Goal-driven Collaborative Filtering

Tamas Jambor

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University College London.

Department of Computer Science
University College London

December 3, 2013

I, Tamas Jambor, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

_____

Tamas Jambor

To my wife, who supported me each step of the way.

Abstract

Recommender systems aim to identify interesting items (e.g. movies, books, websites) for a given user, based on their previously expressed preferences. As recommender systems grow in popularity, a notable divergence emerges between research practices and the reality of deployed systems: when recommendation algorithms are designed, they are evaluated in a relatively static context, mainly concerned about a predefined error measure. This approach disregards the fact that a recommender system exists in an environment where there are a number of factors that the system needs to satisfy, some of these factors are dynamic and can only be tackled over time.

Thus, this thesis intends to study recommender systems from a *goal-oriented* point of view, where we define the recommendation goals, their associated measures and build the system accordingly. We first start with the argument that a single fixed measure, which is used to evaluate the system's performance, might not be able to capture the multidimensional quality of a recommender system. Different contexts require different performance measures. We propose a unified error minimisation framework that flexibly covers various (directional) risk preferences. We then extend this by simultaneously optimising multiple goals, i.e., not only considering the predicted preference scores (e.g. ratings) but also dealing with additional operational or resource related requirements such as the availability, profitability or usefulness of a recommended item. We demonstrate multiple objectives through another example where a number of requirements, namely, diversity, novelty and serendipity are optimised simultaneously. At the end of the thesis, we deal with time-dependent goals. To achieve complex goals such as keeping the recommender model up-to-date over time, we consider a number of external requirements. Generally, these requirements arise from the physical nature of the system, such as available computational resources or available storage space. Modelling such a system over time requires describing the system dynamics as a combination of the underlying recommender model and its users' behaviour. We propose to solve this problem by applying the principles of Modern Control Theory to construct and maintain a stable and robust recommender system for dynamically evolving environments. The conducted experiments on real datasets demonstrate that all the proposed approaches are able to cope with multiple objectives in various settings. These approaches offer solutions to a variety of scenarios that recommender systems might face.

Acknowledgements

I always believed in innovation. For me research provides the opportunity to better understand the world around me and contribute this new understanding back to society. I really enjoyed this period of my life, as it helped me to develop a better way of approaching and solving problems, that I will extensively use in the future for my personal quest of knowledge. Along the way, many people helped and inspired me; they helped many different ways. Making me see problems from different perspectives is a really valuable help, but as simple as pointing out the obvious such as simple things that are taken for granted is also very useful.

First of all, I would like to thank to my advisor, Dr. Jun Wang, whose dedication and enthusiasm for research was the primary source of my motivation. His criticism and discipline were needed in times when things were left for the last minute. I would also like to thank for British Telecom for sponsoring this research and making available some of their data and invaluable insight to shape the direction of my studies. Especially, my industrial supervisors Ian Kegel and Dr. Paul Marrow, who were always helpful and provided me with practical insight that really helped to balance between the two worlds of academia and industry. A "second" opinion that my second advisor, Dr. Licia Capra, provided helped to understand the problems I was working on from a different perspective, I am very grateful for her inspiring feedback. I would also like to thank my examiners, Dr. Cecilia Mascolo and Dr. Emine Yilmaz, who went thought my thesis down to the last little detail and provided insightful and constructive feedback.

I was lucky to be surrounded with a great crowd of intelligent and inspiring people at UCL. Thanks to Neal Lathia for all the inspiration on research and some great feedback on the papers we worked together. Passion for research must really be attributed to Daniele Quercia, who helped a lot to regain some of the enthusiasm that I lost along the way. Thanks to Jagadeesh Gorla for all the personal and academic support (plus being part of the winning team in fussball), and all the people on the seventh floor, especially Hersh Astana, Vasilis Giotsas and Dominik Beste. Thanks to Martin Parsley for the invaluable tea-breaks and Friday pubs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Collaborative Filtering (CF) algorithms have become the mainstream approach to building recommender systems [AT05]. The popularity of CF stems from its ability to explore preference correlations between users and items; it uses a wide range of statistical approaches to identify items (movies, music, books, etc.) of interest to the user based on their historical preferences. In the research domain, this problem is often formalised as a prediction problem: predicting unknown user ratings based on a set of observed preferences [BK07, HKV08]. However, as recommender systems grow in popularity, a notable divergence emerges between research practices and the reality of deployed systems.

In practice, there are certain expectations on what constitutes to a good recommender system. For example, as a user, one would expect their ideal recommender system to match what they would watch, buy or listen given that they had enough time to go thought all the items in the catalogue and select the suitable ones. In addition, one would expect the system to do the discovery for them and recommend something unexpected that they would never find, yet still appealing. By contrast, as an organisation, one would require a recommender system to generate more revenue, by expecting that it would primary satisfy user needs as well as additional requirements that are not directly connected to user satisfaction, such as operating cost and increased revenue.

When CF algorithms are designed, they are evaluated in a relatively *static* context, mainly concerned with a predefined error measure. This approach disregards the fact that a recommender system exists in an environment where there are a number of requirements that the system needs to satisfy; some of these requirements are dynamic and can only be tackled over time. In the literature [RRS11], recommender systems are often modelled as a snapshot at certain points in time, which ignores dynamic aspects of the system, such as how user behaviour affects recommendation quality and performance.

1.2 Approach

This thesis takes a different approach. We study recommender systems from a goal-oriented point of view. This is essentially a top-down approach where we model recommender systems as a means to satisfy certain objectives. These objectives can be grouped into two distinct yet interconnected classes; system and user perspectives.

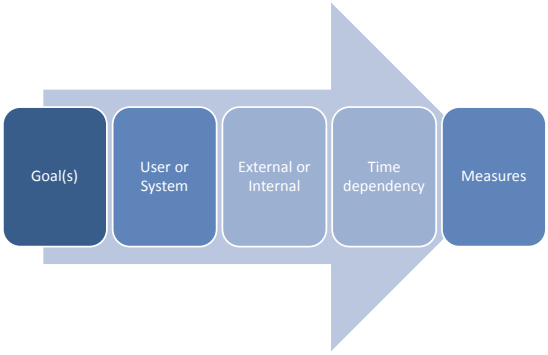


Figure 1.1: A Goal Driven Approach to Building Recommender Systems

- We define the system point of view as a collection of criteria that consider system related objectives as priorities; this includes factors such as maximising the profit margin on the recommended items, reducing cost on delivering content, etc.

- The user centred approach aims to improve user satisfaction, which has many facets depending on how user satisfaction is defined. For instance, generic user satisfaction can be understood as a need to receive quality recommendation, which itself might be user dependent. Quality recommendation includes factors such as satisfying contextual goals (e.g. matching recommendation with user's mood, location, etc), providing recommendation that is novel and diverse (when needed) to the user.

A single measure that is used to evaluate the system's performance might not be able to capture this multidimensional quality of the recommender system. Thus, when this is applied to the design of new algorithms, it is assumed that improving a single measure should not be a goal of the system. Instead, it is more important to investigate how new algorithms affect *various* measures and find the optimal solution that satisfies all the pre-defined goals of such a system. Therefore, first we need to identify various objectives of the recommender systems, justify the need of them, and then design the algorithm accordingly.

We consider the temporal nature of certain user and system goals as the next step in the process. We extend our work to examine how these objectives can be optimised over a period of time. Figure 1.1 summarises the approach:

1. The top-level goal is defined as an informal description of what the algorithm is expected to achieve. This can be decomposed into more specific sub-goals if necessary.

2. The goal can be classified into user-centred or system-centred goals, depending on whether the algorithm focuses on user or system satisfaction or a mixture of these two goals (see Section 1.3.2). It is studied in this work whether focusing on system satisfaction would help improving user satisfaction and vice versa (see Chapter 4 for more discussion).

3. Depending on the type of optimisation used, the goal can be external or internal. If the goal is dealt with outside of the recommender system algorithm (i.e. the algorithm is treated as a black box), it

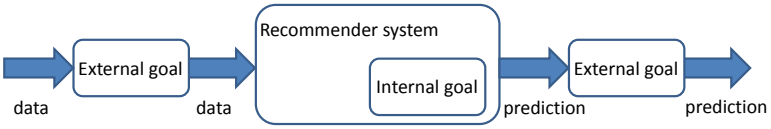


Figure 1.2: Internal and External Goals.

is classified external. If the goal is embedded into the algorithm, it is an internal goal. Figure 1.2 illustrates how internal and external goals can be integrated into a recommender system. It shows that external goals can be used to manipulate the incoming data of the systems (pre-filtering) as well as changing the prediction values (post-filtering), whereas internal goals are used within the recommender algorithm.

4. It is also beneficial to investigate whether the goal is time dependent (i.e. observing a state of the recommender system over a period would change the outcome of the algorithm for a particular goal).

1.3 Underlying Assumptions

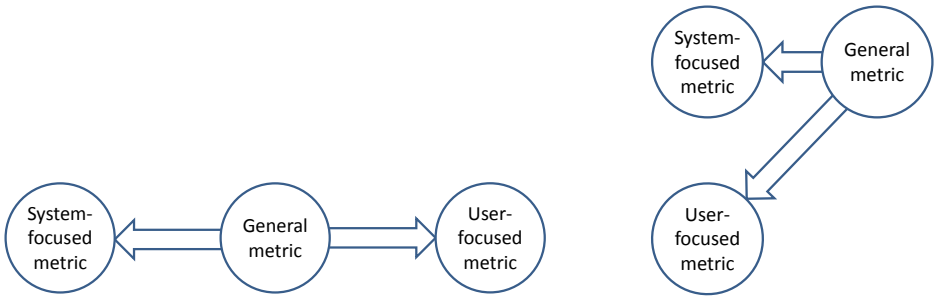
1.3.1 Performance Measures

The use of accuracy as a performance metric is well grounded; previous user studies, such as the one conducted by Swearingen and Sinha [SS01], indicate that recommendation accuracy is a primary factor behind user satisfaction with recommendation systems. This has led to a focus on improving accuracy in recommendation algorithms; state-of-the-art systems score very high on these performance metrics [HKV08]. However, this is not to say that a single measure accuracy alone guarantees satisfactory recommendations. Therefore, the main assumption we make is that an improvement on a given metric that aims to measure certain aspects of the system results in an improvement in the user or system satisfaction of the *same aspect*. This assumption states that given that a metric has been chosen appropriately, it measures the aspect of the algorithm that is designed to measure and it relates back to the real world by being proportional to user or system satisfaction. For example, if we aim capture the biggest part of generic user satisfaction, we select generic performance measures such as the Root Mean Squared Error (RMSE) for that.

This assumption is critical, since, to some extent this connection is not that straightforward. As [CGN+11] shows, notable difference emerges between perceived quality of the recommender system (measured by user studies) and statistical quality (measured by predefined performance metrics). Such differences are appealing, but they are out of the scope of this work.

1.3.2 User-Centric and System-Centric Performance

By definition, user satisfaction and system satisfaction might be opposite to each other to some extent. In other words, if the algorithm is built to improve user satisfaction it might inevitably hurt system satisfaction and vice versa. For example, if a music content provider wants to increase the profit it makes on recommendations, it would recommend items that have higher profit margin out of the items that are



(a) Opposing System-User Goals (b) Complementing System-User Goals

Figure 1.3: User versus System Perspective. The direction of the arrows illustrates the divergence of user and system related metric from the general metric as well as with relation to each other. Opposing system-user goals represent scenarios where improving user performance measures would hurt system performance measures and vice versa, whereas complementing system-user goals might improve the performance of both types of measures (i.e. measures that defined to capture system and user experience).

still desirable to the user (based on the prediction provided by the algorithm), this would increase the short term profit. However, this might affect user satisfaction in the long term, because users might lose trust in the recommendation service and stop using it. This would lead to our proposition that focusing on one or the other perspective is inevitably a *balancing act* where the direction and the intensity of deviation from a central (and general) error measure is crucial (illustrated in Figure 1.3 (a)). In other cases, the difference might be more subtle. For example, improving the effectiveness of the algorithm to incorporate new feedback faster might improve user satisfaction and system satisfaction at the same time. It would decrease computational complexity (thus it would be cheaper to compute predictions) and it might improve prediction accuracy (given that the algorithm has more information to work with) over time (illustrated in Figure 1.3 (b)). Therefore, it is important to make assumptions on the effects that certain goals have on other aspects of the system, and explore the optimal balance based on the goals and the anticipated outcome.

1.3.3 General Error versus Specialised Errors

Figure 1.3 implicitly states that improving some aspects of the system might hurt the general performance of the algorithm. For example, most of the state-of-the-art collaborative filtering algorithms are built to optimise RMSE, which we defined as the general metric of the algorithm. For example, following the goal-driven approach, we might like to improve diversity to boost user satisfaction. In order to evaluate the approach, we can define a metric that would measure diversity of the provided recommender list. Optimising a model based on a diversity measure might result in a lower RMSE (the general measure), simply because RMSE does not measure diversity. Therefore, we suggest that it is good practice to have a general and a number of specialised metrics to be able to balance between generalised and specialised performance.

Table 1.1: Goal Driven Optimisation (Static Perspective) (Chapter 3 and 4)

	User Perspective	System Perspective
Internal goals	Directional Errors (provide quality recommendation) (Chapter 3) [JW10a] Novelty and Serendipity (promote serendipitous items) (Section 4.4) [ZSQJ12]	
External goals	Diversification/Long Tail Items (promote diverse items) (Section 4.3.1) [JW10b]	Stock control (promote items that are in stock) (Section 4.3.1) [JW10b] Optimised content delivery (pre-cache liked items) (Section 4.2)

Table 1.2: Goal Driven Optimisation (Temporal Perspective) (Chapter 5)

	User Perspective	System Perspective
Internal goals	Optimal Control Theory (provide quality recommendation) (Section 6.2.1 - future work)	Optimal Control Theory (maximise profit over time) (Section 6.2.1 - future work)
External goals	Balanced Control Theory (stabilise performance over time) (Chapter 5) [JWL12]	Balanced Control Theory (stabilise resources over time) (Chapter 5) [JWL12]

1.4 Contributions

The contribution of this thesis is to build a number of algorithms following the steps outlined above. The examples aim to show that systematically mapping goals into metrics in order to design collaborative filtering algorithms is more effective and produces more focused algorithms that are aligned with practical objectives. Table 1.1 and 1.2 summarise the aim of the research presented here. These objectives are grouped together based on the criteria described in Section 1.2. As the table shows, we cover most of the goals defined in Figure 1.1.

The following publications and patents form the main part of this thesis:

1. T. Jambor and J. Wang. Goal-driven collaborative filtering - a directional error based approach. In Proceedings of 32nd European Conference on Information Retrieval, 2010.

2. T. Jambor and J. Wang. Optimising multiple objectives in collaborative filtering. In Proceedings of the ACM conference on Recommender systems, ACM, 2010.

3. T. Jambor, J. Wang, and N. Lathia. Using control theory for stable and efficient recommender

systems. In Proceedings of the 21th international conference on World Wide Web. ACM, 2012.

4. Y. C. Zhang, D. O Seaghdha, D. Quercia, and T. Jambor. Auralist: introducing serendipity into music recommendation. In Proceedings of the international conference on Web Search and Data Mining. ACM, 2012.

Pending patents applications are included to illustrate the practical applications of the methods presented in this work:

1. T. Jambor, J. Wang and I. Kegel. Recommender Control System, Apparatus, Method and Related Aspects. European Patent Application 12250046.5, 2012

2. T. Jambor, J. Wang and I. Kegel. Recommender System Set Top Box, Apparatus, Method and Related Aspects. European Patent Application 12250021.1, 2012

1.5 Structure

The rest of the thesis is organised as follows.

- Chapter 2 introduces the main ideas and challenges in this domain, mainly focusing on existing algorithms and their potential extensions. We also investigate related work on how to model the temporal aspects of CF algorithms.

- In Chapter 3, we propose a flexible optimisation framework that can adapt to individual recommendation goals. We introduce a *Directional Error Function* to capture the cost (risk) of each individual prediction. The Directional Error Function aims to improve user satisfaction by focusing on predicting the most important items correctly: this goal is optimised internally in the recommender system algorithm.

- In Chapter 4, we start with an example where the external system objectives are proportional to the performance of the recommender algorithm. Therefore, we aim to discover the relationship between the given objectives and the system performance in order to examine how the system performance (general measures) might affect additional, specialised measures. This is illustrated through an example of system-focused cost based content delivery (in Section 4.2) where the goal is optimised externally. This is followed by Section 4.3 where a general optimisation framework is introduced which not only considers the predicted preference scores (e.g. ratings) but also deals with additional operational or resource related recommendation goals. In essence this optimisation framework solves the problem outside of the recommender system algorithm, therefore these approaches are classed as external goals. In relation to user needs, we illustrate that this optimisation framework can integrate additional user goals such increasing the diversity of the provided recommender list. The system perspective is also considered in this framework, where we show how to optimise external recommendations goals in order to balance between these external factors and accuracy measures. In Section 4.4, we consider novelty, diversity and serendipity as internal user oriented goals to be optimised along with the general performance of the system.

- In Chapter 5, we study the time dimension of the objectives, by modelling the temporal performance of the system and other objectives. This approach is used to gain control of certain objectives over time. Here we illustrate user-focused goals by concentrating on performance, and system-focused goals on resource management.

- In Chapter 6, we conclude this work and present a number of ideas and direction towards which this work can be extended.

Chapter 2

Background

This chapter covers the related work that was used as the basis of the work presented in this thesis. As this work borrows methods and approaches from multiple disciplines, this chapter covers a wide range of relevant topics. Recommender systems form the backbone of the thesis; therefore, this chapter mainly focuses on the state-of-the-art algorithms and research challenges of recommender systems, including the main approaches to rating based predictions and their anticipated problems. We also touch on ideas in relation to recommender systems that include machine learning, probability theory and control theory.

2.1 Data

The amount of data produced across the globe has been increasing exponentially and will continue to grow at an accelerated rate for the foreseeable future. At companies across all industries, servers are overflowing with usage logs, message streams, transaction records, sensor data, business operations records and mobile device data. In addition, user interaction across social networks such as Twitter and Facebook produce a large amount of unstructured data. It is a challenge in the industry to make sense of the data and use it to its full extent. Recommender systems are at the forefront of this trend as the predictive ability that the algorithms offer can generate useful insights of the data. However, depending on the way the data is generated and collected, the algorithms have to make certain assumptions to interpret and predict from the data. In this chapter, we give a short overview of the literature on how the data influences the choice of the algorithm as well as the way the algorithm is evaluated. We discuss the relationship between the way users express preferences and the interpretation of these preferences. Furthermore, we introduce approaches on how additional data sources can be used to enrich feedback datasets.

2.1.1 Explicit Rating Datasets

The bulk of the collaborative filtering research is conducted on rating datasets. In essence, it is generated for users' feedback on certain information items. An early example of this is the GroupLens project [KMM+97, RIS+94], where users were invited to rate news articles on a rating scale of one to five. Later, this approach was used with a wide variety of items (e.g. movies, music, products). The main characteristic of these datasets is that they contain an explicitly expressed preference of a particular user towards a particular item. Obviously, this preference can only be expressed after the user consumed the

item (e.g. watched a movie). These datasets normally contain timestamps which indicates when the event happened, however this might not correlate with the actual consumption of the item, as the time the item was rated can be different from the actual consumption time depending on the way the data was collected. Depending how the data is collected, some datasets might be skewed towards negative or positive ratings. For example, customers might have more incentives for complaining about failed goods, rather than give good feedback on high quality product, conversely, users might remember only the good movies they watched if they have to bulk rate a list of movies when they start using a recommendation service (a registration procedure that is widely used to gather information about new users). Either way, the distribution of the ratings on individual level, as well as, across the whole population should be taken into account as it would effect the performance of the algorithms [AZ12].

2.1.2 Implicit Feedback Datasets

In many occasions, explicit feedback is not available, instead an implicit indication of preference can be collected. This kind of feedback is widely available, as this does not require the user to make additional effort and rate the item, instead a number of different user behaviour can be interpreted as a (mostly weak) indication of preference [OK98], this can include purchase history, browsing history, search patterns, etc. This type of data reflects a better representation of time, as it is likely that items are consumed after purchased. However, in terms of other signals, implicit datasets contain less information than rating dataset. For instance, the indication of preference only contains the initial interest of the user towards the items, as we only know that the user was interested in the item at the time they purchased or clicked it, but there is no information on the opinion of the user after the item was consumed. In addition, implicit feedback datasets contain consumed items only of a particular service provider or seller, so that in the dataset only positive preference feedbacks are available, which assumes that the non-observed points are a mixture of negative feedbacks and missing values. In the case of rating datasets, the user profile is more complete and it is reasonable to assume that most of the items which are not rated are likely to be unknown to the user. As missing values are not always negative ones, a better representation of the problem is to store values in terms of their relationship to other items as introduced in [RFGST09]. Here, items that have received feedbacks are certainly better than items which have not received any feedback, however, between items that do not have feedbacks (or have feedbacks), the relationship is unknown.

2.1.3 Explicit and Implicit Feedback

There is a trade-off between quantity and quality when comparing explicit and implicit feedbacks. Explicit feedback is more precise in terms capturing preferences while implicit feedback data is less abundant in quality, but it is widely available in vast quantities as it is more practical to capture. In [JSK10], the characteristics of implicit and explicit datasets were explored in the context of music listening, it is discussed that a user might only provide positive explicit feedback on a music item after extensive listening, which suggests that items that receive feedback from the user have made some sort of impact (either positive or negative) on the user. However, in [PA11], it is shown that there is a strong correlation between implicit and explicit feedbacks in music listening; it is found that the amount of time the user listens to a music item would increase the explicit rating of the item. It is also shown that the time

elapsed since the user interacted with music item, have a significant effect on the rating. However, other global effects such as popularity do not influence the explicit rating.

2.1.4 Noise in Data

It is frequently assumed that user feedback represents the true cognitive state of the user; hence, the feedback is used as the ground truth in evaluating algorithms. This assumption was challenged in [APO09], where the inconsistency in users feedback strategy was examined. It was found that a significant part of the error in rating-based recommender system algorithms is due to the noise of this inconsistency. In addition, it was demonstrated in [APTO09] that asking users to re-rate items they had previously rated could reduce errors in prediction by as much as 14%. Items that were likely to be noisy are selected to be re-rated in order to highlight items that would increase accuracy for the user.

2.1.5 Content Information

In addition to feedback from the user, additional information can be used to enrich the dataset, content information can be extracted from a wide range of choices. When information has no structure (e.g. text), an additional pre-processing is required to extract structured information from the data, this part of the process usually uses techniques from Information Retrieval [vR79, BYRN99]. This information is used to build a profile of each item in the dataset, an item profile includes certain attributes and characteristic on an item [PB07], for example in the case of music items it can contain the genre, artist and album of the track. The user profile is constructed to represent the relationship with the items from sources described above (i.e. implicit or explicit datasets), or can be extracted from unstructured information (e.g. text comments) to enhance the decision making process [RIS+94]. Content information might be useful to enhance exciting systems but it is argued in [PT09] that content information alone is of little value when it comes to predicting movie ratings compared to the improvement observed when adding additional feedback from the user.

2.1.6 Social Information

Another emerging type of data that is widely available online is users' social information due to the prevalence of Web 2.0 social networking sites. Essentially, this social information contains activities that can be extracted from user profiles of social network sites; this might include status updates, tweets, Facebook likes, etc. This information can be used to build user profiles selecting information that might be useful to predict users' taste. In addition, social network sites can provide a graph of related users, which is also valuable to infer user-user relationships to understand the similarities between users' taste and social trust [BKM07] that influences one's taste. In addition, users' membership in certain groups [YCZ11] can also be valuable to infer characteristic of the user associated with the group they belong. This based on the assumption that users use their social network to obtain information (hence, connected users might have similar taste) and use their trust network to filter this information which represents the degree in their similarities [WBS08]. However, the extent that this holds is highly dependent on the actual datasets, for example, social ties are much weaker on Twitter than on Facebook [MTP10]. Therefore, the connection between users, the flow of information and trust should be interpreted differently.

2.2 Algorithms

Recommender system algorithms grew out of the need to reduce the inevitable information overload of the information age and provide personal content based on preferences that the user expressed towards certain items in the past. Known recommender systems provide either content-based recommendations in which a user is recommended items similar to the ones a user has preferred in the past, or collaborative filtering recommendations in which a user is recommended items that people with similar tastes and preferences have liked in the past, or adopt a hybrid-approach which combines collaborative filtering and content-based methods. Context-aware algorithms provide an additional dimension to the basic recommendation approach by considering factors such as time, mood or location.

Recommender systems are widely used in e-commerce where they are proven to boost sales by helping the user to discover new products; recommender systems are increasingly used to improve the quality of service and user satisfaction [LR04]. In addition, recommender systems might enable users to discover items that are hard to find, thus they can increase the exposure of more diverse items. There are services that are marketed and built on personalised content (e.g. Netflix), this trend can be observed across the web, as many content providers personalise their content as part of the service.

2.2.1 Collaborative Filtering

One of the most popular techniques that are used for recommender systems is collaborative filtering [HKBR99, SFHS07]. The term was first coined in [GNOT92]. Its aim was to develop an automatic filtering system for electronic mail, called *Tapestry*. The basic idea of collaborative filtering is that users who tend to have similar preferences in the past are likely to have similar preferences in the future, and the more similar they are, the more likely they would agree with each other. In other words, collaborative filtering algorithms assume that users who like the same items would have the same interest and this shared interest in relatively constant so that people-to-people correlations can be extracted from the data [SKR01]. This assumption enables the system to build coherent communities based on shared taste and behaviour. These feedback profiling methods use the opinions of users to help individuals to identify content of interest from a large set of choices, which otherwise would be problematic to find. They return a set of recommended items that are assumed not to be known to the user before, but match to their taste. This match is expressed by a rating value that represents the extent that the target user would like the target item. Most of the studies in collaborative filtering concentrate on predicting these ratings. The task is to make these predictions as accurate as possible. These collaborative filtering techniques can be divided into three main categories; model based, memory based approaches and matrix factorisation methods. Within matrix factorisation methods, a special case that is concerned with implicit feedback models is also discussed.

Memory-Based Approaches

Memory-based approaches are the most widely adopted ones. In these approaches, all user ratings are indexed and stored into memory. In the rating prediction phase, similar users or (and) items are sorted based on the observed ratings. Relying on the ratings of these similar users or (and) items, a prediction of an item rating for a test user can be generated. Examples of memory-based collaborative filtering include

user-based methods [HKBR99], item-based methods [DK04] and combined methods [WdVR06]. User-based filtering techniques [WdVR06] concentrate on finding similar users to the active user. First, a set of nearest neighbours of the target user are computed. This is performed by computing correlations or similarities between user records and the active user. The preference prediction is calculated by the weighted-averaging of ratings from similar users. The major problem with this approach is the bottleneck problem, the complexity of the system increases as the number of users grows which could reach an unmanageable number of connections to compute in large commercial systems. Item-based collaborative filtering [SKKR01] aims to find items which are similar to a particular user's preferences. This algorithm attempts to find similar items that are co-rated (or visited) by different users similarly. This is done by performing similarity calculation between items. Thus, item-based algorithms avoid the bottleneck in user-user computations by first considering the relationships among items. Moreover, it is argued that item similarity is more constant than user similarly; therefore, this information can be pre-computed and updated periodically, which makes this approach more suitable for real-time recommendation. This makes item-based approaches attractive for real-world solutions [LSY03, AT05]. It has been shown [AT05] that hybrid recommender systems are more efficient than systems based purely on item-based or user-based collaborative filtering. Furthermore, data sparsity is considered one of the main reasons why these systems perform poorly. A hybrid system has been proposed by [WdVR06] where user-based and item-based collaborative filtering approaches are unified using probabilistic fusion framework, it is showed that this system performs better even with sparse data.

Model-Based Approaches

Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. They use the training samples to generate an abstraction which is then used to predict ratings for items that are unknown to the user. [SMH07] In this regard, many probabilistic models have been proposed. For example, a method called personality diagnosis [PHLG00] computes the probability that users belong to a particular personality type, as well as, the probability that certain personality types like new items. The method treats each user as a separate cluster and assumes that a Gaussian noise is applied to all ratings. Other probabilistic approaches [BHK98] attempted to model item correlation using a Bayesian Network model, in which the conditional probabilities between items are maintained. Some researchers have tried mixture models, explicitly assuming some hidden variables embedded in the rating data. Examples include the Bayesian-clustering and vector-similarity methods [BHK98]. As always, these methods require some assumptions about the underlying data structures and the resulting models solve the data sparsity problem to a certain extent.

Latent models [Can02], similar to matrix factorisation methods discussed below, aim to explain observed preferences with a smaller number of preference patterns, which are assumed to be able to generalise the data generation process. Early methods include neural networks [SMH07], probabilistic Latent Semantic Models [Hof01, Hof04]. Topic models are also used to predict ratings, for example Chen et al [CCL$^+$09] presented a comparison between Latent District Allocation (LDA) and association rule mining for the purpose of community recommendation. Another approach introduced in [HCRC11]

uses LDA to extract latent factors from sparse data, so that each user and item is described by a set of topics which are used as a basis of prediction.

Matrix Factorisation Methods

Alternatively, collaborative filtering can be considered as a matrix factorisation problem; it has emerged as the clear favourite in the Netflix competition [Kor08, KBV09, RS05]. The main reason of their success may be due to the fact that the objective function can be formulated in a way to directly optimise certain performance measures (e.g. RMSE). Moreover, these approaches can incorporate a number of methods into the learning that makes matrix factorisation approaches applicable in a wide range of situations. In general, the approach aims to characterise both items and users by vectors of factors inferred from item-rating patterns. In other words, matrix factorisation methods aim to discover latent features that explain observed ratings through dimensionality reduction. Strictly speaking, matrix factorisation methods are categorised as model-based, because they rely on a model learnt from the data, but some of the method presented in this section also has memory-based characteristics, as they incorporate memory stored ratings into prediction.

One of the most popular matrix factorisation algorithms applied to collaborative filtering uses Simon Funk's approach [Fun06]. The objective function of the Singular Value Decomposition (SVD) algorithm as defined in [KBV09]

$$\underset{q,d}{\operatorname{argmin}} \sum_{u,i} (r_{u,i} - q_i^T d_u)^2 + \lambda(||q_i||^2 + ||d_u||^2) \tag{2.1}$$

where $r_{u,i}$ represents the observed rating, d_u and q_i are f dimensional vectors ($d_u, q_i \in \mathbb{R}^f$). These two vectors represent the user and the item latent features respectively. f is a parameter of the model which defines the extent of the dimensionality reduction applied to the observed user-item matrix.

The most popular solution to SVD [WKS08] uses gradient descent to infer the latent features for each user ($d_{u,f}$) and item ($q_{i,f}$). This can be done by taking the partial derivative of the squared distances between the known entries and their predictions (e^2) with respect to each $d_{u,f}$ and $q_{i,f}$. So that for each feature we compute

$$\begin{aligned} d'_{u,f} &= d_{u,f} + \lambda * \frac{\partial e^2}{\partial d_{u,f}} \\ q'_{i,f} &= q_{i,f} + \lambda * \frac{\partial e^2}{\partial q_{i,f}} \end{aligned} \tag{2.2}$$

After simplification and additional regularisation the final equations are as follows

$$\begin{aligned} d'_{u,f} &= d_{u,f} + \lambda * (e * q_{i,f} - \gamma * d_{u,f}) \\ q'_{i,f} &= q_{i,f} + \lambda * (e * d_{u,f} - \gamma * q_{i,f}) \end{aligned} \tag{2.3}$$

The approximation is found such that it minimises the sum of the squared distances between the known entries and their predictions. The algorithm produces two matrices, user and item matrix, where each row vector represents the user and the item in the latent space. After the matrices are learnt the prediction is calculated by the dot product of the target user's and target item's feature vectors so that

$$\hat{r}_{ui} = q_i^T d_u \tag{2.4}$$

There are a number of variations of this approach, for instance, a faster implementation was presented in [Pat07], where instead of learning the user and the item vectors, the user is modelled as a function of the items the user rated. This approach reduced the computational complexity from $O(NF + MF)$, which is the standard SVD, to $O(MF)$, where N is the number of users, M is the number of items and F is the number of features. Another matrix factorisation method was presented in [ZWSP08], essentially this approach aims solve the same objective function introduced in Equation 2.1, instead of using the gradient descendent solution, it solves the problem using Alternating Least Squares [LH74, GZ79] approach. This approach alternates between fixing the user and the item matrix and solving it for each row vector in the non-fixed matrix given the fixed matrix and the preference vector (which lists all the observed ratings). The advantage of this approach is that all the user/item vectors can be computed independently, so that the computation can be parallelised which makes it ideal to use it on larger datasets. Matrix factorisation was also combined with neighbourhood models in [Kor08] resulting in a SVD++ algorithm that allows neighbourhood and latent factor models to enrich each other. In addition this method allows to exploit both implicit and explicit feedback by directly incorporating both types of feedback (if available) into the objective function. An interesting insight of this approach is that using the same data as both implicit and explicit (i.e. binarising the explicit data) might add an additional aspect to the optimisation, that is expressed by the effect of rating an item (implicit) rather than the actual rating (explicit). In effect, this different aspect of the data might help to improve the overall performance of the system. Another similar approach was presented in [TPNT07] combines the latent factors based on user and item neighbourhoods computed using standard similarity measures where the user neighbourhood was defined between the latent feature vectors rather than the actual user profile. This approach provides a scalable solution as it separates the matrix factorisation phrase from the similarity computation phrase, enabling to compute the similarity offline. Matrix factorisation methods were extended in [Ren10] to combine the advantages of polynomial regression models and factorisation models. This approach has the advantage to take real valued feature vectors as inputs and produce real valued predictions taking advantage of polynomial regression models such as Support Vector Machines (SVM) [STC00] to incorporate various features into the prediction. As opposed to SVMs, factorisation machines model all interactions between variables by the usage of factorisation parameters which helps to reduce the sparsity problem that is usual in the domain of recommender systems. The advantage of this approach is that it can take different types of data, in the form of feature vectors, and it is not restricted to certain kinds of inputs like the approaches introduce above.

Implicit Feedback Models

In [HKV08], implicit datasets were considered for matrix factorisation. As mentioned above, implicit feedback represents different types of data, with a different set of assumptions behind it. One of the drawbacks of implicit datasets that they do not contain negative examples of preference. To overcome this, the model considers missing ratings as an implication of dislike, since the model needs negative training samples to learn what the user might dislike. This enables the model to learn negative preferences on the missing ratings and positive preferences on the observed ratings. Moreover, another variable was

introduced in this model which was used inject confidence associated with the observed and unobserved ratings, so that each data point can be assigned with a confidence value to incorporate additional information into the model (e.g. view counts). In this way, it is possible to smooth the noise that is introduced by considering missing ratings as dislikes by reducing the confidence on the unobserved ratings.

2.2.2 Content-Based Filtering

Strictly speaking, collaborative filtering techniques produce recommendations based on, and only based on, knowledge of users' relationships to items. These techniques require no knowledge of the properties of the items themselves. Another category of recommender systems takes into account domain specific knowledge to generate prediction. In other words, content-based recommendation algorithms try to recommend items similar to those the user has liked in the past, whereas in collaborative recommendation one identifies users whose taste is similar to those of the target user and recommends items accordingly. Content-based [PB07] recommender systems deal with domain specific knowledge in order to build on semi-structured information (e.g. the genre of the movie, the location of the user) and infer relationships between these structures. The best way to integrate the extra pieces of information into the system is to consider recommendation as a learning problem as describes in [BHC98]. For example, the function can take a movie-item pair and classifies it into two categories (liked/disliked), or it can be considered as a regression problem where the system learns a set of parameters for each user with respect to the items in the user's profile (using linear regression as the simplest case). Given the learnt parameters of the user and the features of the item, a prediction score can be produced for each item that is unknown to the user.

Attempts were made to combine content-based filtering and collaborative filtering, the simplest approach is described in [BS97] where user profiles were created based on content analysis of the items and these profiles were used as an input to collaborative filtering algorithms. Another attempt was presented in [PPL01] to combine the two approaches within a generative probabilistic model. In this case a three-way co-occurrence model was used where it was assumed that the user interest could be represented as a set of latent topics, which were generated from user and item content information. A probabilistic approach to combine content-based recommendation with collaborative filtering information was presented in [SHG09]. It models users and items as feature vectors, which is similar to the approach used for SVD, but the way these feature vectors are learned is different. The process is modelled as a combination of Expectation Propagation [Min01] and Variational Message Passing [WB06], where the model parameters are assumed to be independent Gaussian distributions.

2.2.3 Context-Aware Algorithms

Context-aware approaches [Che05] aim to take into account that users might prefer different sorts of treatment in different situations. For example given the time of day, the mood of the user or the location, the system might tailor the approach adapted to the users' needs in the given context. These different goals have been recognised by researchers and practitioners in many disciplines, including e-commerce personalisation, information retrieval, ubiquitous and mobile computing, data mining, marketing, and management. In recommender systems, contextual information might help to improve the quality of the recommendation if the user behaviour tends to change in a different context (which is the case most

of the time). For example, users might prefer to listen to different types of music depending on the time of the day, the weather and the location of the user. If contextual information is available, the recommendation list can be modified accordingly. In order to achieve this, two different approaches have been developed. Contextual information can be used to filter the recommendation list using current context information and specify current user interest in order to search the most appropriate content. For example, location-aware social-event recommendation can be obtained by providing a list restricted by the location of the user [QLC+10], other approaches for tourist [VSPK04] and location based [PHC07] scenarios have been proposed. The second approach attempts to model and learn user preferences by observing the interactions of the users with the systems or by obtaining preference feedback from the user on various previously recommended items. This enables the system to model context-sensitive preferences in order to provide recommendation. Factorisation Machines (FMs), can also be applied to context-aware rating predictions as this model can generate feature vectors for context of categorical, set categorical and real-valued domains, this makes FMs well suited for context-aware recommendation [RGFST11].

2.2.4 Different Types of Data

Most recommender system algorithms are based on models of user interests that are built on the assumption that the data comes from either explicit feedback (e.g. ratings, votes) or implicit feedback (e.g. clicks, purchases). As mentioned above, explicit feedbacks are more precise but more difficult to collect from users as the rating mechanisms has to be built into the user interface where the user is prompted to give feedback on items presented to them. Implicit feedbacks are much easier to collect though less accurate in reflecting user preferences [LXZY10].

Implicit feedback can be mined from viewing logs and purchase data, where a number of assumptions has to be made on how to "translate" observed behaviour captured by the logs into preference scores. For example, server logs collected by the content providers are very often used to mine user behaviour. A number of rule based approaches are in place that compute preference score per user basis from server logs. For example, in [GE00], users' favourite TV programmes are inferred from server logs as a combination of duration and time coupled with a number rules as to how to deal with channel hopping behaviour, fast forwarding, pausing live TV content, etc. This approach can be refined for different types of datasets taking into account what sort of information is captured in the logs and how it might represent user behaviour.

If both types of user feedback are present, it has been shown that combining different types of feedback can improve accuracy [LXZY10]. In [Kor11] a number of scenarios are presented to combine these two types of data. If explicit feedback is captured implicit feedback can be also inferred from data just by processing the corresponding server logs (as described above) and, as the simplest case, injecting implicit ratings as simple weights into the model.

2.3 Evaluation

Traditionally, recommendation quality is measured using one of a number of accuracy metrics, which assess how well the predictive output of a system matches a proportion of known, withheld items for each user. Examples of error based accuracy metrics include Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) which were widely used in Data Science competitions such as the Netflix Prize [BBKV09]. This approach can quantify how the algorithm performs across all available test samples, which is a good indication of performance is that aim of the recommendation is to produce good prediction across all items. On the other hand, researchers also argued that understanding collaborative filtering as a rating prediction problem has some significant drawbacks. In many practical scenarios, such as the Amazon's book recommender system [LSY03], a better view of the task is of generating a top-N list of items that the user is most likely to like [WRdVR08, Kar01, Har03, SLH10]. The simple reason behind it is that users only check the correctness of the prediction at the top positions of the recommended list; they are not interested in items that would be predicted uninteresting to them, therefore low on the ranking list.

Ranking lists are considered a more accessible approach to present a list of items to the user for browsing as it enables the user to produce a number of different forms of feedback actions [Ama12]. For example, as reported in [Ama12], after Netflix launched an instant streaming service in 2007, the way of using recommendation has changed (one reason why the company has not adopted the winning algorithms). Users had more freedom to explore and sample a few videos before choosing one, while the system could gather more viewing statistics and feedbacks such as whether a video was watched fully or only partially and make recommendation decisions based on them.

Ranking based approaches are evaluated by the use of Information Retrieval (IR) measures, such as precision, recall, normalised Discounted Cumulative Gain (nDCG) [JK02], Mean Average Precision (MAP), etc. Each measure empathises different qualities of the ranked list and able to measure different aspects of the performance [HKTR04]. Despite the fact that the aim of evaluation is to propose subjective and easy repeatable results, the multiple ways evaluation can be set up, especially if it is evaluated using rank based measures, can significantly affect the results. This effect was investigated in [BCC11], where different experimental setups and their corresponding results were compared. The authors examined whether the assumption on missing item in the test set, where there is no ground truth available, and missing items in the training set, where there is no training data available would affect the evaluation results. It is found that there is a big difference in the performance of the system depending on which evaluation method is employed. This suggests that using error based measures might produce a more standard score that is easier to compare across different algorithms without understanding the small details of the evaluation methods employed. On the other hand, error based metrics might not measure useful qualities of the system, since it uniformly measures items across the samples, which might not reflect the way users value recommendations; for example more value might be gained from predicting the top items correctly (for more discussion see Chapter 3).

2.4 Challenges and Extensions

2.4.1 Cold Start Problem

One of the key challenge in collaborative filtering is to effectively predict preferences for new users, a problem generally referred to as the user cold start problem [Ahn08, SPUP02]. Content features of users (e.g. age, gender) and items (e.g. genre) can be utilised to solve the problem and reducing the amount of data required to make accurate predictions [GM08, PC09]. For example, in [GDF+10] latent features were constructed from content information of users and items. These features were plugged into a matrix factorisation algorithm to provide recommendation for either new users or new items. If the user's demographic data is not available, a simplest method to tackle the problem is to interview the user to provide additional information (e.g. favourite genre) or ask the user to rate a set of items in order to provide enough data for the recommendation engine. Essentially, the advances made in this domain all built on the "interview" idea and the goal is to come up with a set of items for a new user to rate. They can be selected based on measures such as popularity, entropy and coverage [RAC+02, RKR08], or on a concrete optimisation goal that is in line with the main algorithm [GKL10]. In [ZYZ11], a functional matrix factorisation model was proposed. The algorithm integrates matrix factorisation for collaborative filtering and decision-tree based interview into one unified framework by modelling latent profiles as a function of user responses to the interview questions and selecting the best item by solving the objective function at each node. A similar method was presented in [GKL11] where the decision tree and matrix factorisation are learnt in two separate steps. Active learning is also deployed in order to identify the most informative set of training examples (items) through minimum interactions with the users with respect to some selection criteria, such as the expected value of information [HY08b, JS04, ME95]. All these methods aim to discover more about the user by going through an interview process and then using the result of the interview (in this case the user profile) for prediction. Asking users specifically to give ratings is, however, still time-consuming and sometimes a hurdle for a user to overcome even if the effort has been kept minimal.

2.4.2 Temporal Dynamics

One of the primary aims of recommender systems is to capture the dynamics of user preferences in order to make usable recommendation for unknown items in the future. This change in taste is a reaction to the constantly evolving culture that is expressed through music and arts. In the simplest case, this change in taste can be modelled by frequently updating the recommender system to learn on the latest feedback of the users. However, temporal dynamics can be explicitly modelled and integrated into the predictive model. The temporal dynamics of any recommender system is the combination of two intersecting components. On the one hand, the preferences of the system's users may be subject to change, as they are affected by seasonal trends or discover new content. Evolution of preferences can be modelled as decay, so that, in the longer term, parts of users' profiles can expire [TH08]. Koren [Kor10] also examined this problem, distinguishing between the transient and long term patterns of user rating behaviours, so that only the relevant components of rating data can be taken into account when predicting preferences. In this way, recommender systems can retain separate models of the *core* and *temporary* taste of any user.

Then, the core taste could be used in a longer period of time while the temporary taste can expire if the user becomes interested in items that do not match her previous temporary taste. In particular, we note that the above studies focus on user preference shift as a means toward improving prediction accuracy.

The flip side of the recommendation temporal dynamics relates to any temporal changes subsumed by the recommender *system* itself as time progresses and the system is updated. Current recommender systems address the fact that users continue to rate items over time by iteratively re-training their preference models. Modelling and evaluating the performance of recommender systems from the perspective of accuracy, diversity, and robustness over time was addressed in [Lat10, LHCA10]. The central tenet here is that the regular, iterative update of recommender systems can be both simulated and levied to improve various facets of the recommendations that users receive, which include temporal diversity.

A middle ground between purely static and dynamic approaches to CF is the use of online learning algorithms. For example, the authors in [RST08] proposed online regularised factorisation models that did not require time-consuming batch-training. An active learning approach was proposed to identify the most informative set of training examples through minimum interactions with the target user [HY08a]. The purpose here was to quickly address the problem of *cold-start* items and users.

2.4.3 Controlled Dynamic Systems

Control theory stems from research relating to guiding the behaviour of dynamical systems; notable examples include dealing with network traffic [FLTW89]. At the broadest level, control systems require three components: (a) a *system* which produces an output in a dynamic context (e.g., a recommender system), (b) a *measure* of the quality of the output (such as the *RMSE* on the recommendations) and (c) a *mechanism* to adapt or tune the system's parameters according to the measured quality of the output. The control system binds these three components together into a closed loop; in doing so, it provides a means for feedback relating to the system's performance to be input into the system itself.

Researchers have found that control theory can accurately model the behaviour of software systems and it provides a set of analytical tools that can predict, with high accuracy, the behaviour of a real system. In [ZC11], control theory was used to monitor system growth and control whether the system needs to perform an update. Based on the amount of new data entering the system, it calculated the loss in accuracy if the system was not updated, and decided whether the benefit of preforming an update would outweigh the computational cost associated with the update, based on a predefined tolerable performance loss. This approach assumes that the rate of data growth is a linear function of the performance loss over time. The disadvantage of this is that the system can only rely on a simplified relationship between the data flowing into the system and the performance (which might not be linear and also include additional factors) and cannot directly control the disturbance introduced by the dynamics of the recommender system as the actual output of the system is not known to the controller. In [PGH+02], researchers presented a workflow, based on classical control theory, to design a feedback control system for an email server to maintain a reference queue length. They used a statistical model to estimate system dynamics, this was followed by a controller design phase to analyse the system response. They found that the analysis predicted to behaviour of the real system with high accuracy. Control theory has also been

applied to reputation systems, where the aim was to identify and reduce the impact of malicious peers on recommendation. In [MWZ+06], a feedback controller was used to adjust users' recommendation trust based on the accuracy of their rating.

2.4.4 Popularity and Long Tail Items

In economy, Anderson [And08] introduced the concept of long tail selling pattern, it shows that retailers sell relatively large quantities from a small number of popular items and sell a large number of items which are not that popular in smaller quantities. Given that every individual has a unique taste; it is highly unlikely that this taste can be satisfied through mass media, which means popular items might only meet a small part of the populations' taste. The reason why popular items get popular despite that can be explained by the nature of mass media, people tend to be exposed to multimedia items that are widely available and usually heavily advertised. Therefore, mass media pulls together people who would not normally find themselves together. However, this trend is likely to change. The cost of delivering multimedia content is getting smaller which will enable businesses to target each customer individually and satisfy their personal taste. This trend would likely result in an increase in selling larger quantities of currently unpopular items. As mentioned earlier recommender systems are to close the gap between the vast range of choices and the limited time of the user to explore this content. Ideally, this would entail that recommender systems recommend items that are interesting to the user, which satisfies as many aspect of their individual taste as possible. This is an obvious assumption, but many argue [LHCA10, PT08, SFHS07] that current recommender systems do not fulfil this task. The reason is that current recommender systems choose the easy way; they recommend items that are easy to get right which is the best strategy to improve general performance. However, this approach usually leaves the user with items that are either known or do not appeal to them. Therefore, it might be preferable from the user point of view to increase the number of items recommended from the long tail (i.e. provide a more diverse choice of items). One of the problem researchers faced in this respect is that fewer rating in the long tail might result in a higher risk of getting the prediction wrong. One approach to overcome this is presented in [PT08], where the item set was divided into head and tail and only the items in the tail were used for a method called Clustered Tail. This method used a number of derived variables (both user and item related) and built a model on each cluster in the tail. However, the disadvantage of this approach is that separating the head and the tail of the item set assumes that they are not interconnected at the highest level. A more sensible approach would be to artificially promote items that are in the long tail, given that they are disadvantaged by items that are more popular.

In the study of Information Retrieval, a wide range of work consider the importance of diversity in ranking. This aims to reduce the risk of returning documents from a single topic, which is problematic in cases where the query is ambiguous. In other words, diversification is to maximise both the relevance and diversity of the result page, given a query that is ambiguous or underspecified. The basic idea of these methods is to penalize redundancy by lowering an item's rank if it is similar to the items already ranked [CK06]. In [ZGVGA07], diversification is achieved by the usage of absorbing random walks, which turns ranked items into absorbing states in order to prevent redundant items from receiving a

high rank. In [THW+11], diversification was framed as an optimisation problem where the aim was to maximise an objective function which utilizes the submodular property. Alternatively, diversification can be considered as a means of managing uncertainty and risk in the ranked list [WZ09].

2.4.5 Social Recommendation

The increasing popularity of social media sites (e.g. Facebook, Twitter) paved a way for to enrich traditional collaborative filtering algorithms with social data which can be used to improve product recommendation as well as reduce the increasing information overload on social media sites. In this overview, we only concentrate on the recommender system side of the topic.

Social networks provide useful information of users' social graph including their trust network which might influence one's taste. In [JE09, JE10] this network was incorporated into solving cold-start problems and reducing sparsity in the user-item matrix. In [MKL09], the authors interpreted one users final rating decision as the balance between this user's own taste and his/her trusted users taste, and an ensemble probabilistic matrix factorisation method is proposed to implement this intuition. In [MZL+11], another social recommendation approach is proposed by adding the social regularisation term to the matrix factorisation objective function. In this method, the additional social regularisation term ensures that the distance of the latent feature vectors of two friends will become closer if these two friends share similar taste. In [MA07], a trust-aware method for recommender system is proposed. In this work, the collaborative filtering process is replaced by the reputation of users, which is computed by propagating trust. The degrees of trust are calculated to replace the similarity value between two users. The experiments on a large social recommendation dataset show that this work increases the coverage (number of ratings that are predictable) while not decreasing the accuracy (the error of predictions).In [MZL+11], two Social Regularisation methods have been proposed by constraining the matrix factorisation objective function with user social regularisation terms. Different with previous methods, the proposed methods are very general; they not only work with user trust relationships, but also perform well with user social friend relationships. The experimental analysis indicates that the proposed framework outperforms other state-of-the-art methods.

2.5 Summary

This section aimed to provide an overview the main areas of research in recommender systems. As it was illustrated the main body of algorithmic research concentrates on directly improving a pre-defined measure. This might be the direct result of the Netflix competition where the success of the algorithm was measured by RMSE as a result many researches concentrated on improving their algorithms on that measure. Table 2.1 summarises papers reviewed in this section that aimed to improve collaborative filtering algorithms. It shows that most of the papers were concentrating on improving the *general accuracy* in some sense and measuring the performance gain by a uniform squared error (either RMSE or MAE). In addition, many of the papers that were concentrating on specific aspects of recommender systems measured the success of the approach using the same general accuracy measures. This is understandable in a sense that it allows papers to be easily compared and validated, but this approach misses the op-

Table 2.1: Main recommender goals identified in the chapter that aimed to improve prediction accuracy in general or concentrated on specific aspects of the system.

Algorithmic improvements		
Objective	Reference	Performance Measure
KNN	[BK07]	RMSE
SVD	[Pat07]	RMSE
Item based	[SKKR01]	MAE
Combining SVD and KNN	[Kor08]	RMSE, Top-K list
Top-K accuracy	[Kar01]	Hit-Rate
Top-K accuracy	[LY08]	NDCG
Specific recommendation goals		
Objective	Reference	Performance Measure
Diversification	[ZMKL05]	Precision, Recall
Long tail items	[PT08]	RMSE
Content information	[SHG09]	MAE
Temporal dynamics	[LHC09]	RMSE
Cold start problem	[Ahn08]	MAE

portunity to validate the results on more problem specific evaluation methods that would help capturing improvements in the direction that it was intended to improve. Therefore, we argue that the success of specific approaches that aim to capture specialised aspects of a recommender system is not necessarily proportional to the improvement of a uniformly measured performance metric. In addition, the aim of improving recommender systems should always be directed towards specific goals as opposed to improving general performance, because recommender systems in general always have to balance between certain recommendation goals (i.e. user or/and system defined goals) that could not be captured by a single (general) measure.

Chapter 3

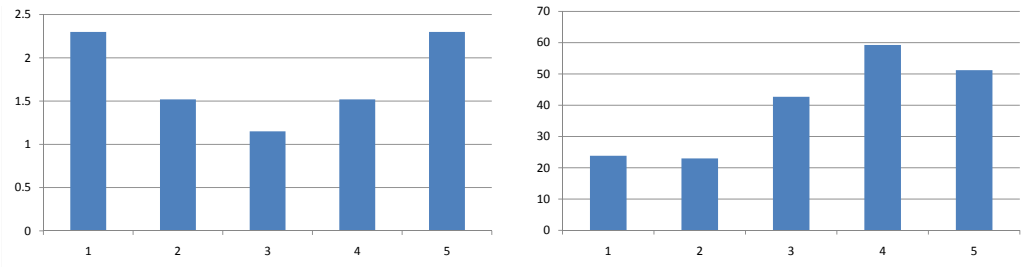
Directional Error Based Approach

In this chapter, we introduce a Directional Error Function to capture the cost (risk) of individual predictions which can be based on specified performance measures at hand. This is the first example of goal driven design where where the goal is identified first and the performance is measured using specialised metrics. In the literature, a common experimental setup in the modelling phase is to minimize, either explicitly or implicitly, the (expected) error between the predicted ratings and the true user ratings, while in the evaluation phase, the resulting model is again assessed by that error. In this chapter, we argue that defining an error function that is fixed across rating scales is somehow limited, and different applications may have different recommendation goals thus error functions. For example, in some cases, we might be more concerned about the highly predicted items than the ones with low ratings (precision minded), while in other cases, we want to make sure not to miss any highly rated items (recall minded). Additionally, some applications might require to produce a top-N recommendation list, where the rank-based performance measure becomes valid. To address this issue, we propose an approach that focuses on these specialised recommendation goals and modify the model based on these requirement. By adjusting the objective function of the recommender system (using an internal approach - as described in Figure 1.2), we aim to improve the quality of the recommendation, focusing on types of errors that are important to the user.

3.1 Problem Statement

As argued above, improving performance on a single measure might not be a good strategy to improve user satisfaction. One example of that is the Netflix competition where the success of the contestant algorithms was validated by RMSE (Root Mean Squared Error), which measures the difference between ratings predicted by a recommendation algorithm and ratings observed from the users. It is defined as the square root of the mean squared error: $\sqrt{E\big((\hat{r}-r)^2\big)}$, where E denotes the expectation, r is the observed rating and $\hat{r}$ is the predicted value. In [HKTR04], researchers have already systematically examined many performance measures and their implications for collaborative filtering.

The RMSE metric measures recommendation error across different rating scales and the error criterion is uniform over all the items. RMSE squares the error before submitting it, which puts more emphasis on large errors. Naturally, large errors can occur at the end of the rating scales. To see this,



(a) The expected value of RMSE per rating.

(b) Percentage of RMSE improvement over random recommendation.

Figure 3.1: Basic RMSE Statistics: measured at the ordinal level.

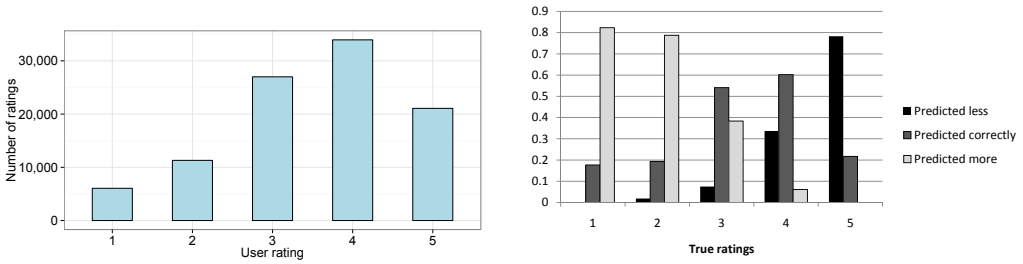
suppose we have a recommendation algorithm which predicts the rating of an item randomly from rating 1 to 5. Figure 3.1(a) shows that it is more likely to get higher error at both ends of the rating scale if a random algorithm is used.

Thus, the question arises whether RMSE should be adapted as the measure of user satisfaction. It measures the error across the system even for items that are not that important for users to be correctly predicted. In other words, the system might not want to penalise predictions that are not important for the user. If the user is only interested in receiving relevant recommendations, RMSE as a measure is not appropriate. Even if the user is interested in items that he or she would dislike, it is questionable whether the middle range of the rating spectrum is interesting to the user at all. If we take rating three out of five as the middle range of predicted values, which range cannot help to explain why an item was recommended, neither can it explain why the item was not recommended.

Another issue may also arise if one directly optimises RMSE. This is due to the fact that the training samples are not uniformly distributed across rating scales. To demonstrate this, Figure 3.1(b) shows RMSE improvement in percentage over random recommendation by using a common recommendation algorithm (in this case an SVD-based approach is used to optimise the metric [KBV09]). Since users are likely to rate items that they liked, in most cases, they give them a rating of four. So the algorithm has more data to minimise the error at that range, which results in a higher improvements rate for rating four.

Improving accuracy on items that the user would like may be desirable from a user point of view, but if the prediction falls into the middle range the error does not matter as much as if the prediction falls into the lower range. It is similar with items that are rated low; reducing the error rate is more desirable as the error rate increases since the item gets a higher prediction. In addition to that, highly accurate predictions on uninteresting items (perhaps rated 3 out of 5) can drown out poor performance on highly/lowly ranked items. Therefore, depending on the rating we need to pay attention to the direction of the offset between the rating and the prediction. Figure 3.2(b) shows that the SVD algorithm tends to over predict items in the middle range. Additionally, it is more likely to over predict lower rated items than under predict higher rated items.

Hence, a distinction can be made between items that are appealing to the user and items that are neutral. Within the interesting category we can differentiate between liked and disliked items. To decide which one is the most important, let us consider two different types of recommendations. Since the



(a) Distribution of the ground truth.

(b) Asymmetric Rating-Prediction Error Offset.

Figure 3.2: Rating-Prediction Error Offset.

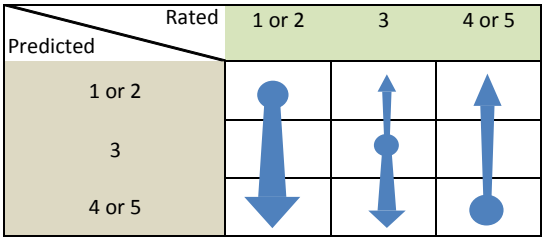


Figure 3.3: Directional Risk Preference of Recommendation Prediction

performance of a recommender system should be evaluated with respect to a specific task, it is useful to define the two main tasks that a typical recommender system fulfils. If the output of the recommender system is the first n items, then RMSE is not an appropriate objective to optimise, since it is not essential to measure the system performance on items that do not fall into the first n good items. As long as the system correctly identified that these items do not fall into the n good items the accuracy is irrelevant. Users might be interested in exploring movies, looking through the database or checking particular movies. In this case everything matters, because users are interested in the justification on how movies are made. Clearly, in both scenarios we can differentiate between two separate kinds of risks. First, the risk of recommending something that is not relevant to the users, second, the risk of not recommending something that is relevant to the users. These are two kinds of errors that should be separated when it comes to measuring the error rate. For example, assume that the system predicts a movie four, and the user watches that movie, which he or she would have rated only three. This is clearly different from the case where the system rates a movie three, which would have been rated four, if the user took the time and watched it. Since the error of the algorithm in the second scenario would never be found out, because the user would never watch a movie that is rated three, from a user point of view this error would be hidden. Therefore, a system that makes hidden errors would not be considered better by the user than a system that makes errors illustrated in the first scenario.

We introduce two new concepts here: *taste boundaries* and the *direction of taste*. Taste boundary could be defined as the interval that is between liked and disliked items. In a rating scale from one to five, this boundary would be three. Direction would represent whether the predicted rating is towards the taste boundary or not, at one level, on another level it would represent whether the error is large

Table 3.1: The two dimensional weighting function, where p is the predicted value of the item and r is the ground truth.

	$r = 1, 2$	$r = 3$	$r = 4, 5$
$p <= 2.5$	w_1	w_2	w_3
$2.5 < p <= 3.5$	w_4	w_5	w_6
$p > 3.5$	w_7	w_8	w_9

enough to cross the taste boundary or not. In other words, the algorithm would consider whether the user would like the item when it is not the case and vice versa. These boundaries are illustrated in the matrix shown in Figure 3.3. It shows that we would like to minimise errors where the prediction is correct and as we go further from the correct prediction, we take higher risk depending on the direction (the risk is illustrated by the size of the arrows). Figure 3.3 can also be applied to a ranking problem since higher predicted items represent higher risk. For example in ranking an error should be penalised more if an item is ranked higher than if it happens the other way around.

3.2 Optimising the Weighted Errors

Based on the discussion above, more risk should be penalised given a specified recommendation goal. Also, risk is directional as shown in Figure 3.3. Previous recommender systems considered the absolute value of the error, taking equally into account negative distance and positive distance from the ground truth. Here, an optimisation framework is to be proposed that would differentiate between negative and positive distance between the prediction and the ground truth rating, assigning a higher penalty for positive distance than negative distance. To achieve this, a weight for each type of error is assigned. As shown in Figure 3.3, the weights are two dimensional, depending on both the prediction and the ground truth. To demonstrate the optimisation framework, an incremental SVD (Singular Value Decomposition) factorisation method is adopted [KBV09], which is defined as follows:

$$\underset{q,p}{\operatorname{argmin}} \sum_{u,i} w(r_{u,i} - q_i^T p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{3.1}$$

where q and p are the model parameters. This algorithm factors the matrix using only user and item pairs where $r_{u,i}$ is known. As mentioned earlier weight w is introduced to control the magnitude on how conservative the system is to be in a given rating sector.

The system learns the model by fitting the previously observed rating. In order to avoid overfitting the second half of the equation regularises the learning parameters and the constant λ is set to control the extent of regularisation. Stochastic gradient descent is used to optimise the equation [KBV09] introduced by [Fun06]

$$\begin{aligned} p'_{u,f} &= p_{u,f} + \lambda * (w * e_{u,i} * q_{i,f} - \gamma * p_{u,f}) \\ q'_{i,f} &= q_{i,f} + \lambda * (w * e_{u,i} * p_{u,f} - \gamma * q_{i,f}) \end{aligned} \tag{3.2}$$

where γ represents the learning rate and $e_{u,i} = q_i^T p_u$.

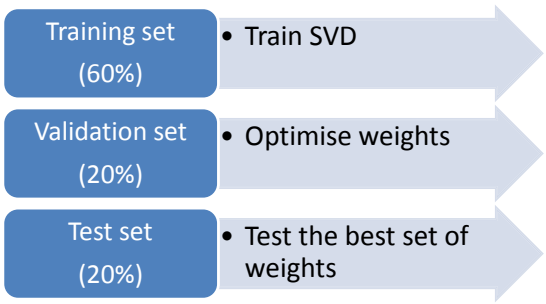


Figure 3.4: Two-Level Optimisation

The next question is how to obtain the optimal weighting w given a recommendation goal. Normally, a recommendation goal can be defined by a performance metric. For example, if the output is a ranked recommendation list, rank-based metrics such as NDCG [JK02] might be suitable. However, as explained above, the aim of this work is not to rely solely on generic performance measures, but examine the system through multiple performance measures that are in line with the defined goal. Therefore, we adopt a Genetic algorithm [Mic96] to obtain the optimal weights. Genetic algorithms are search algorithms that work via the process of natural selection. They begin with a sample set of potential solutions which then evolves toward a set of more optimal solutions. Within the sample set, solutions that are poor tend to die out while better solutions remain in the population, thus introducing more solutions into the set. The genetic algorithm does its best when there is a smooth slope of fitness over the problem space towards the optimum solution. This approach requires a two-level optimisation illustrated in Figure 3.4.

3.3 Experiment Setup

The relationships between the taste boundaries and the CF performance were empirically investigated using the MovieLens 100k dataset [Gro06]. This publicly available dataset consists of 100,000 ratings for 1682 movies by 943 users. The dataset was divided into three parts (Figure 3.4) making sure that ratings from any given user are in all of the sets. Every user in the dataset rated at least 20 movies and the movies from each user distributed randomly when the dataset was divided. This is an important criterion since the performance measures that are discussed below consider users as a point of evaluation. The result is cross-validated using a five-fold cross-validation method and the outcomes are averaged.

The algorithm measures system effectiveness based on two assumptions. First, the recommendation was considered as a ranking problem. Second, risk was defined in nine different sectors (Figure 3.3) which can be adjusted based on the desired outcome of the system. Even if the goal is to measure the effectiveness of the system across all users and items, from a user point of view there are items that are more important than others. If recommendation is considered as a ranking problem, it is sensible to optimise the algorithm using some of the following measures from IR. Therefore, two main concepts from IR should be defined in the domain of recommendations. Relevance shows whether an item is relevant to the query issued. However, the query is hidden in a recommender system, since it is defined by the user's preference which is usually not expressed explicitly. Here, we assume that users would

watch a movie if it was rated four or five on a five point scale (relevant), but we acknowledge that this might be different for individual users. Therefore, relevance is defined on a binary scale. Movies that are rated four or five are considered relevant, the rest of the movies are considered non-relevant. Retrieved items represent a list of items that are presented to the user. This concept might be beneficial if the task of the system is to return the first-N relevant items. In order to reach the desired effect we evaluated the system using measures from IR. For a given user the algorithm ranks unseen movies such that the movies he or she likes most are suggested first. The following performance measures are used in this experiment.

The Mean Reciprocal Rank (MRR) [vR79] is the reciprocal rank of the first relevant item in recommended list, averaged for all users in the dataset. As this measure only takes into account the rank of the first relevant item, the algorithm would achieve a high score if the most relevant item for the user is predicted correctly.

Mean Average Precision (MAP) [vR79] obtains the precision score after each relevant document is retrieved. The mean of this score is calculated for all users to obtain the MAP score. The algorithm would achieve a higher score if it improved the precision in the retrieved list. So in this case all the documents that are retrieved count toward the score.

Normalised Discounted Cumulative Gain (NDCG) [vR79] measures the gain based on the items position in the recommended list. This measure was introduced in [JK00]. It penalises the system if it returns highly relevant documents lower in the ranking list but penalises less if the lower end of the ranking list was retrieved incorrectly. NDCG is normalised by the perfect permutation of all the documents in the set. One of the problems if it is applied to recommendation is that the average number of ratings per user is relatively low, so that the recommender system would likely to return all the items that are relevant to the users (based on their rating history). Thus, the aspect of picking the relevant elements from a big dataset is lost here. Therefore, all users are evaluated on a fixed number of items which is set. So most of the time all the considered items are retrieved. Thus, there is no penalty on having the wrong elements within the retrieved documents; the penalty can only arise from the wrong order. In this experiment we used the formula defined in [VB06].

Since we used a small set, it was also important to define the best solution to the problem and use a measure that is relative to the best solution. This is particularly important for MRR. As mentioned above, in collaborative filtering, the algorithm is tested in a relatively small set compared to sets used in IR. Therefore, it is more likely that the algorithm is tested on users where all the items are non-relevant. So the algorithm does not have a chance to return relevant documents from a set where there are not any relevant documents. This would decrease the performance of the algorithm. Therefore, in this experiment the algorithm disregards users where there are no relevant documents in the test set. This is a reasonable assumption, because if the algorithm runs on the whole database it is very likely that there would be at least one item that is relevant to the user.

Table 3.2: Baseline SVD

	$r = 1, 2$	$r = 3$	$r = 4, 5$
$p <= 2.5$	0.05175	0.01935	0.0106
$2.5 < p <= 3.5$	0.0904	0.1461	0.1391
$p > 3.5$	0.02995	0.10125	0.4115

Table 3.3: SVD with Weights where $w7 > w8 > w4$

	$r = 1, 2$	$r = 3$	$r = 4, 5$
$p <= 2.5$	0.0759	0.04075	0.0264
$2.5 < p <= 3.5$	0.0837	0.16765	0.23815
$p > 3.5$	0.0125	0.0583	0.29665

3.4 Results

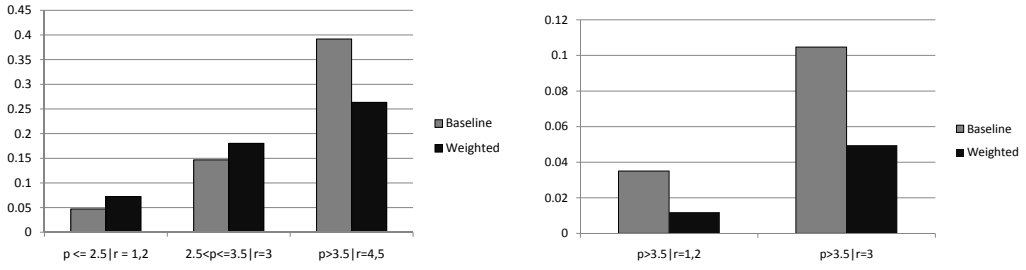
Introducing weight for the error would penalise unwanted categories, therefore, the recommender prefers to have higher error rate in categories that are not penalised. For example if an item's rating was predicted five by the baseline recommender with the ground truth of one, the weighted recommender in Table 3.3 would more likely to predict the rating less than three instead, reducing the error, but increasing the error on items that are not that important (e.g. items where the ground truth is three).

The first experiment aimed to demonstrate that introducing weight in different sectors would reduce the number of items that fell into those sectors. We introduced weights in sectors where the algorithm would make a higher prediction than the ground truth ($w4, w7, w8$) and set the magnitude of the weight in the order of risk illustrated in Figure 3.3. Table 3.3 shows that the probability that an item would fall into those sectors is reduced. However, this is a trade-off since it reduces the number of items in the sector (items that are rated five and predicted five). On the positive side, it increases the accuracy in the middle and lower range.

In the second experiment weights are set to one by default at points where the prediction should be the most accurate as defined by Figure 3.3. In this case, only weights that fall into the interval where prediction were higher than ground truth considered ($w4, w7, w8$). The rationale behind this choice is that the combination of this force (enabling the algorithm to modify only these weights) and the measure would result in an optimal solution for the user where higher rated items are considered more important and items that are over predicted are penalised. Table 3.4 shows the result of the four performance measures that are used in our experiment. The first column indicates the score that is computed using the optimal weights and the second column is the baseline score (without weights). This shows that weights in fact improved the algorithm. The samples in the table were tested and found statistically significant ($p < 0.05$). The reason why this method would provide a more robust recommendation from the user point of view is that it reorganises the ranking in a way that it would take into account our initial criteria defined by the weights and re-ranks the list accordingly. The advantage of the second approach is that it dynamically chooses the parameters for a given measure, however, as we will discuss below the

Table 3.4: Experimental Results

	Measure(Test)	Baseline(Test)
MAP	0.450	0.447
MRR	0.899	0.889
NDCG@10	0.726	0.720
NDCG@5	0.574	0.570
NDCG@3	0.450	0.447



(a) Correct predictions per sector described in Table 3.1

(b) False positives across taste boundaries

Figure 3.5: Prediction Improvement Across Taste Boundaries

measures do not cover all the possibilities given in our initial criteria. In contrast, the first approach can be tuned to reach a result that satisfies these criteria, but it cannot reach an optimal solution for all users.

Essentially this approach aims to minimise the error for the predefined sectors which inevitably results in the increase of error in other sectors. Figure 3.5(a) shows the probability that true ratings are correctly predicted within our predefined taste boundary by the optimised versus the baseline approach using the weights obtained in the second experiment (Table 3.4). As expected the baseline approach predicts higher ratings better than our optimised approach, since the optimised approach does not penalise this type of error (high ratings predicted less), whereas there is some improvement in the lower range where the error was aimed to be reduced. This approach takes the low risk approach, therefore, it hurts the performance at the higher range of the spectrum where it is less risky to predict something less, in exchange it reduces the error for items that are rated low. This means that it is less likely that users get items that are not relevant to them (Figure 3.5(b)).

It is also important to investigate how the improvement of this evaluation metrics can be translated into improvement in user experience. Using MRR as a measure would reduce the probability that an irrelevant item would be presented to the user at the first position in the list. That implies that it would reduce the chance that lower rated items are rated higher for all items and it would also reduce the chance that higher rated items rated lower given that they are relevant items. The only place where it does not fit to our initial specification is that it does not differentiate between item and item within the irrelevant category. Therefore, there is not any difference in the score if an item rated one or an item rated three was ranked higher. Therefore, parameter $w4$ does not add anything extra to this measure since it only penalises low rated (one or two) items being predicted as uninteresting items. The same

applies to NDCG; however, it is a more subtle measure so it is able to differentiate between the orders of the items in the ranked list. Therefore, an NDCG score can tell us how well relevant items are ranked, which would be an optimal solution for the user.

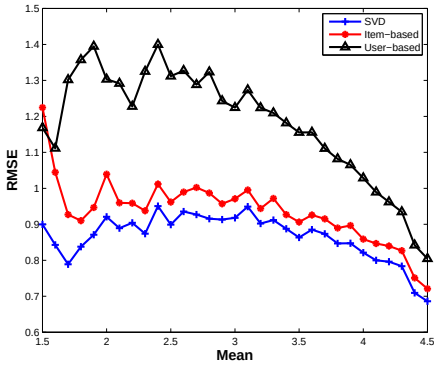
3.5 Possible extensions: Difficult and Popular Items

The main reason to identify the direction of the error was to assign some importance to items with respect to the deviance between their ground truth and predicted rating. This is based on the perception of what is important to users who expect good quality recommendations only on items that are of interest to them. There are other qualities of items that can be captured as a posterior knowledge which would affect the quality of the recommendation. Some of these features of the items can affect recommendation on a global level (as opposed to the user level we considered earlier in this chapter) such as popularity or difficulty to predict score (for a simplest case that can be based on the variance of the item ratings). These qualities of the items can determine whether the item can be *accurately* predicted as well as whether the items is *important* to be predicted correctly. For example, in order to improve the accuracy of a recommender systems for most of the users, items that are frequently rated should be predicted correctly; another strategy is to concentrate on highly rated items instead. In other scenarios, new items that have not received many ratings but has a potential to become popular should be emphasised and predicted correctly. This illustrates that on a global level the importance (e.g. popularity) of the items would help to focus on them in order to improve the accuracy of the overall system.

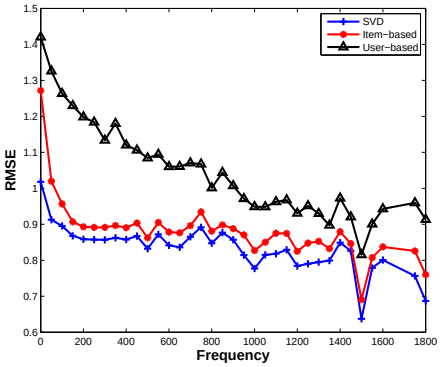
3.5.1 Selective learning

After identifying important items the prediction error on them should be reduced: boosting techniques can help to identify weak learners and improve the performance on these learners by creating a single strong learner. One example of boosting is Adaboost that was first introduced in [SF96], where it was used to generate a highly accurate hypothesis by combining many weak hypotheses, each of which with only moderate accuracy. This extension would consist of applying boosting techniques to a simple SVD algorithm [KBV09], sampling a subset of the whole dataset and concentrating on items that are likely to be predicted wrong. Our initial assumption is that this approach would have two main advantages. First, it would improve the accuracy of the algorithm; which is the basic property of boosting. In addition, it would provide a way to scale an SVD algorithm by enabling to divide the task into small segments which could be used to distribute the task. This part could be scaled using the MapReduce framework [DG08]. This approach would also provide the necessary framework to incorporate directional based errors into a more general framework, since the data can be sampled based on the direction of the error made by the baseline algorithm or the distance between items can be based on this direction.

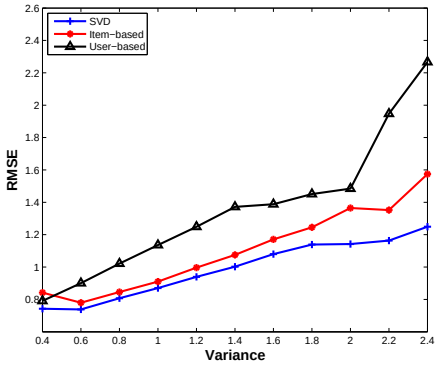
It is important to identify the qualities of the items that tend to be easily predicted, depending on the model these qualities include the frequency, the mean and the variance rating of the items and users in the dataset. We show that popularity can be defined as a combination of these qualities. In order to tackle the problem we need to investigate them separately. The frequency of an item has two very distinct effects of recommendation, first, there are items that received fewer ratings, because they are not popular, also



(a) Popularity is measured by the average rating of the item.



(b) Popularity is measured by the frequency of votes.



(c) Popularity is measured by the variance of the ratings.

Figure 3.6: Popularity of Items against the Error Rate (RMSE) for a number of popular algorithms.

there are items which has less ratings because they are new, so it is important to differentiate between these two. This quality is also important in the phrase of evaluation since the frequency of the item in the training set is proportional to the frequency in the test set given that the data was divided randomly. In the case of RMSE that means that number of test points for a given items are propositional to the number of training points, therefore, emphasizing items that have rated many times is the best way to achieve high performance (Figure 3.6 (b)). The average rating of the item is clearly a good indication of popularity. Figure 3.6 (a) shows that neighbourhood-based models are sensitive to the mean of the item whereas latent factor based models provide a more stable prediction. The reason for that is due to the fact that neighbourhood-based models capture localised relationships whereas latent factor-based models capture a different level of structure that is more global.

In addition, modelling the correlation between items and users can affect the quality of the recommendation. For example, user-based recommender systems have a higher error rate at higher ranking positions. This can be explained by the way different algorithms deal with neighbourhoods and the nature of the data. It is usually the case that the data consist of more users than items, so it is easier to find meaningful correlation between items than users. Furthermore, item correlation based on little information is more meaningful that user correlation. For instance, if we take try to define correlation based on only one rating; surely, if a user agrees on one item with other users, we cannot say that he or

Table 3.5: Average Mean and Average Variance per User

	Mean	Variance
Item-based	3.5961	0.3317
User-based 200	3.6136	0.3774
User-based 500	3.5639	0.2673
SVD	3.5610	0.3354
Observed ratings	3.5827	1.0648

she will agree on a different one. In contrast, if a users rated two different movies similarly that is a more reliable source to conclude that those movies are similar, because similarity between items is more static than between users and probably the similarity between two items can be expressed by less common ratings than a similarity between two users. This suggests that items similarity is more expressive and reliable than user similarly. User similarity involves more dimensions in terms of taste and depends on the particular items, so a single number might not be able to express user similarity, especially if it is based on little information.

The other factor that can be further investigated is the effect of the variance on recommendation. Variance shows the extent users agree on an item, which is a good indication whether an item is easy or difficult to predict. Table 3.5 shows that the predicted mean and the variance of different models vary and generally less than the real variance for the same data. This is due to the fact that each model is confident predicting items that have low variance (Figure 3.6 (c)). So that item that have higher variance are predicted towards their mean, which is responsible for the higher error and the increased variance for the prediction. This clearly suggests that personalisation does not work for items that have high variance. This observation can also be the reason why recommender systems tend to promote popular items which generally have lower variance (discussed later in Chapter 4).

Therefore, the aim is to improve accuracy for the following three distinct features: items and users with a small number of ratings, which is referred to as the cold-start problem (Section 2.4.1), items and users that have lower mean (the issue was identified in Section 3.1) and items and users that have high variance as suggested in the previous paragraph, but it is also a part of the bigger picture discussed in Section 4.3.2.

3.6 Conclusion

We introduced the first example of goal driven design in this chapter. The approach presented here puts an emphasis on the risk of making an incorrect recommendation and identifying items accordingly. The algorithm aims to optimise its performance on those items. This approach can be fine-tuned further by considering how the items would be presented to the user. Depending on the goal certain specialised metrics can be favoured. For example if a user would like to have just one item recommended, the algorithm is best optimised by MRR, or if the user would like to have more items recommended it would be better to optimise it by NDCG. In addition, different strategies depending on user needs could

be identified and the risk preference can be switched accordingly altering the recommendations. The choice of parameters can be tailored to users need penalizing sectors that are more important to predict correctly to a specific user. For example, calculating the mean of all the ratings for a particular user would suggest where the taste boundaries lie, so it can be determined for each user. On a global level (across all users), strategies such as concentrating on popular items (defined by their average rating or their frequency of ratings) would help to increase the quality of recommendation for all the users, but concentrating on items that are hard to predict (e.g. items that have a high variance of ratings) would help to reduce the overall error of the system.

As it was discussed above all the measures only care about relevant items, but for our purposes it is also important to minimise error on disliked items (rated one or two). Thus, we would like to measure how the algorithm performs on both sides of the rating scale. In both cases, the middle range (items rated three) would be considered non-relevant. These two scores could be combined taking the high rated list more into account than the low rated one.

It is a widely discussed topic that accuracy alone is not a sufficient to measure whether a recommender system provides an effective and satisfying experience [HKTR04]. It is also important to note that a data is not homogeneous. In terms of prediction, we can differentiate between easy and difficult items as well as easy and difficult users.

Here, the direction of errors was considered only for items and applied uniformly for all users, in a similar fashion users can differ from each other in terms of risk preferences which could be defined for each user and the direction of errors per user could be modelled accordingly. In this way, we would introduce another layer to the model that considers risk preferences per items first and above this level per user, so that the particular penalty for an item would be defined by the risk preference of the user whom the item will be recommended. Risk preference could be mined for the user profile, for example depending on previous rating strategy or how diverse items the user rated previously.

Chapter 4

Optimising Multiple Objectives

In the previous chapter, we focused on some aspects of the main performance measure with respect to a certain user-centred goal, the main focus of this chapter is to investigate this further and understand how multiple goals and their specialised metrics can be framed. We illustrate a multiple goal optimization approach that not only considers the predicted preference scores (e.g. ratings) but also deals with additional operational or resource related recommendation goals, based on the goal driven design outlined in Section 1.2. We start the chapter with an example of goal driven approach where the objectives are directly connected to the performance of the system. Here we study whether it is feasible to use recommender systems to optimize digital content delivery, by predicting which items would be requested and pre-caching them near the target user. This problem is framed as an external system centric goal. In the second part of the chapter, this is extended to multiple goals where the objectives might not be directly linked to the general performance of the system. Using this framework we demonstrate through realistic examples how to expand existing rating prediction algorithms by biasing the recommendation depending on other goals such as the availability, profitability or usefulness of an item. In the last part of the chapter we set to improve diverse, novel and serendipitous recommendations at the same time, at a slight cost to accuracy, using an internal optimisation approach. To some extent, these goals might be complementing with each other so that the combination of the goals, measured by specialised metrics, would provide the best user experience.

4.1 Problem Statement

To build a practical recommender system, providing items that fit to the target user's taste (recommendation accuracy) is not the only concern. Users' satisfaction also relies on the utility of obtaining recommendations to accomplish a certain information seeking task. Additionally, in a practical operational environment there might be other factors that can affect the effectiveness of the whole system. For example, many recommendation algorithms use, either explicitly or implicitly, the Root Mean Square Error measure as the objective function [Kor08, KBV09] - a typical case is the Netflix competition. To reduce the error, the algorithm has to focus on the popular items (in the training phase), because that strategy would minimise the overall error of the system. As a result the algorithm is more likely to recommend mainstream items which might be already known to the user. However, recommending these items is

likely to be less useful than suggesting "long tail", more disputable yet preferable, items. Such usefulness also depends on how "alternative" a user's preference might be. Thus, the offset of a user's taste from the mainstream has to be defined. It is certain that Root Mean Square Error alone cannot address this problem.

From the system's perspective, the organisation that is operating the recommender system expects that the system could help their users discover their products or services effectively. By the same token, other factors such as users' feedback, available resources, bandwidth and profit can also influence the productivity and performance of a recommender system. For example, recommending items that are not immediately available would be frustrating for both the user and the system. Arguably, these factors can be as important as the recommendation itself, so they should be taken into account and embedded into the system.

A recommender system can only be successful in a competitive environment if it is able to optimise multiple goals without the necessity of redesigning the whole algorithm. This motivated us to conduct a formal study on how to formulate the recommendation construction problem when multiple objectives are considered. To achieve this, the problem is formulated using constrained linear optimisation techniques [BV04]. Our idea can be simplified as follows: items were assigned to a utility score which is related to the rating of the item; the higher the predicted rating is, the more the utility score increases. However, most importantly, the utility also depends on the predefined operational objectives. The rationale of using constrained optimisation is that it can naturally specify accuracy as the main objective and other operational objectives as constraints, then optimise them in a unified framework. Two realistic situations are to be given to show how easily multiple factors can be added to this framework depending on the exact specification of the system. The first scenario is concerned with improving the quality of the recommendation by reducing the likelihood that popular items would be recommended to users with less popular taste, the second scenario concentrates on introducing constraints that can deal with other operational and business related factors.

While considering recommendation as a ranking problem, recommendation accuracy is not the first thing to consider. Instead, an operational/resource related recommendation framework is proposed in order to incorporate multiple objectives. Similar to [Wan09, WZ09, CT09], constrained optimisation techniques are applied. However, the purpose of the specific algorithm is quite different. In [Wan09, WZ09], diversification is modelled by applying portfolio theory in stock markets. In [CT09], robust query expansion is introduced by exploring the risk and reward trade-off. Here, a simple linear relevance objective is proposed, and the emphasis is on to design constraints to specify additional objectives.

4.2 Cost-Based Objectives

In this chapter, we aim to introduce an approach to incorporate certain factors in the algorithm in order to bias the recommendation based on external goals of the system. This approach assumes that users make their choices based on what is recommended to them. However, in a simpler case the outcome of the prediction can be used to make decisions in order to achieve certain objectives. In order to study this, we have to separate two interconnected effects that would influence the system: to what extent a

recommender system would be able to *predict* user behaviour (i.e. how would the user behave in the future) and to what extent a recommender system can *influence* user behaviour (i.e. proactively shaping the behaviour of the user). In this section, we are focusing on the former through an example of content delivery and the rest of the chapter will be concerned with the latter. In essence, the former one is just the study of what aspects of the recommender system can be correlated with satisfying certain objectives, whereas the latter would partly attempt to answer a more complex question on the balancing act between satisfying generic and external objectives of the system. From a different point of view, the former can represent scenarios where the system's performance might be proportional to the defined objective, whereas for the latter would be concerned with (partly) opposing objectives. As mentioned above, the rest of this section focuses on the first problem through a practical example of digital content delivery.

4.2.1 Digital Content Delivery

Video traffic continues to increase its share as more content becomes available online. Apart from watching videos online it is also used for paid video on demand (VOD) services. IPTV service providers often require a significant amount of bandwidth to deliver content to the set-top box at an acceptable quality, which reduces the capacity available to the household for other purposes. Furthermore, additional costs are often incurred to guarantee the quality of service for real-time content streaming, which means that delivering digital content in real time is more expensive than pre-caching it beforehand. This motivated us to study whether it is feasible to use recommender systems to optimise digital content delivery by predicting which items would be on demand by a particular household and pre-caching them closer to the target user. We defined this approach as an external system-focused goal based on the criteria defined in Figure 1.1 and Section 1.2. We present a preliminary study in this section to show how to evaluate current recommender algorithms in this domain and understand whether it is feasible to use recommender systems as a predictor of pre-caching techniques given based on the performance of the recommender system and the cost associated with the delivery. The aim is to identify which aspects of the current recommender systems are important, and how they could be improved to satisfy the requirements for pre-caching. The work is divided into three main parts. First, a preliminary study of the dataset was conducted in order to understand how it could be used to provide efficient recommendations. Second, the implementation of an implicit recommender algorithm was carried out. The optimisation problem of the content delivery approach was also defined and it was evaluated empirically on the available dataset.

4.2.2 Baseline Algorithm

Most of the commercial systems only have access to binary data (which consists of implicit feedback only) as the only data that can be collected is purchase history or click-through data. We considered implicit feedback recommender systems as the baseline algorithm. The model that we used is based on the approach that is outlined in [JW10a], that is user-item pairs are weighted more if we know more about them. In other words, we would like to reduce the error on items that we are confident about, so the algorithm would fit the model to these items. Mathematically this can be expressed as follows, based on the widely used SVD (Singular Value Decomposition) matrix factorisation algorithm [AT05]

$$\hat{q}, \hat{d} = \underset{q,d}{\operatorname{argmin}} \sum_{u,i} w_{u,i}(r_{u,i} - q_i^T d_u)^2 + \lambda(||q_i||^2 + ||d_u||^2) \tag{4.1}$$

where $w_{u,i}$ is the weight assigned to the error associated with the prediction ($p_{u,i} = q_i^T d_u$) for a particular user u and item i that is measured as a deviation from the observed rating $r_{u,i}$. Here we estimate an f dimensional vector for each user (d_u) and each item (q_i).

As we applied this to a binary dataset in this chapter, we needed to take into account zero preference values as an indication of disinterest. Because the model could not be optimised using a standard SVD solution [Kor08], instead it has to be solved on the whole user-item matrix S rather than only on the observed values. To achieve this we adopt an approach described in [HKV08] where the optimisation problem is solved by alternating between fixing the user-factors and the item-factors, so that the function becomes an alternating least-square optimisation process that can be solved in $O(|S|f^2 + |M|f^3)$ while it iterates through all items (q_i), similarly it can be solved in $O(|S|f^2 + |U|f^3)$ while the algorithm calculates user vectors (d_u) where f is a number of features, M is the item matrix (that contains the item (q_i) vectors) and U represents the user matrix (with the user (d_u) vectors). We use the Tikhonov regularisation to solve this model [GHO99]

$$\begin{aligned} minimise||Ax - b||_2^2 + \lambda||x||_2^2 = \\ x^T(A^T A + \lambda I)^{-1} A^T b \end{aligned} \tag{4.2}$$

This problem has the following analytical solution:

$$x = (A^T A + \lambda I)^{-1} A^T b \tag{4.3}$$

Here, A can be replaced by U or M respectively to calculate q_i and d_u, b is to be substituted with the observed rating vector per user and per item (including the unobserved zero values).

4.2.3 Extended Model

We interpreted $w_{u,i}$ (Equation 4.1) based on the unique features of the dataset (described in Section 4.2.5) that was used in this chapter. In the case of our approach, we have two different types of indication for preference. The user might purchase or watch the item, which is a strong indication of preference. In that case, we set $w_{u,i}$ to one, which was increased exponentially if the user watched an item more than once. Another case, when the user only previewed the item (i.e. watched a trailer), we understood it that the user was potentially interested in that item. We calculate this as expressed below

$$w_{u,i} = \lambda P(pu|pr, u) + (1 - \lambda)P(pu|pr, i) \tag{4.4}$$

where we obtain the probability that item i was purchased (pu) or watched given that it was previewed (pr) earlier, this was repeated the same way for the user u. The user probabilities represent the likelihood that the target user would watch the content after they previewed it. These two values can be calculated from historical data to get the item specific and user specific viewing behaviour.

4.2.4 Personalised Content Delivery

Personalised content delivery aims to use recommender algorithms to predict user preference and use this information to deliver content before an item is watched. We set to investigate whether this technique can be used in practice to provide a better user experience and potentially save costs by pre-caching assets on the set-top box or storing it on a server close to the target user. There are numerous advantages of this approach. First, if all the criteria were met it would save delivery cost for the content provider, which inevitably reduces peak-time network traffic as a side effect. It would also increase customer satisfaction by enabling customers with slow connections to watch content instantly.

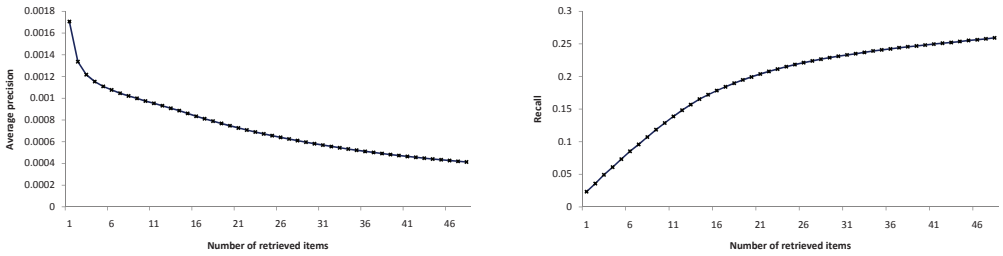
The algorithm described in this section is to be used to choose a set of items that are predicted to be watched by each customer and pre-cache them on the set-top box. In order to evaluate the system we divided the approach into two separate prediction problems that are associated with two different errors. The first step of the prediction is to estimate the number of items that are to be watched by the target user in order to define the number of items that are potentially cached on the set-top box (i.e. users who do not watch anything would not need caching). This can be personalised based on historical data or non-personalised based on the overall trend of the users. This factor is important as that would enable the system to cache the exact number of items which would minimise the overall cost. The second step is the actual recommendation problem. This prediction problem includes the baseline recommender system described in Section 4.2.2. The main problem to tackle here is to provide a low-risk recommendation. In other words, the system should identify users/items which are easier to predict and only provide recommendation if it is more likely to be correct. Users who are easier to predict include users with more historical data and less diverse choices in the past. Items that are easier to predict include items that users watch regularly (series, soap operas etc.), and items that are popular or expected to be popular.

4.2.5 Empirical Study

Dataset

This research was carried out using an anonymised dataset from an IPTV service provider with around 500,000 customers. The dataset contains four months of viewing logs, from July to October 2010. The dataset consists of 56.1 million viewing records for 240,000 users and 14,000 items. The views only indicate that the item in question was purchased to watch, watched as a part of a subscription package or pre-viewed (e.g. watched a trailer of the item). This dataset was sub-sampled, in the final experiment 25% of the users were used (64,000 users and 14,000 items). We used five different samples to cross validate the results. It is also important to note that despite that the dataset has 14,000 items; some of the items might only be available to watch/purchase for a shorter period. We were provided with a list that defined the availability of the items in the dataset; this information was used to pre-filter the recommendation list to make sure that only those items that were available at the time of the recommendation were used to test the performance of the system.

As time is an important factor in this model, we evaluated the performance of the system during a given period of time. The dataset was divided into a training set and test set; such that the first three months (July to September) were used for training the model and the last month (October) was used for



(a) Precision (b) Recall

Figure 4.1: Performance of the Baseline Algorithm (October 2010)

testing.

Evaluation

In this section, we define the theoretical threshold of the performance that should be achieved in order to make the system effective at saving costs and improving user experience. This threshold depends on two different methods of delivery and their cost respectively. We define two content delivery methods that are based on operational content delivery networks: we define best effort (BE) delivery as a method which does not provide any quality of service guarantees, and assured forwarding (AF) which prioritises traffic to ensure a certain quality of service when delivering real-time streams. An asset pre-cached using Best Effort delivery could potentially save the additional cost of Assured Forwarding, if it was subsequently watched by the user. In order to assess whether the performance of the system can stay below this threshold both of the prediction strategies (caching and prediction strategy) should be evaluated simultaneously.

To generalise the pre-caching problem explained above we formulated the problem as follows. Given that there are two different ways of delivering items, assured forwarding and best effort, we have two different values of costs associated with the system, therefore, the overall cost can be defined as follows

$$cost = c_{be} * n_{be} + c_{af} * n_{af} \tag{4.5}$$

All the items (or volume) that were delivered by each method (n_{be} and n_{af}), multiplied by the cost associated with the delivery (c_{be} and c_{af}).

Performance Error

It is also important to measure the performance of the recommender system. We mainly used *precision*, a widely used rank-based IR metrics [SM86], in this case it is defined as follows

$$precision_u = \frac{h_{u,be}}{n_{u,be}} \tag{4.6}$$

where $h_{u,be}$ is the number (or volume) of items that are pre-cached and watched by the user, $n_{u,be}$ is the number (or volume) of items that are pre-cached. We chose precision as the main measure of performance as it measures the ratio between items that are watched from the pre-cached item pool, which ratio is crucial in terms of the cost associated with the delivery of the items. We also defined recall

as

$$recall_u = \frac{h_{u,be}}{rel_u} \tag{4.7}$$

where $h_{u,be}$ is the number (or volume) of items that are pre-cached and watched by the user, rel_u is the number (or volume) of items that are watched by the user. In other words, the overall performance of the systems is defined as the hits the delivered items normalised by the number of relevant items. Figure 4.1 shows the performance of the baseline algorithm using precision and recall.

The lower bound of the performance that is needed to make the system profitable can be calculated as

$$precision_u \geq \frac{c_{be}}{c_{af}} \tag{4.8}$$

which is the ratio between the costs of the two different delivery methods can be interpreted as a lower bound of the performance to save cost. In other words, the price difference between BE and AF delivery would allow some error in the algorithms, because the saving that can be made by delivering content using a cheaper method would allow the method to deliver some (potentially) non-relevant items. The higher the price difference, the more inaccurate the algorithm can be. For example if AF would cost two times of BE, for every two items that is cached only one is enough to be relevant (i.e. precision equals to 0.5) so that the cost of delivering two items half price (BE) (out of which one might not be relevant) would equal to delivering one relevant item using a more expensive method (AF). Note that there is a slight incompatibility between measuring the performance *per item*, and the ratio between the two delivery methods *in volume* (e.g. in gigabytes).

Estimated Assets Watched Error

Figure 4.1 shows that the system has a very low precision with relatively high recall, this is due to the fact that the dataset is very sparse, so most of the users do not have any relevant items (hence the precision is zero for these users), but if there are relevant items present, the system can produce a good recall. This calls for an effective way to detect users that would not view any items, and only produce only caching for users who would likely to use the system. Therefore, we define the maximum number of items (or the volume of the overall downloads) that can be safely downloaded for a given user for any time period as follows

$$n_{u,be} \leq \frac{c_{af} * v_u}{c_{be}} \tag{4.9}$$

where c_{af} represents the cost of using real time streaming, c_{be} is the cost of pre-cached delivery and v_u represents the number of items the user will watch for a given period. As mentioned earlier v_u needs to be predicted to calculate the upper limit of the downloads. In order to measure this, we can define the maximum number of items that is cost effective to download (based on the price difference between the two delivery methods) for each item watched:

$$n_{u,be} \leq \frac{c_{af}}{c_{be}} \tag{4.10}$$

Time Based Simulation

In order to simulate the working of a real system, we designed a time based simulation to evaluate the performance of the algorithm. We evaluated the system for a month. Each day (at midnight), every user

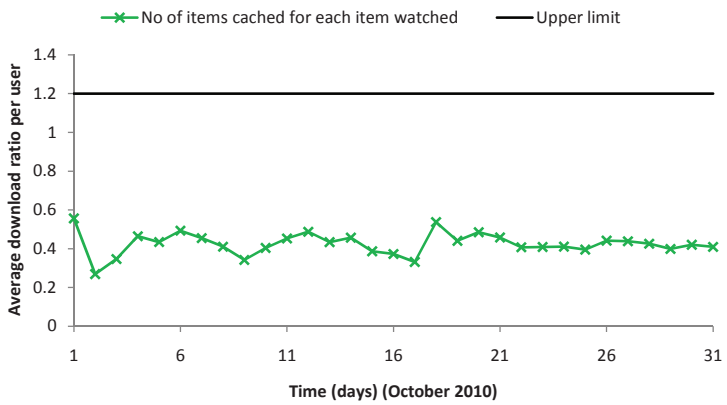


Figure 4.2: The ratio between cached and viewed items per user. Given the pricing structure, the system can cache maximum 1.2 items for each viewed one.

received a recommended list from the baseline algorithm. This was pre-filtered to make sure that the items that were provided were available for viewing (i.e. each item might be available for a certain time period depending on the licence between the producer and the content provider). Then it was checked whether these items were already on the set-top box, if not then the system cached the items. Each day, it also removed items that were not available anymore. This list was then evaluated based on the actual viewing of the user during the day.

4.2.6 Results

We chose values for c_{be} and c_{af} such that the ratio between them was 1.2 (indicating that assured forwarding is 20% more costly than best effort), which was considered representative at the time the experiments were carried out. As an example, we set the price of assured forwarding delivery to 0.6 unit per gigabyte and best effort delivery to 0.5 unit per gigabyte.

The system used viewing history to predict the number of items to download. In this initial experiment we enabled the system to download one item for each user at the beginning of the test period and more later if the user profile suggested that the user would watch more items. We employed a simple approach that is based on the historical data, which determined how many items the user would watch. Figure 4.2 shows the average number of cached items for each viewed item per user (Equation 4.10). Based on this particular pricing structure, this could be increased until caching 1.2 gigabyte data for each watched gigabyte. In other words, the system could cache 1.2 gigabyte of data for each possible gigabyte watch using the BE method on the cheaper price, which would make the cost of the service cheaper if the cashed items are actually watched. However, here we decided to have a more conservative caching strategy in order to ensure that items that are cached would be more likely to be watched (i.e. the system should only cache items that are more likely to be relevant).

The second step was to assess the performance of the recommender algorithm. Figure 4.3 shows how many items (in gigabyte) were watched from the items the system cached for the period of one month (Equation 4.6). The line labelled 'all' represents the overall performance of the system; the

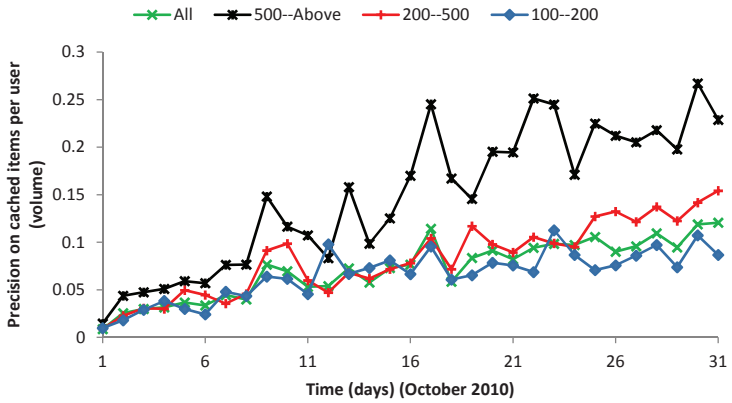


Figure 4.3: Precision on Cached Items per User

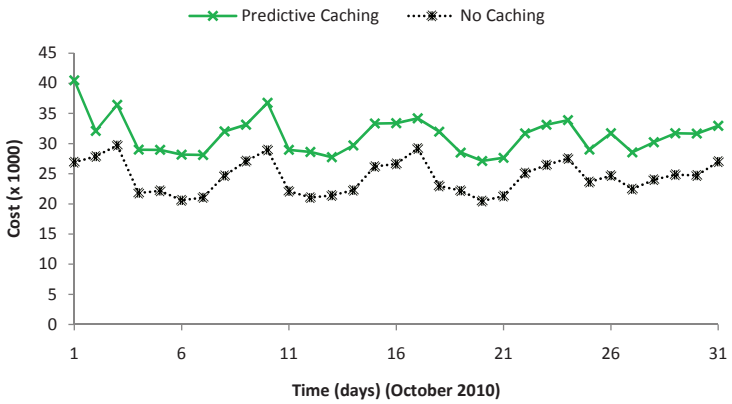


Figure 4.4: Overall Cost. In this example the price was set to 0.6 unit per gigabyte for AF and 0.5 unit per gigabyte for BE.

remaining lines depict the performance based on the profile size of the user. For readability, we only plotted those profile sizes that outperform the overall value. The best value in this case would be one which means each item that was downloaded was also viewed. In order to make the system profitable using the current pricing structure this value should be higher than 0.83 (Equation 4.8). In addition, if the price difference between assured forwarding and best effort increases sufficiently, so that the ratio (Equation 4.8) is lower than the performance (Figure 4.3), the system could become profitable using the current setup.

The reason why this precision is increasing over time (even if the performance of the system decreases in this case) is that there are more items accumulated on the box which increase the performance over time. This is especially noticeable for users which have higher profiles (i.e. over 500 items). Since the performance of the system did not exceed the threshold that is needed to save cost, it is expected that the overall cost of the system using predictive caching would be higher than using purely assured forwarding. Figure 4.4 shows the delivery cost for the October 2010, the difference between the two lines is less if the system achieves higher precision (Figure 4.3), for example on the 17th of October.

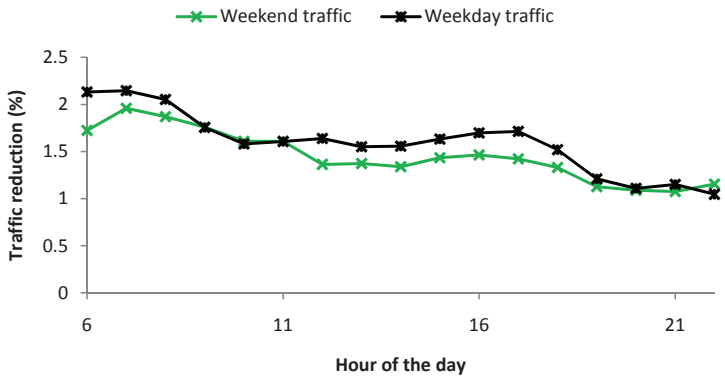


Figure 4.5: Network Traffic Reduction

The next step towards personalised content delivery is to look at the actual user profiles and manage according to the risk the target user poses. The simplest approach is to use the prediction value provided by the recommender algorithm as an indication of risk and only cache items if they reached a certain threshold. This could be used to avoid caching items if they are predicted low, but still positioned high in the ranking list. This would reduce the risk of increasing the number of items cached, which is a trade-off between providing only items that are very likely to be predicted relevant and items that positioned lower in the ranking list.

Reducing peak time traffic

The other important factor this approach has on system dynamics is that predicted items can be delivered off-peak time, which would reduce the traffic that is especially high at peak time. Figure 4.5 depicts the percentage of the network traffic reduced using the approach described above. It shows that predictive caching can reduce traffic by up to 2%. Even at peak time, when the traffic volume increases substantially (which is around 7pm during the week and 3pm at the weekends), it achieved over 1% reduction. The reduction of the traffic is proportional to the performance of the system, so further improvement can be achieved by improving the overall performance of the algorithm.

4.2.7 Possible Extensions

As described above the problem can be broken down to two separate prediction problems. First, it is important to predict how many items the user will watch in the future, this problem was approached by defining a heuristic method that is based on users' history and cached items accordingly. As Figure 4.2 shows this resulted in a performance that is within the cost saving threshold. However, this could be improved by predicting not only how many items will be watched but also when these items will be watched, so that recommendation can be made closer to the time they are watched based on more information which would improve performance. The other important factor is to define a reliable way to measure predictability based on users' profiles, where this approach could work, so that the system could assess whether the target user is easy to predict and apply the approach accordingly. This threshold can be adjusted based on how much space the algorithm has depending on the current price of delivery (i.e.

the error rate that is acceptable to make personalised content delivery profitable).

4.2.8 Complementing Objectives

In this section, we presented an example where external system objectives are proportional to the performance of the recommender algorithm. This assumes that maximising the general performance of the system would also satisfy the given objectives. In this case, the aim was to discover the relationship between the given objectives and the system performance, as the performance of the system might provide a good indication of the performance of the external objectives. This is simply the case, because improving the performance of the system would reduce false positives, which are responsible for increasing the cost of delivery.

4.3 External Multiple Objectives

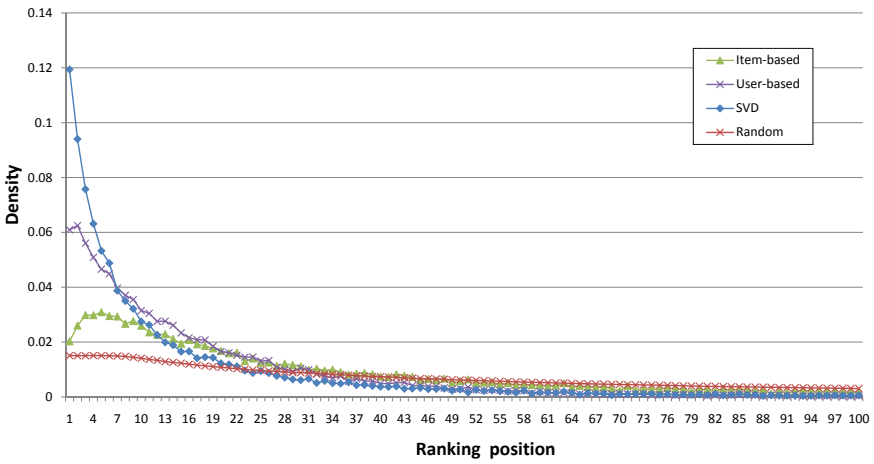
We continue our discussion by focusing on objectives which might not be directly linked to the general performance of the system. Single objectives that are studied here might reduce the general performance. In addition, satisfying multiple objectives could be viewed as a balancing act between the objectives (which can be measured using specific performance metrics) and the general system performance. To formulate this view, first we introduce a general framework. Let us begin with the notation. Suppose that we obtain a rating prediction for each item that can potentially be recommended to the user. The predicted ratings of the items can be conveniently denoted as a row vector $\hat{\mathbf{r}} = \{\hat{r}_1, ..., \hat{r}_{n_u}\}$, where n_u is the number of the candidate items for user u. Normally n_u equals the number of all unseen items for user u.

Conventional systems would already do recommendation by ranking items based on their predicted ratings. Here, however, to distinguish between rating prediction and recommendation utilities, we now, for each of the items, assign a positive weight to describe its importance or utility on the recommendation list. Mathematically, we have a row vector $\mathbf{w} = \{w_1, ..., w_{n_u}\}$. The weight is normalised so that: 1) The summation of its elements equals 1, i.e. $\mathbf{1}^T\mathbf{w} = 1$, where $\mathbf{1}$ is a n_u-dimensional vector whose elements are all 1, and 2) the weights are positive, i.e. $\mathbf{w} \succ 0$, where $\succ$ represents the inequality for all elements in the vector.

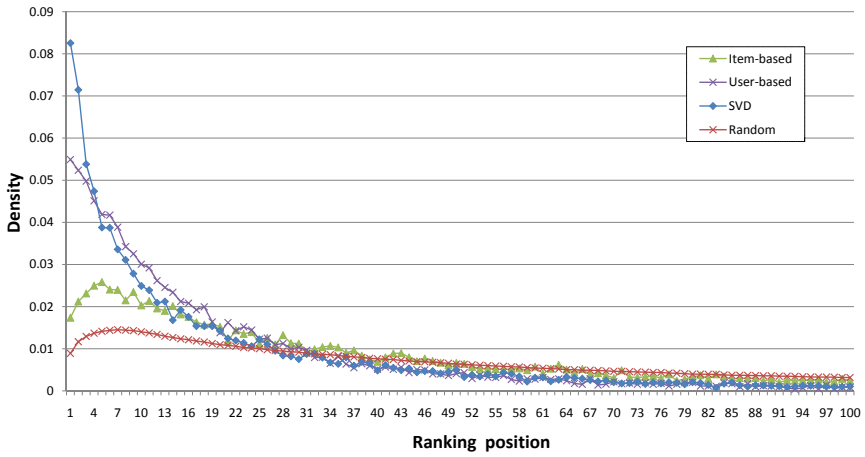
Higher weights represent higher importance on the ranking list. Clearly, an item that has a higher predicted rating would have a higher chance to be recommended, thus a higher weight. The question is how to find the weights by considering not only the predicted ratings but also the constraints enforced to the items. To achieve this, we consider the following optimisation problem:

$$\begin{aligned} &\text{maximise}_{\mathbf{w}}\ \mathbf{w}^T\hat{\mathbf{r}} \\ &\text{subject to: } \mathbf{1}^T\mathbf{w} = 1 \\ &\qquad\qquad \mathbf{w} \succ 0 \\ &\qquad\qquad \textit{Additional constraints of } \mathbf{w} \end{aligned} \tag{4.11}$$

By casting the problem as a simple linear optimisation problem [BV04], we naturally extend the system towards multiple goals; each goal will be considered as a constraint and defined in the following sections.



(a) High Mean



(b) Low Variance

Figure 4.6: The Distribution of Popular Items against Rank Position

4.3.1 Case Studies

We continue our development by introducing two basic scenarios that are designed to illustrate the importance of optimising multiple objectives. We aimed to create scenarios that might be useful in practice and can be generalised to be able to apply them to a wide range of problems.

User Case - Promoting the Long Tail

One of the shortcomings of current recommendation systems is that they promote already popular items. We argue that this can partly be explained by the fact that most of the users prefer popular items, but it is also the result of that it is more likely to get the recommendation right if popular items receive a high prediction from the algorithm. Thus, higher rated items might be recommended higher on the ranking list over *all* users. To show this we obtained the first one hundred popular items and computed the probability of those items occurring at each ranking position. We measured popularity by the average rating (Figure 4.6 (a)) given that items received a sufficient number of votes and also by the variance (Figure 4.6 (b)) such as that lower variance implies popularity. The latter assumption is based on the

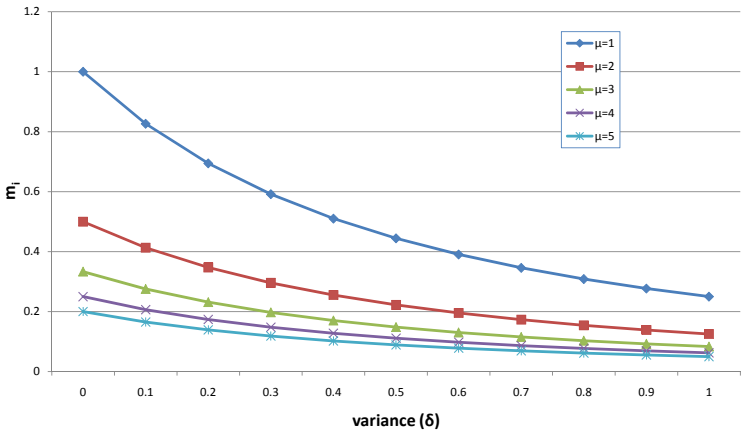


Figure 4.7: The relationship between m_i (as defined in Equation 4.12), mean (μ) and variance (σ).

observation that users' opinion tend to be similar on popular items, hence the low variance. Figure 4.6 (a) and (b) show that some of the widely used algorithms (user-based, item-based and SVD) tend to follow this pattern and recommend items higher if they are popular compared to a random sample. For example the probability that an item is ranked at the first position on the ranking list given that it is popular is 12% (mean) and 8.2% (variance) using the SVD algorithm (Figure 4.6 (a) and (b)), whereas it is only 1.5% (mean) and 0.9% (variance) from a random sample. Furthermore, these popular items (Figure 4.6 (a)) were recommended 2730 times at position one. This means that 45% of all users have one of these items recommended at the highest position and all the users in the dataset have at least one of these items on their top three recommendation list (these statistics was obtained from the MovieLens 1m dataset). To avoid this, it is important to determine the extent to which users are likely to be interested in popular items, so the recommender system could provide recommendations accordingly. Ideally, the algorithm would keep providing popular item for users who are interested in these items and it would provide alternative choices for users who are more likely to be interested in exploring less popular items. Thus, we aim to illustrate how we can propagate items that can be found in the long tail and match those items to users accordingly.

One simple approach that would help to decide whether the user is interested in popular items is to compute a value that would differentiate between items that are popular (based on the observation depicted on Figure 4.6 (a) and (b)) and items that are less popular, yet still interesting for the user. This can be expressed mathematically as follows:

$$m_i = \frac{1}{\mu_i(\sigma_i + 1)^2} \tag{4.12}$$

where the score of an item is the reciprocal of the mean (μ_i) and the variance (σ_i). One is added to the variance in order to avoid division by zero. We also decided to square the variance to emphasise higher values to reflect the disagreement among users which was intended to be the most important part in this formula. In this way, we can identify items with fairly high mean and fairly high variance, because there is no consensus on those items, they might be interesting for the user. Therefore, m_i is lower if we have disputed items with fairly high popularity and it is higher for items where users agreed on whether it

is good or bad. This relationship is illustrated in Figure 4.7 where we plotted the outcome of m_i with respect to mean and variance. It shows that we get the lowest m_i values when we have high variance combined with high mean, which are the disputed items that can be found in the long tail.

In order to measure the extent to which users are interested in disputed items, we used the same approach described above for all items that the user rated relevant

$$m_u = \frac{1}{n_l} \sum_{i=1}^{n_l} \frac{1}{\mu_i(\sigma_i + 1)^2} \tag{4.13}$$

where n_l is the number of relevant items rated by the user.

An extra constraint is added to the original equation which would ensure that users who have mainstream taste get popular items recommended (which would happen by default) and users who are interested in items in the long tail would get their taste satisfied, as well. The Long Tail Constraint (LTC) that is defined in Equation 4.12 and 4.13 can be added as an inequality constraint to our optimisation problem as

$$\begin{aligned} &\text{maximise}_{\mathbf{w}}\ \mathbf{w}^T \hat{\mathbf{r}} \\ &\text{subject to: } \mathbf{m}^T \mathbf{w} \leq \beta m_u \\ &\mathbf{1}^T \mathbf{w} = 1 \\ &\mathbf{w} \succ 0 \end{aligned} \tag{4.14}$$

where $\mathbf{m} = \{m_1, ..., m_{n_u}\}$ is a row vector that contains the corresponding m_i values (Equation 4.12) for all items returned by the recommender engine for the given user and β is a parameter set to control the extent of users' alternative taste.

System Case - Resource Constraint

This constraint is aimed to illustrate how we can embed other external factors into the system that are not directly related to recommendation. At the abstract level this can include all the operational factors such as profit margin of items, the currently available bandwidth to supply digital content etc. The scenario that we present below is the availability of items in stock during recommendation. This might be useful for companies who supply hard copies of items, which cannot be produced instantly. For example, DVD rental companies like Lovefilm (UK) or Netflix (US) could only recommend items that are in stock, which would reduce the probability that users choose movies that cannot be provided. Another application of this approach might be that digital television service providers (e.g. BT Vision) could recommend items that are already stored locally or on a server close to the target machine which would reduce network traffic at peak times.

We embed this into a probabilistic framework that would provide a more flexible interpretation of the problem than simple stock level values. It would be modelled as a stochastic process where the outcome depends on whether the user chooses to consume products or use services. For each user we have a row vector $\mathbf{p} = \{p_{1,u,t}, ..., p_{n_u,u,t}\}$, that is ranked from 1 to n_u based on the initial predictions provided by the recommender engine where $p_{i,u,t}$ represents the accumulated probability that user u chooses item i. Every time an item is shown to the user the probability that the user will choose the item increases:

$$p_{i,u,t}(e_{i,u,t}|r_{i,u,t}) \propto \frac{1}{log_2(k_{i,u}+1)} \tag{4.15}$$

Here, $k_{i,u}$ represents the ranking position for item i on the ranking list of user u. This probability value depends on the position it occupied when it was presented (i.e. the higher its rank was, the most probable that it was chosen). It also depends the accumulated probability values ($p_{i,v,t-1}$), that is the probability that the item was chosen when the item was presented to other users (v) in the past ($t-1$).

$$\begin{aligned} &p_{i,u,t}(e_{i,u,t}|r_{i,u,t}, p_{i,t-1}) = \\ &= \begin{cases} p_{i,v,t-1} + p_{i,u,t}(e_{i,u,t}|r_{i,u,t}) & \text{if } e_{i,v,t-1} = 0 \\ p_{i,v,t-1} + p_{i,u,t}(e_{i,u,t}|r_{i,u,t}) - 1 & \text{if } e_{i,v,t-1} = 1 \end{cases} \end{aligned} \tag{4.16}$$

An event $e_{i,u,t}$ occurs when the cumulative probability p_i for an item exceeds one. In this case, it means that the user chooses to consume item i. We also introduce threshold c that would ensure that items whose p_i value is below the threshold would not be affected by the constraint. Therefore, let us also define $\mathbf{s} = \{s_{1,u,t}, ..., s_{n_u,u,t}\}$ as a row vector where $s_{i,u,t}$ represents the cut-off probability value that will be passed to the algorithm.

$$s_{i,u,t} = \begin{cases} 0 & \text{if } p_{i,u,t} \leq c \\ s_{i,u,t} & \text{if } p_{i,u,t} > c \end{cases} \tag{4.17}$$

In other words, c is set to control the threshold from which the system starts re-ranking items, which depends on our choice of what we consider as a cut-off point in the model. For example if we want to make sure that $s_{i,u,t}$ does not reach one (which will useful for the simulation discussed later) we need to set a threshold that would give enough space to the model to re-rank items as they approach one. The reason for introducing this threshold is to make sure that items which have low $s_{i,u,t}$ values are not penalised, since our model finds the optimal ranking order based on the $s_{i,u,t}$ value of each item, and the relative distance between the $s_{i,u,t}$ values of the items. We express this as an inequality constraint

$$\mathbf{s} \times \mathbf{w} \prec s_u \tag{4.18}$$

where we have a row vector $\mathbf{w} = \{w_1, ..., w_{n_u}\}$ that represents the weights that determine the extent to which items will be re-ranked and $\mathbf{s} = \{s_{1,u,t}, ..., s_{n_u,u,t}\}$ is a row vector that contains the corresponding $s_{i,u,t}$ values (Equation 4.17) of all items returned by the recommender engine for the given user. Also, symbol $\prec$ represents the inequality for all elements in the vector and symbol $\times$ is the operator for element-wise vector multiplication (i.e. $\mathbf{x} \times \mathbf{y} = \{x_1 * y_1, ..., x_n * y_n\}$). Now using vector $\mathbf{s}$ we define s_u for each user as follows:

$$s_u = \frac{1}{\sum_{i=1}^{n_u} \frac{1}{s_{i,u,t}}} \tag{4.19}$$

The Resource Constraint defined in Equation 4.18 is used to extend the optimisation problem. This

inequality constraint would ensure that the system will recommend items that are more likely in stock:

$$\begin{aligned} &\text{maximise}_{\mathbf{w}}\ \mathbf{w}^T\hat{\mathbf{r}} \\ &\text{subject to: } \mathbf{s}\times\mathbf{w} \prec s_u \\ &\mathbf{1}^T\mathbf{w} = 1 \\ &\mathbf{w} \succ 0 \end{aligned} \tag{4.20}$$

One of the drawbacks of Equation 4.19 is if higher ranked items are out of stock, the algorithm starts to rank lower rated items higher, which might hurt the performance of the system. So the if we have an item which is predicted five (out of five) and an item that is predicted one, we might prefer not to place the second item higher than the first one even if the first one is out of stock. Therefore, we introduce another parameter α that could be set to control the extent the algorithm to re-rank lower rated items. We modify Equation 4.19 as follows:

$$\begin{aligned} s_{lb} &= \frac{1}{\sum_{i=1}^{n_u}\frac{1}{s_{i,u,t}}} \\ s_{hb} &= \frac{1}{\sum_{i=1}^{n_l}\frac{1}{s_{i,u,t}}} \\ s_u &= s_{lb} + \alpha(s_{hb} - s_{lb}) \end{aligned} \tag{4.21}$$

where n_u is the number of items returned by the recommender for the given user and n_l is a number of items that are predicted relevant for the same user.

Varying α from zero to one would help to fine-tune the system, giving priority to performance ($\alpha = 1$) where only items that are predicted relevant re-ranked or prioritizing stock availability depending on particular needs. In other words, decreasing α simply increases the number of potential items which are taken into consideration during the process of re-ranking.

4.3.2 Experiments

We empirically investigated the performance of extending the system with the constraints discussed above. The experiments were conducted with the MovieLens 1m dataset. This publicly available dataset consists of 1 million ratings for 3900 movies by 6040 users. We divided the dataset into test (40%) and training (60%) sets making sure that ratings from any given user are in both of the sets. Every user in the dataset rated at least 20 movies and movies from each user are distributed randomly when the dataset was divided. This is an important criterion since the performance measures that are discussed below consider users as a point of evaluation. The results were cross-validated using a five-fold cross-validation method and the outcomes were averaged.

For each scenario, we aimed to illustrate the performance of the particular constraint following the approach introduced in [JW10a], so that first we considered the performance based on the expectation of what aspect(s) of the system would be improved. Most of the time it is trivial and easy to measure (e.g. increasing the gross profit), but sometimes it is not that straightforward (e.g. providing a wider variety of items for the user). In addition to that, we assumed that the aspects in question cannot be measured using a general evaluation metric and they likely hurt the performance of the system. Therefore, we also used other generic evaluation metrics which were set to measure the performance of the system as a whole.

We considered the outcome of the recommendation as a top-N list, because, in most cases, this is the way the system presents recommended items to the user. Therefore, the best way to measure such performance is to use various IR metrics. One of the evaluation metrics we used was Normalised Discounted Cumulative Gain (NDCG) [vR79] that measures the gain based on the position of the items on the recommended list. This measure was introduced in [JK00]. It penalises the system if it returns highly relevant documents lower in the ranking list but penalises less if the lower end of the ranking list was retrieved incorrectly. NDCG is normalised by the perfect permutation of all the documents in the set. The other measure we used was Precision, which is simply the fraction of retrieved documents that are relevant to the given user. We have worked with binary relevance, that is we considered an item relevant if it was rated four or five (on a rating scale of one to five) and non-relevant otherwise. In a similar fashion, an item was considered predicted relevant if its rating was predicted over 3.5 and non-relevant otherwise.

User Case - Promoting the Long Tail

As explained above the aim of this approach is to promote less popular items to users who might be interested in these items without sacrificing accuracy. Since this interest is highly subjective and hard to measure, there is no straightforward way to show that our approach works correctly. As the aim was to promote items that are in the long tail, one approach to measure performance is to monitor the probability whether popular items get recommended higher than randomly selected items. This approach can only provide a satisfactory proof if we accept that our test set contains users who are indeed interested in alternative choices, because satisfying those users would result a decrease in the probability that popular items ranked higher if the model works correctly. However, the extent of this reduction might depend on the number of users whose taste is not popular. In the following experiments, we set to explore the effects of this constraint on the quality of the recommendation combining the result with diversification to produce more personalised outcomes.

The initial reason why we needed diversification was that we aimed to promote items which are disputable with fairly high mean and variance (see Equation 4.12). This would inevitably result in higher risk of making the recommendation wrong. Therefore, adding diversification to the equation, which would promote disputable items that differ from each other (with higher covariance), might reduce performance loss. This was studied in portfolio theory of document ranking [WZ09] and query expansion [CT08], where it was argued that diversification might reduce the risk of expanding such a system without losing on performance by ranking results based on risk and reward. Thus, we extended Equation 4.14 as follows

$$\begin{aligned} \text{minimise}_{\mathbf{w}} \ & -\mathbf{w}^T\hat{\mathbf{r}} + \lambda \mathbf{w}^T \Sigma \mathbf{w} \\ \text{subject to: } & \mathbf{m}^T\mathbf{w} \leq \beta m_u \\ & \mathbf{1}^T\mathbf{w} = 1 \\ & \mathbf{w} \succ 0 \end{aligned} \tag{4.22}$$

where Σ is the $n_u \times n_u$ covariance matrix for the candidate items and λ is the parameter that can control

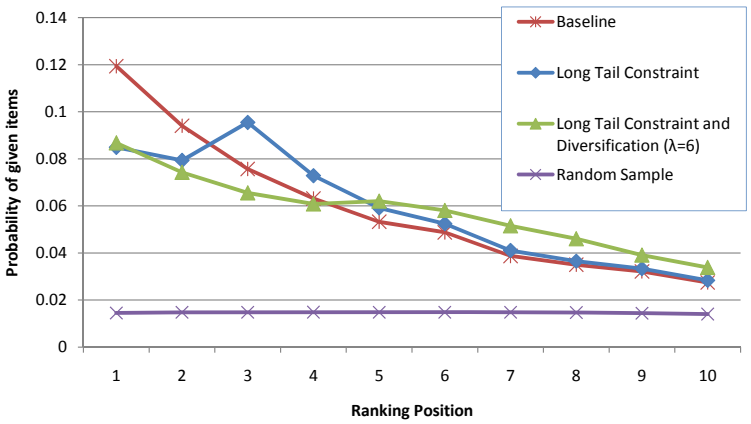


Figure 4.8: The Distribution of Popular Items (High Mean) against Ranking Position

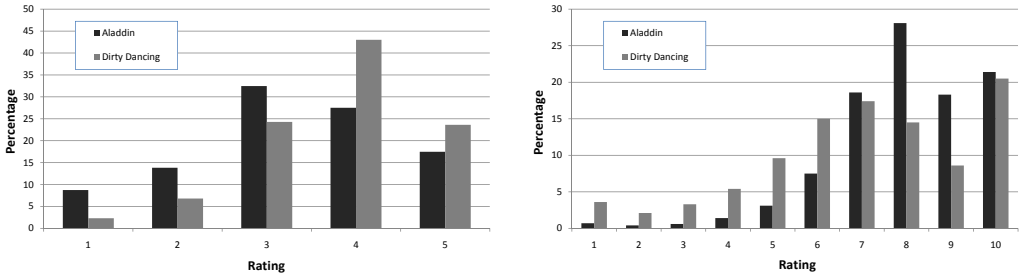
Table 4.1: Performance of the Long Tail Constraint (LTC) and Diversification (Div.).

	Baseline (SVD)	LTC	LTC and Div. ($\lambda = 6$)
NDCG@10	0.8808	0.8780	0.8715
P@10	0.8204	0.8207	0.8177
MRR	0.9518	0.9453	0.9349

the extent of diversification. It is important to note that this addition made our objective function non-linear, so the function was casted as a convex optimisation problem for this part of the experiment.

The other problem was how to obtain the covariance matrix Σ. First, we approached the calculation using a simple unbiased estimator, taking users previously expressed preferences as the observed values. However, for our applications this estimate was not acceptable because the estimated covariance matrix was not guaranteed to be positive semi-definite. Therefore, we obtained two rectangular matrices from the original $user$-by-$item$ matrix using Singular Value Decomposition factorisation ($O = UDV^T$). The obtained matrix V was used to estimate the covariance matrix for Equation 4.22.

We also ran some initial experiment to explore whether diversification would be enough to produce alternative choices for the user. One of the important initial findings was that diversification alone does not reduce the probability that popular items are ranked higher, simply because popular items can be diversified among themselves. It is the constraint that we introduced in Equation 4.14 that promotes items from the long tail and diversification distributes it over the top positions. However, using the constraint without diversification new items that come from the long tail would more likely to be promoted very high on the ranking list, but with the combination of diversification the curve gets smoother. This can be observed in Figure 4.8 where we plotted the probability of the ranking position of the first one hundred most popular items (in the same way as in Figure 4.6 (a)) concentrating on the first ten positions (β is set to one in this experiment). As Figure 4.8 shows that the Long Tail Constraint successfully reduced the probability that items get recommended in ranking position one and two, but it increased the probability for ranking position three and four. This happens because the algorithm places long tail items at the top



(a)source: Movielens 1m (b)source: imdb.com

Figure 4.9: The Distribution of User Ratings for Aladdin and Dirty Dancing

Table 4.2: Top 10 recommendation list using the Long Tail Constraint (LTC) and Diversification (Div.) against the baseline. The performance improved by 78% for NDCG@10 and 50% for Precision@10.

	Baseline (SVD)	LTC and Div.
1	Aladdin(Animation)	Dirty Dancing(Musical)
2	Grease(Musical)	Princess Bride(Adventure)
3	The Prince of Egypt(Musical)	Dead Poets Society(Drama)
4	The Little Mermaid(Musical)	Top Gun(Action)
5	Immortal Beloved(Drama)	The Goonies(Adventure)
6	Miracle on 34th Street(Drama)	Corrina, Corrina(Comedy)
7	Dragonheart(Fantasy)	Titanic(Drama)
8	Peter Pan(Musical)	Willow(Action)
9	Homeward Bound(Adventure)	First Knight(Adventure)
10	Dead Poets Society(Drama)	Dune(Fantasy)

and the items that were there before are ranked lower accordingly. This is where diversification can help to distribute those items evenly. As a result, only 32% of the users had a popular item in the first position of their recommended list (compared to 45% of the baseline algorithm) and 85% of them on their top three list (compared to all the users of the baseline algorithm). In terms of performance Table 4.1 shows that we only have a minor performance loss between the baseline and our combined version, which is 1.05% for NDCG@10, 0.3% for Precision@10 and 1.7% for Mean Reciprocal Rank. The samples in Table 4.1 were tested and found statistically significant ($p < 0.05$).

Furthermore, it is important to highlight that the Long Tail Constraint is very likely to produce a completely different top-10 list which contains new items that would not have been there otherwise. Table 4.2 consists of a randomly selected user's top-10 recommendation lists comparing the baseline algorithm with the Long Tail Constraint (keep it in mind that this dataset was collected in 2000). There are two main things to note here. First, using the Long Tail Constraint we are able to provide a different range of movies that might be not that popular. In addition, a wider range of genre is present on the list. Figure 4.9 (a) and (b) show the rating distribution for the two items that are ranked number one by the baseline algorithm and the Long Tail Constraint. They show that Dirty Dancing has a much higher

variance that Aladdin, yet still a fairy high mean, this suggest that there is less agreement among users on whether Dirty Dancing is a good movie which would suggest that Dirty Dancing might be a good candidate as an alternative choice for adequate users.

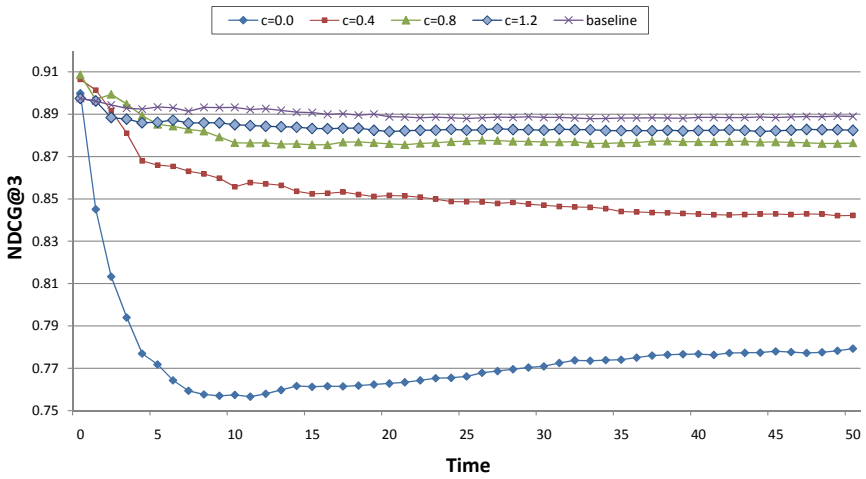
In this case the variance of Dirty Dancing can be accounted for the fact that this movie is aimed for a female audience and high variance arises from a gender related factor, that divides the audience, but high variance can also arise due to other factors such as political orientation, controversy etc. Despite the differences they all share one thing in common, that is higher variance is the result of that users are divided which would occur with items that are worthy of such a division. Therefore, we consider this as one of the distinguishing features of items that can be found in the long tail.

In the case described above ranking Dirty Dancing high was a good decision, the item turned to be a good alternative choice for this user, since we know that the user marked it relevant. Despite that, the baseline algorithm ranked it to the 22nd position. However, the question arises if we can always decide with high confidence whether high variance implies that an item is relevant to the user given that the user is interested in such items. For example, a politically controversial movie that has high variance can only be recommended to users who are interested political controversy. The reason why the algorithm is able to decide is that we apply the combination of two main forces. The main algorithm that predicts whether the user's taste is close to the target item (in our example, romantic movies) and the Long Tail Constraint picks out movies from potentially relevant items that can offer a good alternative to the extent that is determined by the user's interest in such items.

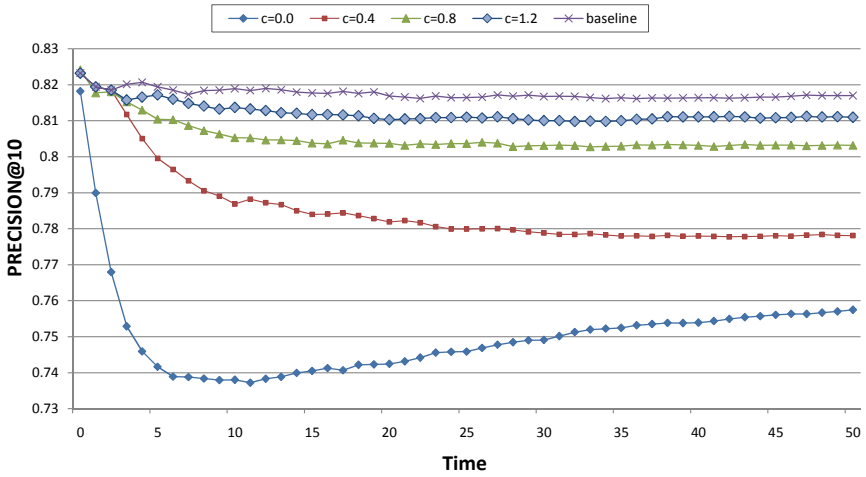
System Case - Resource Constraint

In order to stimulate the scenario that concerns the availability of recommended items (as described in Section 4.3.1), we designed a simulation that reflects a real-world environment. In fact, we only aimed to create a semi-realistic simulation, since we aimed to model the effects of our constraint on the performance of a recommender system separating it from other factors. Therefore, we made a basic assumption that users base their decisions to borrow/buy items purely on recommendations. This set-up would model the relationship between the constraint we introduced and the quality of the recommendation but ignore the part of the users' decision making process that is independent of recommendation.

We set up the simulation as follows. We stimulated time by using time ticks, each time tick represented a day. We modelled a day as a cluster that groups together a decision of certain number of users to choose (or not choose) and item. At each time tick we presented a list of items to a number of randomly selected users from our user base. The number of users we selected varied over the simulation with an average of one hundred. We assigned a probability of choosing an item from the first ten items that are presented to the user assuming that this probability depends on the rank of the item on the list as described in Equation 4.15 and 4.16 based on the assumption that the rank of the items reflects the true preference of the user. The probability for each ranking position was calculated using Equation 4.15 where the function is normalised for the top ten items on the ranking list, modelling users' interested as a cumulative function as described in [JK02]. Since we had approximately one hundred users who used the system each day with the overall probability of taking one item per user, this meant that maximum



(a) NDCG@3



(b) Precision@10

Figure 4.10: Stock Simulation (Resource Constraint)

one hundred items could be borrowed a day. The simulation ran for 50 days and we monitored how the current stock could deal with the demand.

We operated this in an imaginary warehouse that can store 5000 items. We kept the number of items low in the warehouse to model a situation that would be not feasible without introducing restrictions on the recommendation (e.g. many items would run out of stock after a couple of days). In order to make our system more effective we stored more copies of items that were popular (i.e. higher rated), but at least one copy of each item. As items were recommended more, the probability that they would be taken out increased. After an item was taken, the recommender took into account that we had less items in stock, and modified the recommendation accordingly, promoting items that were available. To make the simulation more realistic, we assumed that if an item was out of stock and there was demand for that item, the user put the item on her waiting list (which is the way how DVD rental companies operate) and after it had returned from another user the item would be supplied to users who had already requested it.

Updating the stock availability of items in real-time might be problematic to implement in practice

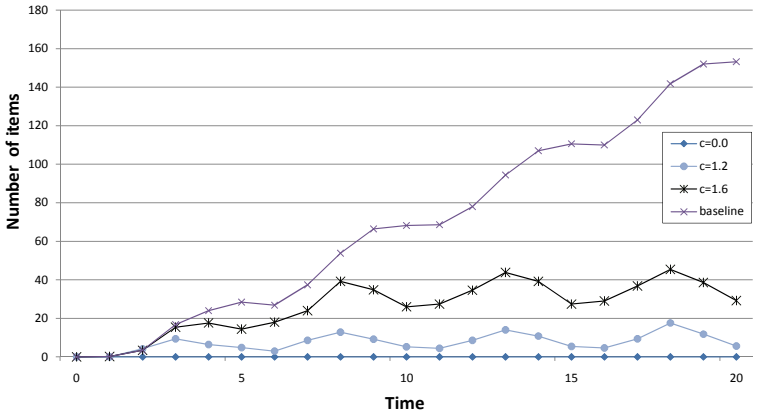


Figure 4.11: The Waiting List Size with respect to Parameter c as defined in Equation 4.17

Table 4.3: Performance Loss Over 50 Days

	$c = 0$	$c = 0.4$	$c = 0.8$	$c = 1.2$	$c = 1.6$
NDCG@3(mean)	12.3%	4.32%	1.03%	0.43%	0.13%
NDCG@3(max)	14.7%	5.12%	1.34%	0.56%	0.50%
P@10(mean)	6.42%	3.37%	0.86%	0.06%	0.03%
P@10(max)	8.42%	3.91%	1.11%	0.24%	0.18%

[Mul06], since it is not practical to do real-time computing and distribute the data over the network in a larger system, instead most of the data is pre-computed and distributed over the system periodically. We took this into account, so during the simulation the algorithm updated the probability (as described in Equation 4.16) of all the items only once a day, at the end of each day.

We found that the best way of monitoring the effectiveness of our algorithm was to keep track of the number of items that were on users' waiting lists, since that would give a good indication of how long users had to wait for an item. Therefore, if we had less items on the waiting list, that meant we had less items that ran out of stock, so users were more satisfied with the service in general.

We plotted the waiting list size over 20 days (Figure 4.11) (the trend after 20 days remained the same) and the corresponding performance metrics NDCG@3 and Precision@10 for 50 days (Figure 4.10 (a) and(b)). We omitted the result of $c = 1.6$ on Figure 4.10, because the performance difference between $c = 1.4$ and $c = 1.6$ was so small that it could not be visualised on the graph. The performance was measured across *all* users who used the system up until the present point (e.g. the latest recommender list provided for each user was taken into account for the score). This ensured that the tendency rather than the daily fluctuation of the system was measured. We tested whether the results were statistically significant using a five-fold cross validation method. As Figure 4.11 shows if we just run the simulation without re-ranking the results (baseline) we would end up with a big waiting list very soon, since the demand for particular items is constant. However, if our constraint (Equation 4.20) is used then we were able to keep the waiting list at bay. Note that we did not include $c = 0.4$ and $c = 0.8$ on Figure 4.11 because they resulted in a waiting list size identical to $c = 0$. In terms of performance if we always

re-ranked the results ($c = 0$) we had a significant performance loss (see Table 4.3) which represents the lower bound of the loss. If we set the threshold higher, the performance improved significantly (Figure 4.10 (a) and (b) and Table 4.3) resulting in a very small performance loss for $c = 1.2$ and $c = 1.6$. As mentioned above the simulation was cross-validated and all the improvements from the lowest c value (including the baseline) were found statistically significant.

As Figure 4.11 shows if we do not control the stock level, the waiting list gets out of control soon. However, we acknowledge that this is an artificial situation as in practice there might be various methods that could help keeping the waiting list in control. For example, the stock level can be increased as the demand grows, but the advantage of our method is that we can find the optimal cost/performance ratio that would help to run a system more efficiently. Besides, the system can respond immediately to demand whereas increasing the stock level might take more time. It is also important to note that this approach could bias the recommender to an extent when recommendation is based only on stock levels, therefore regularisation is needed. One way to regularise the system is to enable the algorithm to increase stock level if the demand for particular items is high. This approach with the combination of the constraint could offer an optimal solution in an ever changing environment.

We also investigated the effect of α (Equation 4.21) on the waiting list size. However, with the current setting where we have more copies of popular items and the stock level is updated daily. We did not expect that enabling to re-rank all the recommended items would reduce the waiting list size, since it enables to recommend items which we have less copies of, therefore, it is more likely that those items would run out of stock earlier. Thus, we slightly modified the experimental setting, so that we only allowed to have one copy of each item (operating with 3900 items). Figure 4.12 depicts the effect of α on the waiting list size. Setting α to zero (all the items recommended for the user are re-ranked) reduced the waiting list size with an average of 10.45% over 50 days. In terms of performance loss we witnessed a fairly big drop that is an average 2.10% for NDCG@3 with the maximum value of 2.99% and an average 5.58% drop for Precision@10 with the maximum value of 7.01%. It is important to emphasise that this drop was expected since we enabled to use items that were not predicted relevant which increased the risk of ranking potentially non-relevant items higher.

The real importance of α also depends on whether the system is able to provide enough relevant items for recommendation. For example, α would not make any difference if all the items that are presented to the user are predicted relevant, which is very likely in practice, because the number of items that form the possible basis of the prediction is the whole set (i.e. all the items in the system), whereas in our case it is just the number of items that can be found in a test set for any given user. Therefore, setting α to less than 1 could only be used useful in practice when the system is designed to present a longer list to the user, where it is likely that the list would contain items that are predicted non-relevant, too.

It might be questionable why one would want to enable ranking items higher when they are predicted non-relevant increasing the risk that users receive non-relevant items, but there are some scenarios when this would be the only way to go. For example, if the user wants an item instantly (e.g. using a video on demand service) and for some reason the demand for some of the items is very high, it is

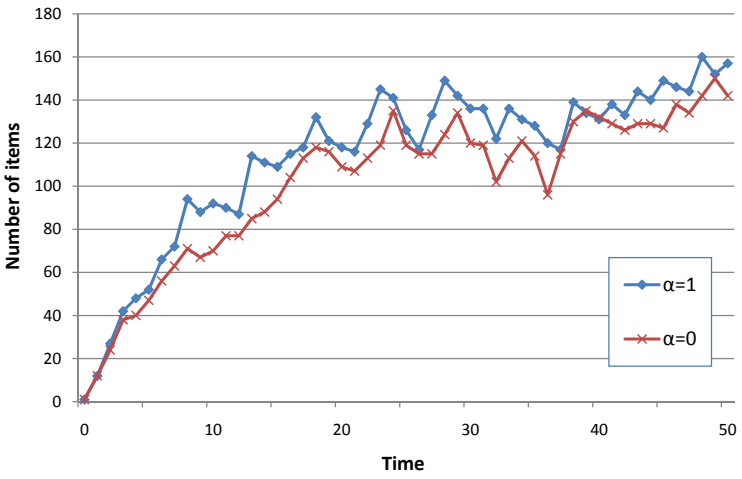


Figure 4.12: The Waiting List Size with respect to Parameter α

crucial to enable re-ranking lower rated items. Otherwise, the system would not be able to recommend any items and the user would scroll down the list and choose from lower rated items anyway. In other cases α could be set dynamically depending on how high the demand is at a given moment.

Despite that, we focused on a very specific scenario in this section, it is important to emphasise that this approach can be applied to any other resource related optimisation problems. For example if a company wants to penalise items that have lower profit margins, $s_{i,u,t}$ (Equation 4.17) can represent the buying price/selling price ratio of a product for a given user and we can set the threshold to penalise items that exceed this ratio.

4.4 Internal Multiple Objectives

In this section, we introduce another approach to multiple objectives. Based on the principles introduced in Section 1.3.3, we aim to define a system that balances between the desired goals of accuracy, diversity, novelty and serendipity. We introduce an approach that attempts to balance and improve all four factors simultaneously with an internal optimisation approach. Using a collection of novel algorithms inspired by principles of 'serendipitous discovery', we demonstrate a method of successfully injecting serendipity, novelty and diversity into recommendations whilst limiting the impact on accuracy.

Instead of using convex optimisation introduced above, this approach applies hybrid rank-interpolation to combine the output of three constituent algorithms in four different combinations: first, it employs solely a new item-based collaborative filtering algorithm based on *Latent Dirichlet Allocation* [BNJ03], the Community-Aware version combines it with a new algorithm that promotes artists with "diverse" listening fan base, the Bubble-Aware version combines the core algorithm with a new Declustering algorithm that identifies and counteracts a user's "music bubble" and the full version is the combination of all three algorithms together.

The approach is evaluated using a comprehensive set of metrics that simultaneously assess accuracy, diversity, novelty and serendipity in user recommendation results. This approach is an attempt to separate

factors that constitute to each goal of the system and measure them separately, in order to investigate how optimising each goal would affect the performance of the system as a whole. We find that the core algorithm produces recommendations that are as accurate as those produced by the state-of-the art *Implicit SVD* algorithm [HKV08], and that both the Community-Aware and Bubble-Aware versions substantially improve all three qualities of diversity, novelty and serendipity with differing trade-offs in accuracy.

4.4.1 Proposed Approaches

Basic Auralist

Basic Auralist is an item-based recommender system that employs *Latent Dirichlet Allocation* as a technique for computing item features. We call this approach *Artist-based LDA*. *LDA* has been used traditionally in topic-modelling, being a fully generative model for document production [BNJ03]. Under this framework, words within a large document set can be clustered into topics based upon co-occurrence, each topic being a probabilistic distribution over word tokens. A "topic composition vector" can then be determined for each document, indicating the estimated level of influence each "topic" would have if the document were to be generated using the *LDA* model. Both topic clustering and document composition can be computed stochastically using the *Gibbs Sampling* algorithm in an unsupervised manner over a training dataset.

This approach applies *Gibbs Sampling* to the unary preferences of the *Last.fm* dataset. Traditionally, recommender systems aim to model user-item relationships simultaneously, here we attempt to separate these relationships and model user-artist relationships as follows. The *Artist-based LDA* model treats artists as documents and preferring users as words, producing a fixed-length topic composition vector for each item. Topics in the Artist-Based model represent *user-communities*, clustering together users with similar preferences. Topic vectors thus represent the distribution of the listener base of an artist, and can be used to characterise them. We define a new *LDA Similarity* metric as the (real-valued) cosine similarity between artist topic vectors:

$$LDASim(i,j) = \frac{\sum_{t \in T} L_{i,t} \times L_{j,t}}{\sqrt{\sum_{t \in T} (L_{i,t})^2} \sqrt{\sum_{t \in T} (L_{j,t})^2}}$$

where $L_{i,t}$, the LDA item-topic matrix, represents the composition proportion assigned to topic t for item i. This similarity metric can be used directly for item-based recommendation: artists can be ranked by the sum-total of their similarity with items in a user's existing history [LSY03]:

$$Basic(u,i) = \sum_{h \in H_u} LDASim(i,h)$$

This produces a ranking list for each user u, with the most similar items awarded the smallest percentage ranks.

Hybrid Versions

To increase the novelty, diversity, and serendipity of the recommendations, we combine the core recommendation with two new algorithms. The first is called *Listener Diversity* and aims to prioritise for

recommendation artists with particularly diverse listener communities, encouraging users to explore beyond a given niche. The second algorithm is called *Declustering* and aims to determine a user's clusters that are within the normal behaviour of the user and then recommend artists outside established cluster groups.

We combine the different algorithms by merging their individual rank outputs. One way of doing so is to produce a hybrid score for each item (artist) [ZMKL05]. Intuitively, the hybrid ranking score of an item i can be taken as a linear interpolation of the percentage [0,1) rank the item has in the output of each of the contributing algorithms. A set of interpolation coefficients λ_a over a set of algorithms A controls the influence of each individual algorithm. In the case of the generalised *Full Auralist* recommender, we have three λ coefficients governing an algorithm set A that includes *Artist-based LDA*, *Listener Diversity* and *Declustering*.

$$Hybrid(u,i) = \sum_{a \in A} \lambda_a (rank_{a,u,i})$$

The final recommendation output consists of the item list sorted by the hybrid rank score. The combination of these algorithms allows an accuracy-focused *Basic Auralist* to be combined with small proportions of diversity or serendipity promoting algorithms, in order to improve the overall balance of qualities.

Community-Aware Auralist

Community-Aware recommendation introduces the *Listener Diversity* metric for artists, which is used to produce a ranked list of the most diverse artists. This list is blended with the core algorithm to promote more diverse artists in recommendation.

We recall that for the *Artist LDA* model is formed are formed over a group of users that can represent the artist. Such a representation offers us a unique perspective on the demographics of listeners, not visible when observing the raw vector of preferences. Certain artists, whilst popular in their own right, might have a listener base concentrated in only a few user communities, whereas the listeners of another artist might be more widely distributed.

Given that a *LDA* topic vector is a probability distribution summing to 1, we use the entropy of such a distribution to measure uniformity. A distribution focused on only a few outcomes will score a less negative entropy; a more evenly and widely distributed event will produce a greater negative entropy. We thus introduce the *Listener Diversity* of an artist i as the entropy over its topic distribution:

$$Listener\ Diversity(i) = -\sum_{t \in T} L_{i,t} \log_2 (L_{i,t})$$

This approach would emphasise users that can appeal to different user community, thus the diversity of these users is measured through their user base. These users might be ideal candidates to be offered in the context of serendipitous recommendation where we seek to expand a user's music taste beyond that of his comfort zone. A strategy for this would be to highlight more diverse artists that include user established music communities, but also introduce elements that the user may be unfamiliar with. This balance can be achieved by interpolating the output of a *Listener Diversity*-sorted list with that of a

conventional accuracy-focused algorithm, as follows:

$$Community(u,i) = (1-\lambda) rank_{Basic,u,i} + \lambda rank_{Diversity,i}$$

Analysis of *Listener Diversity*'s relationship with other factors shows that *Listener Diversity* tends to bias towards globally popular artists. This should be unsurprising; as such artists might gain more exposure and attract a naturally wider fan base. This is compensated by discounting an artist's original *Listener Diversity* with a popularity-diversity regression function, highlighting artists that are diverse for their popularity level (popularity being the number C_i of the artist's unique listeners). The resulting adjusted *Listener Diversity* is:

$$Listener\ Diversity'(i) = \textit{Listener Diversity}(i) - \textit{Offset}_{pop}(i) \tag{4.23}$$

Bubble-Aware Auralist

As a counterpart to *Listener Diversity*, we introduce a graph-based algorithm termed *Declustering*. *Declustering* produces a ranked list of the least "clustered" "clustered items for a user and is interpolated with core algorithm:

$$Bubble(u,i) = (1-\lambda) rank_{Basic,u,i} + \lambda rank_{Declustering,u,i}$$

We compute *Declustering* scores over what we call the "Artist Graph". Formally, this is a graph $G = (N, E)$ where each node $i \in N$ is an artist and edges $(i, j, weight) \in E$ are drawn between artists that have non-zero similarity, according to a similarity metric $weight = sim(i, j)$. The similarity here is defined as the cosine similarly between artist topic vectors.

The *Declustering* algorithm aims to identify nodes that lie on the edge of clusters in a user's graph, in order to discredit artists with higher activity with regard to the target user, whilst still maintaining overall similarity. In this way we hope to help users expand their music taste, literally pushing the boundaries of the region their behaviour occupies in the feature-space. This recommendation strategy is motivated by concepts of social network theory such as clustering and brokerage [EK10] and has been previously used by Celma *et al.* [CH08, CC08] to investigate the long-tail properties of music recommendation in terms of network links.

We found that the *Last.fm* dataset has a power-law distribution of node degrees, suggesting it may have "small-world" properties [WS98]. This implies a graph structure similar to that of a social network, with nodes being clustered around a series of high-degree hubs. Therefore, we employ a metric commonly used in social-network analysis to measure how clustered nodes in a network are. We selected this metric for its simplicity and low computational complexity. The clustering coefficient of a node i is defined as:

$$Clustering(i) = \frac{2 \times |\{(j,k) \in E_u | j,k \in \text{neighbours}(i)\}|}{|\text{neighbours}(i)| \times (|\text{neighbours}(i)| - 1)} \tag{4.24}$$

where *neighbours(i)* is the set of nodes that are neighbours of item i in the local preference graph. The clustering coefficient of a node measures the proportion of possible interconnections that exist amongst neighbours of a node. A node with a high clustering coefficient is surrounded by tightly interconnected

nodes whereas a node with a lower clustering coefficient might have neighbours split between multiple clusters or lie on the edge of an existing cluster.

The *Declustering* algorithm considers in turn all the prospective recommendations for a user. Following the approach used in item-based recommender systems [SKKR01], the algorithm temporarily adds each possible item to the graph and computes the clustering coefficient of that node with respect to the existing elements of the user's local preference graph. The output of the algorithm is an ordered list of the artists positioned outside of clusters in the candidate set, which can be interpolated with a conventional recommender to apply counter-clustering flavour to recommendation items.

4.4.2 Evaluation

As discussed in Section 2.3, recommender systems are often evaluated using conventional error measures such as RMSE or MAE. Here we aim to explore alternative ways of evaluating the system in order to gain more understanding of certain, undiscovered aspects the recommender algorithm in question. Therefore, we introduce a number of additional performance metrics to capture the quality of diversity, novelty and serendipity in recommender systems, comparing them with two conventional rank based performance metrics - average $Rank$ (which measures the average percentage rank of withheld items in the user's history) and Top-20 $Recall$ [MRS08].

Diversity

Diversity in recommender systems refers to how different the recommended objects are with respect to each other. Contrary to popular belief, recommender systems by itself would not improve diversity, as recommender systems naturally bias towards popular items [FH09]. This goes against general tendency of human nature that we enjoy and require diversity in music, depending a number of factors that are unique for each individual. In fact, it has been shown that users will actively choose less-preferred items in an effort to improve the variety of consumption [RKK99, ZMKL05]. We measure diversity through the *Intra-List Similarity* metric introduced by Ziegler *et al.* [ZMKL05], using (binary) cosine similarity to judge the similarity between items.

$$\overline{Intra\text{-}List\,Similarity} = \frac{1}{|S|}\sum_{u\in S}\sum_{i,j\in \mathrm{R}_{u,20}} CosSim(i,j) \tag{4.25}$$

$Intra\text{-}List\ Similarity$ essentially sums the pairwise similarity of all items in a set (simplified in our case due to a symmetric similarity measure). A recommendation list with groups of very similar items will score a high intra-list similarity compared to a list that has more dispersed and diverse recommendations.

Novelty

Novelty can be seen as the ability of a recommender to introduce users to items that they have not previously experienced before in real life. By definition, this is problematic in terms of evaluation as the aim is to capture what users would find interesting that is not in their profile, but still relevant to them. In other words, a recommendation that is accurate but not novel will include items that the user enjoys, but already knows of. A limited proportion of such recommendations has been shown [SS01]

to have a positive, trust-building impact on user satisfaction, but it can also be seen that to be useful a recommender needs to suggest previously unknown items. We measure novelty with a metric previously introduced by Zhuo and Kuscik [ZKL+10]:

$$\overline{Novelty} = \frac{1}{|S|}\sum_{u \in S}\sum_{i \in \mathrm{R}_{u,20}} \frac{1}{20 \log_2 pop_i} \tag{4.26}$$

where pop_i measures the global popularity of an item (fraction of all preferences with respect to the preferences expressed towards the target item). This novelty metric quantifies the average information content of recommendation events – higher values mean that more globally unknown items are being recommended. Given the assumption that the likelihood a user has experienced an item is proportional to its global popularity, this serves an approximation of true novelty.

Serendipity

Serendipity represents the unusualness or surprise of recommendations. Unlike novelty, serendipity encompasses the semantic content of items, and can be imagined as the distance between recommended items and their expected contents. A serendipitous system will challenge users to expand their tastes and provide more interesting recommendations, qualities that can help improve recommendation satisfaction [SS01]. We assess serendipity through a new *Unserendipity* metric, which uses *CosSim* to measure the average similarity between items in a user's history H_u and new recommendations. Lower values indicate that recommendations deviate from a user's traditional behaviour, and hence are more surprising:

$$\overline{Unserendipity} = \sum_{u \in S} \frac{1}{|S||H_u|} \sum_{h \in H_u} \sum_{i \in R_{u,20}} \frac{CosSim(i,h)}{20} \tag{4.27}$$

4.4.3 Experiments

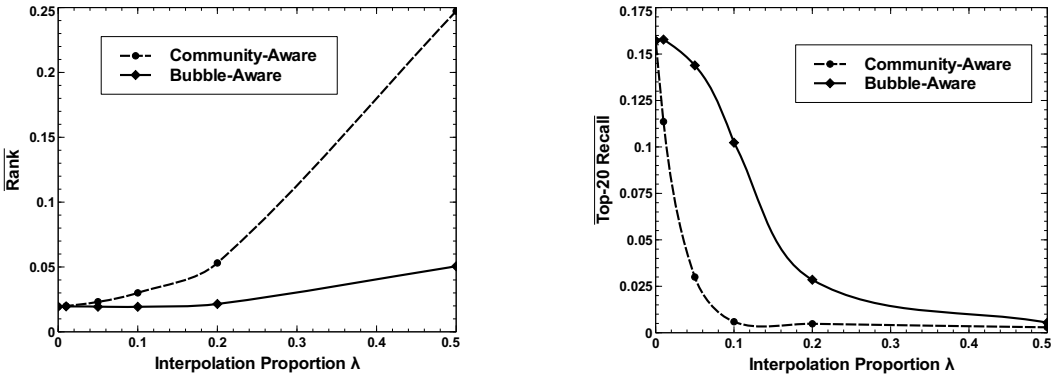
Basic Auralist Recommendation

We evaluate the effectiveness of *Artist-LDA* recommendation method against the state-of-the art *Implicit SVD* method introduced by Hu, Koren and Valinsky [HKV08]. This model enables us to inject a confidence level along with each observation. We thus incorporate the implicit artist play-count as a confidence weight in the matrix factorisation cost function. Metrics are computed over random subsamples of 35k users; larger samples only marginally improve performance. 20% of each user's preferences were randomly withheld as a training sample. An improvement of this evaluation method would be to separate the training and test set based on a cut-off time (rather than randomly), so that the test set would be positioned after the training set in the time dimension. In addition, we bootstrap the *LDA* topic training step with the full 360k user dataset, in order to improve the performance of the model (since Gibbs Sampling runs relatively quickly even on large user samples, compared to other model-based techniques).

Our experimental results are reported in Table 4.4 and show that *Basic Auralist* produces the most overall accurate rankings for user histories ($\overline{Rank} = 0.0194$) whilst *Implicit SVD* produces the highest $\overline{\textit{Top-20 Recall}}$ scores (0.174). Both algorithms score comparatively in diversity ($\overline{Intra\text{-}List\ Similarity}$), whereas *Implicit SVD* has improved serendipity and *Basic Auralist* has slightly improved novelty.

	Rank	Top-20 Recall	Intra-list Similarity	Novelty	Unserendipity
Basic Auralist	**0.019** ±0.0004	0.157 ±0.004	**14.4** ±0.2	**11.8** ±0.06	0.060 ±0.0004
Implicit SVD	0.039 ± 0.0008	**0.174** ±0.002	14.7 ±0.1	10.9 ±0.03	**0.046** ±0.0002
Community-aware(λ=0.05)	0.023 ±0.02	0.030 ±0.0009	3.4 ±0.06	17.2 ±0.1	0.047 ±0.0003
Bubble-aware(λ=0.2)	0.021 ±0.0002	0.029 ±0.0006	3.4 ±0.05	14.2 ±0.1	0.035 ±0.0002
Full Auralist	0.025	0.008	1.54	17.3	0.039

Table 4.4: Performance Results for *Basic Auralist*, the State-of-the-Art *Implicit SVD*, and *Full Auralist*



(a) $\overline{Rank}$ accuracy

(b) $\overline{Top\text{-}20\ Recall}$ accuracy

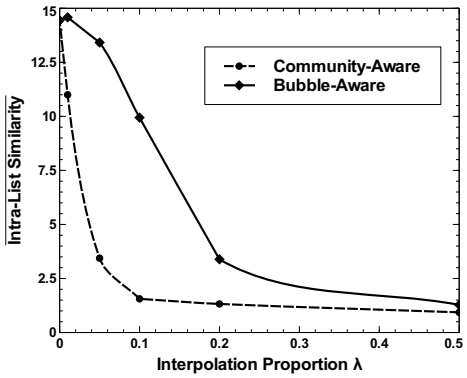
Figure 4.13: Accuracy Performance of *Community-Aware Auralist* and *Bubble-Aware Auralist*

The combination of accuracy scores seems to indicate that *Implicit SVD* does a better job of including items in the *Top-20* list. However, it may be argued that for the use-case of *serendipitous* recommendation, a high recall is not necessary; recall indicates that similar, already known items are being placed in the *Top-20* list, displacing the recommendation of novel items. Interestingly, of the items *Implicit SVD* does recommend, the registered $\overline{Unserendipity}$ is somewhat lower, implying that the generalisation of matrix factorisation does result in some less obvious recommendations as well. We exceed this serendipity value with later versions of *Auralist*.

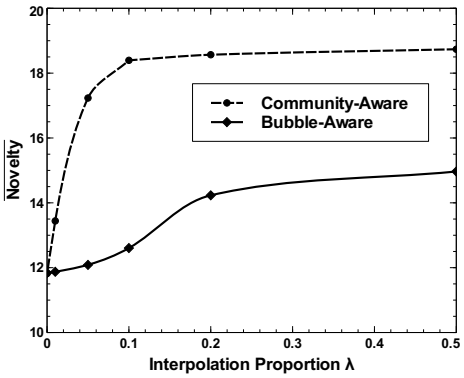
Hybrid Versions of Auralist

Figure 4.13 and 4.14 show the performance results for *Community-Aware* and *Bubble-Aware Auralist*. Table 4.4 also includes their performance at points of interest along the λ curve (note at λ=0, both algorithms reduce to *Basic Auralist*). Given that both hybrid versions of Auralist attempt to bias towards serendipitous recommendations at the expense of more "easily accurate" items, it should be unsurprising that both exhibit an accuracy-serendipity trade-off. More interestingly, both methods increase novelty and diversity, and do so at different rates.

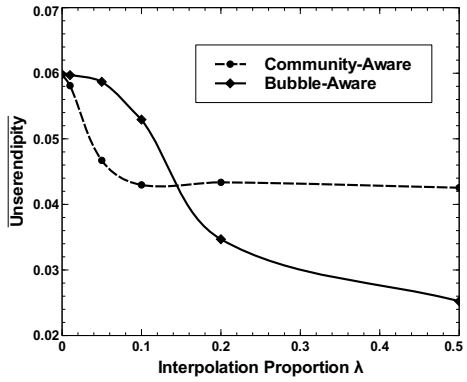
As the *Listener Diversity* interpolation increases, *Community-Aware Auralist*'s rapid improvements in non-accuracy scores (Figures 4.14(a), 4.14(b), 4.14(c)) are tracked by decays in recall (Figure 4.13(b)) and to a lesser extent $\overline{Rank}$ (Figure 4.13(a)), tailing off at higher proportions. *Community-Aware Auralist* hence represents a direct trade-off between accuracy and non-accuracy performance, with the most



(a) Diversity



(b) Novelty



(c) Serendipity

Figure 4.14: Diversity, Novelty, and Serendipity Performance of *Community-Aware Auralist* and *Bubble-Aware Auralist*

activity occurring in the $0 < \lambda < 0.05$ range of Figures 4.13 and 4.14. Compared with the other graphs, *Community-Aware Auralist* maintains a consistently sizable lead over *Bubble-Aware Auralist* in terms of novelty (Figure 4.14(b)), likely due to the popularity correction Offset_{pop} we introduced in Section 4.4.1.

As with *Community-Aware Auralist*, the *Bubble-Aware Auralist*'s performance curves for serendipity, novelty and diversity track that of $\overline{Top\text{-}20\ Recall}$. Unlike *Community-Aware Auralist*, however, *Bubble-Aware Auralist*'s $\overline{Rank}$ decays at a much slower rate, and the performance curves possess sigmoid-like qualities, experiencing the greatest rate of change after about $\lambda = 0.1$ and diminishing returns afterwards. We propose that this is the point when the *Declustering* algorithm is able to successfully overcome the bias towards recommendations embedded within preference clusters and is able to successfully recommend cluster-bordering items. The nature of the $\overline{Rank}$ curve indicates that the bulk of this benefit can be achieved without an overwhelming effect on the overall accuracy, suggesting that *Bubble-Aware Auralist* may be able to supply a good balance between serendipity, diversity and novelty against accuracy.

Bubble-Aware Auralist manages to surpass *Community-Aware Auralist* in terms of serendipity relatively quickly ($\lambda \sim 0.15$), continuing to improve even after *Community-Aware Auralist*'s performance begins to plateau. This suggests that the serendipity improvement is not merely incidental (i.e., from

declining accuracy), and is actively being promoted by *Declustering*.

To sum up, these findings indicate that *Community-Aware Auralist* is best used at smaller interpolations (0–0.05) as a roughly even trade between accuracy and non-accuracy qualities and as a broad stroke in changing the focus of a recommender. They also suggest that using *Bubble-Aware Auralist* during the peak rate of change (Figure 4.13(a)) can improve non-accuracy qualities at very little cost. At $\lambda = 0.2$, a mere 0.7% increase in average history rank is accompanied by a 77% decrease in $\overline{Intra\text{-}List\ Similarity}$, 20% increase in novelty and a 42% decrease in measured unserendipity. Overall, both methods prove to be able to improve diversity, novelty and serendipity.

4.5 Conclusion

In this section, we defined a new way of approaching multiple goals in recommender systems and investigated through a series of examples how to balance between satisfying the goals and other (general) accuracy measures. We presented three main cases above, as follows

1. In the first part of the chapter, we presented a preliminary study on cost based external objectives through an example of digital content delivery using recommender systems. This approach described a system where the objectives were directly connected to the performance of the system. As it was outlined above this is essentially a two-step prediction problem. The optimal solution of the problem not only comes from minimising the error on both of the problems but it is also a trade-off between increasing the number of items cached and reducing the risk of getting the recommendation incorrect, which means minimising the error on items that are ranked lower for the user. We defined this problem as a special case of multiple objectives where the additional objective of the system is proportional to the main performance of the system.

2. In the second part of this chapter, we introduced an external convex optimisation framework to incorporate multiple goals into the recommender system. In this case, the objectives were opposing to each other, where an optimal balance had to be found in order to satisfy the objectives to a certain extent. We studied this problem through two distinct use cases. We investigated how demand affects the availability of items in an online shopping environment and offered a solution that can help to fine-tune the trade-off between cost and performance. We also studied the relationship between items in the long tail (i.e. less popular items) and improving the diversity of the recommended list, an idea that was further expended in the last part of the chapter (Section 4.4).

   Areas that are worth further investigation are the following. It is important to understand how parameter β (Equation 4.14) affects the quality of recommendation. This would provide a more precise interpretation of the relationship between the offset of users' taste from the mainstream and items in the long tail. Furthermore, it is of interest to identify and generalise more features that can be used to describe items in the long tail, for example items that can be potentially interesting for a small group of people or items that are new to the system. To gain more understanding of the framework, a real-life evaluation is needed to test the performance in a constantly changing environment.

3. In the last part, we introduced a framework where we aimed to improve diverse, novel and serendipitous recommendations at the same time, at a slight cost to accuracy, using an internal optimisation approach. This combined the effect of goals which might *complement* each other - in this case they all aimed to increase user experience. We described a series of metrics designed to assess both accuracy and the three additional qualities of diversity, novelty and serendipity. This approach is fundamentally different from the convex optimisation approach as the additional goals are investigated thought interpolation where we made an assumption that the specialised algorithms (*Community-Aware Auralist* and the *Bubble-Aware Auralist*) would enhance certain objectives (in this case diversity, novelty and serendipity) and interpolation would find the best balance between generalised and specialised accuracy.

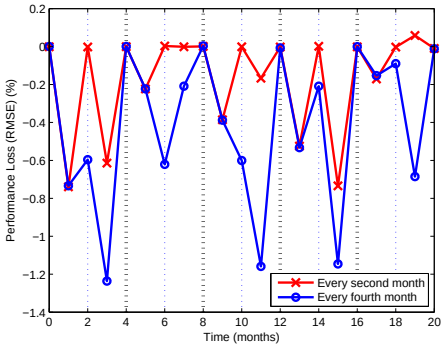
Chapter 5

Using Control Theory for Stable and Efficient Recommender Systems

In the previous chapter, we focused on user and system centred goals that were solved either internally within the recommender system or outside the algorithm, treating recommender systems as a black box. In this chapter, we take the orthogonal dimension of is these goals and add the time dimension for user and system focused goals that are solved externally. Modelling a system over time requires to describe the dynamics as a combination of the underlying recommender model and the its users' behaviour. We propose to solve this problem by applying the principles of modern control theory to construct and maintain a stable and robust recommender system for dynamically evolving environments. In particular, we introduce a design principle by focusing on the dynamic relationship between the recommender system's performance and the number of new training samples the system requires. This enables us to automate the control over factors such as the system's update frequency. The approach illustrates a user-centric objective that concentrates on stable performance and a system-centric goal that deals with resource management.

5.1 Problem Statement

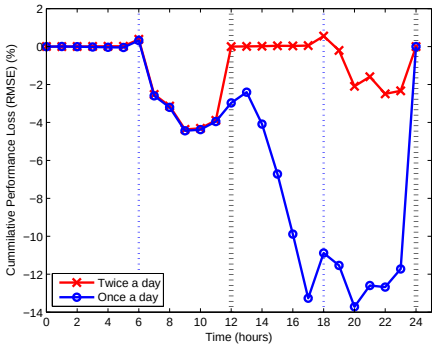
In the research literature, CF algorithms are evaluated in a relatively *static* context: datasets are examined using traditional machine learning methodologies to produce cross-validated results, by randomly splitting the data into training and test sets. In practice, these systems will experience a continuous influx of new ratings: the deployment scenario is dynamic and continuously subject to change [Kle06]. Recent work has delved into this domain and examined the differences that emerge between the two contexts [Bur10, Lat10]. In particular, CF algorithms produce up-to-date recommendations by being regularly re-trained (e.g., daily) in order to incorporate new ratings into the model of user preferences. Given that state-of-the-art CF algorithms are, in general, expensive to update, because of the limited resources available as well as the computation required to train the model. Thus, system developers have to make a critical decision: they must decide *when* and *how frequently* to update their systems. In doing so, they must balance between the benefit of a recently re-trained algorithm (which will produce recommendations based on the latest user ratings) and the cost that the business incurs from re-training. This problem



(a)

Movielens 1m.

The loss is calculated against a baseline of training every month.



(b)

Yahoo! Music Dataset - KDD Cup'11.

The loss is calculated against a baseline of training every six hours.

Figure 5.1: The Effect on Performance by Reducing the Update Time of the System

is often solved by iteratively re-training at a pre-defined interval [Mul06].

The point is further illustrated in Figure 5.1 (a) and (b). The figure depicts the performance dynamics of the Movielens 1m dataset [Gro06] (described in Section 4.3.2) for a period of 20 months and the Yahoo! Music Dataset (KDD Cup'11) [DKKW12] for a period of one day, the day was randomly picked from one of the busiest days in the dataset with around 200 thousand new ratings entering the system. Figure 5.1 (a) shows that training only every fourth month results in a substantial performance loss compared to training every month. We can clarify that similar pattern emerges on Figure 5.1 (b) which shows that training only once a day (at midnight) results in a substantial performance loss compared to training three times a day. This loss is more pronounced during the busiest times (afternoon hours). In addition, as new users enter the system, these users cannot receive reliable recommendations until the system is re-trained on the new data points. This further deteriorates the performance [Ahn08] and subsequently the system may not be able to provide accurate recommendations for those new users. This trend can be observed between *any* time interval as long as new data enters the system, because the additional information would more likely to help improving the performance.

In this work, we propose a control-theoretic approach to designing recommender systems. We propose a new methodology on how to design and manage a complex recommender system in order to maintain the trade-off between the effectiveness and efficiency of the system by applying modern control theory [Oga01]. We demonstrate that modern control theory has the potential to provide the required mathematical tools to both analyse and model the stability and robustness (resistance to error and disturbance) of a recommender system over time. We validate our approach by examining the dynamic relationship between the recommender system's performance and the number of new training samples the system requires. We show that control theory would allow us to design the recommender system and its feedback loop to effectively mitigate the effects of any forces (such as noise in the data) that may arise during operation that would otherwise negatively affect the recommender system's performance over time. We further discuss how, after defining the system dynamics this way, the control loop can be replaced with other signals, such as controlling the computational effort over time and the system's up-

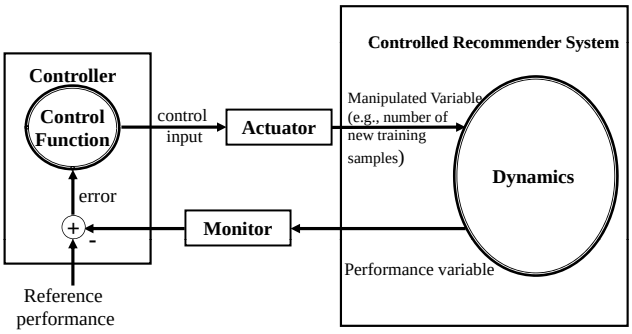


Figure 5.2: The Closed-Loop Control of a Dynamical Recommender System

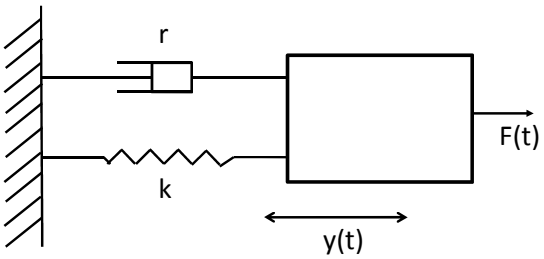


Figure 5.3: A Mass Attached to a Spring and Damper: $y(t)$ is the displacement. The damping coefficient is represented by r in this case, where the spring constant is k. $F(t)$ in the diagram denotes an external force. In our case, $y(t)$ is the performance value while $F(t) = s(t)$ number of new training samples.

date frequency. We believe that, by grounding the operation of recommender systems in control theory, this work not only contributes to the growing body of research addressing temporal dynamics in recommender systems, but will also be of significant interests to the wider research areas such as dynamic information retrieval and filtering.

5.2 A Feedback Control Approach

In this section, we present an adaptive temporal recommendation architecture derived from control theory. As we showed in Figure 5.1, recommendation performance varies according to how often its underlying model is trained, while also being heavily influenced by any external disturbances or changes in the dynamics of the process (thus the fluctuation in the performance). On one hand, if we stop training the model, the performance will decrease; on the other hand, training the model each time new data is received will be computationally expensive. It is thus of great interest to investigate techniques that can dynamically control and maintain the required performance level in order to make full use of the computational resources available, by balancing the trade-off between the effectiveness and efficiency. The proposed mechanism uses the principle of *feedback* to make the output of the system's dynamic process follow a desired reference value, regardless of any external disturbances or changes in the dynamics of the process. The key feature of this architecture is a feedback control loop (illustrated in Figure 5.2) where the *control function* generates an input of the dynamical system that is adequate to maintain per-

formance, this input signal is realised and entered into the system by the *actuator*. Then, the output of the dynamical system is *monitored* and fed back to the controller in order to compare it to the reference value, compute the error and generate the next control input.

We describe the general idea of feedback control by comparing a recommender system to a simple physical system: a weight attached to a wall with a spring and damper, as illustrated in Figure 5.3. This simple system has one input, $F(t)$, which represents how much force has been applied to pull the weight from the wall. The output, $y(t)$, is how much the weight will be displaced from its the current position. If we try to keep the weight from the wall at a certain (stable) distance, we need to balance a number of factors. While there is an inherent relation between the input and output, the amount of movement over time will also be affected by the strengths of the spring and damper, which both exert different forces on the system. In effect, we have a system where the relation between an input and output value is dependent not only on the input itself, but a number of other implicit factors that are built into the system. A recommender system follows a similar pattern: the output (performance) relates to the input (training data) along with a variety of hidden factors in the system. In order to control the output, we, therefore, need to approximate this function: we do so by observing the system over time. The system keeps monitoring the performance and feeds the estimated performance measure back to a controller. The controller then reacts accordingly by modifying the input in order to maintain a particular stable outcome. Note that the motivation of this approach is to maintain instead of optimise the system. It is suggested in [JW10b] that there are multiple objectives that a recommender system needs to fulfil; some of them might directly contradict with the performance of the system. This approach would enable us to directly define and control any aspect of a live system, given that the input-output relationship of the dynamical system can be defined.

In the following sections, we formalise how this metaphor of a controller can be actualised and implemented in a recommender system. We take two distinct steps in designing this system. First, we consider the recommendation accuracy as a stochastic yet controllable variable, and develop a mathematical description of the underlying dynamic process that can be controlled. Such a description allows practitioners to analyse their system with a wide range of analytical techniques that are available from control theory. Then, we use these techniques to define and estimate the best controller strategy to generate the control of the system. We describe the two components in the following subsections.

5.2.1 Modelling Performance Dynamics

We denote the performance of a recommender system as $y(t)$, where t represents time. Without loss of generality, we assume lower values of $y(t)$ indicate better performance. We assume that performance can be observed directly by monitoring user feedback and comparing it to the prediction provided by the model (see Section 5.3). The number of *new* training samples is denoted as $s(t)$, and we consider this to be an input signal to the dynamical system.

A top down approach to model a system's temporal dynamics is to consider the physical laws that govern the system. For instance, Newton's second law of motion says that an object with mass m subject to a force $F(t)$ undergoes an acceleration $v'(t)$ that has the same direction as the force, and its magnitude

is directly proportional to the force and inversely proportional to the mass, i.e., $F(t) = mv'(t)$. By taking the analogy, we consider that the change rate of the performance is proportional to the number of new training samples entering the system ($y'(t) \approx s(t)$). To approximate this, we assume that the number of *new* training samples is also proportional to the current performance of the system ($y(t) \approx s(t)$). This can also be interpreted as the need to have a certain number of new training samples to maintain the performance of the system. As a result, the dynamics of the performance can be approximated by the following simple differential equation:

$$ry'(t) + ky(t) \approx s(t) \tag{5.1}$$

where $y'(t)$ (change rate of the system performance) is the derivative of $y(t)$. It is interesting to see that the above equation is similar to a mechanical system illustrated in Figure 5.3. $ky(t) \approx s(t)$ is the analogy to Hooke's law of elasticity that the extension of a spring is in direct proportion with the load applied to it, while $ry'(t) \approx s(t)$ follows a damping force. We thus have two parameters r and k, which need to be estimated from the performance dynamics of the recommender system. To estimate them, we rewrite the dynamical model in a form of discrete time:

$$y(t+1) = \frac{r}{r + \Delta t k} y(t) + \frac{\Delta t}{r + \Delta t k} s(t) + \varepsilon(t) \tag{5.2}$$

where Δt denotes the unit of time interval. In addition, we tackle the uncertainty in the system by adding random disturbance $\varepsilon(t)$ at time t. Equation (5.2) is in fact a linear regression system. For simplicity, this model can also be written in matrix notation when drawing the observation over a time period $t \in [0, T]$:

$$\mathbf{y} = \mathbf{Y}^T \theta + \varepsilon \tag{5.3}$$

where $\mathbf{y} = (y(1), ..., y(t), ..., y(T))^T$, $\theta = \left(\frac{r}{r+k\Delta t}, \frac{\Delta t}{r+k\Delta t}\right)^T$, and

$$\mathbf{Y} = \begin{pmatrix} y(0) & ... & y(t) & ... & y(T-1) \\ s(0) & ... & s(t) & ... & s(T-1) \end{pmatrix}$$

Linear systems have been well studied in many research fields. One of the standard solutions to obtain the parameters is to employ the Maximum Likelihood (ML) estimation from the observation over a time period $t \in [0, T]$. We assume that the error ε has a multivariate normal distribution with mean 0 and variance matrix $\delta^2\mathbf{I}$, where $\mathbf{I}$ is a column vector of size T and its elements are all ones. Then the log-likelihood function of the parameters is

$$\mathbf{L}(\theta, \delta^2 | \mathbf{Y}) = \ln\left(\frac{1}{(2\pi)^{T/2} |\delta^{\mathbf{I}}|^{1/2}} e^{-\frac{(\mathbf{x} - \mathbf{Y}^T\theta)^T (\mathbf{x} - \mathbf{Y}^T\theta))}{2\delta^{2\mathbf{I}}}}\right) \tag{5.4}$$

Differentiating this expression with respect to θ we find the ML estimates of the parameter θ, thus k and r:

$$\left(\frac{\hat{r}}{\hat{r} + \hat{k}}, \frac{1}{\hat{r} + \hat{k}}\right) = \underset{\theta}{\operatorname{argmax}}\, \mathbf{L}(\theta, \delta^2 | \mathbf{Y}) = (\mathbf{Y}\mathbf{Y}^T)^{-1}\mathbf{Y}\mathbf{y} \tag{5.5}$$

where we set $\Delta t \equiv 1$ to sample the data per day. A problem still remains: we need to choose input $s(t)$ to increase the accuracy and robustness of the parameter estimation. We introduce a flexible way to add

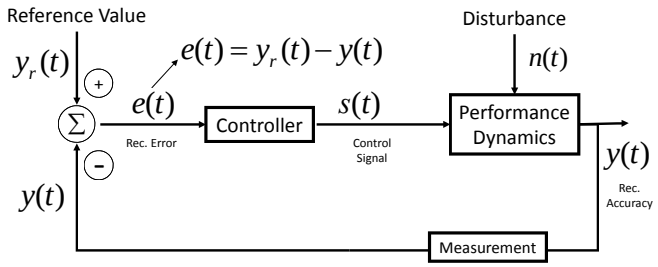


Figure 5.4: The Time Domain Block Diagram

disturbance by using the log-normal random walk model [BDM01, Ein56]. We estimate the mean (μ) and standard deviation (σ) from the input data and simulate an additional time series that has the same mean and variance. Let us define $s(t)$ as the time series of the input values over a predefined time period $[0, T]$. The modified input $s(t)$ becomes as follows:

$$ds(t) = \mu s(t)dt + \sigma s(t)dW(t) \tag{5.6}$$

where $W(t)$ is the Wiener process

$$\text{dW}(t) = \varepsilon\sqrt{dt} \text{ and } \xi \text{ is the noise } N(0, 1) \tag{5.7}$$

Essentially, this approach randomises the time series depending on the standard deviation of the intended input. The advantage of this approach is that setting σ to zero we get the exponential curve that represents the natural way of data growth [Lat10].

In summary, we use the Maximum Likelihood estimation (Equation 5.5) and the lognormal random walk (generating the input) to obtain the parameters r and k for a given time series that represents the dynamics between the input and the output of the system. The detailed experiments to estimate parameter r and k can be found in Section 5.3.1.

5.2.2 Feedback Controller

The next question is how to design the input signal $s(t)$ so that the performance $y(t)$ will stay as stable as possible. That is if $y(t)$ deteriorates from the desired value, how do we change input $s(t)$ to quickly alter the systems dynamics and bring $y(t)$ back to the desired value? This is done by constructing a closed loop system. In a closed loop system, the output is fed back and compared with a reference value $y_r(t)$. The error signal, denoted as $e(t) = y_r(t) - y(t)$, will be sent to a controller (the process is illustrated in Figure 5.4). Upon receiving the error feedback signal, the controller then calculates how much modification is needed for $s(t)$ at this moment. In other words, the feedback on how the system is actually performing allows the controller to dynamically compensate for disturbances to the system and produce a response in the system that perfectly matches the user's wishes. Our discussion here is limited to the situation where the state of the system, in this case the recommendation performance, can be observed. In practice, this is possible as we can measure the recommendation performance periodically by looking at users' feedback or construct a validation set over time. The approach is further explained

in Section 5.3. the PID (Proportional-Integral-Derivative) controller [ACL05]. It not only captures the linear relationship between the error signal and input $s(t)$, but also covers both the derivative and the integral of the error signal. Formally, we have:

$$s(t) \equiv Ce(t) + B\sum_{0}^{t} e(t)\Delta t + D\frac{e(t) - e(t-1)}{\Delta t} \tag{5.8}$$

where the nonlinear relationship is represented by three parameters. The signal ($s(t)$) that the controller produces is the combination of the *proportional* gain (denoted as C) times the magnitude of the error, the *integral* gain (denoted as B) times the integral of the error and the *derivative* gain (denoted as D) times the derivative of the error. The conversion of the integration from continuous to discrete time is done by the *backward Euler* method [Oga87]. The PID controller is the combination of the three basic types of controllers, each adding an extra layer to the system. The proportional controller makes a change to the output that is proportional to the current error value. The integral term accelerates the movement of the process towards the set-point and eliminates the residual error that occurs with a pure proportional controller. The derivative term slows the rate of change of the controller output. In practice, not all elements are needed. For example, in some cases, one can find that D is not required and set to zero. In a nutshell, the above equation feedbacks the error and uses it as the outer force to change the dynamics of the performance. As explained in the previous section we rewrite this dynamic equation in a form of discrete time with uniform sampling ($\Delta t \equiv 1$), combining it with Equation 5.1 gives:

$$\begin{aligned} &\hat{r}\Big(y(t) - y(t-1)\Big) + \hat{k}(y(t)) \\ \approx &Ce(t) + B\sum_{0}^{t} e(t) + D\Big(e(t) - e(t-1)\Big) \end{aligned} \tag{5.9}$$

where parameter $\hat{r}$ and $\hat{k}$ are obtained using the Maximum Likelihood estimation introduced in Equation 5.5. Tuning the three parameters (C, B and D) offline is needed in order to guarantee the desired performance. In control theory, the system is usually modelled by transforming the discrete time signal into the frequency domain using the z-transform. A popular technique to obtain the controller's parameters in z domain is called the Ziegler-Nichols method. In this work, we adopt this method along with a number of software tools and manual tuning. We refer to [Oga01] for the detailed z domain analysis and the underlying mechanism to obtain the three parameters while we primary focus on the design pattern of the system.

5.2.3 Feedback Control versus Relevance Feedback

An interesting discussion is to link the proposed feedback control of recommender systems to the relevance feedback techniques in information retrieval [Roc71, RL03, UY06], despite the fact that their motivations are different. Relevance feedback is used as a technique to improve retrieval performance (e.g., Mean Average Precision) by "correcting" the retrieval model when new relevance judgements come. Its use is quite often limited to one or two iterations, partially because it is practically difficult to obtain a user's feedback over many iterations. Furthermore, in relevance feedback, it is also hard to control the feedback, as there is no established way to detect when the equilibrium (competing influences are balanced) is reached and how to reach it.

$$e(t) = y_r(t) - y(t), \quad s(t) = Ce(t)$$
$$y(t) = F(s(t) + n(t))b(t) = H(y(t)) \tag{5.10}$$

where $n(t)$ denotes a noisy signal injected into the system as illustrated in Figure 5.4. C and F are the parameters for the controller and the system, respectively. Combining those gives:

$$\begin{aligned} y(t) =& F\Big(C\big(y_r(t) - y(t)\big) + n(t)\Big) \\ =& FCy_r(t) - FCy(t) + F(t)n(t) \end{aligned} \tag{5.11}$$

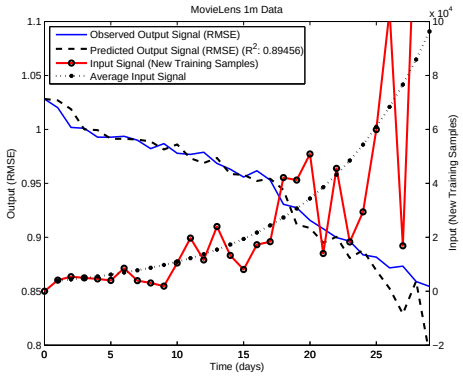
Finally, we obtain the ratio between the IR system performance $y(t)$ and its pre-defined reference $y_r(t)$ as

$$\frac{y(t)}{y_r(t)} = \frac{CFy_r(t) + Fn(t)}{(1 + FC)} = \frac{1}{1/FC + 1} + \frac{1/C}{1/FC + 1}\frac{n(t)}{y_r(t)} \approx 1 + \frac{1}{C}\frac{n(t)}{y_r(t)} \tag{5.12}$$

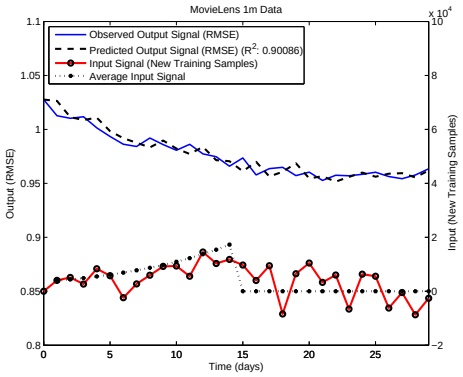
where the approximation holds when $FC \gg 1$. The first term suggests that if feedback control is adopted, the system performance will be close to the reference value regardless of the errors in the system's parameters as long as $FC \gg 1$. Moreover, the second term indicates that any disturbance $n(t)$ is attenuated by a factor of $1/C$. Thus, the feedback control is robust to noise and disturbance. As suggested in Equation 5.11, the intuition of having such protection is due to the fact that although F implies the noise $n(t)$ causes $y(t)$ to increase, an error is immediately created. It is amplified by FC to create an output of FC opposite and nearly equal to $Fn(t)$. As a result, it returns $y(t)$ close to its original value.

5.3 Designing a Controlled Recommender System

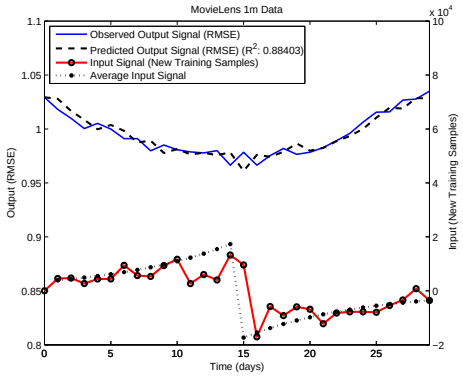
In this section, a series of experiments were conducted in order to evaluate the performance of the proposed control-theoretical approach of recommender systems. Without loss of generality, the empirical study focused on the relationship between the input (the number of training samples) and the output (recommendation accuracy). We chose this relationship, because the input and the output generalise the recommender system's dynamics best, therefore, with minimal modification it is applicable to a wide range of scenarios (see Section 5.4). Other inputs and outputs can be obtained by following the same design principle and steps presented here (for further suggestions see Section 5.5). The experiments were conducted by emulating the real use of recommendation systems with the widely used MovieLens 1m dataset (described in Section 4.3.2). The benefits of such an evaluation setup compared to using an operational recommendation engine are two-fold: first, it allows us to flexibly test various scenarios, and make the individual experiments targeted and focused; second, using public datasets allows others to easily replicate and compare the results, and validate our conclusions. Three widely used recommendation algorithms; the user-based, the item-based and the SVD method were used as the basis of rating prediction. As no significant difference was found among them in terms of controllability, we only report the results obtained using the popular SVD algorithm [Fun06].



(a) Increase



(b) Stable



(c) Decrease

Figure 5.5: Parameter Estimation (MovieLens 1m): The parameters introduced in Equation 5.5 were learned on set (c) and tested on (a) and (b).

The design principle was evaluated in several stages by building a dynamical recommender system step by step and sampling the rating data over time. Specifically, to prevent overfitting, the rating data was randomly divided into three sets; training, test and hold-out set. We fed the data into the system incrementally, so that each day during the predefined period of 30 days we randomly added a pre-specified number of training samples to the training set based on the requirements set by our feedback model and re-trained the recommender system using all the data in the training set. We then estimated the recommendation accuracy using the available user feedback data from the next day. This process was repeated each day, adding all the tested data points to the hold-out set, which was then used to pick the required number of data samples for the next cycle of training.

We started with the parameter estimation of performance dynamics based on historical data. It was then followed by the system analysis that aimed to identify the controller's characteristics and choose the right parameters before deploying the system. With the right parameters, we then evaluated the chosen controllers with real data and simulated scenarios. A potential drawback of such a controlled evaluation configuration might be its lack of testing the robustness of the system against any unseen environment noise. We thus provided additional experiments to test how well the system handles sudden changes over time. Finally, we tested how this approach can be applied to balance the trade-off between computational

Table 5.1: The R^2 Performance of the Three Input Datasets from which r and k were estimated (Equation 5.5): The set marked in bold was used to estimate the parameters which was then tested on the remaining two sets. We used the row coloured grey for the further experiments.

Increase	Stable	Decrease	Mean	Standard deviation
0.9863	0.6699	0.3974	0.6845	0.2947
0.9723	**0.9439**	0.9176	0.9446	0.0274
0.9702	0.9421	**0.9314**	0.9479	0.0200

(a) Input without noise

Increase	Stable	Decrease	Mean	Standard deviation
0.9869	0.8852	0.7766	0.8829	0.1251
0.9646	**0.9078**	0.8726	0.915	0.0464
0.8945	0.9008	**0.8840**	0.8931	0.0084

(b) Input with noise (see Equation (5.6))

cost and performance requirements.

5.3.1 Parameter Estimations

System Dynamics

In order to design a suitable controller, first we need to understand the dynamics of the system by obtaining the parameters that describe the recommender systems dynamics (introduced in Equation 5.1). The dynamics of the system is needed for the controller design to simulate the feedback loop system (that includes the recommender system and the controller). Without an accurate description of the system dynamics, it is not possible to predict the system behaviour and obtain a suitable controller. To estimate the parameters using the approach proposed in Section 5.2.1, we devised three scenarios that the recommender system might encounter. The first one covers the situation when the data is growing exponentially which is the case where many new users start using the system. The second is concerned with the situation where growth flattens after some initial growth, whereas the third one simulates the decrease of the training data. This last scenario was to cover the possibility that the controller might reduce the performance of the system.

Figure 5.5 illustrates the relationship between the number of new samples and output the recommendation accuracy measured by RMSE. The parameters were estimated using Equation (5.5) on one of the scenarios and tested on the remaining ones to predict the accuracy of the system. The random walk technique (described in Section 5.2.1) was used to generate the input sequences (how the influx of new ratings evolves over time). They were estimated from the data to represent the natural way of data growth. Adding noise to the signal helped us model various input combinations. The input was put through our recommender model to generate the performance over a period of 30 days.

A key question for parameter estimation is how to construct an input sequence such that the prediction of output from an unseen input sequence would be as accurate as possible. We measured how well the prediction fits to the observed data using the coefficient of determination [Ham94]

$$R^2 = 1 - \Big(\sum_{t=0}^{T} (y_h(t) - y(t))^2 / \sum_{t=0}^{T} (y(t) - \bar{y})^2\Big) \tag{5.13}$$

where $R^2 \in [0, 1]$, y_h is the predicted output, y is the observed output and $\bar{y}$ represents the mean of the observed output over a time period $t \in [0, T]$. This measure is widely used in statistical models to measure the prediction of future outcomes; the higher the value is, the better it fits to the data. Table 5.1 shows that the most robust parameters across all the three scenarios were obtained by estimating them on the data shown in Figure 5.5 (c) (the 6th row in Table 5.1) as it generates the smallest standard deviation across the three scenarios. This also suggests that adding noise using the random walk model helps improving the robustness of the estimation. As illustrated in Figure 5.5(a) and (b), the model produced consistent predictions despite the fact that the data was noisy, therefore, the parameters were robust to be used for the next steps in the controller design.

Step Response and the Feedback Controls

Having obtained the system dynamics of recommender model, we then studied what kind of controller best suits our predefined control objectives and obtain the parameters defined in Equation (5.8). We evaluated four kind of controllers including a Proportional (P), a Proportional-Derivative (PD), a Proportional-Integral (PI) and a Proportional-Integral-Derivative (PID) controller. To quickly test our ideas, we used a standard PID design tool provided in the Control System Toolbox in Matlab [AH06]. In a nutshell, this tool simulates the behaviour of the dynamical system given the internal dynamics of the recommender system and the controller. It obtains the initial parameters using heuristic approaches, such as the Ziegler-Nichols tuning method [ZN93] to satisfy certain initial requirements: they include closed-loop stability, adequate performance and robustness. Then, this initial design can be fine-tuned manually to achieve the specific requirements for the given system. For details of the method, we refer to [Oga01], while staying focused on finding an ideal controller that has the characteristics that satisfies our requirements.

Four measures were used to obtain an insight into the stability and the sensitivity of a feedback controller before employing it in the system. *Step response* is defined as the time behaviour of the output of a general system when its input suddenly changes from zero to one in a very short time; *Rise time* refers to the time required the output to reach 90% of the reference value. For example if the reference value is set to one and the current output of the system is zero (regardless of the metric used to quantify the output), rise time shows how fast the system can reach 0.9. Essentially, this value indicates how fast the system can respond to a change in the rate of influx of ratings. However, fast rise time might cause the output to exceed the desired value: this is called the *overshoot* of the signal. In some cases, for example when we aim to control the computation of the system, it is desired to keep overshoot as low as possible. Since the system might require more computation than it is available which can cause server overflow. We also measured how long it takes the feedback system to remain within a specified error

Table 5.2: The General Characteristics of the Estimated Recommendation Controllers: The time measured by seconds corresponds to days in our experiments (e.g. one training cycle).

Type	Rise Time (sec)	Settling Time (sec)	Overshoot (%)
P-I	1	2	0
P-II	5	10	0
PD-I	0	2	9.84
PD-II	1	5	8.21
PI-I	0	8	3.24
PI-II	0	10	36.8
PID-I	0	4	8.41
PID-II	0	10	15.7

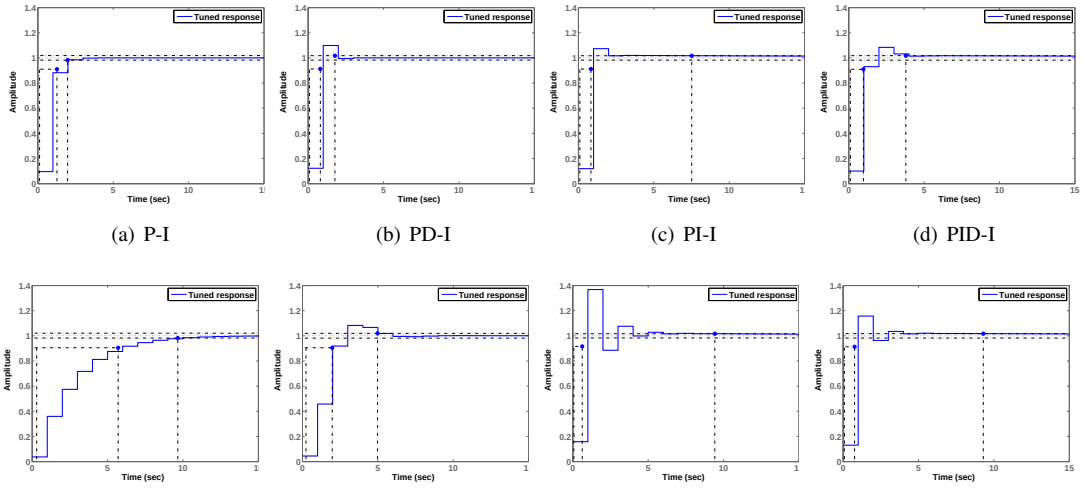
band (*settling time*). The error band is defined to be $\pm 1\%$ of the target value. In some cases, the system stabilises at a different value that is required; thus *steady-state error* is defined as the difference between the output and the target value after the output has reached the steady state.

A recommender system requires a controller with a fairly fast response time, that is the controller should react to changes as fast as possible. Ideally, this would be less five training cycles given the fact that a training cycle in practice can be as long as a day. This should also be accompanied by a fast settling time and a small steady-state error. These requirements would ensure that the system converges to the reference value quickly which is important in order to gain control of the system in a reasonable time frame. Figure 5.6 depicts the step response of the four controllers, whereas the main characteristics are also summarised in Table 5.2. For each of the controllers, we have two configurations, one of them was set to be "fast" (in terms of rise time) (mark I), and the other one is designed to be "slow" (mark II) with a maximum settling time of ten training cycles. The slower controllers are generally more stable and less likely to overshoot, which might be desirable in some scenarios, for example if the computational resources are scarce.

Figure 5.6 shows that all the controllers reach the desired output and remain stable. However, controllers that include an integral part (such as PID and PI) were slower to settle, and overshot the target considerably. This may be due to the fact that the integral controllers are intended to reduce that residual error over time by accumulating the errors. In addition, this becomes an important issue as, compared to physical systems, recommender systems might not always have the required number of training samples available which affects accumulated error over time. We thus introduce a linear discounted output to add it to the integral controller as follows

$$y_i(t) = \begin{cases} y(t) + \frac{1}{r+k}(s_i(t) - s(t)) & \text{if } s_i(t) > s(t) \\ y(t) & \text{if } s_i(t) \leq s(t) \end{cases} \tag{5.14}$$

where $y_i(t)$ represents the modified output (performance) discounted by the difference between the re-



(a) P-I (b) PD-I (c) PI-I (d) PID-I

(e) P-II (f) PD-II (g) PI-II (h) PID-II

Figure 5.6: Step Response (MovieLens 1m): The first point represents the rise time of the system (90% of the target value) and the second point marks when the system reaches steady state. The first and the second rows show the fast controllers with 2-5 sec response time, the third and fourth rows have the slower controllers with 5-10 sec response time.

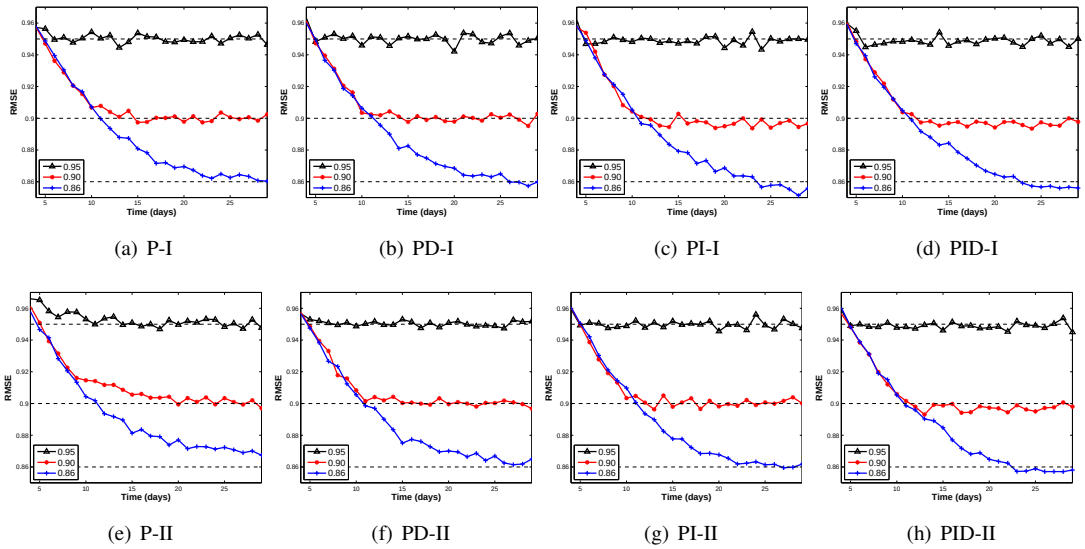
quired input ($s_i(t)$) (defined by the controller) and the actual input ($s(t)$). This compensates the discrepancy between the required and the available training samples. Essentially, the influence from the integral part is discounted when the required number of training samples is not available.

By contrast, we observe that the derivative term helped to increase the response time of the feedback system. The PD controller was faster than the P controller, but it settled at the same time. In this regard, the PD controller possesses more suitable characteristics of controlling a feedback recommender system in this case.

5.3.2 Controlled Recommendation

Having learned the parameters and understood the general characteristics, we are now ready to deploy the controlled recommender systems and evaluate their performance by replacing the simulated system dynamics with a real recommender system. We first studied the initialisation. We picked three different reference values (0.95, 0.90, 0.86). Without any control and the use of all the available data: they can be reached in 4, 9 and 18 days respectively, given the rate of data growth in the dataset. We ran the experiments with the four controllers discussed above to see how the system behaves with respect to the change of the reference value. This is executed by adding the controllers analytically obtained in the previous section to the real system and compare the behaviour of the system predicted by the analysis to the behaviour of the real system.

The five-fold cross-validated results are shown in Figure 5.7. First, we observe that all the controllers can stabilise the system to the reference value. The behaviour of the system was consistent with our understanding of the offline step response analysis. The analysis also correctly predicted which controllers tend to overshoot, but the extent of the overshoot was less than it was anticipated. This was due



(a) P-I (b) PD-I (c) PI-I (d) PID-I

(e) P-II (f) PD-II (g) PI-II (h) PID-II

Figure 5.7: The Characteristics to Reach Certain Reference Values (0.95, 0.90, 0.86) per Controller (MovieLens 1m): The controllers in the first row converge faster whereas the controllers in the second row are are slower but more stable.

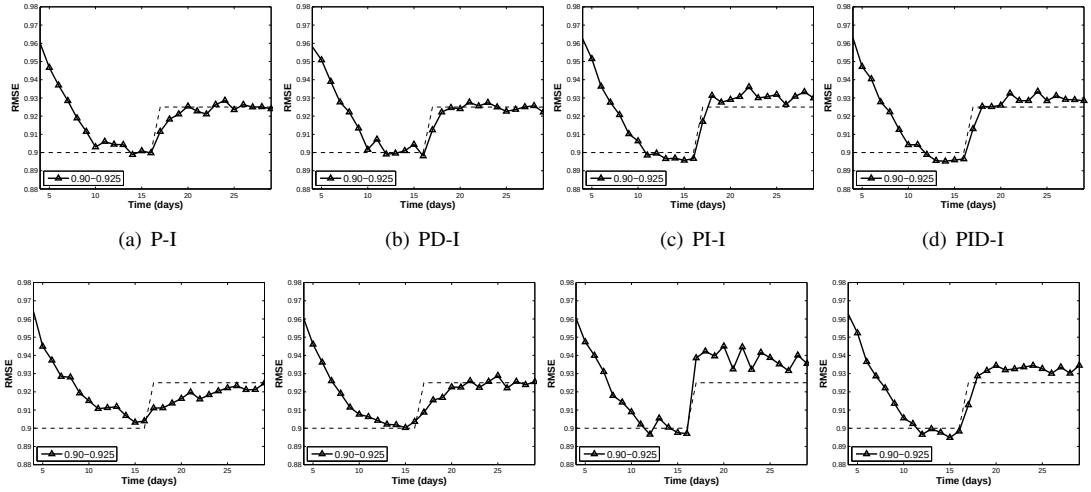
to the fact that the rate of performance improvement slows as the data grows. The rate of overshooting therefore decreased when the system approached a lower reference value (better performance). The PD controller (Fig 5.7(c)) overshot the target, but it was faster to settle with much smaller steady state error. We also observed that the PID controllers (Figure 5.7 (d) and (h)) had a relatively big steady state error, mainly due to the integral part of the controller which accumulated the residual error over time.

Our observations were further quantified in Table 5.3 by introducing the following metrics. Settling time was measured as the time spent to settle within $\pm$ 1% of the target value. We also monitored the stability and the steady state error of the system. To measure stability of the performance over time, we define SD-SS, which measures the error from the mean; to measure the error, we define RMSE-SS as the root mean squared error from the reference value (note that we also used RMSE to measure the recommendation accuracy). Both SD-SS and RMSE-SS were calculated from the point where the system settles. The results show that all the controllers (except the P-II) have relatively small errors. The best controllers in terms of error (RMSE-SS) were the PD-I, PID-I and PID-II controller which are marked grey in the table. The differences between the three best controllers (in terms of RMSE-SS) were not statistically different, but their values were statistically different from the other controllers.

We also observed that that the slowest controller was the P-II (which did not even reach the reference value). It was followed by the PD-II, and the rest of the controllers produced similar results (not statistically different). This differs from Table 5.2, where the setting time varied significantly. In the case of the PI-I controller, it had a long predicted settling time in Table 5.2, but we observed in Figure 5.7(c) that it reached the target value fast and it stayed slightly below the reference value. We believe this may be due to the fact that in practice (as in the simulation) the change of the reference value is much smaller than the theoretical change in step response. Thus, the settling happens faster in the simulation.

Table 5.3: The Speed, Precision, Stability and Overshoot of the Controllers for Reference Value 0.86: The best performing controllers in terms of error are coloured grey.

Type	Settling Time	RMSE-SS (error)	SD-SS (stability)	Overshoot (%)
P-I	18.6	0.00637	0.00479	0.35632
P-II	23	0.01049	0.00346	0
PD-I	19.8	0.00495	0.00448	0.49484
PD-II	20.8	0.00599	0.00377	0.18454
PI-I	18.8	0.00597	0.00620	1.14088
PI-II	18.6	0.00556	0.00479	0.49416
PID-I	19.2	0.00449	0.00460	0.83654
PID-II	18.6	0.00452	0.00455	0.72810



(a) P-I (b) PD-I (c) PI-I (d) PID-I

(e) P-II (f) PD-II (g) PI-II (h) PID-II

Figure 5.8: Reaction to Reference Change (MovieLens 1m)

In addition, fewer overshoots were observed in the simulation than in step response. This is mainly due to the fact that increasing recommendation accuracy becomes more difficult as accuracy increases (the non-linear relationship between the input and output in Equation 5.1). In terms of stability, as predicted, the slower controllers (mark II) were more stable their faster counterparts.

Table 5.4 (a) extends the experiments by considering the reference value change from RSME 0.90 to 0.925 (Figure 5.8). To further evaluate the reaction of system the with respect to unexpected changes, we introduced disturbance within the underlying recommender model (SVD algorithm). We ran the experiments with the same settings as before, but in day 15 we un-tuned the parameters of the SVD model, in order to simulate an extreme version of disturbance in the model. This change has a sudden effect on the overall recommendation accuracy. The controllers did stabilise the system by compensating the number of new training samples. Table 5.4 (b) summarises the results of the experiment.

In Table 5.4(a) and (b), the top performing controllers were marked grey and they were statistically

Table 5.4: The Speed, Precision, Stability and Overshoot of the Controllers: The best performing controllers in terms error are marked grey.

(a) Reference change from RSME 0.90 to 0.925

Type	Settling Time	RMSE-SS (error)	SD-SS (stability)	Overshoot (%)
P-I	18.2	0.00473	0.00476	0.68955
P-II	20	0.00689	0.00529	0.50125
PD-I	18	0.00448	0.00463	0.93336
PD-II	18.8	0.00549	0.00559	0.98527
PI-I	17.6	0.00807	0.00629	1.82811
PI-II	20.6	0.01376	0.00751	2.50716
PID-I	17.8	0.00641	0.00540	1.32989
PID-II	17.8	0.00914	0.00592	1.86331

(b) Untunning the parameters of the used recommendation predictor (SVD algorithm)

Type	Settling Time	RMSE-SS (error)	SD-SS (stability)	Overshoot (%)
P-I	18.5	0.00569	0.00520	0.74247
P-II	25.4	0.00763	0.00325	0.00513
PD-I	18.5	0.00499	0.00503	0.85006
PD-II	20.7	0.00547	0.00489	0.46552
PI-I	17.9	0.00580	0.00543	1.25541
PI-II	17.8	0.00489	0.00505	1.04460
PID-I	18.3	0.00546	0.00499	1.22198
PID-II	17.6	0.00552	0.00518	1.27250

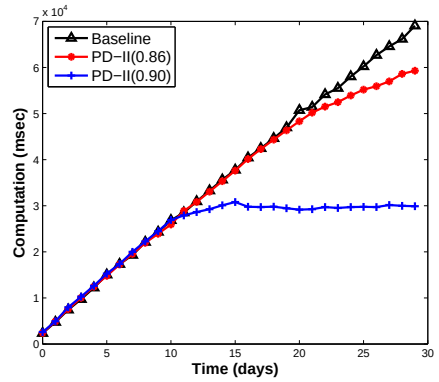
significant from the other ones. It is interesting to see that the PD controllers still performed consistently well in these two experiments, suggesting that the PD controllers are the best for our recommender system. We also observed that the controllers (the PI and the PID controllers) with an integral part were unstable in our system, thus not recommended. The integral part performed well when the system was required to improve the recommendation accuracy rapidly (Table 3 and Table 4(b)), but performed poorly when the performance was to be reduced (e.g., Table 4(a)).

5.4 Evaluation

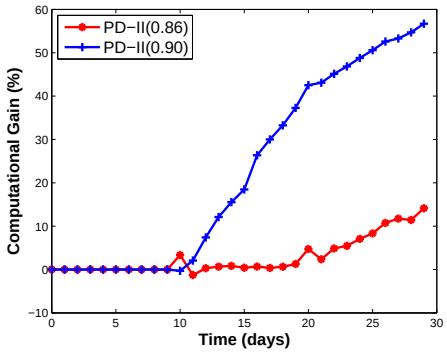
5.4.1 Computation Cost and Update Frequency

One of the benefits of the proposed feedback control is the ability of handling the trade-off between computational complexity and performance[1]. This would enable recommendation providers to have a principled tool to control the system and predict the resources needed for a predefined quality of service

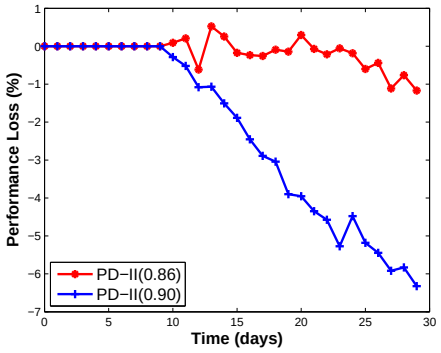
[1]It should be emphasised that an alternative formulation is needed to obtain the optimal control signals by directly optimising the specific goal (either the computation costs or recommendation accuracy). We leave this formulation for future work.



(a) Computational effort



(b) Computational gain over baseline



(c) Performance loss over baseline

Figure 5.9: Computation versus Performance (MovieLens 1m): The baseline approach has no controller employed. This is compared to keeping the performance at 0.86 and 0.90 respectively with the PD-I controller

even if the near future usage of the system is unknown. Computing recommendations is expensive, particularly for a large-scale system. The feedback controller can be fine-tuned to achieve the best level of recommendation performance given limited resources. In addition, by fixing the required computational resources needed to train the model, it is possible to predict the update frequency of the system, given the resources available. As the fixed computational complexity would determine how long it would take to train the model using the available resources. We can also produce highly regular updates independent from the rate the new samples enter the system. To achieve this, two main characteristics of the system should be considered. First, it is critical to reduce the overshoot, as overshooting the target performance could threaten the stability of the system. Second, it is desirable to have the fastest response time so that it would minimise performance loss.

We evaluated the ability with the PD-II controller as it has been proved to be the best in the previous experiments. Figure 5.9 (a) depicts the computational efforts (measured in CPU time) for the baseline (no controller) and for the PD-II controller with two reference values (0.86 and 0.90), respectively. As illustrated, once the required performance has reached, the computational cost becomes stable. Figure 5.9 (b) and (c) further demonstrate this by plotting the computational gain (i.e., reduced computational effort) versus performance loss over the baseline. Setting the reference to 0.86 can save up to 15% in

computation while the accuracy is reduced only by less than 1%. This difference becomes more obvious if we set the reference to 0.90, where the computational effort can be reduced by up to 55% while the performance loss is not more than 6.2%. This ratio is due to the fact that performance improvement slows down as the number of training samples increases (modelled in Equation (5.1)). Moreover, the PD-II controller has 0.5% - 1.0 % overshoot predicted by the analysis and the preliminary experiments, this would guarantee that the system stays stable over time and would not go over the predefined computational cost. Therefore, this approach would enable practitioners to use the system to its *full extent* without risking the overall system stability (by going over the resources available). By controlling the computational effort, we can make accurate predictions of how many times the system can be updated so that the system can provide fresh recommendations when it is required. If we set the reference value to 0.90 (using less training samples), we can provide 3200 updates a day (with our resources it takes 27 seconds to update the model). However, we could only provide 2400 updates with the reference value 0.86 in which case we use more training samples to train.

It is important to note that this approach shows the upper bound of the performance loss, as we chose to randomly subsample the data, but by using simple selection strategies (e.g. always train with the latest data), it would substantially reduce the performance loss. In addition, the approach can easily be tailored to produce incremental updates or updates per user instead of considering the whole data as long as the system dynamics is learnt on the appropriate inputs/outputs.

5.5 Discussion

There are a number of shortcomings of this method that can be addressed by extending a model towards a more fine-tuned way of controller design. We chose to represent the system with one single state (performance) and consequently one controller in this approach. The reason behind this is that we concentrated on understanding the controllability of the system dynamics, and evaluated how a control-theory oriented approach can model system dynamics. This approach can be easily extended and broken down to control each individual user and item or any other way that is convenient in practice.

Instead of controlling the update frequency, we can extend this approach to provide a prediction of when the system needs to be retrained given the worst performance acceptable. As the controller described in the previous section produces a signal of how many training samples are needed to converge to the reference value at a given time, this value can be used to compute the next time the system needs retraining. As shown earlier, the performance of recommender system decreases if it is not retrained periodically, this rate of decrease can be used to calculate the next time our system reaches the predefined reference performance. We define this as the *deterioration rate per sample* and can be estimated from historical data. It shows how long it would take the recommender system to return to its initial performance after adding one sample. In other words, if we know that we need to remove a number of samples to reach the reference value immediately, this would be a good indication of how long it would take for the recommender system the converge to the reference value without removing any samples.

It is important to note that breaking down the performance of the system to days or even hours shows that the system dynamics is sensitive to a number of factors. These include how much the system

knows about the user and the item in question, the temporal change in taste of the user, how obscure or mainstream the item is. This work has not directly provided answers to these questions, but the study about the impact from the number of training sample provides a guideline and design pattern as to how to deal with those factors. If we are able to identify all the factors that constitute to the system dynamics, we might be able to stabilise them over time by designing a control loop for each individual factor, or considering them as disturbance or noise.

5.6 Conclusion

In this chapter, we showed how Modern Control Theory can help us design and analyse dynamical recommender system goals. By proposing to use a simple spring and damper model to deal with the performance dynamics and deploying it with PID controllers, we have achieved a stable control of the recommender system over time. Not only providing a flexible method to handle the trade-off between computational cost and performance, this approach also provides a way to identify and analyse the characteristics of the dynamics, and provides principled tools to achieve the desired objectives.

There are other fruitful opportunities in this research direction. An interesting study would be to consider what types of input signals might be useful to keep the performance steady. For instance, it is worth exploring the control strategies with active learning, that defines which samples should be added to the training set to maximise performance. Instead of choosing randomly from the availably samples the controller can follow pre-defined strategies, e.g., the data points could be selected based on their age, the estimated difficulty of the user/item etc. These factors would provide further practical solutions to manipulate the performance of the system.

The idea is also useful to apply to other online services as they all inherently exist in a time-dependent context [Kle06]. Similar dynamics occurs in a web page content [ATDE09] and revisitation habits [ATD09]: control theory could be used to respond to the flow of information to control the outcome of the dynamical system. Controlling the reference of the recommender system based on the parameters of the model.

Chapter 6

Conclusion and Future Plan

This thesis presented a goal-driven recommender system design, where the focus of the system moved from general accuracy measures toward goal focused algorithms. The consequences of this shift were outlined in the thesis, which includes more specific accuracy metrics that can cover the goal of the system, and a number of approaches that modify the objective function of the system to adjust to specific goals. This includes the differentiation between user or system focused approaches and internal or external objective function design. In this work, we provided examples to illustrate this approach for each of the categories identified. Further in this section, we identify the shortcomings of this approach and present a number extensions that are applicable within this framework.

6.1 Thesis Contribution

In order to evaluate the applicability of a goal-driven system, each approach was broken down to the following categories. At large, these steps correspond to the general work-flow of data mining, with an additional step on the analysis of the aim of the system (Figure 6.1). The following steps we employed for each part of this thesis.

6.1.1 Analysis

The overall purpose of this step was to determine whether it was feasible to detect patterns and relationships across the dataset. This step aimed to clarify the predetermined goal of the system by exploring whether the pattern we suspected was actually there, also to investigate whether the data met the requirements of the potential predictive models (e.g. linearity between features).

6.1.2 Goal Selection

Based on the data and the patterns we found, we determined the type of goals that were possible with this kind of data. We focused on problems that were identified in the literature or by the industrial partner of this work. It is important to note that the first and the second steps are interchangeable depending on the approach and the focus of the work. Furthermore, they are likely to be iteratively or simultaneously defined, depending on the nature of the problem and the data.

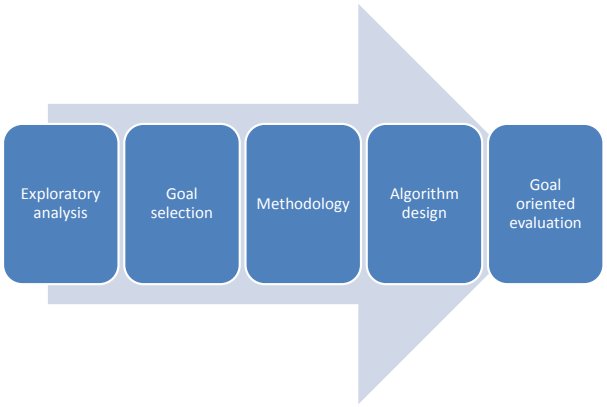


Figure 6.1: General Workflow to Building Goal-Driven Systems

6.1.3 Methodology

The methodology employed in this work was also driven by the goal of the system. This included the selection of the evaluation methods in line with the goal. For external approaches, we modularised the system and evaluated each component separately. For internal approaches, we designed various evaluation metrics to measure the effect of the components on the whole system. In addition, for temporal evaluation, we defined the partitioning of the data in a temporal fashion. For example, we simulated the system in a way the data arrived originally and trained the models at the time and frequency they were usually trained in practice (e.g. once a day at midnight).

6.1.4 Algorithms

The main contribution of the thesis was the algorithm design. As mentioned above, goal-driven design shifted the focus of the development towards more practical metrics and algorithm design. This affected the choices of the algorithms in terms of their potential fit to the problem. This way we employed the 'fit the algorithm to the data' approach. We developed a number of algorithms for specific goals, and an approach to combine them together (for certain external algorithms).

6.1.5 Datasets

Experiments that were used in the thesis were in line with the rigorous experimental design used in collaborative filtering research [HKTR04]. In this work, we mainly used the MovieLens dataset, but also conducted some initial experiments on the Netflix dataset [BL07], both are publicly available. The results in Section 4.2 were investigated using customer data from a large digital TV content provider. This offered a different angle to the problems, since it was collected from a slightly different domain (i.e. IPTV) and represented different customer behaviour (e.g. the indication of likes and dislikes were expressed differently).

6.2 Future work

This section contains a collection of ideas which we identified as possible extensions of this of this work. This can be divided into two main areas; extension on the algorithms to cover the scenarios defined in

Section 1.3.2 and extensions on the evaluation methodology.

6.2.1 Algorithmic Extensions

Diversity and Serendipity

The exact nature of the connection between diversification and long tail items has not been studied in detail yet. It was discussed in Chapter 4 that diversification alone does not necessarily guarantee that items from the long tail would be recommended, since it might only distribute items that are already popular. We ran a simple experiment in Section 4.3.1 that showed that the combination of diversification and the Long Tail Constraint would provide a better result in reducing popular items in the top positions on the recommended list. However, the exact nature and the relationship between diversification and items in the long tail are still to be explored.

Diversifying one's recommendation goes against users' incentives, as users prefer listening to familiar items. The main problem that arises in this setting is to persuade users to pick items that might be unfamiliar to them (e.g. serendipitous items). We could approach this by nudging users towards serendipitous items using the well-studied anchoring effect [Ari09]. Anchoring effect is a cognitive bias that describes the common human tendency to rely on one trait or piece of information when making decisions. In other words, users tend to pick items that are comparable to other items, so that the *value* of the picked item can be assessed (in relation to the anchor).

This can be further explored in the domain IPTV recommender systems. We identified two different levels where this approach can be applied to such a system in practice. The potential of the subscription pack can be maximised by diversifying the possible content available within a pack, thus, increasing the overall value of it. Another approach is to apply it outside the subscribed pack, which might encourage customer to subscribe to other packs. This is riskier compared to the previous level, since the recommendation is based on a new group of items (i.e. different genre) which would be mined from the current subscription details. This decision might be supported by the user's viewing history from the freely available content and live television broadcast.

Optimal Control Theory

The Optimal Control Theory approach, as the name suggests, deals with finding optimal policies for a given system over a period of time. This is a useful extension of the approach introduced in Chapter 5, as most of the scenarios discussed there can be reinterpreted as an optimisation problem. Instead of keeping the objectives of the systems close to a pre-defined reference value, we can to optimise them for a given time period. Recommender systems are normally optimised at a given time in order to provide the best performance after the actual time of the training. However, it is shown that the available samples might not be equally useful to maximise the performance of the system, as some of the samples might give false information due to noise in the data [APO09], therefore, the underlying model generalises incorrectly if it is based only on these pieces of information. This can be interpreted as a one-off generalisation that might not be useful to capture user behaviour. Based on this assumption, this extension would identify ratings that affect the performance of the system *over a longer period of time*, and remove samples that

are not useful for generalisation in a longer term. Thus, we assume that a recommender system might not need all the training samples to maximise performance (i.e. there are some samples that actually hurt performance). It is time dependent when to inject the samples in the system and when to remove them. The approach would control from when and how long training samples stay in the system. This can be viewed as the means to describe the system performance (the objective that needs to be optimised) and gain control over it through discovering how certain training samples affect performance. In order to complete this work, we need to investigate the relationship between the performance of the system and the decision to include certain training samples. The simplest approach to it is to define a number of features that characterise a data point. This can include the age of the sample, how likely that this rating is consistent with the user's and the item's profile, the derivation from the expected rating strategy of the user and the item, etc. These features can be calculated from historical data (prior knowledge) or estimated based on our assumption and observations of the underlying psychological processes that describe how a user rates an item. After the features are estimated, we can compute whether they are predictive to obtain the performance of the system. Then this relationship might be used to describe the system dynamics. The system dynamics can then be modelled over time and solved to obtain the optimal decision that minimises the system performance for a given time period.

6.2.2 Online Evaluation Methodology

One of the assumptions we made in this thesis was that the performance metrics we defined to measure certain aspect of the recommender system captured what it was intended to measure. In reality, there might be some discrepancy between what a metric is indented to capture and what it actually captures. Mainly because the process of defining such measures is based on our assumptions and understanding of what aspects of the problem would be sufficient to monitor in order to evaluate our goals.

Furthermore, we make another assumption during the evaluation process. That is we evaluate on a simulated "future" status of the system that might not have existed if our models were in place. As offline evaluation methods hold parts of the data as test samples, assuming that the alterations we made by presenting our models to the users would not affect generate the same training samples. For example, we introduce a model that would provide novel recommendation to users, we assume that it would not affect the user feedback that happens after the model is introduced. In this way, the alteration that the model introduces in user behaviour might not be captured just by measuring how it would perform on the training sample.

Therefore, it is important to explore how these two assumptions would result in the performance of the system with respect to the relation between perceived and observed coverage of the measures as well as how these altered models affect user behaviour. The only way to explore these complex relations is by evaluating goal driven models through a series of online experiments where both assumptions are tested. This is more important for user-oriented goals where above problems are less clear and harder to simulate.

6.3 Goal Driven Design in the Data Driven World

As explained above, the underlying philosophy in this thesis was to shift the focus towards concentrating on the goals that the recommender system was to fulfil, and keeping these principles in mind at each step of the process. This is increasing important as the volume and variety of the data that is available to use for predictions is exploding. As a result, many turn to the data to explain patterns and use their initial findings to model certain problems. Data driven thinking would benefit from our goal-driven approach as it prioritises the goal (that would emerge as a result of data driven thinking) as opposed to prioritising on certain error measures. This is important in order to take full advantage of new sources of data and effectively incorporate findings that emerge from the data.

Appendix A

Stability of a Controlled System

The controllers is in charge of processing the error from the plant and turning it into a signal that would decide on two things, whether the model will be re-trained or alternatively it could adjust some of the parameters of the model that are defined to capture instant temporal change.

There are numerous advantages of this method, one of them is to separate certain aspects of the system and understand how they contribute to the final prediction value. Another one is that we can find an optimal solution to when the system needs updating after training. This is the trade-off between loosing performance after some time and other practical factors (e.g., the computational complexity of the algorithm, getting the updates from the user).

Moreover, the feedback control is more efficient in dealing with the disturbance. Mathematically, let us look at the block diagram representation, shown in Figure 5.4 (b).

$$\begin{aligned} e(t) &= y_r(t) - y(t) \\ s(t) &= G_1(e(t)) \\ y(t) &= G_2(s(t) + n(t)) \end{aligned} \tag{A.1}$$

where $n(t)$ denotes a noisy signal injected to the system. Converting them into the frequency domain using the z-transform gives

$$\begin{aligned} E(z) &= Y_r(z) - Y(z) \\ S(z) &= G_1(z)E(z) \\ Y(z) &= G_2(z)(S(z) + N(z)) \end{aligned} \tag{A.2}$$

We thus have:

$$\begin{aligned} Y(z) =& G_2(z)\Big(G_1(z)\big(Y_r(z) - Y(z)\big) + N(z)\Big) \\ =& G_1(z)G_2(z)Y_r(z) - G_1(z)G_2(z)Y(z) + G_2(z)N(z) \end{aligned} \tag{A.3}$$

Finally, we obtain the output $Y(z)$ as

$$\begin{aligned} Y(z) =& \frac{G_1(z)G_2(z)Y_r(z) + G_2(z)N(z)}{(1 + G_1(z)G_2(z))} \\ =& \frac{G_1(z)G_2(z)}{1 + G_1(z)G_2(z)}Y_r(z) + \frac{G_2(z)}{1 + G_1(z)G_2(z)}N(z) \end{aligned} \tag{A.4}$$

This equation says any disturbance $N(z)$ is attenuated by $\frac{G_2(z)}{1+G_1(z)G_2(z)}$. If $G_1(z)G_2(z)$ is 100, then only 1% of $G_2(z)N(z)$ is allowed to affect $Y(z)$. Even if $G_1(z)G_2(z)$ is only 10, then only 10% of $G_2(z)N(z)$ gets through, which is a lot better than no protection at all. In other words, if $G_2(z)N(z)$ causes $Y(z)$ to decrease, an error is immediately created. That is amplified by $G_1(z)G_2(z)$ to create an output of $G_1(z)G_2(z)$ opposite and nearly equal to $G_2(z)N(z)$, thereby returning $Y(z)$ close to its original value. Thus, the feedback control is robust to the noise and disturbance.

Secondly, the feedback also protects the system against internal parameters. To see this, let us set $N(z) = 0$, and calculate the ratio between output and input:

$$\frac{Y(z)}{Y_r(z)} = \frac{G_1(z)G_2(z)}{(1 + G_1(z)G_2(z))} \tag{A.5}$$

If $G_1(z)G_2(z)$ has a nominal value of 100, $Y(z)/Y_r(z)$ = 100/101 = 0.99, close to the desired closed-loop gain of 1.00. Now let $G_1(z)G_2(z)$ drop by 50%. Then $Y(z)/Y_r(z)$ = 50/51 = 0.98. So a 50% drop in $G_1(z)G_2(z)$ caused only a 1% drop in closed-loop performance. Even if $G_1(z)G_2(z)$ dropped by a factor of 10, to a gain of 10, still $Y(z)/Y_r(z)$ = 10/11 = 0.91, an 8% decrease. In general, so long as $G_1(z)G_2(z) > 1.0$, then $G_1(z)G_2(z)/(1 + G_1(z)G_2(z))$ will be close to 1.0.

Appendix B

PID Controller Derivation

Based on Equation (5.9), the transform function between the input and output can be obtained as follows:

$$\begin{aligned}
&\text{Time domain: } s(t) = Ce(t) + B\sum_{t=0}^{t} e(t) + D\Big(e(t) - e(t-1)\Big) \\
&\text{Freq. domain: } S(z) = CE(z) + BE(z)\frac{z}{z-1} + DE(z)\frac{z-1}{z} \\
&\text{Freq. domain: } G_1(z) \equiv \frac{S(z)}{E(z)} = C + B\frac{z}{z-1} + D\frac{z-1}{z}
\end{aligned} \tag{B.1}$$

where z is the variable in the frequency domain (usually a complex number). G_1 represents the controller's gain. From the performance dynamics, we equally have

$$\begin{aligned}
&\text{Time domain: } s(t) = r\Big(y(t) - y(t-1)\Big) + ky(t) \\
&\text{Freq. domain: } S(z) = r(1 - z^{-1})Y(z) + kY(z) \\
&\text{Freq. domain: } G_2(z) \equiv \frac{Y(z)}{S(z)} = \frac{z}{z(r+k) - 1}
\end{aligned} \tag{B.2}$$

where G_2 represents the system dynamics. Combing G_1 and G_2 obtains the transfer function of the closed-loop system:

$$\frac{Y(z)}{Y_r(z)} = \frac{G_1(z)G_2(z)}{1 + G_1(z)G_2(z)} \tag{B.3}$$

where the transfer function $Y(z)/Y_r(z)$ is expressed as a ratio of two polynomials in z, and is a function of both the system parameters and controller's parameters. The roots of the numerator are called its *zeros*, whereas the roots of the denominator are its *poles*. One of the popular techniques to obtain the controller's parameters is called Root Locus design, which is a graphical technique that plots the traces of poles of a closed-loop system on the s plane as its controller parameters change. By defining the required property of the transfer function, one can estimate the controller's parameters accordingly.

Bibliography

[ACL05] K. H. Ang, G. Chong, and Y. Li. PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.

[AH06] K. J. Astrom and T. Hagglund. *Advanced PID control*. Instrumentation, Systems, and Automation Society, 2006.

[Ahn08] H. J. Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Information Sciences*, 178(1):37–51, 2008.

[Ama12] X. Amatriain. Netflix Recommendations: Beyond the 5 stars. `http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html`, 2012.

[And08] C. Anderson. *The long tail: Why the future of business is selling less of more*. Hyperion, 2008.

[APO09] X. Amatriain, J. M. Pujol, and N. Oliver. I like it... I like it not: Evaluating user ratings noise in recommender systems. In *Proceedings of the 2009 international conference on User Modeling, Adaptation, and Personalization*, pages 247–258. Springer, 2009.

[APTO09] X. Amatriain, J. M. Pujol, N. Tintarev, and N. Oliver. Rate it again: Increasing recommendation accuracy by user re-rating. In *Proceedings of the 2009 ACM international conference on Recommender Systems*, pages 173–180. ACM, 2009.

[Ari09] D. Ariely. *Predictably irrational: The hidden forces that shape our decisions*. HarperCollins, 2009.

[AT05] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[ATD09] E. Adar, J. Teevan, and S. T. Dumais. Resonance on the web: Web dynamics and revisitation patterns. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1381–1390. ACM, 2009.

[ATDE09] E. Adar, J. Teevan, S. Dumais, and J. Elsas. The Web Changes Everything: Understanding the Dynamics of Web Content. In *Proceedings of the 2009 ACM international conference on Web Search and Data Mining*, 2009.

[AZ12] G. Adomavicius and J. Zhang. Impact of data characteristics on recommender systems performance. *ACM Transactions on Management Information Systems*, 3(1):3, 2012.

[BBKV09] R. Bell, J. Bennett, Y. Koren, and C. Volinsky. The million dollar programming prize. *IEEE Spectrum*, 46(5):28–33, 2009.

[BCC11] A. Bellogín, P. Castells, and I. Cantador. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the 2011 ACM international conference on Recommender Systems*, pages 333–336. ACM, 2011.

[BDM01] E. Bacry, J. Delour, and J. F. Muzy. Multifractal random walk. *Physical Review E*, 64(2):026103, 2001.

[BHC98] C. Basu, H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 1998 AAAI international conference on Innovative Applications of Artificial Intelligence*, pages 714–720, 1998.

[BHK98] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 1998 conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

[BK07] R. M. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 2007 IEEE international conference on Data Mining*, pages 43–52, 2007.

[BKM07] P. Bedi, H. Kaur, and S. Marwaha. Trust based recommender system for semantic web. In *Proceedings of the international joint conferences on Artificial Intelligence*, pages 2677–2682, 2007.

[BL07] J. Bennett and S. Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, page 35, 2007.

[BNJ03] D.M. Blei, A.Y. Ng, and M.I. Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[BS97] M. Balabanović and Y. Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[Bur10] R. Burke. Evaluating the Dynamic Properties of Recommendation Algorithms. In *Proceedings of the ACM conference on Recommender Systems*, pages 225–228. ACM, 2010.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[Can02] J. Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 238–245, 2002.

[CC08] Ò. Celma and P. Cano. From hits to niches?: or how popular artists can bias music recommendation and discovery. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, page 5. ACM, 2008.

[CCL+09] W.Y. Chen, J.C. Chu, J. Luan, H. Bai, Y. Wang, and E.Y. Chang. Collaborative filtering for orkut communities: discovery of user latent behavior. In *Proceedings of the 18th international conference on World Wide Web*, pages 681–690, 2009.

[CGN+11] P. Cremonesi, F. Garzotto, S. Negro, A. Papadopoulos, and R. Turrin. Comparative evaluation of recommender system quality. In *Extended Abstracts on Human Factors in Computing Systems*, pages 1927–1932. ACM, 2011.

[CH08] Ò. Celma and P. Herrera. A new approach to evaluating novel recommendations. In *Proceedings of the 2008 ACM international conference on Recommender Systems*, pages 179–186. ACM, 2008.

[Che05] A. Chen. Context-aware collaborative filtering system: Predicting the users preference in the ubiquitous computing environment. In *Location-and Context-Awareness*, pages 244–253. Springer, 2005.

[CK06] Harr Chen and David R Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 429–436. ACM, 2006.

[CT08] K. Collins-Thompson. Estimating robust query models with convex optimization. In *Advances in Neural Information Processing Systems*, pages 329–336, 2008.

[CT09] K. Collins-Thompson. Reducing the risk of query expansion via robust constrained optimization. In *Proceedings of the 2009 ACM international conference on Information and knowledge management*, pages 837–846. ACM, 2009.

[DG08] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DK04] M. Deshpande and G. Karypis. Item-based top-N recommendation algorithms. *ACM Transactions on Information Systems*, 22(1):143–177, 2004.

[DKKW12] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music Dataset and KDD-Cup'11. *Journal of Machine Learning Research-Proceedings Track*, 18:8–18, 2012.

[Ein56] A. Einstein. *Investigations on the Theory of the Brownian Movement*. Dover Publications Inc, 1956.

[EK10] D. A. Easley and J. M. Kleinberg. *Networks, Crowds, and Markets - Reasoning About a Highly Connected World*. Cambridge University Press, 2010.

[FH09] D. Fleder and K. Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management science*, 55(5):697–712, 2009.

[FLTW89] T. L. Friesz, J. Luque, R. L. Tobin, and B. W. Wie. Dynamic network traffic assignment considered as a continuous time optimal control problem. *Operations Research*, 37(6):893–901, 1989.

[Fun06] S. Funk. Netflix update: Try this at home. `http://sifter.org/~simon/journal/20061211.html`, 2006.

[GDF+10] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the tenth IEEE international conference on Data Mining*, pages 176–185. IEEE, 2010.

[GE00] K. K. Gill and C. A. Eldering. User-friendly electronic program guide based on subscriber characterizations, December 21 2000. US Patent App. 09/742,507.

[GHO99] G. H. Golub, P. C. Hansen, and D. P. O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999.

[GKL10] N. Golbandi, Y. Koren, and R. Lempel. On bootstrapping recommender systems. In *Proceedings of the 19th ACM international conference on Information and Knowledge Management*, pages 1805–1808, 2010.

[GKL11] N. Golbandi, Y. Koren, and R. Lempel. Adaptive bootstrapping of recommender systems using decision trees. In *Proceedings of the fourth ACM international conference on Web Search and Data Mining*, pages 595–604. ACM, 2011.

[GM08] A. Gunawardana and C. Meek. Tied boltzmann machines for cold start recommendations. In *Proceedings of the 2008 ACM conference on Recommender Systems*, pages 19–26. ACM, 2008.

[GNOT92] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[Gro06] GroupLens Research. MovieLens datasets. `http://www.grouplens.org/node/73`, October 2006.

[GZ79] K. R. Gabriel and S. Zamir. Lower rank approximation of matrices by least squares with any choice of weights. *Technometrics*, pages 489–498, 1979.

[Ham94] J. D. Hamilton. *Time series analysis*, volume 2. Cambridge Univ Press, 1994.

[Har03] E. F. Harrington. Online ranking/collaborative filtering using the perceptron algorithm. In *Proceedings of the 2003 international conference on Machine Learning*, volume 20, pages 250–257, 2003.

[HCRC11] M. Harvey, M. J. Carman, I. Ruthven, and F. Crestani. Bayesian latent variable models for collaborative item rating prediction. In *Proceedings of the 20th ACM international conference on Information and Knowledge Management*, pages 699–708, 2011.

[HKBR99] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd international ACM SIGIR conference on Research and Development in Information Retrieval*, 1999.

[HKTR04] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1):5–53, 2004.

[HKV08] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the eighth IEEE international conference on Data Mining*, pages 263–272, 2008.

[Hof01] T. Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning*, 42(1-2):177–196, 2001.

[Hof04] T. Hofmann. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems*, 22(1):89–115, 2004.

[HY08a] A. S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *Proceedings of the 31st international ACM SIGIR conference on Research and Development in Information Retrieval*, 2008.

[HY08b] A.S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 91–98. ACM, 2008.

[JE09] M. Jamali and M. Ester. Trustwalker: a random walk model for combining trust-based and item-based recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 397–406. ACM, 2009.

[JE10] M. Jamali and M. Ester. A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM conference on Recommender Systems*, pages 135–142. ACM, 2010.

[JK00] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 41–48. ACM, 2000.

[JK02] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[JS04] R. Jin and L. Si. A bayesian approach toward active learning for collaborative filtering. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 278–285. AUAI Press, 2004.

[JSK10] G. Jawaheer, M. Szomszor, and P. Kostkova. Characterisation of explicit feedback in an online music recommendation service. In *Proceedings of the fourth ACM conference on Recommender Systems*, pages 317–320. ACM, 2010.

[JW10a] T. Jambor and J. Wang. Goal-driven collaborative filtering - a directional error based approach. In *Proceedings of 32nd European Conference on Information Retrieval*, 2010.

[JW10b] T. Jambor and J. Wang. Optimizing multiple objectives in collaborative filtering. In *Proceedings of the ACM conference on Recommender Systems*, pages 55–62. ACM, 2010.

[JWL12] T. Jambor, J. Wang, and N. Lathia. Using control theory for stable and efficient recommender systems. In *Proceedings of the 21th international conference on World Wide Web*. ACM, 2012.

[Kar01] G. Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and Knowledge Management*, 2001.

[KBV09] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[Kle06] J. Kleinberg. Temporal dynamics of on-line information streams. *Data stream management: Processing high-speed data streams*, 2006.

[KMM+97] J. Konstan, Bo Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 1997.

[Kor08] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 426–434. ACM, 2008.

[Kor10] Y. Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 53(4):89–97, 2010.

[Kor11] Y. Koren. Recommender system utilizing collaborative filtering combining explicit and implicit feedback with both neighborhood and latent factor models, October 11 2011. US Patent 8,037,080.

[Lat10] N. Lathia. *Evaluating Collaborative Filtering Over Time*. PhD thesis, Department of Computer Science, University College London, UK, June 2010.

[LH74] C. L. Lawson and R. J. Hanson. *Solving least squares problems*, volume 161. SIAM, 1974.

[LHC09] N. Lathia, S. Hailes, and L. Capra. Temporal collaborative filtering with adaptive neighbourhoods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 796–797. ACM, 2009.

[LHCA10] N. Lathia, S. Hailes, L. Capra, and X. Amatriain. Temporal diversity in recommender systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 210–217. ACM, 2010.

[LR04] S. K. Lam and J. Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*, pages 393–402. ACM, 2004.

[LSY03] G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[LXZY10] N. Liu, E. W. Xiang, M. Zhao, and Q. Yang. Unifying explicit and implicit feedback for collaborative filtering. In *Proceedings of the 19th ACM international conference on Information and Knowledge Management*, pages 1445–1448. ACM, 2010.

[LY08] N. Liu and Q. Yang. Eigenrank: a ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 83–90. ACM, 2008.

[MA07] P. Massa and P. Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender Systems*, pages 17–24. ACM, 2007.

[ME95] D. Maltz and K. Ehrlich. Pointing the way: active collaborative filtering. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 202–209. ACM, 1995.

[Mic96] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer, 1996.

[Min01] T. P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the seventeenth conference on Uncertainty in Artificial Intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

[MKL09] H. Ma, I. King, and M. R. Lyu. Learning to recommend with social trust ensemble. In *Proceedings of the 32nd international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 203–210. ACM, 2009.

[MRS08] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[MTP10] M. R. Morris, J. Teevan, and K. Panovich. What do people ask their social networks, and why?: a survey study of status message q&a behavior. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1739–1748. ACM, 2010.

[Mul06] M. Mull. Characteristics of High-Volume Recommender Systems. In *Proceedings of Recommenders*, volume 6, 2006.

[MWZ+06] K. Meng, Y. Wang, X. Zhang, C. Xiao, and G. Zhang. Control theory based rating recommendation for reputation systems. In *Proceedings of the 2006 IEEE international conference on Networking, Sensing and Control*, pages 162–167. IEEE, 2006.

[MZL+11] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web Search and Data Mining*, pages 287–296. ACM, 2011.

[Oga87] K. Ogata. *Discrete-time control systems*, volume 1. Prentice-Hall Englewood Cliffs, 1987.

[Oga01] K. Ogata. *Modern Control Engineering*. Prentice Hall PTR, 2001.

[OK98] D. W. Oard and J. Kim. Implicit feedback for recommender systems. In *Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83. Wollongong, 1998.

[PA11] D. Parra and X. Amatriain. Walk the talk: analyzing the relation between implicit and explicit feedback for preference elicitation. In *Proceedings of the 19th international conference on User Modeling, Adaption, and Personalization*, pages 255–268. Springer-Verlag, 2011.

[Pat07] A. Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop*, pages 5–8, 2007.

[PB07] M. Pazzani and D. Billsus. Content-based recommendation systems. *The adaptive web*, pages 325–341, 2007.

[PC09] S. T. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender Systems*, pages 21–28. ACM, 2009.

[PGH+02] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using Control Theory to Achieve Service Level Objectives In Performance Management. *Real Time Systems*, 23(1-2):127–141, 2002.

[PHC07] M. H. Park, J. H. Hong, and S. B. Cho. Location-based recommendation system using bayesian users preference model in mobile devices. *Ubiquitous Intelligence and Computing*, pages 1130–1139, 2007.

[PHLG00] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles. Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach. In *Proceedings of the sixteenth conference on Uncertainty in Artificial Intelligence*, pages 473–480. Morgan Kaufmann Publishers Inc., 2000.

[PPL01] A. Popescul, D. M. Pennock, and S. Lawrence. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *Proceedings of the seventeenth conference on Uncertainty in Artificial Intelligence*, pages 437–444. Morgan Kaufmann Publishers Inc., 2001.

[PT08] Y. J. Park and A. Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender Systems*, pages 11–18. ACM, 2008.

[PT09] I. Pilászy and D. Tikk. Recommending new movies: even a few ratings are more valuable than metadata. In *Proceedings of the third ACM conference on Recommender Systems*, pages 93–100. ACM, 2009.

[QLC+10] D. Quercia, N. Lathia, F. Calabrese, G. Di Lorenzo, and J. Crowcroft. Recommending social events from mobile phone location data. In *Proceedings of the tenth IEEE international conference on Data Mining*, pages 971–976. IEEE, 2010.

[RAC+02] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134, 2002.

[Ren10] S. Rendle. Factorization machines. In *Proceedings of the tenth IEEE international conference on Data Mining*, pages 995–1000. IEEE, 2010.

[RFGST09] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 2009.

[RGFST11] S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 635–644. ACM, 2011.

[RIS+94] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[RKK99] R. K. Ratner, B. E. Kahn, and D. Kahneman. Choosing less-preferred experiences for the sake of variety. *Journal of Consumer Research*, 26(1):1–15, 1999.

[RKR08] A. M. Rashid, G. Karypis, and J. Riedl. Learning preferences of new users in recommender systems: an information theoretic approach. *ACM SIGKDD Explorations Newsletter*, 10(2):90–100, 2008.

[RL03] I. Ruthven and M. Lalmas. A Survey on the Use of Relevance Feedback for Information Access Systems. *The Knowledge Engieering Review*, 2003.

[Roc71] J. J. Rocchio. *Relevance feedback in information retrieval*. Prentice-Hall, Englewood Cliffs NJ, 1971.

[RRS11] F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. *Recommender Systems Handbook*, pages 1–35, 2011.

[RS05] J. DM. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[RST08] S. Rendle and L. Schmidt-Thieme. Online-Updating Regularized Kernel Matrix Factorization Models for Large-Scale Recommender Systems. In *Proceedings of the ACM conference on Recommender Systems*, 2008.

[SF96] R. Schapire and Y. Freund. Experiments with a new boosting algorithm. In *Proceedings of the thirteenth international conference on Machine Learning*, 1996.

[SFHS07] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. *The Adaptive Web*, pages 291–324, 2007.

[SHG09] D. H. Stern, R. Herbrich, and T. Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World Wide Web*, pages 111–120. ACM, 2009.

[SKKR01] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 2001 international conference on World Wide Web*, 2001.

[SKR01] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data mining and knowledge discovery*, 5(1):115–153, 2001.

[SLH10] Y. Shi, M. Larson, and A. Hanjalic. List-wise learning to rank with matrix factorization for collaborative filtering. In *Proceedings of the 2010 ACM international conference on Recommender Systems*, pages 269–272, 2010.

[SM86] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1986.

[SMH07] R. Salakhutdinov, A. Mnih, and G. Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[SPUP02] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th international ACM SIGIR conference on Research and Development in Information Retrieval*, 2002.

[SS01] K. Swearingen and R. Sinha. Beyond Algorithms: An HCI Perspective on Recommender Systems. In *Proceedings of the SIGIR 2001 Workshop on Recommender Systems*, page 11. ACM, 2001.

[STC00] J. Shawe-Taylor and N. Cristianini. *Support Vector Machines*. Cambridge University Press, 2000.

[TH08] T. Tsunoda and M. Hoshino. Automatic metadata expansion and indirect collaborative filtering for TV program recommendation system. *Multimedia Tools and Applications*, 36(1):37–54, 2008.

[THW+11] Hanghang Tong, Jingrui He, Zhen Wen, Ravi Konuru, and Ching-Yung Lin. Diversified ranking on large graphs: an optimization viewpoint. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, volume 11, pages 1028–1036, 2011.

[TPNT07] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9(2):80–83, 2007.

[UY06] M. Utiyama and M. Yamamoto. Relevance feedback models for recommendation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 449–456. ACL, 2006.

[VB06] S. Vassilvitskii and E. Brill. Using web-graph distance for relevance feedback in web search. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 147–153. ACM, 2006.

[vR79] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[VSPK04] M. Van Setten, S. Pokraev, and J. Koolwaaij. Context-aware recommendations in the mobile tourist application compass. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 515–548. Springer, 2004.

[Wan09] J. Wang. Mean-Variance analysis: A new document ranking theory in information retrieval. In *Proceedings of the 31st European Conference on Information Retrieval*, 2009.

[WB06] J. Winn and C.M. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6(1):661, 2006.

[WBS08] 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, 2008.

[WdVR06] J. Wang, A. P. de Vries, and M. J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th international ACM SIGIR conference on Research and Development in Information Retrieval*, 2006.

[WKS08] M. Weimer, A. Karatzoglou, and A. Smola. Adaptive collaborative filtering. In *Proceedings of the ACM conference on Recommender Systems*, 2008.

[WRdVR08] J. Wang, S. E. Roberston, A. P. de Vries, and M. J. T. Reinders. Probabilistic relevance models for collaborative filtering. *Journal of Information Retrieval*, 2008.

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of small-worldnetworks. *Nature*, 393(6684):440–442, 1998.

[WZ09] J. Wang and J. Zhu. Portfolio theory of information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and Development in Information Retrieval*, pages 115–122. ACM, 2009.

[YCZ11] Q. Yuan, L. Chen, and S. Zhao. Factorization vs. regularization: fusing heterogeneous social relationships in top-n recommendation. In *Proceedings of the 2011 ACM international conference on Recommender Systems*, pages 245–252. ACM, 2011.

[ZC11] V. Zanardi and L. Capra. Dynamic updating of online recommender systems via feedforward controllers. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 11–19. ACM, 2011.

[ZGVGA07] Xiaojin Zhu, Andrew B Goldberg, Jurgen Van Gael, and David Andrzejewski. Improving diversity in ranking using absorbing random walks. In *The 2007 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 97–104, 2007.

[ZKL+10] T. Zhou, Z. Kuscsik, J. Liu, M. Medo, J. R. Wakeling, and Y. Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

[ZMKL05] C. N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, 2005.

[ZN93] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Journal of dynamic systems, measurement, and control*, 115(2B):220–222, 1993.

[ZSQJ12] Y. C. Zhang, D. Ó Séaghdha, D. Quercia, and T. Jambor. Auralist: introducing serendipity into music recommendation. In *Proceedings of the 2012 ACM international conference on Web Search and Data Mining*. ACM, 2012.

[ZWSP08] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. *Algorithmic Aspects in Information and Management*, pages 337–348, 2008.

[ZYZ11] K. Zhou, S. H. Yang, and H. Zha. Functional matrix factorizations for cold-start recommendation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 315–324. ACM, 2011.