique to automatically tune parameters to provide greater temporal accuracy.

The focus of this chapter has revolved around optimising recommender system prediction performance from the point of view of RMSE. The results show that these algorithms do not output consistent error, and it becomes difficult to claim that one algorithm outpredicts another when only a static case is investigated (and especially when the static difference in performance is relatively small). For example, the bias model was more accurate than raw-data kNN on the Netflix probe, but did not maintain this advantage when a range of datasets (of varying size) were tested in an iterative set of cross-validated experiments. We have focused on the temporal performance of CF algorithms, without considering (a) the extent to which user preferences and interests will vary over large time intervals, and (b) the temporal effect of malicious ratings [MBW07]. In particular, as the experiments in Section 4.2 highlight, performance does not necessarily improve as the available training data grows.

However, this observation also motivates research that departs from traditional mean-error based evaluations of CF algorithms. These kinds of evaluations aim to explore the *ranking* that emerges from rating prediction, and the utility that users draw from the lists of recommendations they are offered. A further evaluation of temporal CF would therefore also encompass the variation in recommendations that results from the changing data; we begin in the following chapter by shifting our focus to CF's temporal diversity.

Chapter 5

Temporal Diversity in Recommender Systems

As we explored in the previous chapter, CF algorithms are often evaluated according to how *accurately* they predict user ratings [HKTR04]. However, as recommender systems grow dynamically, a problem arises: current evaluation techniques do not investigate the temporal characteristics of the produced *recommendations*. Researchers have no means of knowing whether, for example, the system recommends *the same items* to users over and over again, or whether the most *novel* content is finding its way into recommendations. The danger here is that, as results may begin to stagnate, users may lose interest in interacting with the recommender system.

In this chapter, we investigate one dimension of temporal recommendations: the diversity of recommendation lists over time. We first examine why temporal diversity may be important in recommender system research (Section 5.1) by considering temporal *rating patterns* and the results of an extensive user survey. Based on these observations, we evaluate three CF algorithms' temporal diversity from 3 perspectives (Section 5.2): by comparing the intersection of sequential top-N lists, by examining how diversity is affected by the number of ratings that users input, and by weighting-in the trade-off between accuracy and diversity over time. We finally *design* and *evaluate* a mechanism to promote temporal diversity (Section 5.3), comparing its performance to a range of baseline techniques. We conclude in Section 5.4 by discussing future research directions.

5.1 Why Temporal Diversity?

We explore the importance of temporal diversity from two perspectives: changes that CF data undergoes over time (Section 5.1.1) and how surveyed users responded to recommendations with varying levels of diversity (Section 5.1.2).

5.1.1 Changes Over Time

In Chapter 3, we performed an extensive analysis of how three rating datasets changed over time. In this section, we briefly summarise the main conclusions of this analysis, and how they relate to the direction we examine in this chapter.

1. **Data Growth.** Recommender systems grow over time: new users join the system and new content is added as it is released. Pre-existing users can update their profiles by rating previously unrated content; the overall volume of *data* thus grows over time. Section 3.2.1 shows the movie and user

growth across rating datasets; from these we see that there is a continuous arrival of both new users and movies.

2. **Summary Statistics Change.** As a consequence of the continuous influx of ratings, any summary *statistics* related to the recommender system's content may also change. These changes affect the ratings' summary statistics: in [Kor09a], Koren shows how *global* summary statistics vary over time. Similarly, Section 3.2.2 looks at how changes are further reflected in the global rating mean, median and mode. All the summary values fluctuate over time, reflecting how the overall distribution of ratings shifts as more users interact with the system.
3. **User Interaction.** Lastly, Section 3.2.3 looked at the rating frequency: these plots show the high variability in how users interact with the recommender system. Some users appear more frequently than others, and there is a large variance in the volume of rated items.

What do we learn from observing these changes? The datasets do not only remain incredibly sparse, but they also do not stabilise; recommender systems continuously have to make decisions based on *incomplete* and *changing* data, and the range of the changes we observe in the Netflix data have a strong impact on the predictability of ratings. Furthermore, the continuous rating of content means that the data that an algorithm will be trained with at any particular time is likely to be different than data it trained with previously. The question we explore in this chapter is: does the influx of new data translate to new content being recommended?

5.1.2 User Survey

In order to determine whether temporal diversity is important for recommender system *users*, we designed three surveys that *simulate* systems that produce *popular movie* recommendations over the course of five "weeks." We opted to recommend popular movies in order to avoid a variety of confounds that would emerge had we selected a personalised CF algorithm (e.g., the quality of the algorithm itself and the cumbersome process of asking users to rate films). Survey 1 (S1) and Survey 2 (S2) both recommended popular movies drawn from a list of the 100 all time most profitable box office movies[1]. S1, however, had *no diversity*: it consistently recommended the top-10 box office hits. S2's recommendations, instead, did change over time. Each week, approximately seven of the previous week's ten recommendations would be replaced by other movies in the top-100 box office list. Lastly, Survey 3 (S3) recommended movies that were randomly selected from the Netflix dataset: the recommendation process included full diversity, but was very unlikely to recommend popular movies, given the size of the dataset.

Each survey was structured as follows[2]. The users were first queried for demographic data. They were then offered the first week's recommendations, represented as a list of ten movie titles (and the relative DVD covers and IMDB links) and asked to rate these top-10 recommendations on a 1-5 star scale. After submitting their rating, they were presented with a buffer screen containing thirty DVD covers, and had to click to continue to the subsequent week; this aimed at diverting users' attention before

[1] http://www.imdb.com/boxoffice/alltimegross

[2] Full description and reproduction of the surveys is found in Appendix A

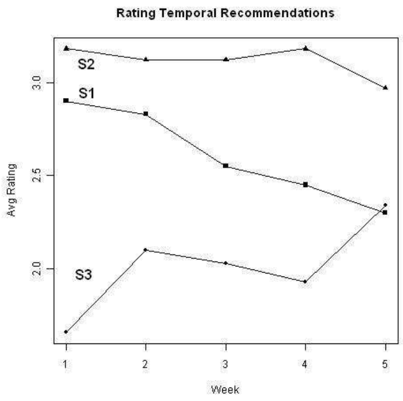


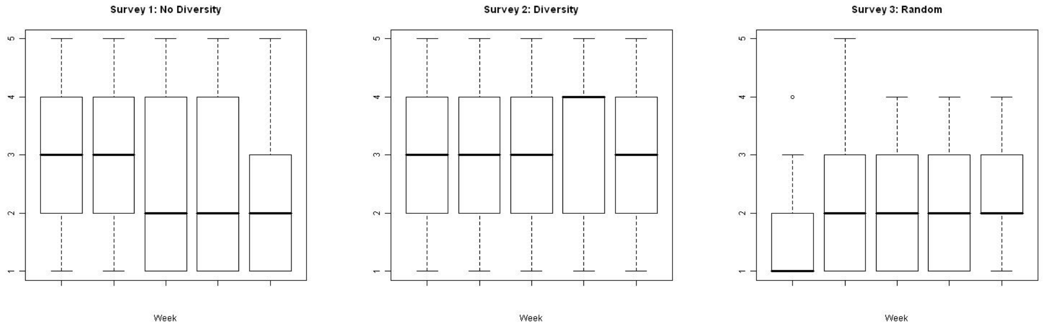
Figure 5.1: Survey Results for (S1) Popular Movies With No Diversity (S2) Popular Movies With Diversity and (S3) Randomly Selected Movies

presenting them with the next week's recommendations. After rating all five week's worth of recommendations, they were asked to comment on the recommendations themselves and answer a number of questions relating to diversity over time. Users were invited to participate in one or more of the surveys via departmental mailing lists and posts on social networks: S1 was completed 41 times, S2 had 34 responses, and S3 was completed 29 times. Due to the surveys' anonymity, we do not know how many users completed more than one survey. We therefore treat each completed survey individually. Of the 104 total responses, 74% of the users were male, 10% were 18-21 years old, 66% were 22-30 years old, and 24% were between 31 and 50 years of age. On average, the users claimed to watch 6.01 ± 6.12 movies per month, and while 61% of them said they were familiar with recommender systems, over half of them claimed they used them less than once a month. On the other hand, 29% use recommender systems weekly or daily: our respondents therefore include a wide variety of movie enthusiasts and both people who do and do not use recommender systems.

We averaged the users' ratings for each week's recommendations and plot the results in Figure 5.1. The S2 results (popular movies with diversity) achieve the highest scores: on average, these five weeks of recommendations were rated 3.11 ± 0.08 stars. The low temporal standard deviation reflects the fact that the rating trend remains relatively flat; the average for each week is about 3 stars. S3's results (randomly selected movies), were consistently disliked: the average rating peaks at 2.34 stars for week 5. In fact, some users commented on the fact that recommendations "appeared to be very random," "varied wildly" and the system "avoid[ed] box office hits." The main result of our surveys is reflected in S1's results (popular movies with no diversity): as the same recommendations are offered week after week, the average ratings *monotonically* decrease. The average for week 1 was 2.9, which falls within the range of values measured in S2, while by week 5 the average score is 2.3, which is lower than the average score for the random movies that same week. Not all users commented on the lack of recommendations diversity; however, most of them modified their ratings for the recommendations as the lack of diversity persisted. This shows that when users rate they are not only expressing their tastes or preferences; they are also responding to the impression they have of the recommender system. In the

Week	P-Value	S1 vs. S2	S1 vs. S3	S2 vs. S3
1	3.209e-6	0.32	7.4e-5	5.7e-6
2	0.003699	0.3003	0.0277	0.0033
3	0.002241	0.0881	0.0881	0.0015
4	0.0006937	0.0302	0.0943	0.0005
5	0.04879	0.07	0.88	0.10

Table 5.1: ANOVA P-Values and Pairwise T-Test Values For The 5 Weeks



(a) Survey 1

(b) Survey 2

(c) Survey 3

Figure 5.2: Boxplots of Each Week’s Ratings for the Three Surveys

case of S1, users commented on the fact that the algorithm was “too naive” or “not working,” and the lack of diversity “decreased [the respondent’s] interest.”

The final part of the surveys asked users about qualities they sought in recommendations: they had to give a rating reflecting how important they believed that accuracy, diversity, and novelty are in their recommendations. Overall, 74% said it was important for recommender systems to provide results that *accurately* matched their taste (23% selected the ‘neutral’ option to this question). 86% said it is important for recommendations to *change over time*; in fact, 95% stated it is important that they are offered new recommendations. It thus quickly becomes apparent that temporal diversity is a highly important facet of recommender systems, both in terms of the direct responses and rating behaviour of the surveyed users. In the following sections, we evaluate the temporal diversity of three state of the art CF algorithms.

Analysis of Variance

In order to test the statistical significance of the three different survey’s results, we performed an analysis of variance (ANOVA): in this case, the null hypothesis is that the ratings for each survey are of the same distribution. We can reject the null hypothesis with 99% confidence with p-values less than 0.01: the p-value we measured for the three methods is 9.72e -14. A pairwise t-test between each survey further shows that the ratings input for each survey cannot be attributed to the sampling of the study.

We also performed an ANOVA on each week’s data, in order to observe the change that S1 went through as it was rated. The null hypothesis, in this case, would state that the differences in ratings are consistent throughout all weeks; the p-value tells us what the chances are of randomly sampled users providing the ratings we have collected. The p-values are shown in Table 5.1; they all remain lower than the 95% confidence threshold. The table also shows pairwise t-test p-value results, week by week. We expected to observe three patterns: (a) a sequence of values that reflected S1 and S2 *diverging* from each other (as users punish S1 for having no diversity), (b) a sequence of values reflecting a convergence between S2 and S3, as the survey with no diversity becomes rated as highly as the random recommendations, and (c) values depicting a continuing difference between S2 and S3. In other words, given the rating distributions of S2 and S3, that are different from each other, we expected to observe S1's distribution change from being similar to S2's to being similar to S3's. The boxplots in Figure 5.2 reflect these changes: they show the per-week distribution of ratings for each survey. Most notably, the distributions in S2 (Figure 5.2(b)) remain relatively consistent over the weeks and S1 (Figure 5.2(a)) decreases over time.

Over the first four weeks of data, we observed the pattern we expected. The p-value between S1 and S2 begins high (0.32) and monotonically decreases until week four (0.03). The p-value between S1 and S3, instead, begins low (7.4e-5) and monotonically increases to 0.09. Lastly, the p-value shared between S2 and S3 remains consistently below 0.01 (the 99% confidence threshold). However, in week five there is an abrupt change in our results: not only is the overall p-value now found at the border of the 95% confidence threshold, but all t-test values become very high. Most worryingly, the divergence between S1 and S2 is no more, and S2 and S3 now share a very high p-value. There are a number of factors that may have skewed our experimental results. The first relates to why S3 may have been rated more positively in week 5:

- **Random Recommendations.** Survey 3 recommended random movies; however, as the surveys were pregenerated, all users were given the same (albeit random) recommendations in S3. One of the potential problems here is that this particular set of selected movies may elicit the same response from many users. In other words, the factors that influence users' responses (i.e., selection of movies) remained constant to all users. In week five, we noted that some of the randomly selected movies were less unknown than movies selected in previous weeks (for example, the first recommendation was the movie Crash[3]). The problem here is that S3's distribution in week five (as can be observed in Figure 5.2(c)) changes, thus impacting any comparisons between the distributions of the three surveys.
- **Edge Effect.** The survey instructions told users that they would be rating five weeks' recommendations. By the time they reached week 5, they thus knew that they had nearly completed the survey—their rating behaviour may have changed at this point. Similarly, they may also be responding more positively in week 5 due to the heavily negative response that was given for week 4. If we consider the rating mode of each week, then S3's results show that the weeks that receive a majority of 1* ratings (week 1 and week 4) are always followed by a week with a higher mode.
- **Lack of Data.** One of the reasons we may be seeing these results is the fact that our surveys have not been answered by many people. In fact, one of the problems that we noted was that

[3] http://www.imdb.com/title/tt0375679/

> many users would abandon the surveys, possibly since, as other users noted, they did not like the recommendations (or thought the system was not working). S1 was visisted 122 times, yet only completed 41 times: overall, all the surveys were fully completed by fewer than 1 in 2 visitors.

There are thus a number of factors that influence our ability to observe a five week trend when comparing the three surveys. However, these do not detract from the main result of the survey: users who are faced with non-changing recommendations tend not to only lose interest, but also to reflect their impatience with the system in the ratings that they input. Temporal diversity is therefore an important quality that all recommender systems should provide.

5.2 Evaluating for Diversity

Given the above, we now aim to examine how diverse CF algorithms are over time. We focus on CF algorithms; a *baseline*, where a prediction for an item is that item's mean rating, the *item-based k-Nearest Neighbour (kNN)* algorithm, and a *matrix factorisation* approach based on Singular Value Decomposition (SVD), as reviewed in Chapter 2. We chose these algorithms since they not only reflect state-of-the-art CF, but also each manipulate the rating data in a different way and may thus produce varying recommendations.

5.2.1 From Predictions to Rankings

All of the algorithms share a common theme: they produce predicted ratings that can then be used to recommend content. The idea is to use the predictions in order to generate a personalised ranking of the system's content for each user. However, it may be the case that items share the same predicted rating. For example, a number of items may all have 5-star predictions. In this case, the predicted rating alone is not conducive to a meaningful ranking. We solve this problem by introducing a *scoring function* to rank items, regardless of the model used to generate predicted ratings. The scoring function uses two pieces of information: the predicted rating, and the *confidence* in the prediction (i.e., number of data points used to derive it) as used in [MLG$^+$03]. Assuming a 5-star rating scale, we first subtract the scale mid-point (3 stars) from the prediction and then multiply by the confidence:

$$s_{u,i} = (\hat{r}_{u,i} - 3.0) \times confidence(\hat{r}_{u,i}) \tag{5.1}$$

This scoring function ensures that items with high prediction and confidence are promoted, and low prediction with high confidence are demoted (i.e., we use it on all predictions and not simply as a tie-breaker). For example, an item with a predicted 5 star rating, derived from 2 ratings, will be ranked lower than another item with a 4 star prediction based on 50 ratings. If two items had the same score, then we differentiated them based on their respective average rating date: the item that had been rated more recently is ranked higher. The greatest advantage of this method, as detailed in [MLG$^+$03], is the heightened *explainability* of recommendations.

5.2.2 Methodology

In order to examine the sequence of recommendations produced by a system, we explore CF algorithms that iteratively re-train on a *growing dataset*. Given a dataset at time t and a window size μ (how often the system will be updated), we train the algorithm with any data input prior to t and then *predict* and *rank all* of the unrated items for each user. The t variable is then incremented by μ, and the entire process is repeated, except that now the latest ratings become incorporated into the training data. In other words, at time t we generate a set of top-N lists—corresponding to the top-N recommendations each user would receive—in order to examine how the sequence of ranked items that we produce will vary as the system is updated. The main difference between this process and the one we used in Chapter 4 is that we predict *all* unrated items for each user (rather than only predicting items that will be rated in the next time window); this allows us to produce ranked lists of recommendations. This method includes a number of advantages: we test the algorithms as data grows (and view more than a single iteration of this process), making predictions based only on ratings that are currently available. We simulate the iterative update of deployed systems, and stay true to the order users input ratings.

Since users do not necessarily log-in consistently to the system, we cannot be certain that each top-N list would have been viewed by each user. We therefore only generate a top-N list for the users who will rate at least one item in time $(t + \mu)$; we assume that if the user is rating an item then they have logged into the system and are likely to have seen their recommendations. The benefit of this is that we compare the current recommendations to those that users are likely to have seen before. It remains possible that users viewed their recommendations without rating any content; however, given this uncertainty in the data, we only consider the scenario where there is evidence that the users have interacted with the system. In this chapter, we continue using the same 5 subsamples of the Netflix dataset (as used in Chapter 4) for cross-validation purposes.

5.2.3 Measuring Diversity Over Time

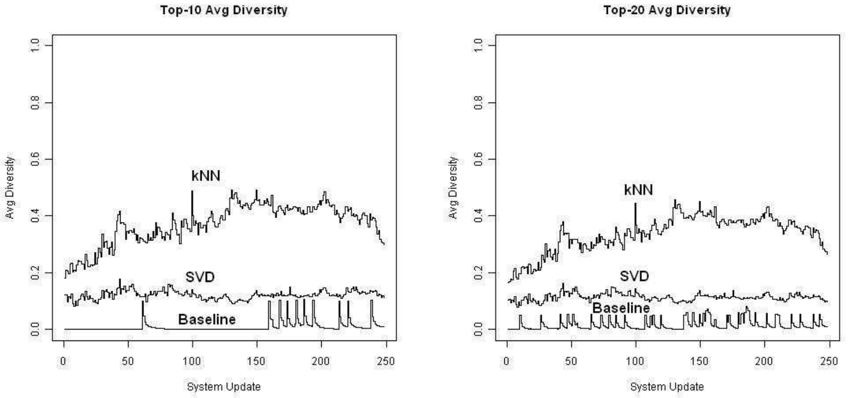
We define a way to measure the diversity between two ranked lists as follows. Assume that, at time t, a user is offered a set of 10 recommendations. The next time the user interacts with the system only 1 of the 10 recommendations is different. Therefore, the diversity between the two lists is $\frac{1}{10} = 0.1$. More formally, given two sets L_1 and L_2, the set theoretic difference (or relative complement) of the sets denotes the members of L_2 that are not in L_1:

$$L_2 \backslash L_1 = \{x \in L_2 | x \notin L_1\} \tag{5.2}$$

In our example above, only 1 of the 10 recommendations was not the same: the set theoretic difference of the two recommendation lists has size 1. We thus define the diversity between two lists (at depth N) as the size of their set theoretic difference divided by N:

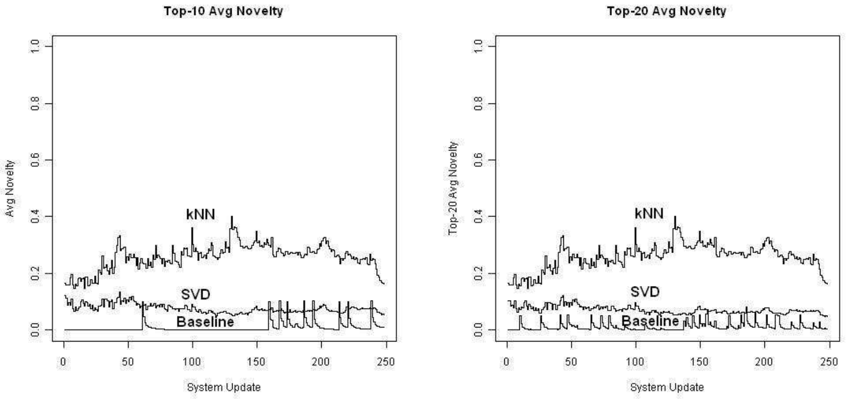
$$diversity(L_1, L_2, N) = \frac{|L_2 \backslash L_1|}{N} \tag{5.3}$$

If L_1 and L_2 are exactly the same, there is no diversity: $diversity(L_1, L_2, N) = 0$. If the lists are completely different, then $diversity(L_1, L_2, N) = 1$. This measure disregards the actual ordering of the items: if a pair of lists are re-shuffled copies of each other, then there continues to be no diversity. However, we can measure the extent that recommendations change as a result of the same content being promoted or demoted by measuring diversity at varying depths (N).



(a) Top-10 Diversity (b) Top-20 Diversity

Figure 5.3: Top-10 and 20 Temporal Diversity for Baseline, kNN and SVD CF



(a) Top-10 Novelty (b) Top-20 Novelty

Figure 5.4: Top-10 and 20 Temporal Novelty for Baseline, kNN and SVD CF

One of the limitations of this metric is that it measures the diversity between two lists only; it thus highlights the extent that users are being *sequentially* offerered the same recommendations. In order to see how recommendations change, in terms of *new* items appearing in the lists, we define a top-N list's *novelty*. Rather than, as above, comparing the current list L_2 to the previous list L_1, we compare it to the set of *all* items that have been recommended to date (A_t):

$$novelty(L_1, N) = \frac{|L_1 \backslash A_t|}{N} \tag{5.4}$$

In this case, a list's novelty will be high if all of the items have *never* been recommended before, and low if all of the items have been recommended at some point in the past (not just in the last update). We further define the *average diversity* δ_t and *average novelty* η_t that is generated by a given CF algorithm (at time t) as the average of the values computed between all the current top-N lists and the respective previous list for all users.

5.2.4 Results and Analysis

We computed δ_t and η_t for each of the 3 algorithms over all 249 simulated system updates outlined in Section 5.2.2, and plotted the results for the top-10 and top-20 recommendations in Figure 5.3. These results provide a number of insights into recommender system temporal diversity. As expected, the baseline algorithm produces little to no diversity. On average, users' top-10 recommendations differ by (at most) one item compared to the previous recommendations. Both the factorisation and nearest neighbour approaches increment diversity; furthermore, the kNN algorithm is, on average, consistently more diverse than the sequence of recommendations produced by the SVD.

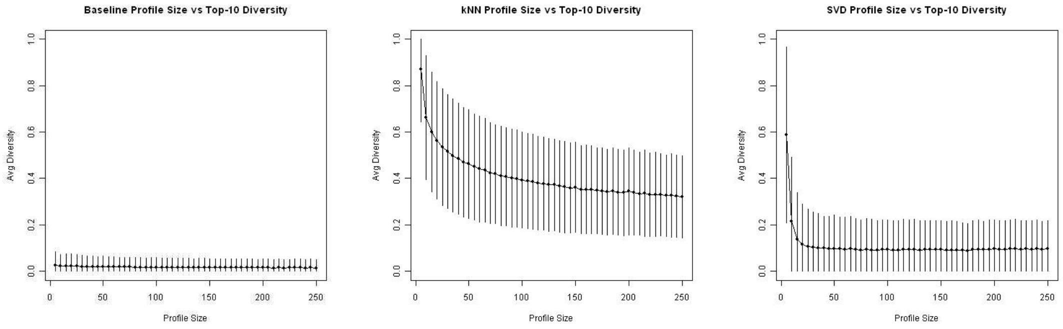
The novelty values (Figures 5.4(a) and 5.4(b)) are lower than the average diversity values. This means that, when a different recommendation appears, it is more often a recommendation that has appeared at some point in the past, rather than something that has not appeared before. There are a variety of factors that may cause this; for example, new items may not be recommended because they lack sufficient ratings: the CF algorithm cannot *confidently* recommend them. However, this metric does not tell us whether the new recommendations are new items to the system, or simply content that has (to date) not been recommended. A full analysis of the novelty of recommendation warrants a closer inspection of when items join the system and when they are recommened. In order to focus our analysis, we thus separate the problems of recommending *new content* from that of *diversifying* sequential recommendations: in this chapter, we focus on the latter.

Both Figure 5.3(a) and 5.3(b) also look very similar: the diversity values for the top-10 and top-20 recommendations are nearly the same. In order for this to happen (i.e., for a comparison between two top-10 lists and two top-20 lists to produce the same value) there must be *more* diversity between the larger lists. For example, if only 1 item changes in the top-10, the diversity is $\frac{1}{10} = 0.1$, and the pair of top-20 lists will only produce this diversity value if 2 items have changed, $\frac{2}{20}$. What this means is that not all of the changes in the item rankings are occuring in the top-10: new items are also being ranked between the $11th$ and $20th$ positions.

At the broadest level, we thus observe that (a) both the baseline and SVD produce less temporal diversity than the kNN approach, and (b) across all CF algorithms, diversity is never higher than appproximately 0.4. However, these are averaged results across many users, who may be each behaving in very different ways: we now perform a finer grained analysis of temporal diversity to explore the relation between users and the diversity they experience.

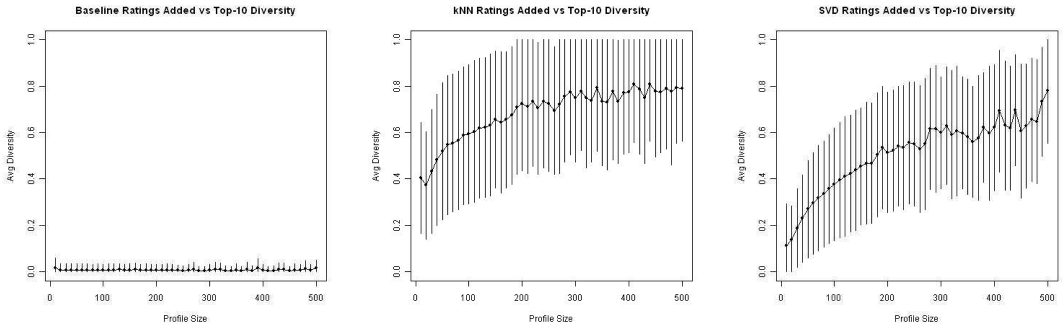
5.2.5 Diversity vs. Profile Size

The metric in Section 5.2.3 does not factor in the fact that the distribution of ratings per user is not uniform. Some users have rated a lot of items, while others have very sparse profiles. Users' *profile size* (i.e., the number of ratings per user) may affect their recommendation diversity. We thus binned the above temporal results according to users' current profile size and then averaged the diversity of each group. We plot the results in Figure 5.5. The baseline (Figure 5.5(a)) continues to show next to no diversity, regardless of how many items users have rated. The rationale behind this is that the only profile information that the baseline factors in when it computes recommendations is whether the user



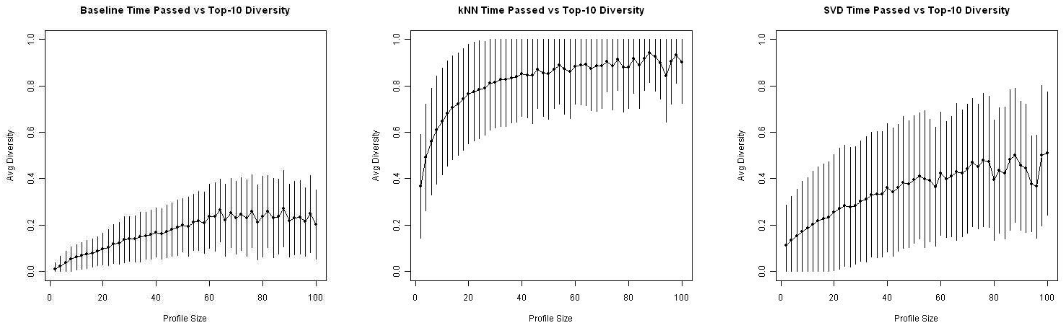
(a) Baseline (b) kNN (c) SVD

Figure 5.5: Profile Size vs. Top-10 Temporal Diversity for Baseline, kNN and SVD CF



(a) Baseline (b) kNN (c) SVD

Figure 5.6: Ratings Added vs. Top-10 Temporal Diversity for Baseline, kNN and SVD CF



(a) Baseline (b) kNN (c) SVD

Figure 5.7: Time Passed vs. Top-10 Temporal Diversity for Baseline, kNN and SVD CF

has rated one of the popular items; results will only be diverse if the user rates all the popular content. The kNN (Figure 5.5(b)) and SVD (Figure 5.5(c)) results, instead, show a negative trend: diversity tends to reduce as users' profile size increases. These results can be interpreted as follows: as users augment the set of ratings that represent their tastes, the breadth of items that are recommended to them via CF reduces, and they will be exposed to less and less new content.

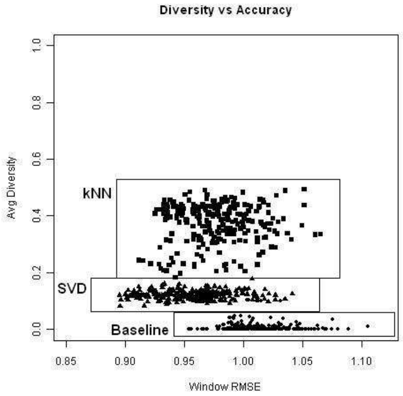


Figure 5.8: Comparing Accuracy with Diversity

5.2.6 Diversity vs. Ratings Input

Our temporal diversity metric is based on *pairwise* comparisons; we compare each sequential pair of top-N lists. One factor that may thus play an important role when determining how diverse a pair of lists will be from one another is *how much* the user rates in a given session. For example, one user may log in and rate two items while another may log in and rate fifty; the temporal diversity that each user subsequently experiences may be affected by these new ratings. We therefore binned users according to how many new ratings they input, and plot the results in Figure 5.6. As before, the baseline remains unaffected by how many new ratings each user inputs. The kNN (Figure 5.6(b)) and SVD (Figure 5.6(c)), instead, show a positive trend. These results can be interpreted as follows: the more you rate now, the more diverse your *next* recommendations will be.

5.2.7 Diversity and Time Between Sessions

The previous analysis was concerned with how diversity is influenced by a *single* user rating content. However, users do not rate alone: an entire community of users rate content over extended periods of time. We highlight this point with an example: some users may consistently log in and rate items every week; others may rate a few items now and not return for another month (and, in their absence, other users will have continued rating). In other words, diversity may be subject to the *time* that has passed from when one list and the next are served to the user. In order to verify this, we binned our diversity results according to the number of weeks that had passed between each pair of lists, and plot the results in Figure 5.7. In this case, all three of our algorithms show a positive trend: the longer the user does not return to the system, the more diversity increases. Even the baseline diversity increases: if a user does not enter the system for a protracted period of time, the popular content will have changed. However, web businesses tend to use recommender systems to *increase* user engagement and activity (e.g. clickthroughs, rented movies), and the natural diversification of recommendations because of time will only be useful for the least active members of the system.

5.2.8 Lessons Learned

Overall, *average temporal diversity* is low. Ranking content based on popularity offers next to no diversity, while the kNN method produces the largest average temporal diversity. Larger *profile sizes* negatively affect diversity; it seems that users who have already rated extensively will see the least diverse recommendations over time. Pairwise diversity between sequential lists is largest when users rate many items before receiving their next recommendations; users should be encouraged to rate in order to change what they will be recommended next. Diversity will naturally improve as users extend the *time between* sessions when they interact with the system (even popular content eventually changes).

A fundamental question to ask is how diversity relates to accuracy, the metric of choice in traditional CF research. To do so, we take the predictions we made at each update, and compute the Root Mean Squared Error (RMSE) between them and the ratings the visiting users will input. We then plot RMSE against average diversity in Figure 5.8. A plot of this kind has four distinct regions: low accuracy with low diversity (bottom right), high accuracy with low diversity (bottom left), low accuracy with high diversity (top right), and high accuracy with high diversity (top left). We find that the results for each algorithm cluster into different regions of the plot, corresponding to the different diversity results that they obtain. In terms of RMSE, different algorithms often overlap; for example, the kNN results sit between the two others—in terms of accuracy—and above them when considering diversity. However, kNN CF is sometimes less accurate than the baseline. The baseline sits toward the bottom right of the plot: it offers neither accuracy nor diversity. The SVD, on the other hand, tends to be more accurate than the baseline, although there is little diversity gain.

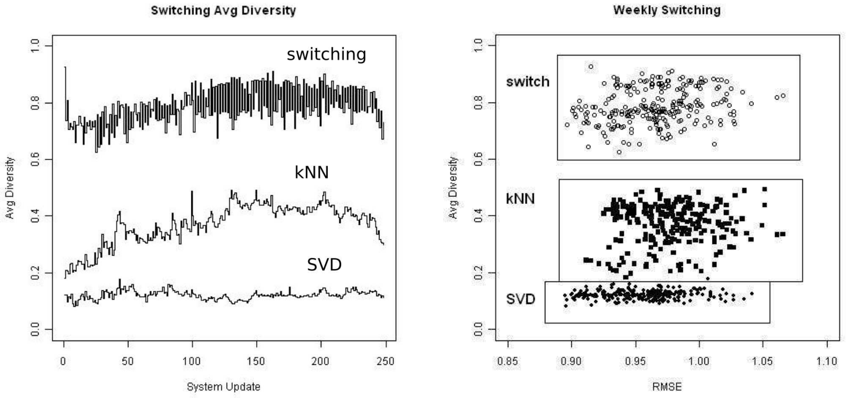
Coupling the low diversity that we have observed in CF algorithms and the high importance users place on temporally diverse recommendations implies that improving the temporal diversity of a recommender system is an important task for system developers. In the following section, we describe and evaluate a number of techniques that meet this goal: they increase temporally diversity, without significantly impacting recommendation accuracy. We then discuss the potential implications that modifying top-N lists may have to promote diversity.

5.3 Promoting Temporal Diversity

The easiest way of ensuring the recommendations will be diverse is to do away with predicted ratings and simply rank items randomly. However, *diversity* then comes at the cost of *accuracy*: recommendations are no longer personalised to users' tastes. The random survey (Section 5.1.2) showed that this is not a viable option, since the recommendations were rated very low. We can thus anticipate that, when promoting diversity, we must continue to take into account users' preferences. We do so with two methods: temporal hybrid switching, from a system (Section 5.3.1) and user (Section 5.3.2) perspective, and re-ranking individual users' recommendations (Section 5.3.3).

5.3.1 Temporal Switching

Many state of the art approaches to CF combine a variety of algorithms in order to bolster prediction accuracy [AT05]. However, as described by Burke [Bur02], another approach to building hybrid CF



(a) Switching Temporal Diversity (b) Switching RMSE vs Diversity

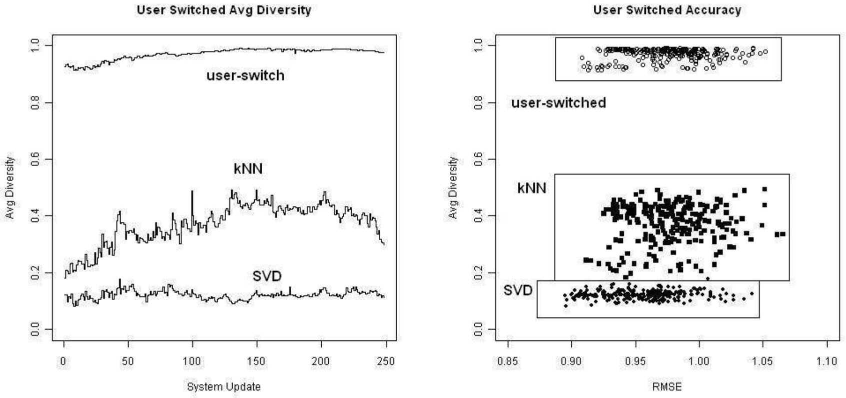
Figure 5.9: Diversity (a) and Accuracy (b) of Temporal Switching Method

algorithms is to *switch* between them. Instead of combining prediction output, a mechanism is defined to select one of them. The rationale behind this approach is as follows: a given a set of CF algorithms, that each operate in a different way, are likely to produce *different* recommendations for the same user; the top-N produced by a kNN may not be the same as that produced by an SVD. We thus switch between the two algorithms: we cycle between giving users kNN-based recommendations one week, and SVD-based recommendations the following week.

We plot the top-10 diversity over time for this switching method in Figure 5.9(a). Diversity has now been incremented to approximately 0.8: on average, 8 of the top-10 recommendations ranked for each user is something that was not recommended the week before. How does this affect accuracy? Intuitively, the overall accuracy that the system will achieve will be somewhere between the accuracy of each individual algorithm. We compare the accuracy and diversity of our switching technique in Figure 5.9(b). The results for the switching method now cluster into two groups; each group lies *above* the candidate algorithms we selected. In other words, accuracy fluctuates between the values we reported for kNN and SVD CF, but the fact that we are switching between these two techniques ensures that diversity has been greatly increased.

5.3.2 Temporal User-Based Switching

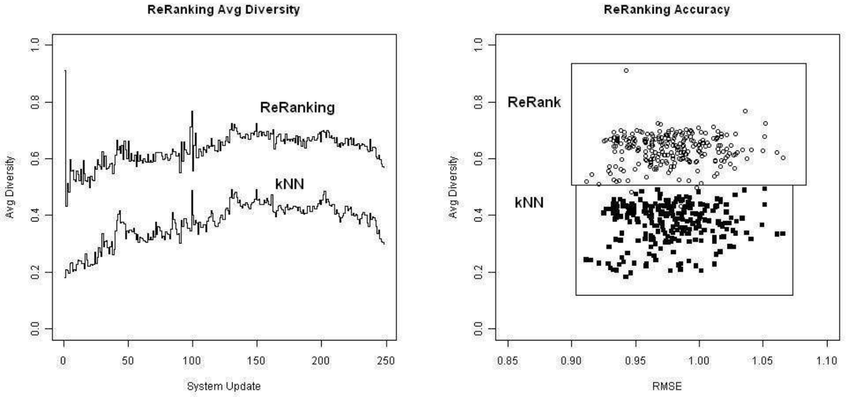
The method described in the previous section is very straightforward: the system changes the CF algorithm that is used from one week to the next in order to favour diversity. However, this method does not take into account how users behave; in particular, we previously noted that not all users have *regular* sessions with the recommender system. In fact, if their sessions were every other week, then the switching technique described in the previous section would be of no use at all. We therefore also tested a user-based switching algorithm. It works as follows: the system keeps track of *when* a user last appeared, and *what* algorithm was used to recommend content to that user during the last session. When the user reappears, the system simply picks a different algorithm to that which it used previously. As before, we switched between using an item-based kNN and an SVD-based approach in our experiments. The results are shown in Figure 5.10.



(a) Temporal Diversity

(b) RMSE vs. Diversity

Figure 5.10: Temporal Diversity and Accuracy vs. Diversity With User-Based Temporal Switching



(a) Temporal Diversity

(b) RMSE vs. Diversity

Figure 5.11: Temporal Diversity and Accuracy vs. Diversity When Re-Ranking Frequent Visitors' Lists

The temporal diversity (Figure 5.10(a)) is now near 1: on average, users are being offered different recommendations to those that they were shown the last time they interacted with the system. On the other hand, accuracy (Figure 5.10(b)) now falls between the kNN and SVD results. In other words, we sacrifice the low-RMSE of the SVD, but still do better than simply using the kNN approach: in return, the average diversity has been greatly amplified.

The only overhead imposed by user-based switching is a single value per user that identifies which algorithm was last used to compute recommendations; however, unlike the temporal switching method in the previous section, we are now required to compute both kNN and SVD at every update, albeit for a subset of users. We do not consider this to be an unsurmountable overhead, given that state of the art algorithms already tend to ensemble the results of multiple CF algorithms.

5.3.3 Re-Ranking Frequent Visitors' Lists

An immediate problem with a temporal switching approach is that it requires multiple CF algorithm implementations. In this section, we provide a means of diversifying recommendations to any desired degree of diversity when only a single CF algorithm is used.

One of the observations we made above is that users who have very regular sessions with the recommender system have low top-N temporal diversity. One way of improving overall average temporal diversity thus entails catering to the diversity needs of this group. To do so, we take advantage of the fact that they are *regular* visitors, and only re-rank their top-N recommendations.

The re-ranking works in a very straightforward manner: given a list that we wish to diversify with depth N (e.g., N = 10), we select M, with $N < M$ (e.g., M = 20). Then, in order to introduce diversity d into the top-N, we replace $(d \times N)$ items in the top-N with randomly selected items from positions $[(N+1)...M]$. In the case of $d = 1$, all elements in the first $[1...N]$ positions are replaced with elements from positions $[(N+1)...M]$. This is the method that we used to diversify the recommendations in the user survey S2 (Section 5.1.2); in that case, $N = 10$ and $M = 100$ (the 100 all time box office hits).

In our experiments, we opted to re-rank the top-10 results for any users who had previously visited the system less than two weeks before (recall that our system is updated weekly). The temporal diversity results, shown in Figure 5.11(a), clearly improves the overall average. Furthermore, the accuracy (Figure 5.11(b)) remains the same: the diversity has simply been shifted in the positive direction. However, how does this not hurt accuracy? There are three points to keep in mind: (a) we are only reranking the lists for frequent visitors, others' recommendations are untouched; (b) the items in the top-N are there due to both high prediction value and high confidence (there is a good chance the user will like those items); and (c) we do not promote items that are likely to be disliked by the user (by only re-ranking the top-M).

How do these techniques affect recommendation novelty? Recall that we defined novelty (Section 5.2.3) as proportional to the number of items being recommended that have *never* been recommended before. If we aggregate the temporal results of Figure 5.4(a), we find that the baseline top-10 recommends, on average, 13.53 ± 2.86 items over time; the SVD top-10 suggests 26.17 ± 12.51 items over time, and the kNN top-10 recommends the highest number of items over time: 79.86 ± 59.33. This ensures that kNN will also produce the highest number of *new* recommendations. Weekly switching slightly lowers kNN's average, to 75.36 ± 53.98 because repeatedly visiting the SVD recommendations reduces the number of total items that can be recommended. However, user based switching maintains the average number of recommended items over time at 79.86 ± 55.12; it highly promotes temporal diversity without impacting the number of new items that enter the top-10 list over time. However, re-ranking bolsters both the average and standard deviation to 97.93 ± 78.82; re-ranking thus seems like a promising approach to solving the related problem of temporal novelty in recommendation.

5.4 Discussion

Diversity is a theme that extends beyond recommender systems; for example, Radlinski and Dumais examine how it can be used in the context of personalised search [RD06]. In other cases, diversifying search results is done in order to reduce the risk of query misinterpretation [AGHI09]. Similarly, diversity relates to user satisfaction; more specifically, to users' impatience with duplicate results [HRT09]. We have observed similar 'impatience' in our survey: users who completed the survey with no diversity began to rate recommendations lower as they saw that they were not changing.

It is certainly possible to envisage a finer grained notion of diversity that takes semantic data into account—by measuring, for example, the extent that the same genre or category of items are being recommended. To that end, diversity may also be measured within a *single* top-N list, rather than a pair or sequence of recommendations; such a metric may, for example, take into account the number of highly related items (such as a movie and its sequels, or multiple albums by the same artist) that are being simultaneously recommended. For example, Smyth and McClave [SM01] apply strategies to improve recommender systems based on case-based reasoning; diversity, in this case, is viewed as the complement of similarity. Zhang and Hurley [ZH08] also focus on intra-list diversity, and optimize the trade off between users' preferences and the diversity of the top-N results. In this chapter, we focus on the temporal dimension (inter-list diversity) and whether the exact same items are being offered to users more than once; we do not take semantic relationships between the recommended items into account nor improve the diversity of individual top-N lists. However, both lines of research are not in conflict: ideally, one would like a recommender system that offers diverse results that *change* over time to suit each users' tastes.

There is one limitation to the work we have performed here. We do not know what users were *actually* recommended: in fact, we do not know if users are clicking on their recommendations or are selecting movies to rate by other means. Assuming that Netflix was not simply recommending popular content, this limitation may explain why the baseline results show so little diversity. While the results certainly show what one may expect from providing popularity-based recommendations, it is also possible to envisage higher diversity for the baseline case, if, for example, users dislike their recommendations so much that they are giving them all 1 star (in the next update they will be shown different results). However, this limitation is a widespread problem with CF research; in fact, to date the relationship between what people are *recommended* and what they *rate* remains largely unexplored.

5.5 Summary

This chapter focuses on temporal diversity: how recommendations change over time. In doing so, we have extended how CF can be evaluated over time; in Chapter 4, we focused on the accuracy of predictions; here we added the diversity and novelty of recommendation lists over time. We found that state of the art CF algorithms generally produce low temporal diversity; they repeatedly recommend the same top-N items to a given user. We then defined a metric to measure temporal diversity, based on the set theoretic difference of two sequential top-N lists, and performed a fine-grained analysis of the factors that may influence diversity. We found that, while users with large profiles suffer from lower diversity, those who rate a lot of content in one session are likely to see very diverse results the next time. We also observed that diversity will naturally improve over time. We then designed and evaluated three methods of improving temporal diversity without extensively penalising recommendation accuracy. Two were based on *switching* CF algorithm over time; users are first given recommendations produced with (for example) a kNN approach, and then offered the results of an SVD algorithm. The last method was based on re-ranking the results of frequent visitors to the system.

Chapter 6

Temporal Defences for Robust Recommendations

Recommender systems are vulnerable to attack: malicious users may deploy a set of sybils to inject ratings in order to damage or modify the output of CF algorithms. Previous work focuses on designing *sybil profile* classification algorithms, which operate independently of CF, and aim to find the current sybils each time they are run. These methods, however, assume that the full sybil profiles have already been input to the system. As previously observed, deployed recommender systems, on the other hand, operate over time: recommendations may be damaged as sybils inject profiles (rather than only when all the malicious ratings have been input), and system administrators may not know when their system is under attack. In this chapter, we address the problem of *temporal* sybil attacks, and propose and evaluate methods for monitoring *global*, *user* and *item* behaviour over time in order to detect rating anomalies that reflect an *ongoing* attack. We conclude by discussing the consequences of our temporal defences, and how attackers may design *ramp-up attacks* in order to circumvent them.

6.1 Problem Setting

When CF algorithms compute recommendations for web users, they do so assuming that the ratings they manipulate are *honest* depictions of user preferences. Unfortunately, this may not be the case: any system that invites participation is also vulnerable to malicious abuse, and the ratings input to CF algorithms may have been fabricated to damage or modify the recommendations the system outputs. Abusing recommender systems this way is often referred to as *shilling*, *profile injection* or *sybil* attacks; Mobasher *et al.* provide an in-depth review of this problem [MBW07]. All of these terms are synonymous: attackers deploy a set of pseudonymous entities (e.g., automated bots) that rate content in a manner that serves the attacker's goals. These attacks are further categorised based on the intent of the attacker: a *random* attack aims to disrupt the system by injecting noise, while *targetted* attacks aim to promote or demote the ranking (and thus, the recommendability) of individual items. There is a growing body of research that addresses the problem of identifying malicious profiles; for example, Williams *et al.* evaluate the potential that a variety of classifiers have to find sybils [WMB09]. In other words, given a matrix of user-item ratings that contains a set of sybil profiles, the problem to be solved is how to divide the honest from the malicious profiles in order to exclude the sybils. The underlying assumption here is that the *full sybil profiles* are already contained within the user-item matrix: all the sybils have rated all the items that they intend to in order to perform their attack. However, recommender systems are subject to change over time, as users input more ratings and CF algorithms are retrained in order to serve the most up-to-date recommendations. This reality of deployed recommender systems presents two challeges to the assumptions held by attack detection algorithms:

1. The sybil profiles may not be fully inserted or may be inserted over an extended period of time; thus reducing their immediate detectability, while not necessarily reducing the damage they may inflict on the system.
2. As the system is updated, the problem of *when to run* expensive detection algorithms arises: how can system administrators know that their system is potentially under attack?

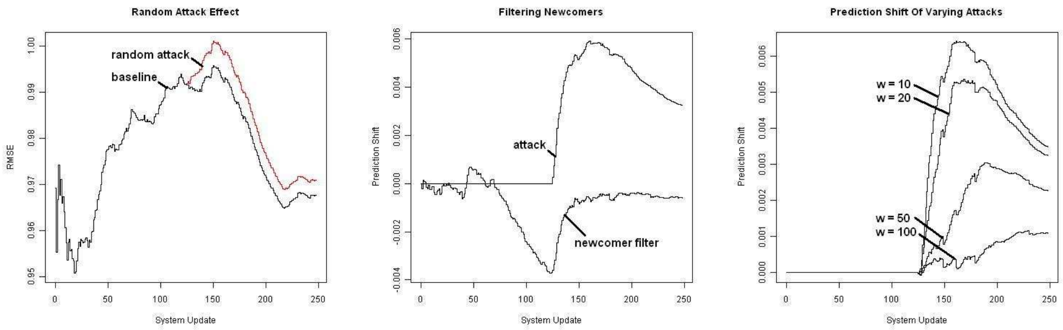
In this chapter, we address these challenges by designing and evaluating algorithms for monitoring recommender system users' behaviour over time. To do so, we make the following contributions:

- We preface this chapter in Section 6.2 by showing how non-temporal profile injection attacks (where the sybils appear, dump malicious ratings over a number of days, and dissapear) can be defeated easily; attackers therefore have an incentive to extend the time taken to inject profiles. We explore this incentive with experiments comparing random profile injection attacks over varying time lengths.
- Based on the previous experiments, we describe the range of potential temporal sybil attacks in Section 6.3. In doing so, we relate the number of sybils and number of ratings input per sybil over time to previously defined (random/targetted) attack models, highlighting the relation between how an attack is carried out and the intent of the attacker.
- In Section 6.4 we describe and evaluate methods of monitoring global, user and item activity over time and identifying anomalies that reflect and flag an ongoing attack.
- Based on the defences we construct, we analyse how attackers may respond, and propose directions for future research by defining adaptive attack models in Section 6.5. We close by discussing related work and concluding in Sections 6.6 and 6.7.

6.2 Defeating Non-Temporal Attacks

We use the same model of recommender system temporal updates as we did in previous chapters: given a dataset at time t, and a window size μ (reflecting how often the system will be updated), we train the algorithm with any data input prior to t and predict any ratings input between t and $(t + \mu)$. We also use the same five subsamples of Netflix user profiles we examined previously. However, in this chapter we reduce the breadth of CF algorithms we consider; we focus only on the item-based kNN algorithm, with the $k = 50$ neighbours weighted as described in [LHC08c] and Section 2.3.2:

$$w(a,b) = \frac{1}{|R_a|}\left(\sum_{i \in R_a} value(a,b,i)\right) \tag{6.1}$$



(a) One-Shot Random Attack (b) Effect of Filtering Newcomers (c) Different Attack Lengths

Figure 6.1: Time-Averaged RMSE Of One-Shot Attack, and Prediction Shift When Pruning Newcomer's Ratings, and Injecting Attacks Over Varying Time Windows

Where the *value* of two ratings is defined as:

$$value(a, b, i) = 1 - \rho|r_{a,i} - r_{b,i}| \tag{6.2}$$

We made this decision for two reasons: (a) comparisons of inter-algorithm attack robustness have already been done [MBW07], and (b) our goal is to design algorithm-independent mechanisms for identifying temporal attacks.

A non-temporal attack would operate as follows: between time t and $(t + \mu)$, a set of sybils S would input a set of ratings X; the non-temporal characteristic of this attack is that all the malicious ratings are input within a single window (while the *actual* time taken to operate the attack may span the size of the entire window). We can then measure the change to the temporal performance with the time-averaged RMSE metric. We visualise the temporal effects of a non-temporal attack with the following example. During the $125th$ week-long window in the Netflix data, we inserted 100 sybils who each rated approximately $10,000$ selected items. In this example, we limit ourselves to exploring the temporal effect of a *random* attack: each sybil randomly picks one of the available items, and then rates it with a random value drawn uniformly from the rating scale. Figure 6.1(a) plots the impact that these ratings have on the time-averaged RMSE.

The random attack has a pronounced effect on the time-averaged RMSE: performance is consistently degraded over the rest of the updates. However, this attack is simple: sybils appear, rate within the window length, and disappear (a "one-shot" attack). Furthermore, at each update the system re-trains using all historic data, regardless of whether the users who have input that data continue to reappear. The natural response is therefore to *distrust newcomers*: any ratings from new users are *suspect*. In Figure 6.1(b) we repeated the previous experiment, but excluded suspect ratings from the item-similarity computation step of the kNN algorithm. By excluding suspect ratings this way, we maintained our ability to formulate recommendations for all users (including sybil and new honest users- should they reappear), while removing the influence that suspect ratings exert on the item neighbourhoods. We plot *prediction shift* values, i.e., the difference between the baseline (predictions with no sybils inserted) and the attack and newcomer-filtered scenarios. While we certainly pruned away the ratings of non-sybil users, the technique not only eliminates the effect of the attack, but also *improves* upon the baseline RMSE in a number of windows prior to the attack taking place (i.e., the prediction shift is negative). Removing the ratings of users who appear, rate, and do not return thus avoids one-shot attacks and seems to take small steps towards de-noising the data [APTO09].

Given this situation, an attacker may simply respond by widening the number of windows taken to inject ratings. Sybils under the attacker's control would therefore appear in multiple windows and, after the first appearance, no longer be suspect. In order to explore the incentives that attackers have to rate-limit their own sybils, we performed a number of random attacks, where a set of 100 sybils rated the same number of items over a varying number of sequential windows $W_a \in \{10, 20, 50, 100\}$. In each case, the *number* of malicious ratings remained the same, the only difference being the *time* taken to insert them; we compare attacks of the same magnitude that differ only in temporal spread (i.e., the ratings per sybil per window varies, as does the number of windows). The results in Figure 6.1(c) show that injecting ratings over longer time periods deviates the RMSE from the baseline less. This is likely to be an effect of the balance between sybil and non-sybil ratings: longer attacks have less of an effect since, during the time taken to operate the attack, there is a larger influx of non-sybil ratings.

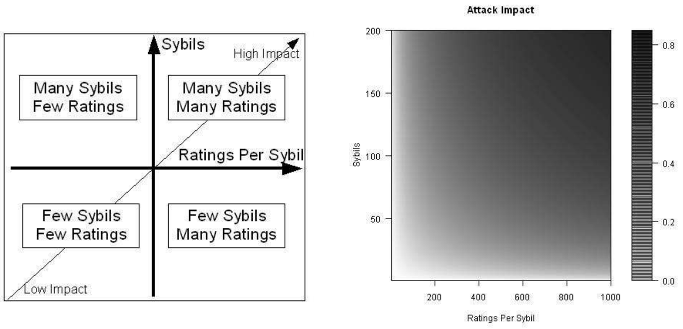
We draw two conclusions from the above experiments: there is an incentive for attackers to (a) inject ratings over more than one window (in order to not have their ratings be suspect), and (b) inject as much data as possible, in order to have the greatest possible effect (since higher volumes of sybil ratings per window has a more pronounced effect). With this in mind, we describe temporal attacks by considering the choices attackers must make when designing a sybil attack.

6.3 Temporal Attack Models

Previous work [MBW07] examines different regions of sybil profiles, looking at what items sybils must rate in order to define different attacks. This structure still holds on the temporal scale; the difference is how long it takes the attacker to construct the sybil profiles. In this chapter, we do not assume that the profiles are populated in the same order (i.e., all sybils rate movie m_1 first, m_2 second, etc), or that they even all contain the same items; instead, we assume that the *rate* at which they are populated is roughly similar (in Section 6.5 we discuss the consequences of breaking the latter assumption).

There are a number of factors that attackers control when they implement a temporal attack: *how many* sybils should rate content, the *rate* at which sybils should inject ratings, and *how long* they should continue rating for. Attacks can thus be classified according to how attackers calibrate these factors, and whether they hit the system with (*many, few*) sybils rating (*many, few*) items per window, for a predefined sequence of windows. Figure 6.2(a) summarises this view. Each quadrant represents a combination of these two variables; a third dimension (not pictured here) would represent the rate of attack. This is important because these variables reflect the rationale of the ongoing attack. For example, many sybils rating many items translates to inputting a high volume of malicious data, and may reflect an ongoing random attack.

The relation we have outlined above is important since it explores how an intelligent attacker would go about achieving particular goals when attacking a recommender system. However, there are a number



(a) Attack Classification (b) Attack Impact With No Defences

Figure 6.2: Attack Types and Impact With No Defences

of factors that attackers cannot control, related to how the non-sybil users behave: how many *non-sybil* users there are, the number of *ratings* that they input per window, *what* they rate, and *how* they rate. We will leverage this information in order to construct a defence to temporal attacks, which we introduce in the next section.

6.3.1 Measuring Attacks

There are a number of ways of measuring the effect of attacks, ranging from prediction shift, to hit ratio, and average rank [MBW07]; each aims to measure changes introduced by malicious ratings. In this chapter, we are interested in measuring how well our defences detect ongoing attacks (rather than how the attacks change recommendations); we thus focus on the detection *precision*, *recall* and the potential attack *impact*. Given a window t, and a set of sybils S_t who rate X_t items each during the window, the impact of the attack is simply the number of sybil ratings input at the current window t, or $S_t \times X_t$, divided by the total number of ratings R_t input in that window:

$$impact_t = \begin{cases} \frac{S_t \times X_t}{R_t}, & \text{if attack is undetected} \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

In measuring attacks this way, we assume that, if a system administrator can be told that the system is under attack, then one of the many sybil-identifying classifiers that are described in the literature can be used to prune the actual sybil ratings. If no attack is flagged, then we measure the relative size of the attack. This metric gives higher weight to attacks that inject more ratings; Figure 6.2(b) plots the attack impact for varying sizes of sybil groups and rating rates when no defences are in place. While it is certainly possible to envisage attacks that, by carefully tuning what the sybils rate, cause more damage with fewer ratings than higher volume equivalents, in this chapter we are not concerned with comparing attacks to each other. Instead, we use the metric to see how many ratings attackers can slip into the system without causing behavioural anomalies. We also measure the number of true positives (TP, attacks that were flagged), false positives (FP, non-attacks that are flagged), and false negatives (FN, attacks that were not flagged). Precision and recall are then computed as:

$$precision = \frac{TP}{TP + FP}; \quad recall = \frac{TP}{TP + FN} \tag{6.4}$$

All these metrics, however, are related: the precision and recall relate the proportions of false positives and negatives to the true positives, while the impact, by being non-zero when an attack slips through, displays the false negatives in a manner that takes into account the size of the attack that failed to be detected. In effect, we have two metrics that explore facets of false negatives. The emphasis we place on false negatives throughout this chapter is motivated as follows: we cannot know (and only assume) that the data we experiment with is the fruit of honest, well-intentioned users; similarly, we can only know that an attack is taking place when we manually insert it. We therefore place a higher importance on reducing false negatives (i.e., finding all the attacks that we insert) within the data that we have: false positives in the real data may very well be attacks that produce anomalous behaviour, and are likely to deserve further inspection. However, we note here that the defences described below produced no false positives when run on the temporal rating data with no attacks manually injected.

6.4 A Temporal Defence

In the above section we outlined the factors that attackers determine: the *time* (number of windows), *size* (number of sybils), *rate* (number of ratings per sybil per window), and *strategy* (which items need to be rated: random/targetted) when implementing an attack. In this section, we describe a method of detecting different forms of attacks, based on monitoring the global behaviour (Section 6.4.1), user behaviour (Section 6.4.2), and item behaviour (Section 6.4.3) for anomalies. The key to our proposal is that attacks may be identifiable by finding consistent anomalies caused by the sybil group's behaviour.

6.4.1 Global Thresholding

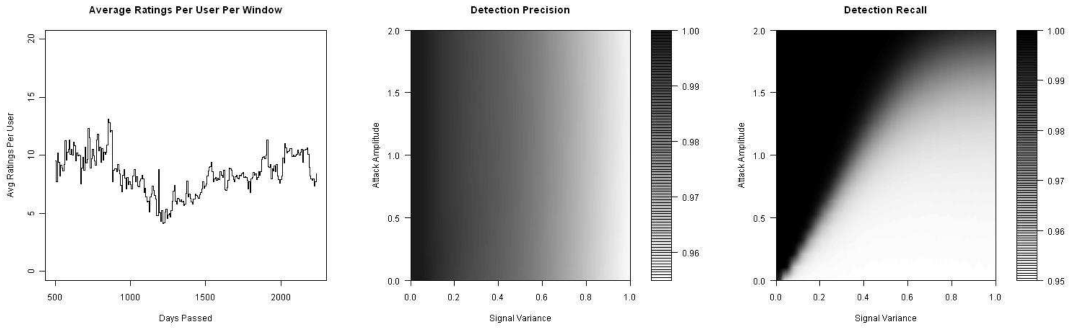
The first perspective of system behaviour that we consider is at the *global*, or aggregate, level. While the number of ratings that users input varies over time, the average ratings per user per window (in the Netflix data) remains relatively flat: Figure 6.3(a) plots this value over time. From this, we see that the average user will rate between $5 - 15$ movies per week. Since the mean is derived from a long-tailed distribution, it is a skewed representation of the "average" user. However, an attacker, by deploying a group of sybils who inject ratings at a pre-defined rate, will modify this aggregate value; the first dimension of our defence thus aims at monitoring changes to the average ratings per user MU_t over time. Given a window t, the current mean ratings per user MU_t, standard deviation σ_t, the R_t ratings input by U_t users an alarm is raised if the volume of incoming ratings departs from the mean measured to date by an amount determined with a global threshold $\alpha_g \geq 1$:

$$\frac{R_t}{U_t} \geq (MU_t + (\alpha_g \times \sigma_t)) \tag{6.5}$$

Otherwise, we update the current MU_t value as an exponentially weighted moving average (with a weighting factor β_t):

$$MU_t = (\beta_t \times MU_{t-\mu}) + ((1 - \beta_t) \times \frac{R_t}{U_t}) \tag{6.6}$$

MU_t is updated *conservatively*: if an attack is flagged, then it is not updated. We also update both the α_t and β_t variables. The β_t variable determines the weight that is given to historical data: relying too heavily on historical data will not capture current fluctuations, while weighting current values too highly



(a) Avg Ratings Per User (b) Simulation Precision (c) Simulation Recall

Figure 6.3: Netflix Ratings Per User Per Week; Global Thresholding Precision and Recall

will disperse temporal trends. We therefore determine the next value $\beta_{t+\mu}$ with the standard deviation measured to date:

$$\beta_{t+\mu} = \min(|\sigma_{t-\mu} - \sigma_t|, 1) \tag{6.7}$$

The value is capped at 1, thus ensuring that when there is high variability in the data, β_t gives higher preference to current values, while smaller standard deviation shifts β_t to give higher weight to historical values. The α_t variable determines the extent to which the current $\frac{R_t}{U_t}$ value can deviate from MU_t before an attack is flagged. When an attack is flagged, we reduce α_t, in effect, making it more difficult for attackers to learn the appropriate threshold. We set α_t to jump between pre-specified values (0.5 and 1.5):

$$\alpha_{t+\mu} \begin{cases} 1.5, & \text{if no attack detected} \\ 0.5, & \text{otherwise} \end{cases} \tag{6.8}$$

Monitoring incoming ratings at the aggregate level is sensitive to two factors: how naturally variable the incoming ratings are, and the amount of variance that attacks introduce. In other words, a mechanism like this may not work if there is already high variance in the average ratings per user and sybils do not displace the mean value. We therefore evaluated this technique with two methods: in the first, we *simulate* a stream of incoming ratings (in order to control both the variance and size of attack); we then turned to *real data* where we could explore the effects of varying attacks in a more realistic setting.

In order to simulate a stream of incoming ratings, we draw a sequence of $\frac{R_t}{U_t}$ values from a normal distribution with (fixed) mean MU and standard deviation $\sigma \in [0, MU]$. Then, at random moments, we simulate an injected attack where a group of sybils shifts the incoming value by the attack *amplitude* $\gamma \in [0, (2 \times MU)]$; in other words, at an attack time t, the window's value is $(\frac{R_t}{U_t} + \gamma)$. We then note whether an attack was flagged, and can compute the detection precision and recall with the results.

When running the simulation, we assumed that, after a brief training phase, the system could be attacked at any time during a period of $1,000$ windows, for a pre-determined number (50) of sequential attack windows. We re-ran each simulation parameter setting $10,000$ times and present averaged results. Figure 6.3(b) shows the resulting precision, which fades as σ increases, but is otherwise dependent on σ (the variability in the ratings per user per window) rather than the attack amplitude γ. In other words,

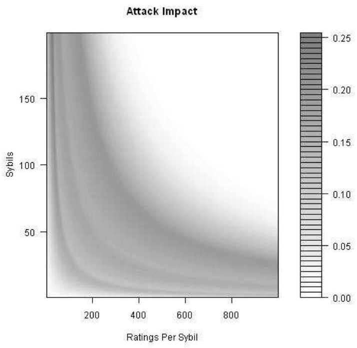


Figure 6.4: Global Thresholding Impact

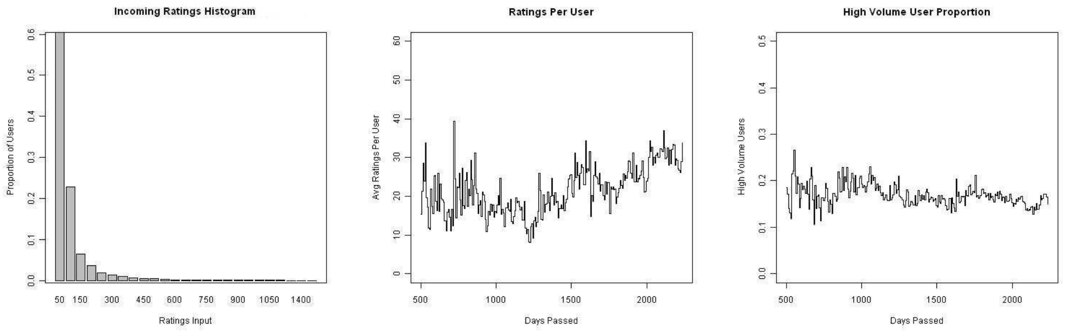
the number of false positives depends on how naturally variable the data is, and, given that the real data displays low spread, the number of false positives is likely to be low. Figure 6.3(c), instead, displays the detection recall. This plot highlights the trade-off between σ and γ: the best recall is when a small σ is modified with a large γ, while the worst values are found when a large σ is deviated by a small γ. However, we note that the minimum precision is slightly below 0.90, while the minimum recall remains above approximately 0.95.

We returned to the Netflix subsets in order to test this method with real data. To do so, we trained our monitor with all ratings per window until the attack time, and then measure the attack impact after injecting the attack. Since the attacker may unleash the sybils at any time, we repeated our experiments, starting attacks at each possible window, and plot average results across all windows. As Figure 6.4 shows, this method catches attacks where large groups of sybils inject their profiles at a very high rate; the top right corner of the plot is flattened to zero impact. However, two sensitive areas remain: first, where *many* sybils inject *few* ratings, and when *few* sybils inject *many* ratings. Attackers can thus respond by either reducing the size of the sybil group, or the the sybil's rate. However, this plays into our hands: in Section 6.4.2 we address the former, while Section 6.4.3 describes how the latter attacks can also be monitored.

6.4.2 User Monitoring

One of the shortcomings of the Global Thresholding detection mechanism is when *few* sybils rate *many* items each. We address this pitfall by designing a user monitor, which aims to detect this particular scenario. Figure 6.5(a) plots an example distribution of ratings input in a single window; we find that majority of the users input a low number of ratings per week, while a minority of outliers rate a high volume of movies. An attack in this context would thus entail setting a group of sybils to rate a high volume of content over a number of windows; detecting this behaviour focuses on examining *how many* high volume raters there are and *how much* these outliers are rating.

(a) How Much High Volume Raters Rate. Given the current mean value of ratings per user per window MU_t, we differentiate *high* from *low* volume raters based on the difference between the ratings



(a) Static Ratings Per User (b) Temporal Ratings Per User (c) Proportion High Volume Raters

Figure 6.5: Example Ratings Per User (1 Week), Proportion of Ratings Per High Volume Raters and High Volume Raters Over Time

that they have input in the current window and MU_t:

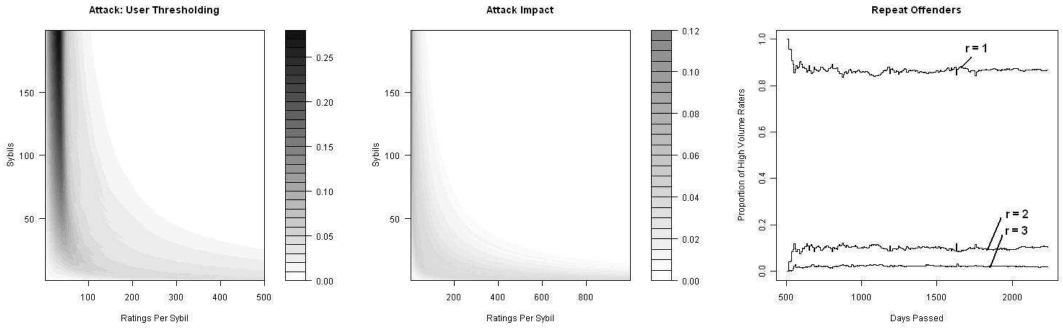
$$high(U_t, MU_t) = \begin{cases} \text{true}, & \text{if } U_t - MU_t > 0 \\ \text{false}, & \text{otherwise} \end{cases} \tag{6.9}$$

The mean ratings per high volume user, *HM*$_t$ can then be monitored, in a similar way that we monitored the entire distribution in the previous section: an exponentially weighted moving average is regularly updated, and large deviations from the expected value flags an ongoing attack. In Figure 6.5(b) we plot the ratings per high volume user over time.

(b) How Many High Volume Raters. Given the high volume raters found with Equation 6.9, we also keep track of how many users HU_t there are relative to all the users who have rated in the current window. In other words, a user is suspect if they are at the highest end of the user-rating distribution, and both the *size* of this group and *volume* of ratings they input may indicate an ongoing attack. As we plot in Figure 6.5(c), the size of this group of users, divided by the total number of high volume raters per window, tends to be relatively stable; injecting different forms of attacks upsets both this and the mean ratings per high volume user values.

We take advantage of both pieces of information in order to amplify our detection mechanism: we create a *combined score* per window by multiplying the *HM*$_t$ value by the proportion of suspect users HU_t. This way, we aim to capture fluctuations in both the *group size* and *rate* that a potential group of sybils will inflict when performing their attack.

We evaluated the user monitor with the Netflix subsets for cross-validated results with real data. We did so in two steps. First, Figure 6.6(a) shows the resulting impact if only part (a) of the above is used to defend the system: this defence can overcome similar scenarios that we addressed in the previous section or while lessening the threat of smaller groups of high-volume rating sybils. This threat is not fully eliminated: the top-left of the plot shows a remaining non-zero impact section. This is the effect of the false negatives of our monitor: sybils who rate at high volume but are not flagged. In Figure 6.6(b), we plot the impact of the *combined* defences. Overall, it reduces the impact of random attacks: Figure 6.6(a) reports attack impacts between approximately $[0, 0.25]$, while the combined defences range



(a) User Thresholding (b) Combined Threshold (c) Size of Repeat Offenders

Figure 6.6: User Monitor/Combined Impact Results, and Proportion of High Volume Raters Who Have Been In The Group for Varying Lengths of Time

between approximately $[0, 0.12]$. Nevertheless, the attacks that have the highest impact are now those where *many* sybils rate *few* items. In the next section, we consider the scenario where this type of attack now dominates.

Future Developments. One aspect that may aid the user monitor, but we leave as future work, is the *consistency* of membership to the high-volume raters group. In Figure 6.6(c) we plot the proportion of high-volume raters who have been in this group for $r \in \{1, 2, 3\}$ number of consecutive windows, after pruning the newcomers' first ratings. We find that over 80% of the high-volume raters are appearing in this group for the first time; as r is incremented, the relative group size falls sharply: at most 12% of the group members are making their second appearance, 3% are making their third. Sybils who are injecting a lot of noise for an extended period of time would become familiar faces in the high-rating group. Furthermore, the extent that honest users who rate large volume of movies per week input valuable data is questionable.

6.4.3 Item Monitoring

The last scenario that we address is the attacks that see *many* sybils rate *few* items each. This form of attack overcomes the previously outline defences: the sybils do not rate enough items each to be detected by the user monitor, and there are enough of them to not shift the rating per user temporal mean and flag their presence. To attempt to detect this kind of attack, we first reason on what items the group of sybils may be rating, and then design and evaluate an *item*-monitor to identify ongoing anomalous behaviour.

CF algorithms, that will be affected by injected profiles, operate on vectors of ratings. It thus seems intuitive that, in order to have the greatest impact possible, groups of sybils who inject very sparse profiles (by rating few items each) will tend to be rating a similar subgroup of items, rather than dispersing the ratings over a broad range of items, which would have a smaller effect. This strategy recalls the structure of *targetted* attacks [MBW07], where injected profiles contain *filler*, *selected*, and *target* item ratings. These profile regions correspond to the ratings that sybils must enter in order to construct an attack; for example, if an attack aims to promote a fantasy movie, the sybils may rate famous *selected* fantasy movies, along with a number of *filler* items to disguise each profile as a "normal" user profile. The difference between a random and targetted attack is thus determined by the *strategy* of how to populate the profiles: what the *selected*, *filler,* and *target* items are (in the case of a random attack, there is no target item) and how they are rated; furthermore, the common subgroup of items that all sybils rate is the *selected* and *target* item. On a temporal scale, this form of attack would entail a large group of sybils rating items amongst this subgroup within a number of windows (proportional to the attack length). We therefore turn to monitoring the items in a system to detect these kinds of attacks. We further assume that it is very unlikely for an item that is *already* popular to be subject to an attack that aims to promote it; similarly, it is unlikely that unpopular items be demoted. In other words, we assume that the purpose of attackers is to maliciously reverse an ongoing trend (rather than reinforce a pre-existing one). Given this, we design an item monitor to identify the target of attacks by focusing on three factors: the *amount* that each item is being rated, the distance the *mean* of the incoming ratings for each item has from an "average" item mean, and a temporal mean *change detector*.

(a) The Item Is Rated By Many Users. At each time t, with R_t ratings input for I_t items, the average ratings per item MI_t (with standard deviation $\sigma_{i,t}$) can be computed. We can then select, from the available items, those that have been rated the most in the current window by selecting all those that received I_t ratings greater than the mean number of ratings per item MI_t:

$$high(I_t, MI_t) = \begin{cases} \text{true}, & \text{if } I_t > MI_t + (\alpha_t \times \sigma_{i,t}) \\ \text{false}, & \text{otherwise} \end{cases} \tag{6.10}$$

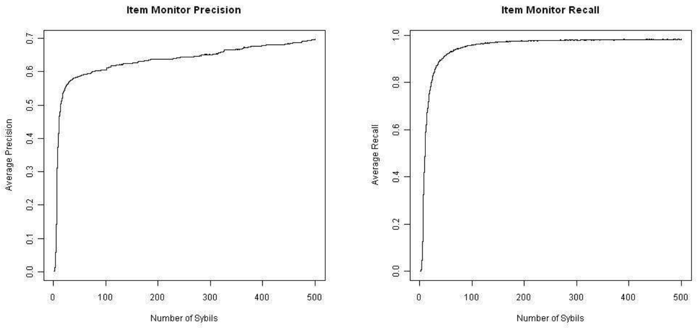
(b) The Item is Rated With Extreme Ratings. Using only the ratings input in the current window w, we determine the *mean* score $\bar{r}_i$ for each item i, and then average these to produce the expected mean score v per item:

$$v = \frac{1}{I_t} \sum_{i \in I_t} \bar{r}_i \tag{6.11}$$

If an item has been targetted for attack (and either demoted or promoted by a group of sybils simultaneously), then the corresponding $\bar{r}_i$ will reflect this by being an outlier of the global average item mean v.

(c) The Item Mean Rating Shifts. We compare the item mean computed with historical ratings and the $\bar{r}_i$ value determined from the ratings in the current window. A successful attack will shift this value by some distance δ: in this work, since we are operating on the Netflix 5-star ratings scale, we set δ to slightly below 2.

An attack is flagged for an item if the above three conditions are met: it is rated more than average, and the mean of the incoming ratings shows that it is both not being rated in the same way as other items are, and a change from the historical value is being introduced. Our monitor therefore focuses on identifying the moments when groups (or subgroups) of sybils rate the *target* item. We therefore modified our evaluation mechanism to test how well we find items when they are attacked, depending on how many sybils push in the target rating at the same time. We evaluated the monitor as follows: at time t, a group of sybils rates a randomly chosen target item. The sybils demote the item if it is popular (it has mean greater than 3), and promote it otherwise. We do not discriminate on the number of ratings



(a) Avg Precision

(b) Avg Recall

Figure 6.7: Item Monitor: Average Precision & Recall

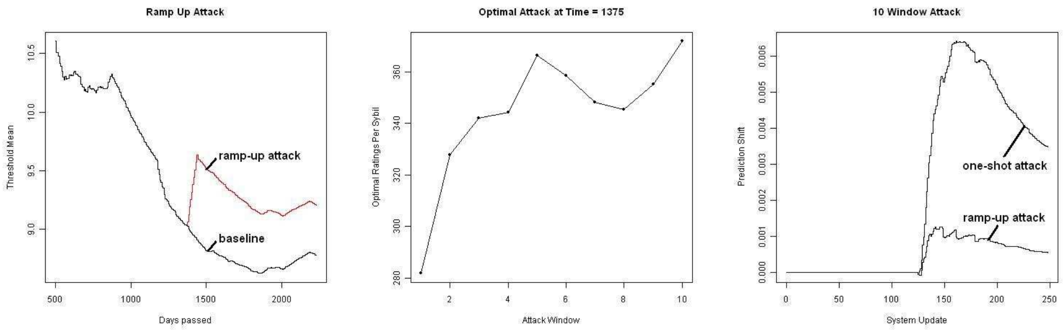
that movies currently have when determining whether to nuke or promote it; however, previous work shows that it is harder to protect sparsely rated items from attack [LR04], and our item selection process is biased toward selecting these items. We then check to see if the monitor flags any suspicious items, and measure the number of true/false positives and false negatives. We repeat the same run (i.e., group size and attack window) for 50 different items, and measure the resulting precision and recall. However, since an attack may begin in any of the available windows, we then repeat this process for each possible window, and average the results across time. Finally, we repeat this entire process with each Netflix subset to produce cross-validated results. The results therefore take into account the differences between sybil group size, target item, attack time, and honest user behaviour.

The average precision and recall values are plotted in Figures 6.7(a) and 6.7(b). They highlight that these methods work best when *many* sybils are rating the same item, with recall near 99% and precision near 70%. The fact that the precision is not performing as well as the recall implies that there are a higher proportion of false positives rather than false negatives: when an item is under attack, it is likely to be flagged as such, but few items that are not attacked may be flagged as well. As with the user monitor, it remains unclear how to deal with items that are being rated anomalously by users who are not the sybils that we explicitly control in our experiment. In fact, we can only be certain that users are malicious if we explicitly injected them: otherwise, we have assumed that the users in the dataset are honest and well-intentioned, which may not be the case. It is therefore preferable, in this case, to have a monitor with higher recall than precision, since we are sure that the sybils we inject are being found.

6.5 Adaptive Attack Models

The previous sections describe methods to detect different forms of automated sybil attacks by spotting anomalies in user behaviour. One of the natural limits of these techniques is when honest users' behaviour deviates from what was previously learned to be normal; a recent example is the vast numbers of web users who searched for news relating to the death of Michael Jackson [1] (a news article recommender system may thus see these articles being read in an anomalously high volume). On the other hand, attackers may modify their methods in order to overcome the defences. In this section, we switch to the

[1] http://tech.slashdot.org/story/09/06/29/003214/Google-Mistook-Jackson-Searches-For-Net-Attack



(a) Global Monitor (b) Optimal Ratings Per Sybil (c) Prediction Shift

Figure 6.8: Example Ramp-Up Attack: How it Affects the Monitor's Values, the Optimal Ratings Per Sybil and Prediction Shift

point of view of the attackers: we describe how attackers may overcome these defences, and evaluate the effect that such attacks have on the Netflix data.

6.5.1 The Ramp-Up Attack

In Section 6.4.1, we described a method of curbing random attacks by means of monitoring an exponentially weighted moving average of the ratings per user per window; the user- and item- monitors had a similar flavour. An attack was flagged if the current ratings per user value exceeded a threshold above the moving mean, determined by the historical values' variance and a weight α_g. The key insight of this detection mechanism is that an attack will *suddenly* and *noticeably* shift the moving mean, and can thus be detected. Similarly, our evaluation assumed that the *rate* at which sybils inject ratings remained constant.

Attackers may attempt to overcome this defence by *incrementally* changing the sybils' rate of attack. In the best case, the attacker would know what the current threshold is, and could set a group of sybils to inject at a rate that would push the mean up to (but not beyond) this value. Doing so would allow the attacker to then increase the rate in the following window; as the moving mean increases (along with the threshold), the attacker may be free to unleash evermore higher rates of profile injection. We call this a *ramp-up attack*.

The ramp-up attack can be used to defeat all three of the above defences; we experiment with this by considering the scenario of a system that only has the global monitor in place, and an attacker who would like to inject as much noise as possible into it during a period of 10 consecutive windows, starting roughly halfway through the dataset's timespan. Furthermore, we only consider the optimal case, where attackers know *a priori* what the incoming ratings per user per window values are, and can thus deduce what the threshold will be and tune sybils' rates accordingly. In doing so, we give attackers an advantage that they are unlikely to have; we discuss this further below.

In Figure 6.8(a) we plot the exponentially weighted moving average of the global threshold over time, both with and without a ramp-up attack. The effect of the ramp-up attack is to shift the mean, which then remains parallel to the original. Longer ramp-up attacks would shift the mean even further. Based on this mean (and corresponding threshold), we computed the maximum ratings per sybil per window that avoids detection throughout the course of the ramp-up attack; we plot this sequence in Figure 6.8(b). An interesting point to note is that, since the mean relies on how both the honest and sybil users are behaving, the optimal values do not increase monotonically: there are windows where the honest users collectively rate less, which would thus highlight the sybils' misbehaviour unless their rate is reduced. Lastly, we measured the prediction shift when sybils inject ratings in this fashion, and compare it to the original attack (with the same duration) that we examined in Section 6.2. The results, in Figure 6.8(c), highlight the importance of our defences: forcing attackers to ramp-up rather than simply dump ratings into the system affects the prediction shift much less.

The results we show here are the best-case scenario for the attacker, since they knew the current threshold and ratings per user value. In practice, it is unlikely that the attackers know the current threshold, for a number of reasons:

1. The incoming ratings per user value (that we assumed that attackers knew) is computed at end of the current window. Attackers who may monitor all users to learn the precise value would then have no time to inject ratings.

2. The current threshold varies as the exponentially weighted moving average is updated; even if attackers knew the previous window's threshold there is no guarantee that the attacker can inject the maximum number of undetectable ratings in the current window.

3. Experimenting, in order to discover the threshold, would be difficult since, as we saw in Figure 6.8(b), avoiding detection in one window does not guarantee that the same rate will avoid detection in the next.

4. Furthermore, attempting to discover the threshold will (a) impact the threshold itself, since α is updated, and (b) reveal a set of sybils once the threshold has been surpassed, requiring attackers to then restart their efforts from scratch.

Similar ramp-up attacks may be performed to overcome the defences in Sections 6.4.2 and 6.4.3. However, the main insight from the above experiments is that ramp-up attacks are more difficult and require more time to execute than attacks on an unmonitored system. This therefore highlights that, while the temporal monitors that we describe above are not infallible, they provide a significantly difficult obstacle that attackers now need to overcome.

6.6 Discussion & Related Work

Anomaly detection algorithms have been used when securing a wide range of systems, in order to, for example, defend against financial fraud [WB08] or defend web servers from denial of service attacks [SP06]. These techniques are readily applicable to recommender systems; the only problem being how to define what an anomaly is, and how to monitor the large volume of users and items. In this chapter, we have introduced novel methods that detect anomalies in various aspects of rating behaviour while learning what normal user behaviour is, thus liberating system administrators from these challenges.

To do so, we leveraged the effect that honest users have on the temporal dynamics of the system. For example, we used the fact that a majority of users rate very few items in order to identify the sybils who are rating a lot. The only way that sybils may dodge pushing the monitored variables over the detection thresholds is by *not rating*: our defences acts as an incentive for attackers to draw out the length of their attack, thus reducing its overall effect (as seen in Section 6.2).

The monitors that we described above each address different forms of attack: the global monitor detects large groups with high rates of profile injection, the user monitor detects the few users who are injecting large volumes of ratings, and the item monitor detects when many users are rating a target item in an anomalous fashion. We thus evaluated each one separately in order to highlight each monitor's strengths and weaknesses. However, we already saw that more than one monitor may flag the same attack; for example, the user monitor detected many of the same attacks as did the global monitor. In the future, we plan to evaluate how multiple defences operate when combined, and the overlap between user, item, and global behaviour as different attacks are taking place.

Anomaly detection has also been seen before in recommender system research. Bhaumik *et al.* [BWMB06] propose a method to monitor *items* as they are under attack, by looking at changes to an item's mean rating over time. Similarly, Yang *at al* [YSKY09] infer user trust values based on modeling the signal of incoming ratings. They use these techniques to monitor when *real users*, who each control 50 sybils, are attacking a system. To that extent, their system is under a variety of potentially conflicting attacks. Our work differs on two main points: first, we evaluate a system that iteratively updates and computes personalised recommendations for each user. We also propose methods that assume a large set of users and items, and flag attacks while monitoring all users and items (rather than simply monitoring users/items individually). We evaluate attacks that may not demonstrate anomalies within a single time window, but appear between system updates, and may be targetted to affect particular users' recommendations. We also explore a wide variety of attacks, ranging from the *random* to *targetted* scenarios, where a key aspect of the attacks is the fact that sybil *groups* of varying size are rating items. There are a number of other particular strategies that attackers may adopt (such as the bandwagon or average attacks strategies [MBW07]) when unleashing a set of sybils that we have not explored above. Our detection mechanisms, in focusing on complementary dimensions of attacks (the *group size* and *rate* of sybils as they attack) hope to detect attacks regardless of the adopted strategy.

The idea of temporality in attacks has also been explored from the point of view of user reputation; Resnick and Sami [RS07] prove a variety of properties of their reputation system, which takes into account the order in which ratings are input. It remains unclear how these systems would work in practice: many reputation or trust-based systems assume that the ratings input by users are the ground truth, without taking into account that users are both naturally inconsistent when they rate [APTO09] and what they rate will be influenced by what they are recommended. Furthermore, one of the most troubling problems that both monitoring techniques and reputation systems suffer from is *bootstrapping*; systems can be easily abused when the variables that monitor or reflect user behaviour have had little to no data. We use all ratings input prior to a pre-specified time ϵ to bootstrap each monitor. System administrators may opt to ask a controlled, closed group of trusted users to rate for varying lengths of time in order to bootstrap [ALP+09]. Alternatively, if the system also offers social networking functionality, defences that identify sybils according to their location on the social graph can be applied [DC08]; in this work we assumed that no such graph was present.

While it is often the case that designing security mechanisms is context-sensitive, there are lessons that we learn here that may be applicable to other scenarios. For example, Yu *et al.* [YSK+09] design a method for recommender systems with binary-voting like Digg, and demonstrate the ability to fend off attacks where the number of sybils greatly exceeds the number of honest users. While our work focuses on the ordinal-scale rating-based Netflix data, the similarity between the two contexts is the need for sybils to outweigh honest user behaviour in order to achieve malicious goals, and doing so in tandem is a key insight into detecting their misbehaviour.

6.7 Summary

In this chapter, we have confronted the problem of sybil attacks to recommender systems. The focal point of the contributions we make here is that sybils are detectable not only via *what* they rate (as state of the art sybil classifiers learn from) but also by *how* they insert these malicious ratings. In fact, the mere act of casting the problem of recommender system robustness onto a temporal scale already makes it harder for attackers to meet their goals: they can no longer simply dump ratings into a system and expect these to have any effect. Furthermore, the actions of the system's honest users can be leveraged in order to identify the automated attacks. We introduced a windowed-view of temporal behaviour, defined the notion of temporal attacks, and then designed and evaluated a *global, user,* and *item* monitor that flags when different forms of attack are taking place: where sybil groups (of varying size) inject item ratings (at varying rates) over time in order to either disrupt the system's recommendations (via a *random* attack) or modify the recommendations of a particular item (with a *targetted* attack).

Lastly, we paved the way for future research by describing how attackers may overcome these defences; namely, by performing a *ramp-up* attack that can fool the defences into believing that no attack is taking place. We compared the effects of a ramp-up attack, when the system is defended by our algorithms, and a one-shot attack on a system with no defences, and concluded that the ramp-up attack is not as immediately effective as an attack on an undefended system: our methods thus increase the time and effort that attackers require to accomplish their goals. Future work should thus investigate how recommender systems can identify an ongoing ramp-up attack and adapt the system's defences accordingly.

Chapter 7

Conclusion

This thesis is grounded on an important observation: there is a disparity between how collaborative filtering is reseached and how it is deployed. The majority of research treats the scenario as a *static* problem: given a dataset, the quality of a particular algorithm's recommendations (measured as accuracy or precision) can be evaluated by training and testing the algorithm with partitions of all the data. Deployed recommender systems, instead, have to cope with a continuous influx of ratings, users, and content. The underlying data changes in size, sparsity, and may even become distributed differently; changes occur that affect performance and can neither be reproduced nor examined under static conditions.

Once the assumption of a static context has been removed, the methodology used to investigate CF needs to be redefined. In Chapter 4, we introduced a novel means of doing so, based on partitioning the data according to rating timestamps and simulating a deployed system by iteratively retraining CF algorithms with incrementally larger portions of the data. There are then a number of novel directions and uncovered results that can be examined when researching collaborative filtering. We have focused on three aspects: recommender systems' temporal accuracy, diversity, and robustness. Each aspect is highly significant: while accuracy has been the focal point of CF evaluation (and the primary tool for comparing algorithms), temporal experiments show that the way accuracy varies with time undermines the usefulness of work comparing algorithms solely on these grounds. Temporal diversity could not be explored from a static perspective, yet (especially in the case where it is missing) elicits passionate responses from surveyed users. Lastly, we determined that learning temporal behaviour and monitoring it for anomalies not only wards off a number of recommender system attacks, but forces attackers to select strategies that are both more costly (in terms of time taken to execute the attack) and less efficient. In the following section, we summarise the contributions we have made.

7.1 Thesis Contributions

At the broadest level, the contributions of this thesis fall into one of three categories:

- **Analysis.** We have shown, through extensive analysis of real user ratings, how CF data changes, including how summary statistics, similarity, and user behaviour fluctuate over time. While different datasets grow at varying rates, they all grow: observing these changes strongly motivates research into how the system *as a whole* performs over time.

- **Methodology.** We have designed a novel methodology for performing temporal collaborative filtering experiments. This method relies on partitioning the data according to the ratings' timestamps and incrementally growing the size of the training set.

- **Algorithms.** We have designed and evaluated hybrid CF algorithms that (a) increase the temporal accuracy, (b) bolster temporal diversity, and (c) secure recommender systems from temporal attacks.

The contributions of this thesis are relevant to (a) *researchers*, who are now challenged to build algorithms that stand the test of time (as well as those imposed by traditional evaluation metrics), and (b) *practicioners*, who may wish to augment their systems with features of this work, by, for example, overlaying a re-ranking algorithm on their CF prediction method. Recent work by Burke [Bur10] furthers the call for dynamic, temporal evaluations of CF by proposing a methodology that is similar, yet finer-grained, than the one we used throughout this thesis.

A general theme emerges from the algorithmic proposals we have made: whether we were focusing on improving accuracy, diversity, or robustness, our solutions proposed to treat users differently from one another. For example, the user-based switching algorithm (Chapter 4) improved overall accuracy by trying to improve each user's accuracy independently of the others; a similar solution was adopted to improve temporal diversity. When it came to defending a recommender system (Chapter 6), part of our proposal was monitoring users and comparing how they behaved with respect to the rest of the community, in order to identify misbehaving sybils. The centrality of users in our proposals reflects the variety of roles that users adopt when interacting with a recommender system: while some users are purely consuming content with the goal of obtaining better recommendations for themselves, other users are actually driven by a desire to help others' recommendations [HKTR04]. The key insight here is that there is a difference between the various system users; they are not all the same. CF research, on the other hand, has ignored this insight and designed "one-size fits all" solutions. In this thesis, we have departed from this approach by testing algorithms that vary how they compute predictions for different users.

The work we have done here is inherently limited by the data that we have used. Recommender systems may span a variety of different domains (both on and off the web and for a wide range of different types of items); however, our datasets only reflect online movie rental web sites. Our work has therefore focused on scenarios where users *explicitly* rate content (we do not use any implicit data). The assumption we hold is that these datasets are sufficiently *representative* of large scale recommender systems, and conclusions that we draw when analysing them are similarly applicable to other recommender system domains.

7.2 Future Work

As we saw in Table 2.1 (Chapter 2), the research problems relating to recommender systems are not limited to those we have addressed in this thesis. In this section, we discuss opportunities for future research. We divide them into two categories: the direct consequences of the methodology we have used throughout this thesis and broader considerations on state-of-the-art CF research.

7.2.1 Using a Temporal Methodology

In this thesis, we focused on accuracy, diversity and robustness from a temporal perspective. The *approach* that we have defined, however, also offers a novel perspective on many other research challenges: attempting to solve them using a temporal methodology is likely to offer insights that were previously unavailable. Examples include:

The Cold-Start Problem. We explored the effect that highly connected users have on predictive accuracy and coverage and validated that only using them still achieves comparable accuracy results (Chapter 3). Cold-start users, who have no profile, may therefore be given a neighbourhood of these users until their profiles are sufficiently large to compute a similarity-based neighbourhood. Can these highly connected users be identified as they rate? To what extent do they vary over time and what effect does any variation have on system performance?

Serendipity. Being able to identify users who consistently seek out and rate new content may help finding the sources of *serendipitous* information. On the other hand, serendipitous ratings may be more prevalent in the sparser profiles. Herein lies a two-fold research problem: first, how can serendipity be measured? Second, is it possible to identify those who are the source of new ratings, trends, and who first rate what will later become popular content? Richer datasets may also offer finer-grained insights. In particular, recent work on multidimensional recommender systems may show why power-users emerge, and how they can best be used [ASST05].

Scalability. The mainstream approaches used to tackle the large number of users involves dimensionality reduction techniques [BK07]. Temporal patterns in neighbourhoods, however, can be taken advantage of to reduce the complexity of recomputing the similarity between every user pair. Identifying the active users in a particular moment can potentially be used to reduce the time complexity of computing recommendations. Furthermore, as discussed in Chapter 3, it is often the case that large proportions of the dataset are not used to generate predictions at all. Identifying and requiring only a small set of power users to generate accurate predictions would vastly reduce the scalability issues that recommender systems face [ALP$^+$09].

Combining Multiple Goals. CF research has traditionally placed a high value on accuracy; in this thesis we designed mechanisms to augment accuracy over time. However, we also noted that both diversity and robustness are equally important. While our proposal regarding temporal monitors to secure recommender system robustness does not interfere with any underlying prediction or ranking algorithm, the diversity and accuracy algorithms may conflict with one another. Future work thus calls for designing and evaluating CF algorithms that meet a variety of requirements; for example, that they produce recommendations that are both (temporally) accurate and diverse. A simple approach to this particular example may entail using the switching algorithm (Chapter 4) to improve accuracy, while re-ranking the top-N recommendations (Chapter 5) in order to diversify the results: this approach ensures both properties without interfering with one another. However, there are likely to be more qualities that users seek from their recommendations, and it is likely that they cannot all be optimised independently of one another.

7.2.2 Beyond Temporal Collaborative Filtering

The themes that emerge from this thesis call for a focus on three key areas in future recommender system research: the *users*, the system *context*, and the *social* aspect of recommender systems.

The methodology that we have used throughout this thesis reflects a *process* that occurs when recommender systems are deployed: users first *rate* content; the ratings are used to compute *predictions*; predictions are ranked to form *recommendations*, and recommendations incentivise users to rate more content. In general, CF research has tended to focus on only two of these three steps. The majority of research (along with the Netflix prize) is dedicated to the problem of computing predictions using ratings. More recently, the importance of ranking (and thus the second step—converting predictions into a ranked list) has emerged, and recommender systems have been evaluated using a variety of information retrieval metrics [ALP$^+$09, Kor08]. However, the last step remains unexplored: given a set of recommendations, why do people rate as they do? Do they rate their recommendations or seek out different content? What affects the way they rate? When confronting the problem of temporal diversity (Chapter 5), we began to see how ratings are not simply reflections of what each user thinks of the *movies* presented to them; the ratings also reflect the users' response to the recommender system and their impression of how well the recommendations are tailored to their needs. A major gap in recommender system research is the focus on the end users' behaviour. While CF has been, for the most part, interesting from the machine learning perspective, it is ultimately an algorithm that has to provide recommendations to people, and further understanding of how the people behave will feed back and improve the algorithms themselves.

A related problem that persists in CF research is that of evaluation: how can researchers demonstrate that their systems are producing "good" results? To date, we have done so by making assumptions of what "good" means: accurate predictions and high precision and recall. In this thesis, we extend that to include, for example, temporally diverse recommendations. We motivated this addition by asking users what they thought of a system that was not temporally diverse. However, what else do users want from their recommendations? The evaluation criteria themselves may be subject to the context in which the recommender system is operating: temporal diversity may make complete sense for a web-based movie recommender system, but may be inappropriate for a system that recommends travel routes to a commuter, unless the diversity is motivated with further reasons (e.g., the current route is congested). On a broader level, systems can be better evaluated if we understand where they will be operating. If future recommender system research focuses on *context*, novel evaluation methods will emerge: for example, to what extent does a travel recommender system on users' mobiles affect their mobility patterns? In other words, are the computed recommendations turning into useful actions?

A final consideration we include is the *social* aspect. There is an overlap between social networks and collaborative filtering; in fact, we were able to draw from social network analysis techniques in order to examine how similarity graphs are structured (Chapter 3). In doing so, we claimed that CF rating data represents an *implicit* social network between the system users, because what one person rates affects others' recommendations. The implication here is that CF ratings are related to one another;

in fact, there may be a causal relationship between users' ratings. These relationships are difficult to understand since the links between users remain hidden; however, future research based on combined (social network/content ratings) datasets will be able to investigate this link further and use it to improve the recommendations each user is given. Recent web-based companies are already gathering data that will serve this purpose: for example, Rummble[1] gathers users' *social network* and *ratings* on different locations around the world.

[1] http://www.rummble.com

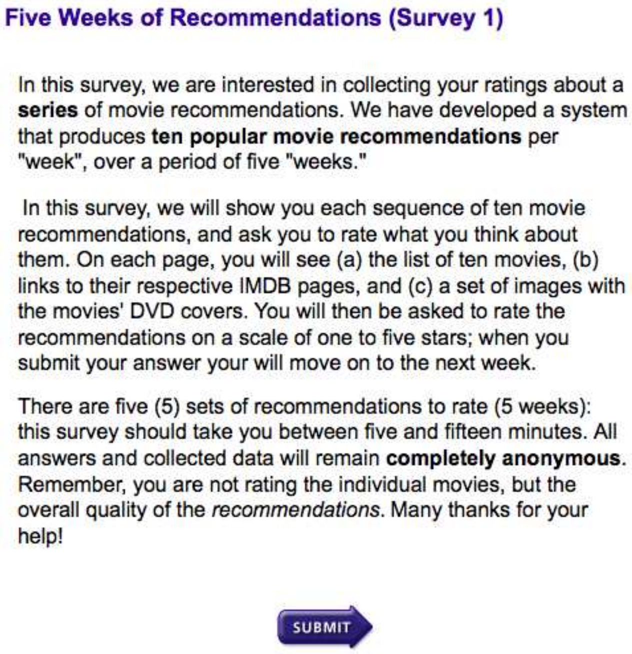
Appendix A

Diversity Surveys

The surveys in Chapter 5 were created using Zoomerang[1]. Users were recruited via Twitter, Facebook and departmental mailing lists and directed to a web page[2] that allowed them to either (a) be taken to a randomly selected survey or (b) access each survey individually.

A.1 Pre-Survey Instructions and Demographics

The users were first given a set of instructions about the survey. These were the same across all three surveys. Figure A.1 show an example screen shot from Survey 1.



Five Weeks of Recommendations (Survey 1)

In this survey, we are interested in collecting your ratings about a **series** of movie recommendations. We have developed a system that produces **ten popular movie recommendations** per "week", over a period of five "weeks."

In this survey, we will show you each sequence of ten movie recommendations, and ask you to rate what you think about them. On each page, you will see (a) the list of ten movies, (b) links to their respective IMDB pages, and (c) a set of images with the movies' DVD covers. You will then be asked to rate the recommendations on a scale of one to five stars; when you submit your answer your will move on to the next week.

There are five (5) sets of recommendations to rate (5 weeks): this survey should take you between five and fifteen minutes. All answers and collected data will remain **completely anonymous**. Remember, you are not rating the individual movies, but the overall quality of the *recommendations*. Many thanks for your help!

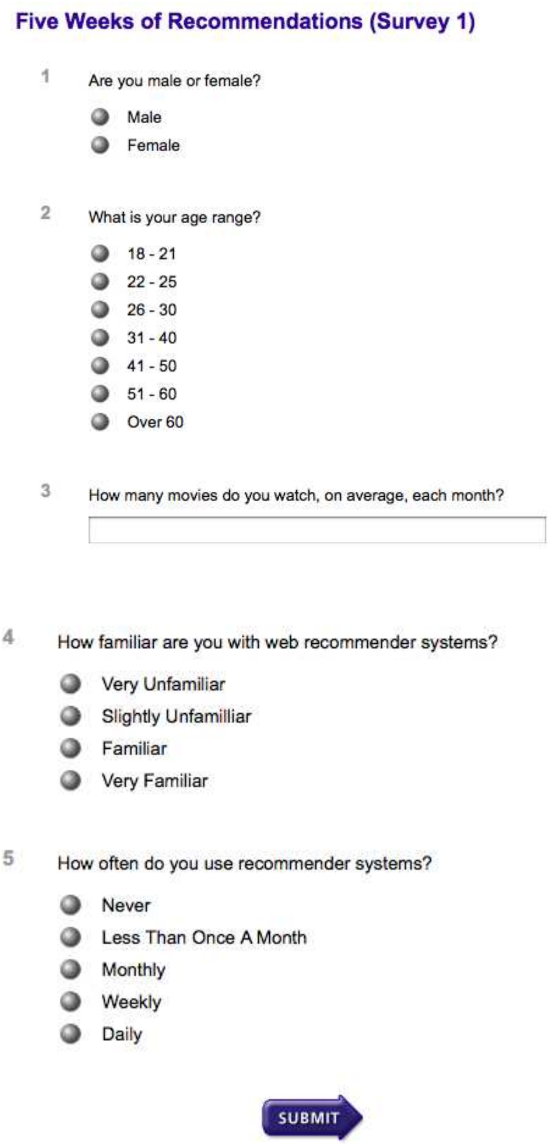
Figure A.1: Example User Instructions from Survey 1

The users were then asked to input some demographic data. The questions related to gender, age, number of movies watched per month, and familiarity with recommender systems.

Users were not asked for any personally identifiable information (e.g., email address), in order to

[1] http://www.zoomerang.com

[2] http://www.cs.ucl.ac.uk/staff/n.lathia/survey.html



Five Weeks of Recommendations (Survey 1)

1 Are you male or female?

○ Male
○ Female

2 What is your age range?

○ 18 - 21
○ 22 - 25
○ 26 - 30
○ 31 - 40
○ 41 - 50
○ 51 - 60
○ Over 60

3 How many movies do you watch, on average, each month?

4 How familiar are you with web recommender systems?

○ Very Unfamiliar
○ Slightly Unfamilliar
○ Familiar
○ Very Familiar

5 How often do you use recommender systems?

○ Never
○ Less Than Once A Month
○ Monthly
○ Weekly
○ Daily

Figure A.2: Demographic Data Questions: Gender, Age, Average Movies Per Month, Familiarity and Use of Recommender Systems

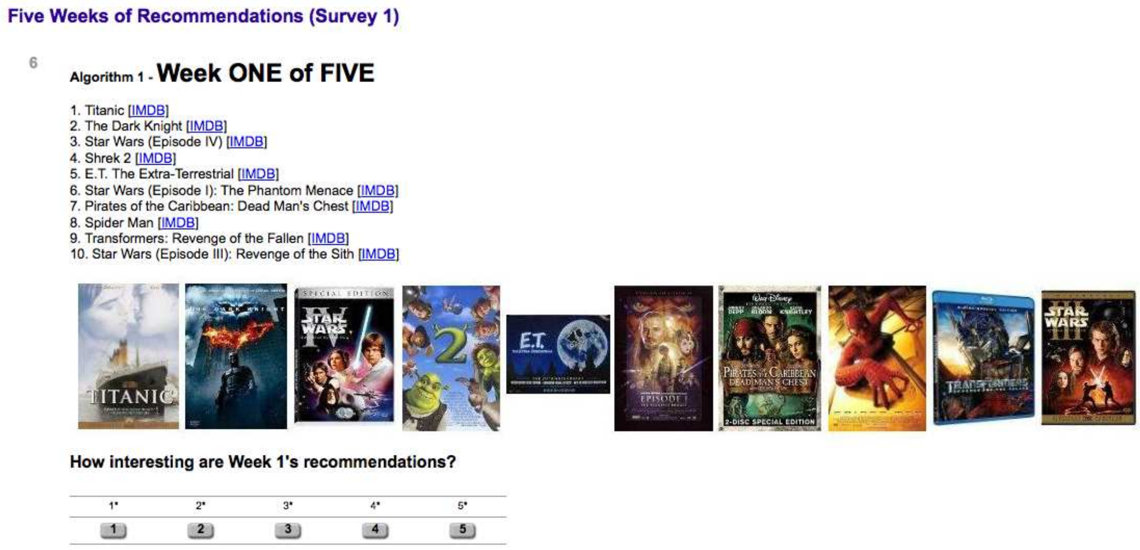
guarantee full anonymity. A consequence of this decision was that we were unable to know how many users completed more than one survey.

A.2 Movie Recommendations

Since all surveys were similarly structured, we first describe how the movie recommendations were presented to the users (Section A.2.1). We then specify the content of each survey: Survey 1, the non-diversified popular movies (Section A.2.2), Survey 2, the diversified popular movies (Section A.2.3) and Survey 3, the diversified randomly-selected movies (Section A.2.4).

A.2.1 Recommendation Structure

As described in the instructions, the users were first shown a page with (a) a list of movie titles, (b) links to each movies' IMDB information pages and (c) a set of images of the movies' DVD covers: an example screen shot is shown in Figure A.2.1. The users were then asked to rate how interesting they found the week's recommendations on a 1-5 star Likert scale.



Five Weeks of Recommendations (Survey 1)

6

Algorithm 1 - Week ONE of FIVE

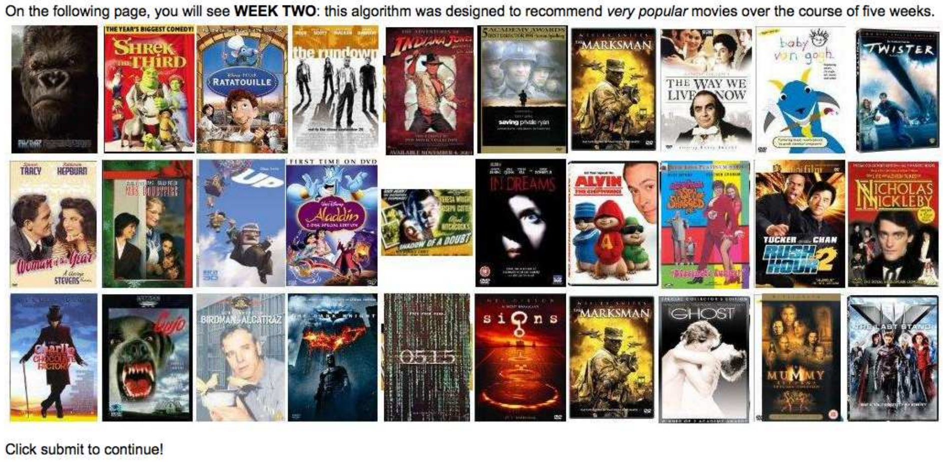
1. Titanic [IMDB]
2. The Dark Knight [IMDB]
3. Star Wars (Episode IV) [IMDB]
4. Shrek 2 [IMDB]
5. E.T. The Extra-Terrestrial [IMDB]
6. Star Wars (Episode I): The Phantom Menace [IMDB]
7. Pirates of the Caribbean: Dead Man's Chest [IMDB]
8. Spider Man [IMDB]
9. Transformers: Revenge of the Fallen [IMDB]
10. Star Wars (Episode III): Revenge of the Sith [IMDB]

How interesting are Week 1's recommendations?

1*	2*	3*	4*	5*
1	2	3	4	5

Figure A.3: Example Screen Shot: Survey 1, Week 1

After rating the recommendations, the users would click through to a buffer screen, which contained a number of DVD covers. An example buffer screen is shown in Figure A.2.1. They would then click through into the next "week's" recommendations, structured as above in Figure A.2.1.



On the following page, you will see **WEEK TWO**: this algorithm was designed to recommend *very popular* movies over the course of five weeks.

Click submit to continue!

Figure A.4: Example Screen Shot: Survey 1, Buffer Screen 1

A.2.2 Survey 1: No Diversity

Survey 1 presented the users with the *same* recommendations for each week. The movies were selected from the IMDB[3] list of all-time worldwide box office hits (accessed December 2009) and are summarised in Table A.1.

Rank	Movie
1	Titanic
2	The Dark Knight
3	Star Wars (Episode IV)
4	Schrek 2
5	E.T. The Extra Terrestrial
6	Star Wars Episode I: The Phantom Menace
7	Pirates of the Caribbean: Dead Man's Chest
8	Spiderman
9	Transformers: Revenge of the Fallen
10	Star Wars Episode III: Revenge of the Sith

Table A.1: S1 (All 5 Weeks): All Time Worldwide Box Office Ranking (December 2009)

A.2.3 Survey 2: Diversified Popular Movies

Survey 2 also contained popular movies from IMDB; this set are shuffled as described in Chapter 5.

Rank	Movie
Week One	
1	The Lord of the Rings: The Return of the King
2	Spider-Man 2
3	The Passion of the Christ
4	Jurassic Park
5	Finding Nemo
6	The Lion King
7	Forrest Gump
8	Pirates of the Caribbean: Dead Man's Chest
9	Transformers: Revenge of the Fallen
10	Iron Man
Week Two	
1	Titanic
2	The Dark Knight
3	Star Wars (Episode IV)
4	Home Alone
5	The Bourne Ultimatum
6	Finding Nemo
7	The Da Vinci Code
8	Forrest Gump
9	Pirates of the Caribbean: Dead Man's Chest
10	Kung Fu Panda

Table A.2: S2 (Weeks 1, 2): Diversified All Time Worldwide Box Office Ranking

[3] http://www.imdb.com/boxoffice/alltimegross?region=world-wide

Rank	**Movie**
Week Three	
1	Pirates of the Caribbean: Dead Man's Chest
2	E.T. The Extra-Terrestrial
3	Independence Day
4	The Sixth Sense
5	Wedding Crashers
6	Terminator 2
7	300
8	Titanic
9	The Dark Knight
10	Shrek 2
Week Four	
1	300
2	Jurassic Park
3	Forrest Gump
4	Spider-Man
5	The Lord of the Rings: The Return of the King
6	The Lion King
7	The Passion of the Christ
8	The Sixth Sense
9	Iron Man
10	Wedding Crashers
Week Five	
1	Iron Man
2	Finding Nemo
3	Wedding Crashers
4	The Sixth Sense
5	Transformers: Revenge of the Fallen
6	Terminator 2
7	300
8	Forrest Gump
9	The Lion King
10	Shrek 2

Table A.3: S2 (Weeks 3, 4, 5): Diversified All Time Worldwide Box Office Ranking

Note that *diversity* does not necessarily imply no repetition of recommendations; in this set of movies, a number of entries appear more than once (in different ranks). For example, *"Pirates of the Caribbean: Dead Man's Chest"* appears in the first three weeks, in ranks 8, 9, and 1 respectively.

A.2.4 Survey 3: Diversified Random Movies

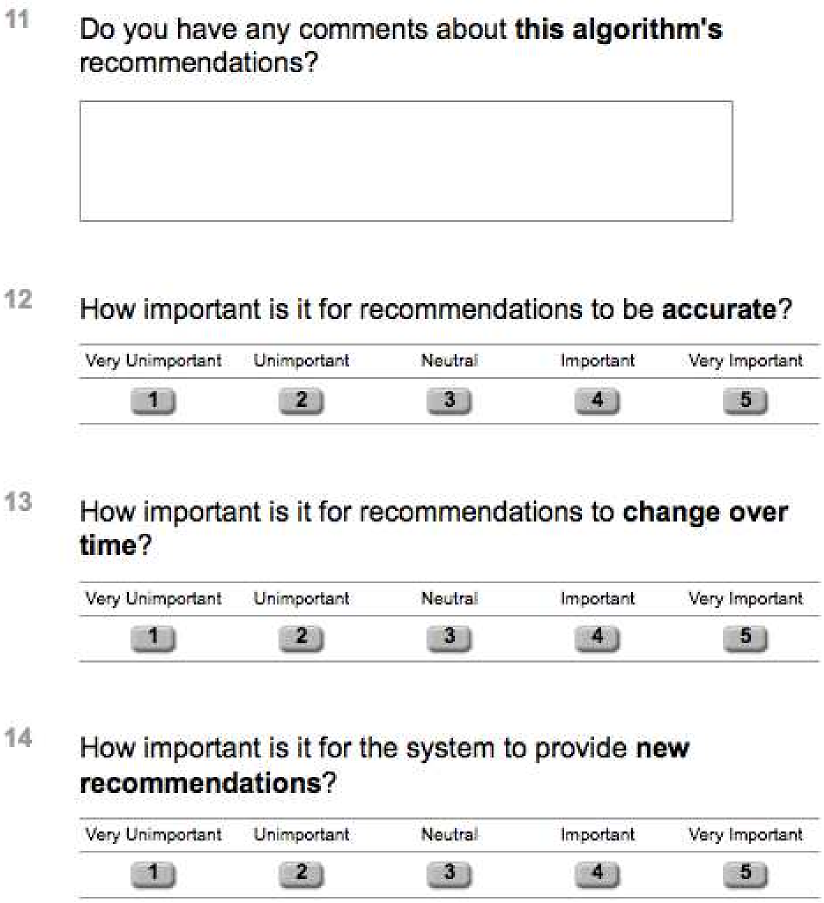
The last set of movies were selected uniformly at random from the Netflix prize dataset. The only condition to be met when selecting a set of movies was that the same movie may not appear twice in the *same* top-10 list. The 5 weeks of recommendations are listed in Table A.4.

Rank	Movie	Rank	Movie
Week One		**Week Two**	
1	Woman of the Year	1	Nightbreed
2	Cujo	2	Predator Island
3	Birdman of Alcatraz	3	Bad Company
4	The Rundown	4	Holiday Heart
5	Shadow of Doubt	5	Jurassic Park III
6	In Dreams	6	Devo Live
7	The Marksman	7	Pursued
8	The Way We Live Now	8	Lionheart
9	Baby Van Gogh	9	Antibody
10	Nicholas Nickleby	10	It Came From Outer Space
Week Three		**Week Four**	
1	G-Men From Hell	1	A Stranger Among Us
2	The Marriage Circle	2	Soul Assassin
3	Harry Potter and the Sorcerer's Stone	3	Jane Eyre
4	Baby Einstein: Baby Da Vinci	4	Annie Lennox: Live in Central Park
5	Island of Dr. Moreau	5	The Magic Flute
6	My Voyage to Italy	6	The Hills Have Eyes 2
7	Koma	7	A Better Tomorrow II
8	The Toy	8	Atomic Train
9	Bulletproof	9	Speed Racer
10	The Englishman Who Went Up a Hill but Came Down a Mountain	10	Vampires: The Turning
Week Five			
1	Crash		
2	Riding the Bullet		
3	Kicked in the Head		
4	Diary of a Serial Killer		
5	Oh God!		
6	French Twist		
7	Degrassi Junior High		
8	Black Adder		
9	Red Dirt		
10	Frequency		

Table A.4: S3 (Weeks 1, 2, 3, 4, 5): Randomly Selected Movies

A.3 Post-Survey Questions

After completing the 5 weeks of recommendations, the users were asked to comment on the quality of the recommendations. A screen shot of the questions is shown in Figure A.3.



11 Do you have any comments about **this algorithm's** recommendations?

12 How important is it for recommendations to be **accurate**?

Very Unimportant	Unimportant	Neutral	Important	Very Important
1	2	3	4	5

13 How important is it for recommendations to **change over time**?

Very Unimportant	Unimportant	Neutral	Important	Very Important
1	2	3	4	5

14 How important is it for the system to provide **new recommendations**?

Very Unimportant	Unimportant	Neutral	Important	Very Important
1	2	3	4	5

Figure A.5: Example Screen Shot: Survey 1, Final Questions

Bibliography

[AG06] O. Alonso and M. Gertz. Clustering of Search Results Using Temporal Attributes. In *ACM SIGIR*, pages 597–598, Seattle, USA, August 2006.

[AGHI09] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying Search Results. In *ACM WSDM*, pages 5–14, Barcelona, Spain, 2009.

[AH08] M. Ahmed and S. Hailes. A Game Theoretic Analysis of the Utility of Reputation Management. *Technical Report RN/08/05, Department of Computer Science, University College London*, January 2008.

[AJOP09] X. Amatriain, A. Jaimes, N. Oliver, and J.M. Pujol. Data Mining Methods for Recommender Systems. In Kantor, Ricci, Rokach, and Shapira, editors, *Recommender Systems Handbook*. Springer, August 2009.

[Alp04] E. Alpaydin. *Introduction to Machine Learning*. MIT Press, Massachusetts, USA, 2004.

[ALP$^+$09] X. Amatriain, N. Lathia, J.M. Pujol, H. Kwak, and N. Oliver. The Wisdom of the Few: A Collaborative Filtering Approach Based on Expert Opinions from the Web. In *ACM SIGIR*, pages 532–539, Boston, USA, July 2009.

[APO09] X. Amatriain, J.M. Pujol, and N. Oliver. I Like It...I Like it Not: Evaluating User Ratings Noise in Recommender Systems. In *User Modeling, Adaptation, Personalization (UMAP)*, pages 247–258, Trento, Italy, 2009.

[APTO09] X. Amatriain, J.M. Pujol, N. Tintarev, and N. Oliver. Rate it Again: Increasing Recommendation Accuracy by User Re-Rating. In *ACM RecSys*, pages 173–180, New York, USA, 2009.

[ARH97] A. Abdul-Rahman and S. Hailes. Using recommendations for managing trust in distributed systems. In *Proceedings of IEEE Malaysia International Conference on Communication*, November 1997.

[ARH98] A. Abdul-Rahman and S. Hailes. A Distributed Trust Model. In *New Security Paradigms Workshop*, pages 48–60, Langdale, United Kingdom, 1998.

[ASST05] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach. *ACM Transactions on Information Systems*, 23(1):103–145, January 2005.

[AT05] G. Adomavicius and A. Tuzhilin. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE TKDE*, 17(6):734–749, June 2005.

[AW97] A. Agresti and L. Winner. Evaluating Agreement and Disagreement Among Movie Reviewers. *Chance*, 10:10–14, 1997.

[AWWY99] C. Aggarwal, J.L. Wolf, K. Wu, and P.S. Yu. Horting Hatches and Egg: A New Graph Theoretic Approach to Collaborative Filtering. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 201–212, San Diego, California, USA, 1999.

[BA02] A. Barabasi and R. Albert. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[BEKR07] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing Privacy and Preserving Accuracy of Distributed Collaborative Filtering. In *Proceedings of Recommender Systems (RecSys '07)*, pages 9–16, Minneapolis, USA, 2007.

[BHK98] J.S. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Tech Rep. No. MSR-TR-98-12*, Microsoft Research, Redmond, WA, USA, 1998.

[BK07] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *IEEE International Conference on Data Mining (ICDM)*, pages 43–52. IEEE, 2007.

[BLW$^+$04] G. Beenen, K. Ling, X. Wang, K. Chang, D. Frankowski, P. Resnick, and R.E. Kraut. Using Social Psychology to Motivate Contributions to Online Communities. In *Computer Supported Cooperative Work (CSCW)*, pages 212–221, Chicago, USA, November 2004.

[Bon04] P. Bonhard. Improving Recommender Systems with Social Networking. In *Addendum of CSCW*, Chicago, USA, November 2004.

[BR99] E. Blanzieri and F. Ricci. A Minimum Risk Metric for Nearest Neighbour Classification. In *Sixteenth International Conference on Machine Learning*, pages 22–31, Bled, Slovenia, June 1999.

[Bur02] R. Burke. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, 2002.

[Bur10] R. Burke. Evaluating the Dynamic Properties of Recommendation Algorithms. In *ACM Recommender Systems (RecSys)*, Barcelona, Spain, 2010.

[BWMB06] R. Bhaumik, C. Williams, B. Mobasher, and R. Burke. Securing Collaborative Filtering Against Malicious Attacks Through Anomaly Detection. In *AAAI Workshop on Intelligent Techniques for Web Personalization*, Boston, July 2006.

[Can02] J. Canny. Collaborative Filtering With Privacy Via Factor Analysis. In *ACM SIGIR*, pages 238–245, Tampere, Finland, 2002.

[CC08] O. Celma and P. Cano. From Hits to Niches? Or How Popular Artists Can Bias Music Recommendation and Discovery. In *Proceedings of the* 2^{nd} *Netflix-KDD Workshop*, Last Vegas, USA, August 2008.

[Cha02] M. Charikar. Similarity Estimation Techniques From Rounding Algorithms. In *Annual ACM Symposium on Theory of Computing*, pages 380–388, Montreal, Canada, 2002. ACM Press.

[CL07] O. Celma and P. Lamere. Music Recommendation Tutorial. In *Presented at the 8th International Conference on Music Information Retrieval*, Vienna, Austria, September 2007.

[CNS03] M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *Int. Conference on Software Engineering and Formal Methods (SEFM)*, pages 54–63, Brisbane, Australia, September 2003.

[CS01] K. Crammer and Y. Singer. Pranking with Ranking. In *Neural Information Processing Systems (NIPS)*, pages 641–647, Vancouver, Canada, 2001.

[DC08] M. Dell'Amico and L. Capra. SOFIA: Social Filtering for Robust Recommendations. In *Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM)*, Trondheim, Norway, 2008.

[DDGR07] A. Das, M. Datar, A. Garg, and S. Rajaram. Google News Personalization: Scalable Online Collaborative Filtering. In *Proceedings of WWW Industrial Practice and Experience Track*, pages 271–280, Banff, Alberta, Canada, May 2007.

[Die00] T. Dietterich. Ensemble Methods in Machine Learning. In *1*st *International Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000.

[DL05] Y. Ding and X. Li. Time Weight Collaborative Filtering. In *ACM CIKM*, pages 485–492, Bremen, Germany, November 2005.

[FO95] C. Faloutsos and D. Oard. A Survey of Information Retrieval and Filtering Methods. In *Tech. Rep. No. CS-TR-3514*, Department of Computer Science, University of Maryland, Maryland, USA, 1995.

[Gam90] D. Gambetta. Can We Trust Trust? *Trust: Making and Breaking Cooperative Relations*, pages 213–238, 1990.

[GFM05] M. Grcar, B. Fortuna, and D. Mladenic. KNN versus SVM in the Collaborative Filtering Framework. In *Workshop on Knowledge Discovery on the Web*, Chicago, USA, August 2005.

[GNOT92] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry. Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, 35:61–70, 1992.

[Gol06] J. Golbeck. Generating Predictive Movie Recommendations from Trust in Social Networks. In *Fourth International Conference on Trust Management*, pages 93–104, Pisa, Italy, May 2006.

[Gol08] J. Golbeck, editor. *Computing With Social Trust*. Springer, 2008.

[GRGP00] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A Constant Time Collaborative Filtering Algorithm. In *Tech Rep. No. UCB/ERL M00/41*, EECS Department, University of California, Berkeley, California, USA, 2000.

[HC07] C-S. Hwang and Y-P. Chen. Using Trust in Collaborative Filtering Recommendation. In *New Trends in Applied Artificial Intelligence*, pages 1052–1060, 2007.

[HJAK05] F. M. Harper and Y. Chen J. A. Konstan, X. Li. User Motivations and Incentive Structures in an Online Recommender System. In *Incentive Mechanisms in Online Systems, Group 2005 Workshop*, 2005.

[HKBR99] J. Herlocker, J. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *ACM SIGIR*, pages 230–237, Berkley, CA, USA, 1999.

[HKR00] J. Herlocker, J.A. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *In proceedings of ACM 2000 Conference on Computer Supported Cooperative Work*, 2000.

[HKTR04] J. Herlocker, J. Konstan, L. Terveen, and J. Riedl. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*, 22:5–53, 2004.

[HKV08] Y. Hu, Y. Koren, and C. Volinsky. Collaborative Filtering For Implicit Feedback Datasets. In *ICDM*, pages 263–272, Pisa, Italy, 2008. IEEE.

[HRT09] D. Hawking, T. Rowlands, and P. Thomas. C-TEST: Supporting Novelty and Diversity in Testfiles for Search Evaluation. In *ACM SIGIR Workshop on Redundancy, Diversity and Interdependent Document Relevance*, Boston, USA, 2009.

[JIB07] A. Josang, R. Ismail, and C. Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. *Decision Support Systems*, 43(2):618–644, 2007.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31:264–323, 1999.

[KBC07] M. Kurucz, A. A. Benczur, and K. Csalogany. Methdods for Large Scale SVD With Missing Values. In *Proceedings KDD Cup and Workshop*, San Jose, California, August 2007.

[KMM$^+$97] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40:77–87, 1997.

[KNT06] R. Kumar, J. Novak, and A. Tomkins. Structure and Evolution of Online Social Networks. In *International Conference on Knowledge Discovery and Data Mining*, pages 611–617, 2006.

[Kor08] Y. Koren. Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In *ACM KDD*, pages 426–434, 2008.

[Kor09a] Y. Koren. Collaborative Filtering with Temporal Dynamics. In *ACM KDD*, pages 89–97, Paris, France, 2009.

[Kor09b] Y. Koren. The BellKor Solution to the Netflix Grand Prize. In *Netflix Prize Report*, August 2009.

[LAP09] N. Lathia, X. Amatriain, and J.M. Pujol. Collaborative Filtering With Adaptive Information Sources. In *IJCAI Workshop on Intelligent Techniques for Web Personalisation and Recommender Systems*, Pasadena, USA, July 2009.

[Lat08a] N. Lathia. *Computing Recommendations With Collaborative Filtering, Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*, chapter 2. IGI-Global, September 2008.

[Lat08b] N. Lathia. Learning to Trust on the Move. In *Joint TIME and SPACE Workshops (IFIPTM)*, Trondheim, Norway, 2008.

[LHC07] N. Lathia, S. Hailes, and L. Capra. Private Distributed Collaborative Filtering Using Estimated Concordance Measures. In *ACM Recommender Systems (RecSys)*, pages 1–8, Minneapolis, USA, 2007.

[LHC08a] N. Lathia, S. Hailes, and L. Capra. kNN CF: A Temporal Social Network. In *ACM Recommender Systems (RecSys)*, pages 227–234, Lausanne, Switzerland, 2008.

[LHC08b] N. Lathia, S. Hailes, and L. Capra. The Effect of Correlation Coefficients on Communities of Recommenders. In *ACM Symposium on Applied Computing (TRECK track)*, pages 2000–2005, Fortaleza, Brazil, March 2008.

[LHC08c] N. Lathia, S. Hailes, and L. Capra. Trust-Based Collaborative Filtering. In *Joint iTrust and PST Conferences on Privacy, Trust Management and Security (IFIPTM)*, Trondheim, Norway, 2008.

[LHC09a] N. Lathia, S. Hailes, and L. Capra. Evaluating Collaborative Filtering Over Time. In *ACM SIGIR Workshop on the Future of IR Evaluation*, Boston, USA, July 2009.

[LHC09b] N. Lathia, S. Hailes, and L. Capra. Temporal Collaborative Filtering With Adaptive Neighbourhoods. In *ACM SIGIR*, pages 796–797, Boston, USA, July 2009.

[LHC10a] N. Lathia, S. Hailes, and L. Capra. Temporal Defenses for Robust Recommendations. In *ECML PKDD Workshop on Privacy and Security Issues in Data Mining and Machine Learning*, Barcelona, Spain, September 2010.

[LHC10b] N. Lathia, S. Hailes, and L. Capra. Temporal Diversity in Recommender Systems. In *ACM SIGIR*, pages 210–217, Geneva, Switzerland, July 2010.

[LK03] Q. Li and B.M. Kim. Clustering Approach for Hybrid Recommender System. In *IEEE/WIC International Conference on Web Intelligence*, page 33, Beijing, China, 2003. IEE Press.

[LM05] D. Lemire and A. Maclachlan. Slope One Predictors for Online Rating-Based Collaborative Filtering. In *SIAM Data Mining Conference*, Newport Beach, California, April 2005.

[LQF10] S.L. Lim, D. Quercia, and A. Finkelstein. StakeNet: Using Social Networks to Analyse the Stakeholders of Large-Scale Software Projects. In *International Conference on Software Engineering*, pages 295–304, Cape Town, South Africa, May 2010.

[LR04] S.K. Lam and J. Riedl. Shilling Recommender Systems for Fun and Profit. In *13th International World Wide Web Conference*, pages 393–402, New York, NY, USA, 2004.

[LSE08] G. Lenzini, N. Sahli, and H. Eertink. Trust Model for High Quality Recommendation. In *International Conference on Security and Cryptography (SECRYPT)*, July 2008.

[LSY03] G. Linden, B. Smith, and J. York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEE Internet Computing*, 7:76–80, 2003.

[MA07] P. Massa and P. Avesani. Trust-aware Recommender Systems. In *ACM Recommender Systems (RecSys)*, pages 17–24, Minneapolis, USA, 2007.

[MAA08] M. Maia, J. Almeida, and V. Almeida. Identifying user profiles in online social networks. In 1^{st} *International Workshop on Social Network Systems (EuroSys)*. ACM Press, 2008.

[Mar94] S. Marsh. Formalising trust as a computational concept. *PhD Thesis, Department of Mathematics and Computer Science University of Stirling UK*, 1994.

[MB04] P. Massa and B. Bhattacharjee. Using Trust in Recommender Systems: An Experimental Analysis. In *iTrust International Conference*, pages 221–235, 2004.

[MBW07] B. Mobasher, R. Burke, and C. Williams. Towards Trustworthy Recommender Systems: An Analysis of Attack Models and Algorithm Robustness. *Transactions on Internet Technology (TOIT)*, 7(4), 2007.

[ME95] D. Maltz and K. Ehrlick. Pointing the Way: Active Collaborative Filtering. In *ACM CHI*, pages 202–209, Denver, Colorado, United States, 1995.

[MH04] M.R. McLaughlin and J. Herlocker. A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience. In *Proceedings of ACM SIGIR*, pages 329–336, Sheffield, United Kingdom, 2004.

[MKL07] H. Ma, I. King, and M.R. Lyu. Effective Missing Data Prediction for Collaborative Filtering. In *Proceedings of ACM SIGIR*, pages 39–46, Amsterdam, Holland, 2007.

[MKL09] H. Ma, I. King, and M.R. Lyu. Learning to Recommend With Social Trust Ensemble. In *ACM SIGIR*, pages 203–210, Boston, MA, USA, July 2009.

[MKR05] B. Miller, J.A. Konstan, and J. Riedl. PocketLens: Toward a Personal Recommender System. *ACM Transactions on Information Systems*, 22(3):437–476, July 2005.

[MLG$^+$03] S.M. McNee, S.K. Lam, C. Guetzlaff, J.A. Konstan, and J. Riedl. Confidence Displays and Training in Recommender System. In *INTERACT IFIP TC13 International Conference on Human-Computer Interaction*, pages 176–183, September 2003.

[MRK06] S.M. McNee, J. Riedl, and J.A. Konstan. Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *Extended Abstracts of ACM CHI*, pages 1097–1101, Montreal, Canada, April 2006.

[Mul06] M. Mull. Characteristics of High-Volume Recommender Systems. In *Recommenders Workshop*, Bilbao, Spain, September 2006.

[MY07] Y. Matsuo and H. Yamamoto. Diffusion of Recommendation through a Trust Network. In *Proceedings of ICWSM*, Boulder, Colorado, USA, 2007.

[MYLK08] H. Ma, H. Yang, M.R. Lyu, and I. King. SoRec: Social Recommendation Using Probabilistic Matrix Factorization. In *ACM CIKM*, pages 931–940, Napa Valley, California, USA, October 2008.

[NDB07] A. Nguyen, N. Denos, and C. Berrut. Improving New User Recommendations With Rule-Based Induction on Cold User Data. In *ACM Recommender Systems (RecSys)*, Minneapolis, USA, 2007.

[OS05] J. O'Donovan and B. Smyth. Trust in Recommender Systems. In *10th International Conference on Intelligent User Interfaces*, pages 167–174. ACM Press, 2005.

[OS06] J. O'Donovan and B. Smyth. Eliciting Trust Values from Recommendation Errors. *International Journal of Artificial Intelligence Tools*, 15(6):945–962, 2006.

[Pat06] A. Paterek. Improving Regularized Singular Value Decomposition For Collaborative Filtering. In *ACM KDD*, Philadelphia, USA, 2006.

[PB07] M.J. Pazzani and D. Billsus. Content-Based Recommendation Systems. *The Adaptive Web*, 4321:325–341, 2007.

[PBH07] M.A Sasse P. Bonhard and C. Harries. The Devil You Know Knows Best: The Case for Integrating Recommender and Social Networking Functionality. In *Proceedings of the 21st British HCI Group Annual Conference*, Lancaster, UK, September 2007.

[PC06] P. Pu and L. Chen. Trust Building with Explanation Interfaces. In *International Conference on Intelligent User Interfaces*, pages 93–100, Sydney, Australia, 2006. ACM.

[PC09] M. Piotte and M. Chabbert. The Pragmatic Theory Solution to the Netflix Grand Prize. In *Netflix Prize Report*, August 2009.

[Pia07] G. Piatetsky. Interview With Simon Funk. *ACM SIGKDD Explorations Newsletter*, 9:38–40, 2007.

[PM97] R. Procter and A. McKinley. Social Affordances and Implicit Ratings for Social Filtering on the Web. In *DELOS Workshop on Collaborative Filtering*, Budapest, Hungary, 1997.

[PM04] G. Pitsilis and L. Marshall. A Model of Trust Derivation from Evidence for Use in Recommendation Systems. In *Technical Report Series, CS-TR-874*. University of Newcastle Upon Tyne, 2004.

[PM05] G. Pitsilis and L. Marshall. *Trust as a Key to Improving Recommendation Systems, Trust Management*, pages 210–223. Springer Berlin / Heidelberg, 2005.

[Pol06] R. Polikar. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems*, 6(3):21–345, 2006.

[Pot08] G. Potter. Putting the Collaborator Back Into Collaborative Filtering. In *Proceedings of the 2^{nd} Netflix-KDD Workshop*, Last Vegas, USA, August 2008.

[PPK05] M. Papagelis, D. Plexousakis, and T. Kutsuras. Alleviating the Sparsity Problem of Collaborative Filtering Using Trust Inferences. In *Proceedings of the 3rd International Conference on Trust Management (iTrust)*, Paris, France, 2005.

[PPM$^+$06] S. Park, D. Pennock, O. Madani, N. Good, and D. DeCoste. Naive Filterbots for Robust Cold-start Recommendations. In *ACM KDD*, Philadelphia, USA, 2006.

[QC09] D. Quercia and L. Capra. FriendSensing: Recommending Friends Using Mobile Phones. In *ACM Recommender Systems (RecSys)*, pages 273–276, New York, USA, October 2009.

[QHC06] D. Quercia, S. Hailes, and L. Capra. B-trust: Bayesian Trust Framework for Pervasive Computing. In 4^{th} *International Conference on Trust Management. LNCS*, pages 298–312, Pisa, Italy, May 2006.

[QHC07] Daniele Quercia, Stephen Hailes, and Licia Capra. Lightweight Distributed Trust Propagation. In *IEEE International Conference on Data Mining*, Omaha, USA, October 2007.

[RAC$^+$02] A.M. Rashid, I. Albert, D. Cosley, S.K. Lam, S.M. McNee, J. Konstan, and J. Rieldl. Getting to Know You: Learning New User Preferences in Recommender Systems. In *International Conference on Intelligent User Interfaces (IUI)*, pages 127–134, Miami, Florida, 2002.

[RD06] F. Radlinski and S. Dumais. Improving Personalized Web Search Using Result Diversification. In *ACM SIGIR*, pages 691–692, Seattle, USA, 2006.

[RL03] I. Ruthven and M. Lalmas. A Survey on the Use of Relevance Feedback for Information Access Systems. *The Knowledge Engineering Review*, 18:95–145, 2003.

[RLKR06] A.M. Rashid, S.K. Lam, G. Karypis, and J. Riedl. ClustKNN: A Highly Scalable Hybrid Model- Memory-Based CF Algorithm. In *ACM KDD*, Philadelphia, Pennsylvania, USA, August 2006.

[ROHS09] R. Rafter, M. O'Mahony, N.J. Hurley, and B. Smyth. What Have the Neighbours Ever Done For Us? A Collaborative Filtering Perspective. In *User Modeling, Adaptation, Personalization (UMAP)*, Trento, Italy, June 2009.

[RS07] P. Resnick and R. Sami. The Influence Limiter: Provably Manipulation Resistant Recommender Systems. In *ACM Recommender Systems (RecSys)*, Minneapolis, USA, 2007. ACM Press.

[RST08] S. Rendle and L. Schmidt-Thieme. Online-Updating Regularized Kernel Matrix Factorization Models for Large-Scale Recommender Systems. In *ACM Recommender Systems (RecSys)*, Lausanne, Switzerland, 2008.

[SBCO09] B. Smyth, P. Briggs, M. Coyle, and M. O'Mahoney. Google Shared. A Case Study in Social Search. In *Proceedings of User Modeling and Personalization(UMAP)*, pages 283–294, Trento, Italy, 2009.

[SBKCM01] C. Shahabi, F. Banaei-Kashani, Y.S Chen, and D. McLeod. Yoda: An Accurate and Scalable Web-based Recommendation System. In *6th International Conference on Cooperative Information Systems*, pages 418–432, Trento, Italy, 2001. Springer.

[Shi09] C. Shirky. It's Not Information Overload. It's Filter Failure. Web 2.0 Expo NY Presentation, November 2009.

[SKKR00] B.M. Sarwar, G. Karypis, J.A. Konstan, and J. Riedl. Application of Dimensionality Reduction in Recommender Systems: A Case Study. In *ACM WebKDD Workshop*, 2000.

[SKKR01] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International World Wide Web Conference*, Hong Kong, China, 2001.

[SKR99] J.B. Schafer, J. Konstan, and J. Riedl. Recommender Systems in E-Commerce. In *ACM Electronic Commerce*, pages 158–166, Denver, Colorado, USA, 1999.

[SKR01] J. Schafer, J. Konstan, and J. Riedl. E-Commerce Recommendation Applications. *The Knowledge Engineering Review*, 5:115–153, 2001.

[SM01] B. Smyth and P. McClave. Similarity vs. Diversity. In *Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 347–361, Vancouver, Canada, 2001.

[SP06] V. A. Siris and F. Papagalou. Application of Anomaly Detection Algorithms for Detecting SYN Flooding Attacks. *Computer Communications*, 29:1433–1442, May 2006.

[TJB09] A. Toscher, M. Jahrer, and R.M. Bell. The Big Chaos Solution to the Netflix Grand Prize. In *Netflix Prize Report*, August 2009.

[TM07] N. Tintarev and J. Masthoff. Effective explanations of recommendations: User-centered design. In *ACM Recommender Systems (RecSys '08)*, Minneapolis, USA, 2007.

[Tof70] A. Toffler. *Future Shock*. Random House, 1970.

[VD02] R. Vilalta and Y. Drissi. A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Review*, 18(2), October 2002.

[WB08] S. X. Wu and W. Banzhaf. Combatting Financial Fraud: A Coevolutionary Anomaly Detection Approach. In *10th Annual Conference on Genetic and Evolutionary Computation*, pages 1673–1680, Atlanta, GA, USA, 2008.

[WBS07] F. E. Walter, S. Battiston, and F. Schweitzer. A Model of a Trust-Based Recommendation System On a Social Network. *Autonomous Agents and Multi-Agent Systems*, 16(1):57–74, February 2007.

[WH07a] F. Wu and B. A. Huberman. Follow the Trend or Make a Difference: The Evolution of Collective Opinions. In *Research Note: Information Dynamics Laboratory*, Palo Alto, CA, USA, 2007. HP Labs.

[WH07b] F. Wu and B.A. Huberman. Public Discourse in the Web Does Not Exhibit Group Polarization. In *Technical Report*, HP Labs Research, Palo Alto, CA, USA, 2007.

[WMB09] C. Williams, B. Mobasher, and R. Burke. Defending Recommender Systems: Detection of Profile Injection Attacks. *Journal of Service Oriented Computing and Applications*, 1(3):157–170, August 2009.

[WMG06] J. Weng, C. Miao, and A. Goh. Improving Collaborative Filtering With Trust-Based Metrics. In *ACM Symposium on Applied Computing*, pages 1860–1864, Dijon, France, 2006.

[YSK$^+$09] H. Yu, C. Shi, M. Kaminsky, P. Gibbons, and F. Xiao. DSybil: Optimal Sybil-Resistance for Recommendation Systems. In *IEEE Symposium on Security and Privacy*, pages 283–298, Oakland, CA, USA, 2009.

[YSKY09] Y. Yang, Y. Sun, S. Kay, and Q. Yang. Defending Online Reputation Systems against Collaborative Unfair Raters through Signal Modeling and Trust. In *ACM SAC TRECK*, pages 1308–1315, Waikiki Beach, Honolulu, Hawaii, USA, 2009.

[YST$^+$04] K. Yu, A. Schwaighofer, V. Tresp, X. Xu, and H. Kriegel. Probabilistic Memory-Based Collaborative Filtering. *IEEE Transactions on Knowledge and Data Engineering*, 16:56–69, 2004.

[YWXE01] K. Yu, Z. Wen, X. Xu, and M. Ester. Feature Weighting and Instance Selection for Collaborative Filtering. In *Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, Munich, Germany, 2001.

[YZLG09] K. Yu, S. Zhu, J. Lafferty, and Y. Gong. Fast Nonparametric Matrix Factorisation for Large-Scale Collaborative Filtering. In *Proceedings of the 32nd ACM SIGIR Conference*, pages 211–218, Boston, MA, USA, 2009. ACM Press.

[ZC08] V. Zanardi and L. Capra. Social Ranking: Filtering Relevant Content in Web 2.0. In *ACM Recommender Systems (RecSys)*, pages 51–58, Lausanne, Switzerland, October 2008.

[ZH08] M. Zhang and N. Hurley. Avoiding Monotony: Improving the Diversity of Recommendation Lists. In *Proceedings of ACM RecSys*, Lausanne, Switzerland, 2008.

[Zie05] C. Ziegler. *Towards Decentralised Recommender Systems (PhD Thesis)*. Department of Computer Science, Freiburg University, Freiburg, Germany, 2005.

[ZWSP08] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, 2008.