Exploiting Latent Information in Recommender Systems

by

Yun Zhang

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2012

Abstract

This thesis exploits latent information in personalised recommendation, and investigates how this information can be used to improve recommender systems. The investigations span three directions: scalar rating-based collaborative filtering, distributional rating-based collaborative filtering, and distributional rating-based hybrid filtering.

In the first investigation, the thesis discovers through data analysis three problems in nearest neighbour collaborative filtering — item irrelevance, preference imbalance, and biased average — and identifies a solution: incorporating "target awareness" in the computation of user similarity and rating deviation. Two new algorithms are subsequently proposed. Quantitative experiments show that the new algorithms, especially the first one, are able to significantly improve the performance under normal situations. They do not however excel in cold-start situations due to greater demand of data.

The second investigation builds upon the experimental analysis of the first investigation, and examines the use of discrete probabilistic distributional modelling throughout the recommendation process. It encompasses four ideas: 1) distributional input rating, which enables the explicit representation of noise patterns in user inputs; 2) distributional voting profile, which enables the preservation of not only shift but also spread and peaks in user's rating habits; 3) distributional similarity, which enables the untangled and separated similarity computation of the likes and the dislikes; and 4) distributional prediction, which enables the communication of the uncertainty, granularity, and ambivalence in the recommendation results. Quantitative experiments show that this model is able to improve the effectiveness of recommendation compared to the scalar model and other published discrete probabilistic models, especially in terms of binary and list recommendation accuracy.

The third investigation is based on an analysis regarding the relationship between rating, item content, item quality, and "intangibles", and is enabled by the discrete probabilistic model proposed in the second investigation. Based on the analysis, a fundamentally different hybrid filtering structure is proposed, where the hybridisation strategy is neither linear nor sequential, but of a divide-and-conquer shape backed by probabilistic derivation. Experimental results show that it is able to outperform the standard linear and sequential hybridisation structures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The explosion of information available in our daily lives makes it impossible for humans to look at everything before making informed decisions. An overload of diffuse and intermingled data can also lead to confusion and compromise decision making. Therefore, it is desirable to have intelligent information filtering systems to strategically filter out unwanted information to aid our decision-making, or even make decisions for us. Existing examples include search engines such as *google.com* and *bing.com* that determine the importance of a web page by techniques such as counting its incoming and outgoing hyperlinks [24]; and movie rating sites such as *imdb.com* and *rottentomatoes.com* that indicate the quality of a movie using global average rating. However, these systems ignore the importance of individuality and personalisation in their information filtering processes, thus are unsuitable for tasks where there are large preference variations among individuals, such as the recommendation of food, movies, music, books, articles, and so on.

This thesis focuses on *recommender systems* (RS), which are intelligent systems designed to provide personalised recommendations based on the user's past activities and preferences. This chapter sets the theme for the thesis: section 1.1 provides overview and motivations; section 1.2 outlines the scope of this thesis; section 1.3 specifies research topics and questions; section 1.4 lists the contributions; section 1.5 provides a structural outline of this thesis.

1.1 Overview and Motivations

The task of personalised recommendation can be formulated as the task of estimating the personalised utility of an *item* to a *user*, the estimation of which can be used as guidance to improve the user's browsing efficiency and decision making. The recommendations can be made manually by human experts, known as *knowledge-based recommendation* [26], or automatically using machine learning algorithms. This thesis focuses on *automated* recommender systems of the latter kind, which are supervised machine learning systems that use training data such as the profile of the user, the properties of the item, the user's feedback on other items, and/or the preferences of other like-minded users to estimate the degree of preference of a user on an item.

Automated recommender systems have been widely adopted in real life to recommend movies [120], music [79], books [81], groceries [75], jokes [38], online news [110], research papers [20, 21, 67], recipes [145], program code [154], and so on. Successfully deployed recommender systems can greatly increase website traffic, system usability, and company profit [167, 170]. For example, digg.com reported significant increase in traffic and user activities after the deployment of their recommender system [170]; eBay reported that recommender systems can potentially increase the amount of products sold by 10% [167].

Recommender systems have also attracted great interest from the research field due to their high practicality, potential commercial gain, and the unique machine learning properties of the problem. In contrast to conventional classification or regression problems which simply model a mapping from the data points to classes or values, recommendation problems have *two* primary dimensions — *users* and *items* — which are mapped onto *ratings* or *rankings* representing the estimated value of the item to the user. This extra dimension on the left-hand-side of the mapping brings in a whole new set of *collaborative* strategies around the pivoted dimension (i.e. either user or item); the unique semantics of the right-hand-side (i.e. rating or ranking) also invites the exploitation of domain-specific heuristics.

1.2 Scope and Focus

The field of recommender system research is vast and diverse. Active research topics include interface design [30, 148], user psychology [88, 22, 74], security and networking [93], anti-spamming [96, 127], active learning [45], temporal diversity study [72], and contextual diversity study [3, 155]. However, one of the most fundamental topics is the study of *recommender algorithms*, which refers to the application, adaptation, and advancement of machine learning algorithms and heuristics to better accommodate recommendation problems; this is also the focus of this thesis.

There are many angles from which a recommender algorithm can be studied and improved. Most research focuses on the general improvement of recommendation accuracy and recommendation effectiveness, which is also the focus of this thesis. Other specialised studies include algorithm adaptivity [26], algorithm scalability [106, 156, 160], the algorithm's recommendation diversity [169, 67, 163, 1, 62], and the algorithm's sparsity handling ability [117, 134, 156, 152, 107, 84].[1] Although not the focus of this thesis, adaptivity, scalability, diversity, sparsity, and various other idiosyncratic aspects of recommender algorithms are surveyed and discussed in this thesis.

There are several directions to improve the general accuracy and effectiveness of recommender algorithms. These include introducing previously unemployed machine learning algorithms to the recommendation problem [120, 27, 17, 23, 12, 53, 51, 54, 123, 136], alternative ways of applying machine learning algorithms to recommendation [38, 129, 87], discovering recommendation-specific heuristics [47, 59, 152, 84, 70, 57], combining multiple machine learning algorithms to form a congregated recommender system [91, 159, 156, 107, 14], and utilising additional information sources [9, 91, 2, 26, 86].[2] This thesis focuses on discovering, through data analysis and logical reasoning, new recommendation-specific heuristics to improve existing recommender algorithms, with an emphasis on exploit-

[1] All citations in this paragraph are discussed in details in chapter 2.

[2] All citations in this paragraph are discussed in detail in chapter 2 as part of the literature review, or in sections 4.4 and 5.6 as closely related work.

ing latent and neglected concepts and information.

There are also many studies that *indirectly* improve recommender algorithms by conducting literature surveys that provide clarity and alternative perspectives for other colleagues in the field. Examples include general-purpose survey [2, 110, 143], empirical analysis [23, 102], logical analysis [89], recommender system taxonomies [98, 41, 26], and evaluation methods surveys [48, 88, 41]. This thesis also intends to contribute in this direction by providing literature reviews on recommender algorithms (sections 2.2, 2.3, 2.5), their taxonomies (section 2.1), comparisons (section 2.4), and evaluations (section 3.3).

1.3 Research Topics and Questions

The theme of this thesis is to investigate the recommendation problem and current techniques, exploit latent information, devise specialised heuristics, and apply them to improve recommender algorithms. Specifically, the research focuses on the following three topics:

1. *Nearest neighbour collaborative filtering*. This investigation looks at Pearson's correlation-based nearest neighbour collaborative filtering — one of the most popular recommender algorithms — and focuses on the following research question: are there any problems with this method? specifically, is there any latent or neglected information that can boost the performance of this method?

2. *Distributional rating-based collaborative filtering*. This investigation builds upon the experimental analysis of the first investigation, which observed the importance of the *range* and the *distribution* of ratings on top of their values. It focuses on the following research questions: what is the best way of modelling rating distribution in the recommendation process? where should distributional rating be used? and how should it be used?

3. *Distributional rating-based hybrid filtering*. This investigation expands the focus from collaborative filtering to hybrid filtering, which uses

items' content data as well as the rating data to make recommendations. It asks the following research questions: how does a user's rating correspond to the content of the item being rated? what does a user rating actually entail? what are the fundamental causes of content-based filtering's relatively poor performance compared to collaborative filtering? do the current hybridisation strategies — which are either linear or sequential — the optimal way of handling the "tangibles" (i.e. the item content) and the "intangibles"? are there latent or mishandled information in the process? and are there better ways of handling them?

1.4 Contributions

This thesis includes four major contributions:

1. The first contribution is the *target-aware similarity computation* or TASK algorithm. It is presented in chapter 4, and corresponds to the first research topic described in section 1.3 — nearest neighbour collaborative filtering. In it, two problems with the nearest neighbour method are identified. The TASK algorithm identifies the cause of both problems as the target identities being ignored in the similarity computation, and resolves the problems by incorporating the ignored information, consequently improve recommendation performances. A preliminary version of this work was published as Zhang and Andreae [166]. Since then, further improvements and more extensive evaluations have been made and are presented in this thesis.[3]

2. The second contribution is the *partial average and double average* or the PANDA algorithm. The algorithm also corresponds to the first research topic, but identifies latent information in the rating deviation computation stage instead of the similarity computation stage. The performance improvement of PANDA is not as prominent as TASK.

[3]Other than the TASK algorithm, which was published in [166], no attempts has yet been made to publish the other parts of this thesis. This could be an important future work for us.

But improvements are achieved by carefully tuning the application preconditions.

3. The third contribution is the *distributional rating-based nearest neighbour* or the DRNN recommendation process. It is described in chapter 5, and corresponds to the second research topic — distributional rating-based collaborative filtering. The main idea is to use a discrete vector-based probability distribution representation throughout the recommendation process, including the ideas of distributional rating input, distributional voting profile, distributional similarity, and distributional rating predictions. The richer distributional modelling structure enables the representation, preservation, computation, and communication of previously undiscovered concepts and ignored information, which are also identified in the thesis. This process is able to improve recommendation set and recommendation list-based accuracies.

4. The fourth contribution is the *Diamond* hybrid filtering framework. It is presented in chapter 6, and corresponds to the third research topic — distributional rating-based hybrid filtering. This framework is inspired and designed based on an analysis of the relationships between rating, item content, item quality, and the "intangibles". Its novelty and distinctiveness lie in its diamond-shaped hybridisation strategy and its reliance on the "distributional rating-based ecosystem" provided by the DRNN recommendation process. Experimental results show that this new hybridisation strategy is able to outperform the standard linear and sequential hybridisation structures.

Apart from the four major algorithm-based contributions, this thesis also contributes to the research community by providing a literature survey that addresses four aspects. They are:

5. A taxonomy of modern recommender systems, which include six not-necessarily-orthogonal dimensions (presented in section 2.1).

6. A review of recommendation algorithms, including collaborative filtering algorithms in sections 2.2, content-based filtering algorithms

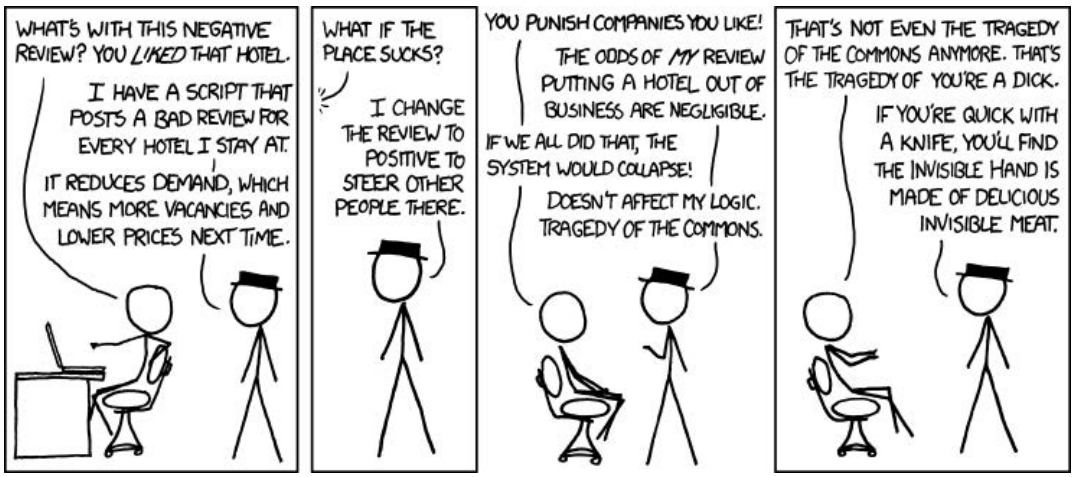
in section 2.3, and hybrid filtering algorithms in section 2.5.

7. A detailed comparison of the pros and cons of collaborative filtering versus content-based filtering (presented in section 2.4). The comparisons are done from various perspectives, including their data requirements and formats, recommendation abilities in various situations and scenarios, recommendation tendencies, and machine learning properties. This analysis is new and does not appear to have been specifically done before to this extent.

8. A review on the evaluation mechanisms for numerical rating-based recommendation algorithms (presented in section 3.3). This survey not only addresses the evaluation of rating-based algorithms in terms of their prediction error, which is the "natural format", but also addresses the procedure and metrics for evaluating the accuracy of rating-based algorithms in the binary set recommendation and the ranked list recommendation settings.

1.5 Thesis Outline

This thesis is organised as follows: chapter 2 provides literature reviews; chapter 3 sets out the datasets, evaluation metrics, and evaluation protocols used throughout the thesis; chapter 4 presents the first investigation — nearest neighbour collaborative filtering, along with the first two contributions — TASK and PANDA; chapter 5 presents the second investigation — distributional rating-based collaborative filtering, along with the third contribution — the DRNN recommendation process; chapter 6 presents the third investigation — distributional rating-based hybrid filtering, along with the fourth contribution — the *Diamond* hybrid filtering system. Finally, chapter 7 concludes the thesis and indicates future work.



The curse of individuality — one of the many difficulties we face as recommender systems.[4]

[4]This is an XKCD comic originally published at `http://xkcd.com/958/`. The "invisible hand" in the last block is coined by 18^{th} century economist Adam Smith as a figurative reference to the invisible force that coordinates the cooperation of individual interests to lead to the common good.

Chapter 2

Literature Review

This chapter provides a literature review of the related topics and concepts. Section 2.1 firstly outlines the field by presenting six different dimensions on which a recommender system can be classified, then delineates the scope of this thesis by specifying its "coordinates" within the six dimensions. The chapter then zooms into the dimension most important to this thesis — collaborative and content-based filtering — sections 2.2 and 2.3 survey the techniques of the two filterings; section 2.4 compares the respective strengths, weaknesses, and recommendation and learning properties of the two filterings; section 2.5 describes the hybridisation of the two filtering. Finally, section 2.6 summarises the chapter.

The intention of this chapter is to provide a holistic view of the field, with emphasis on the areas related to this thesis. It is not a "related work" chapter — closely related work of each investigation is presented in the investigation's own respective chapter.

2.1 Taxonomies of Recommender Systems

There are several taxonomies for classifying recommender systems. Schafer et al. [132] is an early, though slightly out-of-date survey that studied the classification of recommender systems; Montaner et al. [98] uses 37 systems to demonstrate 8 classification dimensions. Other publications that provide unique taxonomies include Gunawardana and Shani [41], which

presents a de facto evaluation-based classification; and Burke [26] that provides a knowledge source-based classification.

This section outlines six dimensions for classifying recommender systems. The six dimensions are chosen because they are the most rational, the most illuminating, the most helpful in placing this thesis within the field, and the least overlapping. Note that the classification dimensions are not necessarily orthogonal. For example, rating prediction systems are more widely used where there is explicit relevance feedbacks, content-based filtering algorithms are mostly model-based, and most model-based systems use a batch updating scheme.

2.1.1 (Implicit or explicit) relevance feedback

Recommender systems can be classified as *implicit* or *explicit* based on the way the training data is collected. *Explicit recommender systems* collect data by explicitly asking users to express their degree of preference on a subject. Examples include the *MovieLens* movie recommender [120] that solicits ratings on a discrete numerical scale; the *Jester* joke recommender [42, 173] that solicits ratings on a continuous numerical rating scale; the *Fab* system [9] that asks users to *rank* the documents forwarded to them; and multi-component RS [122] that operates on the Yahoo! Dataset [175], which collects multi-component ratings.

Implicit recommender systems collect data by unobtrusively tracking user activities in the background. The feedback can be a boolean value indicating a show of interest, or a numerical value representing the time spent examining the item. It greatly reduces the workload of users by eliminating the onerous job of rating items, thus is more user friendly and more applicable [164, 92, 145]. A hybrid of implicit and explicit rating collection is also gaining popularity, since interfaces that provide explicit data also embed collectable implicit data [98].

Since this thesis is algorithm-oriented instead of interface-oriented, the implicit and explicit dimension does not directly apply to it. However, the evaluations are performed on the *MovieLens* and *Jester* datasets, both of which provide explicit numerical rating feedbacks.

2.1.2 (Binary, ranked or rating) recommendation format

Recommender systems can be classified into three categories of *binary*, *ranked-list* or *rating-based*, based on the format of the recommendation feedback provided by the system. *Binary* recommender systems provide *yes-or-no* type recommendations suggesting either to accept or to reject an item. Such recommendations are often presented as an unordered list (*i.e.* a set) including all items considered acceptable. Because the final decision of users is often "hard" rather than soft, namely to accept or reject, binary recommender systems maximally reduce the workload for human users by directly presenting them with a recommended final decision. *Rank-based* recommender systems also provide a list of items, but order the items by their estimated value. It is left to the user to decide whether to accept or reject an item based on its ranking. *Rating-based* recommender systems provide an often numerical rating estimation of how the user would have rated the item if asked. This feedback format provides the most information, since a set of rating recommendations can be easily converted into rank-based or binary list recommendations, but not the reverse.

This thesis studies all three formats with an emphasis on the rating format. Chapter 5 also presents a different recommendation format called *distributional rating*. Since the recommendation format is directly linked to the evaluation metrics suited to evaluate the algorithm, further explanation and citations are presented in section 3.3.

2.1.3 (Memory- or model-based) knowledge representation

Recommender systems can be classified as *memory-based* or *model-based*, based on whether or not the system generates and maintains a generalised knowledge model from which recommendations could be drawn. Memory-based systems do not keep a model, but process the raw dataset every time a recommendation is requested. Model-based systems use the training data to compile a generalised knowledge model first, and answer recommendation queries solely from this learnt model.

Most memory-based methods adopt a nearest-neighbour-based implementation. Examples include [120, 84, 8] that apply nearest neighbour

over collaborative data, and the *daily learner system* [19] that applies nearest-neighbour over content-based data. Model-based methods have a wider variety of implementations, including classification [17, 94], clustering [23, 53, 156], latent variable-based probabilistic models [53, 112, 52, 60], dimensionality reduction methods [131, 119, 147], graph-based models [4, 54, 165, 136], and rule-based methods [101, 126] that utilise a wide range of machine learning algorithms such as naïve bayes [16, 94, 23], Bayesian networks [23, 46], Gaussian process [114, 100], and neural networks [56, 27, 123].

The classification of memory- and model-based recommendation was proposed by Breese et al. [23]. With the progress of recommendation technologies, the distinction has become blurred. As pointed out in [68], traditional memory-based methods have evolved to rely on rigorous models at least for caching purposes, whereas some model-based approaches have started to directly explore the rating dataset in order to improve accuracy. There have also been recommender systems that contain both memory-based and model-based components, such as [156, 112].

Chapter 4 of this thesis focuses on improving memory-based nearest neighbour algorithms. The rest of this thesis utilises both memory- and model-based knowledge representations.

2.1.4 (Online or batch) knowledge model update

Recommendation *tasks* can be classified as *online* or *batch*. In *online* tasks, training data are presented piece-by-piece; whereas in *batch* tasks, all training data are at the disposal of the recommender system from the start. Most real-life recommendation tasks that involve real users are online, whereas batch tasks are widely used in research studies.

Similarly, recommender *systems* can be classified as *online* or *batch* based on whether or not the system updates its knowledge model as data arrives. Memory-based algorithms are generally "online", since they do not keep a knowledge model, thus require no update as data arrives. Model-based algorithms are mostly "batch", but some batch models can be adapted to accept online updates [118].

Online algorithms may not be good for online tasks. In reality, online tasks often demand high recommendation speed, so the advantage of incorporating newly arrived data into the recommendation may give way to the recommendation speed if the two were in conflict, and quite often they are. This is because highly online algorithms such as memory-based collaborative filtering achieve their "*online-ness*" by leaving all the work to the recommendation stage, resulting in slow recommendation speed; whereas many model-based algorithms, although requiring a resource intensive batch learning stage, can recommend very fast due to the compactness of the model. To this end, it may be more practical for online tasks to adopt a fast-predicting model-based batch algorithm, while caching the incoming data for periodic update after new data has accumulated and when the computational power is less occupied, such as at night.

Therefore, although the algorithms proposed in this thesis all adopt a batch updating model, they can still be used in online tasks if so required. However, online recommendation is not the focus of this thesis, and all experiments are conducted under batch recommendation task settings.

2.1.5 (User- or item-based) model orientation

Since recommender systems have two primary dimensions — user and item — the recommendation algorithms can "pivot" around one of the dimensions, reducing the recommendation problem into a set of correlated classification problems. Recommender systems can thereby be classified as *user-based* or *item-based*, based on the pivoting orientation of the algorithm. The most distinctive implementation of the same algorithm with both orientations is nearest neighbour collaborative filtering, which is described in section 4.2 with both orientations clarified. Both user and item orientations are studied in this thesis.

2.1.6 (Collaborative or content-based) knowledge source

Balabanovic and Shoham [9] were the first to propose the classification into *collaborative filtering* (CF) and *content-based filtering* (CBF). They took

a very restricted interpretation, defining content-based filtering as systems that make recommendations to a user based solely on the content of items the user has rated before; and collaborative filtering as algorithms that make recommendation based solely on items other similar users have rated and "*one which does no analysis of the items at all*"[9]. As recommendation techniques advanced, especially with the proposal of item-based collaborative filtering by Sarwar et al. [129], this definition is no longer accurate.

This thesis adopts a strict information source oriented definition. Systems are defined as collaborative or content-based filtering solely based on the information source they base their recommendations on. Collaborative filtering systems are ones that base their recommendations solely on *collaborative data* — a collection of ratings from the target user and other users. Content-based filtering systems are ones that base their recommendations on ratings by the target user and the *content-based data*, which are attributes or models describing users or items, such as the cast of a movie item or the age of a user.

Due to the importance of this dimension to the thesis, further literature reviews are presented in dedicated sections, with the techniques of the two filterings reviewed in sections 2.2 and 2.3; their recommendation and machine-learning strengths and weaknesses compared in section 2.4; and *hybrid filtering* — the incorporation of both collaborative and content-based information sources — described in section 2.5.

2.2 Collaborative Filtering

Collaborative filtering (CF) refers to the class of algorithms that recommend items to users based purely on feedback from other users who have rated a similar set of items in the past. The underlying assumption is that users who agreed in the past tend to agree in the future — a socially sound assumption. Surveys that address collaborative filtering include [48, 102, 143, 23, 17, 127].

This section provides a literature survey of current CF techniques by classifying CF algorithms into memory-based nearest neighbour methods

(section 2.2.1) and four categories of model-based methods, which are matrix factorisation (section 2.2.2), clustering (section 2.2.3), latent variable-based models (section 2.2.4), and other model-based methods such as graph-based and rule-based methods (section 2.2.5).

2.2.1 Memory-based Collaborative Filtering

Memory-based CF [23, 120] refers to nearest neighbour-based collaborative filtering (NNCF) methods that calculate rating predictions by averaging over the corresponding ratings of neighbouring users, weighted by the *preference similarities* between the neighbours and the target user of the prediction task. A detailed procedure is outlined in section 4.2.

The core of NNCF lies in the *similarity measure*, which determines the similarity of two users by calculating the correlation of their past ratings. Existing similarity measures include *Pearson's correlation* [23, 120], *cosine vector similarity* [23, 34], *Spearman correlation* [47], *Euclidean distance* [47], *rating concordance* [71], *entropy-based uncertainty measure* [47], *probabilistic distance measure* [43], and *conditional probability-based similarity* [34]. Among those, empirical evidence suggests Pearson's correlation to be the top performer [23, 47], possibly because it captures both negative and positive correlations, unlike other correlation measures that only capture positive correlations.

The similarity measures listed above, especially the most prominent Pearson's correlation, have been enhanced using various heuristic "tweaks", including *inverse user frequency*, *variance weighting*, and *correlation significance weighting*, which are further elaborated in section 4.4.1. Chapter 4 also proposes two new heuristics for improving the Pearson's correlation-based similarity measure.

Sarwar et al. [129] proposed *item-based* NNCF method, which uses item similarities instead of user similarities to make predictions. Recent studies have also focused on combining user and item-oriented NNCF to improve prediction accuracy and combat data sparsity. Examples include [84, 152], details of which are elaborated in section 4.4.2.

In terms of prediction accuracy, NNCF methods work well once the

users have rated a large number of items [23], this is especially true with item-oriented methods [107]. Efficiency-wise, the memory-based nature of NNCF makes them efficient to learn but slow at producing recommendations. However, the recommendation speed can be improved with caching. The memory-based nature also makes them susceptible to poor scalability (section 2.4.4.1), dataset sparsity (section 2.4.1.2), cold-start problems (section 2.4.2.2), and the non-transitive association problem (section 2.4.2.4).

2.2.2 Matrix Factorisation

Matrix factorisation refers to the class of methods that map the user-item rating matrix[1] into two or more lower-dimensional matrices, from which predictions can be drawn. This class is renowned for their performance when the dataset is sparse. However, when the dataset is dense, excessive reduction in dimensionality could result in the loss of potentially useful information.

The most popular implementation of matrix factorisation is *singular value decomposition* (SVD), which approximates the rating matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$ as the product of two lower-ranking matrices $\mathbf{R} \approx \mathbf{PQ}$, where $\mathbf{P} \in \mathbb{R}^{m \times k}$ and $\mathbf{Q} \in \mathbb{R}^{k \times n}$ can be thought of as lower dimensional feature vectors for users and items respectively. The parameters of $\mathbf{P}$ and $\mathbf{Q}$ are estimated by minimising the sum-squared distance to the target matrix $\mathbf{R}$. This can be calculated using *latent semantic indexing* on an imputation-filled rating matrix [131], in which case it suffers badly from scalability problems; or it can be calculated using *gradient descent* [108, 146] or *alternating least squares* [13] on only the observed coordinates of the rating matrix. After the two lower-ranking matrices have been generated, the prediction queries can then be answered solely based on the inner product of the two matrices.

Other matrix factorisation-based implementations include Billsus and Pazzani [17], which uses SVD as a pre-processor to reduce the dimension of the rating matrix, it then creates a neural network for each of the re-

[1] The *user-item rating matrix* is a matrix whose rows correspond to users, columns correspond to items, and entries correspond to the rating of the indexed user on the indexed item. An abstract visualisation of the user-item matrix is presented in table 3.3.

duced dimensions. *Fast maximum margin matrix factorisation* (MMMF) [119] constrains the overall *strength* of the parameters instead of their size. *Eigen vector* [38] is a principal component analysis method that uses a small set of gauge items to represent orthogonal item dimensions. Sandvig et al. [127] uses matrix factorisation to calculate the similarity between user profiles based on reduced dimensions. Yu et al. [160] attempted to improve the scalability of SVD and PCA (*principal component analysis*) by using flexible factorisation dimensions.

2.2.3 Clustering

Clustering techniques group together users (or items) with similar rating patterns into *clusters*. Recommendations can then be made by consulting the ratings of other users in the same cluster(s). The algorithms can be *sharp* [23] or *soft* [151, 127, 53], based on whether or not the same user (or item) can belong to multiple clusters. They can also be user-oriented [23, 127], item-oriented [29, 136], or *two-sided* [53, 151], where both users and items are grouped into clusters.

Clustering-based recommendations tend to be less personalised due to their group-oriented nature, thus can be less accurate than highly personalised nearest-neighbour approaches [6]. However, accuracy can be improved by reducing cluster sizes. The group-oriented nature also leads to better learning scalability. Clustering has also been used in combination with other CF techniques to boost prediction accuracy and improve scalability. Connor and Herlocker [29] uses clustering to partition the problem space before applying nearest neighbour prediction within the clusters; Xue et al. [156] uses clustering as a pre-processing step to complete the missing entries of the rating matrix before feeding it to nearest neighbour.

Several heuristics have been proposed to generate the clusters. *K-means clustering* [127, 156] minimises the intra-cluster similarity variances; *hierarchical agglomerative clustering* [29] uses item similarities to group together items based on bottom-up pairing; *naïve Bayes clustering* [23] maximises the likelihood of the clusters using an EM (expectation-maximization) algorithm under the naïve assumption that a cluster's preference on a given

item is independent of the other clusters; *Gaussian mixture model-based clustering* [151] maximises the likelihood of the clusters modelled as Gaussian mixture models using an EM algorithm or Gibbs sampling; *hierarchical clustering-based ontology filtering* [136] uses conceptual clustering algorithms to learn a set of item ontologies, which are *directed acyclic graphs* or DAGs representing a set of hierarchical clusters of items. Probabilistic latent variable-based methods can also be considered as clustering, where the clusters are modelled by the latent variable(s) over user-item pairings instead of users or items. This class of methods is elaborated on in section 2.2.4.

2.2.4 Probabilistic Latent Variable Models

Probabilistic latent variable approaches introduce a set of latent variables that appear to "cause" or explain the observed ratings. A probabilistic dependency model is then formulated based on this causal structure around variables representing users, items, ratings, and the newly added set of latent variables. Probability distributions of this model can then be learnt using EM algorithms to maximise the likelihood of the observed data given the model. After the model is properly trained, it can be used to draw predictions of new user-item scenarios.

This family of methods has close ties with the clustering methods of section 2.2.3 and the matrix factorisation methods of section 2.2.2. The latent variable models can be thought of as latent variable-based clustering, with each of the latent variables corresponding to a probabilistic cluster. Many matrix factorisation methods can be thought of as latent variable models, with the entries of the lower-dimensional matrices being the latent variables that model the observed data. One distinctive feature of this family that neither clustering nor matrix factorisation possess is its strong probabilistic and causal semantics that allow statistical techniques to be used for the inference of the model.

The representative method of this class is the *aspect model* [53], which associates a layer of latent variables with each observed user-item pair. Users and items are assumed independent from each other given the latent

class variables. The latent variables can be thought of as clusters on users, items, *and* the correlations between users and items all in one. As a result, this method is able to produce more personalised recommendations than conventional clustering methods, since predictions are no longer identical for all users or items in the same cluster. The method is extended to incorporate item content information in [115, 134]; and is extended to support numerical ratings instead of categorical ratings in [52]. Other work that uses the aspect model includes [60, 45, 159].

Other probabilistic latent variable-based approaches include the *flexible mixture model* [140, 60, 45], which uses two layers of latent variables, instead of one layer as in the aspect model, to characterise the grouping of users and items separately; and *personality diagnosis* [112], which uses latent variables in a naïve bayes model to model user clusters representing user "personalities".

2.2.5 Graph-based, Rule-based, and Others

Other model-based collaborative filtering approaches include graph-based models such as *horting* [4], which builds user-noded DAGs that capture the predictability of ratings between users, and then makes predictions by walking through the graph. Huang et al. [54] uses a Hopfield network to retrieve and model the associations between users and items in a bipartite user-item graph. Breese et al. [23] encodes the dependencies between items using a Bayesian network with one decision tree at each node, which corresponds to an item of interest. Other solutions include neural networks [123], and rule-based solutions [126] that generate discrete association rules for groups of commonly liked items based on patterns of item co-occurrence across user profiles.

2.3 Content-based Filtering

Content-based filtering (CBF) [110] bases its recommendation for a new user-item scenario purely on how well the content of the target item matches the user's preferred content pattern, which is learnt from the user's own

past ratings and the content pattern of the rated items. To this end, the *content-representation*, which provides an abstract characterisation of the item content, becomes an essential part of every CBF system, since it defines how the items are to be perceived by the CBF learning algorithms.

Modern content-representations often use a set of feature attributes, each of which represents one key aspect of the item important for the capturing of user preferences. For examples, in movie recommendation, item attributes can include the movie's genre, synopsis keywords, dates, actors/actresses, directors, producers, editors, distributors, writers, plot keywords, and reviews [172]. For text document recommendation, items attributes can be a set of extracted keywords [9].

CBF essentially views the problem of recommendation as a set of per-user classification (or regression) problems, where the ratings are classes, the items are objects to be classified, and the item attributes are the building blocks of the classifiers. Generally speaking, research on CBF is not as active and techniques not as diverse as CF, as elaborated in Nanas et al. [99]. CBF methods are mostly model-based.[2] Existing implementations include naïve Bayes [16, 2], neural networks [27, 56], Gaussian processes [114, 100], belief nets [165], and decision trees [12].

CBF is user-oriented, since the recommendation process "pivots" around users and classifies items. Its item-oriented counterpart is known as *demographic filtering* [26], which pivots around items and generates per-item models, each of which uses demographic attributes such as the user's age, sex, location, language, occupation, and marital status as building blocks to characterise the demographic profile of the users who would prefer it. Demographic filtering has not been as successful as CBF in research or real-world applications due to the difficulty of gathering sufficient demographic data [26].

CBF has both strengths and weaknesses compared to collaborative filtering. CBF performs well in areas where content data is abundant and

[2]This thesis classifies memory-based CBF such as the *vector space model* [2, 16] as single module hybrid filtering (section 2.5.3.2). This is because in this class of methods, although the neighbourhood similarities are computed purely based on user or item content, the recommendation still uses collaborative rating data from other users or items.

pertinent to the preference of users, such as text document recommendation, and performs badly in areas where these conditions do not hold, such as cross-domain situations (section 2.4.2.3). CBF does not take into account any *collaborative* preference data gathered from *other* users, resulting in the recommendations being localised to the target user's own past experience (section 2.4.3 – serendipity). However, this also makes CBF algorithms more efficient since it needs to deal with less data, and makes it outperform CF when training data is scarce, especially in new-item situations (section 2.4.2.2). When data is sufficient, CF often outperforms CBF in terms of recommendation precision, whereas CBF has been reported to achieve marginally better recall (at the cost of worse precision) [12]. Detailed analysis of the pros and cons of CF and CBF on all the aforementioned aspects are elaborated in section 2.4.

2.4 Comparisons of the Two Filtering Approaches

Collaborative filtering and content-based filtering are two classes of recommendation techniques that possess fundamentally different inductive biases. When used in isolation, they exhibit different strengths, weaknesses, and properties. This section analyses the two filterings from four perspectives: section 2.4.1 discusses their data requirements in terms of the availability, density, and quality of rating data and content-based data; section 2.4.2 discusses their recommendation abilities under general situations and special scenarios such as cold-start, cross-domain, and non-transitive association; section 2.4.3 discusses their ability to incorporate factors such as novelty, user subjectivity, item properties, and item quality in the recommendations; and section 2.4.4 discusses their machine learning properties in terms of scalability, adaptivity, and explainability.

2.4.1 The Data Requirements of CF and CBF

2.4.1.1 The requirements on rating data

Like all supervised machine learning algorithms, the availability, quantity, and quality of the training data is a major factor directly linked to the learning efficacy and recommendation accuracy of both collaborative and content-based filtering. It has been suggested in [26] that when rating data is extremely scarce, human expert-constructed recommendations, which otherwise would be considered limited and biased, can often outperform learning-based CF and CBF methods.

CF and CBF have different requirements on the source of training data. Collaborative filtering makes recommendations by incorporating the wisdom of the crowd, and requires a reasonable number of ratings not only from the target user of the prediction, but also other users in the dataset. Content-based filtering makes recommendations solely based on the ratings provided by the target user, and totally ignores the ratings of the crowd. Therefore, although it is immune to an insufficiency of collaborative data, it does require its only data source — the target user — to have rated a significant number of items.

2.4.1.2 The requirements on the sparsity of rating data

The *sparsity* of a rating dataset is defined as the density of the vacancies in the user-item rating matrix, as in equation 2.1.

$$\text{sparsity} = 1 - \frac{\text{\# ratings}}{\text{\# users} \times \text{\# items}} \tag{2.1}$$

Because most users would have only rated a very small portion of the items, most recommendation datasets have very high sparsity, with sparsity as high as 0.95 considered normal [129].[3] For this reason, in the field of recommender systems, the term "*sparse*" only refers to the extreme cases where the sparsity of the dataset is higher than 0.99. Such extreme sparsity

[3] Table 3.1 in chapter 3 (Evaluation Settings) lists the statistics of the recommendation datasets used in this thesis along with their sparsity values.

normally occurs when new recommender systems are first established, or in datasets where the number of users is small relative to the volume of information in the system due to either a large number of items or regular updates of the item pool or both.

Many publications include evaluations on sparse datasets in order to demonstrate the performance of their algorithms under start-up situations and other similar conditions [115, 104, 54, 40]. These studies normally use a standard recommendation dataset such as MovieLens, and achieve high sparsity by deliberately excluding training data.

High dataset sparsity has no effect on CBF given sufficient number of ratings from the target user, but can greatly reduce the performance of CF, since it is harder to extract collaborative information from a sparse matrix. There have been several heuristics proposed to improve the performance of CF under extreme sparsity. Existing solutions can be classified into explicit rating solicitation, dimensionality reduction, smoothing methods, graph-based methods, "fill-in-the-gap" methods, and hybrid filtering. Note that the classes may not be orthogonal. For example, the "fill-in-the-gap" method proposed in [91] is also hybrid filtering.

A typical *explicit rating solicitation* methods is *active learning* [45], which reduces the sparsity of the dataset by actively calculating the minimum set of queries that would be sufficient to learn the user's preference pattern, and then explicitly querying the user. *Dimensionality-reduction* methods [119, 38, 131] generalises over unrepresentative or insignificant users or items so that the user-item matrix is condensed. *Smoothing mechanisms* [152, 84] combine multiple prediction sources produced by different CF mechanisms to increase the likelihood of a prediction being produced. *Graph-based methods* deal with sparsity structurally. For example, Aggarwal et al. [4] implements a graph-based model that represents user similarities, and makes predictions by walking through the graph, thus is able to simply skip users with no data instead of being stopped by them; Schickel-Zuber and Faltings [136] uses ontology modelling to provide priors and hierarchical generalisation in the recommendation process to combat sparsity.

"Fill-in-the-gap" methods alleviate the lack of data caused by sparsity by

essentially exploiting a bigger part of the rating matrix for every prediction. The idea is to introduce a preprocessing step that fills in the missing entries of the rating matrix before feeding it to the main CF algorithm that produces recommendations, so that the transitive associations (section 2.4.2.4) that would otherwise be lost due to a lack of data connectors can be established through artificial data. Heuristics have also been proposed to only fill in the gaps considered constructive to the recommendation accuracy of active users [84]. Methods for generating artificial filler ratings include using a static "default voting" [23], using item-based CF to generate fillers for user-based CF [115], using CBF to generate filler to be later used by CF [91], using filler ratings provided by user clusters [156], using recursive prediction to refine the filler [162], using a set of common machine learning classifiers such as decision tree, logistic regression, naive Bayes, neural networks, one rule, decision list, and support vector machines to fill in the missing data [144].

Hybrid filtering [115, 91, 107] is one of the most effective ways to deal with data sparsity, due to CBF's immunity to the sparsity of the collaborative data, and the fact that it is always helpful to exploit other data sources (i.e. the content data source) when data from one source is scarce. However, it does require the availability of content data. A detailed survey of hybrid filtering algorithms is presented in section 2.5.

2.4.1.3 The requirement on item's content data

The availability and the quality of content data is the biggest factor that hinders the applicability and the performance of content-based filtering.

There are three sources where the content data can be obtained — high-level feature extraction, low-level feature extraction, and manual construction. Some domains such as text-based recommendation of documents and news articles have highly-extractable high-level features that can be easily parsed into representative content attributes. These are also the domains where CBF has the biggest advantages over CF. Other domains such as multimedia recommendations of music, movies, and images only have

low-level features that are harder but not impossible [77] to extract.[4] Content data can also be manually added by human experts, such as the genre information of a movie item. However, this is not only expensive, but can also add bias [138, 2].

The *quality* of content data refers to how pertinent the content representation is at sufficiently capturing user interests. For example, "leg room" and "ticket price" would be quality content for airline recommendation, but the "seat colour" is not. Since the content-representation defines the building blocks of CBF algorithms, it provides a significant inductive bias that can greatly affect the accuracy of CBF recommendations. The insufficiency of content attributes is considered one of the main reasons why CBF is not as successful as CF [99]. Pilászy and Tikk [113] also argues that in movie recommendation, CF with only 10 ratings on a new movie is better than pure CBF due to the large gap between movie descriptions and the movies themselves — people rate movies, not their descriptions.

There have been studies focusing on developing a richer content representation [137], but this is not the focus of this thesis. Instead, chapter 6 describes a new algorithm partly aspired by better utilisation of the existing content-representation.

2.4.2 The Recommendation Abilities of CF and CBF

2.4.2.1 Recommendation in general situations

Due to the richness of most collaborative data and the inadequacy of content representations given current technology, pure CF methods generally outperform pure CBF methods in terms of their prediction accuracies in general recommendation scenarios [99]. However, CBF methods are able to make faster predictions due to their model-based and non-collaborative (per-user) nature. Basu et al. [12] have also argued that CF techniques tend to produce precision-oriented results, whereas CBF tend to produce recall-oriented results.

[4] Nanas et al. [99] pointed out that, since user interests are not captured completely by free language expressible terms, lower level features have the potential to be as informative or more as higher level ones.

2.4.2.2 Recommendation in cold-start situations

The *cold-start problem* [134, 78] refers to the situations where a recommendation query targets a new or obscure user or item, on which there are no or very few existing ratings to consult on. It can be classified as the *new-user problem* [117] or the *new-item problem* [134, 91, 105] based on which orientation is experiencing the cold-start.[5] The new-user problem is ubiquitous across recommendation domains, stages, and techniques. The new-item problem is especially prevalent in dynamic domains.

CBF does not suffer from the new-item problem, because it relies on the item's content attributes instead of other users' ratings to make recommendations, therefore having a brand new target item with no prior ratings makes no difference. It does however suffer greatly from the new-user problem, because the user's own ratings are its only source of training feedback, and a brand new user with no prior ratings would deprive it of its only data source.

CF does not suffer from the new-user problem as badly as it suffers from the new-item problem. This is because it can simply offer the item's average rating across all users as the recommendation to new users, but recommending the user's average rating to new items does not make as much sense. On top of that, CF is more effective at building user models than CBF when there are only a small number of user ratings because of its ability to take advantage of collaborative ratings from other users.

Existing solutions of the cold-start problem can be classified into four groups. The first group is to *take initiative*; examples include trying to determine the most-informative items for a new user to rate based on heuristics such as item statistics [117, 158], with the purpose of breaking free of the new-item situation with a minimum number of elicited ratings. The second group is *manual interference*; examples include expert-based manual recommendation instead of learnt predictions [26], and the employment of "professional raters" to examine new items as soon as they arrive [8]. The third group is *higher-level generalisation*; examples include

[5] The new-user problem is also known as the *first-rater problem* [91]; the new-item problem is also known as the *recurring startup problem* [73].

non-personalized recommendations [107], and the use of prior knowledge [159, 164] in cold-start situations. The fourth group is the exploitation of wider knowledge sources, which follows the philosophy of incorporating information from different data sources when data from one source is scarce. Most solutions to the cold-start problem belong to this group. Examples include hybrid collaborative and content-based filtering, hybrid user and item-based methods [80, 152], and the exploitation of user's social network information [85].

2.4.2.3 Recommendation in cross-domain situations

In recommender systems, a *problem domain* refers to a specific subject area where items are comparable in their content and can be captured by a common content representation. Popular problem domains include movie, music, books, jokes, and news. Sometimes, recommender systems need to recommend to users who are interested in multiple problem domains. This is known as the *cross-domain situation* [99].

In cross-domain situations, items of different domains do not share a common content representation, thus cannot be compared, filtered, and recommended based on their content descriptions. CBF in this case is severely compromised to the extent of not being applicable, whereas CF is not affected at all because it does not rely on content data.

2.4.2.4 Recommendation in non-transitive association scenarios

Memory-based CF has an inherent machine learning weakness caused by its dependence on the *transitive associations* between users and items. In it, users can only be linked through their common items, and items only linked through their common users. If two items have never been rated by the same user, or two users have never rated the same item, it is not possible to determine whether the two items or users are similar. This is known as the *non-transitive association problem* [63], which is one of the main cause of CF's poor performance in cold-start scenarios and sparse datasets.

The remedies include *fill-in-the-gaps* methods that preserve associations

through pseudo-ratings generated prior to the learning (as described in section 2.4.1.2). Another solutions is hybrid collaborative and content-based filtering, in which case the item relationships lost by the lack of transitive associations in collaborative data can potentially be recovered by content-based filtering, if the items share common content-based traits.

2.4.3 The Recommendation Tendencies of CF and CBF

Apart from the different recommendation abilities under different situations, CF and CBF are also different in terms of their tendency to incorporate factors such as novelty, user subjectivity, item properties, and item qualities in the recommendation.

Since the recommendation process of CBF revolves around item contents, it has a natural advantage at capturing user preference regarding the property and content of items. CF on the other hand does not have this nature advantage. However, if the user's preference of certain item properties or contents is consistently reflected in his or her ratings, CF can still pick it up implicitly.

CBF's dedicated attention to item content is a double-edge sword. While enabling it to easily capture user's preference on item content, it also constrains it to only be able to recommend items that are similar in content to the ones the user has already seen. This is sufficient if users have specific interests in mind and are only looking for related recommendations, but may not be useful when "out-of-the-box" discovery is of interest, such as recommending fiction novels based on the programming books a user likes. This is known as the *serendipity problem*, and was first pointed out in [138]. Three groups of remedies have been proposed — add collaborative filtering [1], add randomness [139], and add into fitness [163], which incorporates the requirement of diversity directly into the optimisation problem. Related research include [67, 62].

A more dire accusation is that CBF's dedication to item content renders it incapable of capturing user's subjective view and item's quality [9, 73], which, if true, would be an inherent problem that greatly affect CBF's real-life applicability even when users are happy to stick with rou-

tine contents. I believe this inability is not due to the inherent flaws of the CBF algorithms, but the general lack of content representation broad enough to sufficiently categorise the "subjectiveness" of users' views and the quality of items. This new perspective is fundamental to the new hybridisation structure proposed in chapter 6, and is further elaborated in section 6.1.1.

2.4.4 The Machine Learning Properties of CF and CBF

2.4.4.1 The learning scalability

The *scalability* of a recommender system [106, 129] refers to the system's ability to cope with the growth of users, items, and ratings, in terms of computational complexity, processing efficiency, and other resource requirements. It is vital to the system's applicability in the real world, where dealing with a large and growing user and item base is often required.

Recommendation algorithms that depend on less data generally have a slower increase in demand as the system grows, thus tend to have better scalability. To this end, CBF generally has better scalability than CF, due to the fact that the content-based data is normally more compact, and that pure CBF only needs to process ratings from the target user alone.

Given the same data load, the scalability of an algorithm depends on the degree of generalisation, partition, and reuse implemented by the algorithm. To this end, memory-based CF has worse scalability than model-based CF. Among model-based CF, matrix factorisation is on the lower end of the scalability scale, and clustering algorithms on the higher end. Algorithms that allow online updating also have better scalability than batch algorithms due to the fact that they allow the models of newly observed ratings to be built on top of the previous model, thus vastly improving scalability through reuse.

2.4.4.2 The learning adaptivity

Scalability measures the performance of a system with the expansion of the training dataset, whereas *adaptivity* measures the system's speed to adapt

to new data with the update of training data. When data is sufficient, it is generally preferable for the recommendation to comply with more recent data to capture the evolution of user interests. A system with low adaptivity may keep recommending baby toys to a mother after her child has grown up.

High adaptivity is not always welcome, as it may compromise the *stability* of the system and opens the door for spamming and injection attacks. This is known as the *plasticity vs. stability* dilemma [26] — a key topic in anti-spamming and system robustness research [127, 97, 96, 90]. Generally, adaptivity and stability can both be improved at the same time through model-based methods with a high-level of abstraction. To this end, model-based CBF generally has better adaptivity as well as stability, whereas memory-based CF is considered to have the poorest stability [127]. Another way to improve adaptivity is temporal decay [18], where long-term information can be partially preserved using priors [164].

2.4.4.3 The explainability of the learnt model

Explainability is a general machine learning concept. On top of good learning accuracies, it is also preferable for the learnt structure to be understandable by human. As [68] points out, explainability is of special importance to recommender systems, as a recommendation with an intuitive reasoning such as "you will like this because ..." can improve the user's confidence and reliance on both the recommendation and the system. Related studies include [148].

Rule-based methods generally have the best explainability by nature. Between the two filterings, CBF has an edge in this aspect because of the structured nature of the content-based data. However it is not always the case, as a neural network-based CBF system would still suffer from poor explainability.

2.5 Hybrid Filtering

As explained in section 2.4, CF and CBF have a rather complementary set of strengths and weaknesses in terms of their operational requirements, recommendation abilities, and machine learning properties. Modern research therefore developed *hybrid filtering* techniques in order to alleviate the weaknesses of CF and CBF. The earliest hybrid filtering system dates back to the Fab system in the late 90s [9]. Since then, hybrid filtering has flourished, with modern techniques surveyed in [26, 2].

As specified in section 2.1.6, this thesis adopts an information source-oriented definition of CF and CBF. Collaborative filtering is defined as techniques that base their recommendations solely on collaborative data, whereas content-based filtering is defined as techniques that base their recommendations solely on content-based data. Hybrid filtering thereby refers to techniques that utilise both collaborative and content-based information to make recommendations. This is different from early publications that adopt a procedure-based definition of hybrid filtering, ruling systems that utilise both information sources but process them separately as non-hybrid [28].

The core of a hybrid system is its *hybridisation strategy* [25], which refers to the mechanism that coordinates the CF and CBF components to make cooperative recommendations. This section surveys the existing hybridisation strategies, and classifies them into three categories of combinatorial hybridisation, sequential hybridisation, and non-communicative hybridisation in sections 2.5.1, 2.5.2, and 2.5.3 respectively.[6]

2.5.1 Combinatorial Hybridisation

As shown in figure 2.1, combinatorial hybridisation strategies operate collaborative and content-based modules separately to produce independent

[6]Note that a hybrid system can implement more than one hybridisation strategy. For example, the Fab system [9] implements a sequential hybridisation strategy using recommendations as its hybridisation media (section 2.5.2.1). However, its CF component uses both collaborative and content-based information to compute peer users, implementing a single module non-communicative hybridisation strategy (section 2.5.3.2).

recommendations, which are then combined through rating integration or list merging to form the final recommendation. Existing rating prediction-based combinatorial hybridisation strategies include weighting, switching, voting, and stacking.

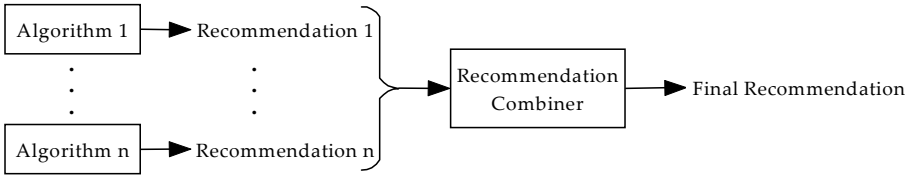


Figure 2.1: The Combinatorial Hybridisation Structure

2.5.1.1 Weighting

The *weighting*-based combinatorial hybridisation strategies [26] compute the final rating prediction as a weighted average of the predictions independently produced by separate CF and CBF modules. The weighting scheme can be static or dynamic. *Static weighting schemes* as in [28, 14][7] determine the weights by calculating the empirical means on a validation set. The weights are not altered upon assignment (i.e. static), thus is only suitable where the relative power of the modules is consistent over the problem space [26]. *Dynamic weighting schemes* [28, 10] implement an adaptive weight-assignment method, allowing the weights to change as conditions change. The simplest example is a piecewise function that assigns a different weight-balance to different users. Another common heuristic is to bias the weights towards CBF modules at the start when data is scarce, and gradually adjust towards CF modules as more data is collected.

2.5.1.2 Switching

Switching [26, 3] implements a conditional *winner takes all* scheme, in which the hybrid system switches between its modules based on the context, and the chosen module takes over the recommendation completely. This

[7] [14] implements a weighted combination of multiple CF algorithms with no CBF components, thus is not "hybrid filtering" per se, but its "hybridisation strategy" is similar to that of static weighting-based hybrid filtering.

scheme is suitable where different modules possess distinctive strengths in different parts of the problem space. The switching criteria can either be statically predetermined, or dynamically decided based on certain recommendation quality metrics [3], such as the "degree of confidence" used by [18], or the "consistency with past user ratings" used by [149].

2.5.1.3 Voting

Voting schemes are essentially a variation of weighting and switching, with the weighting or switching formulated as a voting or a bidding problem. Examples include [111] that proposed a voting scheme to combine list recommendations, and [153] that implements a market-based approach where different filtering algorithms bid with each other under various heuristics to offer their recommendations to the users.

2.5.1.4 Stacking

Instead of pre-determining the way of combining the base recommenders, the *stacking* scheme [10] uses a meta-learner to learn the optimal combination pattern. It treats the combination problem as a classification problem that maps the recommendation outputs of all the base recommenders to a final recommendation. The training data of the meta learner is provided by cross validation.

2.5.2 Sequential Hybridisation

As shown in figure 2.2, sequential hybridisation strategies stream two or more CF and CBF modules into a sequence. The modules are then applied one after another, with the output of the earlier module guiding the learning of the module following it. The final recommendation is produced by the last module in the sequence.

Sequential hybridisation strategies can be classified based on the type of information being passed down the sequence, which can be recommendations (section 2.5.2.1), feature vectors of users or items (section 2.5.2.2), or abstract knowledge models (section 2.5.2.3). The strategies can also be

classified based on the ordering of the CF and the CBF modules in the sequence, as described in sections 2.5.2.4.

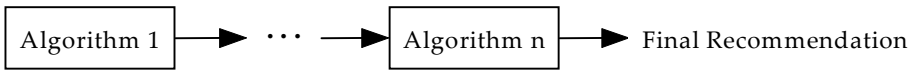


Figure 2.2: The Sequential Hybridisation Structure

2.5.2.1 Recommendations as hybridisation media

The most straightforward form of sequential hybridisation is to pass down recommendations. In this type, each individual module in the sequence is a recommender by itself. The recommendations made by the earlier module is passed down to the next module to be further refined [9], to serve as tie-breakers [25], or to be used as additional training data [91, 159, 130, 39, 107, 31].

For example, in the *Fab* system [9], both the items gathered by content-based *collection agents* and the items recommended collaboratively by the neighbouring users are passed into the content-based *selection agent* of the user to be further filtered before being recommended to the user. *EntreeC* [25] implements a "cascading" strategy that assigns priorities to recommendation modules, with the lower priority ones serving as tie-breakers in the scoring of the higher priority ones. *Rating augmentation* [91, 159, 31] uses content-based predictions to complete the missing entries of the rating matrix, which is then used by a CF module to make the final recommendations. *Filterbots* [39, 107] expand the user pool with fictitious users called *filterbots*, whose item ratings are provided by content-based recommendation modules solely based on the content data of the items. The filterbots are then treated as ordinary users in the recommendation process of the main collaborative recommendation modules.

2.5.2.2 Features vectors as hybridisation media

Passing down feature vectors, or *feature augmentation* requires the earlier algorithm to serve as a feature extraction mechanism to create or expand

the existing feature set describing items or users. The augmented features are then fed into the main recommendation algorithm to produce the final recommendation.

Examples that use CF as a feature extractor and CBF as the final recommender include [104] and the *Ripper* system [12]. Both systems use association rule mining over the collaborative data to derive new features, which are then fed into a rule induction-based CBF recommender. Another example is [113], which firstly applies matrix factorisation (section 2.2.2) on the rating matrix to generate a feature matrix representing a set of feature vectors for the movie items. A CBF module is then used to find an approximated linear transformation that transforms the item content features to the CF-induced feature vectors. This system, once trained, is able to make collaborative recommendations on new items that do not yet have user ratings.

Examples that use CBF as a feature extractor and CF as the final recommender include [124, 111]. In [124], CBF techniques are used to generate a set of intermediate ratings corresponding to the different aspects of the item's content representation, such as a rating for each actor or each film genre. CF is then applied on the generated content ratings to produce collaborative recommendations.[8] In [111], a content-based naïve Bayes technique is used to build feature vectors representing users' preferences of restaurants. CF is then applied to the feature vectors to identify peer users and produce recommendations.

2.5.2.3 Knowledge models as hybridisation media

In this class, the *knowledge model* learnt by the earlier module is fed as input into the next module, which then produces the recommendations. The difference between a knowledge model and a feature set (as described in the previous section) is that a knowledge model does not directly correspond to a partial aspect of users or items.

[8]The hybridisation media in this case are feature vectors but not recommendations, because the intermediate ratings that get passed down the sequence reflect the user's preference on a general concept, or a pre-defined cluster, instead of on individual items.

Examples include [64], which uses the item-item similarity matrix as the hybridisation media. In this system, a CBF module that applies k-means clustering over the item contents is applied to calculate item similarities, which are then combined through weighted sum with the item similarities calculated using conventional CF methods to form the final similarity matrix to be used in an item-based CF recommendation setting. Another example is [109], which firstly builds a directed tree structure called a *conceptual graph* for each user based on content-based data. Peer users are then identified by comparing users' conceptual graphs, and used in a CF setting to make the final recommendations.

2.5.2.4 The ordering of CF and CBF

The previous three sections defined three classes of sequential hybridisation strategies based on the different hybridisation media being passed along the module sequence. An alternative classification dimension is based on the order in which CF and CBF modules are applied.

Content-then-collaborative sequential hybridisation strategies refer to the scenarios where CBF comes before CF in the module sequence. CF would serve as the main recommendation engine, whereas CBF serves as a pre-processing auxiliary helper. Examples include using CBF to provide additional artificial training data [91, 159, 130, 39, 107, 31] (section 2.5.2.1), as a feature extractor [111, 124] (section 2.5.2.2), or as an abstract knowledge extractor [109, 64] (section 2.5.2.3).

Collaborative-then-content sequential hybridisation strategies refer to the scenarios where CF comes before CBF in the module sequence. CBF would serve as the main recommendation engine, whereas CF serves as a pre-processing auxiliary helper. Examples include using CF as a feature extractor [12, 104, 113] (section 2.5.2.2), or as a recommender system whose recommendations are to be further refined by a content-based filter [9].

2.5.3 Non-communicative Hybridisation

Non-communicative hybridisation strategies refer to scenarios where both collaborative and content-based information sources are explored, thus qual-

ify as hybrid filtering, but there is no explicit hybridisation step of the CF and CBF modules in the filtering process. This class can be further divided into two subclasses of *parallel hybridisation* and *single module hybridisation*, which are described in sections 2.5.3.1 and 2.5.3.2 respectively.

2.5.3.1 Parallel Hybridisation

As shown in figure 2.3, *parallel hybridisation* refers to the scenario where a mixture of collaborative and content-based recommenders are implemented to produce independent recommendations, which are all presented to the end user without further attempt to combine the results. It is also known as the *mixed* hybridisation strategy [26], and is proposed mainly to combat sparsity. Published implementations include ProfBuilder [5] and the PTV system [142].

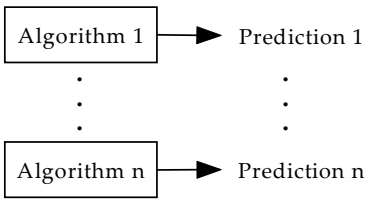


Figure 2.3: The Parallel Non-communicative Hybridisation Structure

2.5.3.2 Single Module Hybridisation

As shown in figure 2.4, *single module hybridisation* refers to the scenario where both collaborative and content-based information sources are explored in a *single* recommendation setting. There are no distinctive CF and CBF modules, thus is a type of non-communicative hybridisation.

Many systems of this class are built upon the idea of adapting collaborative filtering techniques to incorporate both collaborative and content-based data. Examples based on nearest neighbour collaborative filtering (section 2.2.1) include using the correlations between the profiles of users' preferred document keywords to calculate user similarities [9], or using the correlations between the content attributes of items [2, 16] or noise-reduced feature vectors of items [95] to calculate item similarities. Such

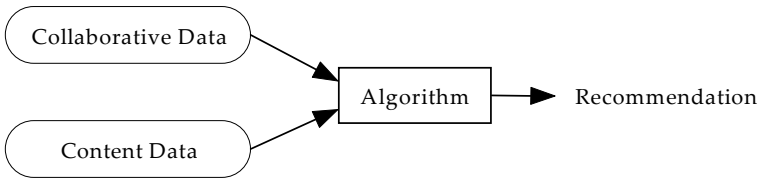


Figure 2.4: The Single Module Hybridisation Structure

content data-induced similarities are then used to identify peer users or items in a nearest neighbour CF recommendation setting.

Examples based on matrix factorisation (section 2.2.2) include [15], which proposed a "distributive" recommender system by partitioning the rating matrix into several smaller matrices, each corresponding to a content-based concept of the problem domain (e.g. movie genre). These content-derived matrices form several distributive collaborative filtering systems, which then communicate with each other to make the final recommendations. Another example is [141], which tries to integrate the relations between user, item, movie genre, and movie actor using the matrix factorisation method.

Examples based on clustering (section 2.2.3) include [115, 134], which proposed a probability-based clustering method that extends the aspect model-based CF method [53] (also see section 2.2.4) by using both collaborative and content-based data to build the aspect models, through the learning process of which a balanced cooperation between collaborative and content-based data is reached.

Other examples include [11], which features a support vector machine-like model with a kernel function based on a combination of collaborative ratings and content-based feature attributes; and *preference network* [150] that models content-based and collaborative data in a single conditional Markov random field.

2.6 Summary

The purpose of this chapter is to present a general review of the field, to clarify the scope of this thesis, and to provide new perspectives to various concepts. It was written with the hope that it could be a stand-alone contribution on its own, and could be useful as a general reference for other colleagues in the field. It is not a "related work" chapter — closely related work for the newly proposed algorithms will be presented in the algorithms' own respective chapters.

This chapter has three main components — the taxonomies of recommender systems (section 2.1); a comprehensive comparison of the pros and cons of CF and CBF (section 2.4); and a survey on the existing CF, CBF, and hybrid filtering techniques (sections 2.2, 2.3, and 2.5).

- Section 2.1 outlines six recommender system taxonomies based on the system's input format, output format, knowledge representation structure, knowledge updating model, (user or item) orientation, and (collaborative or content-based) knowledge source. Table 2.1 summaries the coordinate of this thesis within the six dimensions.

- Sections 2.2, 2.3, and 2.5 provide literature surveys on the techniques used in collaborative filtering, content-based filtering, and hybrid filtering respectively. Section 2.2 covers collaborative filtering techniques by classifying them into memory-based methods, matrix factorisation methods, clustering methods, probabilistic latent variable model, and graph and rule-based models; section 2.3 clarifies the importance of "content representation" in content-based filtering; section 2.5 organises hybrid filtering systems by their hybridisation structures, and classifies them into combinatorial hybridisation (include weighting, switching, voting and bidding), sequential hybridisation (include using recommendation, feature vectors, or knowledge models as sequential hybridisation media), and non-communicative hybridisation (include parallel and single module hybridisation).

- Section 2.4 presents a thorough and comprehensive comparison of CF and CBF from various aspects, including their data requirements

(in terms of rating data, content data, and data sparsity), recommendation abilities (in general situations, cold-start situations, cross-domain situations, and non-transitive association situations), recommendation tendencies (in terms of the system's ability to incorporate item content, item quality, item novelty, user subjectivity, and user individuality), and machine learning properties (in terms of scalability, adaptivity, and explainability of the learnt model). Although most of the recommender system publications do mention the differences between CF and CBF, based on my research, a comprehensive comparison like this has never been done before. Hopefully this survey could serve as a useful and compact information source for other collagues in the field.

Dimension	Section	This Thesis
(Implicit or explicit) Relevancy Feedback	2.1.1	The experimental datasets provide explicit rating feedback, although chapter 5 does refer to implicit rating in its supporting arguments.
(Binary, ranked or rating) Recommendation Format	2.1.2	Numerical ratings are used as the primary recommendation format. However, binary and ranked-list recommendations are also formulated to test the power of the proposed methods. Chapter 5 also proposes a different recommendation format — distributional ratings.
(Memory- or model-based) Knowledge Representation	2.1.3	Chapters 4 and 5 are memory-based; chapter 6 adopts both memory- and model-based representations.
(Online or batch) Model Update	2.1.4	Batch mode.
(User- or item-based) Model Orientation	2.1.5	The algorithms of this thesis are described in a user-oriented way, but can be easily adapted to an item-oriented setting.
(Collaborative or content-based) Knowledge Source	2.1.6	Chapters 4 and 5 only use collaborative ratings; chapter 6 explores both collaborative and content-based knowledge sources.

Table 2.1: Where this thesis lies with respect to the six dimensions of classification outlined in section 2.1 .

Chapter 3

Evaluation Settings

The best way to test a system is to put it into practice. Therefore, recommender systems are best evaluated through online real user-based experiments. However, such experiments are very expensive due to the cost of creating and maintaining an online system, the difficulty of gaining a user-base, and the time required to gather a sufficient amount of data from the end users. Online recommendation experiments also lack coherence and consistency — properties important for comparing two different systems or system settings.

For these reasons, offline experiments remain dominant in recommender system research. Over time, standard offline data sets gathered from real online systems have been established, experimental protocols developed, and evaluation metrics refined. Survey and interesting discussions regarding the experimentation and evaluation of recommender systems include [41, 48, 23, 88].

This chapter describes the evaluation settings adopted by this thesis. The datasets, evaluation protocols, and evaluation metrics are described in sections 3.1, 3.2, and 3.3 respectively. Section 3.4 serves as a summary of this chapter, as well as a compendium of the evaluation practices of this thesis.

3.1 Datasets

A *rating dataset* can be viewed as a set of $\langle u,\ i,\ r \rangle$ triplets, each of which represents a user u's preference rating r on item i. The rating can be binary, as in implicit datasets, or numerical, as in most of the explicit rating datasets. It is the data source for collaborative filtering recommendations.

Some datasets also provide content attributes of items on top of the rating data, known as the *content data*. To carry out content-based filtering, both rating data and content data are required. This thesis requires both rating and content data, which are described in sections 3.1.1 and 3.1.2.

3.1.1 The Rating Datasets

Three rating datasets are used in this thesis. They are the *MovieLens Standard* (MLS) dataset [120], the *MovieLens Million* (MLM) dataset [120], and the *Jester* joke dataset (JST) [38] , all of which contain explicit numerical ratings gathered from real users. The three datasets are very widely used in modern RS research, making it easier to reproduce experiments and compare system results. The MovieLens datasets especially is considered *the* industry-standard, and is "representative of most data sets used in collaborative filtering recommender systems" [162, 35]. Other rating datasets include the *Each Movie* dataset [171] and the *NetFlix* dataset [174], both of which are similar to MovieLens in terms of data source, rating format, and dataset characteristics.

The three datasets used in this thesis possess different statistical attributes, as presented in table 3.1, thus provide diversity to the evaluation. The *MovieLens Standard* dataset contains 100,000 ratings for 1682 movies by 943 users. The ratings are integers from 1 to 5, with the average rating being 3.53 and the standard deviation of the ratings being 1.17. Each user in the dataset has at least 20 and at most 737 ratings, with the average number being 106 and the standard deviation of the number of ratings being 101. Each item in the dataset has at least 1 and at most 538 ratings, with the average being 59 and the standard deviation being 80. The *sparsity* of the dataset, as defined in formula 2.1, is 0.937. Note that the *user orientation*

		ML Standard	ML Million	Jester
Size	User	943	6,040	59,132
	Item	1682	3,706	140
	Rating	100,000	1,000,209	1,761,439
Rating	Range	integer [1–5]	integer [1–5]	float [1–5][1]
	(μ, σ)	(3.53, 1.13)	(3.58, 1.17)	(3.32, 1.06)
User	# rating	[20 – 737]	[20 – 2314]	[1 – 140]
	# rating (μ, σ)	(106, 101)	(166, 193)	(30, 33)
Item	# rating	[1 – 538]	[1 – 3428]	[166 – 59122]
	# rating (μ, σ)	(59, 80)	(270, 384)	(12582, 11914)
sparsity		.937	.955	.787

Table 3.1: Statistics of Rating Datasets

is denser than the *item orientation*, with the average user possessing more ratings then the average item.

The equivalent statistics of the *MovieLens Million* and the *Jester* joke datasets can be read off table 3.1. In contrast to from *MovieLens Standard*, both datasets have a denser item orientation than the user orientation. The *MovieLens Million* dataset, albeit similar in domain and rating format, is much bigger and much more sparse than the *MovieLens Standard* dataset. The *Jester* dataset is yet again different. It features real number ratings gathered from an unlabelled scroll-bar interface, resulting in a different kind of rating bias. It also features an extremely unbalanced user and item orientations, with only 140 items being ranked by almost 60,000 users. This results in many users having ranked the complete set of items, effectively providing full information in the problem space.[2]

[1]The Jester joke dataset originally featured continuous ratings from -10 to 10. In this thesis, they are scaled to real numbers between 1 and 5 in order to make the evaluation results comparable to the MovieLens datasets.

[2]The purpose of the Jester dataset is to demonstrate the algorithms' performances under near-complete information. Therefore, only its user-orientation is presented in the experiments of this thesis since its item-orientation does not hold such property. Also that since the Jester dataset does not contain content-based data, it is not used in the evaluation of content-based algorithms in chapter 6.

3.1.2 The Content Data

The MovieLens datasets come with preliminary content-based data, where each movie item is described by its movie title, release year, and 18 different genres flattened into 19 boolean-valued attributes[3]. Some publications use this as the sole content source for CBF recommendations [1], but it is generally considered insufficient information to fully exploit the recommendation power of CBF algorithms [10]. To compensate, the *Internet Movie Database* (IMDb) [172] is often used to provide additional content attributes to the MovieLens datasets. This method was pioneered by [12, 39], and has now become a common practice used in many publications [76, 10, 161, 165].

The following content features are extracted from IMDb and used in our experiments: release year, release month, running time, USA certificate, colour information, production country, production company, genre, words in title, keywords, cast (actors and actresses), directors, producers, and writers. Note that not all movies contain complete information. For each attribute, there are on average 9% *MovieLens Standard* movies and 12% *MovieLens Million* movies with the attribute value "unknown".

The content-representation adopted by this thesis takes the form of single-valued attributes with boolean, categorical, or numerical values. Multivariate attributes such as genre, keywords and cast are "flattened" into multiple binary-valued attributes, producing boolean attributes such as *genre-being-action*, *keyword-being-prison*, and *actor-being-will-smith*.

3.2 Experimental Protocols

In standard offline recommender system evaluations, the dataset at hand is partitioned into a *training set* and a *test set*. The training set is visible to the recommender system and is used in its training. The test set, hidden during training, is used later as a reference to evaluate the trained system.

There are cases where a recommendation algorithm contains tweakable parameters whose values are to be determined by empirical trials. In

[3]The nineteenth attribute is "unknown".

these cases, the dataset is partitioned into three instead of two subsets, serving as the *training set*, the *validation set*, and the *test set*. The validation set is used to determine the best parameter settings, and the system's final credit is judged by its performance on the test set, with the system parameters set to the best performing values on the validation set.

In order to minimise the bias introduced by a particular partitioning, the above process is often repeated several times and the reports averaged. Generally, it is a good practice to offset the partitioning to make sure that every data point is used exactly once as test data.

The exact procedure of the partitioning is determined by the *partitioning protocols*, with random partitioning being its simplest form. Over the years, specialised partitioning protocols tailored to recommender systems have been developed, including *skip-every-nth*, *given-n*, and *held-out-k*, which are described in sections 3.2.1, 3.2.2, and 3.2.3 respectively. These partitioning protocols serve different purposes. This thesis uses *skip-every-nth* for general purpose experiments, and *given-n* and *held-out-k* for sparsity experiments.

3.2.1 General experiments via "*skip every* n^{th}"

The *skip every nth* protocol firstly lines up all the data points by positioning the ratings of each user contiguously, with the user ordering and the rating ordering within each user randomised. Then, as its name states, it traverses down this line-up and assigns every n^{th} data point as test data and the rest as training data. This protocol ensures that the sparsity of the training dataset is minimally disturbed, and guarantees a $(n-1):1$ size ratio between training and test data for all users up to decimal rounding. This thesis adopts *skip every* 10^{th} as its general purpose partitioning protocol. This is coupled with ten-fold cross validation [65], with the ten partitions using the same randomised data line-up but offset, thus ensuring that every data point is used exactly once as test subject.

3.2.2 Robustness experiments via "*given n*"

Robustness refers to the sparsity endurance capability of the system. Note that all recommendation datasets are more or less sparse, with the MovieLensMillion datasets featuring a sparsity as high as 0.955 (see table 3.1). Robustness is concerned with the extreme situations where the sparsity is over 0.99. More discussions on dataset sparsity can be found in section 2.4.1.2.

Robustness experiments generally take a regular dataset and thin it down to the desirable sparsity. This is done using the *given n* partitioning protocol, in which n ratings are randomly selected for each user or item as observed training data, and the rest are treated as test subjects. Common configurations are *given-2*, *given-5*, *given-10*, and *given-20*. The most generous user-oriented *given-20* protocol will thin the two MovieLens datasets down to a sparsity of 0.988 and 0.995 respectively;[4] the draconian *given-2* protocol will thin the three datasets down to a sparsity of 0.9988, 0.9995, and 0.9975 respectively. The effect of the given-n protocol on the sparsity of the three datasets is presented in table 3.2.

Table 3.2: The effect of *skip-every-n^{th}* and *given-n* on dataset sparsity.

		Dataset Sparsity		
		ML Standard	ML Million	Jester
Original Sparsity		0.9370	0.9553	0.7872
Skip Every n^{th}	2	0.9685	0.9777	0.8936
	5	0.9496	0.9643	0.8298
	10	0.9433	0.9598	0.8085
Given n (on user)	2	0.9988	0.9995	0.9975
	5	0.9970	0.9987	0.9937
	10	0.9941	0.9973	0.9874
	20	0.9881	0.9946	0.9748
Given n (on item)	2	0.9980	0.9997	0.99997
	5	0.9953	0.9992	0.9999
	10	0.9914	0.9985	0.9998
	20	0.9850	0.9971	0.9997

3.2.3 Controlled robustness experiments via "*held out k*"

The *held out k* protocol [152, 84, 64] partitions the dataset using users and items as boundaries, instead of ratings. The protocol selects k_u users as test users and k_i items as test items, hence partitions the rating dataset into four quarters of test users on test items $\{r_{u_t,i_t}\}$, test users on observed items $\{r_{u_t,i_o}\}$, observed users on test items $\{r_{u_o,i_t}\}$, and observed users on observed items $\{r_{u_o,i_o}\}$, as shown in table 3.3. The first (top-left) partition forms the test dataset, and the remaining three partitions form the training dataset. The *held-out-k* protocol can be coupled with x-fold cross validation to minimise the noise of a single partition, with x being the number of distinct test set partitions. For example, if the protocol is to hold out 20% users and 20% items, x would be $\frac{1}{20\%} \times \frac{1}{20\%} = 25$; whereas if the protocol is to hold out 25% of both users and items, x would be 16, and so forth.

Table 3.3: The user-item matrix with the *held-out-k* partitioning protocol.

	$i_1 \ \ldots \ i_{k_i}$	$i_{k_i+1} \ \ldots \ i_n$	
u_1 $\vdots$ u_{k_u}	$\{r_{u_t,i_t}\}$	$\{r_{u_t,i_o}\}$	test users $\{U_t\}$
u_{k_u+1} $\vdots$ u_m	$\{r_{u_o,i_t}\}$	$\{r_{u_o,i_o}\}$	observed users $\{U_o\}$
	test items $\{I_t\}$	observed items $\{I_o\}$	

The *given n* protocol can be used in combination with *held out k* to control the dataset sparsity of similarity calculation and rating prediction separately. For the three partitions that form the training dataset, with $\{r_{u_o,i_o}\}$ remains unchanged, applying the *given n* protocol on $\{r_{u_t,i_o}\}$ can be used to control the data sparsity in user similarity computation, and applying the *given n* protocol on $\{r_{u_o,i_t}\}$ can be used to control the dataset sparsity in rating prediction calculation.

[4]Due to the low (user-oriented) sparsity of the Jester joke dataset, the given-10 protocol will only increase the sparsity to 0.929, which does not qualify for "extreme" and is not suitable for robustness experiments.

3.3 Evaluation Metrics

In offline recommender system evaluation, two recommendation formats have been formalised to mimic the utilities of real-world users. They are the *rating prediction* format and the *list recommendation* format. The latter can be further divided into *ranked list recommendation* format and *binary list recommendation* format, based on whether or not the list is ordinal.

In the *rating prediction* format, recommendations are presented to the target user in the form of a numerical rating for each queried item, indicating the expected user preference for that item. The original GroupLens recommender [66] is of this type. In the *list recommendation* format, recommendations are presented as a list of best-qualified items selected from a set of candidate items known as the *bucket*. If the list is ordered based on the expected preferences, then it is of the *ranked recommendation* format, otherwise it is of the *binary recommendation* format. The fundamental difference between the (rating) prediction task and the (list) recommendation task is that the latter focuses on the *relative* instead of the *absolute* comparison of the items, or that it focuses on *rankings* instead of *ratings*. An item with a rating of a 2 out of 10 can still be the top-ranked item thus strongly recommended, so long as the rating of 2 is the highest of all. Real-world application of binary recommendation include E-commerce websites such as *Amazon* [81]; applications of ranked recommendation include *Siteseer* [121] and the essence of internet search engines.

Various research has pointed out the importance of using appropriate evaluation metrics for tasks of different formats [48, 41, 88], and the potential of misleading evaluation results if inappropriate metrics are used [48, 88]. Metrics for evaluating rating prediction tasks operate by comparing the numerical prediction score with the actual user ratings. They are known as the *predictive accuracy metrics*, and are described in detail in section 3.3.1. Metrics for evaluating binary and ranked recommendation tasks are known as the *binary-* and *ranked- recommendation accuracy metrics*. These metrics often have an information retrieval background, and are described in detail in sections 3.3.2 and 3.3.3 respectively.

The algorithms proposed in this thesis are rating-prediction based, mak-

ing the predictive accuracy metrics the primary evaluation choice. Although a rating-based recommendation feedback provides more information to the end users than the list recommendation formats, the ultimate decision the user would make based on this recommendation is often binary, namely *yea* or *nay*. User studies have also shown that *rankings*, instead of *ratings*, may sometimes be a more appropriate indication of whether an item is to be accepted by a user, thereby a better measure of user experience [48, 41]. Therefore, to better understand a rating-prediction algorithm, it is worthwhile evaluating its decision-support effectiveness by testing it on binary- and ranked- recommendation tasks. Rating prediction systems like the ones proposed in this thesis can be easily adapted to solve list-recommendation tasks — for binary recommendation, a rating-threshold can be chosen so that the system recommends all items with ratings above the threshold; for ranked recommendation, a rank based on the predicted rating can be imposed.

For the above reasons, a set of evaluation metrics of *all three* formats were selected to evaluate the different facets of the rating-prediction algorithms proposed in this thesis. Sections 3.3.1 to 3.3.3 provide a detailed survey and analysis of the state-of-the-art evaluation metrics of each format, with an emphasis on the metrics used in this thesis. Section 3.3.4 outlines another important aspect of comparative evaluation — the significance tests.

3.3.1 Predictive Accuracy Metrics

The standard way of evaluating rating-prediction recommender systems is to compare the prediction score provided by the system against the actual rating provided by the user for each user-item pair in the test data set, thereby measuring the *accuracy* of the rating predictions. Such evaluation metrics are known as the *predictive accuracy metrics* (PAMs).

PAMs can only be applied to systems that produce *explicit ratings* — a condition not met by list recommendation-only systems. It has also been pointed out in various studies such as [23, 48, 87, 41] that PAMs are not fit to evaluate the ranking effectiveness of an algorithm, even if the algorithm

does produce explicit rating predictions. In many cases, a very small difference in an algorithm's predictive accuracy can lead to a very significant improvement in its recommendation accuracy [68]. This is mostly because PAMs weigh all errors equally regardless of the consequence of the error on the item's ranking. They can also be overly sensitive to unnecessary details, especially when a prediction of a 4 as a 5 or a 2 as a 1 makes no difference to the user. Therefore, PAMs should only be considered as one component for evaluating rating prediction-based recommender systems, and should be used alongside of other binary- and/or ranked- recommendation accuracy metrics for omnifaceted evaluations.

This thesis adopts two commonly used PAMs to evaluate the rating-prediction accuracy of the proposed algorithms. They are the *mean absolute error* (MAE) and the *root mean squared error* (RMSE), and are explained in detail in sections 3.3.1.1 and 3.3.1.2. Other predictive accuracy metrics such as the *normalized mean average error* [48] and the *prediction-rating correlations* [50, 48] are less popular due to their added complexity and strong linear agreement with the MAE, thus are not adopted in this thesis.

3.3.1.1 Mean Absolute Error (MAE)

Given a test dataset consisting of N prediction queries in the form of user-item pairs; suppose $p_{u,i}$ is the predicted rating and $r_{u,i}$ is the actual rating of user u on item i; the *mean absolute error* (MAE) of the recommender system on the test dataset is defined in formula 3.1:

$$\text{MAE} = \frac{\sum_{(u,i)\in\text{testset}} |p_{u,i} - r_{u,i}|}{N} \tag{3.1}$$

MAE measures the deviation of the predictions from their true user-specified values. Lower MAE indicates better prediction accuracy. The earliest published research that uses MAE on recommender systems is Shardanand and Maes [138] in 1995. Since then, MAE has become one of the most popular metrics in recommender system research due to its computational simplicity and strong linear agreement with many other more complicated metrics [48]. The same metric is sometimes referred as the *average absolute deviation* [23], or simply the *prediction error*. Dedicated

empirical studies on MAE in the context of recommender systems can be found in [48, 87, 88, 41, 143].

3.3.1.2 Root Mean Squared Error (RMSE)

MAE weighs all errors equally, which may not be a preferred evaluation bias. The *root mean squared error* (RMSE) has a similar effect as MAE, but weights highly out-of-whack predictions more severely, thus is a better measure especially when small prediction errors such as predicting a 4 as 5 or 2 as 1 do not matter much for the user experience. The RMSE metric is defined as follows in formula 3.2:

$$\text{RMSE} = \sqrt{\frac{\sum_{(u,i)\in\text{testset}}(p_{u,i} - r_{u,i})^2}{N}} \tag{3.2}$$

RMSE measures the *distance* between the predictions and the true user-specified values. Lower RMSE indicates better prediction accuracy. Studies have shown that, in recommender system evaluations, an improvement as small as a 0.01 on the RSME can lead to a very significant improvement when ordering items by their predicted preference [68], thus is considered significant and meaningful to the end user [120, 48, 152]. Studies of the RMSE metric can be found in [130, 143, 41].

3.3.2 Binary Recommendation Accuracy Metrics

Binary recommendation accuracy metrics (or simply *binary metrics*) measure the ability of recommender systems at simply telling *yea*s from *nay*s. They are normally used to evaluate systems designed to make binary (non-ranked) recommendation sets, but can also be applied to systems that output ranking or rating predictions. For the latter usage, a threshold-cutoff is often implemented for compatibility.

This thesis uses binary metrics to evaluate the *decision support* characteristics of the proposed rating-based algorithms. More often than not, the real-world decision the end user ultimately makes based on the recommendation is binary, namely acceptance or rejection. Binary recommen-

dation tasks and metrics allow us to look beyond the format of the recommendation and evaluate how well the system is at assisting the final decision of the user. As pointed out in [169], although binary metrics do not map directly to the user's utility, they do serve as a good complement to the predictive accuracy metrics such as the MAE.

The rest of the section is organised as follows: 3.3.2.1 describes the methods and options for adapting binary recommendation accuracy metrics to rating-prediction based recommender systems; the later two sections outlines the two families of binary metrics — 3.3.2.2 presents precision and recall, and their implied PRC curve and F1-measure; 3.3.2.3 presents sensitivity and specificity, and their implied ROC curve and AUC-measure.

3.3.2.1 How to apply binary metrics to rating-prediction systems

All binary recommendation accuracy metrics deal with the statistical associations of two sets — the set of items *recommended* by the system to the user, and the set of items that are actually *preferred* by the user. An item can be classified as either *positive* or *negative*, based on whether or not it is recommended by the recommender system; this classification can be either *true* or *false*, based on whether or not the recommendation agrees with the actual user preference. Therefore, an item can fall into one of the four categories based on its *recommendation* and *preference* status. This relationship is illustrated in table 3.4.

	Recommended	Not Recommended
Preferred	True-Positive (TP)	False-Negative (FN)
Not Preferred	False-Positive (FP)	True-Negative (TN)

Table 3.4: Classification of items based on the item's recommendation and its actual user preference

The algorithms proposed in this thesis are of the rating-prediction format, where the system produces a numerical or distributional rating indicating the expected preference of the user on the queried item. To evaluate such algorithms with binary recommendation accuracy metrics, an extra

step is needed to convert rating prediction scores into binary yes-or-no recommendations. Since the datasets used in this thesis are also rating-based, an extra step of converting users' actual ratings into users' actual binary preferences is also needed. The rest of the section outlines various ways of converting rating predictions into binary recommendations, and ways of converting user ratings into binary user preferences.

There are three ways of converting rating predictions into binary recommendations. The first one is the *top-N approach*, which generates a recommendation list of size N consisting of the N items with the highest rating predictions. For web-based recommender systems, it is common practice to choose an N between 1 and 20, as it is unlikely for users to be interested in a longer list [87]. Published applications adopting this approach include [128, 34, 48, 169, 161]. The second approach is the *top-quartile approach*, which recommends all items whose predicted ratings are in the top quarter. This approach is not as popular as the *top-N* approach due to its inability to control the size of the recommendation list. Actually its only published usage is in [12], where the approach was originally proposed. The third approach is the *threshold cutoff approach*, which specifies a rating-threshold indicating the strictness of the recommender system, and recommend items whose predicted ratings exceed the threshold. Publications adopting this approach include [130, 47]. The threshold approach can also be used in conjunction with the top-N approach, meaning that only the top N highest-rated items with ratings exceeding the threshold are recommended. If there are not enough items satisfying the threshold, fewer will be recommended. Publications adopting this conjunctive approach include [8]. It is also the approach this thesis adopts, since it has the added benefit of taking predicted ratings into account, as well as allowing explicit control of the (maximum) size of the recommendation list.

There are also three ways of converting actual user ratings into user's binary preferences. The first approach is to use existence as an indication of preference, namely to consider all items present in the test dataset as preferred. This approach is more suitable for implicit recommendation domains such as online purchasing or browsing, as although a purchase or a browse does not prove user satisfaction, it does indicate a certain level

of user interest. Published applications adopting this approach include [128, 34, 169]. The second approach is the *top-quartile* approach, where only items whose actual ratings are in the top quarter are considered as *preferred*. This approach, like its top-quartile recommendation counterpart described in the previous paragraph, has only been used in [12], where it was originally proposed. The third approach is to use a *threshold-cutoff*, where only items whose actual ratings exceed a certain threshold are considered as *preferred*. If preference status is enquired on a recommended item that is absent in the test dataset thus actual rating unknown, it can either be ignored [130, 39, 47, 91, 8, 161], or considered as not-preferred [87]. Ignoring unrated items results in an overestimation of systems that recommend obscure items no one has seen, as such "mistakes" are ignored instead of penalised; considering unrated items as not preferable results in an underestimation of the system, since chances are that some of the unrated items are actually preferred thus correctly recommended.

Certain prediction-to-recommendation conversion approaches can only be paired with certain rating-to-preference conversion approaches. Examples of existing pairing are presented in table 3.5. The pairings adopted by this thesis are also marked.

		Binary Recommendation Conversions		
		Top-N	Top-Quartile	Threshold
Binary Preference Conversions	Existance as Preference	[34]	–	–
	Top-Quartile As Preferred	–	[12]	–
	Threshold (unrated as ignored)	[161]	–	[8], chapter 4, 5, 6
	Threshold (unrated as misses)	[87]	–	chapters 5, 6

Table 3.5: Exemple publications with different approaches of converting rating predictions to binary recommendations (the columns), and of converting actual user ratings to binary user preferences (the rows).

This thesis adopts the *threshold-cutoff* approach as it is the most rating-friendly. Both unrated-items-as-ignored and unrated-items-as-unpreferable are implemented to provide a more holistic evaluation, as shown in table 3.5.

3.3.2.2 Precision, Recall, and the F1 Measure

Precision, recall, and the F1-measure are statistical metrics widely used in the evaluation of binary information retrieval systems [69]. Given a set of instances, suppose the task is to tell the positives (desired, relevant, recommended) from the negatives (undesired, irrelevant, not recommended). *Precision*, as shown in equation 3.3a, measures the ratio of correctly predicted positives over all the positives suggested by the system; whereas *recall*, as shown in equation 3.3b, measures the ratio of correctly predicted positives over all the positive instances out there in the data set. Precision reflects the fidelity of the system's performance on a task, whereas recall reflects its completeness.

$$precision = \frac{\text{\# correctly recommended}}{\text{\# recommended}} = \frac{\text{\#TP}}{\text{\#TP} + \text{\#FP}} \tag{3.3a}$$

$$recall = \frac{\text{\# correctly recommended}}{\text{\# preferred in the data set}} = \frac{\text{\#TP}}{\text{\#TP} + \text{\#FN}} \tag{3.3b}$$

Precision and recall are often conflicting in nature. For example, simply recommending all instances can achieve perfect recall at the cost of low precision, whereas being overly cautious at claiming a positive can vastly improve precision at the cost of low recall. Although there are publications that use solely precision without recall [8], or solely recall without precision [61, 34] in their evaluations, it is more often for the two metrics to be presented together as a package for a more complete evaluation. The *precision-recall curve* or PRC curve is a curve that plots the precision (on the y-axis) against recall (on the x-axis) with varying recommendation strictness, which is controlled by the N value in a top-N recommendation scheme or the rating threshold in a threshold-cutoff recommendation scheme. Since the curve provides a holistic view of the correlations between precision and recall, it can be used to guide the choice of the optimal recommendation strictness on the deployment of the system. The curve can also be used to compare the performances of competing recommender systems on a holistic precision-and-recall-combined scale, as a dominant PRC curve with more area underneath indicates better overall

decision-support accuracy. Recommender systems using the PRC curve as part of their evaluation scheme include [130, 87, 161].

A more compact way of comparing the decison-support performances of competing recommender systems is the *F1-measure*, which is a single metric featuring the harmonic mean of precision and recall, as shown in equation 3.4. Recommender systems evaluated using the F1-measure include [128, 161].

$$F1 = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.4}$$

In most cases, it is desirable to reach a balance between precision and recall, as both are important for the quality judgement of the system. However, it has been pointed out in [12] that the nature of the recommendation tasks induces a greater demand on precision over recall. This is because when it comes down to user experiences, fidelity trumps completeness — users in general do not mind not seeing some potentially interesting recommendations, so long as there are other equally-interesting recommendations occupying their minds; whereas being recommended an item that is not of interest to the user would quickly drain the user's confidence on the system. However, in the rare occasions where the cost of overlooking an item is high, such as the recommendation of medical diagnoses or legal documents, user experiences give in to the risk of missing any positive items, and recall-biased recommendations are preferred. Information retrieval tasks such as web search are also recall-biased, but for a different reason. It is because the cost of verifying a returned item is relatively low, and the user can always refine their search to obtain higher fidelity.

Interesting studies have been done on the trade-offs between precision and recall on different recommender system techniques. Basu et al. [12] showed that collaborative filtering techniques tend to produce precision-oriented results, content-based filtering techniques tend to be recall-oriented, and hybrid filtering combines the advantage of the two filtering techniques thus can achieve an improvement on both precision and recall.

3.3.2.3 ROC, GROC and CROC Curves

Like precision and recall, another pair of conflicting yet complementary binary recommendation metrics are the *hit-rate* and *miss-rate*, or sometimes also referred as *sensitivity* and $(1-)$ *specificity* [134], or *true* and *false positive rate* [41]. *Hit-rate*, as shown in equation 3.5a, is exactly the same as *recall*. *Miss-rate*, as shown in equation 3.5b, measures the ratio of wrongly predicted positives (recommended but not preferred) over all the items in the dataset that are not-preferred by the user. Semantically, the hit-rate represents the probability of a preferred item being recommended by the system, whereas the miss-rate represents the probability of a not-preferred item being recommended by the system.

$$hit\ rate(recall) = \frac{\text{\# correctly recommended}}{\text{\# preferred}} = \frac{\text{\#TP}}{\text{\#TP} + \text{\#FN}} \tag{3.5a}$$

$$miss\ rate = \frac{\text{\# wrongly recommended}}{\text{\# not preferred}} = \frac{\text{\#FP}}{\text{\#FP} + \text{\#TN}} \tag{3.5b}$$

Like the precision-recall curve, the *receiver operating characteristic* or ROC curve [36, 69] plots the hit-rate (on the y-axis) against the miss-rate (on the x-axis) with varying recommendation strictness. A good algorithm with higher-than-random performance has an ROC curve that lies in the upper left triangle of the axis. As pointed out in the previous section on page 58, it is more important for a recommender system to have high precision than high recall due to the nature of the recommendation task. Similarly here, it is more important to focus on the left hand side of the ROC curve to keep the false positive (miss) rate low, as a wrongly recommended item could very quickly drain out the confidence of the user on the system.

The area under the ROC curve is a combined performance measure known as the *ROC sensitivity*, a higher value of which represents a higher overall ability for the system to recommend more good items as well as reject more bad items. A perfect performance, where all and only the preferred items are recommended, has an ROC sensitivity of 1.0; random guessing has an ROC sensitivity of 0.5; for interim situations, the value of the ROC sensitivity can be estimated using quadrature [44], a popular ap-

proach of which is known as the AUC (area under curve) approximation as presented in formula 3.6.

$$\text{AUC (Area Under Curve)} = 1 - \frac{1}{2}\sum_{k=1}^{n}(x_k - x_{k-1})(y_k + y_{k-1}) \qquad (3.6)$$

where x_k and y_k are the coordinates of the points defined by the hit-miss-rate pairs. The AUC measure, like the F1-measure in precision and recall, aptly sums up the ROC curve as a single number, thus is handy for comparing algorithms with intersecting ROC curves. However, it does lose important trade-off information provided by the ROC curve, thus should only be considered as a complement rather than a replacement to the curves. Survey papers that discuss the use of ROC metrics on recommendation tasks include [48, 41, 143]. Recommender systems that employ the ROC curves as part of their evaluations include [134, 40]; recommender systems that present only the numerical ROC-sensitivity or AUC-measure without the curve as part of their evaluations include [130, 39, 47, 91].

The ROC metrics (curve and sensitivity) are widely used in machine learning and information retrieval to evaluate binary classification problems [69]. The difference here is, due to the personalised nature of recommender systems, both the system's overall accuracy as well as its per-user accuracy are of interest. Schein et al. [135] introduced the terminologies of *global ROC* (GROC) and *customer ROC* (CROC), the former refers to the application of the ROC metrics on the entire test dataset, and the latter refers to the application of ROC on a per-user bases. For rating threshold-controlled binary recommendation, the difference between GROC and CROC is small; for top-N binary recommendation, GROC requires the N-value to be defined on a cross-user scale, whereas CROC requires N to be defined on a per-user scale, and then averages the per-user ROCs to form the final aggregated ROC known as the CROC curve. If GROC is to be applied, the task of recommending the top-N items to the user would be interpreted as recommending the top-N user-item pairs, so that the system is allowed to make more recommendations to some users than the others; if CROC is to be applied, everyone gets the same number

of recommendations. For this reason, CROC is considered a more appropriate metric than the GROC, as it can be used to guide the choice of the number of recommendations to each user. Surveys that studied the GROC and CROC metrics include [41, 143]; recommender systems that present both the GROC and CROC as part of their evaluations include [134, 40].

A property worth noticing for the CROC curve is that, if there exist some users in the test dataset who have fewer than N preferred items (which is the case for most real datasets), a perfect recommender would no longer have a curve with area 1, since it can never recommend N items all of which are preferrable to those users. Therefore, for each CROC evaluation, a curve for the perfect recommender should be provided to mark the upper boundary of the system performance.

3.3.3 Ranked Recommendation Accuracy Metrics

Ranked recommendation accuracy metrics (or *ranking metrics*) measure the ability of recommender systems to impose an item ordering that correctly reflects the user's ordinal preference, which has been recognised as a vital part of the user experience [48]. As mentioned previously,[5] predictive accuracy metrics do not incorporate the fact that the intervals on the rating scale are not equidistant, resulting in two undesirable properties in their evaluation bias. Firstly, they do not take into account the effect an error has on the ranking placement of items — a wrongful prediction of a 2.5 as 1 has the same impact as a wrongful prediction of a 3.5 as 5 on the MAE, despite the fact that the latter would more obviously lead to user dissatisfaction. Secondly, predictive metrics can be overly sensitive to small errors that have little effect on the user experience, such as predicting 4.5 as 5 or 1.5 as 1.

Both biases can be compensated by using ranking metrics as a complement to the predictive accuracy metrics in the evaluation. In the following, section 3.3.3.1 firstly explains how the conversion from rating to ranking can be made, so that ranking metrics can be applied to rating-based systems; section 3.3.3.2 then outlines the different types of ranking metrics;

[5] As mentioned in section 3.3.1 on page 52.

sections 3.3.3.3, 3.3.3.4, and 3.3.3.5 in turn describe the three most popular ranking metrics used in modern recommender system publications.

3.3.3.1 How to apply ranking metrics to rating-prediction systems

Ranking metrics are normally used to evaluate systems designed to make ranked recommendation lists, but can also be applied to systems that make rating predictions. For the latter usage, an extra step of rating to ranking conversion is normally required.

In general, the conversion is done by firstly converting the rating predictions into a binary set of *recommended items* according to the procedures described in section 3.3.2.1, then imposing an ordinal ordering based on descending order of predicted ratings over the recommended items, converting the binary recommendation *set* into an ordered or ranked recommendation *list*. Similarly, the *actual ranking* to evaluate against is obtained by firstly converting the actual ratings provided by the test dataset into a binary set of *actually preferred items*, upon which an actual-rating-based ordering can be imposed. Metrics such as the correlation-based measures described in section 3.3.3.2 and the ARHR metrics described in section 3.3.3.3 follow this mechanism.

Some other ranking metrics such as ERU and NDCG are rather different. Albeit being ranking metrics, they still use the *rating prediction values* in their formulae. This makes them not applicable to systems that only produce list recommendations without ratings, but at the same time makes them more appropriate, popular, and convenient in evaluating the *ranking* performance of *rating* prediction systems. These metrics do not require the explicit step of converting rating predictions into binary recommendation set first, but instead has a threshold-based built-in mechanism that tell apart recommended vs. not-recommended items. In ERU it is done by the *default rating* r_d, and in NDCG it is done by the parameter n. More on these two metrics are presented in sections 3.3.3.4 and 3.3.3.5 respectively.

3.3.3.2 Types of ranking metrics

Most ranking metrics are either *correlation* based or *importance decay* based. Correlation based ranking metrics such as *Spearman's correlation* [48], *Kendall's Tau correlation* [48], and the *normalized distance-based performance measure* [9] operate by directly comparing the correlations between two vectors — the vector of the recommended ranking, and the vector of the actual preferential ranking. This mechanism suffers from the *interchange weakness*, meaning that a miss-rank at the top of the ranking has the same weight as if it happens at the bottom of the rank. Such metrics, albeit rank-based, do not incorporate the semantics of "rank", thus are not adopted by this thesis.

Importance-decay based ranking metrics incorporate the essence of "ranking" by imposing an importance decay over the recommended items along the ranking list. Three such metrics that feature linear, exponential, and logarithmic decay are presented in sections 3.3.3.3, 3.3.3.4, and 3.3.3.5 respectively.

3.3.3.3 Linear Decay — Average Reciprocal Hit-rank (ARHR)

The *average reciprocal hit-rank* or ARHR [34] is a ranked recommendation accuracy metric that imposes a *linear* decay over the expected importance of the recommended items along the ranking. Given a weakly ordered (i.e. tie allowed) list of recommended items, let i be the item in this list that is also actually preferred by the user,[6] let rank(i) be the rank of item i, the ARHR metric is defined as follows in formula 3.7a.

$$\text{ARHR} = \frac{1}{\#\text{preferred}} \sum_{i \in \{\text{recommended \& preferred}\}} \frac{1}{rank(i)} \tag{3.7a}$$

$$\text{hit rate}(recall) = \frac{1}{\#\text{preferred}} \sum_{i \in \{\text{recommended \& preferred}\}} 1 \tag{3.7b}$$

The ARHR is essentially the rank-weighted version of the *hit-rate* bi-

[6] As explained in section 3.3.3.1, ARHR requires the generation of a binary recommendation set first, upon which a rating-based ranking order can be imposed. The binary concept of "actually preferred" and "recommended" is provided by this binary set conversion.

nary metric.[7] The value of ARHR varies from *hit-rate* to $\frac{hit-rate}{N}$, where N is the size of the ranked recommendation list. The former happens when all the hits occur in the first rank, whereas the latter happens when they all occur in the last rank. Recommender systems that use ARHR as part of their evaluations include [34].

3.3.3.4 Exponential Decay — Expected Rank Utility (ERU)

Expected rank utility (ERU) [23] or the *half-life utility metric* [48] is a ranked recommendation accuracy metric that imposes an *exponential* decay on the importance or the "expected utility" of the recommended items along the ranking. It is one of those "different" ones that is dedicated to evaluate the *ranking* performance of *rating*-producing systems, and still uses the values of rating predictions in its formula. It also only evaluates the ranking performance on a per-user basis, as opposed to the previously described metrics such as ARHR, which can be applied to evaluate both the per-user ranking performance and the global ranking performance (of a large ranking list across all users).

Suppose U_Q is the set of queried users. For each $u \in U_Q$, a ranked recommendation list is generated for this user by imposing an ordering based on this user's ratings over all items in the dataset. Suppose n is a experimenter-specified parameter corresponding to the length of the recommendation list; i_k is the kth item in the ranked list; r_{u,i_k} is the actual rating that user u gives to item i_k; r_d is a pre-determined constant known as the *default rating* that normally takes on a slightly below-average value, representing the rating threshold of minimal user satisfaction; α is the *half-life*, representing the rating of the item that has a 50% chance to be viewed by the user. The *expected rank utility* is defined as follows in formula 3.8a:

[7] Hit-rate is a binary metrics described in formula 3.5a on page 59, and is reformulated here in formula 3.7b.

$$R = 100\frac{\sum_{u\in U_Q} R_u}{\sum_{u\in U_Q} R_u^{\max}} \quad \text{,where} \tag{3.8a}$$

$$R_u = \sum_{k=1}^{n} \frac{max\ (r_{u,i_k} - r_d\ ,\ 0)}{2^{\ (k-1)/(\alpha-1)}} \tag{3.8b}$$

where $R_u^{\max}$ is the maximum possible utility computed using formula 3.8a on user's actual ranking — a ranked list generated based on items' actual ratings instead of their predicted ratings.

This mechanism treats an unrated item as a miss, though instead of penalising it completely, it effectively assigns it the default rating r_d. The metric is most appropriate for domains that value the top of the recommendation list greatly and have a sharp exponential drop in the item's expected utility further down the list. Publications that use the expected rank utility as part of their evaluations include [23, 115, 48].

3.3.3.5 Logarithmic Decay — The NDCG Performance

The *normalised discounted cumulative gain* or NDCG is a ranked recommendation accuracy metric widely used for evaluating ranked results in information retrieval, and was first adapted to recommender systems in [55]. It is also the main ranking metrics adopted by this thesis. It is similar in attributes to that of ERU in terms of the use of rating prediction values in its calculation, and that it only evaluates the per-user ranking performance but not the global ranking performance. Suppose U_Q is the set of queried users; n is the (capped) length of the recommendation list for each queried user; i_k is the kth recommended item; r_{u,i_k} is the actual rating that user u gives to item i_k. The NDCG of a ranked recommendation list is defined as follows in formula 3.9:

$$\text{NDCG}(U_Q, N) = \frac{1}{|U_Q|} \sum_{u\in U_Q} \left(Z_u \cdot \sum_{k=1}^{N} \frac{2^{r_{u,i_k}} - 1}{log(1+k)} \right) \tag{3.9}$$

where Z_u is the normalisation factor so that the NDCG of the optimal ranking has a value of 1. With normalisation, the value of NDCG ranges from 0 to 1, with a higher value representing closer-to-truth ranking performance. Like the expected rank utility introduced in section 3.3.3.4, the NDCG metric puts more weight on, thus is more sensitive to, the top of the ranking list. It also takes into account users' actual ratings in its computation. However, it features a logarithmic decay instead of an exponential decay as in the expected rank utility metric. Publications that use the NDCG performance as part of their system evaluations include [55, 82, 83]. It is also the ranking metric used in the evaluation of this thesis.

3.3.4 Statistical Significance Tests

Statistical significance tests [33] are a way of validating the comparison result of different algorithms under certain numerical metrics to be sure it is statistically valid and not due to chance. They can be applied to all the numerical metrics described in the previous sections, including MAE, RMSE, precision and recall, F1-measure, and AUC.

Given a *hypothesis* that algorithm A is better than algorithm B for a certain task under a certain evaluation metric, the *null hypothesis* is that our hypothesis is wrong, namely A is no better than B. The *significance level* or ρ-value is the probability that our hypothesis is true only due to chance, or that the null hypothesis is indeed the truthful one. The lower the ρ-value, the less likely our hypothesis is true due to chance, and the more *statistically significant* our hypothesis is. It is common practice to consider a hypothesis statistically significant if its ρ-value is lower than 0.05.

A significance test is either a *paired test* or a *multiway test*, based on whether or not the hypothesis that the test is trying to validate involves only two algorithms or more than two algorithms. Detailed explanations are presented in section 3.3.4.1 and 3.3.4.2 respectively.

3.3.4.1 Paired Test

This thesis uses a *paired t-test* to test hypotheses involving only two algorithms. Suppose n is the number of data queries (i.e. the number of rec-

ommendations to make in this context), a_i and b_i are the performance of algorithms A and B on the ith query under a given numerical metric such as the MAE. $d_i = a_i - b_i$ is the difference in performance, and $\overline{d}$, as shown in formula 3.10a, is the expected difference between the performance of algorithms A and B. The hypothesis of that algorithm A outperforms B would then be translated into the hypothesis of that $\overline{d} > 0$. The t-value of this hypothesis is computed using the student t-distribution with $n - 1$ degrees of freedom, as shown in formula 3.10b:

$$\overline{d} = \frac{1}{n} \sum_{i=1}^{n} a_i - b_i \tag{3.10a}$$

$$t = \frac{\overline{d}}{s.e.(d)} = \frac{\overline{d}}{\sqrt{\frac{1}{n(n-1)} \sum_{i=1}^{n} (d_i - \overline{d})^2}} \tag{3.10b}$$

where $s.e.(d)$ is the estimated standard difference or the *standard error*. The paired t-test assumes a normal distribution over the hypothesised variable, namely d in this case. This assumption is true only if d is naturally normally distributed, or that there are enough (more than 30) data sets to hedge the risk [33]. Although the former condition is uncertain and the latter untrue for most of the published recommendation experiments out there, the paired t-test is still the most widely adopted significance test for recommender systems [91, 34, 45] due to its relative simplicity and accuracy.

3.3.4.2 Multiway Test

Multiway test refers to a significance test with the purpose of validating for example the performance superiority of one algorithm over *all* other algorithms of interest. It has been shown statistically that hypotheses involving multiple algorithms cannot be validated by simply performing paired tests multiple times [33, 41]. Specialised multiway tests have thus been developed, including the *Bonferroni test* [33, 41], the *Friedman test* [33], and the ANOVA test [33].

This thesis adopts the *Bonferroni test* for significance tests of multi-way hypotheses because of its simplicity, accuracy, and popularity within other

recommender systems [34, 41]. The Bonferroni test is also known as the *Bonferroni correction*. As its name suggests, it operates by performing the paired tests multiple times, and then applies a statistical correction to reduce inaccuracy. Suppose the hypothesis H states that algorithm A outperforms all N other algorithms. In order to claim H with significance level ρ, namely to claim the validity of H with a confidence level of more than $1 - \rho$, N paired tests need to be performed between algorithm A and each of the other algorithms, with the significance level for each paired test not exceeding $1 - \sqrt[1/N]{1 - \rho}$. In this way, to claim that algorithm A outperforms 10 other algorithms with significance level 0.05, 10 paired tests need to be performed, each of which must reach a significance level below 0.0057.

3.4 Summary

Evaluation is a vital part of scientific studies. In recommender system research, a set of evaluation datasets, protocols, and metrics have been established to accommodate the special characteristics of the field, its dual-orientationess (user and item), and its personalised nature.

As described in section 3.1, this thesis uses the two MovieLens datasets as its primary datasets. The choice is made because of the MovieLens's long-established industrial status, its widespread adoption, its stereotypical dataset statistics, and the extensibility of content attributes through the IMDb movie content collection. On top of this, diversity is provided with the additional use of the Jester joke dataset, which possesses a different set of statistical characteristics from the MovieLens datasets.

As described in section 3.2, this thesis follows the standard RS evaluation protocols. Skip-every-10^{th} with 10-fold cross validation is used for general experiments. A combination of given-n and held-out-k is used to increase the sparsity of the training dataset in order to test the robustness of the system.

Section 3.3 classifies current RS evaluation metrics into three classes of predictive accuracy metrics in section 3.3.1, binary recommendation accuracy metrics in section 3.3.2, and ranked recommendation accuracy met-

rics in section 3.3.3. The algorithms proposed in this thesis are evaluated using all three classes of metrics, with MAE and RMSE for the evaluation of rating prediction accuracy; precision, recall, and F1-measure for the evaluation of binary set recommendations; and NDCG (logarithmic decay that takes ratings into account) for the evaluation of ranked-list recommendations. This thesis also provides statistical significance for all the hypotheses proposed. The *paired-t test* described in section 3.3.4.1 is used to verify paired hypotheses, and the *Bonferroni test* described in section 3.3.4.2 are used to verify multiway hypotheses. Unless specified otherwise, all experimental results presented in this thesis pass the statistical significance test with significance level of $\rho = 0.05$.

Chapter 4

Investigating Nearest Neighbour Collaborative Filtering

Collaborative filtering (CF) makes recommendations purely based on the preference feedbacks gathered from users, making the assumption that users who agreed in the past will tend to agree in the future. Among CF methods, *nearest neighbour-based collaborative filtering* (NNCF) is one of the top-performing algorithms [105, 107] despite its early establishment [120], and remains a popular research topic in recent publications [162, 133].[1] This chapter attempts to improve the effectiveness of NNCF by discovering and in turn utilising previously "cloaked" information in the recommendation process. Two such heuristics are proposed following the identification of three problems.

This chapter is organised as follows: section 4.1 outlines notation and terminologies; 4.2 explains the algorithm of the standard NNCF. The rest of the chapter presents two improvements over standard NNCF from two different angles. Section 4.3 identifies the *item-irrelevancy problem* and the *preference imbalance problem*, and proposes the *Target-Aware Similarity Computation* or TASK as the solution. Its related work and experimental analysis are presented in sections 4.4 and 4.5 respectively. Section 4.6 identifies the *biased-average problem*, and proposes *partial average and double average* or PANDA as solutions. Its experimental analysis is presented in section 4.7.

[1] Different types of CF algorithms are surveyed in section 2.2, with NNCF elaborated in section 2.2.1.

Finally, section 4.8 concludes the chapter, and explains how observations in this chapter lead to the development of chapter 5.

4.1 Notation and Problem Formalisation

The three fundamental elements of collaborative filtering are users, items, and ratings, which are symbolised in this thesis as u, i, and r respectively. Lower-case letters with subscripted indices represent individual elements, such as user u_s, item i_t, and rating r_{u_s,i_t} or $r_{s,t}$. Capitalised letters represent a set of elements, such as the set of users U, items I, and ratings R. A right arrow over a set notation represents the vector over the elements in that set, ordered by their indices unless specified otherwise. For example, $\overrightarrow{U} = \langle u_1, \ldots, u_m \rangle$ represents the vector of users in set U ordered by user indices, and so forth.

Given a set of ratings R, user u's rating set R_u is the subset of ratings gathered only from u, with its corresponding set of items noted as I_u. Consequently, $\overrightarrow{R_u} = \langle r_{u,i_1}, \ldots, r_{u,i_n} \rangle$ represents user u's rating vector ordered by item indices. The average rating of user u over all $i \in I_u$ is represented as $\overline{r_u}$, as shown in formula 4.1. The same convention applies to item-oriented notations, with the calculation of $\overline{r_i}$ shown in formula 4.2. The global average, noted as $\overline{r}$, is calculated according to formula 4.3.

$$\overline{r_u} = \frac{1}{|R_u|} \cdot \sum\nolimits_{i \in I_u} r_{u,i} \tag{4.1}$$

$$\overline{r_i} = \frac{1}{|R_i|} \cdot \sum\nolimits_{u \in U_i} r_{u,i} \tag{4.2}$$

$$\overline{r} = \frac{1}{|R|} \cdot \sum\nolimits_{u \in U} \sum\nolimits_{i \in I_u} r_{u,i} \tag{4.3}$$

In explicit rating-based recommender systems, the task of recommending new items to a target user can be translated into the problem of predicting the potential ratings the user would give to a pool of items. A requested rating prediction of a user on an item is called a *query*, and is formalised as a triplet of $\langle\ u_\star,\ i_\star,\ r_{u_\star,i_\star}{=}?\ \rangle$, where $u_\star$ is the *target user*, $i_\star$ is the *target item*, and $r_{u_\star,i_\star}$ is the rating to be predicted.

The predictions are based on a set of training data in the form of users' explicit ratings on items, namely a set of triplets $\langle u,\ i,\ r_{u,i} \rangle$ from the space of $U \times I \mapsto R$. In most datasets, users would have only rated a very small subset of the items, making the total number of ratings $|R|$ significantly smaller than $|U| \times |I|$. The training dataset can also be formatted into the so-called *rating matrix* — a $|U| \times |I|$ matrix whose entries take on values $\in (\oslash, [r_{\text{min}}, r_{\text{max}}])$, where $\oslash$ indicates unknown ratings. Figure 4.1 is an illustrative rating matrix of size $m \times n$.

$$
\begin{array}{c}
\begin{array}{c} \\ u_1 \\ \vdots \\ u_m \end{array}
\overbrace{\begin{array}{c} i_1 \quad \cdots \quad i_n \\ \left[\begin{array}{ccc} r_{u_1,i_1} & \cdots & r_{u_1,i_n} \\ \vdots & \ddots & \vdots \\ r_{u_m,i_1} & \cdots & r_{u_m,i_n} \end{array}\right] \end{array}}^{I = \{i_1, \ldots, i_n\}}
\left.\begin{array}{c} \\ \\ \\ \\ \end{array}\right\} U = \{u_1, \ldots, u_m\}
\end{array}
$$

Figure 4.1: An illustrative rating matrix of size $m \times n$.

4.2 Nearest Neighbour Collaborative Filtering

NNCF can be user-oriented [120, 23, 47] or item-oriented [129, 34, 81]. User-oriented NNCF makes predictions based on the preferences of users like-minded to the target user; item-oriented NNCF makes predictions based on the popularities of items similar to the target item. This section outlines the procedure of user-oriented NNCF with Pearson's correlation as the similarity measure. The computation of item-oriented NNCF can be derived by reversing the orientation.

Given a prediction query $\langle\ u_\star,\ i_\star,\ r_{u_\star,i_\star}{=}?\ \rangle$, NNCF computes $r_{u_\star,i_\star}$ — the rating to be predicted — following three steps:

1. **Calculate user similarities:**

   The first step is to compute the similarities between the target user $u_\star$ and every other $u \in U$. The similarity between two users u_1 and u_2 can be calculated as the Pearson's correlation between their rating vectors

$\overrightarrow{R_{u_1}}$ and $\overrightarrow{R_{u_2}}$, as defined in equation 4.4:

$$sim\ (u_1, u_2) = \textit{Pearson's correlation}(\overrightarrow{R_{u_1}}, \overrightarrow{R_{u_2}}) \tag{4.4}$$
$$= \frac{\sum_{i \in I_{u_1} \cap Iu_2} (r_{u_1,i} - \overline{r_{u_1}})(r_{u_2,i} - \overline{r_{u_2}})}{\sqrt{\sum_{i \in I_{u_1} \cap Iu_2} (r_{u_1,i} - \overline{r_{u_1}})^2} \sqrt{\sum_{i \in I_{u_1} \cap Iu_2} (r_{u_2,i} - \overline{r_{u_2}})^2}}$$

2. Define the effective neighbourhood:

Based on the similarities calculated in step 1, the *effective neighbourhood* of the target user is defined as the subset of users who are the "nearest" (i.e. most similar) to the target user. The size of the effective neighbourhood is controlled by an exogenous cut-off parameter k, thus the name k-nearest neighbours. In this thesis, the effective neighbourhood of user $u_\star$ with size k is represented as $U^k_{u_\star}$.

3. Compute the predicted ratings:

Given the target user's effective neighbourhood $U^k_{u_\star}$ and similarities, the prediction $r_{u_\star,i_\star}$ is calculated as the average of the neighbours' ratings on the target item $i_\star$, weighted by the neighbours' similarities to the target user:

$$r_{u_\star,\, i_\star} = \frac{\sum_{u \,\in\, U^k_{u_\star}} sim\ (u_\star, u) \cdot r_{u,\, i_\star}}{\sum_{u \,\in\, U^k_{u_\star}} sim\ (u_\star, u)} \tag{4.5}$$

Resnick et al. [120] introduced *rating normalisation* to the equation to handle the different rating habits of users. This is to deal with the fact that, for example, a rating of 4 out of 5 may mean "outstanding" to one user, but "mediocre" to another. To compensate this, predictions are carried out on *normalised* ratings by recentring the ratings with respect to their user averages, as shown in formula 4.6:

$$r_{u_\star,\, i_\star} = \overline{r_{u_\star}} + \frac{\sum_{u \,\in\, U^k_{u_\star}} sim\ (u, u_\star) \cdot (r_{u,\, i_\star} - \overline{r_u})}{\sum_{u \,\in\, U^k_{u_\star}} sim\ (u_\star, u)} \tag{4.6}$$

4.3 TASK: Incorporating item-relevancy into the computation of user similarity

This section digs into the mechanism of the standard NNCF described in section 4.2, and identifies two problems that hinder the pertinence of the standard correlation-based similarity computation. They are the *item-irrelevancy problem* described in section 4.3.1, and the *preference-imbalance problem* described in section 4.3.2. A solution called the *Target-Aware Similarity (K)computation* or TASK is subsequently proposed in section 4.3.3.

4.3.1 The item-irrelevancy problem

Standard Pearson's correlation, as described in equation 4.4, measures the similarity between two users as the correlation between their entire rating vectors of the items they have in common. This approach has a potential problem, which this thesis names as the *item-irrelevancy problem*. It refers to the situation where some ratings in the rating vectors may be on items that are very different from the target item $i_\star$ of the prediction task. By incorporating such "irrelevant" items into the computation of user similarities, which are ultimately going to be used to make predictions on the target item $i_\star$, noise rather than constructive information may be introduced, resulting in impaired prediction accuracy.

Table 4.1 uses an illustrative example to clarify the problem. The top portion of the table shows the preference data from a group of 10 users, 2 of whom have a *science* background and like the movie *Matrix*, 8 of whom have an *art* background and like the movie *Emma*. In both groups, half of the people from each group happen to like *cheese*, and they tend to have the same preference on *candy*. The bottom row of the table shows two prediction tasks: to predict *Zoe*'s preference on *movie* and *candy*, based on the information she provided on *academics* and *cheese*.

In this simplified world, the *movie* preference of a user is closely related to his *academics* background, but is irrelevant to his preference on *cheese*. Similarly, to predict the preference on *candy*, it is better to look at the person's preference on *cheese*, rather than his *academic* background.

		Items				**Similarities to Zoe based on**		
		Academics	Cheese	Movie	Candy	Both	Academics	Cheese
Users	Ann	Science	like	Matrix	like	1	1	1
	Bob	Science	hate	Matrix	hate	.5	1	0
	Cal	Art	like	Emma	like	.5	0	1
	Dan	Art	hate	Emma	hate	0	0	0
	Eve	Art	like	Emma	like	.5	0	1
	Fay	Art	hate	Emma	hate	0	0	0
	Gil	Art	like	Emma	like	.5	0	1
	Han	Art	hate	Emma	hate	0	0	0
	Ivy	Art	like	Emma	like	.5	0	1
	Jim	Art	hate	Emma	hate	0	0	0
	Zoe	Science	like	?	?	**Corresponding prediction results**		
						Matrix: 3/7 Candy: 6/7	Matrix: 100% Candy: 50%	Matrix: 20% Candy: 100%

Table 4.1: An illustration of the "item-irrelevancy problem"

Given the query of predicting the preferred *movie* of user *Zoe*, who has been known to have a *science* background and have a fondness for *cheese*, conventional NNCF would compute the similarities between Zoe and the training users using their ratings on both known items (i.e. *academics* and *cheese*). This would result in a neighbourhood similarity shown in the "Both" column of table 4.1, leading to a final prediction a $4:3$ preference of *Emma* over *Matrix*, which is inaccurate.

The inaccuracy is caused by *item irrelevancy*, meaning that some of the items used to compare the users are irrelevant to the target item of the prediction task. This thesis proposes a solution in section 4.3.3 based on the idea that only the items relevant to the target item should be used for the similarity computation. In the example of table 4.1, it would be only to use *academics* to predict *movie* and *cheese* to predict *candy*. In this way, the noise brought in by irrelevant items is eliminated, resulting in a neighbourhood similarity as shown in the last two columns of table 4.1. The prediction for *Zoe* would be a 100% preference towards *Matrix* based on the similarities shown in the *Academics* column, and a 100% preference towards *candy* based on the similarities shown in the *cheese* column.

4.3.2 The preference-imbalance problem

There are two contributing factors to standard NNCF's inaccurate prediction of user *Zoe*'s *movie* preference. Although the fundamental cause is the *item-irrelevancy problem*, it is made worse by what this thesis calls the *preference-imbalance problem*.

The preference imbalance problem refers to the situation where most of the users in the training dataset share a common preference on an item, making the less common preference on that item hard to predict, thus resulting in a low recommendation recall on the minority preference. In the example of table 4.1, such imbalance is manifested by 8 out of 10 users prefer the movie *Emma*, whereas the target user *Zoe*'s true preference is the minority item *Matrix*.

The prediction of *candy* does not suffer from this problem, since the preferences of *candy* in the training dataset is split half-and-half. Only affected by item-irrelevancy but not preference-imbalance, standard NNCF is able to predict a $\frac{6}{7}$ chance that *Zoe* likes *candy*. This is only $\frac{1}{7}$ off the truth, which is more accurate than the prediction regarding her choice of movies, which is $\frac{4}{7}$ off the truth.

Preference-imbalance and item-irrelevancy are not independent problems. Without preference-imbalance, item-irrelevancy is still a problem, but not as severe. However, without item-irrelevancy, preference-imbalance is no longer a problem, as it is automatically eliminated by the similarity-based prediction process. This is further clarified using the examples in tables 4.2 and 4.3.

Table 4.2 is an illustrative dataset similar to that of table 4.1, but modified to rid the preference-imbalance on items *academics* and *movie*. In this case, standard NNCF would produce similarities as shown in the "Both" column of the table, and predicts the target user *Zoe* as being 80% of a Matrix-fan — much closer than the $\frac{3}{7}$ prediction when preference-imbalance existed. However, the residue of item-irrelevancy is still manifested by the 20% deviancy from *Zoe*'s true preference.

Table 4.3 is an illustrative dataset similar to that of table 4.1, but modified to rid the item-irrelevancy problem. In this case, standard NNCF

would produce similarities as in the last column of the table, and predicts the target user *Zoe* as 100% of a Matrix-fan, which is correct despite the preference-imbalance.

		Items				**Similarities to Zoe based on**		
		Academics	Cheese	Movie	Candy	Both	Academics	Cheese
Users	Ann	Science	like	Matrix	like	1	1	1
	Bob	Science	hate	Matrix	hate	.5	1	0
	Cal	Science	like	Matrix	like	1	1	1
	Dan	Science	hate	Matrix	hate	.5	1	0
	Eve	Science	like	Matrix	like	1	1	1
	Fay	Art	hate	Emma	hate	0	0	0
	Gil	Art	like	Emma	like	.5	0	1
	Han	Art	hate	Emma	hate	0	0	0
	Ivy	Art	like	Emma	like	.5	0	1
	Jim	Art	hate	Emma	hate	0	0	0
	Zoe	Science	like	?	?	**Corresponding prediction results**		
						Matrix: 80%	Matrix: 100%	Matrix: 60%
						Candy: 80%	Candy: 60%	Candy: 100%

Table 4.2: The illustrative dataset of table 4.1, modified to rid the "preference-imbalance" on items *academics* and *movie*.

		Items				**Similarities**
		Academics	Sci-fi	Movie	Math	**to Zoe**
Users	Ann	Science	like	Matrix	like	1
	Bob	Science	like	Matrix	like	1
	Cal	Art	hate	Emma	hate	0
	Dan	Art	hate	Emma	hate	0
	Eve	Art	hate	Emma	hate	0
	Fay	Art	hate	Emma	hate	0
	Gil	Art	hate	Emma	hate	0
	Han	Art	hate	Emma	hate	0
	Ivy	Art	hate	Emma	hate	0
	Jim	Art	hate	Emma	hate	0
	Zoe	Science	like	?	?	**Prediction results**
						Matrix: 100%
						Math: 100%

Table 4.3: The illustrative dataset of table 4.1, modified to rid the "item-irrelevancy" by replacing items *cheese* and *candy* with *sci-fi* and *math*

The preference-imbalance problem is similar to the *class-imbalance problem* [103] well-known in classification studies. It refers to the imbalanced distribution of instances among different classes, so that the rare classes are hard to get right, because the classifier would have a high overall performance so long as it gets the majority classes (which in most cases also occupies most of the test cases) right. Sometimes the class-imbalance problem is serious, such as earthquake forecast, where it is very important to get the very rare occasions (classes) right. Existing solutions to the class-imbalance problem include *cost-sensitive learning*, where the evaluation function is modified to emphasize the importance of rare classes; and the *under-sampling strategy*, which cuts off instances in the majority classes.

Preference imbalance is different from class imbalance in that it is the *relative* preference imbalance among items that matters, instead of the *absolute* preference imbalance within a single item. In other words, unlike the class imbalance problem which stands on its own, the preference imbalance problem resides within the irrelevancy between items. In the example of table 4.3 where there is no item irrelevancy, preference imbalance no longer hinders the accuracy of the similarity computation of the standard NNCF method.

Section 4.3.3 proposes a solution to the item-irrelevancy problem. Since the preference imbalance problem only arises when there is item-irrelevancy, it is automatically settled when the item-irrelevancy is eliminated.

4.3.3 Target-Aware Similarity Computation (TASK)

The *Target-aware Similarity Computation* or TASK for NNCF is a new approach for calculating user similarities that takes the relevancy of the item orientation into account, thus eliminating the item-irrelevancy problem [2].

In TASK, a *target-item-aware* user similarity $sim^{i\star}(u_a, u_b)$ is used to replace the standard Pearson's correlation similarity $sim(u_a, u_b)$, which is described in formula 4.4 and repeated here as formula 4.7 for convenience. The *target-item-aware* similarity is still based on the correlations between

[2] Without loss of generality, the TASK algorithm is presented as user-oriented. The item-oriented version can be derived by inverting the orientations.

users' rating vectors. However, each rating $r_{u,i}$ is weighted by the *relevancy* between its corresponding item i and the target item $i_\star$ of the prediction task. Suppose $\overset{u_a,u_b}{\mathcal{R}}(i, i_\star)$ is the relevancy between items i and $i_\star$ for the similarity computation of users u_a and u_b, then the *target-item-aware* similarity between users u_a and u_b is given by formula 4.8.

$$sim(u_a, u_b) = \frac{\sum_{i \in I_{u_a} \cap I u_b} (r_{u_a,i} - \overline{r_{u_a}})(r_{u_b,i} - \overline{r_{u_b}})}{\sqrt{\sum_i (r_{u_a,i} - \overline{r_{u_a}})^2} \sqrt{\sum_i (r_{u_b,i} - \overline{r_{u_b}})^2}} \tag{4.7}$$

$$sim^{i_\star}(u_a, u_b) = \tag{4.8}$$
$$\frac{\sum_{i \in I_{u_a} \cap I u_b} (r_{u_a,i} - \overline{r_{u_a}})(r_{u_b,i} - \overline{r_{u_b}}) \cdot \overset{u_a,u_b}{\mathcal{R}}(i, i_\star)}{\sqrt{\sum_i (r_{u_a,i} - \overline{r_{u_a}})^2 \cdot \overset{u_a,u_b}{\mathcal{R}}(i, i_\star)} \sqrt{\sum_i (r_{u_b,i} - \overline{r_{u_b}})^2 \cdot \overset{u_a,u_b}{\mathcal{R}}(i, i_\star)}}$$

The introduction of the relevancy factor $\overset{u_a,u_b}{\mathcal{R}}(i, i_\star)$ addresses the item-irrelevancy problem described in section 4.3.1. The rationale is that when calculating the correlation between two rating vectors $\overrightarrow{R_{u_a}} : \langle \ldots, r_{u_a i_x}, \ldots \rangle$ and $\overrightarrow{R_{u_b}} : \langle \ldots, r_{u_b i_x}, \ldots \rangle$, if item i_x is not relevant to the target item $i_\star$, the corresponding relevancy factor $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_\star)$ should approach to zero weighting, which would effectively nullify item i_x's contribution to the similarity computation; whereas if i_x is strongly related to $i_\star$, positively or negatively, the weighting factor $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_\star)$ should be high so that the corresponding ratings are emphasised. By using the item relevancy as weights, TASK captures the *importance* of each particular item to the current prediction task. By filtering out irrelevant information, more accurate predictions can be expected.

A straightforward way to compute the relevancy factor $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_y)$ is to make it the standard item-oriented similarity between items i_x and i_y, namely $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_y) = sim(i_x, i_y)$. However, $sim(i_x, i_y)$ also suffers from a similar irrelevancy problem in the user-orientation, namely the *user-irrelevancy problem*. It refers to the situation where some users in the items' rating vectors $\overrightarrow{R_{i_x}}$ and $\overrightarrow{R_{i_y}}$ are irrelevant to the target users of the similarity computation, thus introducing noise and ultimately affecting the accuracy of the similarity computation. Since the purpose of the computation of $sim(i_x, i_y)$ is no longer the prediction of the target user $u_\star$, but as a

relevancy-based weight in the similarity computation of users u_a and u_b, the *target user*s in this case are users u_a and u_b instead of the target user of the prediction task $u_\star$. The item relevancy $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_y)$ is consequently the user-aware item similarity $sim^{u_a,u_b}(i_x, i_y)$ instead of $sim^{u_\star}(i_x, i_y)$. Further more, TASK defines $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_y)$ as the *absolute value* of $sim^{u_a,u_b}(i_x, i_y)$, as shown in formula 4.9:

$$\begin{aligned} \overset{u_a,u_b}{\mathcal{R}}(i_x, i_y) = & \left| sim^{u_a,u_b}(i_x, i_y) \right| \\ = & \left| \frac{\sum_{u \in U}(r_{u,i_x} - \overline{r_{i_x}})(r_{u,i_y} - \overline{r_{i_y}}) \cdot f(u)}{\sqrt{\sum_u (r_{u,i_x} - \overline{r_{i_x}})^2 \cdot f(u)} \sqrt{\sum_u (r_{u,i_y} - \overline{r_{i_y}})^2 \cdot f(u)}} \right| \end{aligned} \tag{4.9}$$

where $f(u)$ is the *user-relevancy* between user u and the two users u_a and u_b, who are the *target* of the similarity computation of $sim^{u_a,u_b}(i_x, i_y)$:

$$\begin{aligned} f(u) = & \overset{i_x,i_y}{\mathcal{R}}(u, \{u_a, u_b\}) \\ = & \frac{\overset{i_x,i_y}{\mathcal{R}}(u, u_a) + \overset{i_x,i_y}{\mathcal{R}}(u, u_b)}{2} \end{aligned} \tag{4.10}$$

where $\overset{i_x,i_y}{\mathcal{R}}(u, u_a)$ and $\overset{i_x,i_y}{\mathcal{R}}(u, u_b)$ are computed using the user-oriented version of formula 4.9, forming an *iterative* pattern. The number of iterations is controlled by an exogenous parameter η, with the iteration terminated by using standard similarities instead of the *target-aware* similarities to weight the formula. In this way, the standard similarity computation can be viewed as *target-aware* similarity with $\eta = 0$. This process is clarified in table 4.4, which provides the pseudocode of the calculation of $\overset{u_a,u_b}{\mathcal{R}}(i_x, i_y)$.

proc relevance$(i_x, i_y, u_a, u_b, \eta)$:
if $\eta \leq 0$ **then**
  return $sim(i_x, i_y)$ *as in formula 4.7;*
else
  return $\mathcal{R} = \left| sim^{u_a,u_b}(i_x, i_y) \right|$ *as in formulae 4.9 and 4.10;*
  where $\overset{i_x,i_y}{\mathcal{R}}(u, u_{a/b}) =$ **revelance**$(u, u_{a/b}, i_x, i_y, \eta - 1)$*;*
end

Table 4.4: The pseudocode that illustrates the iterative process of TASK.

The TASK method was published in Zhang and Andreae [166] under the less-descriptive name of *Iterative Neighbourhood Similarity Computation*. This version did not contain the discussion of preference-imbalance, and the relevancy factor was defined as the user-aware similarity $sim^{u_a,u_b}(i_x, i_y)$ instead of its absolute value as in formula 4.9. This was a previous oversight that missed the fact that items negatively correlated to the target item provide as much constructive information to the similarity computation as positively correlated items. Upon fixing this oversight, further improvement on the prediction accuracy is achieved.

4.4 TASK: Related Work

This section presents related work of TASK. Section 4.4.1 presents studies that also weigh items by their *importance* to the similarity computation of users. Section 4.4.2 presents studies that combine user- and item-oriented NNCF at the rating-prediction stage. The algorithms are described in a user-oriented fashion, but can be easily adapted to be item-oriented by reversing the orientations.

4.4.1 Incorporating Item Importance

Inverse user frequency proposed in Breese et al. [23] is based on *inverse document frequency* [125] in information retrieval. The idea is that universally liked items are not as useful in capturing similarity as less commonly liked items. Therefore, it suggests to weighting the (binary) votes for item i by $|log\frac{n}{n_i}|$, where n is the total number of users and n_i is the number of users who voted in favour of i. In case that all users voted in favour of i, the logarithmic factor would converge to zero, nullifying its contribution to the user similarities. A problem with this method is that it is only applicable under a binary rating scheme. The simplistic attempt to adapt it to numerical ratings by using a threshold-cut would result in a problem — instead of degrading items with high common views, it would only degrade items with high *positive* common views, but actually amplifies the effect of items with high *negative* common views.

Variance weighting proposed in Herlocker et al. [47] is a more sophisticated attempt to adapt inverse user frequency to numerical ratings. The idea is to consider the *variance* of the ratings of an item, and emphasise items with high variance in user similarity computations. This has an immediate problem, which they pointed out themselves. The problem is that it does not take into account the fact that, if two users' ratings show agreement with each other in a way that disagrees with the popular opinion, the information this exhibits should be more significant. Their experiments also showed that weighting items using their rating variances led to a slightly worse prediction accuracy than no weighting. Other failed attempts that follow the same philosophy include [157], which proposed an entropy-based item diversity measure as the weighting factor.

Automatic weighting scheme proposed in Jin et al. [59] is a more complicated attempt based on the same philosophy that universally liked items should carry minimal weights in the similarity computation of users. The idea is to dynamically learn a set of item weights that maximise the total user-on-user "asymmetric explainability" under a probabilistic model. The item weights are then used directly as multipliers in the Pearson's correlation similarity computation.

One problem with the aforementioned heuristics is that they are not "*target-aware*", meaning that the item weights are the same for all user similarity computations regardless of the pair of users involved. Therefore, these heuristics are not able to handle the item-irrelevancy problem or the preference-imbalance problem described in section 4.3.

Correlation significance weighting is another weighting heuristic proposed in Herlocker et al. [47] and refined in Herlocker et al. [48] and Ma et al. [84]. It is less related to the work of this thesis in that, instead of weighting *individual items* based on their estimated pertinence to the similarity computation, it weights the computed similarity between two users by the number of common items between the two users used in the similarity computation, namely $sim'(u_1, u_2) = \lambda \cdot sim(u_1, u_2)$, where $\lambda = \frac{max(|I_{u1} \cap I_{u2}|, \gamma)}{\gamma}$ in [48] and [47], and $\lambda = \frac{min(|I_{u1} \cap I_{u2}|, \gamma)}{\gamma}$ in [84]. The idea is that a higher availability of data points in the similarity computation suggests a higher confidence in user correlations, thus higher similarity between users. This heuristic,

albeit also similarity weighting-based, is actually orthogonal to, thus can be combined with other *item* weighting heuristics presented in this section, as well as TASK.

4.4.2 Combining user- and item-oriented predictions

The TASK method can be viewed as a hybrid user- and item-based NNCF approach, since both user similarities and item relevancies are computed in order to make predictions. This section outlines the publications that are also hybrid user- and item-based NNCF, but combine the two orientations at the rating-prediction stage instead of the similarity computation stage.

Effective missing data prediction proposed in Ma et al. [84] is a Pearson's correlation-based NNCF method that incorporates both the user and the item orientations by making the final prediction a weighted average of the predictions of the two. *User- and item-based similarity fusion* proposed in Wang et al. [152] is another NNCF method that makes predictions by using a weighted sum of three sources: a user-based approach that bases its prediction on other similar users; an item-based approach that bases its prediction on other similar items; and a third source that bases its prediction on similar users' ratings on other similar items, where each contributing rating is weighted by a Euclidean combination of its user- and item-similarity with the target query.

This class of methods mostly targets the sparsity problem described in section 2.4.1.2 by following the philosophy that when the dataset suffers from extreme sparsity, a bigger chunk of the dataset should be exploited by analysing both the user and the item orientation. Algorithmically, such methods are predominantly along the lines of averaging user- and item-based predictions in a close to linear fashion, resulting in all predictions being pushed towards the average point.

4.5 TASK: Experimental Analysis

This section presents experiments and evaluations of the TASK algorithm described in section 4.3. Section 4.5.1 firstly presents the general perfor-

mance of TASK; sections 4.5.2 and 4.5.3 then proceed to examine the TASK algorithm's two indigenous parameters — the orientation, and the number of iterations η; sections 4.5.4 and 4.5.5 evaluate the effect of dataset sparsity on the performance of TASK. Finally, TASK is compared with its related work: comparisons with other similarity weighting algorithms is presented in section 4.5.6; comparisons with other algorithms that combine user- and item-oriented predictions is presented in section 4.5.7.

4.5.1 The general performance of TASK

Table 4.5 presents the experimental results of user-oriented TASK and NNCF on all three rating datasets MLS, MLM, and JST described in section 3.1. The experiments use the skip-every-10^{th} partitioning protocol. The displayed results are the average of 10-fold cross validation. In terms of evaluation metrics, MAE and RMSE are used as predictive accuracy metrics; precision, recall, and F1-measure are used as binary recommendation metrics; NDCG (with $N = 30$) is used as ranked recommendation metrics. The binary and ranked recommendations are generated based on rating predictions, as explained in sections 3.3.2.1 and 3.3.3.1. A threshold cut-off of *threshold* $= 3$ is used for both the *predicted rating to binary recommendation* conversion and the *actual rating to binary preference* conversion, as explained in section 3.3.2.1. If a recommended item does not have an actual user rating in the test dataset to judge its true user preference, this recommendation is simply ignored, as explained in table 3.5.

Table 4.5: The general performance of TASK

Dataset	Algorithm	MAE	RMSE	Precision	Recall	F1	NDCG
MLS	TASK	**0.719**	**0.924**	**0.887**	**0.917**	**0.887**	**0.695**
	NNCF	0.737	0.938	0.886	0.907	0.881	0.668
MLM	TASK	**0.704**	**0.900**	**0.901**	**0.937**	**0.912**	**0.676**
	NNCF	0.718	0.912	0.899	0.919	0.902	0.655
JST	TASK	**0.674**	**0.788**	**0.812**	**0.896**	**0.838**	**0.389**
	NNCF	0.696	0.813	0.806	0.868	0.806	0.342

Table 4.6 formats the data of table 4.5 in a different way. It shows

the performance improvement of TASK over NNCF as a percentage of NNCF's performance. For prediction accuracy measures such as MAE and RMSE where a smaller value indicates a better performance, the presented values are "NNCF minus TASK"; for recommendation accuracy measures such as precision, recall, F1 and NDCG, where a larger value indicates a better performance, the presented values are "TASK minus NNCF". In this way, a positive value always indicates a positive improvement, and vice versa.

Table 4.6: The percentage improvement of TASK over NNCF

Dataset	MAE	RMSE	Precision	Recall	F1	NDCG
MLS	2.44%	1.49%	0.11%	1.10%	0.68%	4.04%
MLM	1.95%	1.32%	0.22%	1.96%	1.11%	3.21%
JST	3.16%	3.08%	0.74%	3.23%	3.97%	13.74%

Table 4.6 shows that TASK consistently outperforms NNCF in terms of predictive accuracy (MAE and RMSE), binary recommendation accuracy (F1-measure), and ranked recommendation accuracy (NDCG) on all three datasets. It is able to perform 2.44% better than NNCF in terms of MAE, which is a significant improvement in recommender system terms [47]. It makes more improvement on recall without sacrificing precision, resulting in a higher overall F1-value.[3] This means that TASK is able to better tell apart items above and below the rating of 3 for each user. It makes a significant improvement on NDCG, indicating that it is able to better capture the *relative* ranking between items on top of the better *abslolute* rating predictions reflected by the MAE.

4.5.2 How does TASK respond to the user and item orientations?

In contrast to most of the other methods that combine user- and item-oriented collaborative filtering [156, 152, 54], TASK is still orientation sensitive, meaning that user-oriented TASK is different from item-oriented

[3]The precision, recall, and F1 presented here are the average of 10-fold cross validation, thus the equality $F1 = 2 \times precision \times recall/(preciaion + recall)$ does not hold.

TASK. Table 4.7 presents the experimental results of TASK and NNCF with both the user and the item orientations on both the MLS and the MLM datasets. The partitioning protocol and evaluation metrics are the same as section 4.5.1.

Table 4.7: The effect of (user or item) orientation on the performance of TASK

Dataset	Algorithm	MAE	RMSE	Precision	Recall	F1	NDCG
MLS	TASK-user	**0.719**	**0.924**	0.887	**0.917**	**0.887**	**0.695**
	TASK-item	0.728	0.928	**0.892**	0.895	0.882	0.690
	NNCF-user	0.737	0.938	0.886	0.907	0.881	0.668
	NNCF-item	0.752	0.951	0.890	0.896	0.880	0.680
MLM	TASK-user	0.704	0.900	0.897	**0.937**	0.905	0.660
	TASK-item	**0.692**	**0.880**	**0.909**	0.916	**0.909**	**0.676**
	NNCF-user	0.718	0.912	0.899	0.919	0.902	0.655
	NNCF-item	0.707	0.899	0.906	0.914	0.901	0.661

Table 4.7 shows that, in terms of both predictive accuracy measures (MAE and RMSE) and recommendation accuracy measures (F1 and NDCG), both NNCF and TASK seem to perform better in a user-oriented setting on the MLS dataset, but perform better in an item-oriented setting on the MLM dataset. We speculate that this is due to the difference between the *orientation densities* of the two datasets. As specified in table 3.1, MLS has a denser user-orientation, with the average user possessing more ratings (106) than the average item (59); whereas MLM has a denser item-orientation, with the average item possessing more ratings (270) than the average user (166). Therefore, the results here can be translated into: TASK always performs better on the *denser* orientation (i.e. the user-orientation for MLS and the item-orientation for MLM).

Another interesting observation is regarding precision and recall. The results show that in terms of precision, the item-oriented TASK and NNCF always outperform their user-oriented counterparts regardless of the dataset and the orientation density, whereas the user-oriented algorithms always perform better on the recall.

4.5.3 How does TASK respond to the number of iterations?

The *number of iterations* is the number of iterative similarity computations to use as specified on page 81. With $\eta = 0$, TASK is effectively equivalent to the standard NNCF algorithm; with $\eta = 1$, the user similarity computation is weighted by item similarities, which are computed by the standard NNCF; with $\eta = 2$, the user similarity computation is weighted by item similarities, the computation of which is weighted by user similarities, which are computed by the standard NNCF, and so forth.

The computational complexity of TASK is polynomial to the number of data items with a degree equal to the number of iterations. In other words, it is exponential to the number of iterations. Due to hardware limitations, only experiments on the MLS dataset with up to three iterations were carried out. The results are shown in table 4.8.

Table 4.8: The effect of the number of iterations η on the performance of TASK.

	MAE	RMSE	Precision	Recall	F1	NDCG
$\eta = 0$	0.7367	0.9381	0.8864	0.9065	0.8811	0.6681
$\eta = 1$	0.7194	0.9241	0.8871	0.9173	0.8868	0.6947
$\eta = 2$	0.7161	0.9221	0.8877	0.9182	0.8860	0.7003
$\eta = 3$	0.7159	0.9220	0.8877	0.9182	0.8860	0.7006

The experiments show that TASK with one iteration makes a consistent improvement over zero iterations (i.e. the standard NNCF). However, the improvement made by $\eta = 2$ over $\eta = 1$ is much smaller, with improvements on the binary recommendation metrics (i.e. precision, recall, and F1-measure) only observable beyond four decimal points. The improvement made by $\eta = 3$ over $\eta = 2$ is even smaller, though still positive. Paired t-tests are also conducted between each pair of $\eta = x$ and $\eta = x + 1$ under the hypothesis of that $\eta = x + 1$ performs no worse (i.e. the same or better) than $\eta = x$ in terms of MAE. Both $(\eta = 0, 1)$ and $(\eta = 1, 2)$ passed the 95% confidence level, but $(\eta = 2, 3)$ did not. The combined experimental results suggest that the improvement potential converges very fast as the number of iterations goes up. Therefore, based on the combined consideration of computational complexity and improvement potential, TASK with exactly one iteration is recommended. In the rest of this the-

sis, all experiments on TASK are performed with one iterations without further specifications.

4.5.4 How is TASK affected by different types of dataset sparsity?

There are two aspects to dataset sparsity — the similarity computation sparsity, and the rating prediction sparsity. They can be controlled separately through a combination of the held-out-k and the given-n protocol, as described in section 3.2.3.

Figure 4.2(a) presents the MAE of user-oriented TASK algorithm with varying degrees of the similarity calculation sparsity (as shown in the columns) and the rating prediction sparsity (as shown in the rows); figure 4.2(b) shows the MAE *improvements* of TASK over NNCF under the corresponding sparsity settings. The sparsity is controlled using the user-oriented given-n protocol, with the resulting sparsity value for each n indicated in table 3.2.[4] The experiments are performed on the MLS dataset. The displayed results are the average of $(80 = 16 \times 5)$-fold cross validation, where the *held-out*-25% protocol commences a 16-fold dataset partitioning, each of which incorporates a 5-fold cross validation over the given-n protocol to eliminate noise and randomness in the results.[5]

The table cells are coloured based on the cell values. In figure 4.2(a), a smaller value and a lighter colour corresponds to a better performance; in figure 4.2(b), a larger value and a greener colour indicates a more substantial improvement of TASK over NNCF.

The experiments shows that the performance of TASK is severely affected by the similarity calculation sparsity (SCS). Both the prediction accuracy indicated by (the inverse of) MAE and the improvement margin over NNCF are monotonically decreasing with respect to the SCS, which

[4] The values displayed in figure 3.2 are the dataset sparsity of applying given-n on the entire dataset. This value is slightly bigger (i.e. more sparse) than applying given-n to control only the similarity calculation sparsity or the rating prediction sparsity. However, the general gesture can be inferred.

[5] Detailed explanations of the relationship between given-n, held-out-k, and cross validation are described in section 3.2.3.

		Similarity Calculation Sparsity				
		Given-2	Given-5	Given-10	Given-20	Given-all
Rating Prediction Sparsity	Given-2	1.097	1.095	1.067	1.027	0.918
	Given-5	1.099	1.088	1.046	0.982	0.875
	Given-10	1.094	1.086	1.022	0.957	0.825
	Given-20	1.088	1.066	1.014	0.858	0.793
	Given-all	1.082	1.029	0.959	0.832	0.771

(a) The MAE performance of TASK (the lower the better).

		Similarity Calculation Sparsity				
		Given-2	Given-5	Given-10	Given-20	Given-all
Rating Prediction Sparsity	Given-2	-0.547%	-0.302%	0.094%	0.292%	0.872%
	Given-5	-0.546%	-0.216%	0.191%	0.306%	1.257%
	Given-10	-0.457%	-0.092%	0.196%	0.522%	2.060%
	Given-20	-0.459%	-0.094%	0.395%	0.933%	2.774%
	Given-all	-0.185%	-0.097%	0.626%	1.322%	2.661%

(b) The percentile improvement made by TASK over NNCF in terms of MAE (the higher the better).

Figure 4.2: MAE of controlled robustness experiments of TASK.

is inversely correlated to the n in given-n. When the SCS is extremely high, as in the given-2 and given-5 scenarios shown in columns 1 and 2, the MAE performance of TASK is slightly worse than that of NNCF, as manifested by the negative (red) values in figure 4.2(b). This makes sense, because the functional advantage of the TASK heuristic is that it is capable of delving into the niche of the dataset and exploring previously ignored details, which naturally requires a reasonably amount of data for the *niches* and *details* to manifest. Another way to look at it is that, TASK takes on extra steps to filter out item-irrelevancy and preference-imbalance. This extra filtering process requires a buffer of data, or else the item-irrelevancy and preference-imbalanced observed by TASK may be inaccurate and due to random noise. Even if the filtering process is accurate, when the dataset is extremely sparse, the remaining data after the dataset has been rid of item-irrelevancy and preference-imbalance would become even more sparse. The negative performance caused by increased sparsity may overpower the positive performance caused by item-irrelevancy

and preference-imbalance filtering, resulting in an overall decrease of prediction accuracy. On the bright side, TASK is only severely affected to be worse than the standard NNCF when extreme sparsity exists. In the given-10 scenario presented in column 3, although the sparsity of the dataset is 0.9941, which still reaches the "*extremely sparse*" level of over 0.99, TASK is already able to leverage the existing data and achieve a performance improvement over NNCF.

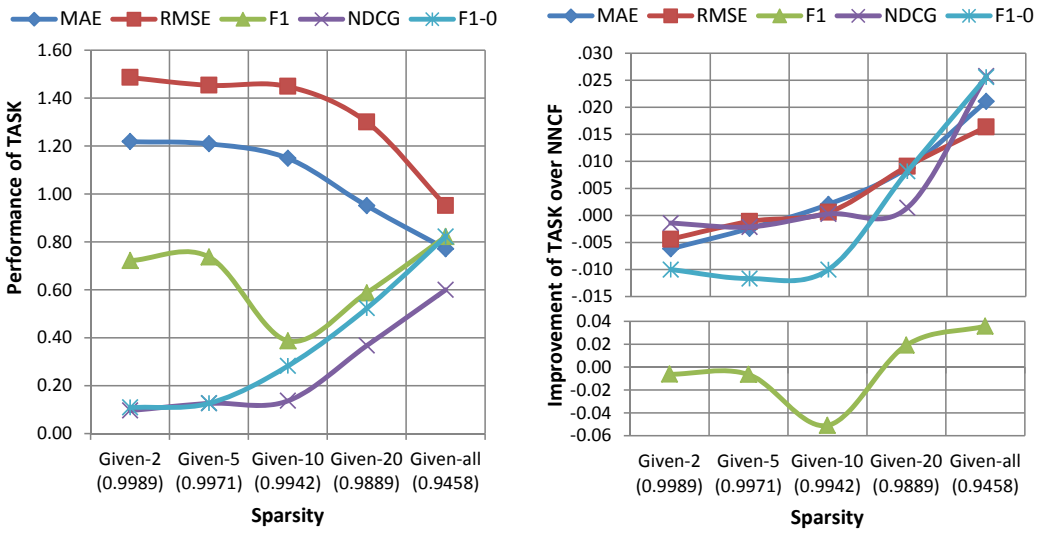
The TASK algorithm is less affected by the rating prediction sparsity (RPS). The prediction accuracy (inverse of MAE) and the improvement potential of TASK are still monotonically decreasing with respect to RPS. However, when the SCS is low enough for the similarity to be calculated properly, as in the last three columns in the tables, TASK is still able to make a positive improvement over the NNCF despite the extremely high PRS sparsity introduced by the given-2 protocol. This indicates that the power of TASK entirely resides within the similarity computation, and indeed produces more appropriate similarities compared to the standard NNCF. More analysis on the similarity produced by TASK is presented in section 4.5.6.

4.5.5 How does TASK perform under different dataset sparsity in general?

This section investigates the effect of dataset sparsity on the performance of TASK from a general angle. Here, the given-n protocol is applied to the entire training dataset, as opposed to section 4.5.4 that only applies given-n on part of the dataset to control similarity computation sparsity and rating prediction sparsity separately. In reality, the two types of dataset sparsity are often quite similar, and both are similar to the overall sparsity of the entire dataset. Therefore, experiments like this provide a holistic view of how the algorithm performs under the specified sparsity value.

Figure 4.3 presents the performance of TASK (in (a)) and the improvements TASK makes over the standard NNCF (in (b)) on the MLS dataset. The MAE, RMSE, F1, and NDCF curves correspond to the performances under the respective evaluation metrics; the F1-0 curve is explained in

the next paragraph. In figure 4.3(a), the units of different curves in the same diagram are not the same, since they correspond to different evaluation metrics. So the y-coordinates of different curves should not be compared with each other. It is the growth of the curves with respect to the x-coordinate that should be looked at. In figure 4.3(b), a *positive* value indicates a *better* performance by TASK. This means for prediction accuracy measures like MAE and RMSE, the values presented are "NNCF minus TASK", since a smaller value indicates a better performance; whereas in F1-measure and NDCG, the values presented are "TASK minus NNCF", since a larger value indicates a better performance. Note that the improvement over F1 is plotted on a different y-scale to the other three curves due to the range difference.



(a) The performance of TASK.

(b) Improvements over NNCF.

Figure 4.3: The performance of TASK under different dataset sparsity.

Figure 4.3(a) shows a consistent improvement of MAE, RMSE, and NDCG as the data sparsity decreases. F1 also shows a monotonic increase beyond given-10. Its bizarrely good performance with given-2 and given-5 puzzled us at first. However, further investigations show that it is caused by extremely low coverage — very often (over 85% of the time for given-2), the algorithm is not able to provide any item recommendations to a given user. Since the F1-value presented here is the average of only the *valid*

F1-values over all users and all (cross validation) folds, 85% of the more difficult cases that hurt the F1 accuracy of given-10, given-20, and given-all are simply ignored and goes unpunished, resulting in the seemingly high performance of given-2 and given-5. If the F1 of the uncovered cases are set to 0 instead of simply being ignored, the resulting averaged F1 on given-2 and given-5 becomes much lower, as indicated by the "F1-0" curve in figure 4.3, and the overall monotonic increase of F1 can be observed.

Figure 4.3(b) shows that the improvements of TASK over NNCF in terms of MAE, RMSE, and NDCG also increase monotonically as the data sparsity decreases. With given-10, although the dataset sparsity still reaches the "extremely sparse" level of over 0.99, TASK is already able to make a positive improvement over NNCF in terms of all three metrics. The curve of the F1-measure exhibits the same pattern to the F1 curve in figure 4.3 due to the same reason explained in the previous paragraph. It also seems like it is more difficult for TASK to make an improvement in terms of the binary recommendation accuracy (i.e. F1) than the rating-related accuracies (MAE, RMSE, and NDCG). [6] However, given a reasonable amount of data, TASK is still able to make a positive improvement in F1.

4.5.6 How does TASK compare with other similarity weighting algorithms?

In this section, the TASK algorithm is compared with two other algorithms that also incorporate item-based weighting in user similarity computation: the inverse user frequency (IUF),[7] and the variance weighting (VW). Both algorithms are described in details in section 4.4.1.[8]

Figure 4.4 present the performance comparison of user-based TASK, NNCF, VW, and IUF on the MLS dataset. The displayed results are the av-

[6] NDCG measures the ranked recommendation accuracy, however, it still utilises the prediction *ratings* in its calculation, as described in section 3.3.3.

[7] The IUF implemented here uses a threshold cut of $r = 3$ to decide if a user voted in favour of an item and in turn whether to apply the item weight, as explained in 4.4.1.

[8] The other related work *automatic weighting scheme* is not included due to incomplete implementation details.

erage of 10-fold cross validation with skip-every-10^{th}. It shows that TASK is able to achieve better performance than NNCF in terms of all the displayed metrics, whereas IUF and VM, albeit also using item weights in user similarity computations, fail to do so. This is consistent with the findings in their original publications.

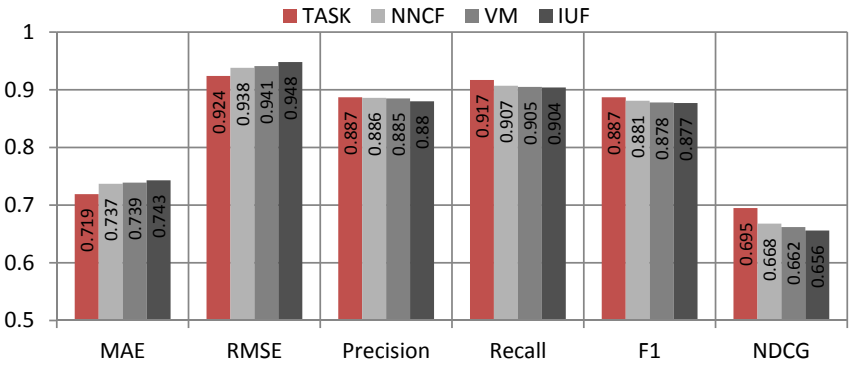


Figure 4.4: Comparison of TASK with other similarity weighting algorithms.

Figure 4.5 plots the user similarities of NNCF, TASK, IUF, and VW against individual user-user pairs, with the x-axis being an imposed user-user pairing index ordered by descending y-coordinates (i.e. descending user similarity) of NNCF. The MLS dataset contains 943 users, resulting in 422,710 unique pairs of calculable user-user similarities,[9] which make up the x-axis. Each y-axis value is the average similarity of 10-fold cross validation with skip-every-10^{th}.

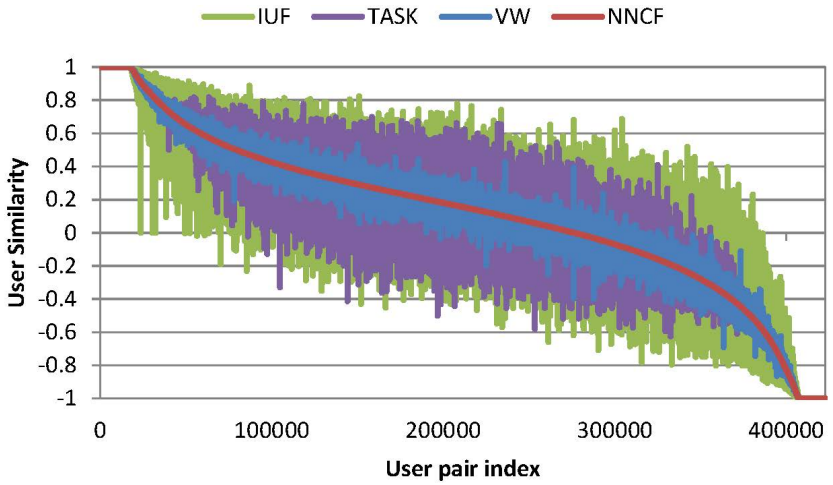


Figure 4.5: User Similarity Comparison of NNCF, TASK, IUF and VW

One thing this figure shows is the *variance* of the three item weighting mechanisms — IUF, TASK, and VW — against the original NNCF similarities. According to the figure, VW is the most conservative and has the smallest deviation to NNCF; IUF is the most aberrant with the biggest deviation; TASK is in between the two. Given the fact that TASK is the only algorithm that makes a positive improvement over NNCF, it would seem that there is a *Goldilocks zone* of performance improvement with respect to the similarity deviation to NNCF — too much or too little deviation both cause a decrease in prediction and recommendation accuracy. However, to make this observation into an allegation would require the substantiation of more experiments over a wider variety of algorithms. It is only pointed out here as an interesting observation regarding VW, IUF, and TASK.

A more significant observation is the loom-like shape exhibited by the TASK (purple) curve. This means that TASK is more in line with NNCF at the two ends when the similarities approach 1 and -1. Upon further investigations, it appears that the extreme similarities (at both ends) are correlated with, or are largely caused by a lack of common ratings. Figure 4.6 plots the relationship between NNCF user similarities and the number of common ratings between users. Although there are on average 16.004 common ratings between all user pairs, there are only 1.577 common ratings between users when the resulting NNCF similarities are over 0.9 or below -0.9. It is a positive attribute that, when there is not enough data to back up the similarity computation, TASK is able to converge to the simpler model of standard NNCF, instead of keep piling up complexities over the inadequate thus noisy data, like IUF and VW do.

A clearer visualisation of this allegation is presented in figure 4.7, which plots the same data to that of figure 4.5 in a different format. It contains three separate subfigures for the three algorithms. In each subfigure, the x-axis represents the user similarity calculated by NNCF; the y-axis represents the difference between the similarity calculated by the target algorithm (i.e. TASK, VW, or IUF) and that of NNCF. Figure (a) shows that TASK deviates from NNCF in the middle range — when there are more

[9] 943 users would result in $444,153 = \frac{943 \times 943 - 943}{2}$ pairs of irreflexive and asymmetric user-user relations. 21,443 of those are not calculable due to a lack of common ratings between the involved users, resulting in 422,710 pairs of calculable user-user similarities.

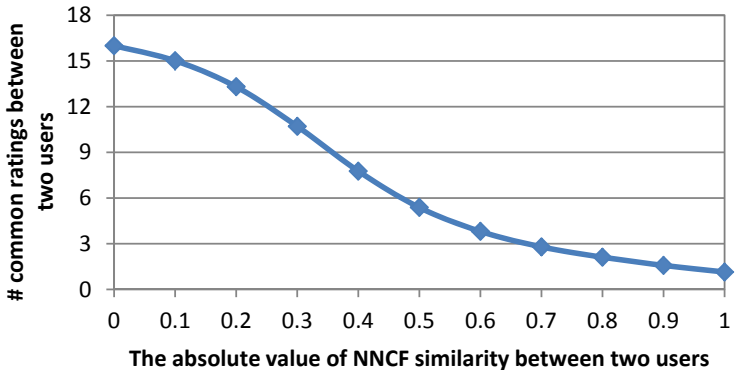
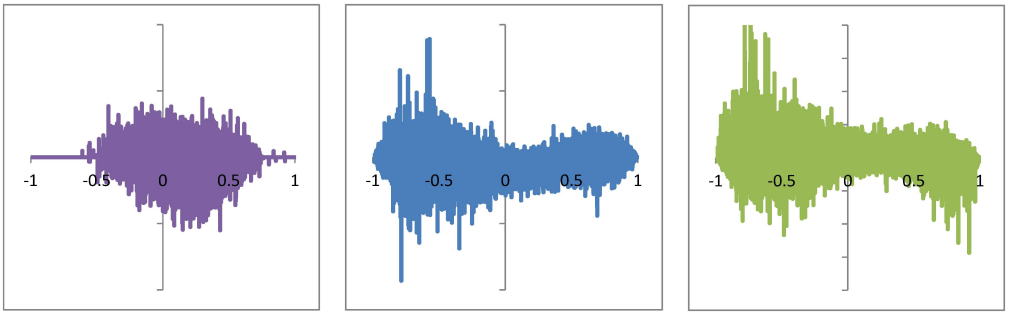


Figure 4.6: The number of common ratings between users w.r.t. user similarities

common ratings to support more complex similarity calculations. Both VW and IUF do the opposite, as shown in figures (b) and (c). They exhibit a bigger deviation from NNCF at both ends when there are fewer common ratings, and a smaller deviation in the middle where there are more.



(a) TASK (b) VW (c) IUF

Figure 4.7: Similarity differences over NNCF. The x-axis is the user similarity calculated by NNCF; the y-axis is the difference between the similarity calculated by the target algorithm (i.e. TASK, VW, or IUF) and that of NNCF.

To summarise, among the three similarity weighting algorithms presented here, TASK outperforms both VW and IUF, and is the only algorithm that can make a positive improvement over NNCF under all metrics tested. It appears that one of the contributing factor to this is TASK is able to automatically converge to the simpler NNCF model under insufficient data, whereas the other two algorithms do the opposite. Apart from this, we believe that the true advantage of TASK over IUF and VM is its *target awareness*, which tailors the item weights to the target similarity

calculation, whereas IUF and VW simply impose a canonical item weight indiscriminately across all situations.

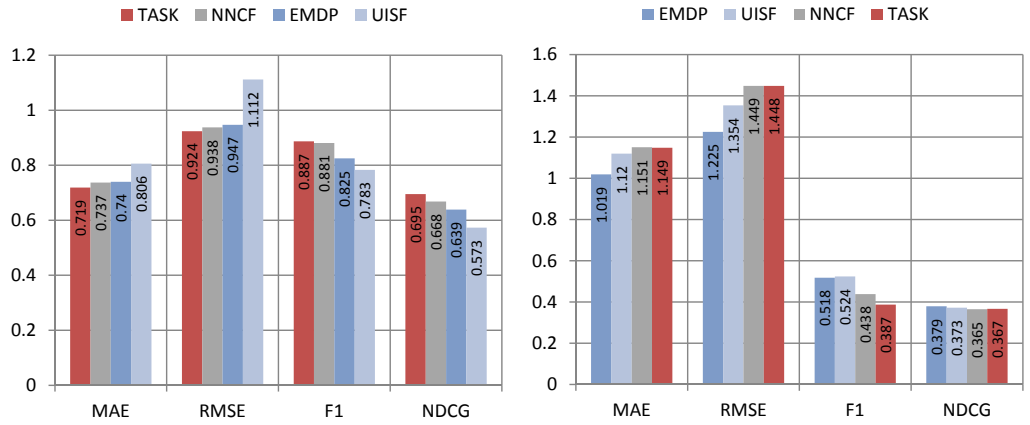
4.5.7 How does TASK compare with other dual-orientation algorithms?

TASK can be considered as an algorithm that combines user and item-oriented NNCF. Therefore, it is interesting to see how it compares with other algorithms of this type. This section compares user-oriented TASK and NNCF with *effective missing data prediction* (EMDP) [84][10] and *user and item-based similarity fusion* (UISF) [152], both of which are described in details in section 4.4.2 as part of the closely related work.

Figure 4.8 shows the experimental results of the four algorithms on the MLS dataset. Figure (a) displays general experimental results under skip-every-10^{th}; figure (b) shows robustness performance under the extreme dataset sparsity of 0.994 provided by given-10. Results for TASK are marked in red, NNCF marked in grey, and the other dual-orientation algorithms marked in different shades of blue. The algorithms in figure (a) are sorted in descending order of performance.

The results show that both EMDP and UISF make a positive improvement over both TASK and NNCF under extreme sparsity, but perform worse than both TASK and NNCF in regular situations. The offsetting behaviour between TASK and other dual-orientation algorithms is not surprising, because they actually have opposite design objectives. TASK focuses on improving the general prediction accuracy by applying an additional layer of filtering that uses item similarities to eliminate the item-irrelevancy and preference-imbalance in user similarity computation, thus can be thought of as a *multiplication* of the item-based and user-based NNCF; whereas the conventional dual-orientation approaches such as EMDP

[10]The EMDP paper includes two heuristics: *correlation significance weighting* (CSW), and *effective missing data prediction* (EMDP). Here only the EMDP part is implemented, because CSW is a separate heuristic that has nothing to do with combining user- and item-oriented collaborative filtering. It is a heuristic orthogonal to, thus can be combined with NNCF, TASK, EMDP, and UISF. Detailed explanation of the CSW heuristic is presented in the last paragraph of section 4.4.1.



(a) General Experiments (b) Sparsity Experiments (sparsity=0.994)

Figure 4.8: Comparison of TASK with other dual-orientation algorithms.

and UISF focus on improving the chance of producing a prediction under extreme dataset sparsity by effectively combining the predictions of both user-based and item-based NNCF with the hope that at least one of them will be able to produce a prediction, or that in the cases where they both do, "averaging" their predictions to reduce the noise and randomness caused by insufficient data. Therefore, they can be thought of as an *addition* of item-based and user-based NNCF. Different focus, opposite effects.

This explanation is corroborated by the standard deviations of the predicted ratings, as shown in figure 4.9. Both EMDP and UISF show a much lower standard deviation of their rating predictions compared to TASK and NNCF, suggesting that their predictions are "blurred" towards the global average. Such behaviour may help under extreme sparsity, but is not optimal on regular datasets.

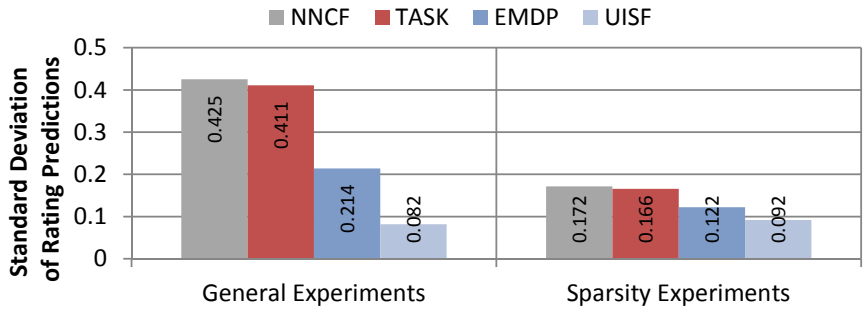


Figure 4.9: The standard deviation of rating predictions.

4.6 PANDA: Analysing the effect of items indirectly involved in the similarity computation through biased identity averages

This section identifies, analyses, and proposes solutions to the *biased-average problem*. The problem is explained in section 4.6.1. Two solutions — *partial averaging* (PA) and *double averaging* (DA) are then proposed in section 4.6.2 and further discussed in sections 4.6.3 and 4.6.4 respectively. The general combined idea will also be referred to using the abbreviation PANDA, meaning "PA *and* DA".

4.6.1 The Biased-Average Problem

Standard NNCF calculates the similarity between two users as the Pearson's correlation between their ratings over items they have both rated, known as their *common ratings* over their *common items*. This is presented in equation 4.4 on page 74, and reformulated here as equations 4.11.

$$sim_{PC}\left(u_1, u_2\right) = \frac{\sum_{i \in I_{u_1} \cap u_2} r'_{u_1,i} \cdot r'_{u_2,i}}{\sqrt{\sum_{i \in I_{u_1} \cap Iu_2} {r'_{u_1,i}}^2} \cdot \sqrt{\sum_{i \in I_{u_1} \cap Iu_2} {r'_{u_2,i}}^2}} \tag{4.11}$$

where $r'_{u,i}$ is the *rating deviation* (RD) defined as:

$$r'_{u,i} = r_{u,i} - \overline{r_u} \tag{4.12}$$

and $\overline{r_u}$ is user's average rating over all items:

$$\overline{r_u} = \frac{1}{|R_u|} \cdot \sum_{i \in I_u} r_{u,i} \tag{4.13}$$

This approach has a potential mismatch — the similarity calculation of equation 4.11 is carried on the *common* ratings of the two users, namely $R_{u_1, i \in I_{u_1} \cap Iu_2}$ and $R_{u_2, i \in I_{u_1} \cap Iu_2}$, whereas the user deviation in equation 4.12 is calculated against the user average $\overline{r_u}$ on *all* items the user has rated, namely $R_{u_1, i \in I_{u_1}}$ and $R_{u_2, i \in I_{u_2}}$, as shown in equation 4.13. The calculation of $\overline{r_u}$ ignores the information that it is part of the similarity calculation

between two users u_1 and u_2, thus is not sensitive to which items are involved in the similarity computation, or more precisely, which of the items are *not* involved. The rated items that are not part of the "common items" could potentially incorrectly bias the averages, causing undesirable effects in the similarity computation.

Table 4.9 uses an illustrative example to clarify the problem. Assume two users u_1 and u_2 have exactly the same preferences thus should have a similarity of 1. They have both rated three items i_3, i_4, and i_5 and have rated them identically, as shown in the second and third rows of table 4.9. However, u_1 also rated two other items i_1 and i_2 that he disliked, whereas u_2 rated two other items i_6 and i_7 that he liked. Therefore, the average ratings of the users, as shown in the $\overline{r_u}$ column of table 4.9, are different despite their identical preferences. The rating deviations calculated against their general rating averages, as shown in the last two rows of table 4.9, would skew the otherwise consistent ratings, ultimately result in an incorrect similarity of 0.44.

		i_1	i_2	i_3	i_4	i_5	i_6	i_7	$\overline{r_u}$	sim(u_1, u_2)
Original User	u_1	1	1	2	4	5			2.6	-
Ratings	u_2			2	4	5	5	5	4.2	
Rating Deviations	u_1	-1.6	-1.6	-.6	1.4	2.4			-	.44
by equation 4.12	u_2			-2.2	-.2	.8	.8	.8	-	

Table 4.9: Users with identical preferences, thus should have a Pearson's correlation similarity of 1, but happen to have rated different sets of non-common items, and the standard NNCF outputs a similarity of 0.44.

4.6.2 Solution: Partial Average and Double Average (PANDA)

This section proposes two solutions to the biased-average problem. The fundamental idea is to obtain a user average that is not biased by the non-common items of the users. *Partial average*, as shown in equation 4.14b, calculates the user averages only using the *common ratings* between the two users; *double average*, as shown in equation 4.14c, calculates the user averages using ratings that have been "pre-adjusted" by the *item averages* of the items involved.

$$\overline{r_u} = \frac{1}{\mid I_u \mid} \sum\nolimits_{i \in I_u} r_{u,i} \tag{4.14a}$$

$$\overline{r_u}^{\text{partial}} = \frac{1}{\mid I_{u \cap v} \mid} \sum\nolimits_{i \in I_{u \cap v}} r_{u,i} \tag{4.14b}$$

$$\overline{r_u}^{\text{double}} = \frac{\sum_{i \in I_u} (r_{u,i} - \overline{r_i})}{\mid I_u \mid} + \overline{r} = \overline{r_u} - \overline{r_{i \in I_u}} + \overline{r} \tag{4.14c}$$

Figure 4.10 use twelve illustrative examples on two users and seven items to show the effects of $\overline{r_u}^{\text{partial}}$ and $\overline{r_u}^{\text{double}}$. The effects differ depending on the relation between the user similarity, users' voting habits, and user's preferences on their common and non-common items. The examples are presented as twelve tables in a format similar to that of table 4.9. Each example is based on the ratings of two users (u_1 and u_2) on seven items (i_1 to i_7), where u_1 has rated items i_1 to i_5, and u_2 has rated items i_3 to i_7. Items $\{i_3, i_4, i_5\}$ are their "*common items*", and items $\{i_1, i_2\}$ and $\{i_6, i_7\}$ are their "*non-common items*". The ratings are integers from 1 to 5, just like in the *MovieLens* datasets.

The first row of each table shows the item averages $\overline{r_i}$ of the seven items. Note that it may not equal to the average rating of u_1 and u_2 on that item, since there are unshown users in the dataset. The item averages are designed to indicate the general quality of the items. Mathematically, they are only used in the calculation of double average $\overline{r_u}^{\text{double}}$, but not $\overline{r_u}$ or $\overline{r_u}^{\text{partial}}$. Their effect is discussed in section 4.6.4.

The third and fourth rows of each table show the raw ratings of the two users, with their *cosine similarity* shown in the "similarity" column. The cosine similarity shown in equation 4.15 is another way of calculating user similarities [3]. It does not use rating deviations as in Pearson's correlation (see formula 4.12), but estimates the similarity between users' raw rating vectors, thus is not subject to the biased-average problem. In cosine calculations, in order to obtain a $[-1, 1]$ similarity range for comparison purposes with Pearson's similarities, the rating range is "*recentred*" from

$r \in [1, 5]$ to $\widetilde{r} \in [-2, 2]$ by subtracting 3 from each rating.

$$sim_{\text{cosine}}(u_1, u_2) = \frac{\sum_{i \in I_{u_1} \cap u_2} \widetilde{r}_{u_1,i} \cdot \widetilde{r}_{u_2,i}}{\sqrt{\sum_{i \in I_{u_1} \cap u_2} \widetilde{r}_{u_1,i}{}^2} \cdot \sqrt{\sum_{i \in I_{u_1} \cap u_2} \widetilde{r}_{u_2,i}{}^2}} \tag{4.15}$$

Each of the following three two-row-blocks, marked with different colours for reading convenience, shows the rating deviations or RDs, the corresponding rating averages used to calculate the RDs, and the resulting Pearson's similarities, when the RDs are calculated using the standard averages $\overline{r_u}$, partial averages $\overline{r_u}^{\text{partial}}$, and double averages $\overline{r_u}^{\text{double}}$ respectively.

For presentation convenience, the twelve examples in figure 4.10 can be viewed as a 4×3 *example matrix*, or e. Each example can be referred to by its index. For example, $e_{1,1}$ refers to the top-left example, $e_{4,3}$ refers to the bottom-right example, and so forth. The examples are designed to demonstrate twelve situations spanning four dimensions, which are shown as the four coloured blocks at the bottom of each example. The first dimension represents whether u_1 and u_2 are indeed positively correlated (green), unrelated (yellow), or inversely correlated (red). The optimal similarities in the three cases would be 1, 0, and -1 for both Pearson's and cosine similarity. The second dimension — voting habit — demonstrates the advantage of Pearson's similarity over cosine similarity; the third and fourth dimensions demonstrate the situations where the biased average problem becomes harmful to the similarity estimation. The dimensions are discussed further below.

Dimension 2 — Users' Voting Habits

The *voting habit* is the user's internal mapping of the rating scale to their true preference of the item. In a numerical rating scale from 1 to 5, a "positive" voter may use 5 and 3 to express like and dislike, whereas a "negative" voter may use 3 and 1 to express like and dislike.

When the two users have different voting habits, direct comparison of their rating vectors may lead to inaccurate similarity. For example, row 2

Item Avg.		2	4	2	3	4	2	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	5	1	3	5				1.00
	user2			1	3	5	1	5		
RD (Standard)	user1	-2.00	2.00	-2.00	0.00	2.00			3.00	1.00
	user2			-2.00	0.00	2.00	-2.00	2.00	3.00	
RD (Partial)	user1	-2.00	2.00	-2.00	0.00	2.00			3.00	1.00
	user2			-2.00	0.00	2.00	-2.00	2.00	3.00	
RD (Double)	user1	-2.50	1.50	-2.50	-0.50	1.50			3.50	1.00
	user2			-2.50	-0.50	1.50	-2.50	1.50	3.50	

Similar Users	Similar Voting Habits	Similar Preferences on Non-common Items	Similar Ranges of Common and Non-common Item Ratings

Item Avg.		2	4	3	3	4	2	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	5	1	3	5				0.00
	user2			5	3	5	1	5		
RD (Standard)	user1	-2.00	2.00	-2.00	0.00	2.00			3.00	0.00
	user2			1.20	-0.80	1.20	-2.80	1.20	3.80	
RD (Partial)	user1	-2.00	2.00	-2.00	0.00	2.00			3.00	0.00
	user2			0.67	-1.33	0.67	-3.33	0.67	4.33	
RD (Double)	user1	-2.30	1.70	-2.30	-0.30	1.70			3.30	-0.04
	user2			0.90	-1.10	0.90	-3.10	0.90	4.10	

Unrelated Users	Similar Voting Habits	Similar Preferences on Non-common Items	Similar Ranges of Common and Non-common Item Ratings

Item Avg.		2	4	3	3	3	2	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	5	1	3	5				-1.00
	user2			5	3	1	1	5		
RD (Standard)	user1	-2.00	2.00	-2.00	0.00	2.00			3.00	-1.00
	user2			2.00	0.00	-2.00	-2.00	2.00	3.00	
RD (Partial)	user1	-2.00	2.00	-2.00	0.00	2.00			3.00	-1.00
	user2			2.00	0.00	-2.00	-2.00	2.00	3.00	
RD (Double)	user1	-2.50	1.50	-2.50	-0.50	1.50			3.50	-0.83
	user2			1.50	-0.50	-2.50	-2.50	1.50	3.50	

Dissimilar Users	Similar Voting Habits	Similar Preferences on Non-common Items	Similar Ranges of Common and Non-common Item Ratings

Item Avg.		2	4	2	3	4	2	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	3	5	3	4	5				-0.20
	user2			1	2	3	1	3		
RD (Standard)	user1	-1.00	1.00	-1.00	0.00	1.00			4.00	1.00
	user2			-1.00	0.00	1.00	-1.00	1.00	2.00	
RD (Partial)	user1	-1.00	1.00	-1.00	0.00	1.00			4.00	1.00
	user2			-1.00	0.00	1.00	-1.00	1.00	2.00	
RD (Double)	user1	-1.50	0.50	-1.50	-0.50	0.50			4.50	1.00
	user2			-1.50	-0.50	0.50	-1.50	0.50	2.50	

Similar Users	Different Voting Habits	Similar Preferences on Non-common Items	Similar Ranges of Common and Non-common Item Ratings

Item Avg.		2	4	3	3	4	2	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	3	5	3	4	5				-0.45
	user2			3	2	3	1	3		
RD (Standard)	user1	-1.00	1.00	-1.00	0.00	1.00			4.00	0.00
	user2			0.60	-0.40	0.60	-1.40	0.60	2.40	
RD (Partial)	user1	-1.00	1.00	-1.00	0.00	1.00			4.00	0.00
	user2			0.33	-0.67	0.33	-1.67	0.33	2.67	
RD (Double)	user1	-1.30	0.70	-1.30	-0.30	0.70			4.30	0.02
	user2			0.30	-0.70	0.30	-1.70	0.30	2.70	

Unrelated Users	Different Voting Habits	Similar Preferences on Non-common Items	Similar Ranges of Common and Non-common Item Ratings

Item Avg.		2	4	3	4	3	2	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	3	5	3	4	5				-1.00
	user2			3	2	1	1	3		
RD (Standard)	user1	-1.00	1.00	-1.00	0.00	1.00			4.00	-1.00
	user2			1.00	0.00	-1.00	-1.00	1.00	2.00	
RD (Partial)	user1	-1.00	1.00	-1.00	0.00	1.00			4.00	-1.00
	user2			1.00	0.00	-1.00	-1.00	1.00	2.00	
RD (Double)	user1	-1.30	0.70	-1.30	-0.30	0.70			4.30	-0.76
	user2			0.70	-0.30	-1.30	-1.30	0.70	2.30	

Dissimilar Users	Different Voting Habits	Similar Preferences on Non-common Items	Similar Ranges of Common and Non-common Item Ratings

Item Avg.		2	2	2	3	4	4	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	1	1	3	5				1.00
	user2			1	3	5	5	5		
RD (Standard)	user1	-1.20	-1.20	-1.20	0.80	2.80			2.20	0.61
	user2			-2.80	-0.80	1.20	1.20	1.20	3.80	
RD (Partial)	user1	-2.00	-2.00	-2.00	0.00	2.00			3.00	1.00
	user2			-2.00	0.00	2.00	2.00	2.00	3.00	
RD (Double)	user1	-2.10	-2.10	-2.10	-0.10	1.90			3.10	0.90
	user2			-2.90	-0.90	1.10	1.10	1.10	3.90	

Similar Users	Similar Voting Habits	Different Preferences on Non-ommon Items	N/A

Item Avg.		2	2	3	3	4	4	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	1	1	3	5				0.00
	user2			5	3	5	5	5		
RD (Standard)	user1	-1.20	-1.20	-1.20	0.80	2.80			2.20	-0.12
	user2			0.40	-1.60	0.40	0.40	0.40	4.60	
RD (Partial)	user1	-2.00	-2.00	-2.00	0.00	2.00			3.00	0.00
	user2			0.67	-1.33	0.67	0.67	0.67	4.33	
RD (Double)	user1	-1.90	-1.90	-1.90	0.10	2.10			2.90	-0.01
	user2			0.50	-1.50	0.50	0.50	0.50	4.50	

Unrelated Users	Similar Voting Habits	Different Preferences on Non-ommon Items	N/A

Item Avg.		2	2	3	3	3	4	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	1	1	3	5				-1.00
	user2			5	3	1	5	5		
RD (Standard)	user1	-1.20	-1.20	-1.20	0.80	2.80			2.20	-1.00
	user2			1.20	-0.80	-2.80	1.20	1.20	3.80	
RD (Partial)	user1	-2.00	-2.00	-2.00	0.00	2.00			3.00	-1.00
	user2			2.00	0.00	-2.00	2.00	2.00	3.00	
RD (Double)	user1	-2.10	-2.10	-2.10	-0.10	1.90			3.10	-0.84
	user2			1.10	-0.90	-2.90	1.10	1.10	3.90	

Dissimilar Users	Similar Voting Habits	Different Preferences on Non-ommon Items	N/A

Item Avg.		2	2	2	3	4	2	2		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	1	1	3	5				1.00
	user2			1	3	5	1	1		
RD (Standard)	user1	-1.20	-1.20	-1.20	0.80	2.80			2.20	1.00
	user2			-1.20	0.80	2.80	-1.20	-1.20	2.20	
RD (Partial)	user1	-2.00	-2.00	-2.00	0.00	2.00			3.00	1.00
	user2			-2.00	0.00	2.00	-2.00	-2.00	3.00	
RD (Double)	user1	-2.10	-2.10	-2.10	-0.10	1.90			3.10	1.00
	user2			-2.10	-0.10	1.90	-2.10	-2.10	3.10	

Similar Users	Similar Voting Habits	Similar Preferences on Non-common Items	Different Ranges of Common and Non-common Item Ratings

Item Avg.		2	2	3	3	4	2	2		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	1	1	3	5				0.00
	user2			5	3	5	1	1		
RD (Standard)	user1	-1.20	-1.20	-1.20	0.80	2.80			2.20	0.36
	user2			2.00	0.00	2.00	-2.00	-2.00	3.00	
RD (Partial)	user1	-2.00	-2.00	-2.00	0.00	2.00			3.00	0.00
	user2			0.67	-1.33	0.67	-3.33	-3.33	4.33	
RD (Double)	user1	-1.90	-1.90	-1.90	0.10	2.10			2.90	0.03
	user2			1.30	-0.70	1.30	-2.70	-2.70	3.70	

Unrelated Users	Similar Voting Habits	Similar Preferences on Non-common Items	Different Ranges of Common and Non-common Item Ratings

Item Avg.		2	2	3	3	3	2	2		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg	similarity
Ratings	user1	1	1	1	3	5				-1.00
	user2			5	3	1	1	1		
RD (Standard)	user1	-1.20	-1.20	-1.20	0.80	2.80			2.20	-0.61
	user2			2.80	0.80	-1.20	-1.20	-1.20	2.20	
RD (Partial)	user1	-2.00	-2.00	-2.00	0.00	2.00			3.00	-1.00
	user2			2.00	0.00	-2.00	-2.00	-2.00	3.00	
RD (Double)	user1	-2.10	-2.10	-2.10	-0.10	1.90			3.10	-0.99
	user2			1.90	-0.10	-2.10	-2.10	-2.10	3.10	

Dissimilar Users	Similar Voting Habits	Similar Preferences on Non-common Items	Different Ranges of Common and Non-common Item Ratings

Figure 4.10: 12 examples to demonstrate the “biased average problem”

of the *example matrix* features three examples of a similar construction to the examples in row 1, except the users' voting habits are different, with u_1 being a positive voter and u_2 being a negative voter. This difference causes the raw-rating-based cosine similarities (in the grey cells) to drop from a perfect estimation of 1, 0, -1 in examples $e_{1,1}$, $e_{1,2}$, $e_{1,3}$ to an inaccurate estimation of -0.2, -0.45, and -1 in examples $e_{2,1}$, $e_{2,2}$, and $e_{2,3}$.

This inaccuracy can be fixed by using rating deviation-based similarity measures such as Pearson's correlation. For example, in examples $e_{2,1}$, $e_{2,2}$, and $e_{2,3}$ where there are voting habit differences, although the raw rating-based cosine measures in the grey cells are compromised, the rating deviation-based Pearson's measures in the blue cells still show perfect estimates of 1, 0, and -1 for correlated, unrelated, and opposite users.

Voting habit differences between users have been a long observed factor since Resnick et al. [120]. The proposed way of dealing with it is through *rating normalisation* in the prediction stage, as shown in formula 4.6. However, the effect of voting habit differences in the similarity calculation stage has never been explicitly pointed out. Based on their voting habits, users can be roughly classified as "positive" versus "negative" based on the *shift* of their internal voting scale, and "mild" versus "extreme" based on the *span* of their internal voting scale. For example, an "extreme" voter may use 5 and 1 to express dislike and like, a "positively mild" voter may use 4 and 3, and a "negatively mild" voter may use 3 and 2. In NNCF, rating normalisation in the prediction stage can only deal with span-based voting habit differentials. To deal with shift-based differentials, rating deviations instead of raw ratings need to be used in the similarity calculation stage. For this reason, this thesis claims that rating deviation is necessary in correlation-based similarity calculations, and Pearson's correlation is therefore superior to cosine similarity measure.

Dimension 3 — The Users' Non-common Preferences

This third dimension — *(different) non-common preferences* — refers to the situation where the non-common items the users happen to have rated (e.g. $\{i_1, i_2\}$ and $\{i_6, i_7\}$) happen to be very different items, so that even

if the users have identical preferences, as in example $e_{3,1}$, they would still have rated their non-common items very differently. This would cause their rating averages $\overline{r_{u_1}}$ and $\overline{r_{u_2}}$ to be biased towards the opposite directions by the non-common ratings, triggering the *biased-average problem*.

Rows 3 of the example matrix illustrates this situation by having u_1 rated $\{i_1, i_2\}$ unfavourably and u_2 rated $\{i_6, i_7\}$ favourably. In this situation, standard Pearson's correlation as shown in the blue cells tends to underestimate the correlation between users, resulting in similarities smaller than the truth — 0.61, -0.12, and -1 for correlated, unrelated, and inverse users.

This problem can be solved by calculating the rating deviations using unbiased averages, such as $\overline{r_u}^{\text{partial}}$ and $\overline{r_u}^{\text{double}}$. In examples $e_{3,1}$, $e_{3,2}$, and $e_{3,3}$, both partial average in the green cells and double average in the red cells are able to work around the "*non-common preference differentials*" and greatly improve the similarity estimations. Partial average is able to achieve perfect estimations of 1, 0, and -1. Double average estimates the similarities to be 0.9, -0.12, and -1, which is still an improvement, but not as consistent. The reason for this is explained in section 4.6.4.

Dimension 4 — Non-common vs. Common Ratings

The fourth dimension — *(different) common and non-common ratings* — demonstrates another scenario that would trigger the *biased average problem*. It refers to the situation where the averages of the users' non-common ratings are not vastly different from each other as in dimension 3, but are different from the average of the same user's common ratings.

Row 4 of the example matrix illustrates this situation by having both u_1 and u_2 rated their non-common items extremely negatively. In this situation, standard Pearson's correlation as shown in the blue cells tends to overestimate the correlation between users, resulting in similarities larger than the truth — 1, 0.36, and -0.61 for correlated, unrelated, and inverse users. Just like in dimension 3, this problem can also be solved by unbiased averages $\overline{r_u}^{\text{partial}}$ and $\overline{r_u}^{\text{double}}$. In the examples, partial average in the green cells is still able to achieve perfect estimations; double average in

the red cells is able to achieve substantial improvements subject to minor fluctuations.

The examples have demonstrated that, firstly, it is necessary to use rating deviations instead of raw ratings in the similarity computation; secondly, rating deviations will be biased by the *biased average problem*, which is caused by indirectly involving in the similarity computation users' non-common ratings, whose quality and biases we have no base line to compare and judge. Partial average and double average seem to provide a solution, but they also have their own sets of problems, which are explained in sections 4.6.3 and 4.6.4 respectively.

4.6.3 Analysis of Partial Average

Partial average $\overline{r_u}^{\text{partial}}$, which seems to be the perfect method in the examples, also has its own set of problems. Firstly, the average rating of u_1 becomes a variable of not only u_1, but also its counterpart u_2. Therefore, it needs to be recomputed for every other u_2 in the data set. This slightly reduces the efficiency of the recommendation.

Secondly and more seriously, it runs the risk of insufficient information when the number of common ratings is significantly smaller than the number of non-common ratings, in which case the common ratings alone may not provide a reliable picture of the users' voting habits, causing the partial average to be inaccurate.

Figure 4.11 uses two examples to clarify this problem. In the two examples, all four users are supposed to have very well-balanced voting habits. However, this can only be gauged by looking at all their ratings. In figure 4.11(a), users u_1 and u_2 are supposed to have similar voting habits but opposite tastes thus negative similarity. However, without the non-common ratings to justify their *de facto* similar voting habits, the partial average $\overline{r_u}^{\text{partial}}$ calculated solely based on the three common items would mistakenly attribute the difference in their ratings as a difference in their voting habits, which is neutralised in the rating deviations. Rating deviations calculated as such would magnify the micro-agreement between the users — that they both prefer i_5 slightly more than i_3 and i_4 — and consider the

users as positively correlated, resulting in a wrongly estimated similarity of 0.5 as shown in the green similarity cell.

Item Avg.		2	3	2	3	4	2	3		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg.	similarity
Ratings	user1	4	5	1	2	2				-0.83
	user2			4	4	5	1	2		
RD (Standard)	user1	1.20	2.20	-1.80	-0.80	-0.80			2.80	-0.78
	user2			0.80	0.80	1.80	-2.20	-1.20	3.20	
RD (Partial)	user1	2.33	3.33	-0.67	0.33	0.33			1.67	0.50
	user2			-0.33	-0.33	0.67	-3.33	-2.33	4.33	
RD (Double)	user1	0.50	1.50	-2.50	-1.50	-1.50			3.50	-0.56
	user2			0.10	0.10	1.10	-2.90	-1.90	3.90	

(a) Inversely correlated users appear to have a positive correlation based on common ratings alone.

Item Avg.		2	3	3	4	5	2	3		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg.	similarity
Ratings	user 3	1	2	3	4	5				0.20
	user 4			5	4	3	1	2		
RD (Standard)	user 3	-2.00	-1.00	0.00	1.00	2.00			3.00	0.20
	user 4			2.00	1.00	0.00	-2.00	-1.00	3.00	
RD (Partial)	user 3	-3.00	-2.00	-1.00	0.00	1.00			4.00	-1.00
	user 4			1.00	0.00	-1.00	-3.00	-2.00	4.00	
RD (Double)	user 3	-2.10	-1.10	-0.10	0.90	1.90			3.10	0.10
	user 4			1.90	0.90	-0.10	-2.10	-1.10	3.10	

(b) Positively correlated users appear to be inversely correlated based on common ratings alone.

Figure 4.11: Examples where the partial-average method fails, due to the fact that common ratings alone may not provide sufficient information to gauge the voting habit of users.

Figure 4.11(b) shows a similar but reversed example. u_3 and u_4 are supposed to have similar and well-balanced voting habits, and weakly positively correlated tastes, thus should have a small positive similarity. Pearson's correlation based on $\overline{r_u}$ and $\overline{r_u}^{\text{double}}$ is able to correctly gauge this, and produce similarity estimations of 0.2 and 0.1 respectively. However, partial average $\overline{r_u}^{\text{partial}}$, without the full spectrum of ratings to accurately gauge the users' voting habits, considers both users positive voters, and in turn magnifies their micro-disagreement in the rating deviations and considers the two users completely inversely correlated with a similarity of -1.

The problem can be worse. Figure 4.12 presents a scenario where it cannot be decided whether the two users indeed have similar taste and similar voting habits, but happen to have voted on non-common items of different qualities, like example $e_{3,1}$ in the example matrix of figure 4.10; or whether they indeed have different taste on items and different voting habits, with u_1 being a negative voter and u_2 being a positive voter. If the former was true, then standard Pearson's calculation suffers from the *biased average problem*, and partial average-based similarity in the green cell is more accurate; if the latter was true, then standard Pearson's similarity of -0.42 as shown in the blue cell is more accurate.

Item Avg.		2	2	2	3	3	4	4		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg.	similarity
Ratings	user1	1	1	3	4	4				1.00
	user2			3	4	4	5	5		
RD (Standard)	user1	-1.60	-1.60	0.40	1.40	1.40			2.60	-0.42
	user2			-1.20	-0.20	-0.20	0.80	0.80	4.20	
RD (Partial)	user1	-2.67	-2.67	-0.67	0.33	0.33			3.67	1.00
	user2			-0.67	0.33	0.33	1.33	1.33	3.67	
RD (Double)	user1	-2.70	-2.70	-0.70	0.30	0.30			3.70	0.55
	user2			-1.50	-0.50	-0.50	0.50	0.50	4.50	

Figure 4.12: An example where it is ambiguous whether the standard average or the partial average is better.

4.6.4 Analysis of Double Average

Double average does not suffer from the "incomplete voting habit" problem that partial average suffers from. However, it has its own problem, which is that similarities based on double averages are overly sensitive to the item averages of the items involved.

Figure 4.13 uses two examples to clarify this problem. The examples are otherwise identical to figure 4.12 except for the item averages. In figure 4.13(a), the item averages are made to be slightly more consistent with user ratings, in which case the double average-based similarity in the red cell is 1. In figure 4.13(b), the item averages are made to be different from user ratings, in which case the double average-based similarity in the red cell is -0.42.

The problem here is, double average-based similarities not only measure the preference similarity between users, but also the similarities between the user ratings and the general item averages. The bigger the differences between users' ratings and the item averages are, the smaller the final "user" similarities will be. This is not only unnecessary but also incorrect. Actually, if both users disagree with the general public in the same way, the disagreement should serve as an even stronger factor to promote their similarity, instead of punish it.

Item Avg.		1	1	3	3	4	5	5		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg.	similarity
Ratings	user1	1	1	3	4	4				1.00
	user2			3	4	4	5	5		
RD (Standard)	user1	-1.60	-1.60	0.40	1.40	1.40			2.60	-0.42
	user2			-1.20	-0.20	-0.20	0.80	0.80	4.20	
RD (Partial)	user1	-2.67	-2.67	-0.67	0.33	0.33			3.67	1.00
	user2			-0.67	0.33	0.33	1.33	1.33	3.67	
RD (Double)	user1	-2.70	-2.70	-0.70	0.30	0.30			3.70	1.00
	user2			-0.70	0.30	0.30	1.30	1.30	3.70	

(a) The item averages are consistent with user ratings.

Item Avg.		3	3	3	3	3	3	3		
		item 1	item 2	item 3	item 4	item 5	item 6	item 7	user avg.	similarity
Ratings	user1	1	1	3	4	4				1.00
	user2			3	4	4	5	5		
RD (Standard)	user1	-1.60	-1.60	0.40	1.40	1.40			2.60	-0.42
	user2			-1.20	-0.20	-0.20	0.80	0.80	4.20	
RD (Partial)	user1	-2.67	-2.67	-0.67	0.33	0.33			3.67	1.00
	user2			-0.67	0.33	0.33	1.33	1.33	3.67	
RD (Double)	user1	-2.10	-2.10	-0.10	0.90	0.90			3.10	-0.43
	user2			-1.70	-0.70	-0.70	0.30	0.30	4.70	

(b) The item averages are different from user ratings.

Figure 4.13: Two examples that are otherwise identical to figure 4.12 except for the item averages. Comparison of the two examples shows how double average-based similarity computations (in red) are affected by the item averages.

4.7 PANDA: Experimental Analysis

This section presents experimental analyses of the partial average (PA) and double average (DA) heuristics presented in section 4.6. 4.7.1 focuses on the general performance of partial and double average. 4.7.2 and 4.7.3 investigate different heuristics that selectively apply partial and double averages in the right circumstances to achieve better performance.

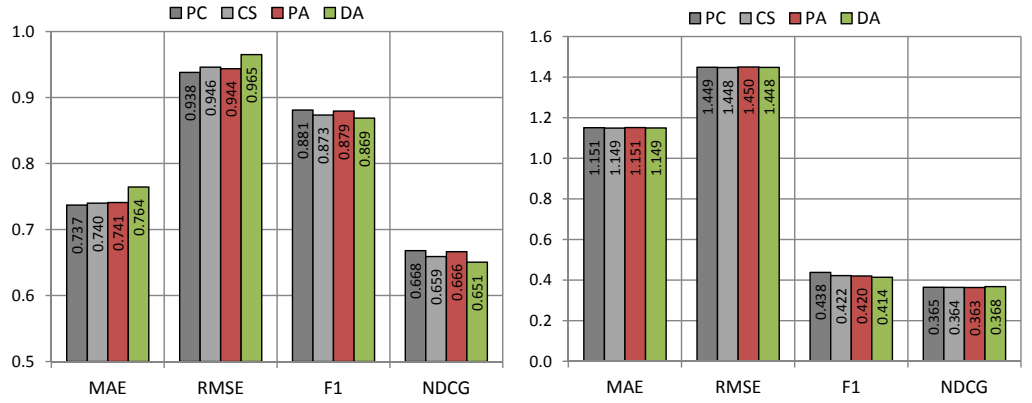
4.7.1 General performance

Figure 4.14 compares the general prediction and recommendation performance of four NNCF algorithms: Pearson's correlation (PC), cosine similarity (CS, as described in formula 4.15), partial average (PA), and double average (DA). The experimental settings are the same to that of figure 4.8. Figures (a) and (c) display general experimental results under skip-every-10^{th}; figures (b) and (d) show robustness performance under the extreme dataset sparsity of 0.994 provided by given-10. Results for PA and DA are marked in red and green; the baseline algorithms PC and CS are marked in different shades of grey.

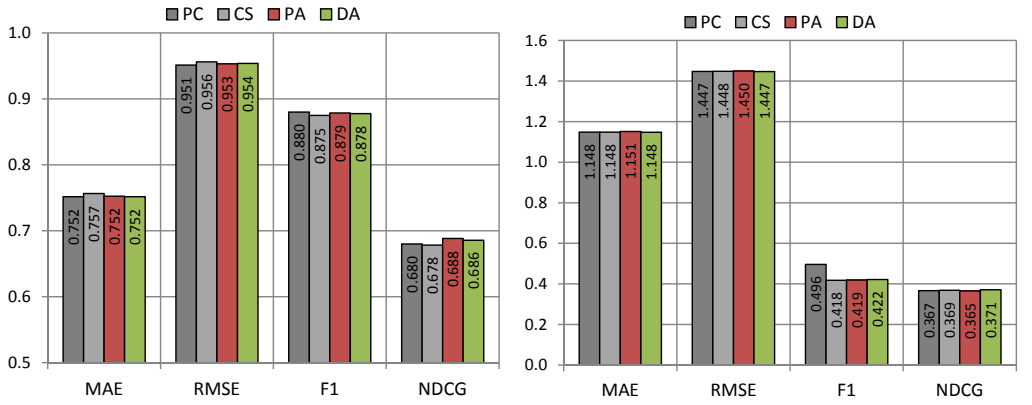
Judging from figure 4.14(a), PA and DA fail to make a positive improvement over the standard PC.[11] This is disappointing but expected. As already pointed-out in sections 4.6.3 and 4.6.4, although partial and double averages solve the *biased-average problem*, they each suffer from the *incomplete voting habit problem* and the *overvalued item averages problem* respectively. As the experimental results manifest, the negative effects of the newly introduced problems override the positive effects of solving the old problem, resulting in an overall decrease of performance.

Between DA and PA, user-oriented DA exhibits significantly worse performance compared to user-oriented PA, especially in terms of the prediction accuracies MAE and RMSE. This is consistent with the illustrative examples in figure 4.10, which demonstrates that DA tends to experience more fluctuation with minute change of circumstances due to the instabil-

[11] Recall that a positive improvement means smaller MAE and RMSE, and bigger F1 and NDCG.



(a) User-oriented General Experiments (b) User-oriented Sparse (.994) Experiments



(c) Item-oriented General Experiments (d) Item-oriented Sparse (.994) Experiments

Figure 4.14: Comparison of partial average (PA) and double average (DA) with the baseline algorithms of Pearson's correlation (PC) and cosine similarity (CS).

ity caused by the *overvalued item averages* problem. However, this performance difference is much smaller in item-oriented PA and DA, as shown in figure 4.14(c).[12] This is due to the fact that in the MLS dataset, each user has at least 20 ratings, whereas 8% of the items only have one rating, and 44% of the items have less than 20 ratings — the minimum amount a user has. This makes the item averages much less stable than the user averages, in turn causing the *overvalued item averages problem* to affect the performance of user-oriented DA much more than the *overvalued user averages* problem

[12]The paired t-tests over the hypotheses of that item-oriented PA performs better than item-oriented DA in terms of MAE only passed the 85% confidence level. In other words, item-oriented PA is not statistically significantly better than item-oriented DA.

affecting item-oriented DA.

Figures 4.14(b) and (d) show the performance comparison under the dataset sparsity of 0.994. In this case, the four algorithms exhibit much smaller performance differences. This is because in regular situations, the performance of PA and DA is affected by the noise caused by the over-filtering thus a lack of data. In sparse situations, all four algorithms are subject to noisy data due to the lack thereof, thus shrinking the performance gap.

Inspired by the fact that the performance gap diminishes as the dataset sparsity increases, the next two sections propose two different types of heuristics focusing on selectively applying partial and double averages only when the newly introduced problems are less likely to exist.

4.7.2 Piecewise partial average

The *incomplete voting habit problem* explained in section 4.6.3 states that when the number of common ratings between two users is too small, especially in comparison with the total number of ratings of the involved users, the common ratings alone may not be able to accurately capture the voting habits of users, resulting in the partial average being inaccurate, in turn hurting the performance of the recommender system. Partial average also has a much lower coverage of 81% compared to NNCF's 95%,[13] further hurting its prediction accuracy.[14]

This section investigates these problems by studying two parameters: *number of common ratings* or NCR between a pair of users, and the *rating range differential* or RRD between a pair of users. Suppose across all items the user has rated, user u_1's rating range is $[r_{u_1}^{\text{min}}, r_{u_1}^{\text{max}}]$, user u_2's rating range is $[r_{u_2}^{\text{min}}, r_{u_2}^{\text{max}}]$; u_1's ratings on the common items between u_1 and u_2 have the range of $[c_{u_1}^{\text{min}}, c_{u_1}^{\text{max}}]$, and u_2's common ratings have the range of

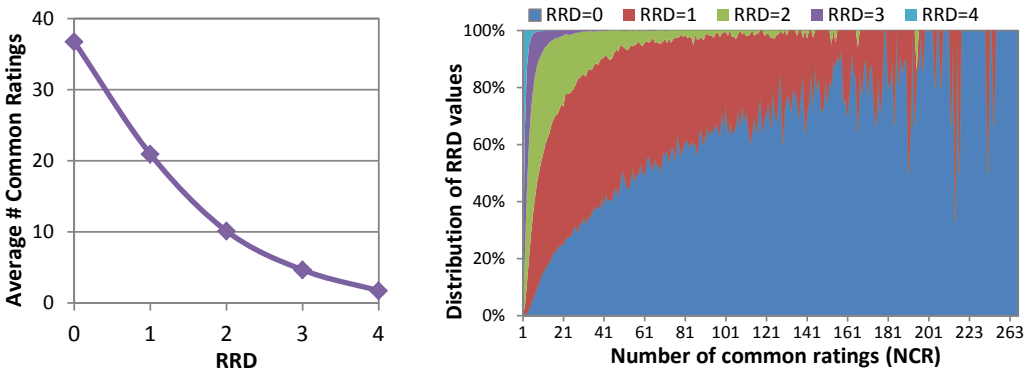
[13]The lower coverage is caused by all the common ratings being equal to the partial average, resulting in the rating deviations to be entire vectors of zeros, thus corrupting Pearson's correlation from producing a similarity.

[14]In our implementation, when a rating prediction is not able to be produced, a default rating of 3 is provided. In this way, the coverage is reflected in the prediction accuracy.

$[c_{u_2}^{\min}, c_{u_2}^{\max}]$. The rating range differential RRD is calculated as follows:

$$\begin{aligned} \text{RRD}(u_1, u_2) &= max\big(\ \text{RRD}(u_1), \text{RRD}(u_2)\ \big) \qquad \text{, where} \\ \text{RRD}(u_x) &= |r_{u_x}^{\max} - c_{u_x}^{\max}| + |r_{u_x}^{\min} - c_{u_x}^{\min}| \end{aligned} \tag{4.16}$$

Figure 4.15 demonstrates the relationship between RRD and NCR. Figure (a) plots the monotonically decreasing relationship between RRD and the averaged NCR; figure (b) plots for each NCR value in the x-axis the percentile distribution of RRD across all user pairs. An RRD value of zero means the rating range for both users overlap with their common rating range. According to the figures, it happens more often when there are abundant common ratings (and user ratings) to cover the intended range. On the other hand, an RRD value of 4 indicates that the common rating range and users' rating range are very different. It happens the most when the number of common ratings is extremely low, with the average NCR being only 1.72. 85% of the user pairs where there is only one common rating has an RRD of 4; the maximum NCR that is associated with an RRD of 4 is 21.



(a) RRD vs. NCR (b) NCR vs. RRD

Figure 4.15: The relationship between RRD and NCR.

The rest of this section proposes *piecewise partial average* or PPA, which uses regular average instead of partial average in two scenarios — (a) when the *number of common ratings* or NCR is less than a threshold α; and (b) when the *rating range differential* or RRD is more than a threshold β. This two scenarios are selected because they are prone to causing

the incomplete voting habit problem, and their preconditions can be easily tested during the recommendation process.

Figure 4.16(a) presents the distribution of user pairs with respects to the α and β thresholds. Each cell represents the percentage of user pairs with its NCR $\leq \alpha$ or its RRD $\geq \beta$. Since there is no user pairs with NCR ≤ 0 or RRD ≥ 5,[15] The first column and the first row of the table are equivalent to only applying RRD related heuristics and NCR related heuristics respectively.

β \ α	≤ 0	≤ 1	≤ 2	≤ 3	≤ 4	≤ 5	≤ 6	≤ 7	≤ 8	≤ 282
≥ 5	0	6.9%	14.5%	22.0%	28.9%	35.2%	40.9%	46.0%	50.4%	100.0%
≥ 4	10.1%	11.1%	16.1%	22.7%	29.2%	35.4%	41.0%	46.0%	50.4%	100.0%
≥ 3	25.5%	25.5%	27.3%	30.6%	34.9%	39.4%	43.9%	48.2%	52.1%	100.0%
≥ 2	53.2%	53.2%	53.4%	54.3%	55.7%	57.5%	59.5%	61.6%	63.7%	100.0%
≥ 1	84.3%	84.3%	84.4%	84.4%	84.6%	84.8%	85.2%	85.6%	86.0%	100.0%
≥ 0	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

(a) The percentage of user pairs that fall into the corresponding scenarios.

β \ α	≤ 0	≤ 1	≤ 2	≤ 3	≤ 4	≤ 5	≤ 6	≤ 7	≤ 8	≤ 282
≥ 5	0.741	0.741	0.740	0.739	0.738	0.737	0.735	0.735	0.736	0.737
≥ 4	0.739	0.739	0.738	0.738	0.737	0.735	0.735	0.735	0.736	0.737
≥ 3	0.738	0.738	0.738	0.736	0.735	0.734	0.733	0.736	0.736	0.737
≥ 2	0.736	0.736	0.736	0.736	0.736	0.735	0.735	0.736	0.736	0.737
≥ 1	0.737	0.737	0.737	0.737	0.737	0.737	0.737	0.737	0.737	0.737
≥ 0	0.737	0.737	0.737	0.737	0.737	0.737	0.737	0.737	0.737	0.737

(b) The MAE of *piecewise partial average* with the corresponding thresholds. A green cell means the performance is better than Pearson's correlation; a red cell means the performance is worse.

Figure 4.16: The effect of different α and β threshold.

Figure 4.16(b) presents the MAE of piecewise partial average or PPA with the respective α and β threshold. The experiments are conducted under the skip-every-10^{th} protocol on the MLS dataset. Each cell corresponds to applying normal average instead of partial average on a percentage of similarity computations equal to what is shown in the corresponding cell

[15] Cases where two users have no common ratings are not included in this study.

of figure 4.16(a). Note that when α is set to zero and β is set to the maximum, PAA is equivalent to PA; when α is set to the maximum or when β is set to zero, PAA is equivalent to PC. They are to be referred as the PA-side and the PC-side of the threshold scale respectively, and can be viewed as the diagonal line of figures 4.16(a) and 4.16(b).

In figure 4.16(b), a green colour indicates a better performance than Pearson's correlation, and a red cell means the performance is worse. The figure shows that the idea behind PAA is on the right track. With the right combination of α and β, piecewise application of partial average is able to achieve a statistically significant improvement over NNCF.[16]

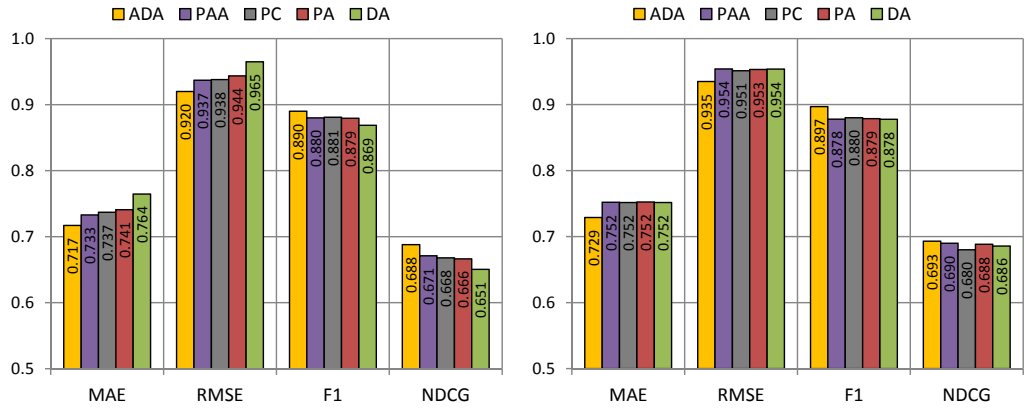
4.7.3 Applying AdaBoost on top of partial and double averages

Section 4.7.2 achieved performance improvement by using manually designed heuristics to selectively apply PA. This section investigate an *automatic* way of selectively applying partial, double, and normal average dynamically based on the data. To do this, *adaptive boosting* or *AdaBoost* [37] is used as an ensemble learner, with partial average, double average, and normal average (standard Pearson's correlation) as its base learners. *Boosting* [7] is an ensemble learning mechanism that generates complementary learners by actively training the next learner on the mistakes of the previous learners. *AdaBoost* is a particular version of boosting that is chosen here because it is suitable for situations when data is not abundant, and when the base learners are "weak but not too weak" [7].

Figure 4.17 presents the performance of AdaBoost compared with the three base learners (Pearsons, partial, and double), and the piecewise partial average. The experiments are carried out under the same settings to that of section 4.7.1.

The results show that AdaBoost is able to significantly improve the performance of Pearson's correlation in terms of both prediction accuracy and the recommendation accuracy in general situations, and is also able

[16]With $\alpha = 6$ and $\beta = 3$, the hypothesis of that PAA performs equal to or better than PC passd the paired t-test with a confidence level of 95%.



(a) User-oriented Results (b) Item-oriented Results

Figure 4.17: Comparison of Pearson's correlation (PC), partial average (PA), and double average (DA) with piecewise partial average (PAA) and Adaboosted average (ADA).

to improve the recommendation accuracy under extreme sparsity. When both PPA and AdaBoost are able to make a performance improvement, AdaBoost normally makes the bigger one.

The promising performance of AdaBoost is well expected. Given that the three types of averages have different weaknesses — with Pearson's correlation prone to the biased-average problem, partial average prone to incomplete voting habit, and double average prone to overvaluing item averages — ensemble learning is a natural path to investigate. The positive performance improvements corroborated the hypothesis that the three different type of user averages indeed possess different and offsetting inductive biases.

4.8 Conclusion

This chapter presented two improvements over the nearest neighbour-based collaborative filtering following the identification of three problems. The *target-aware similarity computation* (TASK) presented in section 4.3 adopts a dynamic relevancy-based item weighting mechanism to tackle the *item-irrelevancy problem* and the *preference-imbalance problem*. The *partial* and *double* averages presented in section 4.6 provide a rating deviation calculation

mechanism that is not biased by which of the non-common ratings the user happen to have rated, thus targets the *biased-average problem*.

As pointed out in section 4.4, there has been previous attempts to incorporate item weights in user similarity computations. What distinguishes TASK from other weighting mechanisms is its "*target-awareness*" — the item weights are aware of and are tailored to the target similarity computation. Another fundamental difference is that existing weighting mechanisms are all designed to discount the importance of universal views, whereas TASK is designed to explore a completely new aspect — the item-irrelevancy and the preference-imbalance. For this reason, one future direction is to investigate the possibility of combining TASK with other item-weighting mechanisms to achieve an even bigger improvement. Another direction is to find implementational heuristics to improve the computational speed of TASK.

The PANDA or *partial average and double average* heuristics do not have closely-related counterpart algorithms like TASK does. However, their differences to the standard Pearson's correlation is very similar to that of *cosine similarity*. On their own, partial and double averages are not able to make a performance improvement over the standard Pearson's correlation due to the "incomplete voting habit" problem and the "overvalued item averages" problem identified in sections 4.6.3 and 4.6.4. However, as presented in section 4.7, by carefully tuning the application preconditions and applying AdaBoost-based ensemble learning, performance improvements can be achieved.

Section 4.7.2 presented two heuristics that control the application precondition of partial average to try to avoid the incomplete voting habit problem and boost its performance. They are (a) a condition on the number of common ratings — a common size-based heuristic, and (b) a condition on the *rating range differential* (RRD). While conducting experiments on heuristic (b), the importance of the *range* of ratings starts to manifest. Sometimes, especially when the user's voting habits is in play, it is the *range* or the *distribution* of ratings instead of the specific rating values that is important. This leads to the development of the "distribution rating" idea presented in chapter 5.

Chapter 5

Distributional Rating

This chapter explores the concept of *distributional rating*. Instead of using scalar numbers between 1 to 5 to express user ratings and their voting habits, and using scalar numbers between -1 to 1 to represent user similarities, the idea is to use a *probabilistic belief distribution* to represent user ratings and their voting habits, and to use a *similarity distribution* to capture the users' relative predictive powers to each other.

The idea arose from the PANDA algorithm described in chapter 4. While trying to obtain a better formulation of users' voting habits using partial and double averages, it was discovered that the *range* of a user's ratings is as important as the mean at capturing the user's voting pattern. This led to the idea of using probabilistic distributional vectors instead of single scalar numbers to represent users' voting patterns, which expanded to the idea of using distributional vectors to represent a wider range of concepts including ratings, rating predictions, and user similarities.

The rest of the chapter is organised as follows: section 5.1 defines a recommendation-based probability space, which will be the basis of all discussions of chapters 5 and 6; sections 5.2 and 5.3 define *distributional rating* and *distributional voting profile* respectively, with their utility, purpose, and ramifications discussed in section 5.4; section 5.5 presents the workflow of using distributional rating and distributional voting profile to make recommendations; section 5.6 summarises related work; section 5.7 presents experimental results and analysis; section 5.8 concludes the chapter and indicates future work.

5.1 The Probability Space of Recommendation

In probability theory, a *probability space* is a mathematical model for studying the probabilities of *events* regarding a set of *variables*. It consists of four parts: 1) a set of *variables*; 2) a *sample space* Ω, which represents the set of all possible *outcomes* regarding the value assignments of the variables; 3) a set of *events* $E \subseteq \text{PowerSet}(\Omega)$, each of which is a subset of the sample space; 4) a *probability function* $\mathcal{P} : E \to [0, 1]$ that assigns probabilities to events.

As pointed out in section 4.1, the three fundamental elements of recommendation are user, item, and rating, which can be represented as three *variables* — the user-variable u that takes on values of user IDs; the item-variable i that takes on values of item IDs; and the rating-variable r that takes on numerical values in the range of $[r_{\text{min}}, r_{\text{max}}]$ (or $[1, 5]$ in the datasets of this thesis). The probability space of collaborative filtering recommendation can be defined regarding the three variables u, i, and r. The task of recommendation can be thought of as the study of such probability space.

In this probability space, an *outcome* is a $\langle u = u_x, i = i_y, r = r_z \rangle$ triple representing the outcome of that user u_x's rating on item i_y is r_z. This outcome can be an already-observed state of nature, such as an entry in the training dataset, or an unobserved theoretical state, such as a rating prediction. An *event* is a set of outcomes of interest where a probability can be assigned to express knowledge or belief. Examples include *single observed outcome-based events* such as "user Tom has given movie Avatar a rating of 4", *single unobserved outcome-based events* such as "we predict that Tom will rate movie *Inception* a 4.7", or *aggregated events* such as "Tom's average rating across all movies is 2.5". A training dataset such as table 5.1 or a set of predictions made by a recommender system are also examples of recommendation events for which probabilities can be assigned.

The rest of this section lists the important probability distributions within this probability space. In general, this thesis uses variable names as abbreviations to the *variable-value pairings* in probabilistic formulae. For example, $\mathcal{P}(u)$ is the abbreviation of $\mathcal{P}(u{=}u_x)$ for the generic user u_x, $\mathcal{P}(r|\ u, i)$ is the abbreviation of $\mathcal{P}(r{=}r_a|\ u{=}u_x, i{=}i_y)$ for the generic r_a, u_x, and i_y, and so forth.

User	Item	Rating
Tom	Titanic	Good
Tom	Inception	Bad
Jerry	Titanic	Bad
Jerry	Avatar	Good
Jerry	Inception	Bad

Table 5.1: An Illustrative Recommendation Dataset

The *general prior probabilities* regarding the three variables u, i, and r are $\mathcal{P}(u)$, $\mathcal{P}(i)$, and $\mathcal{P}(r)$, which can be computed by simply "counting-up" the respective instances in the dataset. In the illustrative dataset of table 5.1, the prior of the user being *Tom* would be $\frac{2}{5} = 0.4$; the prior of the item being *Avatar* would be $\frac{1}{5} = 0.2$, and the prior of a *Bad* rating would be $\frac{3}{5} = 0.6$, as illustrated in formulae 5.1a, 5.1b, and 5.1c.

$$\mathcal{P}(u = \text{Tom}) = \frac{\text{\# entries involving Tom as the user}}{\text{total \# entries}} = \frac{2}{5} \tag{5.1a}$$

$$\mathcal{P}(i = \text{Avatar}) = \frac{\text{\# entries involving Avatar as the item}}{\text{total \# entries}} = \frac{1}{5} \tag{5.1b}$$

$$\mathcal{P}(r = \text{Bad}) = \frac{\text{\# entries that has a } \textit{Bad} \text{ rating}}{\text{total \# entries}} = \frac{3}{5} \tag{5.1c}$$

The *conditional prior* probability distributions that involve two variables are $\mathcal{P}(i \mid u)$, $\mathcal{P}(u \mid i)$, $\mathcal{P}(r \mid u)$, and $\mathcal{P}(r \mid i)$. The other pairings $\mathcal{P}(u \mid r)$ and $\mathcal{P}(i \mid r)$ are not discussed because they are not of interest. These probabilities can also be calculated by counting up the instances, but only using a subset of the data that satisfy the specified conditions, namely as follows:

$$\mathcal{P}(u = \text{Tom} \mid i = \text{Avatar}) = \frac{\text{If Tom voted for Avatar}}{\text{\# entries regarding Avatar}} = \frac{0}{1} \tag{5.2a}$$

$$\mathcal{P}(i = \text{Avatar} \mid u = \text{Jerry}) = \frac{\text{If Jerry voted for Avatar}}{\text{\# entries regarding Jerry}} = \frac{1}{3} \tag{5.2b}$$

$$\mathcal{P}(r = \text{Bad} \mid u = \text{Jerry}) = \frac{\text{\# entries that Jerry rated "Bad"}}{\text{\# entries Jerry rated}} = \frac{2}{3} \tag{5.2c}$$

$$\mathcal{P}(r = \text{Bad} \mid i = \text{Titanic}) = \frac{\text{\# "Bad" ratings Titanic has}}{\text{\# entries regarding Titanic}} = \frac{1}{2} \tag{5.2d}$$

The conditional probability distribution of importance that involves all three variables is $\mathcal{P}(r|\, u, i)$ (i.e. $\mathcal{P}(r{=}r_a|\, u{=}u_x, i{=}i_y)$). It is the probability of the *outcome* that user u_x gives item i_y a rating of r_a. For observed outcomes such as a training data, this probability is 1; for unobserved outcomes such as a proposed prediction, the value of $\mathcal{P}(r{=}r_a|\, u{=}u_x, i{=}i_y)$ represents the degree of belief that a prediction of $r = r_a$ on user u_x and item i_y will be true.

5.2 Distributional Rating

In recommender system research, the rating datasets contain numerical rating data that use a *single scalar number* to represent the user's preference on an item. This section proposes the idea of *distributional rating* (DR), which uses a *discrete vector representation* of the probability distribution $\mathcal{P}(r|u, i)$ instead of a single scalar rating $r_{u,i}$ to model user u's preference on item i. A distributional rating is a 1×5 vector symbolised as $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u, i)$, whose entries are specified in formula 5.3:

$$\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u, i) = \begin{pmatrix} \mathcal{P}(r{=}1|u,i) & \mathcal{P}(r{=}2|u,i) & \mathcal{P}(r{=}3|u,i) & \mathcal{P}(r{=}4|u,i) & \mathcal{P}(r{=}5|u,i) \end{pmatrix} \tag{5.3}$$

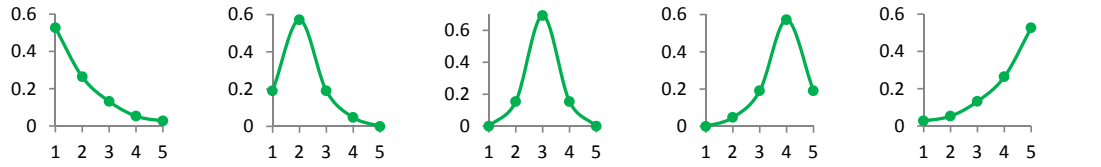
where $\mathcal{P}(r{=}x|\, u, i)$ is the probability that user u gives item i a scalar rating of x. For observed ratings such as the ones in the training dataset, this probability is 1 for $\mathcal{P}(r = r_{u,i}|\, u, i)$ and 0 for all other $\mathcal{P}(r \neq r_{u,i}|\, u, i)$, where $r_{u,i}$ is the *observed* scalar rating provided by the rating dataset. For example, an observed rating of 2 would have a distributional rating of $(\,0\;1\;0\;0\;0\,)$. For unobserved ratings such as a rating prediction made by the recommender system, the value of $\mathcal{P}(r = x|\, u, i)$ represents the level of confidence of the prediction being x. For example, instead of a scalar prediction of 2, the prediction can be a distributional rating of $(\,.19\;.57\;.19\;.05\;0\,)$.

Just like a scalar number has to be within the range of $[1, 5]$ to qualify as a valid scalar rating, for a 1×5 vector $(\,p_1\;p_2\;p_3\;p_4\;p_5\,)$ to qualify as a valid distributional rating, each p_i has to be within the range of $[0, 1]$, and the

sum of the vector has to add up to one:

$$\begin{aligned} 0 \leq & \mathcal{P}(r \mid u, i) \leq 1 && \text{for each } r \in \underset{\sim}{\mathrm{r}} \\ \sum_{r \in \underset{\sim}{\mathrm{r}}} & \mathcal{P}(r \mid u, i) = 1 \end{aligned} \tag{5.4}$$

To provide more perspective, figure 5.1 shows the visualisation of five distributional ratings whose maxima correspond to the scalar ratings of 1, 2, 3, 4, 5. In each figure, the distributional rating vector is shown at the bottom; the x-axis represents the rating values from 1 to 5; the y-axis represents the probability $\mathcal{P}(r{=}x|\ u, i)$ for each x-value. Take figure 5.1(a) for example, the decreasing curve communicates that in this prediction, the system thinks the user is most likely to rate the item a 1, however, the chances of the rating being $2, 3, 4,$ and 5 are non-zero with diminishing probabilities.



(a)(.53 .26 .13 .05 .02) (b)(.19 .57 .19 .05 0) (c)(0 .15 .69 .15 0) (d)(0 .05 .19 .57 .19) (e)(.02 .05 .13 .26 .53)

Figure 5.1: Distributional rating examples

Advantage over Scalar Ratings

Distributional ratings can also convey the *confidence* of a recommendation. Figure 5.2 presents five distributional ratings with the same maxima as those in figure 5.1, but with a flatter shape, indicating less confidence in the prediction.

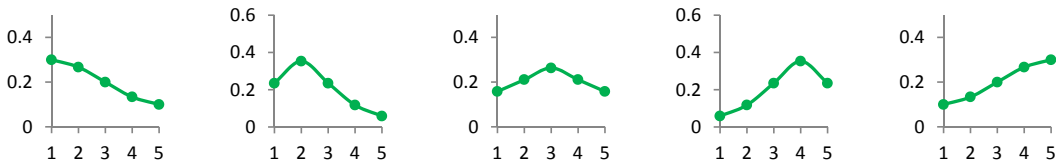
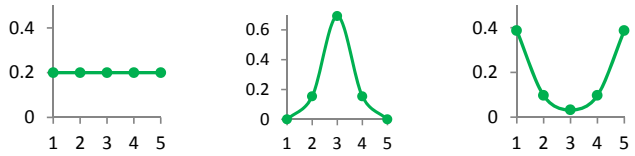


Figure 5.2: Distributional ratings similar to that of figure 5.1 but with bigger variances, indicating a lower confidence in the ratings.

Advantage over Gaussian Distribution

Imposing a Gaussian distribution over the rating range can also capture the confidence of the recommendation. However, compared to it, distributional rating has the added benefit of being able to communicate the "ambivalence" in a recommendation. For example, figure 5.3 shows three distributional ratings with the same expected value of $r_{u,i} = 3$, so in scalar terms they would be the same. However, they communicate very different messages. Rating (a) shows a complete lack of confidence, or an inability to produce any recommendations; rating (b) suggests a recommendation of 3 or mediocre; rating (c) communicates an ambivalent recommendation, stating that "*I think you will either like or hate this item, but either ways, it is definitely going to leave an impression*". For me personally, this is the spot-on recommendation for the movie *Borat*.



(a) (b) (c)

Figure 5.3: Three distributional ratings with the same expected value yet very different semantics. They demonstrate the expressiveness of distributional ratings over scalar ratings and a Gaussian distribution-based representation.

Advantage over a Mixture of Gaussian

In order to capture both the confidence and the "ambivalence" in the recommendations, an alternative representation would be to use a mixture of Gaussian. However, this would be very computationally expensive, and would require a lot of data to construct. The discrete vector representation used by distributional rating is light-weighted and carries relatively limited computational and storage overhead.

5.3 Distributional Voting Profile

Section 5.2 presents the use of a probability distribution $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u, i)$ instead of a scalar number $r_{u,i}$ to represent a single rating. Similarly, this section presents *distributional voting profile* or DVP, which uses a discrete representation of the probability distribution $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)$ instead of the scalar average $\overline{r_u}$ to represent user u's voting profile. It is defined as follows:

$$\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u) = (\ \mathcal{P}(r{=}1|u)\ \ \mathcal{P}(r{=}2|u)\ \ \mathcal{P}(r{=}3|u)\ \ \mathcal{P}(r{=}4|u)\ \ \mathcal{P}(r{=}5|u)\ ) \tag{5.5}$$

where, for each $x \in [1, 5]$, $\mathcal{P}(r{=}x|\ u)$ is calculated according to equation 5.6, which is an abstracted formalisation of equation 5.2c.

$$\mathcal{P}(r{=}x|\ u) = \frac{1}{|R_u|} \sum_{i \in I_u \text{where } r_{u,i}=x} 1 \tag{5.6}$$

For example, if a user rated 28 items, 11 of which he rated a 1, 7 of which he rated a 2, 4 of which he rated a 3, 3 of which he rated a 4, and 3 of which he rated a 5, his DVP would be a discrete vector-represented probability distribution of $(\ \frac{11}{28}\ \frac{7}{28}\ \frac{4}{28}\ \frac{3}{28}\ \frac{3}{28}\ )$ or $(\ 0.39\ 0.25\ 0.14\ 0.11\ 0.11\ )$.

This DVP can be visualised as in figure 5.4. Like distributional ratings, the x-axis represents the rating values from 1 to 5; the y-axis represents the probability $\mathcal{P}(r = x|\ u)$ for each corresponding x-value.

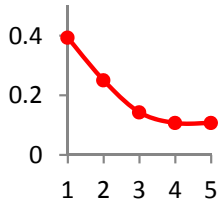


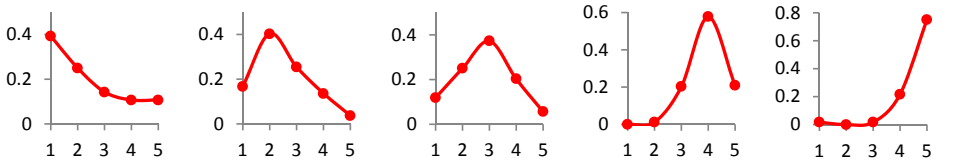
Figure 5.4: The visualisation of the distributional voting profile of $(0.39\ \ 0.25\ \ 0.14\ \ 0.11\ \ 0.11)$.

Semantically, the DVP represents the voting patterns of this user. The rest of this section explains some typical aspects of it, including *shift*, *spread*, *peaks*, and *simplified mental scale*.

Shift

The *shift* of a user's voting pattern can be conceptually interpreted as *positive* versus *negative* voters. In a numerical rating scale from 1 to 5, suppose there is a user who uses 5 and 3 to express like and dislike, and another user who uses 3 and 1 to express like and dislike, the first user would be a relatively positive user, and the second relatively negative. This difference can be caused by selective voting, different interpretations of the rating scale, or different personality.

Shift can be fairly well represented by the scalar average rating $\overline{r_u}$. Consequently, DVP is able to capture *shift* because it encompasses $\overline{r_u}$ — $\overline{r_u}$ can be directly calculated as the expected value of DVP. To provide more perspective, figure 5.5 plots five DVPs of real users in the MLM dataset. The DVPs are selected to cover different *shift* patterns, and are ordered by increasing values of the user's average rating.

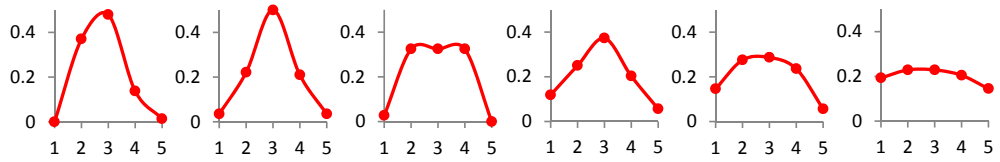


(a) $\overline{r_u}$=2.286 (b) $\overline{r_u}$=2.472 (c) $\overline{r_u}$=2.827 (d) $\overline{r_u}$=3.982 (e) $\overline{r_u}$=4.683

Figure 5.5: Distributional voting profiles of real users, ordered by increasing $\overline{r_u}$.

Spread

In addition to *shift*, there is also the *spread*, which can be mathematically captured by σ_u — the standard deviation (or variance) of user ratings. Conceptually, it can be construed as *broad* voting patterns versus *narrow* voting patterns. Figure 5.6 plots six real-user-based DVPs with similar *shift* (around $\overline{r_u} = 3$) but different *spread*. The left-hand-side patterns are "narrower" with smaller variances, demonstrating a strong tendency for the user to assign a single rating value (in this case $r = 3$) to the items; the right-hand-side patterns have larger variances and flatter shapes, demonstrating a more even use of all five rating values.



(a) σ_u=0.798 (b) σ_u=0.907 (c) σ_u=1.008 (d) σ_u=1.067 (e) σ_u=1.150 (f) σ_u=1.384

Figure 5.6: Distributional voting profiles with a similar *shift* of $\overline{r_u}$=3 but different *spread*, ordered by increasing standard deviation σ_u.

Like *shift*, the differences in *spread* can be caused by selective voting, personalities, preferences, or different interpretations of the rating scale. For example, users (a), (b), and (c) appear to have seldom used the rating values 1 and 5, effectively converting the 5-way rating scale into a 3-way rating scale of "bad-medium-good" represented by the rating values 2, 3, and 4. Such behaviour is one of the factors that decreased the variance of their DVPs. Actually, by ignoring the rating values of 1 and 5 and stretching the middle range from 2 to 4 to the full range, user (b) would appear to be similar to (d), and user (c) similar to (e).[1]

Simplified Mental scales

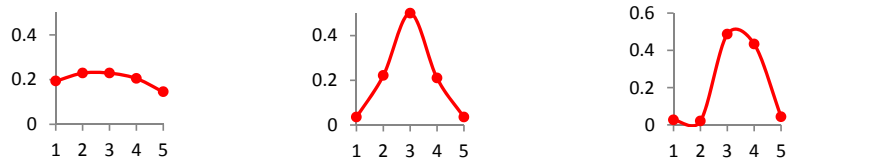
The MLS and MLM datasets use a 5-way input interface, where users could choose an integer between 1 to 5 to communicate their preferences on the movie. The five integers can be interpreted as "*terrible*, *no good*, *medium*, *good*, and *brilliant*" respectively. This is a very common rating interface. In fact, the lecturer evaluation at Victoria University uses this format. However, analysis of the data shows that this appears to be overly complex for some users, causing them to consciously or subconsciously form a simplified mental scale.

As opposed to 5-way voters who make full use of the provided rating scale, a 2-way voter subconsciously picks two out of the five rating values, and assigns them the mental interpretation of "good" versus "bad",

[1] In general, the DVPs do not suffer from a lack of data support, because the MLS and MLM datasets guarantee at least 20 ratings for each user. In other words, the obscurely sharp or flat shapes of the DVP presented here are all caused by the voting habits of users.

effectively converting the rating scale into a binary scale to simplify the process. Similarly, a 3-way voter picks three out of the five rating values, and interprets them as "good", "medium", and "bad". 4-way voters are not distinguished here because 1) it is intuitively less natural; 2) since there is only one omitted rating value, it is hard to distinguish if it is caused by simplified mental scale or simply as a dip of the 5-way voting pattern. To classify it as a separate mental scale runs the risk of overkill.

Figure 5.7 shows three real user-based DVPs with 5-way, 3-way, and 2-way mental scales respectively. The simplified mental scale phenomenon is further addressed in the discussion of *peaks*.



(a) 5-way mental scale $|R_u| = 83$ (b) 3-way mental scale $|R_u| = 86$ (c) 2-way mental scale $|R_u| = 113$

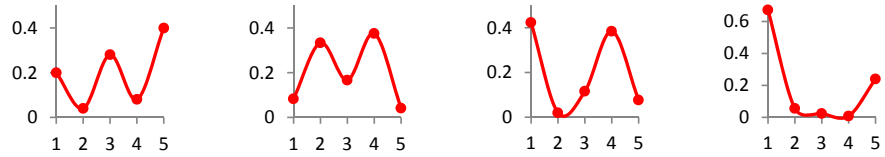
Figure 5.7: DVPs with different levels of rating scale simplification. The number of ratings of each user or $|R_u|$ is also shown in the captions, testifying that the simplified mental scale is not caused by a lack of data support.

Peaks

One advantage of the discrete vector representation that DVP uses over the "mean and variance" representation is it is able to clearly display the different mental scales. Another advantage is it is able to lightly and smoothly model multiple *peaks* in a distribution.

Figure 5.8 plots four real-user-based DVPs with multiple peaks. User (a) has a triple-peaked DVP that peaks at rating values 1, 3, and 5, and can be viewed as having a 3-way simplified rating scale; users (b), (c), and (d) have double-peaked DVPs that peaks at $(2, 4)$, $(1, 4)$, and $(1, 5)$ respectively, and can be viewed as having 2-way simplified rating scales.[2]

[2]In a simplified rating scale, the bucket of an omitted rating value does not have to be completely empty for it to qualify as "omitted". A 2-way or 3-way voter, if contributing a



(a) triple-peaked (b) double-peaked (c) double-peaked (d) double-peaked

Figure 5.8: Multi-peaked distributional voting profiles of real users.

Although a multi-peaked DVP is more likely to correspond to a simplified mental scale, the two concepts do not imply each other. Figure 5.9 uses 15 real user-based DVPs to demonstrate that different mental rating scales can happen with all shapes of peaks. In this table, the rows correspond to different mental rating scales, and the columns correspond to different peak-shapes.

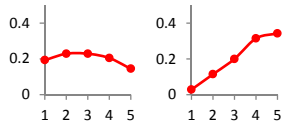
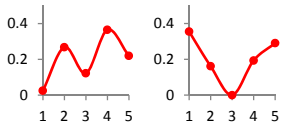
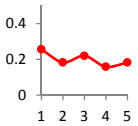
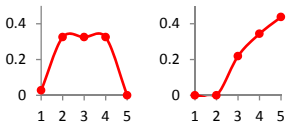
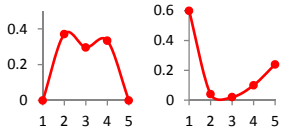
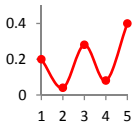
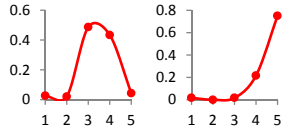
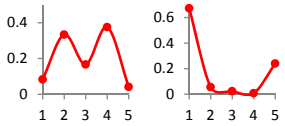
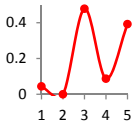
	Single-peaked	Double-peaked	Triple-peaked
5-way Mental Scales			
3-way Mental Scales			
2-way Mental Scales			

Figure 5.9: Different mental rating scales can happen with all peak-shapes.

large number of votes, can still realise the existence of the omitted rating values from time to time, and occasionally assign a few sporadic ratings to them. Whether or not a rating value is omitted should be decided by its relative bucket size instead of the absolute number of ratings with this value.

5.4 Why Distributional?

The advantages of distributional representation lie in its capability of representing, preserving, carrying, and communicating more information in all stages of the recommendation process. This section elaborates on the necessity, benefits, and power of distributional rating from four aspects. 5.4.1 presents the necessity of using the distributional voting profile to represent user's voting habits; 5.4.2 and 5.4.3 explain the benefits of using distributional rating to represent rating data gathered from users (the inputs) and the recommendations provided to the users (the outputs); 5.4.4 clarifies the benefit of using a distributional similarity representation in the recommendation process.

5.4.1 More Complete Portrayal of Users' Voting Habits

Distributional voting profile or DVP provides a more complete portrayal of the user's voting habit compared to the scalar representation using the average rating $\overline{r_u}$. The DVP is able to model various aspects of the voting patterns, including *shift*, *spread*, *peak*, and *user's mental scale*, whereas $\overline{r_u}$ can only model *shift*.

Figure 5.10 shows the DVP of six real users from the MLM dataset. All six users have an equal average rating of $\overline{r_u} = 3.0$, based on which they would be considered as having the same voting habit. However, their DVPs suggest that they are indeed very different: user (a) appears to be a strongly opinionated extremist who either likes or hates a movie with limited neutral ground; user (b) is also opinionated but less extreme, as indicated by the elevated rating count in the middle range; user (c) is not as balanced (with an asymmetrical DVP) and somewhat negative. He seems to treat 1 and 4 as "very bad" and "all right" and considers most movies to fall into these two categories; user (d) clearly utilises the 5-way rating scale very uniformly, assigning a similar number of ratings to each rating value; user (e) is a mediocre-to-negative user, who considers most of the movies he viewed "not very good but not extremely bad"; user (f) is the opposite of user (a). He considers most of the movies mediocre, and does not seem

to utilise the rating values of 1 and 5 at all, effectively treating the 5-way rating scale as a 3-way "bad-medium-good" scale. By only using a single number $\overline{r_u}$ to capture users' voting habits, all these differences would be lost, and all the users treated in exactly the same way.

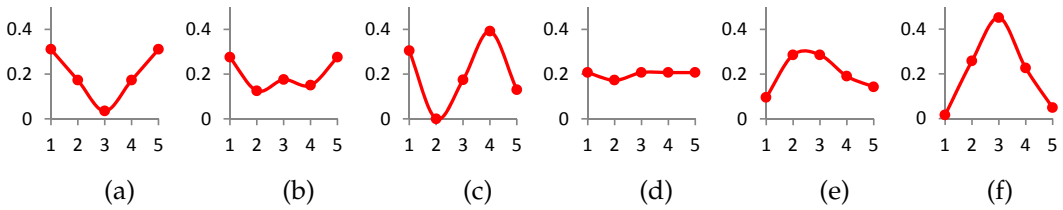


Figure 5.10: The *distributional voting profiles* of six real users selected form the MLM dataset. All six users have the same *average rating* of $\overline{r_u}$=3.0 despite of their very different voting patterns.

An upgrade would be to use the mean-variance pair ($\overline{r_u}$, σ_u^2) to represent the user's voting habit. This method is also inferior to DVP. Figure 5.11 illustrates why using four real users selected from the MLM dataset. This time, both the rating average *and* the rating SD are near equal between user pairs (a) and (b) and user pairs (c) and (d). A mean-variance representation would consider them as being the same within the pairs, but the DVPs still show their differences very clearly. Between the first pair, user (a) is positive but not extreme, and uses the 5-way rating scale very well; user (b) appears to be more opinionated, more likely to have strong feelings towards movies, and appears to be having a simplified mental scale. Between the second pair, user (c) is clearly different from user (d) who is more conservative and neutral, and is actually more similar to user (a).

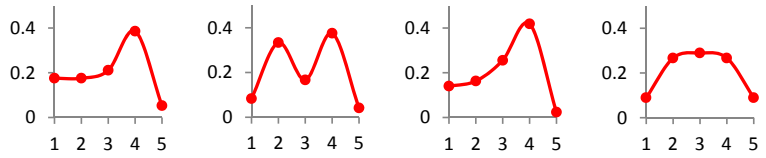


Figure 5.11: The *average rating distributions* of four real users in the MLM dataset. Both (a), (b) and (c), (d) have very similar rating average and rating SD, but the shape of their *average rating distributions* still differ greatly.

5.4.2 Noise-Tolerant Representations of User Inputs

The previous section demonstrates that the probability distribution $\mathcal{P}(r|\, u)$ provides a more complete portrayal of user's voting habits compared to a single number $\overline{r_u}$. Similarly, the probability distribution of $\mathcal{P}(r|\, u, i)$ is a more expressive and sensible representation of user ratings compared to a single integer number $r_{u,i}$. In terms of rating *inputs*, it is particularly helpful at modelling the noise in user data, as explained in the rest of this section; in terms of rating *outputs*, it is particularly helpful at communicating the uncertainty of the prediction, as elaborated in section 5.4.3.

Users' ratings on items are noisy reflections of their true underlying preferences of these items [112, 48, 3, 155], thus are better handled probabilistically. User studies have shown that users tend to rate the same item differently when asked to rate it in different sessions [48]. This is potentially caused by accidental wrong inputs, temporal shift of interests, the user's physical state such as his stomach content or the lack thereof, the user's mental state such as his general mood or alertness, and environmental factors such as the weather, noise, time of the day, or the quality of the movie he rated right before this one. New subfields of recommender systems have been established to study these aspects, including temporal diversity studies [72] and contextual diversity studies [3, 155].

Distributional rating is a simpler way to capture such diversities without the burden of explicitly handling the extra temporal or contextual dimensions. On one hand, it can be used to represent the level of confidence based on the condition the data is collected — if the user is known to be hungry or the time is known to be late, a distribution with a bigger variance can be imposed to indicate reduced reliability; if the weather is known to be gloomy, maybe an increased probability on the upper rating values can be imposed to counteract the negative effect caused by the weather; a bigger variance can also be imposed on *implicitly* gathered user feedbacks,[3] which are more prone to noise. Figure 5.12 presents four distributional ratings all correspond to a scalar input rating of $r_{u,i} = 4$ to show the estimated noise patterns in the three aforementioned circumstances as

[3]Implicit and explicit relevance feedback is described in section 2.1.1.

well as the standard case.

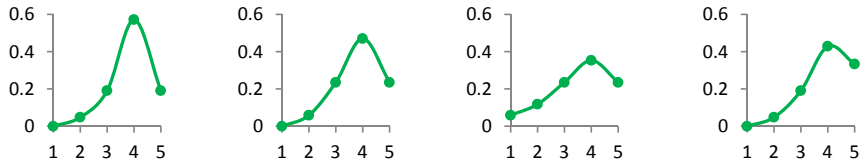


Figure 5.12: Four different constructions of distributional ratings for the scalar input rating $r_{u,i} = 4$, subject to different noise patterns, with (a) being the normal circumstance, (b) and (c) demonstrate an increased variance due to reduced reliability, and (d) shows an elevated probability at rating value $r = 5$ to reflect the fact that the underlying rating $r_{u,i} = 4$ may be subject to a negative mental shift caused by environmental factors such as gloomy weather.

On the other hand, if conditions permit, the same rating could be solicited from the user multiple times, preferably in different physical, mental, and environmental conditions through different (implicit or explicit) interfaces. The different solicitations of the same rating can all be aggregated in the distributional rating to form a more reliable reflection of the user's true preference on the item.

5.4.3 More Informative Form of Recommendation Feedback

Distributional ratings are a more informative form of recommendation to present back to the users. Firstly, it enables the communication of the level of confidence of the recommendation. Secondly, it allows the feedback format to be more dynamic, subtle, and user-friendly.

To elaborate on the first point, one of the differences between a rating provided by the user to the recommender system (i.e. an input rating) and a rating prediction the system provides to the user (i.e. an output rating) is the latter is a *guess*, thus has an innate level of uncertainty that should be communicated to the users. Distributional rating is able to facilitate this. For example, figure 5.13 shows three distributional ratings that correspond to three different "flavours" of the "5" prediction with diminishing degrees of confidence.

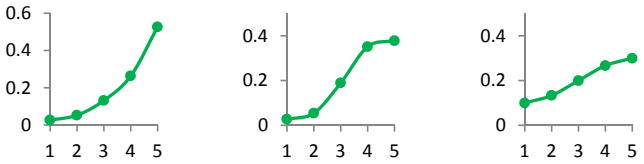


Figure 5.13: Different "distributional" ways of communicating a prediction rating of 5 with diminishing degrees of confidence.

Distributional ratings are capable of communicating much more than just the degree of confidence. They are also able to model fundamentally different or in a way "higher dimensional" feedbacks that are more dynamic, subtle, and friendly to the users. For example, figure 5.14 shows three distributional rating predictions all with the same expected value of $r_{u,i} = 3$. However, they convey totally different messages. From left to right, they can be interpreted as "*Sorry, I do not have enough information to make a recommendation for you this movie*", "*I think you will consider this movie mediocre*", and "*I think you will either love or hate this movie*" respectively.

Figure 5.15 presents another three examples with more subtle differences. All three distributions have the expected rating value of $r_{u,i} = 3.6$, so they are equivalent predictions in scalar terms. The messages they convey are similar in general, but different in a subtle way. From left to right, they can be interpreted as "*You may find it a fine movie, not likely to be extremely good though*", "*it could be really good for you, or at least it is unlikely to be too bad*", and "*you are very likely to find the movie not bad, but it is definitely not a mind-blower*". Since they all have the same expected rating value, these subtle differences will be lost in a scalar prediction.

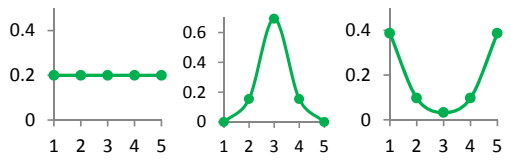


Figure 5.14: Three distributional prediction feedbacks that all have the same expected value of $r_{u,i} = 3$, but with very different interpretations.

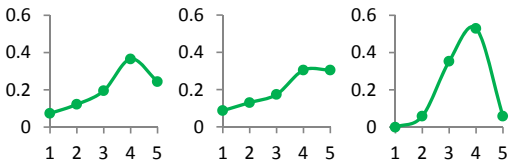


Figure 5.15: Three distributional prediction feedbacks that all have the same expected value of $r_{u,i} = 3.6$, but with different interpretations in a subtle way.

5.4.4 Better Information Carrier in the Recommendation Process

Standard collaborative filtering algorithms revolve around scalar number-based controllers. In nearest neighbour methods, a scalar number is used to model the similarity between a pair of users; in clustering methods, a scalar number is used to model the membership association between a user and a cluster; in latent variable methods, a scalar number is used to model the dependency between a user and a latent variable. We believe that "distributionalising" these scalar controllers can lead to more flexible and more expressive knowledge models during the recommendation process, and consequently improve the quality of the final recommendations.

This thesis investigates the use of such a "distributionalised controller" to represent user similarities in nearest neighbour-based collaborative filtering. The idea is presented in section 5.5.3, which allows the communication of complicated knowledge concepts such as "*Ann agrees with Bob on movies he hates, but not the ones he likes*" during the recommendation process.

5.5 Distributional Rating-based Recommendation

This section describes how the distributional rating (DR) and the distributional voting profile (DVP) can be used in the nearest neighbour-based collaborative filtering process to form a *Distributional Rating-based Nearest Neighbour* (DRNN) recommender. Section 5.5.1 firstly describes how to transform distributional ratings to and from scalar ratings, which is the native rating form provided by the rating datasets. The next three sections present the DRNN process, with the distributional rating-based normalisation, distributional similarity computation, and distributional rating prediction explained in sections 5.5.2, 5.5.3, and 5.5.4 respectively.

5.5.1 Scalar and Distributional Rating Transformation

Rating datasets such as MLS, MLM, and JST all come with scalar ratings. However, the DRNN recommendation process requires distributional ratings as its input and output format. So in order to use these available datasets to train the system, there has to be a mechanism for transforming the scalar rating inputs into distributional rating inputs; and in order to use these datasets to evaluate the system, there has to be a mechanism for transforming the distributional rating predictions back to scalar predictions. This section describes this transformation process.

Index Vector

The *index vector* can be considered as the intermediate step between scalar and distributional ratings. In an integer rating scale from 1 to 5, the preference of a user u on an item i is expressed using a single scalar number $r_{u,i} \in [1, 5]$. The corresponding *index vector* of this scalar rating is defined as a 1×5 vector of *ones* and *zeros*, symbolised as $\mathbf{I}$, and is obtained by assigning a *one* to the index that matches the actual rating, and *zero*s elsewhere. For example, a rating of $r_{u,i} = 1$ would have an index vector of $(\,1\;0\;0\;0\;0\,)$, a rating of $r_{u,i} = 2$ would have an index vector of $(\,0\;1\;0\;0\;0\,)$, and so forth.

An index vector assumes the same form as a distributional rating, yet contains the same information as a scalar rating. It can also be considered as an extreme form of distribution rating with absolute certainty, since it conforms with the conditions described in formula 5.4.

Scalar to Distributional Rating Transformation

Ideally, the scalar to distributional rating transformation should be based on noise patterns of user inputs backed by real data. As described in section 5.4.2, one way to obtain this pattern is to solicit the same rating on the same item from the same user multiple times under different environmental and temporal conditions, then aggregate them to form the noise-adjusted probabilistic pattern of this user's true preference on this item.

Since this is not feasible, DRNN imposes a manually selected probability distribution as the assumed noise pattern for all ratings. For example, a scalar rating of 3 can be mapped onto a distributional rating of $(\ .05\ .15\ .60\ .15\ .05\ )$, a scalar rating of 4 can be mapped onto a distributional rating of $(\ .04\ .06\ .12\ .65\ .13\ )$, and so on.

Formally, this transformation process is controlled by a 5×5 *base matrix* $\mathbf{B}$. Suppose $r_{u,i}$ is the scalar rating to be transformed, the symbol $\mathbf{I}$ represents its *index vector*, which has dimension 1×5; the symbol $\mathbf{R}$ represents the transformation result, namely the corresponding distributional rating of $r_{u,i}$, and also has dimension 1×5. The transformation from $r_{u,i}$ to $\mathbf{R}$ can be obtained by firstly transforming it into its index vector $\mathbf{I}$, then following equation 5.7:

$$\mathbf{R}^{1\times 5} = \mathbf{I}^{1\times 5} \times \mathbf{B}^{5\times 5} \tag{5.7}$$

Figure 5.16 shows three example base matrices. Each entry in the base matrix is a real number between 0 and 1. The ith row of the base matrix corresponds to the distributional rating that the scalar rating i will be transformed into. For example, with base matrix $\mathbf{B_2}$, a scalar rating of 4 will be converted to the fourth row of $\mathbf{B_2}$, i.e. $(\ .04\ .06\ .12\ .65\ .13\ )$, and so forth.

$$\mathbf{B_1} = \begin{bmatrix} .50 & .25 & .15 & .07 & .03 \\ .20 & .45 & .20 & .10 & .05 \\ .10 & .20 & .40 & .20 & .10 \\ .05 & .10 & .20 & .45 & .20 \\ .03 & .07 & .15 & .25 & .50 \end{bmatrix} \quad \mathbf{B_2} = \begin{bmatrix} .80 & .10 & .05 & .03 & .02 \\ .10 & .70 & .10 & .06 & .04 \\ .05 & .15 & .60 & .15 & .05 \\ .04 & .06 & .12 & .65 & .13 \\ .03 & .05 & .07 & .15 & .70 \end{bmatrix} \quad \mathbf{B_3} = \begin{bmatrix} .96 & .01 & .01 & .01 & .01 \\ .01 & .96 & .01 & .01 & .01 \\ .01 & .01 & .96 & .01 & .01 \\ .01 & .01 & .01 & .96 & .01 \\ .01 & .01 & .01 & .01 & .96 \end{bmatrix}$$

Figure 5.16: Three Base Matrices. $\mathbf{B_1}$ is flat and "symmetrical"; $\mathbf{B_2}$ is sharp and oblique; $\mathbf{B_3}$ is crisp.

Among the three base matrix examples, $\mathbf{B_1}$ assumes more noise in the input ratings than $\mathbf{B_2}$, and provides a "flatter" conversion. It is also "symmetrical", meaning that the distributional ratings of 1 and 5 mirror each other, and so do the ones for 2 and 4. The second base matrix $\mathbf{B_2}$ assumes less noise in the input ratings, thus convert the scalar ratings into sharper distributions, representing an increased confidence in the correctness of the scalar ratings. It is also oblique (i.e. non-symmetrical), stating that it

assumes less noise if the input rating is a 4 compared to if it is a 2. The third base matrix $\mathbf{B_3}$ does not assume the "gradual dropping" effect of the previous two matrices, stating that when the rating is 1, it is no more likely for it to actually be a 5 than it is actually a 2.

The base matrix can be considered as a set of tunable parameters of the DRNN recommender system. The choice of the base matrix carries a certain inductive bias. One approach to obtaining a good base matrix would be to learn it from the data, by searching for the base matrix that maximises recommendation accuracy on a validation dataset. An exploratory experiment using a hill climbing search based on simulated annealing was conducted, but had several problems. Firstly, it was slow, since the entire validation dataset had to be evaluated in order to make one step of adjustment to one of the parameters, and there are $5 \times 5 = 25$ of them; secondly, the amount of extra performance improvement achieved by tuning this set of parameters is insignificant compared to the overall performance improvement achieved by DRNN over the scalar-based NNCF method.[4] Therefore, the conclusion from the experiment was that to do significantly better than a reasonable guess at the base matrix (such as $\mathbf{B_2}$ in figure 5.16), one would either need a much more sophisticated learning strategy, or invest in the resources to gather multiple solicitations of the same rating from real users to properly establish the noise pattern of each rating. The results in the rest of the chapter therefore use the matrix $\mathbf{B_2}$.

Distributional to Scalar Rating Transformation

Another benefit of the base matrix is it allows consistent conversion from the distributional rating back to the scalar rating, which is necessary if a scalar rating prediction is required for evaluation purposes. Specifically, to convert a distributional rating $\mathbf{R}$ back to a scalar, it is firstly converted back to the index vector to rid the "contamination" of the base matrix. This is done by multiplying the distributional rating by the inverse matrix of the

[4]The experimental results of this exploration is shown in section 5.7.

base matrix, or $\mathbf{B^{-1}}$, as shown in equation 5.8:

$$\mathbf{I} = \mathbf{R} \times \mathbf{B^{-1}} \tag{5.8}$$

The scalar rating r can be subsequently calculated as the expected value of the index vector, as shown in formula 5.9:

$$r = \sum_{i=1}^{k} i \cdot \mathbf{I_{1,i}} \tag{5.9}$$

This mechanism guarantees that a scalar rating, after being transformed to a distributional rating, can be transformed back to its original scalar value without any "contamination". Note that for most of the distributional rating predictions, the *index vector* calculated in this way will not be vectors of ones and zeros. Only the distributional ratings that match the base matrix rows will have an index vector of zeros and ones.

5.5.2 Distributional Rating Normalisation

A user's *voting habit* is the user's internal mapping of the rating scale to their true preference of the items, and can vary greatly from user to user. It was first noted in Resnick et al. [120], which proposed to *normalise* the input ratings against the user's average rating $\overline{r_u}$ to eliminate the *shift* bias caused by the different rating habits before it can be used to aid the prediction for others. This is presented in formula 4.6 in chapter 4, and is reformatted below in formula 5.10.

Scalar Rating Normalisation — a Reminder

Suppose the task is to predict target user $u_\star$'s preference on item i; u represents other users in $u_\star$'s neighbourhood that already rated the item; $sim(u, u_\star)$ represents the similarity between u and $u_\star$. The *normalised prediction* or $r'_{u_\star,i}$ is calculated by summing over the neighbours' ratings on i weighted by their similarities:

$$r'_{u_\star,i} = \frac{\sum_u sim\,(u, u_\star) \cdot r'_{u,i}}{\sum_u sim\,(u, u_\star)} \tag{5.10a}$$

Where $r'_{u,i}$ is the normalised rating of $r_{u,i}$ to eliminate the voting habits:

$$r'_{u,i} = r_{u,i} - \overline{r_u} \tag{5.10b}$$

The final prediction $r_{u\star,i}$ is calculated by *de-normalising* the normalised prediction $r'_{u\star,i}$ obtained in formula 5.10a against the average rating of the target user in order to translate the prediction back into the "original language" of the user, namely:

$$r_{u\star,i} = r'_{u\star,i} + \overline{r_{u\star}} \tag{5.10c}$$

The problem with this method is, user's average rating $\overline{r_u}$ is only a partial representation of the user's voting habit. Specifically, it only captures the *shift* aspect, but fails to grasp other equally important aspects such as *spread*, *peaks*, or *actual mental scale*.

Distributional Rating Normalisation

Distributional rating normalisation is a step in the DRNN process. It is based on the same principle of eliminating users' voting habits, but instead of normalising scalar ratings against the user's rating average, it normalises distributional ratings against the user's DVP.

Suppose $\mathcal{P}(\underset{\sim}{\mathrm{r}}\,|u,i)$ is a distributional rating and $\mathcal{P}(\underset{\sim}{\mathrm{r}}\,'|u,i)$ represents its normalised version, $\mathcal{P}(\underset{\sim}{\mathrm{r}}\,|u)$ is user u's distributional voting profile, and $\mathcal{P}(\underset{\sim}{\mathrm{r}})$ is the *global voting profile*, which is computed in the same way as $\mathcal{P}(\underset{\sim}{\mathrm{r}}\,|u)$ but across all users, as shown in formula 5.1c. The distributional rating normalisation process is defined in formula 5.11a, and the de-normalisation process is defined in formula 5.11b.

$$\mathcal{P}(\underset{\sim}{\mathrm{r}}\,' \mid u,i) = \mathcal{F}\left(\frac{\mathcal{P}(\underset{\sim}{\mathrm{r}})}{\mathcal{P}(\underset{\sim}{\mathrm{r}}\,\mid u)} \cdot \mathcal{P}(\underset{\sim}{\mathrm{r}}\,\mid u,i) \right) \tag{5.11a}$$

$$\mathcal{P}(\underset{\sim}{\mathrm{r}}\,\mid u,i) = \mathcal{F}\left(\frac{\mathcal{P}(\underset{\sim}{\mathrm{r}}\,\mid u)}{\mathcal{P}(\underset{\sim}{\mathrm{r}})} \cdot \mathcal{P}(\underset{\sim}{\mathrm{r}}\,' \mid u,i) \right) \tag{5.11b}$$

where the multiplication and division of the distributional vectors are "element-wise operations", namely:

$$\left(x_1 \ x_2 \ x_3 \ x_4 \ x_5 \right) \cdot \left(y_1 \ y_2 \ y_3 \ y_4 \ y_5 \right) = \left(x_1 y_1 \ x_2 y_2 \ x_3 y_3 \ x_4 y_4 \ x_5 y_5 \right) \tag{5.11c}$$

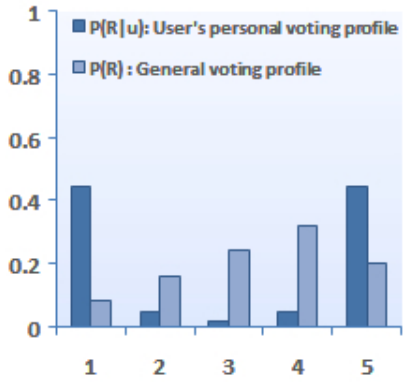
$$\frac{\left(x_1 \ x_2 \ x_3 \ x_4 \ x_5 \right)}{\left(y_1 \ y_2 \ y_3 \ y_4 \ y_5 \right)} = \left(\frac{x_1}{y_1} \ \frac{x_2}{y_2} \ \frac{x_3}{y_3} \ \frac{x_4}{y_4} \ \frac{x_5}{y_5} \right) \tag{5.11d}$$

and the function $\mathcal{F}(\underset{\sim}{\mathrm{x}})$ formats vector $\underset{\sim}{\mathrm{x}}$ back to meet the distributional rating formatting constraints specified in formula 5.4, namely

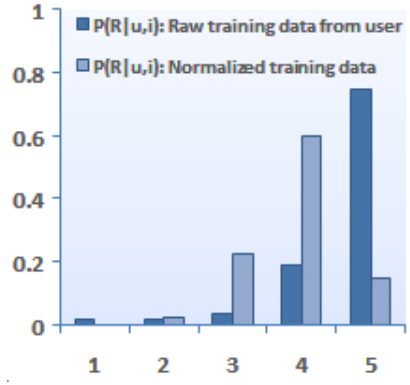
$$\mathcal{F}(\underset{\sim}{\mathrm{x}}) = \mathcal{F}(\left(x_1 \ x_2 \ x_3 \ x_4 \ x_5 \right)) = \left(\frac{x_1}{\sum \underset{\sim}{\mathrm{x}}} \ \frac{x_2}{\sum \underset{\sim}{\mathrm{x}}} \ \frac{x_3}{\sum \underset{\sim}{\mathrm{x}}} \ \frac{x_4}{\sum \underset{\sim}{\mathrm{x}}} \ \frac{x_5}{\sum \underset{\sim}{\mathrm{x}}} \right) \tag{5.11e}$$

Figures 5.17 and 5.18 demonstrate the effect of this form of distributional rating normalisation on two users Ann and Bob. Ann is an extremist 2-way voter who tends to rate items either a 1 or a 5, whereas Bob is a positive voter who tends to rate items on the high end. In each figure, subfigure (a) shows the global DVP and the DVP of the respective user; (b) shows a distributional rating input and its normalised distributional value against the user's DVP based on equation 5.11a; (c) and (d) show two distributional rating predictions that peak at 4 and 5 respectively, with their corresponding de-normalised distributional value calculated based on equation 5.11b.

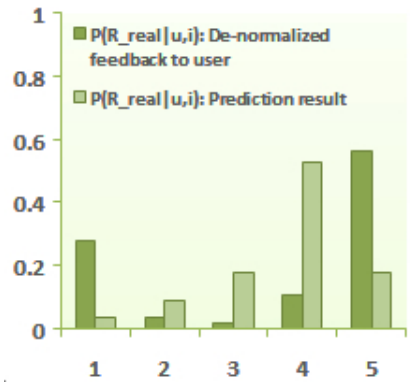
The examples show that this normalisation process is able to eliminate the effect of the users' strong voting habits from the individual distributional ratings, and the de-normalisation process is able to add the voting habits back to the ratings and translate the predictions back to the users' respective vocabularies. Figure 5.18(b) also shows a particularly interesting feature. It normalises a strong 5 by a positive user (mostly a 4 and 5 voter) into a rating that peaks at 3 and 5, stating that the user has a high chance of really liking the movie (i.e. a 5 being true), but he also has a solid chance of considering this movie mediocre, because he has given too many items a high rating, so that his high rating is no longer trustworthy.



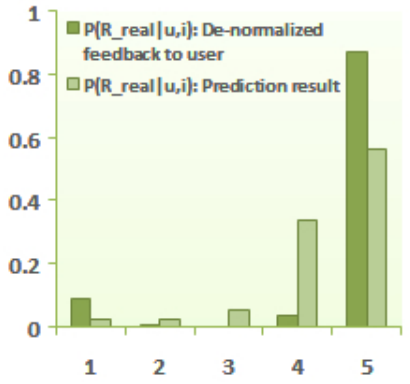
(a) User Ann's DVP (distributional voting profile), compared with the global DVP.



(b) A distributional rating input, and its normalisation result against Ann's DVP.

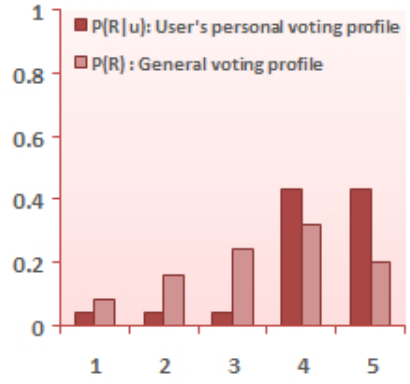


(c) A distributional rating prediction to Ann that peaks at 4, and its de-normalised form.

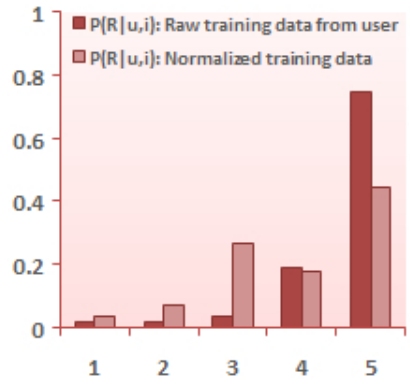


(d) Another distributional rating prediction to Ann that peaks at 5, and its de-normalised form.

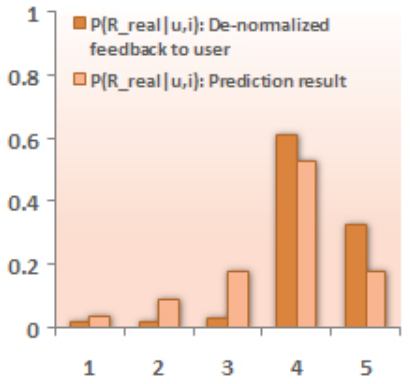
Figure 5.17: A distributional rating normalisation example on user Ann, who is an extremist 2-way voter.



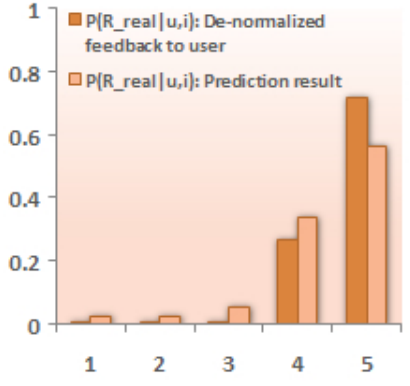
(a) User Bob's DVP (distributional voting profile), compared with the global DVP.



(b) A distributional rating input, and its normalisation result against Bob's DVP.



(c) A distributional rating prediction to Bob that peaks at 4, and its de-normalised form.



(d) Another distributional rating prediction to Bob that peaks at 5, and its de-normalised form.

Figure 5.18: A distributional rating normalisation example on user Bob, who is a positive voter.

5.5.3 Distributional Similarity Computation

In nearest neighbour collaborative filtering (NNCF), the "learning" resides within the calculation of user similarities, which are represented as real scalar numbers in the range of $s \in [-1, 1]$, where 1, 0, and -1 correspond to total positive correlation, a lack of correlation, and total negative correlation respectively.

DRNN proposes to use *distributional similarity vectors* to model such learnt knowledge. Specifically, as opposed to using one real number in the range of $s \in [-1, 1]$ to capture the correlation between the preference patterns of two users, the idea is to use a vector of similarities where each entry corresponds to a rating value on the rating scale. In a five-way rating scale, it would use a vector of five, or $\underset{\sim}{\mathrm{s}}\,(u_1, u_2) = (\, s_1 \;\; s_2 \;\; s_3 \;\; s_4 \;\; s_5 \,)$ with $s_x \in [-1, 1]$. Formally, it can be expressed as formula 5.12:

$$\underset{\sim}{\mathrm{s}}\,(u_1, u_2) = \big(\, s_1(u_1,u_2) \;\; s_2(u_1,u_2) \;\; s_3(u_1,u_2) \;\; s_4(u_1,u_2) \;\; s_5(u_1,u_2) \,\big) \tag{5.12}$$

where $s_x(u_1, u_2)$ is the similarity of users u_1 and u_2 on the rating value x. Suppose $\big\{\, \mathcal{P}(\underset{\sim}{\mathrm{r}}\,|u, i) \,\big\}_{i \in I}$ is the vector of user u's distributional ratings on all items $i \in I$ ordered by item indices. This vector can be viewed as a $5 \times |I|$ matrix where each column corresponds to the transpose of user u's distributional rating on item i. The x^{th} row of this matrix would be a vector of size $|I|$ consisting of real numbers in the range of 0 to 1, each of which represents the probability of u giving item i a rating of x, namely $\big\{\, \mathcal{P}(r{=}x|u, i) \,\big\}_{i \in I}$. Then $s_x(u_1, u_2)$ — the similarity of u_1 and u_2 on rating value x — can be calculated as the Pearson's correlation of the two vectors $\big\{\, \mathcal{P}(r{=}x|u_1, i) \,\big\}_i$ and $\big\{\, \mathcal{P}(r{=}x|u_2, i) \,\big\}_i$ namely:

$$s_x(u_1, u_2) = \tag{5.13}$$
$$\textit{Pearson's correlation}\Big(\big\{\, \mathcal{P}(r{=}x|u_1, i) \,\big\}_{i \in I_{u_1} \cap I u_2}, \big\{\, \mathcal{P}(r{=}x|u_2, i) \,\big\}_{i \in I_{u_1} \cap I u_2}\Big)$$

The distributional similarity vector allows the user similarity of different rating values to be modelled and learnt separately, thus enables the discovery, representation, and communication of complicated knowledge concepts such as "*Ann agrees with Bob on movies he hates, but not the ones*

he likes". Figure 5.19 presents five real user-based distributional similarity vectors to demonstrate the expressiveness and diversity of the structure. Figure (a) shows two users who are similar over the entire spectrum; (b) shows two users who tend to agree on items they like but disagree on items they don't like; (c) shows two users who also agree on items they like, but have no reciprocal predictive power on items they may not like; (d) shows two users who agree on the "middle range", but tend to disagree on the two extreme ends; (e) shows two users who are in a way the opposite of (d) — they agree on the extreme ends, but have limited receiprocal predictive power in the middle range.

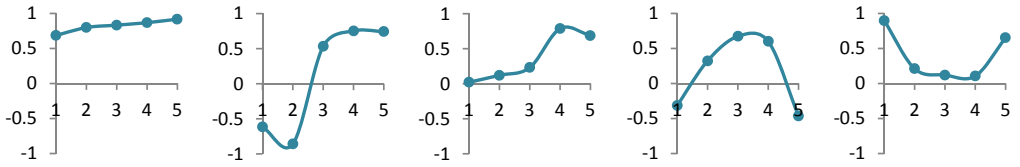


Figure 5.19: Distributional similarity vector examples based on real users in the MLS dataset.

5.5.4 Distributional Rating Prediction

Upon obtaining the distributional similarity vectors, the distributional rating prediction of user $u_\star$'s preference on item $i_\star$ can be calculated based on the *normalised* distributional ratings of the neighbouring users, as shown in equation 5.14:

$$\mathcal{P}(\underset{\sim}{\mathrm{r}}' \mid u_\star, i_\star) = \frac{\sum_u \underset{\sim}{\mathrm{s}}\,(u, u_\star) \cdot \mathcal{P}(\underset{\sim}{\mathrm{r}}' \mid u, i_\star)}{\sum_u \underset{\sim}{\mathrm{s}}\,(u, u_\star)} \tag{5.14}$$

where u represents the neighbouring users of $u_\star$; and the multiplication and division in the formula are element-wise operations as shown in formulae 5.11c and 5.11d.

Equation 5.14 only produces the *normalised* distributional prediction $\mathcal{P}(\underset{\sim}{\mathrm{r}}' \mid u_\star, i_\star)$, which needs to be de-normalised to $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u_\star, i_\star)$ based on

formula 5.11b, so that user $u_\star$'s voting habits can be added back into the prediction to ensure that it communicates to the user in his own language.

The de-normalised $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u_\star, i_\star)$ forms the final distributional prediction that represents user $u_\star$'s projected preference on item $i_\star$. To transform it to a scalar rating prediction, formula 5.8 needs to be applied first to get rid of the bias of the base matrix, and the final scalar prediction can be calculated as the weighted sum of the distribution according to formula 5.9, which is repeated below:

$$r_{u,i} = \sum\nolimits_{r \in \underset{\sim}{\mathrm{r}}} r \times p(r|u,i) \tag{5.15}$$

5.6 Related Work

This section outlines related work in the order of increasing relevancy. Section 5.6.1 presents studies that use a probability space similar to that described in section 5.1; section 5.6.2 presents studies that assume a Gaussian distribution; section 5.6.3 presents studies that also use a discrete vector-based representation.

5.6.1 Work that uses a similar probability space

Breese et al. [23] is the earliest publication that views the task of making a scalar rating prediction as calculating the expected value of its probability distribution, namely $r_{u,i}{=}\sum_{k=r_{\min}}^{r_{\max}} k \times \mathcal{P}(r{=}k \mid u, i, \text{data})$.

From that point on, various forms of probability-based collaborative filtering have been developed. Nowadays, most latent-variable based methods such as aspect model, probabilistic latent semantic analysis, and probabilistic clustering use a probability space similar to the construction described in section 5.1. Examples include methods described in section 2.2.4, and [78] which is elaborated in the following section.

5.6.2 Work that assumes a Gaussian distribution

This section describes studies that assume a Gaussian distribution for various probability distributions such as the rating distribution of users $\mathcal{P}(r \mid u)$ [49], the rating distribution of items $\mathcal{P}(r \mid i)$ [70], and the rating distribution of user clusters $\mathcal{P}(r \mid i, z)$ [51, 78].

In collaborative filtering, the most common place to see a Gaussian distribution is in a probabilistic model-based approach, such as aspect model, PLSA, or probabilistic clustering. Examples include the *probabilistic latent semantic analysis* [51] and *latent variable-based clustering* [78], both of which assume a Gaussian distribution over $\mathcal{P}(r \mid i, z)$, where z represents a latent hidden variable or a user cluster, and can be interpreted as the "group consciousness" or "group preference". This class of methods is not very similar to DRNN. It is only mentioned here to provide perspective and outline the scope of Gaussian distribution in collaborative filtering.

The *top-N adjusted filtering* [70] noticed the monotonically increasing relationship between the rating variance and the prediction error on items, and proposed to discount the items with high variance when generating the ranked recommendation list from rating predictions. This can be interpreted as imposing a Gaussian distribution over $\mathcal{P}(r \mid i)$, and uses it to improve the list recommendation accuracy instead of the rating prediction accuracy. This method is similar to DRNN in that they both realised the importance of identifying (user or item) rating patterns beyond the simple mean to the recommendation quality of the system. Other than that, they are still quite different.

The *z-score rating normalisation* [49] is the Gaussian-based method that is more similar to DRNN, or specifically, the DVP part of DRNN. It imposes a Gaussian distribution $\mathcal{N}(\overline{r_u}, \sigma_u)$ over $\mathcal{P}(r \mid u)$ to represent the user's voting habits, and then uses the *Z-score* defined as $z_i = \frac{r_{u,i} - \overline{r_u}}{\sigma_u}$ to compute the offset for rating normalization. It is similar to DRNN in that they both use more than the simple mean in rating normalisation to eliminate the different voting patterns of users. However, as demonstrated in section 5.4.1, although a Gaussian representation is able to capture both *spread* and *shift*, it fails to capture more complex aspects such as *mental scales* and

peaks, thus is not as powerful a representation as the DVP. Another fundamental difference is that DVP normalises distributional ratings, whereas the z-score only normalises scalar ratings.

5.6.3 Work that uses a discrete vector representation

This section describes studies that use the same discrete vector-based probability distribution representation as in DRNN to model various concepts, including the user prior $p(r|\ u)$ [58], the item prior $p(r|i)$ [57] and the normalised rating differences $p(r_{u,i} - \overline{r_u}|u,i)$ [48].

The *halfway accumulative distribution* (HAD) [58] is the first to use a discrete vector format to represent users' voting habits. Specifically, it proposes a new *scalar* rating normalisation heuristic that converts the scalar rating $r_{u,i}$ into $r'_{u,i} = \sum_{r \leq r_{u,i}} p(r|\ u) - p(r_{u,i}|u)/2$, where the prior probability $p(r|u)$ is represented as a discrete vector like the DVP. This method uses the rating to be normalised (i.e. $r_{u,i}$) as an index to indicate which part of the prior vector (i.e. DVP) to count up, instead of using its value directly. It is similar to DVP in terms of syntax. However, the power of DVP is only touched superficially, and its expressiveness and its relation to the prediction error are not investigated. It also only normalises scalar ratings in a scalar-based recommendation process, whereas DVP is used to normalise distributional ratings in a probability distribution-based recommendation environment.

The *easy recommendation based on probability model* (ERPM) [57] is a model-based collaborative filtering method that uses the underlying idea of distributional rating without formulating it as such. Essentially, like DRNN, it calculates the final rating prediction by "summing over" the probability distribution, namely $r_{u,i} = \sum_{r \in \underset{\sim}{\mathrm{r}}} r \times p(r|u,i)$. However, the value of each $p(r=x|u,i)$ is calculated directly from the priors $p(r=x|u)$ and $p(r=x|i)$ without an integrated representation for $p(r{=}x|u,i)$, and the values of the priors are "counted" from the dataset as in DVP. Like the HAD algorithm described in the previous paragraph, ERPM only notes the value of using a full probability distribution representation on the prior ratings $p(r|u)$ and $p(r|i)$ but not $p(r|u,i)$. Their improvement over HAD is that they use dis-

tributional prior ratings to make predictions, whereas the former only uses them in rating normalisation.

The *belief distribution algorithm* [87] proposes a form of distributional rating most similar to this thesis. Specifically, it uses a discrete vector to represent the probability distribution over the *normalised rating difference*, namely $p(r_{u,i} - \overline{r_u}|u, i)$. In an integer rating range of 1 to 5, this would be a discrete vector of size 9, corresponding to the rating difference indices of $(\ -4\ -3\ -2\ -1\ 0\ 1\ 2\ 3\ 4\ )$. This *difference distribution* is only used in the final prediction stage. The normalisation and the similarity calculation are all performed using scalar ratings only. This is very different from HAD and ERPM, and is also different from DRNN. Their rating transformation also uses a manually defined mapping similar to that of the base matrix. However, the mechanism is implemented in an arbitrary manner without the mathematical cohesion of the base matrix, and as a result they do not use the inverse of the base matrix to get rid of the bias introduced by the base matrix before producing the final scalar prediction.

5.7 Experimental Analysis

This section evaluates the DRNN recommendation process. Section 5.7.1 outlines the general performance of DRNN by comparing it with scalar-based nearest neighbour and three other related algorithms. The next three sections examine the different aspects of DRNN: section 5.7.2 evaluates the power of distributional rating predictions, section 5.7.3 evaluates the scalar to distributional rating transformation, and section 5.7.4 evaluates the concept of distributional similarity. The last two sections show that DRNN's performance copes better with users with complicated voting patterns, such as when the user has multi-peaked DVP (section 5.7.5), and when the user's DVP reflects complex mental scale (section 5.7.6).

5.7.1 The general performance of DRNN

This section presents the general performance of DRNN by comparing it with four baseline algorithms with increasing relevancy. The first baseline is the standard scalar-based nearest neighbour method (NNCF). The second baseline is the *z-score* method (NNCF$_z$) described in section 5.6.2, which uses z-score instead of deviation from the mean as the normalisation function. This method is scalar-based, but it effectively imposes a Gaussian distribution to model users' voting patterns. The third baseline is the *easy recommendation based on probability model* method (ERPM) described in section 5.6.3. This method produces distributional predictions, but only uses distributional representations minimally during the recommendation process.[5] The fourth baseline is the *belief distributional algorithm* (BDA) described in section 5.6.3. It also produces distributional predictions, and is most similar to DRNN, but models the probability distribution $p(r_{u,i} - \overline{r_u}|u,i)$ instead of $p(r_{u,i}|u,i)$, does not use distributional similarity, and does not perform the inverse base matrix transformation.

Experiments are performed on MLS, MLM, and JST datasets. Since the JST dataset features continuous floating point ratings instead of discrete integer ratings, in the scalar to distributional rating transformation, the index vectors are generated by finding the simplest vector whose expected value equals the floating point rating. For example, rating 3.8 maps onto the index vector $(\,0\;0\;.2\;.8\;0\,)$, and rating 4.3 maps onto $(\,0\;0\;0\;.7\;.3\,)$.[6] For the MLS and MLM datasets, DRNN is implemented in the user-user orientation, and for JST, it is implemented in the item-item orientation. This ensures there is a reasonable number of ratings for each user or item so their distributional voting profile and the distributional similarities can be calculated properly.

Compared with chapter 4, the evaluation of this chapter uses two new evaluation metrics: "F1 (misses)", and "NDCG (misses)". These two metrics are the same as the standard F1 and NDCG measures, but in the case a

[5]The ERPM method here is implemented with the recommended parameter values of $\alpha = 1.2$ and $\beta = 10$.

[6]Note that these are only index vectors. To obtain the distributional rating, they need to be multiplied with the base matrix as described in section 5.5.1.

recommended item does not appear in the test dataset so that its true user preference cannot be judged, instead of ignoring this recommendation as in the standard F1 and NDCG, it is regarded as a miss, and is counted (negatively) towards the final F1 and NDCG scores. This was explained in more detail in section 3.3.2.1 and summarised in table 3.5. This is an overly stringent measure, since at least some of these misses, if their user ratings were available, would probably be quite positive. However, since this is true for all algorithms being evaluated, these metrics still provide a comparison of the relative recommendation accuracy of different algorithms and settings.

Table 5.2 presents the experimental results using the skip-every-10^{th} partitioning protocol with 10-fold cross validation on all three datasets. The bold numbers indicate the best performers. A threshold cut-off of 3 is used for prediction to recommendation conversion with precision, recall, and F1; $N = 30$ is used for NDCG. The settings of the experiments are generally similar to that of section 4.5.1.

Table 5.2: The general performance of DRNN and counterpart methods.

Dataset	Algorithm	MAE	RMSE	F1	NDCG	F1(miss)	NDCG(miss)
MLS	NNCF	0.737	0.938	0.881	0.668	0.131	0.155
	NNCF$_z$	**0.723**	**0.927**	0.888	0.670	0.172	0.170
	ERPM	0.787	0.992	0.871	0.635	0.178	0.168
	BDA	0.767	0.973	0.902	0.679	0.326	0.298
	DRNN	0.749	0.956	**0.921**	**0.698**	**0.383**	**0.334**
MLM	NNCF	0.718	0.912	0.902	0.655	0.166	0.181
	NNCF$_z$	**0.709**	**0.908**	0.909	0.667	0.171	0.201
	ERPM	0.752	0.968	0.886	0.621	0.179	0.191
	BDA	0.744	0.946	0.925	0.672	0.342	0.347
	DRNN	0.736	0.938	**0.932**	**0.688**	**0.402**	**0.389**
JST	NNCF	0.673	0.841	0.774	0.757	0.332	0.379
	NNCF$_z$	**0.658**	**0.822**	0.782	0.791	0.373	0.408
	ERPM	0.701	0.881	0.762	0.748	0.352	0.391
	BDA	0.680	0.844	0.803	0.799	0.523	0.605
	DRNN	0.676	0.843	**0.853**	**0.813**	**0.637**	**0.689**

The result shows that the scalar-based NNCF and NNCF$_z$ methods perform best in terms of prediction accuracy (i.e. lowest MAE and RMSE). NNCF$_z$ performs better than NNCF, showing that taking into account the variance of users' voting profiles indeed helps. In terms of the binary and ranked recommendation accuracy metrics (i.e. F1, F1(misses), NDCG, and NDCG(misses)), the distributional rating-based BDA and DRNN methods perform best, with DRNN being the clear winner. The disparity between the prediction metrics and the recommendation metrics was also noted by Mclaughlin and Herlocker [87], who suggest that cause is that scalar-based methods are "*plagued with obscure or highly inaccurate recommendations at the top positions*", causing their binary and ranked recommendation accuracies to be low.

The result also shows that DRNN makes a much larger improvement over other methods under the "miss"-based metrics (i.e. F1(miss) and NDCG(miss)). For example, it makes a 4% improvement over NNCF on the MLS dataset under the standard F1, but makes a 25% improvement under F1(miss). Figure 5.20 plots the results in table 5.2 but only shows the "miss"-based metrics. The advantage of DRNN (in red) over scalar methods (in grey) and other distributional rating methods (in blue) can be clearly observed in this diagram.

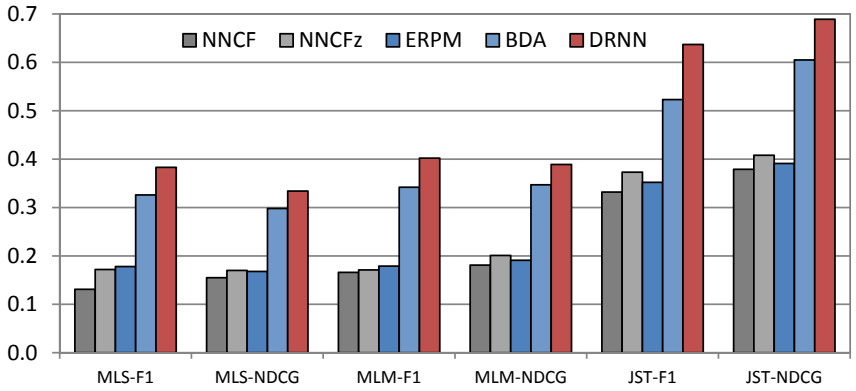


Figure 5.20: Comparison of DRNN and baseline algorithms using the "miss"-based metrics (i.e. F1 and NDCG with the unverifiable recommendations treated as misses).

Since both BDA and DRNN are distributional rating-based nearest neighbour algorithms with similar discrete probability representations, the bet-

ter performance of DRNN over BDA could be due to three reasons: firstly, DRNN uses the inverse base matrix transformation to eliminate the "contaminations" introduced by the manually constructed base matrix (further evaluated in section 5.7.3); secondly, DRNN uses distributional instead of scalar similarity computation (evaluated in section 5.7.4); and thirdly, DRNN provides a "smoother flow" compared to BDA, which requires many steps of scalar–distributional rating conversion and the "rounding down" of non-integer ratings during the recommendation process, which loses information.

5.7.2 Indirectly evaluating distributional rating predictions in DRNN

As argued in section 5.4.3, one major advantage of DRNN is it can produce distributional rating predictions, which can serve as a more informative form of recommendation feedback, and is capable of communicating complicated recommendations such as "*sorry, I don't have enough information to make a recommendation*", "*I think you will either love or hate this movie*", or "*you are likely to find the movie not bad, but it is definitely not a mind-blower*".

Due to the lack of higher-order user feedback in the datasets, there is no way to directly examine the effect of distributional recommendation. This section proposes to indirectly evaluate distributional recommendations by measuring its decision support performance using binary and ranked metrics, where the binary recommendation sets and ranked recommendation lists are generated using different manipulations of the distributional predictions.

Five manipulation approaches are evaluated. They are 1) *scalar rating* (SR), which is the standard approach of converting the distributional predictions into scalar predictions first, then generating the recommendation set or list based on the scalar predictions; 2) *highest peak with scalar rating* or HP(SR), which uses the most prominent peak of a distributional prediction (i.e. the rating value with the highest probability) as the key to sort the predictions, then generate recommendation set or list based on this ordering. In case there is a tie, the converted scalar predictions of the tied

distributional predictions are used as tie-breakers; 3) *highest peak with variance* or HP(VAR), which is the same as HP(SR), but uses the variance of the distributional predictions instead of the converted scalar rating as tie breakers (small variance first); 4) *sum probability 345* or SP345, which uses the sum of the probabilities of rating values 3, 4, and 5 as sorting key; 5) *sum probability 45* or SP45, which uses the sum of the probabilities of rating values 4 and 5 as sorting key.

Figure 5.21 shows the evaluation results of the five approaches on the MLS dataset, with SR marked in red,[7] the highest peak-based approaches (i.e. HP(SR) and HP(VAR)) marked in yellow, and the probability sum-based approaches (i.e. SP345 and SP45) marked in purple. The results show that different conversion approaches indeed make a difference, with the standard SR (scalar rating) approach used in the rest of this chapter being only the third best. It demonstrates the potential power of distributional rating predictions — individual users can set their recommendation list or set to use a particular approach based on their special needs for maximum satisfaction.

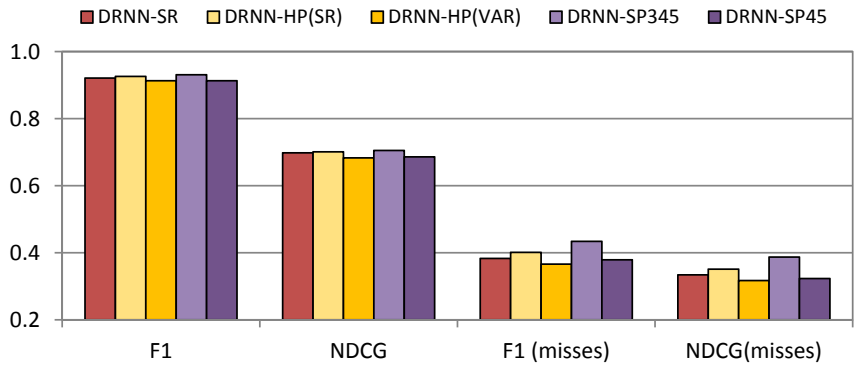


Figure 5.21: The recommendation accuracy using different (binary and ranked) recommendation generation approaches based on distributional predictions.

[7]The experimental results of this chapter try to use the same colour for the same algorithm and settings. For example, the red colour in figure 5.21 (i.e. DRNN-SR) and the red colour in figure 5.20 (i.e. DRNN) both correspond to the same algorithm and settings.

5.7.3 Evaluating the effects of base matrix rating transformation in DRNN

One unique contribution of DRNN is the use of a base matrix to transform scalar ratings (i.e. its corresponding index vectors) to distributional ratings, and the use of the matrix inversion of the base matrix to transform distributional predictions back to scalar predictions. In this way, the bias introduced by the manually constructed base matrix is eliminated from the final predictions.

This section evaluates the effect of the inverse base matrix transformation by comparing predictions generated with this extra step with predictions generated without it. The results are shown in figure 5.22. The results suggest that predictions generated without the extra step are generally less accurate based on both the prediction and the recommendation metrics.

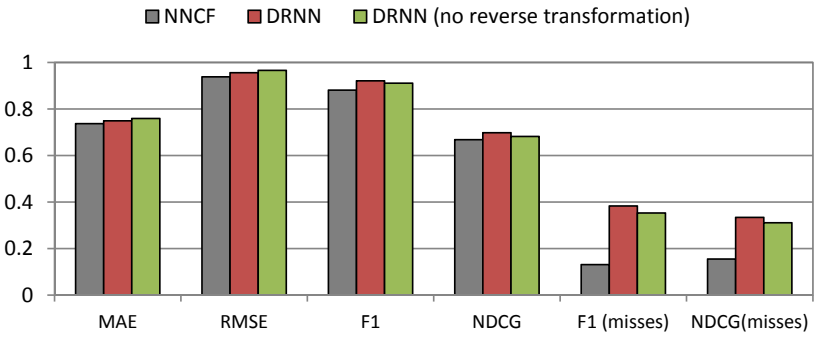


Figure 5.22: The performance of DRNN with and without the inverse base matrix transformation step.

This section also examines the effects of different base matrices on the performance of DRNN. Six base matrices presented in figure 5.23 are examined here. The six matrices are chosen to show different properties in terms of their *variance*, *symmetry*, and *smoothness*. The properties are reflected in the subscript of the matrix's name. The subscripts "s" (sharp) and f (flat) correspond to the variance of the matrix rows. A sharp base matrix assumes less noise in the input ratings than a flat matrix. The subscript c (crisp) corresponds to the smoothness of the base matrix. A crisp base matrix does not assume the "gradual dropping" effect, implying that a rating of 5 is just as likely to actually be a rating of 1 as to actually be a

rating of 4. The subscripts "**b**" (balanced) and **u** (unbalanced) correspond to the symmetry of the base matrix. An unbalanced or non-symmetrical base matrix assumes different noise level (i.e. variance) between rows 1 and 5 and between rows 2 and 4. Based on this, the six matrices can be interpreted as: $\mathbf{B_{sb}}$: sharp and balanced; $\mathbf{B_{fb}}$: flat and balanced; $\mathbf{B_{cb}}$: crips and balanced; $\mathbf{B_{su}}$: sharp and unbalanced; $\mathbf{B_{fu}}$: flat and unbalanced; $\mathbf{B_{cu}}$: crips and unbalanced.

$$\mathbf{B_{sb}} = \begin{bmatrix} .80 & .10 & .05 & .03 & .02 \\ .10 & .70 & .10 & .06 & .04 \\ .05 & .15 & .60 & .15 & .05 \\ .04 & .06 & .10 & .70 & .10 \\ .02 & .03 & .05 & .10 & .80 \end{bmatrix} \quad \mathbf{B_{fb}} = \begin{bmatrix} .50 & .25 & .15 & .07 & .03 \\ .20 & .45 & .20 & .10 & .05 \\ .10 & .20 & .40 & .20 & .10 \\ .05 & .10 & .20 & .45 & .20 \\ .03 & .07 & .15 & .25 & .50 \end{bmatrix} \quad \mathbf{B_{cb}} = \begin{bmatrix} .96 & .01 & .01 & .01 & .01 \\ .01 & .96 & .01 & .01 & .01 \\ .01 & .01 & .96 & .01 & .01 \\ .01 & .01 & .01 & .96 & .01 \\ .01 & .01 & .01 & .01 & .96 \end{bmatrix}$$

$$\mathbf{B_{su}} = \begin{bmatrix} .80 & .10 & .05 & .03 & .02 \\ .10 & .70 & .10 & .06 & .04 \\ .05 & .15 & .60 & .15 & .05 \\ .04 & .06 & .12 & .65 & .13 \\ .03 & .05 & .07 & .15 & .70 \end{bmatrix} \quad \mathbf{B_{fu}} = \begin{bmatrix} .60 & .20 & .10 & .07 & .03 \\ .10 & .55 & .20 & .10 & .05 \\ .10 & .20 & .40 & .20 & .10 \\ .05 & .10 & .20 & .45 & .20 \\ .03 & .07 & .15 & .25 & .50 \end{bmatrix} \quad \mathbf{B_{cu}} = \begin{bmatrix} .96 & .01 & .01 & .01 & .01 \\ .01 & .96 & .01 & .01 & .01 \\ .05 & .05 & .80 & .05 & .05 \\ .05 & .05 & .05 & .80 & .05 \\ .05 & .05 & .05 & .05 & .80 \end{bmatrix}$$

Figure 5.23: Six different base matrices, where the subscripts indicate the properties of the matrices: **s**: *sharp*, **f**: *flat*, **c**: *crisp*, **b**: *balanced*, and **u**: *unbalanced*.

Figure 5.24 presents the performance of DRNN with different base matrices on the MLS dataset. It shows that the sharp and smooth matrices $\mathbf{B_{sb}}$ and $\mathbf{B_{su}}$ perform the best, with the unbalanced, sharp, and smooth matrix $\mathbf{B_{su}}$ (in red) being the best performing matrix by a slight margin. $\mathbf{B_{su}}$ is also the base matrix used in other experiments presented in this thesis without further specifications. The crisp matrices $\mathbf{B_{cb}}$ and $\mathbf{B_{cu}}$ perform significantly worse, showing that smoothness is important in designing base matrices.

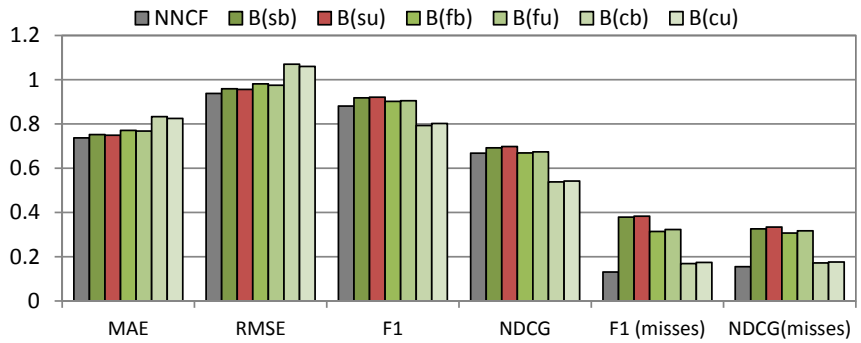


Figure 5.24: The performance of DRNN with different base matrices.

5.7.4 Examining distributional similarity in DRNN

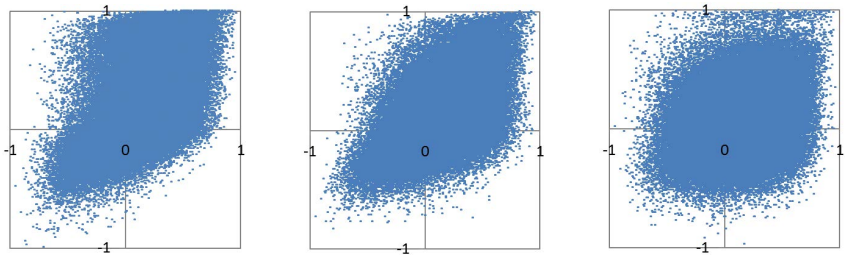
This section investigates the idea of *distributional similarity vector*, which was proposed in section 5.5.3 with its benefits explained in section 5.4.4. Basically, instead of using one scalar number to capture the correlation between users, the correlation is captured by a vector of 5, with each entry corresponds to a rating value on the rating scale. This allows the representation and learning of more complex user correlations such as "*Ann agrees with Bob on movies she hates, but on not the ones she likes*".

Figure 5.25 plots the correlations between the scalar similarity of NNCF (x-axis) and the distributional similarities of DRNN (y-axis). In the six diagrams, (a) to (e) correspond to the rating values of $r = 1$ to $r = 5$ respectively, and (f) is the average of the first five diagrams along the y-axis. In each diagram, every data point represents a user pair; the x-coordinate corresponds to the scalar similarity of this user pair; the y-coordinate corresponds to the distributional similarity of this user pair at the specified rating value. Only the user pairs with more than 10 common ratings are plotted, because below that, there is not enough data to justify calculating the similarities for different rating values separately, and the similarities tend to be extreme (i.e. 1 or -1) and erratic.[8]

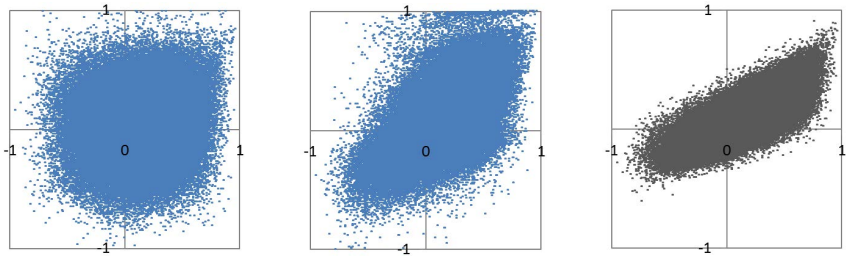
Diagram (f) shows a clear positive linear correlation between the scalar similarity (x-axis) and the distributional similarity averaged over all rating values (y-axis). This is well expected, since both similarities are supposed to measure the same thing — the correlations between user ratings.

Diagrams (a) to (e) also show positive linear correlations between the x and y coordinates, but with much larger variances. This means that, although correlated, the similarity of a user pair on a particular rating value can be very different from this user pair's overall similarity across all rating values. This demonstrates that the key scenario raised by distributional rating — "*A agrees with B on movies he hates, but not on the ones he likes*" — indeed exists and quite prevalent.

[8]For the same reason, in DRNN, distributional similarity is only used when users have more than 10 common ratings, and the scalar similarity is used otherwise. This number is determined by empirical experiments, and includes about 50% of the calculations.



(a) $s_{r=1}(u,v)$ (b) $s_{r=2}(u,v)$ (c) $s_{r=3}(u,v)$



(d) $s_{r=4}(u,v)$ (e) $s_{r=5}(u,v)$ (f) Average: $\frac{1}{5}\sum_r s_r(u,v)$

Figure 5.25: The correlations between the scalar similarity (x-axis) and the distributional similarities (y-axis) with rating values 1 to 5 (in (a) to (e)), and the average (in (f)). Each data point corresponds to a user pair in the dataset.

A more interesting observation is the "rounder" shapes of diagrams (c) and (d) compared to (a), (b), and (e). It means at rating values 3 and 4, the distributional similarity is more "variegated" and tend to comply less with the overall similarity. This could be due to the fact that there are more ratings with values 3 and 4,[9] and the distributional similarity has the nice property of converging to the average when there is less data.

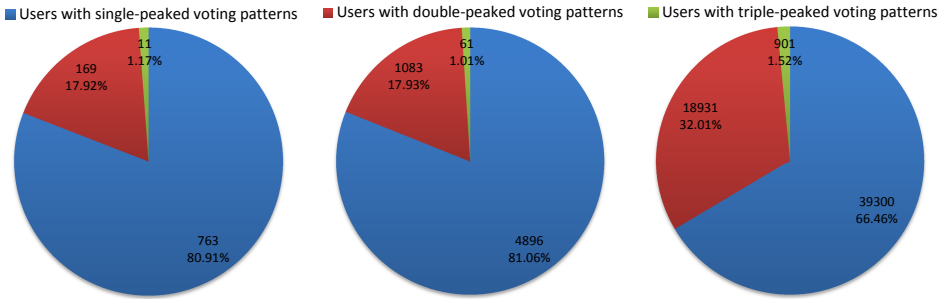
5.7.5 The performance of DRNN on multi-peaked users

This section investigates DRNN's performance on users with multi-peaked voting profiles — a concept identified on page 128. It firstly demonstrates the correlations between the recommendation accuracy and the number of peaks, proving that the number of peaks is indeed an important metric. It then shows that DRNN tends to make a bigger improvement over the

[9] 61% of the MLS and MLM datasets have these *two* ratings values (i.e. 3 and 4), and the other *three* rating values (i.e. 1, 2, and 5) only occupy 39% of the data. Similarly, 62.4% of the JST dataset have ratings between 2.5 and 4.5.

scalar method on multi-peaked users.

A significant portion of users have multi-peaked rating patterns. As shown in figure 5.26, about 20% of the users in the MLS and MLM datasets and 33.5% users in the JST datasets have multi-peaked voting patterns, among which slightly more than 1% are triple-peaked.



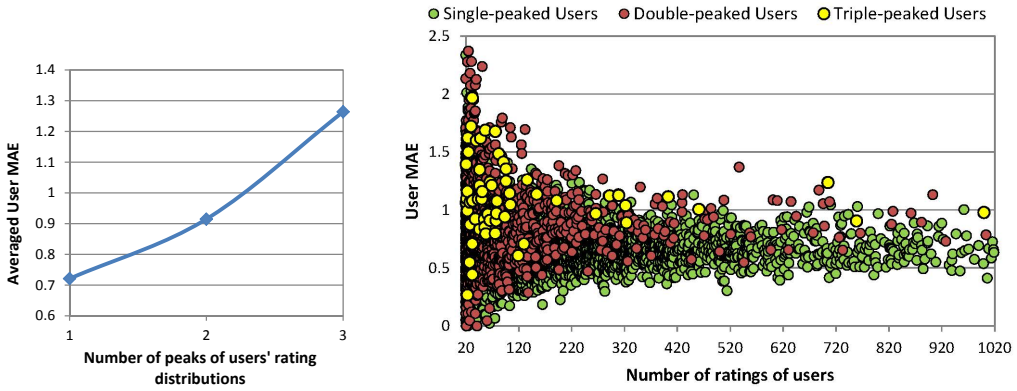
(a) MLS Dataset (b) MLM Dataset (c) JST Dataset

Figure 5.26: The distribution of single, double, and triple-peaked user voting patterns in the MLS, MLM, and JST datasets.

The number of peaks is strongly correlated to the prediction error of NNCF.[10] Figure 5.27(a) shows the relationship between the number of peaks (x-axis) and the user MAE averaged over all users with the same number of peaks (y-coordinates). The data is based on the MLS dataset with skip-every-10^{th}. The positive slope of the curve clearly indicates that more prediction mistakes are made with multi-peaked users.

Figure 5.27(b) further backs up this observation by trying to eliminate the effect of the number of ratings of users — another factor that also tends to affect prediction accuracy. In this figure, each circle represents a user, the x-coordinate is the number of ratings a user has, and the y-coordinate represents the user's average MAE. For all x-values, the yellow dots (triple-peaked users) tend to have a higher error than the red dots (double-peaked users), which tend to have a higher error than the green dots (single-peaked users). It shows that in general, more prediction mistakes are made to users with more peaks despite the number of

[10]The correlation between the number of peaks and MAE is is also tested on other recommender algorithms, including TASK, PANDA, naive Bayes, K-means clustering, and PLSA. The results are very similar to the NNCF results shown here.



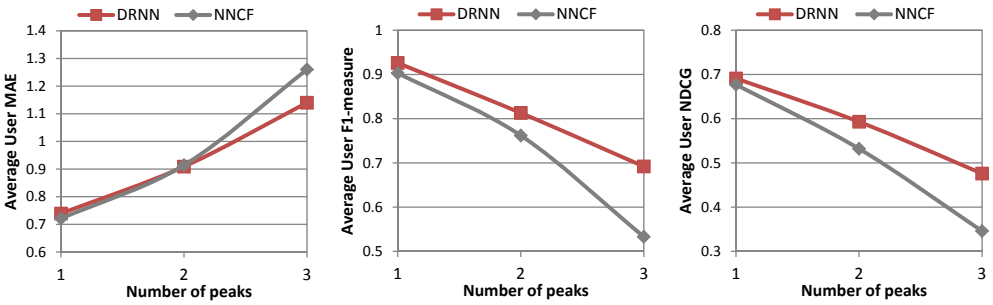
(a) The general effect of the number of peaks on the prediction MAE.

(b) User's prediction MAE vs. the number of ratings the user has, with users with different number of peaks plotted separately.

Figure 5.27: The effect of the *number of peaks* on the prediction MAE of users.

ratings they have.

Figure 5.28 is similar to figure 5.27(a), but shows the effect of the number of peaks on the performance of both NNCF and DRNN under the MAE, the F1 measure, and the NDCG metrics. It can be observed that in all three subfigures, the curves for DRNN (red) tend to be flatter than the curves for NNCF (grey). This means that the performance of DRNN is less affected by the number of peaks. In other words, DRNN is able to cope better with difficult users who have multi-peaked voting profiles.



(a) MAE (b) F1 Measure (c) NDCG

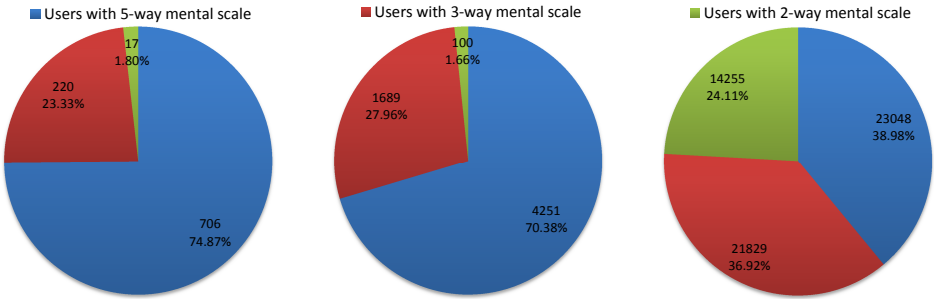
Figure 5.28: The general effect of the number of peaks on the performance of NNCF and DRNN.

5.7.6 The performance of DRNN on users with different mental scales

Another DVP-inspired value that strongly correlates to the recommendation accuracy is the user's *true mental scale* (as presented on page 127). It refers to how the user actually views and uses the rating scale. Three mental scales are identified, they are 2-way, 3-way, and 5-way. In a 5-way mental scale, the user utilises all five rating values as intended; in a 2-way mental scale, the user ignores three out of the five rating values, and only uses two rating values to indicate a binary preference of *like* and *dislike*; a 3-way mental scale ignores two rating values, and map the rest three onto the preference of *like*, *mediocre*, and *dislike*.

This section investigates DRNN's performance on users with different mental scales by firstly demonstrating the correlations between the mental scales and recommendation accuracy, then showing that DRNN tends to make more improvement on users with complicated (i.e. 5-way) mental scales.

A significantly portion of users have simplified (i.e. 2-way or 3-way) mental scales. As shown in figure 5.29, about 25%, 30%, and 61% of users in the MLS, MLM, and JST datasets have simplified mental scales, with 3-way mental scales much more common than 2-way ones.

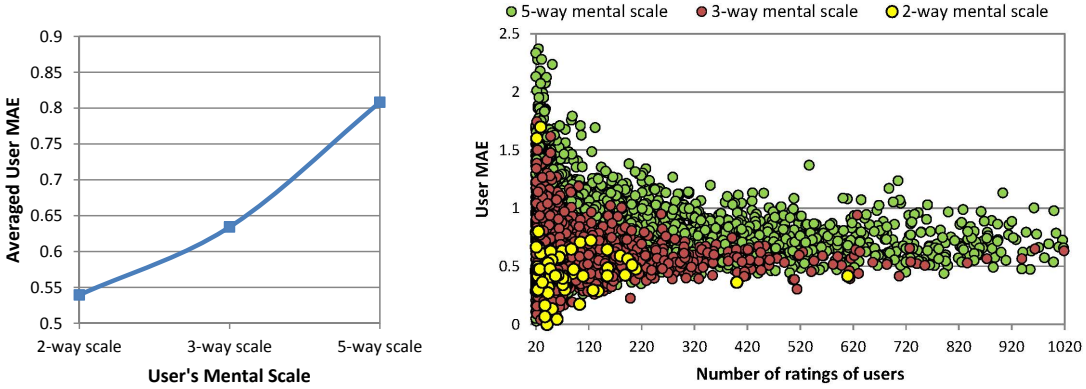


(a) MLS Dataset (b) MLM Dataset (c) JST Dataset

Figure 5.29: The distribution of 2-way, 3-way, and 5-way mental scaled users in the MLS, MLM, and JST datasets.

The simplicity of the mental scales is strongly correlated to the prediction error of NNCF. As shown in figure 5.30(a), the simpler the men-

tal scale, the lower the average MAE, which indicates better prediction accuracy. Figure 5.30(b) further backs up this conclusion by eliminating the effect of the number of ratings of users, which is another factor correlated with the MAE. In this figure, each data point represents a user; the x-coordinate is the number of ratings this user has; the y-coordinate is the user's average MAE across all items tested. The figure shows that across the entire x-axis, the yellow dots are generally below the red dots, which are below the green dots. It means that regardless of the number of ratings the user has, NNCF tends to perform better on users with simpler mental scales. This is understandable, since a simpler mental scale can be viewed as an easier recommendation task.

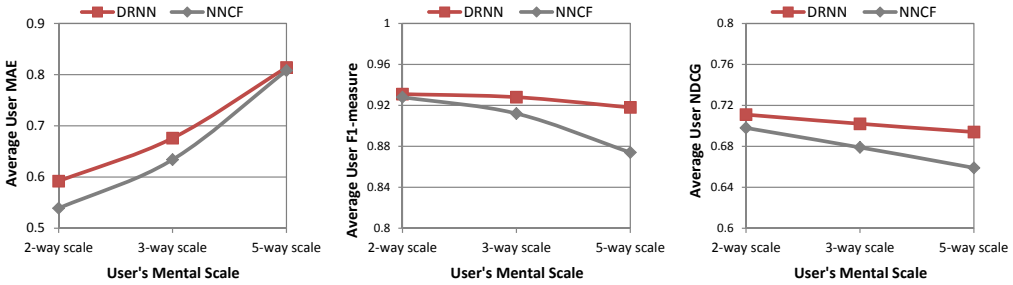


(a) The general correlation between mental scales and prediction MAE.

(b) User MAE vs. the number of ratings the user has, with users with different mental scales coloured differently.

Figure 5.30: The correlation between the users' *actual mental scales* and the prediction MAE under the standard NNCF algorithm.

Figure 5.31 is similar to figure 5.30(a), but shows the effect of different mental scales on the performance of both NNCF and DRNN under the MAE, the F1 measure, and the NDCG metrics. It can be observed that in all three subfigures, the curves for DRNN (red) tend to be flatter than the curves for NNCF (grey). This means that the performance of DRNN is less affected by the variation of mental scales. In other words, DRNN is able to cope better with difficult users who have complicated (i.e. non-simplified 5-way) use of the rating scale.



(a) MAE (b) F1 Measure (c) NDCG

Figure 5.31: The general effect of the user's actual mental scale on the performance of NNCF and DRNN.

5.8 Conclusion

This chapter presented DRNN — a distributional rating-based nearest neighbour recommendation process. It includes a proper definition of the probability space for recommendation (section 5.1), the syntax and semantics of distributional rating (section 5.2) and distributional voting profile (section 5.3), the transformation from scalar rating to distributional rating (5.5.1), the normalisation of distributional ratings against distribution voting profiles (5.5.2), distributional similarity computation (5.5.3), distributional rating prediction (in 5.5.4), and if required the transformation from distributional predictions back to scalar predictions (5.5.1).

As pointed out in section 5.6, there have been previous attempts that use probability distributions to model various parts and stages of the recommendation process, ranging from using a Gaussian or a discrete vector representation to capture user priors, item priors, group or cluster priors, or rating differences in the rating normalisation, rating prediction, or recommendation list generation stages. However, each one of these previous attempts only investigates parts of the whole picture: the algorithms that study distributional voting profiles do not use distributional rating, and vice versa; the algorithms that use distributionalised rating normalisation do not use distributionalised rating prediction, and vice versa. On top of that, no algorithm has ever addressed the power of distributional similar-

ity measure, which allows the communication of complicated knowledge model such as "*Ann only agrees with Bob on movies he hates, but not the ones he likes*".

DRNN is the first implementation that encompasses both distributional ratings and distributional voting profiles, and uses them in all stages of the recommendation process, including rating normalisation, similarity computation, and rating prediction. It is also the first to properly define a rating transformation process between scalar and distributional ratings in a way that is mathematically consistent — a distributional to scalar transformation on a distributional rating that was transformed from a scalar rating is guaranteed to be transformed back to its original value. Apart from the algorithmic contributions, this chapter also provides a comprehensive discussion on the utility, purpose, and ramification of distributional rating in section 5.4.

Empirical studies presented in section 5.7 show that the DRNN process is able to improve the recommendation quality of NNCF in terms of F1-measure and NDCG. Comparison with two existing distributional implementations also show the advantage of DRNN's integrated use of distributional rating, distributional voting profile, and distributional similarities in all stages of the recommendation process. More in-depth investigations show that DRNN is less affected by complicated user patterns such as multi-peaked voting profiles or non-simplified mental scales.

Future work on DRNN includes the following directions: 1) to construct user studies to establish the proper noise pattern of user input based on real data, thus eliminating the need to manually constructing a canonical base matrix and the inductive bias brought by this manual construction; 2) to construct user studies to properly evaluate the effect of distributional rating predictions on end users, instead of having to convert the distributional predictions back into scalar forms for evaluation; 3) to use distributional rating and distributional voting profile in other recommendation structures such as clustering or aspect models.

Another less concrete direction is to use the expressiveness and power of DRNN to explore alternative thoughts and philosophies that were previously too ethereal to become a concrete algorithm. An example is the

Diamond algorithm described in chapter 6, which is an alternative way of implementing hybrid filtering based on a previous idea that is only made possible by the ecosystem provided by DRNN.

Chapter 6

The *Diamond* Hybrid Filtering System

Hybrid filtering is the mechanism of using both collaborative filtering (CF) and content-based filtering (CBF) techniques to make recommendations. The concept was first proposed in the late 1990s [9, 32], and quickly grew to become a prolific field in terms of both research and industrial use. A detailed survey on current hybrid filtering techniques was provided in section 2.5.

The core of a hybrid filtering system is its *hybridisation strategy* [25], which defines how CF and CBF cooperate to make recommendations. Existing hybridisation strategies are designed to leverage the different learning and recommendation properties of the two filterings, and typically implement either a *combinatorial* or a *sequential* hybridisation structure.[1] This chapter digs down into the fundamental divisions of recommendation tasks, and proposes an intrinsically different hybridisation structure — *Diamond*— based on the idea of splitting each recommendation task into a collaborative sub-task and a content-based sub-task, so that both filterings can shine in their own respective area without interfering with each other.

[1]Other than *combinatorial* and *sequential*, there are also non-communicative hybridisation strategies (described in section 2.5.3) where CF and CBF merely co-exist but do not cooperate with each other.

The chapter is organised as follows: section 6.1 outlines the conceptual reasoning behind the design of *Diamond*; sections 6.2 and 6.3 present the theoretical basis of *Diamond*, with 6.2 formulating the probabilistic dependencies, and 6.3 presenting the mathematical calculation; section 6.4 describes the core of the system — its hybridisation strategy; experimental evaluations are presented in sections 6.5; finally, section 6.6 concludes the chapter and indicates future work.

6.1 The Conceptual Underpinning of *Diamond*

This section explains the conceptual underpinning of *Diamond* by going through the thinking process that led to its invention. The intention of this section is to create an abstract and conceptional understanding, instead of making scientific assertions, thus it contains postulations without citations or experimental backup.

6.1.1 The Inductive Bias of the Content Representation

Since content-based filtering makes recommendations solely based on item content, there have been arguments regarding its inability to model user subjectivity and item quality. For example, Wei et al. [153] wrote "content-based approaches are based on objective information about the items and do not take the user's perceived valuation of such *subjective attributes* into account"; Lee [73] stated that "An even more serious drawback to purely content based methods is that even the state of the art methods are usually not able to capture *subjective qualities* that affect ratings in domains such as movies and music"; Montaner et al. [98] argued that "Content-based approaches are based on objective information about the items ... However, [user] selecting one item or another is based mostly on *subjective attributes* of the item (e.g., a well-written document or a product with a spicy taste) ... which are not taken into account".

Such accusations are both right and wrong. They are right because CBF has not been seen to capture subjectivity and item quality adequately. They are wrong because such failure is not due to some intrinsic weaknesses of the CBF recommender algorithms, but the incompleteness of the content representations upon which CBF operates.

As described in section 2.3, the *content representation* is an often manually constructed abstract characterisation of the item content that defines how the items are to be perceived by the CBF learning algorithms. In content-based filtering, recommendations are made by firstly characterising users by the content of the items they are interested in, and then recommending new items whose content conforms to the user's preferred content pattern. If the content representation of items misses important components essential to the representation of certain parts of the user's preference pattern, these parts would never be captured regardless of how good the actual learning algorithm is. For example, a content representation of movie items that only provides the names of actresses but not the actors would not satisfy William Shatner fans; a content representation that does not provide the composer of the movie's theme song would not pick up on John Williams fans; a content representation that does not provide the author of the movie's original novel would fail to recommend *Dolores Claiborne* to people who loved *The Shining*.[2]

Similarly, *user subjectivity* and *item quality* are not distinguished properties but are two broad concepts describing aspects of items the content representations often fail to adequately represent. To elaborate, the *quality* of an item refers to how well the item is made. Although rarely represented directly, items normally have a set of content attributes which, if present, would give some indication of its quality. Examples include the thread-count of a duvet cover, the number of layers of a cake, the turnover of a movie, or the price of a handbag.

The *subjectivity* of a user refers to the user's view on the more subtle aspects of an item where the preferences across all users appear to be too volatile to capture in any simple way. When a user exhibits different

[2] *Dolores Claiborne* and *The Shining* are completely unrelated in terms of genres, directors, stars, plots, and so on, except that they are both based on novels by Stephen King.

preferences on two seemingly identical items, or two users who have otherwise exhibited identical preferences rated oppositely on the same item, we blame subjectivity. The problem here is that what seems to be identical is limited by our perception, or in this case the content representation of the CBF system. When talking about the subjectivity of human preferences, most of the time, there are some latent yet quantifiable factors that the users themselves may not even be consciously aware of that resulted in it. For example, a mother may possess a preference for one of two identical twins who behave very similarly; she may not know why, but it could be that it was the twin that she always happened to pick up and tend to in their infancy, when her and her husband were supposed to tend to one each at random.

Generally speaking, the more complete and pertinent the content representation, the better the chance the CBF algorithms will capture user subjectivity and item quality. For example, if the content representation of fine-art items include attributes such as "*consistency, skilfulness, creativity, individuality, imagination, harmony, visual balance, coordination, message*", rather than the objective "*name, author, author-age, genre, medium, year, institute*", both user subjectity and item quality would be much better represented, and consequently better recommended by the same CBF algorithm.

To summarise, the pertinence of content representation provides a strong inductive bias that is vital to the performance of content-based filtering. A poorly concocted content representation will greatly hinder the recommendation efficacy of the otherwise capable CBF algorithms. The common observations on CBF's inability to model user subjectivity or item quality are merely examples of the leg-dragging of inadequate content representation — since "*item quality*" is just a broad concept that generalises many quality-related attributes; and "*user subjectivity*", just like probability, is just an attempt to explain the lack of knowledge and understanding, or in this case, content representations.

6.1.2 The Underlying Meaning of Rating

The rating a user gives to an item represents the user's overall satisfaction regarding all aspects of the item. Collaborative filtering treats an item as a whole and does not distinguish its different aspects. On the other hand, content-based filtering explicitly categorises each item into different aspects based on its content representation. As a result, a numerical rating is effectively treated as a conglomerated indicator of the user's preferences on all the *represented* aspects of the item, and a rating prediction is produced by examining how well each *represented* aspect of the item matches the target user's preference pattern.

Like the aforementioned user subjectivity and item quality, all influential aspects regarding a user's preference on an item can potentially be explained, quantified, learned, and properly recommended if adequate building blocks are available.[3] If a system had an impossibly complete content representation that encompassed all aspects of the item as well as other temporal and contextual factors, it could in principle (i.e. with a *prefect* CBF algorithm) be able to consistently and perfectly accurately map this representation to the user's rating. This is presented in the first line of figure 6.1, where f_u represents the perfect CBF algorithm for the target user.

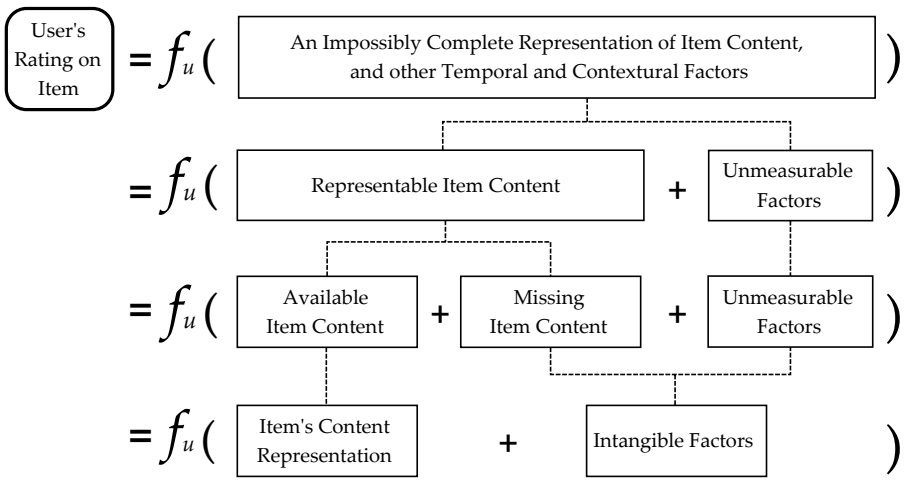


Figure 6.1: What a rating truly entails.

[3]This statement assumes a *Deterministic* view, which states that "for everything that happens, there are conditions such that, given them, nothing else could happen — *Wikipedia*".

However, due to our limited understanding of the universe and human behaviour, some complicated aspects, such as user subjectivity, may appear to be too chaotic to quantify. In this case, the user's rating on an item would depend not only on his preference on the *representable* content, but also on his preference of the *unmeasurable* factors. This is represented in line 2 of figure 6.1.[4] Consequently, even the perfect CBF algorithm would not be able to perfectly model user preferences due to its inability to model the *unmeasurable factors* — as they are absent from the representation.

It is safe to say that, all modern recommendation datasets only feature a *subset* of all the *representable* content. An exhaustive content representation is not practical due to 1) the complexity of its construction, and 2) the stress it would put on the content-based learners — current computer hardware and machine learning methods (such as decision trees and belief nets) cannot yet tractably handle the construction of a model where there are millions of attributes to choose from. Therefore, given a dataset, all representable item content can be classified as either *available* in the dataset, or *missing* from the dataset, as presented in line 3 of figure 6.1.

The incompleteness of the content representation indicates the absence of potentially important factors. Whether it is caused by missing attributes or innate unmeasurability does need not be distinguished in practice. This thesis introduces the *intangible factor* to represent a conglomeration of all aspects of the item that are pertinent to the modelling of user preferences, but are absent from the content representation. This is presented in line 4 of figure 6.1.

The *intangible factor* of an item can be considered as a single attribute representing a soft-generalisation of all the absent attributes in the content representation, and therefore complete the representation to some extent while maintaining tractability. It is also worth noticing that when an item's content data is not available, the item's content representation would be null, making the user's rating on the item correspond to the user's preference only on the *intangible factor* of the item, thus converging to the view of collaborative filtering.

[4]Since the thesis does not deal with temporal or contextual factors, they are also grouped into the *unmeasurable* factors without affecting the general argument.

6.1.3 A New Reasoning for Hybrid Filtering

Recommender system data comes in the form of ratings that indicate users' preferences on items. Based on the last line of figure 6.1 (reformatted here as figure 6.2), each rating datum can be considered as containing a combination of two kinds of information — *content preference*, which is the user's preference on the content attributes of the items, and *intangible preference*, which is the user's preference on the other aspects of the items not covered by its content representation.

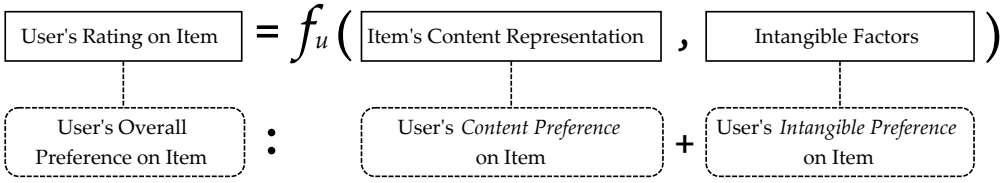
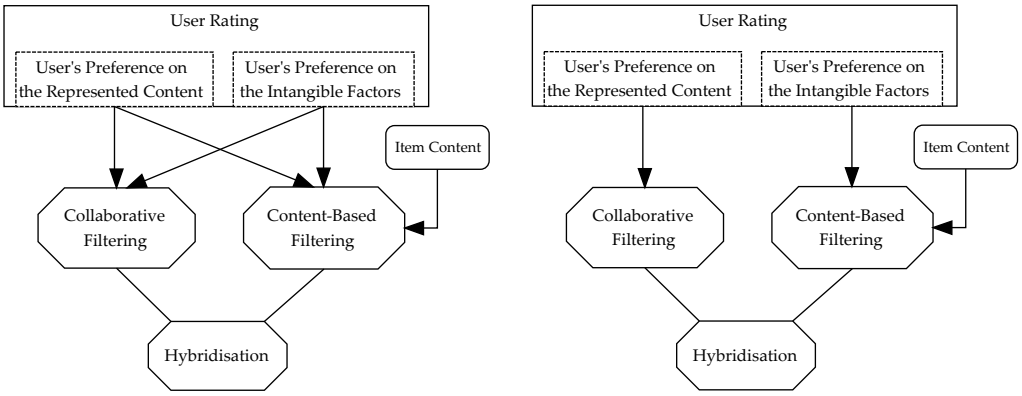


Figure 6.2: This figure is an extension of the last line of figure 6.1. The symbol f_u represents the "perfect recommender system". It states that user's rating on item, which corresponds to the user's overall preference on the item, can be viewed as a combination of the user's *content preference* and his *intangible preference*.

The learning mechanisms and knowledge representation of CBF is tailored to model item content, thus are attribute-oriented and highly structural. Forcing it to model the intangible preferences, which by definition means "not in the content representation", is an impossible task and is only going to confuse the learning engine, resulting in overfitting and inferior performance. On the other hand, CF mechanisms do not have the necessary building blocks — the content data — to effectively model users' content preferences, thus are significantly disadvantaged in the learning process. The extra burden would also result in worse performance at modelling the intangible preferences.

This thesis argues that, ideally, a recommender system should distinguish the content and the intangible preference, and learn the preference functions separately — content preferences should be learned only by content-based filtering; intangible preferences should be learned only by collaborative filtering. Because the distinction between content and intangible preferences were not made, all modern hybrid filtering systems

use user ratings, which encompasses both the content and intangible preferences, to train both its CF and CBF components. This is illustrated in figure 6.3(a). This is suboptimal, because the incompatible preference type (i.e. content preference for CF and intangible preference for CBF) does not fit the learning and modelling bias of the CF and CBF algorithms. Forcing them to handle both is like forcing a brainy child and a sporty child to excel both intellectually and athletically. Not only will they not be able to do very well in their disadvantaged fields, they will also perform worse than they could have in their advantaged fields due to split attention.



(a) The conventional approach used by other hybrid filtering systems — both content preferences and intangible preferences are used to train both the collaborative filtering and content-based filtering engines.

(b) The new approach used by *Diamond* — content preferences are handled only by content-based filtering, intangible preferences are handled only by collaborative filtering.

Figure 6.3: The different ways that CF and CBF engines in a hybrid filtering setting treat user's content preferences and his intangible preferences.

In this chapter, a new hybrid filtering framework — the *Diamond*— is proposed. It has a unique separation mechanism that separates users' general preference data (i.e. user's rating data) into users' *content preference* data and users' *intangible preference* data prior to training. The two separated sets of data are then handled separately by the content-based and the collaborative filtering engines. This is shown in figure 6.3(b). The separation of data eases the pressure on the filtering engines and allows them to concentrate on the forms of learning that fit their natural bias.

6.2 The Two New Variables

The *Diamond* hybrid filtering system advocates the separate treatment of the content preferences and the intangible preferences. However, users' content preferences and their intangible preferences do not come readily available. Instead, the recommendation datasets only provide a *single* indicator to represent the user's *overall* preference on *all* aspects of the item, that is, the user's rating on the item — $r_{u,i}$.

To solve this problem, *Diamond* has a mechanism that enables the separation of the content preferences and the intangible preferences from user ratings. This is presented in section 6.4 as part of *Diamond*'s hybridisation strategy. The mechanism has a strong probabilistic basis, so in order to properly describe it, this section firstly lays out the necessary background for its presentation.

As pointed out in sections 4.1 and 5.1, the three fundamental variables in a recommendation problem setting are the user u, the item i, and the rating $r_{u,i}$. The *Diamond* system is effectively proposing to split the item variable i into two separate variables that represent item content (i_{C}) and the intangibles ($i_{\times}$) respectively. This section discusses the ramification of this separation on the probabilistic correlations between the variables.

6.2.1 The Three Original Variables

This section uses *belief nets* [7] to clarify the probabilistic correlations between the three original variables — u, i, and r — within the probability space described in section 5.1. A belief net, as shown in figure 6.4, is a *probabilistic graphical model* that provides a compact visual view of the probabilistic (in)dependence relationships between a set of random variables. In a belief net, the nodes represent variables, and the edges represent probabilistic correlations between variables — for two adjacent nodes, an edge between them represents probabilistic dependency, and a lack thereof represents probabilistic independence; for non-adjacent nodes, variables are conditionally independent of their non-descendants given their parents.

Figure 6.4 is the belief net visualisation of the dependencies between the three variables u, i, and r postulated by this thesis. In this model, both u and i are parents of r. This is easy to understand, as it takes both the preference of a user and the properties of an item for a rating to be drawn. The user-rating and item-rating relationships can even be viewed as causal — the fact that user *Tom* voted item *Titanic* a rating of *good* can be interpreted as that the user being *Tom* and the item being *Titanic* "caused" the rating to take on a value of *good*.

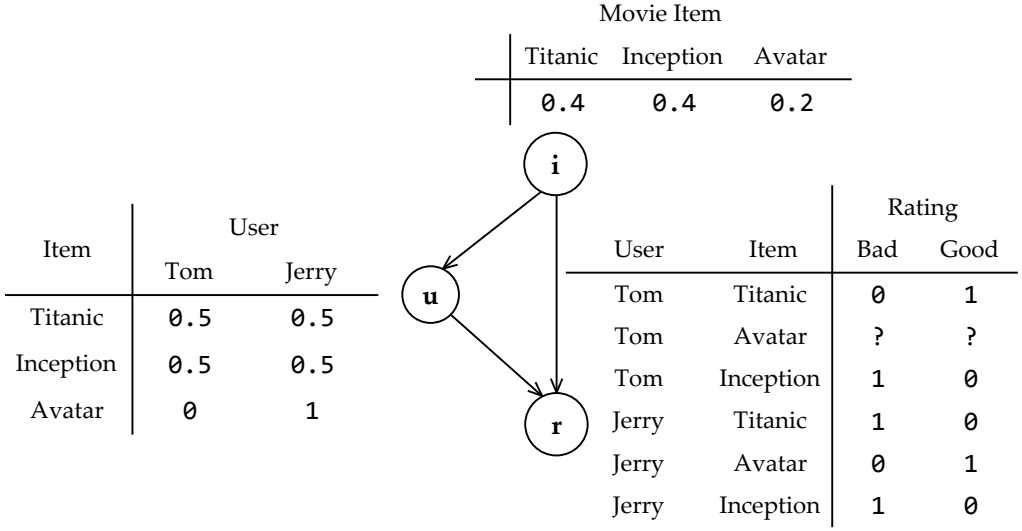


Figure 6.4: A belief nets view of the dependencies of the three original variables.

In terms of variables u and i, firstly, a connection between them definitely exists because they are not independent from each other. This can be validated by the fact that $\mathcal{P}(i) \neq \mathcal{P}(i \mid u)$, or that formulae 5.1b and 5.2b do not yield the same result. Secondly, the direction of this particular connection does not matter to the description of the probabilistic relationships between the three variables, thus can be considered as an "implementation choice". However, the direction of $i \rightarrow u$ is chosen because it makes the consequent belief nets (such as that of figure 6.5) more semantically sound.

In a belief net, each node is associated with a probability function or table that takes as input the value(s) of the parent node(s) and returns the probability of the current node. This allows probabilistic inference [7] to be carried out throughout the network. Figure 6.4 uses the illustrative dataset in table 6.1 to demonstrate this. Taking variable r as an example, its asso-

User	Item	Rating
Tom	Titanic	Good
Tom	Inception	Bad
Jerry	Titanic	Bad
Jerry	Avatar	Good
Jerry	Inception	Bad

Table 6.1: An illustrative recommendation dataset used to construct the probability tables associated with the belief net nodes in figure 6.4. (This dataset is identical to that of table 5.1).

ciated probability table contains all combinations of users and items (i.e. both its parents) on the left, and the corresponding probabilities on the right. The known or observed entries in the table constitute the training dataset in table 5.1, whereas the unobserved entries are potential prediction queries waiting to be answered. The process of recommendation can be viewed as the task of filling in the unknown entries of the table, and a recommender system can be viewed as a function approximation-based oracle for such task.

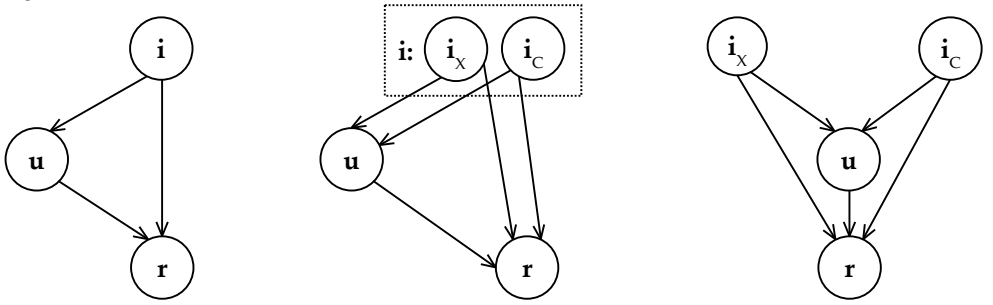
6.2.2 The Two New Variables

The *Diamond* system introduces two new variables — i_{C} that represents item i's content, and $i_{\times}$ that represents item i's intangibles — to replace the variable i that represents the item as a whole.[5] This section describes the probabilistic consequences of this action.

Upon the substitution of variable i with its two constitutional variables i_{C} and $i_{\times}$, the belief net structure in figure 6.4 (repeated as figure 6.5(a)) that represents the statistical correlations between variables u, i, and r also

[5] In theory, the *intangible* should represent all relevant factors that are not in the content representation, thus should theoretically include temporal factors (e.g. time of the day a recommendation is made) and contextual factors (e.g. the weather was sunny), which are not strictly properties of the item. However, these aspects are not the focus of this thesis. For further notes, they have their own dedicated research fields, known as temporal diversity study [72] and contextual diversity study [3, 155]. Therefore, this thesis simplifies the intangible to mean only the intangible part of the item — $i_{\times}$.

needs to be updated. Since the intangible $i_\times$ is defined as "everything about i except i_C", by definition, $i_\times$ and i_C together are supposed to fully "explain" the statistical correlations that variable i has with the other two variables u and r. Therefore, in figure 6.5(b), the variable i is replaced by $i_\times$ and i_C with both the $i \to u$ edge and the $i \to r$ edge duplicated for the new variables. This is then tidied up into figure 6.5(c), which is the final belief net that represents the statistical correlations between variables u, $i_\times$, i_C, and r.



(a) Dependencies between the three variables u, i, and r, as a repeat of figure 6.4.

(b) Substituting i with its constituent variables $i_\times$ and i_C. The dependencies (edges) of i are duplicated for both i_C and $i_\times$.

(c) The final probabilistic dependencies between the four variables u, $i_\times$, i_C, and r.

Figure 6.5: Belief nets that represent the probabilistic correlations between variables u, r, i, $i_\times$, and i_C.

Figure 6.5(c) specifies that the value of the rating r depends on the user u, the item content i_C, and the item intangible $i_\times$. It also describes the relationships between the newly introduced variables i_C and $i_\times$. Firstly, their prior probabilities are independent, namely $i_\times \perp\!\!\!\perp i_\mathrm{C}$. This makes sense, because the item's intangible $i_\times$ is defined as "things that matter but are not already in (i.e. not dependent on) i_C". Secondly, they are conditionally dependent given u or r, namely $i_\times \not\perp\!\!\!\perp i_\mathrm{C} \mid u$ and $i_\times \not\perp\!\!\!\perp i_\mathrm{C} \mid r$. This means when u or r is observed, the value that i_C takes on would bias the probability of the values of $i_\times$. For example, in a simplified world where the item content i_C is "*SciFi or not*", and the item intangible $i_\times$ is "*well made or not*". Given user *Tom* — who is very picky when viewing SciFi — watched *Star Trek* (a SciFi) and really liked it, the probability of *Star Trek* being *well made* has become substantially higher.

6.3 The Content and the Intangible Preferences

Based on the probability dependencies established in section 6.2, this section presents the mathematical relationships between the user's content preferences, intangible preferences, and overall preferences. This forms the theoretical basis of the hybridisation strategy of *Diamond*, which is to be presented in section 6.4.

In the probability space defined in section 5.1, the probability distribution $\mathcal{P}(r|\ u, i)$ represents the likelihood of a rating value (i.e. an integer from 1 to 5) being representative of user u's overall preference on item i. With the introduction of the two new variables i_{C} and $i_{\times}$, there are two additional "preference distributions": the content preference distribution $\mathcal{P}(r|\ i_{\mathrm{C}}, u)$, which represents the proability that user u gives the *represented content* of item i a certain rating; and the intangible preference distribution $\mathcal{P}(r|\ i_{\times}, u)$, which represents the probability that user u gives the *intangible* part of item i a certain rating.

Given certain independence assumptions, the mathematical relationship between the three preference probabilities $\mathcal{P}(r|\ u, i)$, $\mathcal{P}(r|\ i_{\mathrm{C}}, u)$, and $\mathcal{P}(r|\ i_{\times}, u)$ is presented in equation 6.1, the derivation of which is presented in equation 6.2.

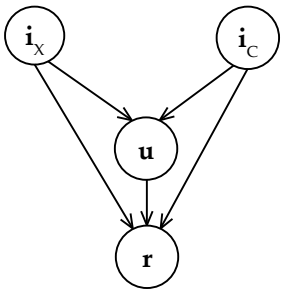
$$\mathcal{P}(r|\ \mathrm{i},\ \mathrm{u}) = \frac{\mathcal{P}(r|\ \mathrm{i_C},\ \mathrm{u})\ \ \mathcal{P}(r|\ \mathrm{i_\times},\ \mathrm{u})}{\mathcal{P}(r|\mathrm{u})} \tag{6.1}$$

$$\begin{aligned}
&\mathcal{P}(r|\ \mathrm{i},\ \mathrm{u}) && && (6.2)\\
&= \mathcal{P}(r|\ \mathrm{i_C},\ \mathrm{i_\times},\ \mathrm{u}) && \textit{The definition of intangible factors} && \text{(a)}\\
&= \frac{\mathcal{P}(\ \mathrm{i_C},\ \mathrm{i_\times}|r,\ \mathrm{u}) \cdot \mathcal{P}(r|\mathrm{u})}{\mathcal{P}(\ \mathrm{i_C},\ \mathrm{i_\times}|\mathrm{u})} && \textit{Bayes' Law} && \text{(b)}\\
&= \frac{\mathcal{P}(\ \mathrm{i_C}|r,\ \mathrm{u}) \cdot \mathcal{P}(\ \mathrm{i_\times}|r,\ \mathrm{u}) \cdot \mathcal{P}(r|\mathrm{u})}{\mathcal{P}(\ \mathrm{i_C}|\mathrm{u}) \cdot \mathcal{P}(\ \mathrm{i_\times}|\mathrm{u})} && \textit{Ind. Asmp. } \mathrm{i_C} \perp\!\!\!\perp \mathrm{i_\times} \mid \mathrm{u}\,[\,, r\,] && \text{(c)}\\
&= \frac{\mathcal{P}(\ \mathrm{i_C}|r,\ \mathrm{u}) \cdot \mathcal{P}(r|\mathrm{u})}{\mathcal{P}(\ \mathrm{i_C}|\mathrm{u})}\ \ \frac{\mathcal{P}(\ \mathrm{i_\times}|r,\ \mathrm{u}) \cdot \mathcal{P}(r|\mathrm{u})}{\mathcal{P}(\ \mathrm{i_\times}|\mathrm{u})}\ \ \frac{1}{\mathcal{P}(r|\mathrm{u})} && \textit{Rearrange} && \text{(d)}\\
&= \frac{\mathcal{P}(r|\ \mathrm{i_C},\ \mathrm{u}) \cdot \mathcal{P}(r|\ \mathrm{i_\times},\ \mathrm{u})}{\mathcal{P}(r|\mathrm{u})} && \textit{Bayes' Law} && \text{(e)}
\end{aligned}$$

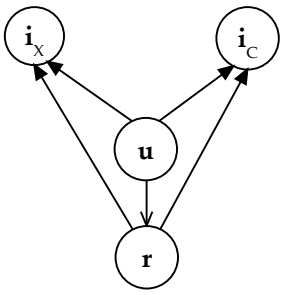
In formula 6.2, line (a) describes the substitution of the item variable i with i_{C} and $i_{\times}$. Line (b) expands the formula using Bayes' Law. Line (c) makes two conditional independence assumptions — the transformation of the denominator assumes $\mathcal{P}(i_{\mathrm{C}}, i_{\times}|\, u) = \mathcal{P}(i_{\mathrm{C}}|\, u) \cdot \mathcal{P}(i_{\times}|\, u)$, or that $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u$ (i_{C} is conditionally independent from $i_{\times}$ given the user); the transformation of the numerator assumes $\mathcal{P}(i_{\mathrm{C}}, i_{\times}|\, u, r) = \mathcal{P}(i_{\mathrm{C}}|\, u, r) \cdot \mathcal{P}(i_{\times}|\, u, r)$, or that $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u, r$ (i_{C} is conditionally independent from $i_{\times}$ given both the user and the rating). Line (d) rearranges the formula by multiplying $\mathcal{P}(r|u)$ to both the numerator and the denominator at the same time. Finally, line (e) simplifies the formula using Bayes' Law.

The Independence Assumptions

The probabilistic correlations of variables u, $i_{\times}$, i_{C}, r are specified in figure 6.5(c), which is repeated here as figure 6.6(a). The derivation of formula 6.1 requires the independence relationships of $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u$ and $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u, r$, which are clearly not satisfied by figure 6.5(c). So to make the derivation, they are *assumed true* as two *naïve* independence assumptions, essentially transferring the statistical correlations captured by figure 6.6(a) into that of figure 6.6(b) (note the directional changes of the edges). Since in this belief net structure, assuming $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u, r$ would automatically enforce $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u$, the rest of the discussion only focuses on $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u, r$.



(a) Probability correlations between variables u, $i_{\times}$, i_{C}, and r; a repeat of figure 6.5(c).



(b) The de facto structure by assuming $i_{\times} \perp\!\!\!\perp i_{\mathrm{C}} \mid u,\ r$ and $i_{\times} \perp\!\!\!\perp i_{\mathrm{C}} \mid u$.

Figure 6.6: The ramification of the *naïve* independence assumptions.

To further clarify what the assumption of $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u, r$ entails, con-

sider a simplified binary world where the user preferences can only be *like* or *dislike*. Intuitively, the user's overall preference on i, his content preference on i_{C}, and his intangible preference on $i_{\times}$ should follow a logical relationship varying from a *logical* `OR` to a *logical* `AND`, with the variability subject to individual users.

To elaborate, since i_{C} and $i_{\times}$ together are supposed to fully capture all aspects of item i, if both i_{C} and $i_{\times}$ are *liked* (or both are *disliked*), then item i as a whole should also be *liked* (or *disliked*), since there is no third factor to explain otherwise. This is displayed as row 1 and 4 in table 6.2. However, the situation becomes uncertain when the user preferences on i_{C} and $i_{\times}$ disagree, as shown in row 2 and 3. In these cases, the logical relationship would be up to individual users — a fastidious user would implement an `AND` strategy, only liking items whose content and intangibles are both good; an easy-going user could implement an `OR` strategy, liking an item so long as something about it is good; there could also be users who would like an item so long as the content is good, and so on.

	Content Preference	Intangible Preference	AND	OR	Overall Preference
1.	Like	Like	Like	Like	Certainly Like
2.	Like	Dislike	Dislike	Like	Uncertainty
3.	Dislike	Like	Dislike	Like	Uncertainty
4.	Dislike	Dislike	Dislike	Dislike	Certainly Dislike

Table 6.2: The intuitive logical relationships between content preference, intangible preference, and overall preference, assuming a binary and crisp world.

To view it in another way, in a simplified world where movie items only have two attributes: *genre* and *resolution*. Suppose *genre* is in the content representation (thus constitutes i_{C}) and *resolution* is not (thus constitutes $i_{\times}$). Suppose Tom is a SciFi fan who likes dazzling scenes (i.e. Tom likes SciFi *or* high resolution). If Tom really likes a Romance movie, chances are that the movie is in Blu-ray. By assuming the conditional independence of $i_{\mathrm{C}} \perp\!\!\!\perp i_{\times} \mid u, r$, the system is effectively saying that "Tom likes this Romance movie" does not say anything about this movie's potential resolution, which clearly is a loss of deductive power.

Compensating for the Independence Assumptions

Based on this understanding, an amendment is made to compensate for the loss of information (thereby recommendation accuracy) caused by the assumption of $i_{\times} \perp\!\!\!\perp i_{\mathrm{C}} \mid u, r$ in the derivation of formula 6.1, which is repeated here:

$$\mathcal{P}(r|\, i, u) = \frac{\mathcal{P}(r|\, i_{\mathrm{C}}, u)\ \ \mathcal{P}(r|\, i_{\times}, u)}{\mathcal{P}(r|\, u)} \tag{6.1}$$

This formula effectively describes the user's overall preference $\mathcal{P}(r|\, u, i)$ as a probabilistic-AND of the user's content preference $\mathcal{P}(r|\, u, i_{\mathrm{C}})$ and his intangible preference $\mathcal{P}(r|\, u, i_{\times})$, normalised by the denominator $\mathcal{P}(r|\, u)$. As explained previously, this is an overly strict restriction. Therefore, this thesis proposes to relax this relationship by gradually blending the probabilistic-AND with a probabilistic-OR. The probabilistic-AND and OR are presented below in figures 6.7 and 6.8 (and equations 6.3 and 6.4).

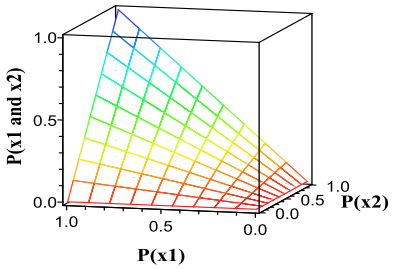


Figure 6.7: The probabilistic-AND of two independent variables x_1 and x_2, calculated using equation 6.3

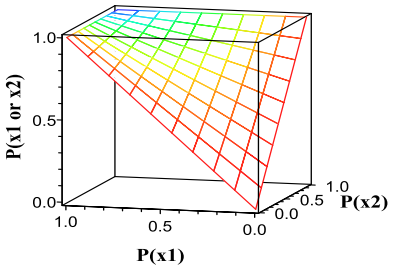


Figure 6.8: The probabilistic-OR of two independent variables x_1 and x_2, calculated using equation 6.4

$$\mathcal{P}(x_1 \cap x_2) = \mathcal{P}(x_1)\,\mathcal{P}(x_2) \tag{6.3}$$

$$\mathcal{P}(x_1 \cup x_2) = \mathcal{P}(x_1) + \mathcal{P}(x_2) - \mathcal{P}(x_1)\,\mathcal{P}(x_2) \tag{6.4}$$

Formally, this thesis defines the blending of a probabilistic-AND and a probabilistic-OR to be a *probabilistic-JOINT*, symbolised as $x_1 \bowtie_a x_2$, where $a \in [0, 1]$ controls the degree of the blend. Its calculation is defined in equation 6.5 and visualised in figure 6.9. The blending mechanism adopted here is linear. As a approaches 0, the probabilistic-JOINT converges to a pure

probabilistic-AND; as a approaches 1, it converges to a pure probabilistic-OR.

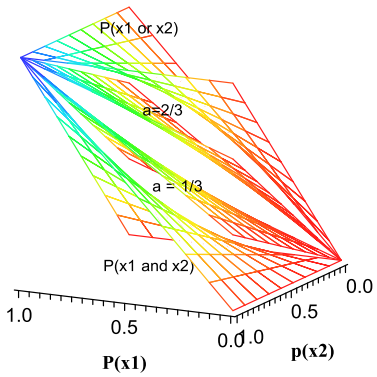


Figure 6.9: Probabilistic-JOINT with varying degrees of a ($a = 0, \frac{1}{3}, \frac{2}{3}, 1$), calculated using equation 6.5:

$$\begin{aligned}\mathcal{P}(x_1 \bowtie_a x_2) &= (1-a)\cdot\mathcal{P}(x_1 \cap x_2) + a\cdot\mathcal{P}(x_1 \cup x_2) \\ &= (1-2a)\mathcal{P}(x_1)\mathcal{P}(x_2) + a(\mathcal{P}(x_1)+\mathcal{P}(x_2))\end{aligned} \tag{6.5}$$

Replacing the *de facto* probabilistic-AND between $\mathcal{P}(r|\, i_{\mathrm{C}}, u)$ and $\mathcal{P}(r|\, i_{\times}, u)$ in equation 6.1 with a probabilistic-JOINT leads to equation 6.6, which is the basis of the hybridisation strategy of *Diamond*.

$$\mathcal{P}(r \mid i, u) = \frac{(1-2a)\cdot\mathcal{P}(r \mid i_{\mathrm{C}}, u)\cdot\mathcal{P}(r \mid i_{\times}, u) + a\cdot\Big(\mathcal{P}(r \mid i_{\mathrm{C}}, u) + \mathcal{P}(r \mid i_{\times}, u)\Big)}{\mathcal{P}(r \mid u)} \tag{6.6}$$

Note that this amendment does not completely recover the information lost by the assumption of $i_{\times} \perp\!\!\!\perp i_{\mathrm{C}} \mid u, r$. Firstly, the probabilistic-JOINT assumes the two variables x_1 and x_2 to be conditionally independent — a condition that the content preference $\mathcal{P}(r|\, i_{\mathrm{C}}, u)$ and the intangible preference $\mathcal{P}(r|\, i_{\times}, u)$ clearly do not meet. Secondly, the relationship between the two preferences and the overall preference can only be expressed as a linear combination of an AND and an OR if i_{C} and $i_{\times}$ cannot be subdivided such that the subcomponents can interact in a complex and combinatorial way. This is not true in general: suppose i_{C} constitutes of two factors: *genre* and *actor*, and $i_{\times}$ constitutes of *resolution* and *soundtrack*, the overall preference of *Tom* is positive if genre is SciFi and resolution is high, or actor is William Shatner and soundtrack is by John Williams, making $\mathcal{P}(r|\, i_{\mathrm{C}}, u)$ and $\mathcal{P}(r|\, i_{\times}, u)$ no longer of a clear-cut relationship. However, through parameter-tweaking, it will be possible to find an optimal α for each specific user which gives the best average approximation. Experiments on the effect of using equation 6.6 with different a values is examined in section 6.5.2.

6.4 The Hybridisation Strategy of *Diamond*

The core of a hybrid filtering system is its *hybridisation strategy*, which is the part that handles the cooperation of the CF and the CBF engines. Conventional hybridisation strategies (surveyed in section 2.5) are either sequential or combinatorial (or not cooperative at all), meaning that CF and CBF are either applied one after another, or parallel to each other, with the final recommendation being a simple combination of the predictions of the two.

The hybridisation strategy of *Diamond* is fundamentally different from conventional approaches. Instead of simply combining the two filterings on an operational level, a divide-and-conquer process on the recommendation task itself is proposed. Firstly, the training data, which represents users' overall preference of the items, is separated into the *content preference* data, which represents users' preference on the content attributes of the items, and the *intangible preference* data, which represents users' preference on the *intangible* factors not covered by the items' content attributes. The recommendation task is thus divided into two sub-tasks — the task of recommending the content preferences, and the task of recommending the intangible preferences.

The two sub-tasks are then handled separately and independently by CF and CBF. Since the sub-tasks are divided in a way that matches the learning biases of the two filtering engines, better recommendation accuracy is expected due to reduced complexity and enhanced clarity of the learning tasks.

The final recommendation output of the system is a mathematically-justified combination of the intangible preference prediction produced by CF, and the content preference prediction produced by CBF. This *dividing* of the task and *conquering* of the results can be visually symbolised as a diamond shape $\diamondsuit$, thus the system name — the *Diamond*.

This section firstly outlines the overall structure of *Diamond* in section 6.4.1, then presents the "divide" mechanism — the recommendation task separator — in section 6.4.2, and the "conquer" mechanism — the recommendation results combinator — in section 6.4.3.

6.4.1 The Overall Architecture of *Diamond*

The *Diamond* hybrid filtering recommender system has four components: 1) a collaborative filtering engine; 2) a content-based filtering engine; 3) a hybridisation mechanism, which coordinates the collaboration of CF and CBF; and 4) a distributional rating transformer, which converts scalar ratings to and from distributional ratings.

Figure 6.10 presents the structure and components of *Diamond*. In this figure, the three oval shapes represent the data inlets and outlets of the system; the four rectangular boxes represent the four components, with the core component — the hybridisation strategy — highlighted in yellow; the annotated arrows connecting the shapes represent the data flow. There are two kinds of data flows. The *training flow* feeds the training data through the system for supervised training. After the system is properly trained, a *recommendation flow* can be issued to use the trained system to make recommendations. The rest of this section illustrates the functionality of each component by going through the training flow and the recommendation flow respectively.

The Training Flow

The training flow of *Diamond* is illustrated in the left half of figure 6.10 (with pink background). The flow starts at the intake of the scalar training data, as represented by the oval shape at the upper-left corner of the figure. Since *Diamond* operates on distributional ratings (presented in chapter 5), before training can start, the scalar ratings in the form of $\{..(u, i, r)..\}$ need to be transformed into distributional ratings in the form of $\{..(u, i, \mathcal{P}(\underset{\sim}{\mathrm{r}} |i, u))..\}$. This is done by the *scalar to distributional rating transformer* part of the *rating transformer*. The distributional ratings are then separated into the *content preference ratings* $\{..(u, i, \mathcal{P}(\underset{\sim}{\mathrm{r}} |i_{\mathrm{C}}, u))..\}$ and the *intangible preference ratings* $\{..(u, i, \mathcal{P}(\underset{\sim}{\mathrm{r}} |i_{\times}, u))..\}$ by the *recommendation task separator* part of the *hybridisation mechanism*. After the separation, the content preference data is used to train the content-based filtering engine, while the intangible preference data is used to train the collaborative filtering engine.

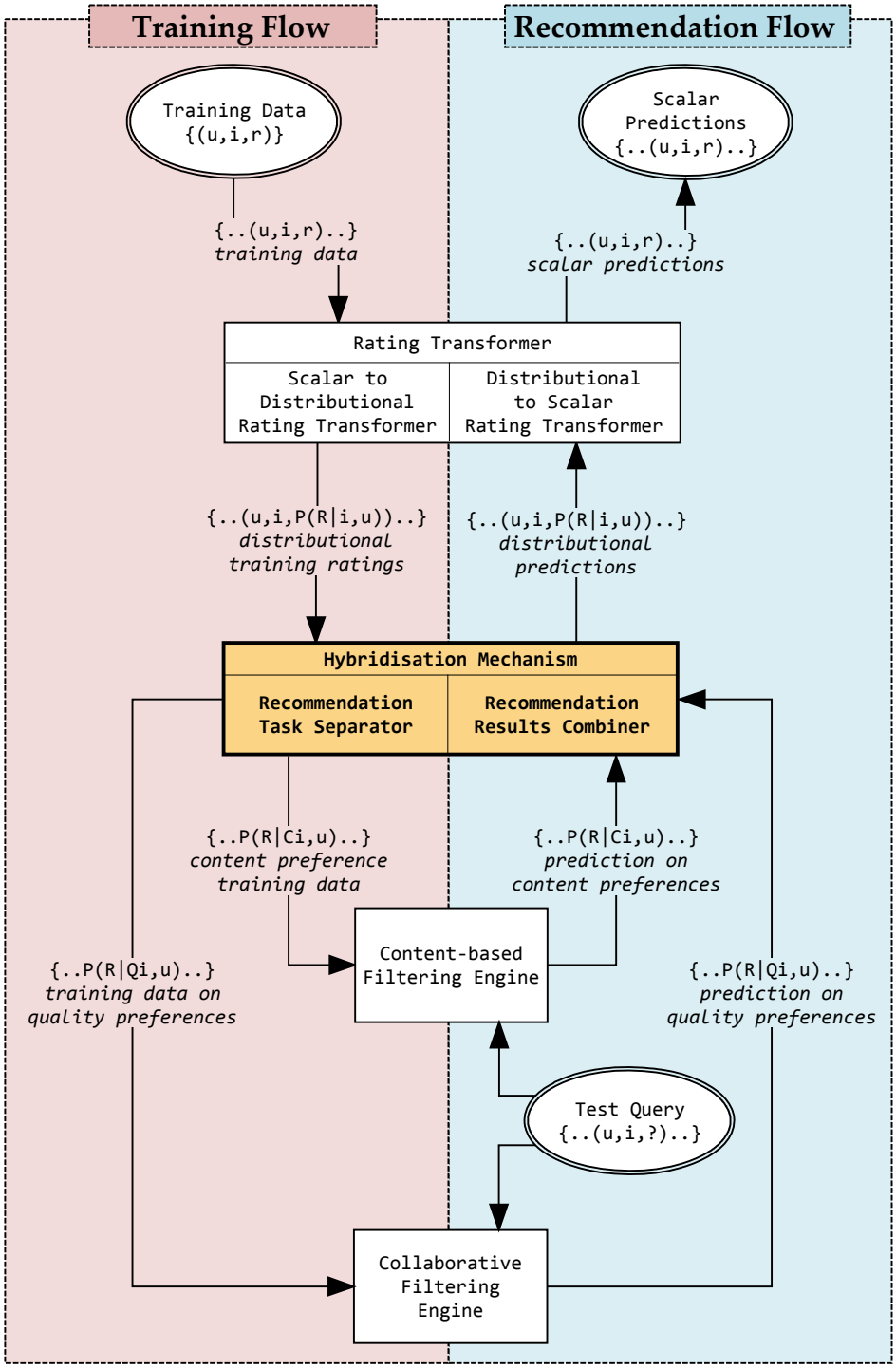


Figure 6.10: The structure and components of the *Diamond* hybrid filtering system. The left hand side (pink) and the right hand side (blue) present the components involvements and data flow in the training of the system and the utilisation of the trained system (i.e. recommendation) respectively.

The Recommendation Flow

The recommendation flow of *Diamond* is illustrated in the right half of figure 6.10 (with blue background). The flow starts at the intake of some recommendation queries represented by the oval shape labelled "test query" at the lower-right conner of the figure. For each query, the CF engine makes a prediction on how much it thinks the user would like the item's *intangibles*; the CBF engine makes a parallel prediction on how much it thinks the user would like the item's *represented content*. The two predictions are then combined together by the *recommendation results combiner* of the *hybridisation mechanism* to form a general prediction, which comes in the form of a distributional rating $\mathcal{P}(\underset{\sim}{\mathrm{r}}\,|i,u)$. This distributional prediction can be directly given back to the user, providing the user with a more informative recommendation, or it can be transformed back to a single scalar prediction through the *distributional to scalar rating transformer* part of the *rating transformer*.

6.4.2 The Recommendation Task Separator

The *recommendation task separator* or RTS separates the learning task into two sub-tasks, each of which fits the learning bias of CF and CBF respectively. The separation of the task is achieved by the separation of the training data $\{\ldots\mathcal{P}(\underset{\sim}{\mathrm{r}}\mid i,u)\ldots\}$ into the training data of users' *intangible preferences* — $\{\ldots\mathcal{P}(\underset{\sim}{\mathrm{r}}\mid i_{\times},u)\ldots\}$, which is to be learnt by collaborative filtering, and the training data of users' *content preferences* — $\{\ldots\mathcal{P}(\underset{\sim}{\mathrm{r}}\mid i_{\mathrm{C}},u)\ldots\}$, which is to be learnt by content-based filtering.

Note that data *separation* is different from data set *partitioning*, as the latter is an inter-rating splitting while the former being an intra-rating splitting. In data separation, the splitting happens within each rating. Each rating is split into two, resulting in two datasets the same size as the original; in data partitioning, the splitting happens for the entire dataset. The dataset is partitioned into two, resulting in two datasets half the size as the original.

6.4.2.1 Obtaining the intangible preference data

Suppose the content preference data is known, the intangible preference data $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\times}, u)$ can be derived from the general distributional ratings $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i, u)$ and the content preference data $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)$ based on equation 6.7, which is a rearrangement of equation 6.6.

$$\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\times}, u) = \frac{\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i, u)\ \ \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u) - a \cdot \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}},\ u)}{(1 - 2a) \cdot \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}},\ u) + a} \tag{6.7}$$

In this formula, a is the *blending factor*, which is a system parameter to be determined by empirical experiments. $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i, u)$ is the distributional rating of user u's preference on item i, which is provided by the training dataset and the distributional rating transformer. $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)$ is the user's distributional voting profile, which is also provided by the training dataset. The only unavailable factor is $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)$ — the user's content preference on item i. Its calculation is presented in the following subsection.

6.4.2.2 Obtaining the content preference data

The computation of content preference $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)$ is closely related to the content representation, which in the context of this thesis is a set of single-valued attributes, such as *movie-name*, *era*, *release-year*, *production company*, *certification* (e.g. PG-13), and so on. Set-valued attributes such as *genre* and *actor* can also be represented by "flattening" their values into a set of single-boolean-valued attributes, such as *genre-being-action*, *genre-being-comedy*, *actor-being-william-shatner*, and *actor-being-brad-pitt*.

Let $\{A_1, \ldots, A_n\}$ be the set of single-valued attributes that constitute the content-representation. The content of item i can thereby be represented as a set of attribute-value pairings, namely $i_{\mathrm{C}} = \{A_1{=}v_1^i, \ldots, A_n{=}v_n^i\}$, or simply $i_{\mathrm{C}} = \{A_1^i, \ldots, A_n^i\}$. For example, if i is the movie *Star Trek* and A_t is the attribute *genre-being-sci-fi*, A_t^i or in this case $A_{\text{sci-fi}}^{\text{Star Trek}}$ would be the attribute-value pairing of *genre-being-sci-fi = True*. $A_{\text{sci-fi}}^{\text{Matrix}}$ and $A_{\text{sci-fi}}^{\text{Aliens}}$ also represent the same pairing, making the three semantically identical.

Under the content representation of $i_{\mathrm{C}} = \{A_1^i, \ldots, A_n^i\}$, the user's content preference $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)$ can be expressed as $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_1^i, \ldots, A_n^i,\ u)$, which can be computed from the user's preferences of the individual attributes — the $\{\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_t^i,\ u)\}$s — as shown in equation 6.8:

$$\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u) \tag{6.8}$$

$$= \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_1^i, \ldots, A_n^i,\ u) \qquad \textit{Single-valued attributes as content-representation}\ \text{(a)}$$

$$= \frac{\mathcal{P}(A_1^i \ldots A_n^i \mid \underset{\sim}{\mathrm{r}}, u)\ \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)}{\mathcal{P}(A_1^i \ldots A_n^i \mid u)} \qquad \textit{Bayes' Law}\ \text{(b)}$$

$$= \frac{\mathcal{P}(A_1^i \mid \underset{\sim}{\mathrm{r}}, u) \ldots \mathcal{P}(A_n^i \mid \underset{\sim}{\mathrm{r}}, u)\ \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)}{\mathcal{P}(A_1^i \mid u) \ldots \mathcal{P}(A_n^i \mid u)} \qquad \textit{Ind. Asmp.}\ A_1 \perp\!\!\!\perp \ldots \perp\!\!\!\perp A_n \mid u\ [, \underset{\sim}{\mathrm{r}}\ ]\ \text{(c)}$$

$$= \frac{\mathcal{P}(A_1^i \mid \underset{\sim}{\mathrm{r}}, u)\ \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)}{\mathcal{P}(A_1^i \mid u)} \cdots \frac{\mathcal{P}(A_n^i \mid \underset{\sim}{\mathrm{r}}, u)\ \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)}{\mathcal{P}(A_n^i \mid u)}\ \frac{1}{\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)^{\ n-1}} \qquad \textit{Rearrange}\ \text{(d)}$$

$$= \frac{\prod_{t=1}^{n}\ \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_t^i, u)}{\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)^{\ n-1}} \qquad \textit{Bayes' Law}\ \text{(e)}$$

The transformation from line (b) to line (c) assumes that the attributes are conditionally independent given the user (i.e. $A_1 \perp\!\!\!\perp \ldots \perp\!\!\!\perp A_n \mid u$), and given both the user and the rating (i.e. $A_1 \perp\!\!\!\perp \ldots \perp\!\!\!\perp A_n \mid u, \underset{\sim}{\mathrm{r}}$).

The *attribute preference* $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_t^i, u)$ in line (e) can be "counted-up" from the original scalar rating data. For example, suppose A_t^i is *genre-being-action = True*, the probability that user Tom gives an *action* movie a rating of x ($\in [1, 5]$) can be calculated according to equation 6.9; the attribute preference $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_{\text{action}} = true, u = Tom)$ would be a vector of size 5 calculated according to equation 6.10:[6]

$$\mathcal{P}(r = x \mid A_{action}\text{=True},\ u\text{=Tom}) = \frac{\text{\# action movies rated an } x \text{ by Tom}}{\text{\# action movies rated by Tom}} \tag{6.9}$$

$$\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A_{action}\text{=True},\ u\text{=Tom}) = \mathcal{S}\Big(\big[\ \mathcal{P}(r{=}1 \mid A, u), \ldots,\ \mathcal{P}(r{=}5 \mid A, u)\ \big]\Big) \tag{6.10}$$

[6]The symbol $\mathcal{S}$ in equation 6.10 represents the *smoothing factor*, which in the implementation of *Diamond* is calculated using the "scalar to distributional rating transformation" mechanism described in section 5.5.1, with the x in $\mathcal{S}(x)$ treated as an index vector.

Figure 6.11 provides an integrated view of the recommendation task separator. The overall processes are as follows: firstly, the user's *attribute preferences* $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid A, u)$ are counted up "vertically" across all items the user has rated; secondly, the user's *content preference* on item i or $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)$ is calculated based on the attribute preferences of the user and the content representation of the item; the user's *intangible preference* $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)$ can then be calculated based on section 6.4.2.1 and formula 6.7.

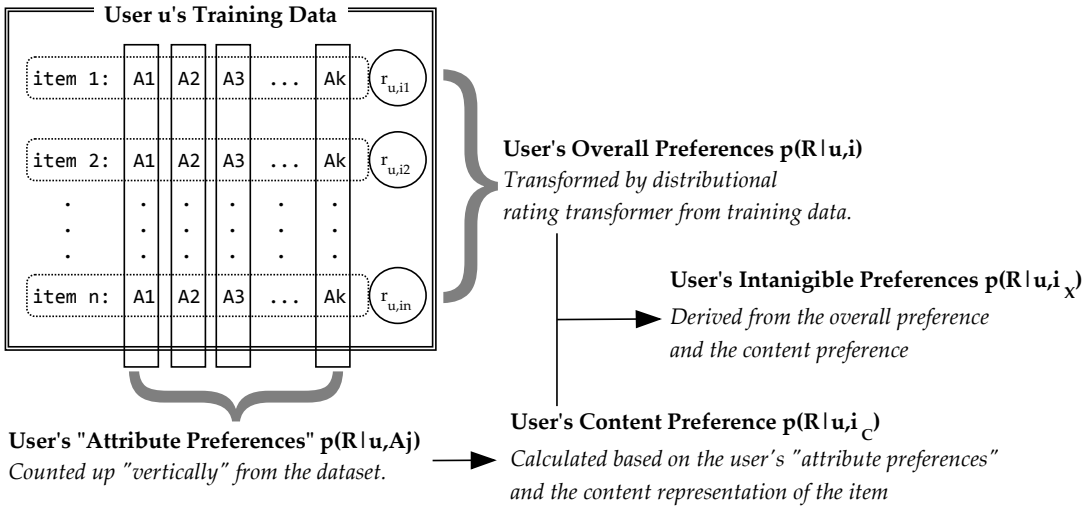


Figure 6.11: An integrated view of the recommendation task separator.

6.4.3 The Recommendation Results Combinator

This section describes the other half of *Diamond*'s hybridisation strategy — the *recommendation results combinator*. The final prediction $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u, i)$ is obtained by combining $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}},\ u)$ and $\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\times},\ u)$ — the prediction made by CBF on user's content preference of the item, and the prediction made by CF on the user's intangible preference of the item. Basically, the predictions are combined based on equation 6.11, which is a simple rewriting of equation 6.6.

$$\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i, u) = \frac{(1-2a)\cdot\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u)\cdot\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\times}, u) + a\cdot\Big(\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\mathrm{C}}, u) + \mathcal{P}(\underset{\sim}{\mathrm{r}} \mid i_{\times}, u)\Big)}{\mathcal{P}(\underset{\sim}{\mathrm{r}} \mid u)} \tag{6.11}$$

6.5 Experimental Analysis

The section evaluates the *Diamond* hybridisation strategy. Section 6.5.1 illustrates the general performance of *Diamond*, and compares it with pure CF and CBF algorithms; section 6.5.2 tries to determine the optimal value for the blending parameter α. The next two sections evaluate *Diamond*'s performance under different dataset properties: section 6.5.3 evaluates the effect of different levels of content availability, and section 6.5.4 evaluates the effect of different rating dataset sparsity. Finally, section 6.5.6 evaluates the hybridisation strategy of *Diamond* by comparing it with other linear or sequential hybridisation strategies with the same base engines.

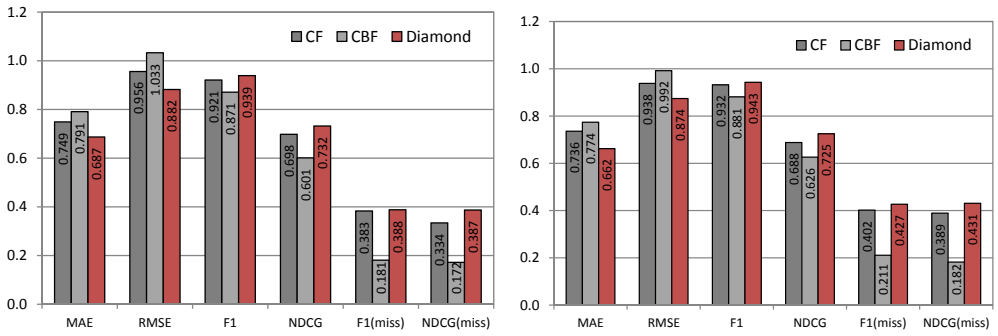
6.5.1 The General Performance

This section presents the general performance of the *Diamond* hybrid filtering system by comparing it with its constituent CF and CBF parts.

In the current implementation of *Diamond*, user-oriented DRNN (presented in chapter 5) is used as the base CF engine; and a cosine similarity-based nearest neighbour approach on the *item content matrix* (ICM) is used as the base CBF engine. The ICM is a matrix whose rows are items, and columns are item attributes. To make a recommendation, the rows of ICM are compared using cosine similarity, based on which item weights are generated, and new CBF predictions can be made as a weighted average of the existing item ratings. This approach is chosen because it can be easily adapted to process distributional ratings, it is widely used as the base CBF engine in other hybrid filtering systems [168, 10], and it performs competitively with more complex CBF algorithms despite its simplicity [110].

Figure 6.12 presents the general performance of *Diamond* on the MLS and MLM datasets. The (distributional rating-based) CF and CBF are marked in different shades of grey; the *Diamond* algorithm is marked in red. The experimental setting and metrics are similar to section 5.7.1: skip-every-10^{th} with 10-fold cross validation is used as the dataset partitioning protocol; a threshold cut-off of 3 is used for prediction to recommendation

conversion with precision, recall, and F1 measure; and $N = 30$ is used for NDCG. The *Diamond* system has an endogenous blending parameter α, which is set to 0.7 based on empirical experiments (presented in section 6.5.2); the setting of content attributes is explained in section 6.5.3.



(a) The MLS Dataset (b) The MLM Dataset

Figure 6.12: The general performance of the *Diamond* hybrid filtering system.

Experimental results show that *Diamond* outperforms both CF and CBF in terms of prediction accuracy (MAE and RMSE), binary recommendation accuracy (F1, F1_{miss}), and ranked recommendation accuracy (NDCG, $\text{NDCG}_{\text{miss}}$). A significant improvement is achieved on the prediction accuracies in particular. Compared to CF, *Diamond* is able to make an 8% improvement on the MLS dataset (i.e. 0.687 vs. 0.749) and a 10% improvement on the MLM dataset (i.e. 0.662 vs. 0.736) on the MAE metric.

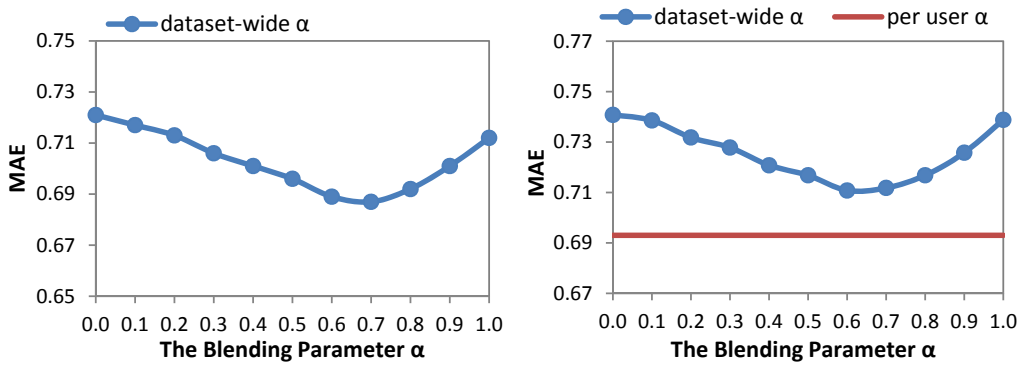
The improvements on the binary and ranked recommendation accuracies are not as prominent. It is interesting, because the CF performance shown here is not the performance of the standard NNCF, but that of the distributional rating-based DRNN. Since DRNN makes a significant improvement over NNCF in terms of the recommendation accuracies but not the prediction accuracies,[7] by transitivity, one can conclude that *Diamond* indeed performs significantly better than standard NNCF under both the recommendation and the prediction accuracy measures, with its distributional rating aspect contributing to the improvement on the recommendation accuraccies, and its hybrid filtering aspect contributing to the improvement on the prediction accuracies.

[7]This result was presented in section 5.7.1 on page 149 under the same experimental settings as this section.

6.5.2 How does *Diamond* respond to the blending parameter α?

This section evaluates the effect of the blending parameter α introduced in formula 6.5. While trying to calculate the mathematical relationships between the overall preference $\mathcal{P}(r|\ i, u)$, the content preference $\mathcal{P}(r|\ i_{\mathrm{C}}, u)$, and the intangible preference $\mathcal{P}(r|\ i_{\times}, u)$, various independence assumptions were made, which led the calculation to suggest that the overall preference is a normalised probabilistic AND of the content and intangible preferences, as shown in equation 6.1. This is overly strict, as argued on page 180. Therefore, the calculation is refined to formula 6.6 by introducing the blending parameter $\alpha \in [0, 1]$. As α approaches 0, the calculation converges to a pure normalised probabilistic AND, as α approaches 1, it converges to a pure normalised probabilistic OR.

Figure 6.13 shows the effect of α on the performance of *Diamond* using the MLS dataset and the MAE evaluation metric. The x-coordinates of the figures correspond to the α values, and the y-coordinates correspond to the MAE. Figure 6.13(a) shows the result of applying the same α setting on the entire dataset. The results suggest that α indeed affects the performance of the system. The optimal α appears to be around $\alpha = 0.7$, which makes the overall preference $\mathcal{P}(r|\ i, u)$ more towards a probabilistic OR of the content preference $\mathcal{P}(r|\ i_{\mathrm{C}}, u)$ and the intangible preference $\mathcal{P}(r|\ i_{\times}, u)$ than a probabilistic AND. This also means that our previous suspicion of "a pure probabilistic AND is too strict" is well-founded.



(a) training-test partitioning (b) training-test-validation partitioning

Figure 6.13: The effect of the blending parameter α.

Intuitively, the relationship between the overall, content, and intangible preferences can be very different from user to user. As presented in table 6.2 on page 179, we can easily imagine a user who likes an item only if both the content and the intangibles (e.g. quality) of this item are good; a more easy-going user could be happy with the item so long as one of the content or the intangibles is good. Therefore, validation set-based empirical experiments are conducted to try to find the optimal α for each user. In this experiment, the dataset is partitioned into three subsets containing 70%, 20%, and 10% of the data using skip-every-n^{th} to serve as the training, validation, and test set respectively. The system is trained on the training set using 11 different α values equal to the x-coordinates of figure 6.13; the trained systems with different α values are compared using the validation set, and the best performing α is chosen for each user; the overall performance of parameter-turned system is then examined on the test set, which forms the final result shown in figure 6.13(b).

In figure 6.13(b), the blue curve represents the MAE of dataset-wide α, and the red line (i.e. its y-coordinate $y = 0.693$) represents the MAE of per-user α. All data points in this figure are generated under the same training and test data.[8] It shows that per-user α indeed performs better than applying the same α on the entire dataset. However, due to the extra burden of parameter tuning, other experiments in this chapter all use $\alpha = 0.7$ on the entire dataset without further specification.

Figure 6.14 plots the distribution of users with different α as their personal optima. It is interesting to see that most users have strong bias towards either the AND side or the OR side. This means that although $\alpha = 0.7$ appears to the optimum of the entire dataset, it is the result of a dataset-wide compromise from everybody, but not that most users have $\alpha = 0.7$ as their personal choice.

[8]Per-user α requires 20% of the data to be used for parameter tuning. This data is simply discarded for experiments with dataset-wide α to keep the data supply of all experiments consistent.

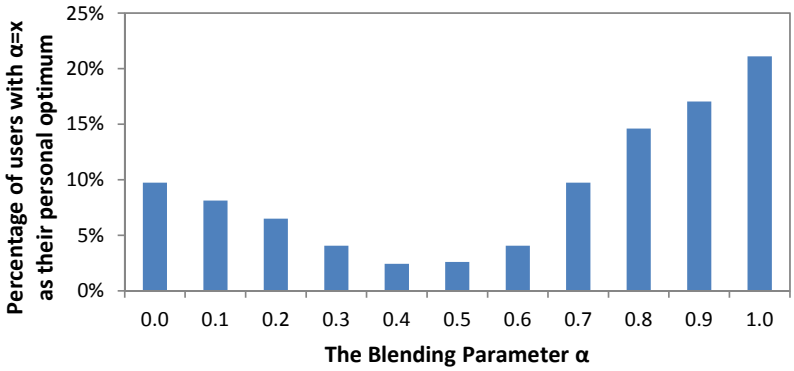


Figure 6.14: The distribution of users' personal optimal α.

6.5.3 How does *Diamond* respond to the extent of content attributes?

As described in section 3.1.2, *Diamond* uses movie attributes provided by IMDb as content data. The set-valued attributes such as genre, actors, and directors are flattened into single valued attributes by converting the attribute values into individual attributes, such as *genre-being-action* or *actor-being-will-smith*. This results in a large set of attributes, some of which carry very little value (such as an unimportant actor who only appeared in one movie). This not only slows down the content-based recommendation process, but also adds complexity to the recommendation problem.

This thesis uses a forward feature selection-based preprocessing step [7] to select a set of n most informative attributes based on their *expected information gain* [116], and only uses these n attributes in the content-based recommendation process.

Suppose $D : \{\langle u, i, r\rangle\}$ is the set of rating data, $A : \{a\}$ is the set of available attributes, and suppose an attribute $a \in A$ can take on one of a set of values $\{a_1, \ldots, a_k\}$. The *information gain* of knowing the value of a on dataset D is represented as $IG(a, D)$, and is defined in formula 6.12:

$$IG(a, D) = \mathcal{H}(D) - \sum_{t=1}^{k} \mathcal{P}(a_t)\mathcal{H}(D|a_t) \tag{6.12}$$

where $\mathcal{H}(D)$ is the entropy of the dataset, which is the same for all a thus can be ignored; $\mathcal{H}(D|a_t)$ is the conditional entropy and is defined as fol-

lows:

$$\mathcal{H}(D|a_t) = -\sum_{r'=1}^{5} \mathcal{P}(D|a_t, r') \cdot \log \mathcal{P}(D|a_t, r')$$

This section investigates the effect of the number of content attributes n on the performance of *Diamond*. Figure 6.15 shows the experimental results of *Diamond* under varying n. The x-axis represents the number of attributes n, and the y-axis is the MAE on the MLS dataset with the corresponding number of attributes used in the content-based filtering process. Note that due to the information gain-based attribute selection that choose the most informative attribute first, a larger set of attributes generally correspond to lower averaged attribute quality.

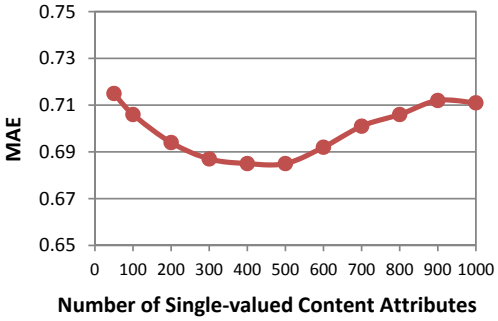


Figure 6.15: The effect of the number of attributes on the MAE accuracy of *Diamond*.

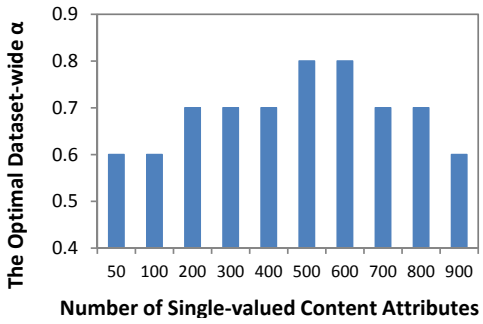


Figure 6.16: The effect of the number of attributes on the choice of the optimal dataset-wide blending parameter α.

The results show that "the more attributes the better" is not true. And for the particular experimental set-ups of this thesis, $n = 300$ seems to be the most optimal choice, since it results in near-best performance, and is the smallest (i.e. simplest and fastest) n value for the curve to start to level off. It is also the setting chosen for other experiments presented in this chapter.

The value of n also affects the optimal value of the dataset-wide blending parameter α. Figure 6.16 shows their correlations. Basically, the optimal value of α seems to be within the range of $[0.6, 0.8]$.

The reason for the declining MAE performance as n gets large could be twofold. Firstly, as argued at the start of this section, as n gets larger, there would be more and more attributes that carry little information, and not

using them reduces data noise and the workload of the system. Secondly, the independence assumptions made in the derivation of *Diamond*'s hybridisation strategy tend to be less true as n — the number of attributes gets bigger. Based on how item attributes are normally collected, the more attributes there are, the bigger the overlapping among the attributes there will be, and the more the attributes correlate with each other.

6.5.4 How does *Diamond* respond to dataset sparsity?

The previous section investigated the effect of the extent of content data on the performance of *Diamond*. This section investigates the effect of the sparsity of collaborative rating data on the performance of *Diamond*.

The general setting of this experiment is similar to that of section 4.5.5. The given-n dataset partitioning protocol is applied to control dataset sparsity, with n being 5, 10, 20, and all. Figure 6.17 shows the performance of CF, CBF, and *Diamond* under these settings. It can be observed that, compared with CF, *Diamond* is much better at handling sparse datasets. This is not surprising, since it can rely on the content data and content-based filtering when there are very few collaborative rating data. Compared with CBF, *Diamond* is affected more by dataset sparsity, as shown by the steeper slope of the *Diamond* curve. However, it recovers relatively quickly and is able to outperform CBF at "Given-20", where the dataset sparsity (0.9889) is still quite high.

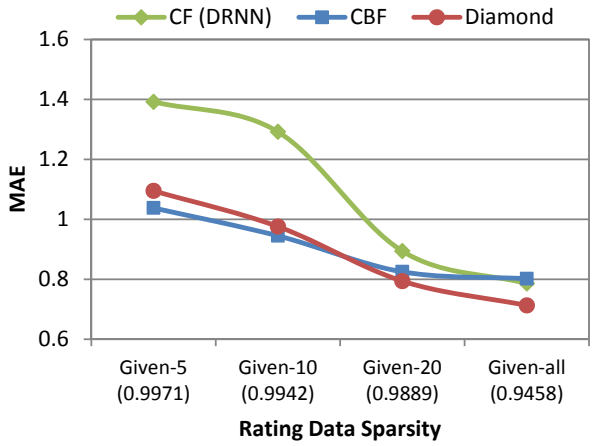


Figure 6.17: The effect of dataset sparsity on the MAE accuracy.

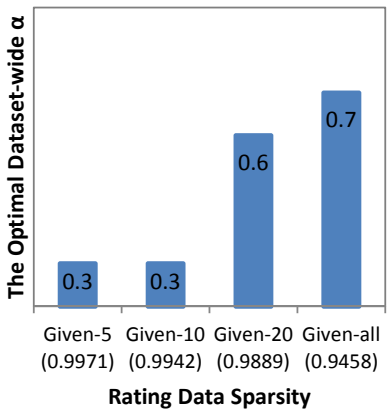


Figure 6.18: Dataset sparsity vs. the choice of α.

Another interesting observation is that the sparsity of the collaborative dataset affects the optimal value of the dataset-wide blending parameter α. Figure 6.18 shows their correlations. Basically, when the dataset is very sparse, a small α is preferred; as the sparsity reduces, the value of α increases to the normal setting of 0.7.

6.5.5 How does *Diamond* compare with other hybridisation strategies?

As described in section 2.5, existing cooperative hybrid filtering systems can be classified as either linear or sequential. This section evaluates *Diamond* against the standard linear and sequential hybrid filtering systems. In order to create a controlled environment and compare only the hybridisation mechanisms, the same CF and CBF algorithms are used as base engines for all hybrid filtering systems compared.

The following three hybridisation strategies are chosen: 1) static linear weight (SLW) which uses a linear hybrid of CF and CBF with equal weight; 2) linear regression (LR), which is another linear strategy that trains CF and CBF separately, then uses linear regression [7] to find the optimal weighted combination of the recommendation outputs of the base CF and CBF algorithms; 3) sequential hybridisation (SH), which applies content-based filtering first to complete the user-item rating matrix, which is then used as training data in collaborative filtering, the recommendation of which constitutes the final recommendation by the hybrid filtering system.

Figure 6.19 presents the experimental results on the MLS dataset with 10-fold cross validation. The CF and CBF methods are marked in different shades of grey; the baseline hybrid filtering algorithms are marked in different shades of blue; and *Diamond* is marked in red. The results show that although the linear and sequential hybrid filtering systems are able to make good improvements over the base CF and CBF engines, *Diamond* is able to outperform all of them, making an even bigger improvement under all evaluation metrics.

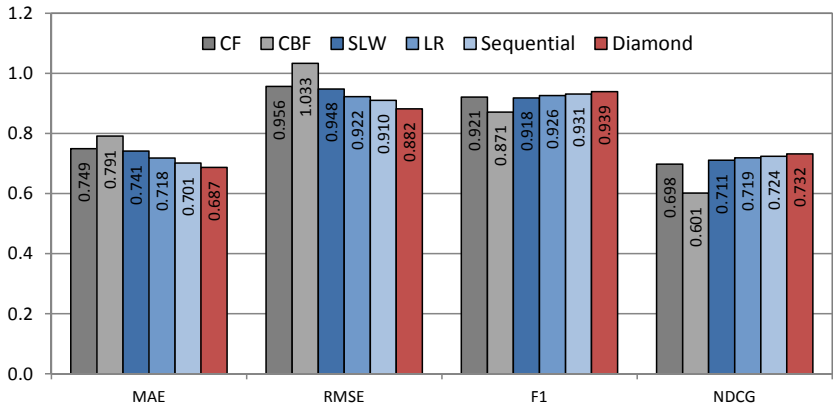


Figure 6.19: Comparison of *Diamond* with other linear and sequential hybridisation strategies using the same base (CF and CBF) engines.

6.5.6 An integrated comparison of all algorithms proposed in this thesis

This section provides an integrated comparison of all of the algorithms proposed in this thesis.

Since the experimental settings used in this thesis consistently follow the guidelines drawn in chapter 3, many results presented in different chapters are actually comparable. Figure 6.20 provides an integrated view of the standard NNCF and all five algorithms proposed in this thesis (TASK, PA, DA, DRNN, and *Diamond*) under the MLS dataset and the user-oriented implementations. The data presented here is a combination of the results from table 4.5 (for TASK), figure 4.14(a) (for PANDA), table 5.2 (for DRNN), and figure 6.12(a) (for *Diamond*).

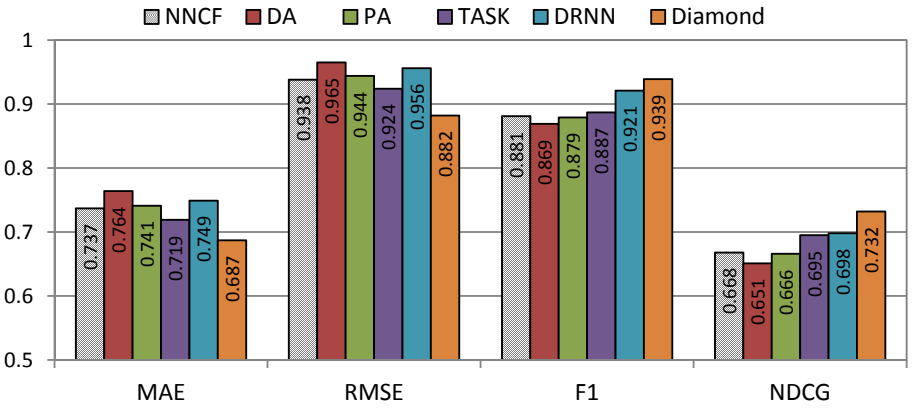


Figure 6.20: Integrated comparison of all the algorithms proposed in this thesis.

The results show that *Diamond* consistently outperforms all other algorithms under all evaluation metrics. Among the collaborative filtering methods, TASK outperforms the standard NNCF under all metrics; DRNN performs worse under the predictive accuracy metrics (MAE and RMSE), but makes a larger improvement under the binary and ranked recommendation metrics comparing to TASK.

6.6 Conclusions

This chapter presents a new hybrid filtering system — *Diamond*, which has a unique diamond-shaped hybridisation structure and a mathematically deduced hybridisation mechanism, which is based on a unique insight regarding the underlying meaning of "rating".

Section 6.1 lays down the conceptual underpinning of *Diamond* by arguing that rating data reflects two components — the user's preference on the item's represented content (the content preference), and the user's preference on the other intangible parts of the item (the intangible preference). Content-based filtering algorithms are content-oriented and highly structural, thus are unfit to model the intangible preferences, which by definition means "not in the content representation"; on the other hand, collaborative filtering lacks important building blocks provided by the item content data, thus is severely disadvantaged at modelling the content preferences.

As described in section 2.5, existing hybrid filtering systems fall into three categories: linear, sequential, and non-cooperative, all of which train both their CF and the CBF base engines on both content and intangible preferences (they simply do not distinguish the two aspects). *Diamond* takes notice of the different aspects, actively separating them, and using CBF to model only the content preferences, and CF to model only the intangible preferences.

The key of *Diamond* is a probability-based mechanism that splits each training data into a content preference training data and an intangible preference training data, in turn splitting each recommendation task into two subtasks that fit the learning bias of CBF and CF respectively. This

mechanism and its derivation is presented in sections 6.2, 6.3, and 6.4. The empirical evaluations presented in section 6.5 showed that *Diamond* outperforms both collaborative and content-based filtering, as well as standard linear and sequential hybrid filtering structures with the same base (CF and CBF) engines.

Future work on *Diamond* include the following directions: 1) to further evaluate the idea of content-intangible separation by testing it under a wide variety of base CF and CBF engines, such as aspect models, PLSA, and probabilistic clustering; 2) to examine the possibility of combining *Diamond* with other linear or sequential hybridisation mechanisms, forming a "hybrid within a hybrid"; 3) to refine the hybridisation strategy of *Diamond* by further studying the effects of the independence assumptions made during the derivation process, and potentially compensate for them; 4) to explore the more general application of "divide of each individual task and conquer of the results", which I believe is not only limited to hybrid filtering, but can also be applied to general ensemble learning tasks where the data points possess unique and separable aspects.

Chapter 7

Conclusions

Recommender systems are rapidly gaining attraction in this information-flushed century. From a research perspective, it provides an interesting and unique machine learning environment due to the specific semantics of recommendation and the extra dimension introduced by the *personalised* aspect of the problem, allowing the exploitation of domain-specific representations, adaptations, and heuristics.

The goal of this thesis is to provide a general understanding of the current recommender algorithms, spot weaknesses, and discover through data analysis and logical reasoning new heuristics that can improve existing solutions, and new representations that is more "tailor-made" to the target domain. Specifically, the thesis focuses on achieving these goals through the exploitation of already-existing but latent information in the recommendation process.

The thesis has achieved its overall goal. In terms of understanding the field, it presented a detailed literature review, including four surveys on recommender system taxonomy, recommendation algorithms, comparison of collaborative and content-based filtering, and recommender system evaluation.

In terms of improving existing recommender algorithms, the thesis follows a "problem identification $\rightarrow$ solution proposition $\rightarrow$ algorithm evaluation" pipeline, and presented four improvements in the format of two algorithms (TASK and PANDA), one process (DRNN), and one frame-

work (*Diamond*) around three different yet correlated directions — nearest neighbour collaborative filtering, distributional rating-based collaborative filtering, and distribution rating-based hybrid filtering.

The four improvements are all based on identifying and utilising previously hidden, neglected, or "tangled" information. Specifically, TASK and PANDA incorporate the target user of the prediction task — an important piece of information previously hidden from the computation — into the similarity computation and rating deviation computation respectively. DRNN explicitly models the *distribution* of users' rating inputs, voting habits, and user similarities — concepts previously only modelled by a scalar number with their distributional-aspects neglected, and consequently enables the production of more informative distributional predictions and better set- and list-recommendation accuracies. The *Diamond* hybrid filtering system untangles the previously tangled concepts of the "tangibles" and the "intangibles", thus reducing the workload of individual collaborative and content-based filtering engines, consequently improving their combined recommendation performance.

The rest of this chapter concludes this thesis by firstly going through in section 7.1 the important concepts and major conclusions drawn in each chapter, and then outlining future work in section 7.2.

7.1 Chapter Summary and Conclusions

Chapter 2 of this thesis firstly presents six classifications of recommender systems in order to provide a better perspective of where this thesis lies. It then zooms into the most important classification — collaborative filtering (CF) and content-based filtering (CBF), and provides a detailed survey on the standard algorithms in the respective class, as well as a comprehensive comparison of the two classes. It elaborates that in general, in terms of data requirements, CF is more sensitive to the sparsity of the overall dataset, yet CBF is more sensitive to the amount of ratings from the target user, and requires additional content data on top of rating data. In terms of recommendation ability, based on current technologies, CF is more accurate than CBF in general situations, new user situations, and cross-domain sit-

uations; whereas CBF wins out in new-item situations and non-transitive association scenarios. In terms of recommendation tendencies, CBF is able to but is also limited to recommending items with similar contents. In terms of machine learning properties, CBF generally has better scalability, adaptivity, stability, and explainability.[1] Subsequently, the chapter outlines modern approaches to hybrid filtering — the combination of the two classes, and points out that modern hybridisation strategies are either linear or sequential, laying down the ground for the *Diamond* hybrid filtering system in chapter 6.

Chapter 3 describes evaluation settings of the thesis, including datasets, experimental protocols, and evaluation metrics, which are classified into prediction accuracy metrics, binary recommendation metrics, and ranked list recommendation metrics. The predictive accuracy metrics are quite similar, with RMSE slightly more sensitive to the highly out-of-whack predictions than MAE. Binary accuracy metrics include precision, recall, GROC, CROC, and so on, and are generally similar yet complementary. Ranked list accuracy metrics such as ARHR, ERU, and NDCG are mostly different in terms of the decay rate they assume against the rankings. Both binary and ranked metrics require adaptations in order to be applied to evaluating rating-based recommender algorithms. The different mechanisms of the adaptations are outlined in the chapter. Other than evaluation metrics, another important concept in this chapter is the identification of the systematic testing of different types of dataset sparsity, which include the similarity computation sparsity and the rating prediction sparsity.

Chapter 4 answers the first research question proposed in section 1.3 with two new algorithms — TASK and PANDA. The chapter firstly presents the nearest neighbour-based collaborative filtering method, and identifies three problems with it — item irrelevance, preference imbalance, and biased average. It also identifies the cause of the problems as the target user of the prediction being hidden from the similarity computation and the

[1] The summary here assumes the most general case, thus are subject to "grouping bias" and may not suit individual cases. For example, although CBF is more scalable than CF in general, there are many CF algorithms that are more scalable than some CBF algorithm. Section 2.4 discusses the detail.

rating deviation computation processes. Two new algorithms — TASK and PANDA — are subsequently proposed to solve these problems by utilising the hidden information. Experiments show that TASK consistently outperforms standard NNCF and other rival algorithms in general situations under both user- and item-oriented settings. However, it does not show strong performance under *extreme* dataset sparsity above 0.994, which is expected, because the extra "digging" requires a reasonable amount of data to start to take effect. TASK is also more susceptible to similarity computation sparsity than rating prediction sparsity. PANDA's performance is not as prominent as TASK. On its own, it fails to make a performance improvement over NNCF, with reasons analysed in the chapter. However, by carefully tuning the application preconditions through a piecewise application or by applying ensemble learning, performance improvements can be achieved. During the analysis of PANDA's initial failure, the importance of the range and the distribution of ratings manifested, and eventually led to the development of the "distributional" ideas presented in the rest of this thesis.

Chapter 5 answers the second research question proposed in section 1.3 and presents the *Distributional Rating-based Nearest Neighbour* or DRNN recommendation process. The chapter firstly establishes the "ground rules" by formalising the probability space for collaborative filtering recommendation. It then identifies the discrete probability representation as one of the best ways of modelling rating distributions due to its efficiency and expressiveness trade-off, and applies it to model rating inputs, users' voting habits, user similarities, and rating predictions. Compared to the scalar model, the distributional model allows the preservation, representation, and communication of a much wider set of concepts — distributional rating inputs allow the representation of potential noise patterns in user inputs; distributional voting profiles allow a more complete portrayal of users' voting patterns, such as multiple peaks and simplified mental scales; distributional similarities provide more information carriers in the recommendation process, and enable the communication of concepts such as "*Users A and B agree on items they both like, but disagree on what each of them dislikes*"; distributional rating predictions allow the communication of the

uncertainty and potential ambivalence in the predictions. Experiments show that the DRNN is able to improve the recommendation accuracy of the system, and performs especially well under difficulty situations such as multi-peaked users or users with complex mental scales.

Chapter 6 answers the third research question proposed in section 1.3 and presents the *Diamond* hybrid filtering system. The chapter starts by arguing that a numerical rating can be construed as the user's preference on 1) the item's content representation and 2) the item's "intangible factors", which include a) missing item content and b) other innately unmeasurable factors. It then argues that the reason of CBF's generally worse performance (compared to CF) is not due to the composition of the CBF algorithms, but the incompleteness of the content representation that CBF operates on. If the recommendation problem can be reduced to the extent where the content representation is reasonably complete, CBF should be able to provide much better support to CF in a hybrid filtering setting. Based on this, it presents *Diamond*, which is a distributional rating-based hybrid filtering system, where the hybridisation strategy is neither linear nor sequential, but of a mathematically sound divide-and-conquer (i.e. diamond) shape. The "divide" stage firstly splits every recommendation task into a content-based subtask and a collaborative subtask, which the CBF engine and the CF engine can operate on separately to produce recommendations that correspond to the user's content preferences and the user's intangible preferences respectively. The "conquer" stage then combines the predictions on the two preferences into one final recommendation. *Diamond* is evaluated against the linear and sequential hybridisation structures, and outperforms both of them in terms of all evaluation metrics, but especially on the binary and ranked list-based metrics.

7.2 Future Work

Future work on TASK and PANDA

One future direction with TASK is to combine it with other user-item combining heuristics to improve its performance under extreme dataset spar-

sity such as in cold-start situations. As pointed out in section 4.4, TASK is radically different from other algorithms that also combine user- and item-oriented predictions. Instead of producing more "averaged" predictions and makes more improvements under extreme sparsity, TASK produces more targeted predictions and makes more improvements under regular situations. It would be interesting to see the combining effects of TASK and these sparsity-targeting algorithms.

Although TASK can be considered as an algorithm that combines the user and the item orientations, it is still orientation sensitive — a user-oriented TASK is different from an item-oriented TASK. Therefore, another direction is to try to combine user- and item-oriented TASK to try to improve the general performance. The same extension can be performed on PANDA.

Another direction is to further study the effect and extent of *preference imbalance* in recommender systems — a problem that is only touched superficially in this thesis as a parasite problem to item irrelevancy.

Future work on DRNN and *Diamond*

The most fundamental and one of the most important future work of DRNN is dataset building. In order to fully study distributional rating, it would be extremely beneficial to have a truly distributional dataset, where the ratings are distributions based on multiple solicitations of the same rating from real users under different environmental and temporal conditions. However, this will be very costly and slow, because it involves building a public interface and more importantly, finding or attracting willing participants.

Another important work is to properly evaluate through user studies the effect of distributional rating predictions on user experiences. Due to resource limitations, this thesis evaluates the performance of distributional predictions by converting it back into scalar predictions, which is a big compromise. With the "distributional dataset" proposed in the previous paragraph, distributional predictions will be able to be better evaluated, but it would still be really interesting to hear what the end users

have to say about the "*I think you will either extremely love or extremely hate this movie, but I cannot tell which*" type recommendations.

I believe there is great power to be harnessed from the extended ecosystem provided by distributional rating, and *Diamond* is only one example. One future work along this line is to explore the use of distributional rating and discrete vector representation in other recommendation algorithms, such as clustering, aspect models, latent semantic models, and so on.

There is also more to be done along the lines of content-intangible separation. It could be very interesting to consider other mechanisms or other directions completely different from *Diamond* that can separate user's content-based preference and intangible preference.

I would like to end this thesis by a future direction that I always find interesting — *asymmetrical similarity*. When computing the similarity between two users, the actual question being asked is that, if one wants to make predictions on user u_2, how much can I base the predictions on user u_1. Note that this does not have to be reciprocal — if I think u_1 is a really good basis to predict u_2, u_2 does *not* have to be an equally good basis to predict u_1. It would be very interesting to pursuit this insight, design an algorithm that enables the flexibility of asymmetrical similarity, and study its implications.

Bibliography

[1] ABBASSI, Z., AMER-YAHIA, S., LAKSHMANAN, L. V., VASSILVITSKII, S., AND YU, C. Getting recommender systems to think outside the box. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 285–288.

[2] ADOMAVICIUS, G., AND TUZHILIN, A. 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 (June 2005), 734–749.

[3] ADOMAVICIUS, G., SANKARANARAYANAN, R., SEN, S., AND TUZHILIN, A. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst. 23*, 1 (2005), 103–145.

[4] AGGARWAL, C. C., WOLF, J. L., WU, K.-L., AND YU, P. S. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of the ACM Knowledge Discovery and Data Mining Conference (KDD-99).* (San Diego, CA, 1999), pp. 201–212.

[5] AHMAD WASFI, A. M. Collecting user access patterns for building user profiles and collaborative filtering. In *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces* (New York, NY, USA, 1999), ACM, pp. 57–64.

[6] ALBERS, B. Collaborative filtering rs and new approaches for rs. Work In Progress, 2005.

[7] ALPAYDIN, E. *Introduction to Machine Learning*. The MIT Press, 2004.

[8] AMATRIAIN, X., LATHIA, N., PUJOL, J. M., KWAK, H., AND OLIVER, N. The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2009), ACM, pp. 532–539.

[9] BALABANOVIC, M., AND SHOHAM, Y. Fab: content-based, collaborative recommendation. *Communications of the ACM 40*, 3 (March 1997), 0. System Fab– a hybrid system.

[10] BAO, X., BERGMAN, L., AND THOMPSON, R. Stacking recommendation engines with additional meta-features. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 109–116.

[11] BASILICO, J., AND HOFMANN, T. Unifying collaborative and content-based filtering. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning* (New York, NY, USA, 2004), ACM, p. 9.

[12] BASU, C., HIRSH, H., AND COHEN, W. Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence* (July 1998), pp. 714–720.

[13] BELL, R. M., AND KOREN, Y. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. *icdm 0* (2007), 43–52.

[14] BELL, R. M., KOREN, Y., AND VOLINSKY, C. The bellkor solution to the netflix prize. Tech. rep., AT&T, 2007.

[15] BERKOVSKY, S., KUFLIK, T., AND RICCI, F. Distributed collaborative filtering with domain specialization. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems* (New York, NY, USA, 2007), ACM, pp. 33–40.

[16] BILLSUS, D., AND PAZZANI, M. A hybrid user model for news story classification. In *Proceedings of the Seventh International Conference on User Modeling* (Banff, Canada, June 1999), Springer-Verlag, pp. 99–108.

[17] BILLSUS, D., AND PAZZANI, M. J. Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning* (San Francisco, CA, 1998), Morgan Kaufmann, pp. 46–54.

[18] BILLSUS, D., AND PAZZANI, M. J. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction 10*, 2-3 (2000), 147–180.

[19] BILLSUS, D., PAZZANI, M. J., AND CHEN, J. A learning agent for wireless news access. In *IUI '00: Proceedings of the 5th international conference on Intelligent user interfaces* (New York, NY, USA, 2000), ACM, pp. 33–36.

[20] BOGERS, T., AND VAN DEN BOSCH, A. Recommending scientific articles using citeulike. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 287–290.

[21] BOLLACKER, K. D., LAWRENCE, S., AND GILES, C. L. A system for automatic personalized tracking of scientific literature on the web. In *DL '99: Proceedings of the fourth ACM conference on Digital libraries* (New York, NY, USA, 1999), ACM, pp. 105–113.

[22] BONHARD, P., HARRIES, C., MCCARTHY, J., AND SASSE, M. A. Accounting for taste: using profile similarity to improve recommender systems. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1057–1066.

[23] BREESE, J., HECKERMAN, D., AND KADIE, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the*

Fourteenth Conference on Uncertainty in Artificial Intelligence (Madison, WI, July 1998), Morgan Kaufmann.

[24] BRIN, S., AND PAGE, L. The anatomy of a large-scale hypertextual web search engine. *Computer Networks 30*, 1-7 (1998), 107–117.

[25] BURKE, R. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction 12*, 4 (November 2002), 331–370.

[26] BURKE, R. Hybrid web recommender systems. *The Adaptive Web LNCS 4321* (2007), 377–408.

[27] CHEN, Q., AND NORCIO, A. F. Modeling a user's domain knowledge with neural networks. *International Journal of Human-Computer Interaction 9*, 1 (1997), 25–40.

[28] CLAYPOOL, M., GOKHALE, A., MIRANDA, T., MURNIKOV, P., NETES, D., AND SARTIN, M. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM-SIGIR Workshop on Recommender Systems: Algorithms and Evaluation* (August 1999). System Tango — a hybrid system for recommending online newspapers.

[29] CONNOR, M., AND HERLOCKER, J. Clustering items for collaborative filtering. In *ACM-SIGIR workshop on recommender systems* (1999).

[30] COSLEY, D., LAM, S. K., ALBERT, I., KONSTAN, J. A., AND RIEDL, J. Is seeing believing?: how recommender system interfaces affect users' opinions. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2003), ACM, pp. 585–592.

[31] CREMONESI, P., TURRIN, R., AND AIROLDI, F. Hybrid algorithms for recommending new items. In *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems* (New York, NY, USA, 2011), HetRec '11, ACM, pp. 33–40.

[32] DELGADO, J., ISHII, N., AND URA, T. Content-based collaborative information filtering: Actively learning to classify and recommend documents. In *CIA '98: Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet* (London, UK, 1998), Springer-Verlag, pp. 206–215.

[33] DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res. 7* (2006), 1–30.

[34] DESHPANDE, M., AND KARYPIS, G. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst. 22*, 1 (2004), 143–177.

[35] DING, S., ZHAO, S., YUAN, Q., ZHANG, X., FU, R., AND BERGMAN, L. Boosting collaborative filtering based on statistical prediction errors. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 3–10.

[36] EGAN, J. *Signal Detection Theory and ROC analysis*. Academic Press, New York, 1975.

[37] FREUND, Y., AND SCHAPIRE, R. E. Experiments with a new boosting algorithm. In *ICML* (1996), pp. 148–156.

[38] GOLDBERG, K., ROEDER, T., GUPTA, D., AND PERKINS, C. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval 4*, 2 (2001), 133–151.

[39] GOOD, N., SCHAFER, J. B., KONSTAN, J. A., BORCHERS, A., SARWAR, B. M., HERLOCKER, J. L., AND RIEDL, J. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the 16th National Conference on Artificial Intelligence* (1999), pp. 439–446.

[40] GUNAWARDANA, A., AND MEEK, C. Tied boltzmann machines for cold start recommendations. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 19–26.

[41] GUNAWARDANA, A., AND SHANI, G. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research 10 10* (2009), 2935–2962.

[42] GUPTA, D., DIGIOVANNI, M., NARITA, H., AND GOLDBERG, K. Jester 2.0: A new linear-time collaborative filtering algorithm applied to jokes. In *ACM-SIGIR Workshop on Recommender Systems: Algorithms and Evaluation* (1999).

[43] HA, V., AND HADDAWY, P. Toward case-based preference elicitation: Similarity measures on preference structures. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (San Francisco, 1998), Morgan Kaufmann Publishers, pp. 193–201.

[44] HAND, D. J., AND TILL, R. J. A simple generalisation of the area under the roc curve for multiple class classification problems. *Mach. Learn. 45*, 2 (2001), 171–186.

[45] HARPALE, A. S., AND YANG, Y. Personalized active learning for collaborative filtering. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2008), ACM, pp. 91–98.

[46] HECKERMAN, D., CHICKERING, D. M., MEEK, C., ROUNTHWAITE, R., AND KADIE, C. M. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research 1* (2000), 49–75.

[47] HERLOCKER, J. L., KONSTAN, J. A., BORCHERS, A., AND RIEDL, J. An algorithmic framework for performing collaborative filtering. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 1999), ACM, pp. 230–237.

[48] HERLOCKER, J. L., KONSTAN, J. A., TERVEEN, L. G., AND RIEDL, J. T. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst. 22*, 1 (2004), 5–53.

[49] HERLOCKER, J. L. *Understanding and Improving Automated Collaborative Filtering Systems*. PhD thesis, University of Minnesota, September 2000.

[50] HILL, W., STEAD, L., ROSENSTEIN, M., AND FURNAS, G. Recommending and evaluating choices in a virtual community of use. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1995), ACM Press/Addison-Wesley Publishing Co., pp. 194–201.

[51] HOFMANN, T. Collaborative filtering via gaussian probabilistic latent semantic analysis. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (New York, NY, USA, 2003), ACM, pp. 259–266.

[52] HOFMANN, T. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst. 22*, 1 (2004), 89–115.

[53] HOFMANN, T., AND PUZICHA, J. Latent class models for collaborative filtering. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1999), Morgan Kaufmann Publishers Inc., pp. 688–693.

[54] HUANG, Z., CHEN, H., AND ZENG, D. Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst. 22*, 1 (2004), 116–142.

[55] JÄRVELIN, K., AND KEKÄLÄINEN, J. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst. 20*, 4 (2002), 422–446.

[56] JENNINGS, A., HIGICHI, H., AND LIU, H. A user model neural network for a personal news service. *Australian Telecommunication Research 27*, 1 (1993), 112.

[57] JIAN, C., JIN, H., AND HUAQING, M. Easy recommendation based on probability model. In *SKG '08: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 441–444.

[58] JIN, R., SI, L., AND CALLAN, J. Collaborative filtering with decoupled models for preferences and ratings. In *Proceedings of the 12th International Conference of Information and Knowledge Management (CIKM 2003)* (Nov 2003).

[59] JIN, R., CHAI, J. Y., AND SI, L. An automatic weighting scheme for collaborative filtering. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2004), ACM, pp. 337–344.

[60] JIN, R., SI, L., AND ZHAI, C. A study of mixture models for collaborative filtering. *Inf. Retr. 9*, 3 (2006), 357–382.

[61] KARYPIS, G. Evaluation of item-based top-n recommendation algorithms. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management* (New York, NY, USA, 2001), ACM, pp. 247–254.

[62] KAWAMAE, N. Serendipitous recommendations via innovators. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2010), ACM, pp. 218–225.

[63] KIM, B. M., AND LI, Q. Probabilistic model estimation for collaborative filtering based on items attributes. In *WI '04: Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 185–191.

[64] KIM, B. M., LI, Q., PARK, C. S., KIM, S. G., AND KIM, J. Y. A new approach for combining content-based and collaborative filters. *Journal of intelligent information systems 27*, 1 (2006), 79–91.

[65] KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conferences on Artificial Intelligence (IJCAI)* (1995), Morgan Kaufmann, pp. 1137–1143.

[66] KONSTAN, J. A., MILLER, B. N., MALTZ, D., HERLOCKER, J. L., GORDON, L. R., AND RIEDL, J. Grouplens: Applying collaborative

filtering to Usenet news. *Communications of the ACM 40*, 3 (1997), 77–87.

[67] KONSTAN, J. A., MCNEE, S. M., ZIEGLER, C.-N., TORRES, R., KAPOOR, N., AND RIEDL, J. T. Lessons on applying automated recommender systems to information-seeking tasks. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence* (2006), AAAI Press, pp. 1630–1633.

[68] KOREN, Y. Tutorial on recent progress in collaborative filtering. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 333–334.

[69] KOWALSKI, G. *Information Retrieval Systems: Theory and Implementation*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[70] KWON, Y. Improving top-n recommendation techniques using rating variance. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 307–310.

[71] LATHIA, N., HAILES, S., AND CAPRA, L. Private distributed collaborative filtering using estimated concordance measures. In *Proceedings of the 2007 ACM conference on Recommender systems* (New York, NY, USA, 2007), RecSys '07, ACM, pp. 1–8.

[72] LATHIA, N., HAILES, S., CAPRA, L., AND AMATRIAIN, X. Temporal diversity in recommender systems. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2010), ACM, pp. 210–217.

[73] LEE, W. S. Collaborative learning for recommender systems. In *Proc. 18th International Conf. on Machine Learning* (2001), Morgan Kaufmann, San Francisco, CA, pp. 314–321.

[74] LEINO, J., AND RÄIHÄ, K.-J. Case amazon: ratings and reviews as part of recommendations. In *RecSys '07: Proceedings of the 2007*

ACM conference on Recommender systems (New York, NY, USA, 2007), ACM, pp. 137–140.

[75] LI, M., DIAS, B. M., JARMAN, I., EL-DEREDY, W., AND LISBOA, P. J. Grocery shopping recommendations based on basket-sensitive random walk. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2009), ACM, pp. 1215–1224.

[76] LI, Q., AND KIM, B. M. Clustering approach for hybrid recommender system. In *WI '03: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence* (Washington, DC, USA, 2003), IEEE Computer Society, p. 33.

[77] LI, Q., KIM, B. M., GUAN, D. H., AND OH, D. W. A music recommender based on audio features. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2004), ACM, pp. 532–533.

[78] LI, Q., KIM, B. M., AND MYAENG, S. H. Clustering for probabilistic model estimation for cf. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM, pp. 1104–1105.

[79] LI, Q., MYAENG, S. H., AND KIM, B. M. A probabilistic music recommender considering user opinions and audio features. *Information Processing and Management 43*, 2 (2007), 473–487.

[80] LI, Y., LU, L., AND XUEFENG, L. A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in e-commerce. *Expert Syst. Appl. 28*, 1 (2005), 67–77.

[81] LINDEN, G., SMITH, B., AND YORK, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing 7*, 1 (2003), 76–80.

[82] LIU, N. N., AND YANG, Q. Eigenrank: a ranking-oriented approach to collaborative filtering. In *SIGIR '08: Proceedings of the 31st annual*

international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA, 2008), ACM, pp. 83–90.

[83] LIU, N. N., ZHAO, M., AND YANG, Q. Probabilistic latent preference analysis for collaborative filtering. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management* (New York, NY, USA, 2009), ACM, pp. 759–766.

[84] MA, H., KING, I., AND LYU, M. R. Effective missing data prediction for collaborative filtering. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2007), ACM, pp. 39–46.

[85] MA, H., KING, I., AND LYU, M. R. Learning to recommend with social trust ensemble. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2009), ACM, pp. 203–210.

[86] MARLIN, B. M., AND ZEMEL, R. S. Collaborative prediction and ranking with non-random missing data. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 5–12.

[87] MCLAUGHLIN, M. R., AND HERLOCKER, J. L. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *SIGIR-04: Proceedings of the 27th annual international conference on Research and development in information retrieval* (New York, NY, USA, 2004), ACM Press, pp. 329–336.

[88] MCNEE, S. M., RIEDL, J., AND KONSTAN, J. A. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1097–1101.

[89] MCNEE, S. M., RIEDL, J., AND KONSTAN, J. A. Making recommendations better: an analytic model for human-recommender interaction. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1103–1108.

[90] MEHTA, B., AND NEJDL, W. Attack resistant collaborative filtering. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2008), ACM, pp. 75–82.

[91] MELVILLE, P., MOONEY, R. J., AND NAGARAJAN, R. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the 18th National Conference on Artificial Intelligence(AAAI-2002)* (Edmonton, Canada, July 2002), pp. 187–192.

[92] MIDDLETON, S. E., SHADBOLT, N. R., AND DE ROURE, D. C. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst. 22*, 1 (2004), 54–88.

[93] MILLER, B. N., KONSTAN, J. A., AND RIEDL, J. Pocketlens: Toward a personal recommender system. *ACM Trans. Inf. Syst. 22*, 3 (2004), 437–476.

[94] MIYAHARA, K., AND PAZZANI, M. J. Collaborative filtering with the simple bayesian classifier. In *In: Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence* (2000), pp. 679–689.

[95] MOBASHER, B., JIN, X., AND ZHOU, Y. Semantically enhanced collaborative filtering on the web. In *Proceedings of the First EuropeanWeb Mining Forum – EWMF 2003* (2004), Springer, pp. 57–76.

[96] MOBASHER, B., BURKE, R., AND SANDVIG, J. J. Model-based collaborative filtering as a defense against profile injection attacks. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence* (2006), AAAI Press, pp. 1388–1393.

[97] MOBASHER, B., BURKE, R., BHAUMIK, R., AND SANDVIG, J. J. Attacks and remedies in collaborative recommendation. *IEEE Intelligent Systems 22*, 3 (2007), 56–63.

[98] MONTANER, M., LÓPEZ, B., AND DE LA ROSA, J. L. A taxonomy of recommender agents on the internet. *Artif. Intell. Rev. 19*, 4 (2003), 285–330.

[99] NANAS, N., ROECK, A., AND VAVALIS, M. What happened to content-based information filtering? In *ICTIR '09: Proceedings of the 2nd International Conference on Theory of Information Retrieval* (Berlin, Heidelberg, 2009), Springer-Verlag, pp. 249–256.

[100] NANAS, N., VAVALIS, M., AND DE ROECK, A. A network-based model for high-dimensional information filtering. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2010), ACM, pp. 202–209.

[101] NGUYEN, A.-T., DENOS, N., AND BERRUT, C. Improving new user recommendations with rule-based induction on cold user data. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems* (New York, NY, USA, 2007), ACM, pp. 121–128.

[102] O'MAHONY, M., HURLEY, N., KUSHMERICK, N., AND SILVESTRE, G. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol. 4*, 4 (2004), 344–377.

[103] ORRIOLS, A., AND BERNADÓ-MANSILLA, E. The class imbalance problem in learning classifier systems: a preliminary study. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation* (New York, NY, USA, 2005), ACM, pp. 74–78.

[104] O'SULLIVAN, D., SMYTH, B., AND WILSON, D. C. Preserving recommender accuracy and diversity in sparse datasets. *International Journal on AI Tools 13*, 1 (2004), 219–235.

[105] PAPAGELIS, M., AND PLEXOUSAKIS, D. Qualitative analysis of user-based and item-based prediction algorithms for recommendation agents. In *CIA* (2004), M. Klusch, S. Ossowski, V. Kashyap, and R. Unland, Eds., vol. 3191 of *Lecture Notes in Computer Science*, Springer, pp. 152–166.

[106] PAPAGELIS, M., ROUSIDIS, I., PLEXOUSAKIS, D., AND THEOHAROPOULOS, E. Incremental collaborative filtering for highly-scalable recommendation algorithms. In *Proceedings of the 15th Inter-*

national Symposium on Methodologies for Intelligent Systems ISMIS-05 (Saratoga Springs, NY, May 25-28 2005), pp. 553–561.

[107] PARK, S.-T., PENNOCK, D., MADANI, O., GOOD, N., AND DECOSTE, D. Naïve filterbots for robust cold-start recommendations. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2006), ACM, pp. 699–705.

[108] PATEREK, A. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup and Workshop* (2007).

[109] PAULSON, P., AND TZANAVARI, A. Combining collaborative and contentbased filtering using conceptual graphs. In *Modeling with Words (Lecture Notes In Artificial Intelligence Series)* (2003), J. Lawry, J. Shanahan, , and A. Ralescu, Eds., Springer-Verlag. conceptural graph.

[110] PAZZANI, M., AND BILLSUS, D. Content-based recommendation systems. *The Adaptive Web -* (2007), 325–341.

[111] PAZZANI, M. J. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev. 13*, 5-6 (1999), 393–408.

[112] PENNOCK, D., HORVITZ, E., LAWRENCE, S., AND GILES, C. L. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, UAI-2000* (Stanford, CA, 2000), Morgan Kaufmann, pp. 473–480.

[113] PILÁSZY, I., AND TIKK, D. Recommending new movies: even a few ratings are more valuable than metadata. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 93–100.

[114] PLATT, J. C., BURGES, C. J. C., SWENSON, S., WEARE, C., AND ZHENG, A. Learning a gaussian process prior for automatically gen-

erating music playlists. In *Advances in Neural Information Processing Systems 14* (2002), pp. 1425–1432.

[115] POPESCUL, A., POPESCUL, R., UNGAR, L. H., PENNOCK, D. M., AND LAWRENCE, S. Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence* (San Francisco, CA, USA, 2001), Morgan Kaufmann Publishers Inc., pp. 437–444.

[116] QUINLAN, J. R. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[117] RASHID, A. M., ALBERT, I., COSLEY, D., LAM, S. K., MCNEE, S. M., KONSTAN, J. A., AND RIEDL, J. Getting to know you: learning new user preferences in recommender systems. In *IUI '02: Proceedings of the 7th international conference on Intelligent user interfaces* (New York, NY, USA, 2002), ACM Press, pp. 127–134.

[118] RENDLE, S., AND SCHMIDT-THIEME, L. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 251–258.

[119] RENNIE, J. D. M., AND SREBRO, N. Fast maximum margin matrix factorization for collaborative prediction. In *ICML '05: Proceedings of the 22nd international conference on Machine learning* (New York, NY, USA, 2005), ACM, pp. 713–719.

[120] RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work* (Chapel Hill, North Carolina, 1994), ACM Press, pp. 175–186.

[121] RUCKER, J., AND POLANCO, M. J. Siteseer: personalized navigation for the web. *Commun. ACM 40*, 3 (1997), 73–76.

[122] SAHOO, N., KRISHNAN, R., DUNCAN, G., AND CALLAN, J. P. Collaborative filtering with multi-component rating for recommender systems. In *Proceedings of the Sixteenth Workshop on Information Technologies and Systems* (Milwaukee, WI, USA, 2006).

[123] SALAKHUTDINOV, R., MNIH, A., AND HINTON, G. Restricted boltzmann machines for collaborative filtering. In *ICML '07: Proceedings of the 24th international conference on Machine learning* (New York, NY, USA, 2007), ACM, pp. 791–798.

[124] SALTER, J., AND ANTONOPOULOS, N. Cinemascreen recommender agent: Combining collaborative and content-based filtering. *IEEE Intelligent Systems 21*, 1 (2006), 35–41.

[125] SALTON, G., AND MCGILL, M. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1986.

[126] SANDVIG, J. J., MOBASHER, B., AND BURKE, R. Robustness of collaborative recommendation based on association rule mining. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems* (New York, NY, USA, 2007), ACM, pp. 105–112.

[127] SANDVIG, J. J., MOBASHER, B., AND BURKE, R. D. A survey of collaborative recommendation and the robustness of model-based algorithms. *IEEE Data Eng. Bull. 31*, 2 (2008), 3–13.

[128] SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. Analysis of recommendation algorithms for e-commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce* (New York, NY, USA, 2000), ACM, pp. 158–167.

[129] SARWAR, B., KARYPIS, G., KONSTAN, J., AND REIDL, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International World Wide Web Conference* (New York, NY, USA, 2001), ACM, pp. 285–295.

[130] SARWAR, B. M., KONSTAN, J. A., BORCHERS, A., HERLOCKER, J. L., MILLER, B. N., AND RIEDL, J. Using filtering agents to im-

prove prediction quality in the grouplens research collaborative filtering system. In *Proceedings of Computer Supported Cooperative Work* (Seattle, WA, 1998), pp. 345–354.

[131] SARWAR, B. M., KARYPIS, G., KONSTAN, J. A., AND RIEDL, J. T. Application of dimensionality reduction in recommender system - a case study. In *In ACM WebKDD Workshop* (2000).

[132] SCHAFER, J. B., KONSTAN, J., AND RIEDI, J. Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce* (New York, NY, USA, 1999), ACM, pp. 158–166.

[133] SCHCLAR, A., TSIKINOVSKY, A., ROKACH, L., MEISELS, A., AND ANTWARG, L. Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 261–264.

[134] SCHEIN, A. I., POPESCUL, A., UNGAR, L. H., AND PENNOCK, D. M. Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2002), ACM, pp. 253–260.

[135] SCHEIN, A. I., POPESCUL, A., UNGAR, L. H., AND PENNOCK, D. M. Croc: A new evaluation criterion for recommender systems. *Electronic Commerce Research 5*, 1 (2005), 51–74.

[136] SCHICKEL-ZUBER, V., AND FALTINGS, B. Using hierarchical clustering for learning theontologies used in recommendation systems. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2007), ACM, pp. 599–608.

[137] SEMERARO, G., LOPS, P., BASILE, P., AND DE GEMMIS, M. Knowledge infusion into content-based recommender systems. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 301–304.

[138] SHARDANAND, U., AND MAES, P. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems* (Denver, CO, May 1995), vol. 1, ACM Press/Addison-Wesley Publishing Co., pp. 210–217.

[139] SHETH, B., AND MAES, P. Evolving agents for personalized information filtering. In *the 9th IEEE Conference on Artificial Intelligence for Applications* (March 1993).

[140] SI, L., AND JIN, R. Flexible mixture model for collaborative filtering. In *In Proc. of ICML* (2003), AAAI Press, pp. 704–711.

[141] SINGH, A. P., AND GORDON, G. J. Relational learning via collective matrix factorization. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2008), ACM, pp. 650–658.

[142] SMYTH, B., AND COTTER, P. Personalised electronic program guides for digital tv. *AI Magazine 22*, 2 (2001), 89–98.

[143] SU, X., AND KHOSHGOFTAAR, T. M. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence 2009* (2009), 1–20.

[144] SU, X., KHOSHGOFTAAR, T. M., ZHU, X., AND GREINER, R. Imputation-boosted collaborative filtering using machine learning classifiers. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing* (New York, NY, USA, 2008), ACM, pp. 949–950.

[145] SVENSSON, M., HÖÖK, K., AND CÖSTER, R. Designing and evaluating kalas: A social navigation system for food recipes. *ACM Trans. Comput.-Hum. Interact. 12*, 3 (2005), 374–400.

[146] TAKÁCS, G., PILÁSZY, I., NÉMETH, B., AND TIKK, D. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res. 10* (2009), 623–656.

[147] TAKÁCS, G., PILÁSZY, I., NÉMETH, B., AND TIKK, D. A unified approach of factor models and neighbor based methods for large

recommender systems. In *1th IEEE Workshop on Recommender Systems and Personalized Retrieval* (Ostrava, Czech Republic, August 4, 2008).

[148] TINTAREV, N., AND MASTHOFF, J. Effective explanations of recommendations: user-centered design. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems* (New York, NY, USA, 2007), ACM, pp. 153–156.

[149] TRAN, T., AND COHEN, R. Hybrid recommender systems for electronic commerce. In *Knowledge-based electronic markets, papers from the AAAI workshop* (2000), AAAI Press, pp. 78–83.

[150] TRUYEN, T. T., PHUNG, D. Q., AND VENKATESH, S. Preference networks: probabilistic models for recommendation systems. In *AusDM '07: Proceedings of the sixth Australasian conference on Data mining and analytics* (Darlinghurst, Australia, Australia, 2007), Australian Computer Society, Inc., pp. 195–202.

[151] UNGAR, L., AND FOSTER, D. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems* (1998), AAAI Press.

[152] WANG, J., DE VRIES, A. P., AND REINDERS, M. J. T. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2006), ACM, pp. 501–508.

[153] WEI, Y. Z., MOREAU, L., AND JENNINGS, N. R. A market-based approach to recommender systems. *ACM Trans. Inf. Syst. 23*, 3 (2005), 227–266.

[154] WEIMER, M., KARATZOGLOU, A., AND BRUCH, M. Maximum margin matrix factorization for code recommendation. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems* (New York, NY, USA, 2009), ACM, pp. 309–312.

[155] WENG, S.-S., LIN, B., AND CHEN, W.-T. Using contextual information and multidimensional approach for recommendation. *Expert Syst. Appl. 36*, 2 (2009), 1268–1279.

[156] XUE, G.-R., LIN, C., YANG, Q., XI, W., ZENG, H.-J., YU, Y., AND CHEN, Z. Scalable collaborative filtering using cluster-based smoothing. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2005), ACM, pp. 114–121.

[157] YU, K., WEN, Z., ESTER, M., AND XU, X. Feature weighting and instance selection for collaborative filtering. In *DEXA '01: Proceedings of the 12th International Workshop on Database and Expert Systems Applications* (Washington, DC, USA, 2001), IEEE Computer Society, p. 285.

[158] YU, K., SCHWAIGHOFER, A., TRESP, V., XU, X., AND KRIEGEL, H.-P. Probabilistic memory-based collaborative filtering. *IEEE Transactions on Knowledge and Data Engineering, special issue on Mining and Searching the Web 0* (2003), 0.

[159] YU, K., TRESP, V., AND YU, S. A nonparametric hierarchical bayesian framework for information filtering. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2004), ACM, pp. 353–360.

[160] YU, K., ZHU, S., LAFFERTY, J., AND GONG, Y. Fast nonparametric matrix factorization for large-scale collaborative filtering. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2009), ACM, pp. 211–218.

[161] ZENEBE, A., AND NORCIO, A. F. Representation, similarity measures and aggregation methods using fuzzy sets for content-based recommender systems. *Fuzzy Sets Syst. 160*, 1 (2009), 76–94.

[162] ZHANG, J., AND PU, P. A recursive prediction algorithm for collaborative filtering recommender systems. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems* (New York, NY, USA, 2007), ACM, pp. 57–64.

[163] ZHANG, M., AND HURLEY, N. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), ACM, pp. 123–130.

[164] ZHANG, Y. Using bayesian priors to combine classifiers for adaptive filtering. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2004), ACM, pp. 345–352.

[165] ZHANG, Y., AND KOREN, J. Efficient bayesian hierarchical user modeling for recommendation system. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval* (New York, NY, USA, 2007), ACM, pp. 47–54.

[166] ZHANG, Y., AND ANDREAE, P. Iterative neighbourhood similarity computation for collaborative filtering. In *Web Intelligence* (2008), pp. 806–809.

[167] ZHENG, J., WU, X., NIU, J., AND BOLIVAR, A. Substitutes or complements: another step forward in recommendations. In *EC '09: Proceedings of the tenth ACM conference on Electronic commerce* (New York, NY, USA, 2009), ACM, pp. 139–146.

[168] ZIEGLER, C.-N., LAUSEN, G., AND SCHMIDT-THIEME, L. Taxonomy-driven computation of product recommendations. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management* (New York, NY, USA, 2004), ACM, pp. 406–415.

[169] ZIEGLER, C.-N., MCNEE, S. M., KONSTAN, J. A., AND LAUSEN, G. Improving recommendation lists through topic diversification.

In *WWW '05: Proceedings of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM, pp. 22–32.

[170] The digg announcement: http://about.digg.com/blog/digg-recommendation-engine-updates.

[171] The EachMovie dataset: http://www.grouplens.org/node/76.

[172] The internet movie database (IMDB), downloadable at http://www.imdb.com/interfaces, 2009.

[173] The Jester dataset: http://www.ieor.berkeley.edu/~goldberg/jester-data/.

[174] The NetFlix prize: http://www.netflixprize.com//index.

[175] The Yahoo! Webscope ratings data sets: http://research.yahoo.com/Academic_Relations.