Recommender Systems

by

Bronwyn Catherine Dumbleton

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science (Mathematical Statistics) in the Faculty of Science at Stellenbosch University

Supervisor: Dr. S. Bierman

December 2019

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: December 2019 .....

Abstract

Recommender Systems

B.C. Dumbleton

Thesis: MSc

December 2019

A Recommender System (RS) is a particular type of information filtering system used to propose relevant items to users. Their successful application in online retail is reflected in increased customer satisfaction and sales revenue, with further application in entertainment, e-commerce and services, and content. Hence it may be argued that recommender systems currently present some of the most successful and widely used machine learning algorithms in practice.

We provide an overview of both standard and more modern approaches to recommender systems, including content-based and collaborative filtering, as well as latent factor models for collaborative filtering. A limitation of standard latent factor models is that their input is typically restricted to a set of item ratings. In contrast, general purpose supervised learning algorithms allow more flexible inputs, but are typically not able to handle the degree of data sparsity prevalent in recommendation problems. Factorisation machines, which are supervised learning methods, are able to incorporate more flexible inputs and are well suited to deal with the effects of data sparsity. We therefore study the use of factorisation in recommender problems and report an empirical study in which we compare the effects of data sparsity on latent factor models, as well as on factorisation machines.

Currently in RS research, emphasis is placed on the advantages of recommender systems that yield recommendations that are simple to explain to users. Such recommender systems have been shown to be much more trustworthy than more complex, unexplainable systems. Towards a proposal for explainable recommendations, we also provide an overview of the connection between the recommender problem and Multi-Label Classification (MLC). Since some of the recent MLC approaches facilitate the interpretation of predictions, we conduct an empirical study in order to evaluate the use of various MLC approaches in the context of recommender problems.

Opsomming

Aanbevelingstelsels

("Recommender Systems")

B.C. Dumbleton

Tesis: MSc

Desember 2019

'n Aanbevelingstelsel (ABVS) is 'n spesifieke tipe inligting-siftingstelsel wat gebruik word om relevante items aan gebruikers voor te stel. Die suksesvolle toepassing van hierdie stelsels in aanlyn-aankope word gereflekteer in hoër gebruikersatisfaksie en wins, met verdere toepassings in die vermaaklikheidswêreld, e-handel, dienste, en inhoud. Derhalwe sou 'n mens kon argumenteer dat aanbevelingstelsels huidiglik van die suksesvolste en algemeenste masjienleer-algoritmes in die praktyk is.

Hierdie tesis gee 'n oorsig oor beide die standaard-, en ook oor die moderner benaderings tot aanbevelingstelsels, insluitend inhoudsgebaseerde- en samewerkingsifting, sowel as latente faktor modelle vir samewerkingsifting. 'n Beperking van standaard latente faktor modelle is dat hulle invoer tipies slegs in die vorm van 'n versameling itemgraderings kan wees. In teenstelling hiermee, laat algemene ondertoesig leer-algoritmes buigsamer invoer toe, maar is hulle nie instaat om die graad van dataskaarsheid te hanteer wat in aanbevelingsprobleme aanwesig is nie. Faktoriserings-algoritmes, as ondertoesig leer-algoritmes, is daartoe instaat om buigsamer invoere te inkorporeer, en is geskik om die gevolge van dataskaarsheid to hanteer. Die gebruik van faktoriserings-algoritmes in aanbevelingsprobleme word derhalwe in hierdie tesis bestudeer, en die gevolge van dataskaarsheid op latente faktor modelle, sowel as op faktoriserings-algoritmes, word empiries vergelyk.

Huidiglik in ABVS navorsing, word die voordele van stelsels wat aanbevelings lewer wat makliker is om aan gebruikers te verduidelik, beklemtoon. Dit is bewys dat sulke stelsels meer betroubaar as ingewikkelder, onverduidelikbare stelsels is. In aanloop tot 'n voorstel vir meer verklaarbare aanbevelings, word 'n oorsig gegee oor die verband tussen die aanbevelingsprobleem en meervuldige-Y klassifikasie (MYK). Aangesien sommige van die onlangse meervuldige-Y klassifikasie benaderings die interpretasie van vooruitskattings

fasiliteer, word 'n empiriese studie gedoen ten einde die gebruik van 'n aantal MYK benaderings in aanbevelingsprobleme te evalueer.

Acknowledgements

I would like to express my sincere gratitude to the following people and organisations:

- Dr. S. Bierman for her patience, enthusiasm and motivation over the entire duration of this thesis.
- My friends and family for their unfailing support and encouragement throughout my studies.
- The National Research Foundation (NRF) for their financial assistance. *Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.*

Contents

List of Figures

List of Tables

List of Abbreviations

AFM	Attentional Factorisation Machine
ALS	Alternating Least Squares
BP-MLL	Back-Propagation Multi-Label Learning
BR	Binary Relevance
BR-kNN	Binary Relevance k-nearest Neighbours
CARS	Context-Aware Recommender System
CB	Content-based
CC	Classifier Chains
CF	Collaborative Filtering
CTR	Click Through Rate
CW	Curds-and-Whey
FM	Factorisation Machine
FFM	Field-Aware Factorisation Machine
FICYREG	Filtered Canonical Y-variate Regression
FwFM	Field-Weighted Factorisation Machine
GP	Global Popular
HOFM	Higher-Order Factorisation Machine
IDF	Inverse Document Frequency
IFS	Information Filtering System
IMDb	Internet Movie Database

IP	Item Popular
LFM	Latent Factor Model
LP	Label Powerset
MAE	Mean Absolute Error
MF	Matrix Factorisation
MLC	Multi-Label Classification
ML-RBF	Multi-Label Radial Basis Function
ML-kNN	Multi-label k-nearest Neighbours
NMF	Non-Negative Matrix Factorisation
OLS	Ordinary Least Squares
PCA	Principal Component Analysis
PITF	Pairwise Interaction Tensor Factorisation
RAkEL	Random k-labelsets
RARS	Remedial Actions Recommender System
RMSE	Root Mean Squared Error
RR	Reduced Rank
RS	Recommender System
SGD	Stochastic Gradient Descent
SPTF	Scalable Probabilistic Tensor Factorisation
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TF	Term Frequency
TF-IDF	Term Frequency Inverse Document Frequency
UP	User Popular
VSM	Vector Space Model

Chapter 1

Introduction

Since its invention by Tim Berners-Lee in 1989 (Berners-Lee, 1989), the growth of the World Wide Web has facilitated the ease with which knowledge may nowadays be shared. However, the resulting abundance of information may quickly cause web users to experience information overload (Mak *et al.*, 2003). When presented with any form of information, humans tend to naturally filter out details that are irrelevant to them. Consider, for example, the decision to buy a magazine. We expect a person to buy a magazine that is particular to his/her interest, such as travel or cookery. Furthermore, within that magazine, an individual would only take note of articles that he/she finds appealing. An Information Filtering System (IFS) does this on a much larger scale. More specifically, the objective of an IFS is to narrow the amount of information shown to a user, based on their preferences, in an automatic way.

A Recommender System (RS) is a particular type of information filtering system. It aims to reduce the volume of information presented to an individual, by suggesting items (for example books, movies, or websites) that correspond to their specific interests and requirements (Burke, 2002). According to the taxonomy of recommender systems described by Montaner *et al.* (2003), recommender systems may be organised into four general domains, depending on the type of content that they recommend. These domains, which by far the majority of recommender systems focus on, are entertainment, content, e-commerce and services. More recently, some research has been devoted to recommendations during data exploration, data visualisation, and work-flow design. These areas are explored by Drosou and Pitoura (2013), Ehsan *et al.* (2016) and Jannach *et al.* (2016), respectively. The items that may be recommended in the aforementioned domains are named in Table 1.1.

Table 1.1: Application Domains of Recommender Systems.

Entertainment	Movies, music
Content	Documents, web pages, newspapers
E-commerce	Products to purchase, such as fashion, books, furniture
Services	Accommodation, beauty salons, travel services, restaurants
Databases	Data exploration, data visualisation, work-flow design

The simplest type of recommendations are given by non-personalised recommenders, which are not unique to an individual or user. Non-personalised recommenders suggest items that may be interesting to all users. Certain events may acquire a large amount of attention in a short period of time, causing them to appear in a 'trending' section, such as on the video streaming site YouTube. The most popular product might be recommended on an e-commerce website, while a magazine might list the top ten bestselling books of the month. Since non-personalised recommenders generally recommend the most sought-after items, they are very straightforward to implement.

On the other hand, personalised recommenders guide users to items that are most likely to meet their particular needs. Therefore, the recommendations provided by a personalised RS will differ greatly between users. In this study, we focus on personalised recommenders (as does most research in this field). Personalised recommenders are useful not only to the users of the system, but to the service provider (Ricci *et al.*, 2011). E-commerce websites which make use of personalised recommendations have been shown to have a huge positive impact on sales revenue. These websites display items particular to users' interests, and therefore user satisfaction is improved. As users are more likely to purchase products that appeal to their needs, the number of items sold by the provider is likely to increase. An example of such a recommender is Netflix, the well-known online movie and TV show streaming service. Netflix provides recommendations that are based on users' past preferences, thereby allowing a user to easily select a new movie to watch. By providing reliable recommendations, Netflix was able to save approximately $1 billion dollars by preventing customer churning (Gomez-Uribe and Hunt, 2015). The online store Amazon also experienced an improved turnover from implementing a recommender system. It was estimated that 35% of the sales on Amazon emanate from accurate recommendations (MacKenzie *et al.*, 2013). Thus, we can see that personalised recommenders have an added advantage over non-personalised recommender systems. In our study, we focus on two types of personalised recommendations. In the first type of recommendation, we consider the typical problem of recommending a new item to a user that they are most likely to enjoy. The second type of recommendation that we consider is suggesting the best context in which an item should be consumed by the user.

In the remainder of this chapter, we discuss the general ideas underlying recommender systems. The notation which is commonly used in an RS context, and based largely on the notation utilised by Lops *et al.* (2011) and Desrosiers and Karypis (2011), is also established. We introduce the notation, along with concepts and terminology that are fundamental to the study of recommender systems in Section 1.1. The format in which data may be stored in an RS database is discussed in Section 1.2, while the most common types of recommender systems are introduced in Section 1.3. In Section 1.4 we describe key factors that need to be considered when designing an RS. Following this, common metrics used for the evaluation of recommenders are discussed in Section 1.5. The purpose of our study is given in Section 1.6 and we close this chapter with an overview of the thesis, given in Section 1.7.

1.1 Concepts, Terminology and Notation

The terms *users* and *items* are continually used when discussing recommender systems. Objects to be recommended by the system, such as movies, clothing, or books, are generally referred to as *items*. Individuals who make use of the system in order to find items to their liking, are called *users*.

Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ and $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ respectively denote the set of n users and m items in a system. Let also the recorded ratings in the system be denoted by the set $\mathcal{R}$, and the values that these ratings may assume be contained in the set $\mathcal{S}$.

We assume that each user is allowed to rate a particular item only once, and let the rating given by a user $u \in \mathcal{U}$ to an item $i \in \mathcal{I}$ be denoted by r_{ui}. The subset of items rated by user u is denoted by $\mathcal{I}_u$, and similarly, the subset of users that have rated item i is given by $\mathcal{U}_i$. To denote the exclusion of a set $\mathcal{I}_u$ from a set $\mathcal{I}$, we use the notation $\mathcal{I} \setminus \mathcal{I}_u$. Also, note that we usually use the notation $|\mathcal{X}|$ to denote the number of elements in the set $\mathcal{X}$. When we need to avoid confusion with the absolute value sign, we use the notation $n(\mathcal{X})$ instead. Typically, when referring to a specific user under consideration, he/she is called the *active user*, represented by u_a.

Based on users' feedback, an RS aims to determine which items a particular user is likely to enjoy. An RS may obtain feedback in an implicit or explicit manner. Alternatively, implicit and explicit feedback may be combined. *Explicit feedback* entails users directly evaluating objects, while *implicit feedback* is obtained in an indirect manner, for example by monitoring users' activity on the system.

Explicit feedback is incredibly useful since it allows the system to learn exactly how the user perceives an object (Levinas, 2014). Although rating objects seems to be a simple task, it may happen that users interpret rating scales differently. For example, two users may in fact both like an item, but one may be more generous in their rating than the other. A further drawback of explicit ratings is that they are difficult to acquire. Users may find it tedious and inconvenient to rate items after consuming them, or they might simply forget. Thus no rating is obtained for that user regarding the item.

In contrast to explicit feedback, implicit feedback does not require the user to actively state their preferences. Instead, the system attaches relevance scores to users' actions in order to decide if a specific user values a particular acquired object (Lops *et al.*, 2011). Implicit feedback tends to be less biased. For example, there is no need for the user to rate an item highly simply because it seems to be popular at the moment. There are, however, drawbacks to this form of feedback. The system will assess a user's actions, such as browsing time or purchase history, and base recommendations on these. Whereas an item purchased by a user should imply that the user likes the item, he/she may have purchased it for someone else, or the account could be shared among individuals.

One may of course use different scales in order to determine whether or not a user appreciates an item

(Schafer *et al.*, 2007). These include binary, ordinal, numerical, and unary rating scales. Since some scales are more suited to certain contexts, the type of rating scale utilised typically depends on the RS domain. When a binary rating scale is implemented, a user is simply asked to decide whether an item is good or bad. An example is the 'thumbs up' and 'thumbs down' feature on YouTube. Surveys or questionnaires generally make use of ordinal scales, where the user selects the level to which they agree with a statement. For example, a user will select from the list {*strongly agree*, *agree*, *neutral*, *disagree*, *strongly disagree*}. One of the most well-known rating scales is the numerical scale, where a user expresses his/her interest by a 1-to-N rating, where 1 indicates complete disinterest or dislike, and N indicates that the user thoroughly enjoyed the item. Finally, unary ratings are typically associated with the situation where there is only an option for 'liking' the item, and not for 'disliking' it. An example of unary ratings may be found on the social media platform Facebook, where users can 'like' a post, but not 'dislike' it (Aggarwal, 2016). Note that unary ratings may also be collected as part of a user's implicit feedback, where the time spent viewing an item, or purchasing it, would convey to the system that a user is interested in the item.

1.2 Data Representation

A database is used to store information concerning items that may be recommended to a user. The database can take on either a structured or an unstructured format. In the case of a structured database, a set of features or attributes are used to describe the items. The features remain the same for all items, and each feature can assume a known set of values. For example, consider Table 1.2, which depicts the first five entries in a very simple database concerning books. Each row represents a book, while the columns represent the features. A unique ID is assigned to each book in order to ensure that books can be distinguished from each other, should there be more than one book with the same title.

Table 1.2: A book database.

Item ID	**Title**	**Author**	**Year**	**Genre**
i_{01}	Jane Eyre	Charlotte Bronte	1847	Social Romance
i_{02}	Pride and Prejudice	Jane Austen	1813	Romance
i_{03}	Dune	Frank Herbert	1965	Science fiction
i_{04}	Ender's Game	Orson Scott Card	1985	Military science fiction
i_{05}	The Hobbit	J. R. R. Tolkien	1937	Fantasy

On the other hand, unstructured data do not contain observations with clearly defined features. Therefore, unstructured data cannot be arranged in row and column format as seen in Table 1.2. An example of an unstructured database would be an audio or video file, or a collection of product reviews.

Figure 1.1 is an excerpt of a product review taken from Amazon. Each review is different in terms of style, length and language, and therefore has no structure. Typically, a mixture of structured and unstructured data are used in recommender systems, thereby causing RS databases to be semi-structured.

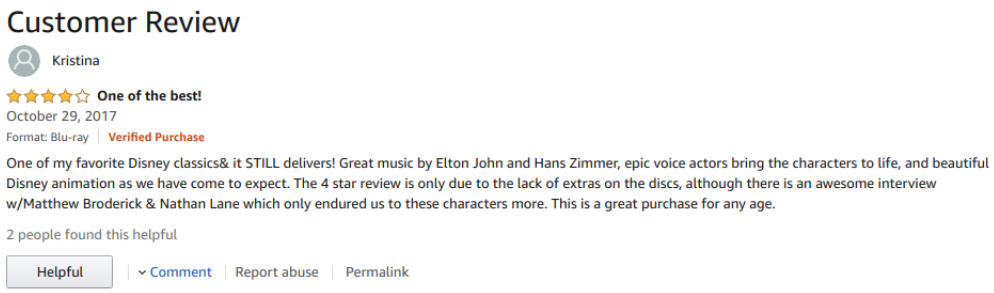


Figure 1.1: An example of a product review on Amazon

Source: Amazon [Online].

An RS database will also store the ratings that a user has given items. Consider again the book database in Table 1.2. The ratings that users u_{01} and u_{03} have given to the different books are shown in Table 1.3. From Table 1.2 and Table 1.3 we can deduce that u_{01} is likely to enjoy romance books, while u_{03} would rather read science fiction.

Table 1.3: Users' ratings of books.

User ID	Item ID	Rating
u_{01}	i_{01}	4
u_{01}	i_{02}	3
u_{03}	i_{02}	1
u_{03}	i_{03}	5
u_{03}	i_{04}	4

1.3 Types of Recommender Systems

Most frequently in the literature, personalised recommender systems are categorised into three main classes, *viz.* content-based, collaborative filtering, and hybrid approaches. There are, however, also more extensive taxonomies, as given for example in Burke (2002). Thereby, recommender systems may be partitioned into content-based, collaborative filtering, knowledge-based, demographic and hybrid systems. When deciding between these recommenders, there are a number of factors that should be taken into consideration. The type of information available, the domain in which the recommendations are to be made, as well as the algorithms to be used, can play a role in selecting the type of recommender system to implement (Montaner *et al.*, 2003).

We focus in this section on the way in which the type of data used by an RS determines the way in which it may be categorised. In the RS context, it is possible that the only available data is the ratings given by users to items. In certain domains, external knowledge, such as attribute information associated with sets of users and items, may also be available. Keeping the above data scenarios in mind, we provide brief descriptions of the five different recommender strategies identified in Burke (2002).

In Content-based (CB) systems, the item features and the ratings that a user gave to items are used to recommend new items. Content-based recommendation is based on the idea that the user is likely to enjoy new items that have similar features to the ones that they have enjoyed in the past. Since only the items that an active user has rated in the past are considered when making recommendations, content-based recommendation is user-specific. By not exploiting interactions between different users in the system, clearly CB methods are severely limited. A news recommender system, called NewsWeeder, is an example of one of the earliest content-based recommenders (Lang, 1995).

Collaborative Filtering (CF) systems find users that have rated items similarly to the active user, and make recommendations based on these similar users. In other words, recommendations are based on the idea that the active user should enjoy items that users with a similar taste to them have enjoyed. The first implementation of this type of recommender is attributed to Goldberg *et al.* (1992). The two types of CF algorithms are memory- and model-based. The former is further divided into user- and item-based CF. The latter is based on using typical regression or classification models to make a recommendation. The main drawback of a CF system is that an item needs to have been rated before it can be recommended, or a user has to have rated an item before they can receive recommendations. In other words, a sufficient amount of information is needed regarding the items and/or users before recommendations can be made. Scenarios where this is not the case, i.e. if there is no information available regarding certain users' preferences, or on certain item ratings, are referred to as occurrences of the *cold-start problem*. If many item ratings are missing, the data to be used to in the RS can become very sparse. Several model-based algorithms have been proposed in an attempt to alleviate the data sparsity problem, such as latent factor models, which are based on Singular Value Decomposition (SVD).

A knowledge-based system aims to meet users' requirements by using domain knowledge about the items, as well as information on user preferences. Knowledge-based systems are useful for recommending items that are seldom purchased, and therefore associated with either a limited number of ratings, or no ratings at all. An example of such a system is the online classified advertisement website, Gumtree[1], where users input certain requirements they need from an item, and the recommender determines which items best match these needs.

In a demographic RS, rather than only relying on ratings or item information, the available demographic information of a user is used to make a recommendation. Users are grouped together according to the demographic information that they have provided, such as age, gender or location, and based on the group or niche into which a user falls, a recommendation is made. The drawback of this method is that demographic data is generally difficult to acquire due to privacy concerns, therefore the applications are often limited. In the paper by Wang *et al.* (2012), they consider the recommendation of tourist attractions using this approach on data from TripAdvisor. Different machine learning algorithms are considered to

[1] https://www.gumtree.co.za/

group users according to their demographic information. A recommendation is made to the active user by identifying to which class they belong, and then suggesting an attraction based on the ratings of the users in the identified class.

When various forms of inputs are available, it is possible to use different types of recommender systems. Hybrid recommender systems, as the name suggests, are mixtures of the above mentioned techniques. They allow for the incorporation of the useful qualities of one system to be used in conjunction with another system that lacks these qualities. For example, content-based recommenders do not suffer from the new item problem as they rely on the content of items. Collaborative filtering systems, however, cannot recommend an item that has no ratings. Thus, by combining these two techniques, the performance of a RS is no longer impaired by not being able to suggest new items (Ricci *et al.*, 2011).

1.4 Properties of Recommender Systems

One of the most desirable properties of an RS is that it provides accurate suggestions to users. The accuracy is typically measured using one of several evaluation metrics, to be discussed in Section 1.5. However, there are a number of other factors that should also be considered when designing an RS. Common properties to be considered include scalability, robustness, diversity, novelty and serendipity. While we provide a brief description of each of these factors, a more extensive examination may be found in Aggarwal (2016) and in Shani and Gunawardana (2011). Moreover, metrics used to evaluate recommender systems based on a number of these factors can be found in Kaminskas and Bridge (2016).

- **Scalability**. As the number of users and items in an RS increases, the volume of data in the form of explicit and implicit ratings also increases. This means that the size of a database for recommender systems continues to grow over time. An important consideration therefore is whether an RS is able to provide good recommendations in an efficient and effective manner, even on very large datasets. The scalability of an RS is typically assessed in terms of training time, prediction time and memory requirements.

- **Robustness**. A recommender system is said to be 'under attack' when false or fake ratings are purposefully entered into the system. This is generally done in order to skew the popularity of an item. For example, a restaurant owner might create false profiles to leave positive reviews for their restaurant, while leaving negative ones for the surrounding restaurants. An RS is said to be robust when its recommendations remain stable and unaffected by such fake ratings.

- **Diversity**. Consider the top five recommendations provided by a book recommender. If the recommender suggests books written by only one author, its recommendations are very similar. If the active user does not like the first recommendation, it is very likely that he/she will not enjoy

any of the remaining books. Hence, in such a case, no useful recommendations were provided. It therefore seems sensible to require an RS to provide recommendations that are diverse in nature. Diversity can be measured using the similarity between items.

- **Novelty**. When an RS recommends items that the active user was previously unaware of, the recommendation is said to be novel. An easy way to ensure novelty is to remove items that the user has rated from the list of recommended items. A user-study can be conducted in order to determine the novelty of an RS. The users will be asked to explicitly state whether or not they have seen the items before.

- **Serendipity**. A recommendation which is new and unexpected, but enjoyed by a user, may be regarded as a *serendipitous* recommendation. Serendipitous recommendations are novel, however novel recommendations are not necessarily serendipitous. This is because the only requirement of a novel recommendation is that the user should previously have been unaware of the item. Ge *et al.* (2010) and Kotkov *et al.* (2016) discuss methods to evaluate the serendipity of an RS.

1.5 Evaluation of Recommender Systems

In this section we consider common metrics used for the evaluation of recommender systems. This discussion is based largely on Desrosiers and Karypis (2011).

One may view the item recommendation task to be either a prediction or ranking task (Han and Karypis, 2005). In a prediction setup, the aim is to obtain the single (best) item deemed most likely to be of interest to the active user. This means that for user u_a, a single, unseen item $i \in \mathcal{I} \setminus \mathcal{I}_{u_a}$ is proposed. In other words, we aim to either predict a numerical rating for an unseen item, or to classify an unseen item according to, for example, a binary or ordinal scale. This formulation of the recommendation problem clearly fits into a regression or a (multi-class) classification framework.

In order to evaluate a best item recommender, before the recommender is built, the set of ratings $\mathcal{R}$ are divided into a training set $\mathcal{R}_{train}$ and a test set $\mathcal{R}_{test}$. Let f be a function such that $f : \mathcal{U} \times \mathcal{I} \to \mathcal{S}$. Using $\mathcal{R}_{train}$, we then learn the model $\hat{f}$, and for each item $i \in \mathcal{I} \setminus \mathcal{I}_{u_a}$, the rating that user u_a would give to the item is predicted via the function $\hat{f}(u, i)$. The item i^* with the highest rating is shown to the user, where

$$i^* = \arg\max_{j \in \mathcal{I} \setminus \mathcal{I}_{u_a}} \hat{f}(u_a, j). \tag{1.1}$$

When ratings are predicted, the accuracy measures commonly used to evaluate the system are the Mean Absolute Error (MAE) or the Root Mean Squared Error (RMSE). These are given by Equations 1.2

and 1.3, respectively.

$$\text{MAE}(\hat{f}) = \frac{1}{n(\mathcal{R}_{test})} \sum_{r_{ui} \in \mathcal{R}_{test}} |\hat{f}(u,i) - r_{ui}|, \text{ and} \tag{1.2}$$

$$\text{RMSE}(\hat{f}) = \sqrt{\frac{1}{n(\mathcal{R}_{test})} \sum_{r_{ui} \in \mathcal{R}_{test}} (\hat{f}(u,i) - r_{ui})^2}. \tag{1.3}$$

When explicit ratings are unavailable, and only the purchase history of the active user is available, for example, we have a ranking setup. This is commonly used in content-based recommendation, where the system recommends items that are similar to the items previously purchased by the user. Here, a list $L(u_a)$ of N items that a user u_a would most likely enjoy are displayed. The value of N is typically a small value decided before recommendations are made.

For evaluation purposes, before the recommender is built, the items $\mathcal{I}$ are split into a training and test set, denoted by $\mathcal{I}_{train}$ and $\mathcal{I}_{test}$, respectively. Additionally, a test user is selected and the subset of test items that the user has purchased is denoted by $T(u) \subset \mathcal{I}_u \cap \mathcal{I}_{test}$. The two most common measures to assess the performance of an RS that recommends a list of items are precision (Equation 1.4) and recall (Equation 1.5). Whereas precision is the proportion of items predicted to be relevant that are in fact relevant, recall is the proportion of actual relevant items that have been suggested.

$$\text{Precision}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|L(u)|} |L(u) \cap T(u)|, \text{ and} \tag{1.4}$$

$$\text{Recall}(L) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|T(u)|} |L(u) \cap T(u)|. \tag{1.5}$$

Using the method of displaying a list of N recommendations means that the user is inconvenienced by the fact that all items are regarded as being equally applicable (Desrosiers and Karypis, 2011). In other words, no specific item is presented as being more likely to be enjoyed by the user, and so the user would still have to determine this for himself from the given list L.

It is also possible that the goal of the recommender is not to suggest items to users, but rather, for example, tags associated with items or even the context in which the item should be used. The formulation clearly fits into a mulit-label classification framework. The metrics used for the evaluation of the recommender may also be precision and recall.

1.6 Purpose of the Study

1.6.1 Motivation

With the exponential growth of information being readily available to individuals, recommender systems are proving to be crucial in reducing information overload. A number of studies have been conducted

comparing the performance of recommender algorithms. A comparative study of algorithms was conducted by Stomberg (2014), however only user-based, item-based and SVD algorithms were considered. Furthermore, the effect of data sparsity on the algorithms was not considered. Cremonesi *et al.* (2010) present a similar comparison of non-personalised algorithms, neighbourhood methods and latent factor models on two movie datasets. The methods are evaluated based on top-N recommendation, therefore the metrics used are precision and recall. As in Stomberg (2014), sparsity is not considered.

As the field of recommender systems is ever-expanding, there is scope for an extended comparison of recommender algorithms, beyond that of the traditional approaches considered in the above mentioned papers. In order to understand the state-of-the-art approaches, a thorough study of the traditional approaches is necessary and is therefore also included in this study. Furthermore, the task of recommendation typically focuses on the recommendation of an item to a user, and does not consider recommending appropriate contexts in which to consume the item. This leads to a further avenue of research, namely recommendation via the use of multi-label classification methods. Some recently proposed multivariate regression approaches to MLC assist with the interpretability of MLC output. As explainable recommender systems is an important topic of interest in RS research, we believe the use of MLC in recommender systems to be a promising research direction.

1.6.2 Objectives

The aim of this thesis is to provide an overview of both the traditional and more recent, state-of-the-art algorithms used for recommender systems. We aim to:

- Demonstrate the construction and properties of different types of recommender algorithms.
- Empirically investigate the effects of data sparsity on their performance.
- Test the validity of the use of various MLC approaches for the recommendation problem.

1.7 Thesis Overview

The remainder of the thesis may conceptually be partitioned into two main sections. Chapters 2 to 5 provide an overview of the various recommendation techniques, while in Chapters 6 and 7 we report on our empirical work.

In the first part of the thesis, we start with an overview of content-based recommender systems in Chapter 2. The architecture and data pre-processing steps for this class of recommender systems are described, and augmented with a discussion of the advantages and disadvantages of content-based recommenders. In Chapter 3 we consider the most popular method of recommendation, namely collaborative filtering. The two CF approaches and their main drawbacks are discussed. This leads us to an overview

of more advanced CF methods in Chapter 4, where we we consider extensions to model-based CF, known as Latent Factor Models (LFMs). The latter class of models aim to address the data sparsity problem. For an overview of an entirely different perspective on the recommendation problem, in Chapter 5 we approach the recommender problem from a multi-label classification point of view. Both established and more recently proposed (regression) approaches to MLC are described. Performance measures deemed relevant in an MLC context, are also given.

The second part of the thesis is devoted to a discussion of the two empirical studies undertaken during the study. The first empirical study, with the aim of evaluating the impact of data sparsity on various CF and LFM algorithms, is described in Chapter 6. The second empirical study was carried out in order to investigate the use of MLC algorithms in the context of recommender systems. The MLC experiments and results are discussed in Chapter 7. We conclude the thesis in Chapter 8, with a summary, and with a few suggestions regarding avenues for further research.

Chapter 2

Content-based Recommenders

2.1 Introduction

The focus of this chapter is on one of the two main approaches to personalised recommenders, namely content-based recommendation. Based on items that a given user has previously liked or rated, a Content-based (CB) recommender systems suggests similar items to a user. The idea underlying CB recommender systems is that a user should enjoy new items that have features in common with items that they previously enjoyed. For example, a movie recommender might suggest new movies that have the same genre as the movies that the user enjoyed in the past. In order to make these recommendations, the content of the active user's rated items are examined, thereby creating a unique user profile (Balabanović and Shoham, 1997).

Therefore, in content-based recommendation, descriptive profiles are needed for the users. These profiles often rely on external information regarding the items. For example, in the case of a movie recommender, commonly used item information include movie genre and actors, or tags used to describe the movie. In this way, Magnini and Strapparava (2001) developed a movie recommender by analysing movie synopses available from Internet Movie Database (IMDb). Another successful implementation of content-based filtering is a book recommender called Learning Intelligent Book Recommending Agent, or LIBRA (Mooney and Roy, 2000). Here, product descriptions, available from the online store Amazon, were analysed and a bag-of-words naive Bayesian text classifier was used to learn a user profile.

The remainder of this chapter is structured as follows. The architecture of a CB recommender system is discussed in Section 2.2, followed by an overview of a regularly employed technique which is used by CB recommender systems in order to represent items in a meaningful way. This overview may be found in Section 2.3. As a simple illustration, we discuss how linear regression can be used to learn a user profile in Section 2.4. A discussion of the advantages and common problems that may be expected when using a CB recommender system may be found in Sections 2.5 and 2.6, respectively. We conclude the chapter

with a summary in Section 2.7.

2.2 The Architecture of Content-based Recommenders

The main components of a CB recommender are the Content Analyser, the Profile Learner and the Filtering Component. Each component is responsible for a different step in the recommendation process. In short, the Content Analyser receives the items and converts them into a usable format. The Profile Learner aims to use feedback in order to discover the preferences of a user, whereafter the Filtering Component uses the learned profile to recommend new items to an active user.

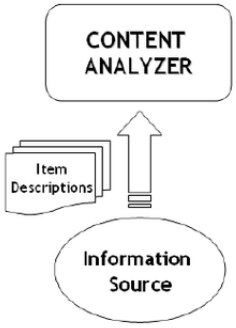


Figure 2.1: The input of the Content Analyser.

Source: Desrosiers and Karypis (2011).

The Content Analyser is responsible for the first step in the recommendation process. As seen in Figure 2.1, this component receives items from some information source. Examples of information sources are web pages, journals, and social media platforms. An information source contains descriptions of the items, which the Content Analyser may then convert into a usable format.

In the case of structured data, the CB recommender is said to be feature-based, while in the case of unstructured data, the CB recommender is described as text categorisation-based (Mooney and Roy, 2000). Difficulties arise when incorporating unstructured data (such as unrestricted text) into a recommender system. Unrestricted text can include product descriptions, item tags, movie synopses or news articles, to name a few. So-called Vector Space Models (VSMs) were developed for the purpose of semantic text processing, and may successfully be used in order to transform unstructured text into a structured data matrix. The use of VSMs in text analysis is a very specialised field. Hence, in Section 2.3 we only briefly discuss the use of VSMs in order to shed some light on the way in which text can be used as input in a CB recommender.

After converting unstructured data to a structured item representation, as shown in Figure 2.2, the resulting data matrix may be passed on to the Profile Learner component. Consider an active user u_a. Using items $\mathcal{I}_{u_a}$ rated by u_a, an initial user profile may be constructed. In more detail, for a given user

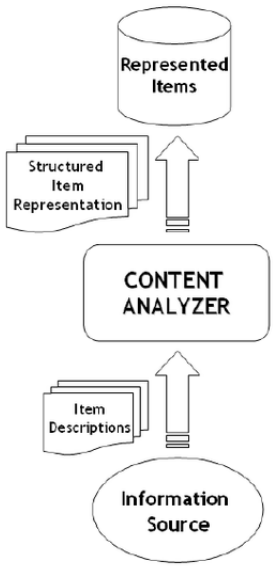


Figure 2.2: The output of the Content Analyser.

Source: Desrosiers and Karypis (2011).

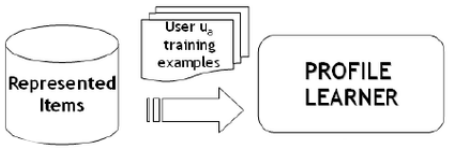


Figure 2.3: The input of the Profile Learner.

Source: Desrosiers and Karypis (2011).

u_a we have available a training set consisting of ratings on K items. That is, we have the item-rating pairs $T_{ak} = \{(i_{ak}, r_{ak}),\ k = 1, 2, \ldots, K\}$, where K is the total number of items rated by u_a.

A user's preference may be inferred based upon items he/she has previously rated. The Profile Learner receives represented items that have been rated by a given user (Figure 2.3), and applies supervised learning techniques to build a predictive profile (or model) for the user. These techniques include, but are not limited to, linear classifiers, Naive Bayes classifiers, decision trees, k-nearest neighbour algorithms, and relevance feedback. In Section 2.4, we consider how the user profile is learned using a linear classifier. Once complete, the learned profile is stored in the Profiles Archive, as depicted in Figure 2.4.

The user profiles, together with items that have not yet been rated by the active user u_a, are finally passed to the Filtering Component (Figure 2.5). This component is used to suggest items to the user that he/she may find interesting. This is done by comparing the learned profile to features of the new represented items, and by determining which of the new items are the most similar to the learned profile. A list of items is suggested to the user, ordered according to their preference, as deduced by the Filtering Component.

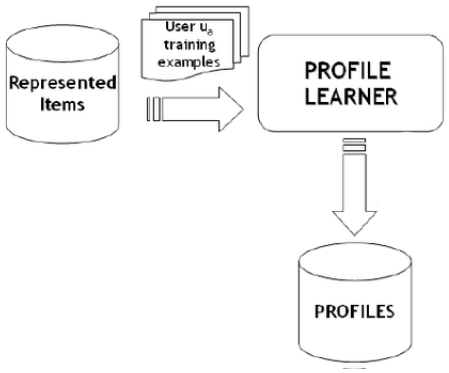


Figure 2.4: The output of the Profile Learner.

Source: Desrosiers and Karypis (2011).

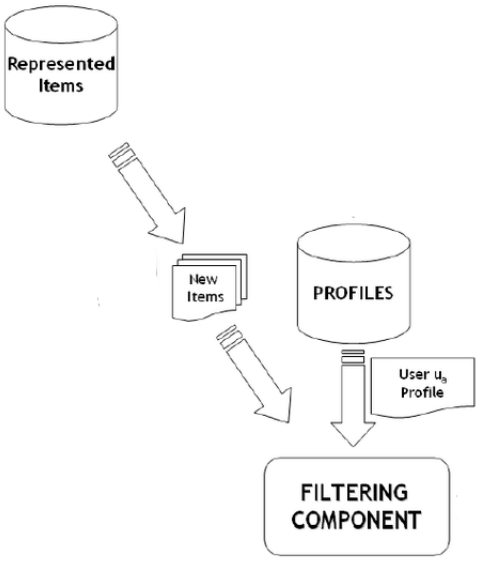


Figure 2.5: The input of the Filtering Component.

Source: Desrosiers and Karypis (2011).

Of course the taste of a user may change over time. Therefore, in order to ensure that recommendations are as accurate as possible, the user profile should continually be updated. For this purpose, feedback may (either implicitly or explicitly) be acquired and used by the Profile Learner in order to update the profiles.

Finally in this section, the individual components of a content-based recommender system are combined in Figure 2.6, thereby conveying its general architecture.

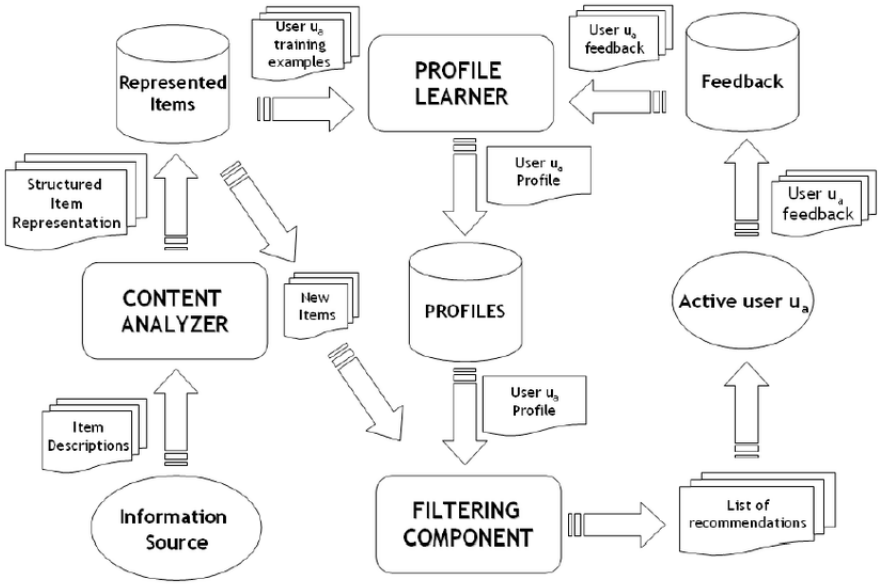


Figure 2.6: Architecture of a Content-based Recommender System.

Source: Desrosiers and Karypis (2011).

2.3 Item Representation in Content-based Recommenders

Given a set of items, together with a user, a content-based recommender system needs to find similar (or other relevant) items for the user. As mentioned, it is often the case with content-based recommenders that items are not represented by a defined set of features, but rather by textual descriptions. Therefore, the Content Analyser needs to convert the textual features into a usable format. In this section, we describe how this process may be carried out.

A traditional feature extraction technique used in information retrieval to achieve a structured representation of textual data is a vector space model. A VSM is an algebraic model that is used to spatially represent a set of items, and can assist an RS in determining the similarity between items. This is achieved in two steps, *viz.* a pre-processing step, followed by the calculation of term weights.

Phrases, keywords or single words used to describe an item are referred to as 'terms'. During the preliminary processing of items, all punctuation, special characters and stop-words in the item descriptions are removed. A so-called bag-of-words representation is then used to represent the items, and each unique word used to describe the items is used as a feature. In other words, each item is subsequently represented as a vector of words (terms). For each item, the presence or absence of a particular word can then be indicated either by the number of times that it appears in the item description, or by a Boolean value.

Next, the term vectors are converted into a usable numerical format, known as a vector of term weights. These term weights indicate how closely each term is related to an item. The frequency distribution of terms within an item description, as well as within the entire collection of terms used to describe all the items, plays an import role in determining how meaningful a term is.

A commonly used method for calculating term weights is Term Frequency Inverse Document Frequency (TF-IDF) (Turney and Pantel, 2010). As explained by Lops *et al.* (2011), TF-IDF is based on the characteristics of text documents. A frequently occurring word in an item description (Term Frequency (TF)) is more closely related to the item if it occurs infrequently throughout all the item descriptions (Inverse Document Frequency (IDF)). Before expanding on how term weights are calculated in TF-IDF, we first import some notation, following the notation introduced by Lops *et al.* (2011).

Let the set of item descriptions be denoted by $\mathcal{I} = \{\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_m\}$, and let the entire set of unique terms found in $\mathcal{I}$ after pre-processing the item descriptions be given by $T = \{t_1, t_2, \ldots, t_{|T|}\}$. The n-dimensional item vectors are denoted by $\mathbf{i}_j = \{w_{1j}, w_{2j}, \ldots, w_{nj}\}$, where w_{kj} indicates the weight for term t_k, $k = 1, 2, \ldots, |T|$, in item $\mathbf{i}_j$, $j = 1, 2, \ldots, m$. Additionally, let f_{kj} be the number of times that term t_k appears in the description of item $\mathbf{i}_j$. Note that f_{kj} is referred to as the *raw count* of term t_k.

The term frequency of an item may take on a number of forms. The simplest form is to directly use the raw count as the term frequency. The standard term frequency of item $\mathbf{i}_j$ is calculated by dividing the raw count f_{kj} by the number of terms in the item description, as is given in Equation 2.1.

$$\text{TF}(t_k, \mathbf{i}_j) = \frac{f_{kj}}{\sum_{z \in \mathbf{i}_j} f_{zj}}. \tag{2.1}$$

However, using Equation 2.1 alone allows for the possibility of items with shorter descriptions being ignored in favour of items with longer ones. This is because longer descriptions will contain a larger number of words, potentially allowing a particular word to have a higher count, regardless of its importance in the item description. Subsequently, we may want to rather make use of the so-called augmented frequency, given by Equation 2.2.

$$\text{TF}(t_k, \mathbf{i}_j) = 0.5 + 0.5 \cdot \frac{f_{kj}}{\max_z f_{zj}}. \tag{2.2}$$

Intuitively, a term that occurs repeatedly throughout all item descriptions is uninformative. Therefore we want small weights to be associated with frequently occurring terms, and large weights to be associated with rare terms. In order to achieve this, we scale the TF using the IDF. Let n_k denote the number of item descriptions in which the term t_k appears at least once. Thus it follows that

$$\text{IDF}(t_k) = \frac{N}{n_k}, \tag{2.3}$$

where N is the total number of item descriptions. Scaling the TF by the factor in Equation 2.3 is, however, quite severe. In order to dampen the effect of the IDF terms, we instead take the logarithm of the IDF term. Since the logarithmic function is a monotonically increasing function, we have that $\log(\frac{N}{n_k})$ is non-negative whenever $\frac{N}{n_k}$ is non-negative. To rephrase, $\frac{N}{n_k} \geq 1$ implies that $\log(\frac{N}{n_k}) \geq 0$. Therefore, the TF-IDF function is given by

$$\text{TF-IDF}(t_k, \mathbf{i}_j) = \text{TF}(t_k, \mathbf{i}_j) \log\left(\frac{N}{n_k}\right). \tag{2.4}$$

As a term occurs more frequently in the set of item descriptions (i.e. as n_k increases), $\frac{N}{n_k}$ will tend to one. This will in turn cause the IDF (and thus also the TF-IDF) to approach zero.

When using Equation 2.2 to compute term frequency, it is possible that the frequency f_{kj} of term t_k in item $\mathbf{i}_j$ is such that $f_{kj} = \max_z f_{zj}$. In this case,

$$\begin{aligned} \text{TF-IDF}(t_k, \mathbf{i}_j) &= \log(\frac{N}{n_k}) \\ &= \text{IDF}(t_k). \end{aligned} \tag{2.5}$$

Equation 2.4 is often used to directly obtain the term weights. However, it is not uncommon to go one step further and normalise this equation. Normalisation ensures that $w_{ij} \in [0, 1]$ for all $i = 1, 2, \ldots, n$, and $j = 1, 2, \ldots, N$. Thus the final weight w_{kj} for term t_k in item $\mathbf{i}_j$ is calculated using Equation 2.6, where $|T|$ is the total number of words in set of item descriptions.

$$w_{kj} = \frac{\text{TF-IDF}(t_k, \mathbf{i}_j)}{\sqrt{\sum_{s=1}^{|T|} \text{TF-IDF}(t_s, \mathbf{i}_j)^2}}. \tag{2.6}$$

A large value for w_{kj} indicates that the word t_k is a particularly important distinguishing word in the description of item $\mathbf{i}_j$. That is, the word t_k occurs frequently in $\mathbf{i}_j$, but not throughout the entire collection of item descriptions. Small values imply that the word t_k appears frequently throughout the collection of item descriptions, and is therefore not a distinguishing word in the item description.

2.4 Learning A Profile

Recall that in a CB recommender system, each user profile is learned in isolation. That is, the learning process does not exploit similarities between users to assist in the recommendation process. As mentioned in Section 2.2, various supervised machine learning algorithms can be employed in order to construct a user profile, based on the contents of items rated by the user. When items are rated by users using a numeric scale, we view the recommendation process as a regression task. A user's interests are modelled by a function learned by a machine learning algorithm. Once learned, the model (or user profile) is used to predict the user's ratings of unseen items. Items are sorted according to their predicted ratings, and the items with the highest predicted ratings are recommended to the user. In this section, for a simple explanation, we focus our attention on linear regression as a means of learning a user profile.

In Section 2.3, we saw how item descriptions may be represented as vectors of word frequencies. The underlying assumption of using linear regression to predict ratings is that the ratings can be modelled as a function of the word frequency. Suppose we have a set of items $\mathcal{I}$, as well was the entire set of terms T found in $\mathcal{I}$ after pre-processing the item descriptions. Let $\mathbf{I}_u$ be an $n \times |T|$ matrix representing the n training items rated by user u. The ratings given to these n items by user u are represented in the

n-dimensional vector $\mathbf{y}$. The linear model relating the word frequency to the user ratings is given by

$$\mathbf{y} \approx \mathbf{I}_u \mathbf{w}^T, \tag{2.7}$$

where $\mathbf{w}$ is a $|T|$-dimensional row vector representing the coefficient (weight) of each word, and needs to be estimated. The n-dimensional vector of prediction errors for the model can be calculated using Equation 2.8.

$$PE = \mathbf{I}_u \mathbf{w}^T - \mathbf{y}. \tag{2.8}$$

The linear model achieves maximum predictive performance when Equation 2.8 is minimised or, equivalently, when the squared norm of the prediction errors are minimised. Therefore, we need to find $\mathbf{w}$ that minimises

$$OF = ||\mathbf{I}_u \mathbf{w}^T - \mathbf{y}||^2 + \lambda ||\mathbf{w}||^2, \tag{2.9}$$

where $\lambda > 0$ is regularisation parameter. The regularisation parameter λ ensures that the model does not overfit to the training data, and the optimal value of λ can be determined via cross-validation. The weight vector $\mathbf{w}$ is obtained by taking the gradient of Equation 2.9 with respect to $\mathbf{w}$ and setting it equal to zero. Let $\mathbf{I}$ be a $|T| \times |T|$ identity matrix, then we have that

$$\begin{aligned} {\mathbf{I}_u}^T(\mathbf{I}_u \hat{\mathbf{w}}^T - \mathbf{y}) + \lambda \hat{\mathbf{w}}^T &= \mathbf{0} \\ \implies (\mathbf{I}_u^T \mathbf{I}_u + \lambda \mathbf{I}) \hat{\mathbf{w}}^T &= \mathbf{I}_u^T \mathbf{y} \qquad (2.10) \\ \implies \hat{\mathbf{w}}^T &= (\mathbf{I}_u^T \mathbf{I}_u + \lambda \mathbf{I})^{-1} \mathbf{I}_u^T \mathbf{y}. \qquad (2.11) \end{aligned}$$

Since $(\mathbf{I}_u^T \mathbf{I}_u + \lambda \mathbf{I})$ is a positive definite matrix, it is invertible. Therefore, Equation 2.11 follows from Equation 2.10.

Let $\mathbf{X}_u$ be an $m \times |T|$ matrix representing the m test items for user u. Furthermore, let $\hat{\mathbf{w}}_u$ be estimated weight vector for user u. The ratings of these items can be predicted via Equation 2.12. The user model can then be evaluated using, for example, the RMSE.

$$\hat{\mathbf{y}} = \mathbf{X}_u \hat{\mathbf{w}}_u^T. \tag{2.12}$$

Since a model is needed for each user in content-based recommendation, this process will be repeated for each user in the system.

2.5 Advantages of Content-based Recommenders

We have seen that content-based recommenders rely on the content of items rated by a user in order to learn a profile for that user. Each item is described by the same set of features, and recommendations

are made based on the similarity between items. This is advantageous in the sense that the RS can recommend items that have not been rated by any user before.

The user profile is based on items rated by the user, therefore the user knows exactly why an item is recommended to them. Thus, CB systems are transparent in their recommendations and users are more likely to trust the recommendations given to them. Another advantage of using only the active user's ratings when creating a profile is user independence, meaning that the system does not need to rely on information from other users. In other words, even in the case of an item that has never been rated before, if that item's features match those in the user profile, it may be recommended to the active user.

2.6 Limitations of Content-based Recommenders

One of the most common issues of recommender systems is the so-called *new user* or *cold-start* problem (Burke, 2002). In order for the learned profile to accurately represent the user's preferences, a sufficient number of items need to have been rated by the user. A new user will not have any rated items, and is therefore often prompted by the RS to give initial ratings to a set of initial items. However, due to the lack of training data, initial suggestions by the RS may not be applicable to the user.

Since the features of items are used in learning techniques employed by CB recommenders, the quality of the features is essential (Burke *et al.*, 2011). A learned profile will only be representative of the user if the items are described by detailed, key attributes. However, it is often difficult to acquire attributes for items. Furthermore, since a user profile has to be learned for each user in a CB system, content-based recommenders do not scale well with the number of users. Finally, a CB recommender can be subject to over-specialisation. This means that items are only recommended if they are very similar to ones already rated by a user (Iaquinta *et al.*, 2008). Therefore, content-based recommenders tend to only present a narrow selection of recommendations to the user, and generally do not perform well in terms of the novelty and diversity of recommended items.

2.7 Summary

In this chapter, we discussed the way in which content-based recommenders aim to identify items with equivalent features to the items that a user has previously enjoyed. The architecture of a CB system, and the steps taken to make recommendations, were described. These steps include embedding unstructured features into a vector space, creating a user profile and finding items that are similar to a user profile. In terms of the advantages and limitations of CB recommender systems, it was noted that while these systems are transparent in terms of their recommendations and do not suffer from the new item problem, they generally do not supply novel or diverse recommendations.

Chapter 3

Collaborative Filtering Recommender Systems

3.1 Introduction

Unlike a content-based recommender system, a collaborative filtering recommender system does not make use of the content of items to make suggestions to a user. Instead, CF systems take advantage of a given user's available rating history, together with the history of other users in the system, to determine the relevance of an item to an active user. CF recommender systems are based on the assumption that if two users have purchased or rated the same item, they are likely to share similar interests (Jannach *et al.*, 2010). In other words, CF systems use the 'wisdom of crowds', or *collaboration* among users to make predictions for an active user.

The idea of CF was presented by Goldberg *et al.* (1992), the authors of the first commercial recommender system called Tapestry. In this system, documents are recommended to users from a collection of electronic documents. The documents are sourced from email or news wire stories, for example. Users annotate the documents, which then allow the documents to be filtered according to preferences.

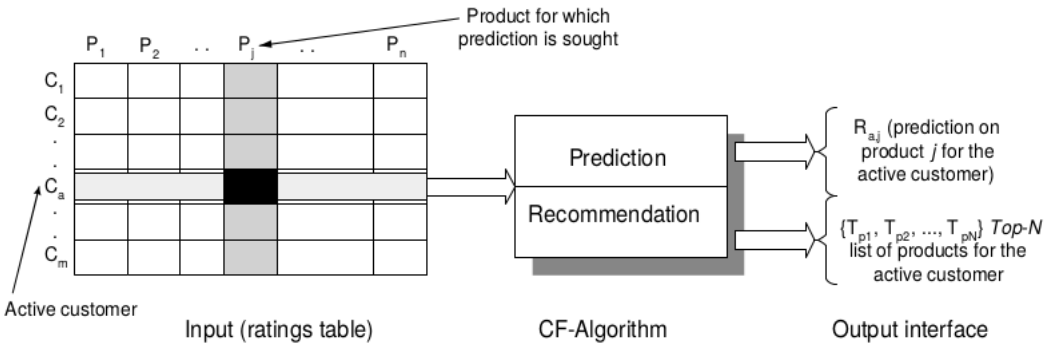


Figure 3.1: Collaborative Filtering Process.

Source: Sarwar *et al.* (2002).

A high-level overview of the way in which a collaborative filtering system can make recommendations

is depicted in Figure 3.1. The input to a CF system is a data matrix consisting of ratings allocated by the users to items which they have bought or viewed. These inputs in a so-called ratings matrix $\mathbf{R}$, are illustrated in the left part of Figure 3.1. Note that the rows $\{C_i, i = 1, 2, \ldots, m\}$ represent users (or customers), and the columns $\{P_j, j = 1, 2, \ldots, n\}$ represent items (or products in this application). The black square indicates that we are attempting to predict the rating P_j that user C_a would give to the item j. An algorithm can be applied to the data to acquire this prediction, and outputs a recommendation (or a list of recommendations) to the user. In other words, given a ratings matrix $\mathbf{R}$, a collaborative filtering system aims to estimate the two-dimensional rating function h_R given by

$$h_R : \mathcal{U} \times \mathcal{I} \to \mathcal{S}, \tag{3.1}$$

where $\mathcal{U}$ is the set of users, where $\mathcal{I}$ is the set of items, and where $\mathcal{S}$ is the set of possible values for the ratings.

In this chapter, we explore the two approaches that CF recommender systems can apply to make predictions, namely the memory-based (or neighbourhood-based) approach and the model-based approach (see Figure 3.2). In the former approach, the objective is to directly predict the relevance of an item to a user based on stored ratings. Since no assumptions regarding the functional form of h_R are made, the memory-based approach is a non-parametric approach. The two methods that can be used to do this (called user-based and item-based recommendation) form the focus of Section 3.2. On the other hand, in a model-based approach, we make an assumption regarding the functional form of h_R. Therefore, a model that is able to accurately predict missing item ratings needs to be learned. This approach is introduced in Section 3.3, and we continue the discussion in more detail in Chapter 4. Memory-based and model-based CF are compared in Section 3.4, followed by a comparison of content-based and collaborative filtering in Section 3.5. We summarise the chapter in Section 3.6.

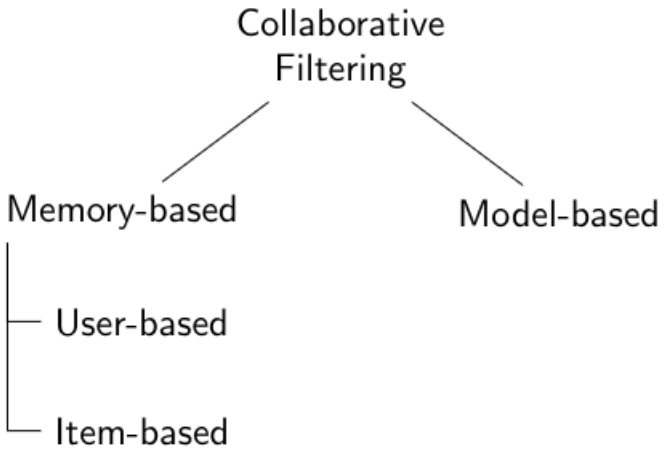


Figure 3.2: Collaborative Filtering Approaches.

3.2 Memory-based Collaborative Filtering

Similarities between items or users play a pertinent role in memory-based CF recommender systems. In the memory-based approach, similar items (or users) are identified. The rating function h_R, given in Equation 3.1, is typically estimated using the k-nearest neighbour algorithm. Therefore the memory-based approach is often referred to as a neighbourhood-based approach and, as the name suggests, similarities are calculated in-memory (Levinas, 2014).

Su and Khoshgoftaar (2009) separates the memory-based CF algorithm into three steps. The similarity between users (or items) are first calculated. Using these similarities, the ratings of new items are then predicted for an active user. This is followed by producing the recommended items that the user is most likely to enjoy. Depending on whether similarities between users or items are obtained, one may distinguish between user-based or item-based collaborative filtering. Stated very simply, the explanations associated with a recommendation from each method are:

User-based: "Similar users to you also liked..."
Item-based: "Users who liked this item also liked..."

In other words, the goal of the user-based filtering is to identify users with similar preferences to the active user and to base the rating prediction on the ratings provided by these users, while item-based filtering finds similar items to the target item and bases the rating prediction on these item ratings.

A well-known example of a user-based CF system is the music recommender Ringo, which employs the user-based technique to make recommendations (Shardanand and Maes, 1995). More recently, the CF approach Effective Trust (Guo *et al.*, 2015), extends the user-based method to allow users to specify other users that they trust. This extension resulted in improved recommendation performance over that of a user-based recommender system. An example of an item-based CF approach is the Slope One approach (Lemire and Maclachlan, 2005). This method considers the differences between the ratings of items for users when predicting a new item.

3.2.1 User-based Filtering

In user-based filtering, we assume that users can be clustered into groups based on their preferences (Deshpande and Karypis, 2004). Similarities of users are based on the items they have consumed, as well as on how they have responded to them. To recommend items to the active user, his or her nearest neighbours (that is, the group of users he or she is most similar to) are identified. Based on the ratings given to items by the active user's neighbours, new items are suggested by predicting the rating the active user would give to these items.

More formally, consider an $m \times n$ ratings matrix $\mathbf{R}$, wherein the rating r_{ui} is missing since item i has not yet been experienced by user u. Our aim is therefore accurate prediction of rating r_{ui}. We start by identifying the k users most similar to user u, and denote this set of neighbours by $\mathcal{N}^k(u)$. The value of k can either be pre-specified, or an optimal value can be determined by means of a grid-search. Next, we make use of the ratings for item i, given by the neighbours of user u, in order to estimate r_{ui}. That is, we base the prediction of r_{ui} on the set of ratings $\{r_{\tilde{u}i}, \tilde{u} \in \mathcal{N}^k(u)\}$.

In order to obtain $\mathcal{N}^k(u)$ for each user, of course some similarity measure is needed. There are many options in this regard. Arguably, two of the most frequently used similarity measures are the cosine similarity or the Pearson correlation. Let s_{uv} denote the similarity between user u and user v. Then, the s_{uv}-value according to the Pearson correlation is given by

$$s_{uv} = PC(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}}. \tag{3.2}$$

In Equation 3.2, $\mathcal{I}_{uv}$ indicates the items rated by both users u and v. Also, the mean rating of user u is defined as

$$\bar{r}_u = \frac{1}{|\mathcal{I}_u|} \sum_{k \in \mathcal{I}_u} r_{uk},\ u = 1, 2, \ldots, n, \tag{3.3}$$

where $|\mathcal{I}_u|$ denotes the number of items rated by user u. Since Pearson correlation takes into account the differences in mean and variance of the ratings given by the users u and v, the effects of users having different rating habits are reduced. By subtracting the mean ratings, and by dividing by the standard deviations, the potential of some users rating items more generously than other users, does not influence the similarities between users as much.

In a regression setting, once the similarities between user u and all other users have been obtained, the predicted rating $\hat{r}_{ui}$ is calculated quite simply as the average rating given to item i by the users in $\mathcal{N}^k(u)$. That is,

$$\hat{r}_{ui} = \frac{1}{|\mathcal{N}_i^k(u)|} \sum_{v \in \mathcal{N}_i^k(u)} r_{vi}, \tag{3.4}$$

where $\mathcal{N}_i^k(u) \subseteq \mathcal{N}^k(u)$ denotes the subset of neighbours of u who have rated item i. A variation of Equation 3.4 is a weighted rating which takes the similarity between user u and user v into account. This implies that a rating given to item i by a user who is very similar to u will be weighted more heavily than a rating given to i by a user with dissimilar interests to u (Schafer *et al.*, 2007). The weighted rating is calculated as follows:

$$\hat{r}_{ui} = \frac{\sum_{v \in \mathcal{N}_i^k(u)} r_{vi} s_{uv}}{\sum_{v \in \mathcal{N}_i^k(u)} |s_{uv}|}. \tag{3.5}$$

As discussed in Section 1.1, ratings can be categorical, in which case we would use classification to predict the ratings. In this setting, we predict the rating r_{ui} by

$$\hat{r}_{ui} = \arg\max_{r \in \mathcal{S}} \sum_{v \in \mathcal{N}_i^k(u)} \delta(r_{vi} = r) s_{uv}, \tag{3.6}$$

where

$$\delta(r_{vi} = r) = \begin{cases} 1 & r_{vi} = r \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

and where $\mathcal{S}$ is the set of possible (categorical) ratings.

Once the ratings r_{ui} have been predicted for all items $i \in \mathcal{I} \setminus \mathcal{I}_u$, new items may be recommended to the user. In either the regression or classification setting, the items recommended to user u are those for which the predicted ratings $\hat{r}_{ui}$ have the highest values.

3.2.2 Item-based Filtering

Item-based filtering and user-based filtering are very much alike. However, instead of finding other users who are similar to user u, the recommender system distinguishes items that are similar to the items rated by user u. The idea of item-based filtering is to predict the rating r_{ui} that user u would give to a target item i, based on the ratings that user u gave to items that that are similar to item i. Hence, whereas user-based filtering is based upon similarities between users (or rows of the ratings matrix), item-based filtering is based upon similarities between items (or columns of the ratings matrix). Therefore, similarities in item-based filtering can be determined using Equation 3.2, modified to calculate the similarity between two items, i and j, instead of between two users. In this case, the Pearson correlation is thus given by

$$s_{ij} = PC(i,j) = \frac{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2 \sum_{u \in \mathcal{U}_{ij}} (r_{uj} - \bar{r}_j)^2}}, \tag{3.8}$$

where $\mathcal{U}_{ij}$ indicates the set of users that have rated both items i and j. The mean rating of item i and item j is given by $\bar{r}_i$ and $\bar{r}_j$ respectively, and can be computed using a similar equation to Equation 3.3.

The k items rated by user u that have the highest similarity s_{ij} to item i, form a neighbourhood, denoted by $\mathcal{N}_u^k(i)$. These similarities are used to determine a weighted prediction r_{ui} for item i, and then items with the largest ratings are recommended. The predicted ratings are evaluated using Equation 3.4 or Equation 3.5, where both are adjusted in order to determine similar items rather than similar users.

3.2.3 Comparison of Item-based and User-based Models

In constructing a memory-based collaborative filtering system, the choice between user-based and item-based filtering depends largely on the stability of the system. As noted by Aggarwal (2016), the stability of

a recommender system is dependent on the amount by which the number of users and items in the system changes, as well as how on often these changes occur. A user-based method would be more beneficial in the situation where the number of items changes, while the number of users remains relatively constant. This is due to the fact that user similarities would not have to continually be recomputed. Contrarily, when the number of users frequently changes, an item-based approach is desirable (Levinas, 2014).

3.3 Model-based Collaborative Filtering

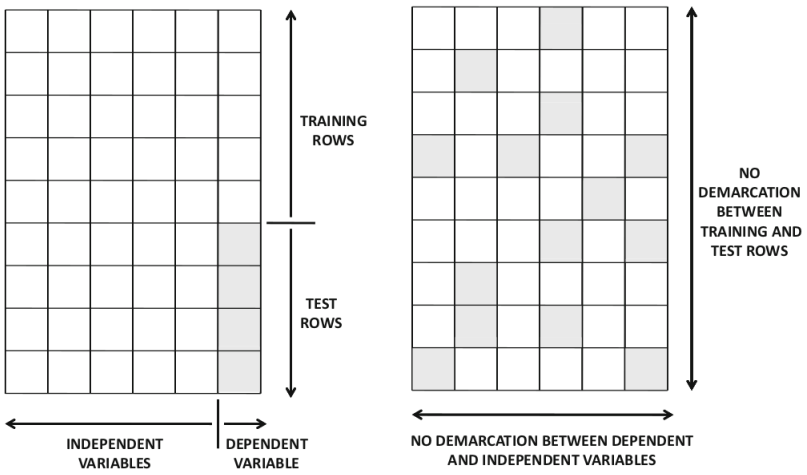
In Section 3.2 we discussed memory-based collaborative filtering, and the two approaches thereof. To make recommendations, these techniques require that all of the information regarding users and items be stored in-memory. All of the available data cases are utilised each time a prediction is made, and hence these methods are often known as instance-based algorithms.

Model-based CF differs from memory-based CF in the sense that the available data is encapsulated in a model. Each time a prediction is made, this previously learned model is applied. This is comparable with machine learning approaches for classification (or regression) problems, and accordingly, we can consider model-based CF as a generalisation of these problems. In the remainder of this section, we describe the relationship between model-based CF and machine learning approaches. In Chapter 4 we present some of the well known model-based CF approaches, as well as more advanced techniques.

With reference to Figure 3.3, the classification setting and collaborative filtering setting can be compared and contrasted. Figure 3.3 (a) depicts the data for a standard task of classification, which consists of a matrix of dimension $m \times (p + 1)$. We see that the features (independent variables) are specified by the first p columns, and that there are no missing entries. Observations of the response (or dependent variable) is contained in column $(p+1)$. Since it is a classification task, the response will specify the class to which the object belongs. Typically, before modelling, the dataset is split into a training and test set, as shown. The grey blocks indicate that the response values for the test cases are missing. Once a model has been established from determining patterns in the training data, the missing response values of the test data can be predicted.

We noted in the beginning of Chapter 3 that we use a ratings matrix $\mathbf{R}$ as input for a collaborative filtering recommender system. This $m \times p$ matrix is represented in Figure 3.3 (b), with rows indicating users, and columns indicating the ratings given to items viewed/experienced by the users. Since users have not rated all possible items, there are missing values throughout the matrix. We do not wish to predict the class to which the user belongs, but the value the user might give to an item. The implication is that we do not have distinct dependent and independent variables. Furthermore, as indicated in Figure 3.3 (b), there is no clear demarcation between the training and test data as in the traditional classification setup.

Based on these observations, we clearly see that the model-based CF approach is a generalisation of the



(a) Classification (b) Collaborative filtering

Figure 3.3: Data entries in a classification and collaborative filtering setting.

Source: Aggarwal (2016).

classification (or regression) problem, thus the machine learning algorithms used in the latter task can be modified to estimate the rating function h_r given in Equation 3.1. Some of the common approaches to model-based CF are decision trees, SVD or matrix factorisation (which we discuss in Chapter 4), as well as neural networks. The performance of the models can be evaluated using, among others, RMSE, MAE, precision and recall (*cf.* Section 1.1).

3.4 Memory-based versus Model-based Collaborative Filtering

Memory-based and model-based collaborative filtering techniques were discussed in Section 3.2 and Section 3.3, respectively. We now examine the limitations, as well as the merits, of each of these methods.

Memory-based techniques are deemed to be more simple than model-based techniques. Memory-based methods are not only intuitive, but they are also relatively uncomplicated to implement. When using this type of collaborative filtering approach, the only "parameter" that needs to be specified is the number of users (or items) to consider when forming neighbourhoods. Furthermore, unlike model-based techniques, there is no training aspect. This reduces the cost of memory-based methods, since machine learning models generally need to be retrained as more data becomes available, rendering them less efficient (Levinas, 2014).

If the dataset is small, it is possible for memory-based techniques to provide virtually instantaneous recommendations. The neighbours can be pre-determined offline and, since they do not require large amounts of memory, they can be stored. Typically, a user-based method will have $O(n^2)$ space complexity, where n is the number of users, and an item-based method will have $O(m^2)$ space complexity, where m is the number of items. Thus, when the dataset increases in size, there is a much greater memory

usage, accompanied by a decrease in performance. It can therefore be argued that although model-based techniques may take longer to train, they have a much lower memory requirement. This is because the learned model summarises the data, and is much smaller than the ratings matrix.

Data sparsity can have a significant impact on collaborative filtering systems in terms of the difficulty of generating predictions, and in terms of the accuracy of the resulting recommendations. Herlocker *et al.* (2004) define coverage to be the proportion of items for which an RS is able to generate predictions for. In the situation where the number of available items is large and the number of users is small, it is unlikely that any given user has rated all items, or even a large majority of them. This data sparsity means that the coverage of the system is reduced. In a memory-based CF system, the number of common items between users is therefore limited, and similar users may be overlooked. In a model-based CF system, the lack of ratings means that there is not a sufficient amount of information for an effective model to be learned. For both memory-based and model-based CF, data sparsity is highly likely to negatively impact the accuracy of recommendations, and user satisfaction may be diminished. Thus, one of the main challenges of CF is the fact that the ratings matrix is often sparse. In Chapter 4, we explore latent factor models for collaborative filtering. These dimensionality reduction methods can be used to represent sparse data in a more favourable manner, and as a result, more accurate predictions can be made in the case of sparse data.

An increase in the number of items and users in a system directly affects the scalability of the system. Neighbourhood methods especially do not scale well with an increase in the dimension of the ratings matrix. Linden *et al.* (2003) propose addressing the scalability problem by reducing the size of the dataset. This can be done by randomly sampling users, or by removing users who have not provided a satisfactory number of ratings. Further dimension reduction techniques include SVD and Principal Component Analysis (PCA).

3.5 Content-based versus Collaborative Filtering

We have already examined a number of drawbacks to content-based recommender systems in Section 2.6. These included over-specialisation, the scalability problem, as well as poor feature quality. In this section, we highlight some ways in which collaborative filtering recommender systems can overcome the drawbacks of CB systems. Scenarios in which a CB system may be preferred are also mentioned.

While a CB systems relies on the content of items, a CF system completely ignores any content that may be available. Instead, CF system make use of the ratings provided by users. This means that a CF system is able to recommend items even when features are poorly described. CF recommenders also have the potential to be applied to a very wide variety of domains (Felfernig and Burke, 2008). Furthermore, items can still be recommended even when the content differs from previously seen items, or when content

is not available. However, the dependence of collaborative filtering systems on the ratings matrix can be detrimental. As a result of this reliance, recommendations given by a CF recommender can be unreliable in situations where there is not a substantial amount of ratings data available, and therefore when the ratings matrix is sparse. As noted by Felfernig and Burke (2008), the cold-start problem often hinders the success of more advanced machine learning methods. The acquisition of user feedback therefore forms a crucial step in the recommendation process. Users, however, often find the task of providing feedback tedious. Furthermore, Resnick and Varian (1997) note that it may cause concern for privacy as users do not want their preferences and habits tracked.

Content-based systems would be preferred to collaborative filtering systems in situations where the opinion density of the items is low. For example, not many people have owned multiple cars or houses in order to give their opinion on them, and therefore the rating matrix in these situations would be very sparse. Another issue with memory-based CF systems is that neighbourhoods cannot be formed between users unless they have rated very similar items (Balabanović and Shoham, 1997). CB systems do not have this issue since they have access to the content of items. However, the recommendations provided by a CB recommender do become noisy when a user has given feedback on only a small number of items (Burke *et al.*, 2011), and when, as a consequence, the user profile cannot accurately learn the users' preferences.

Contrary to content-based recommenders, collaborative filtering systems cannot recommend an item to a user unless some other user has rated it in the past. CB recommenders avoids the cold-start problem by studying the content of an item and comparing it to the learned user profile. The disassociation from other users in content-based systems can be advantageous. This is because it is easier for users to understand the recommendations that have been made as they are based only on their own preferences, independent of others.

Finally, we note that an active user whose tastes are vastly different to the other users in the system will not receive meaningful recommendations in a collaborative filtering system. The lack of similarities between users leads to a small neighbourhood and in turn a limited number of items to recommend.

3.6 Summary

In this chapter, we explored how collaborative filtering systems examine the rating history of users to make recommendations. The two approaches through which collaborative filtering can be carried out, namely memory-based and model-based filtering, were discussed. Additionally, we described the two subdivisions of memory-based filtering known as user-based and item-based filtering.

In our discussion, we compared and contrasted content-based and collaborative filtering recommender systems. From this, we noted that one of the key issues of CF is that the sparsity of the ratings matrix severely impacts the accuracy of the recommendations. Thus, dimensionality reduction on the ratings

matrix may serve to improve prediction accuracy. We therefore consider some of these techniques, known as latent factor models, in Chapter 4.

Chapter 4

Latent Factor Models

4.1 Introduction

The two most prominent methods of recommendation were discussed in Chapter 2 and Chapter 3, namely content-based and collaborative filtering, the latter being the most successful (Ricci *et al.*, 2011). The CF techniques capture relationships between either users or between items, based on ratings that users provide. Since these techniques do not rely on the the content of items (as in content-based recommendation), they are not restricted to specific domains. However, as discussed in Chapter 3, collaborative filtering suffers from the cold-start problem since the ratings matrix is typically sparse.

An extension to model-based CF which aims to address the data sparsity problem is the use of latent factor models. By applying these factorisation models to the observed ratings matrix, a latent representation of the users and items, based on the provided ratings, may be found. Hence, the foundation of latent factor models is the assumption that the relationship between users and items can accurately be encapsulated by representing users and items in a lower dimensional space (Sammut and Webb, 2010). Following an LFM approach, essential information regarding the users and items is preserved, while reducing the dimension of the data. In this way, latent factor models are comparable to dimensionality reduction methods such as singular value decomposition. However, there is a fundamental difference between LFMs and dimensionality reduction techniques. Methods such as SVD are typically employed as precursors to model-based methods that are used to predict missing ratings (Aggarwal, 2016). Latent factor models, on the other hand, aim to directly predict the missing ratings in the ratings matrix by means of the determined latent factors. This is advantageous as it eliminates the need to apply further learning methods in order to predict the ratings. This chapter is dedicated to various model-based approaches used in CF to alleviate the data sparsity problem.

Dimensionality reduction is a common method used to diminish the sparsity problem (Shani and Gunawardana, 2011). Therefore, in Section 4.2, we present a brief discussion of SVD as a well-known dimen-

sionality reduction technique. Since SVD can only be applied to dense matrices (that is, matrices with no missing values), missing ratings must first be estimated in order to render this technique applicable to the recommendation problem (Shani and Gunawardana, 2011). A few methods that have been used to estimate the missing values, are discussed.

In Section 4.3, we then move on to established LFMs that are used to directly estimate missing ratings, thereby addressing the data sparsity problem of the ratings matrix. We start with a discussion of Matrix Factorisation (MF) and of tensor factorisation models. MF is a direct extension of SVD and the most basic latent factor model (Shani and Gunawardana, 2011). Tensor factorisation models are further extensions of MF that allow for the inclusion of additional ratings information.

Factorisation Machines (FMs) are described in Section 4.4. An FM is a supervised learning algorithm that not only provides more accurate results than the basic latent factor models, but that can also be modified in order to mimic them (Rendle, 2010). One of the main advantages that FMs have over LFMs is that they allow for the easy incorporation of additional user and item information. Advancements of factorisation machines are discussed in Section 4.5, followed by a summary of the chapter in Section 4.6.

4.2 Singular Value Decomposition

In the previous chapter we have seen that the most commonly used feedback or information in a CF system is in the form of explicit ratings, stored in the ratings matrix, with the users and items representing different dimensions. SVD is a well-established dimensionality reduction technique used in information retrieval (Dhumal and Deshmukh, 2016). In recommender systems, SVD can be used to represent users and items in terms of a specified number k of latent factors.

Suppose that we have a fully specified matrix of ratings $\mathbf{R}$ of size $n \times m$, where n indicates the number of users and m indicates the number of items in a recommender system. SVD can be used to decompose $\mathbf{R}$ into an $n \times r$ matrix $\mathbf{P}$, an $r \times r$ diagonal matrix $\boldsymbol{\lambda}$, and an $m \times r$ matrix $\mathbf{Q}$, as follows:

$$\mathbf{R} = \mathbf{P}\boldsymbol{\lambda}\mathbf{Q}^T. \tag{4.1}$$

The diagonal matrix $\boldsymbol{\lambda}$ contains non-negative singular values λ_i, $i = 1, 2, \ldots, r$, arranged in decreasing order. Additionally, the columns of $\mathbf{P}$ and $\mathbf{Q}$ are constrained to be mutually orthogonal. By only considering the first $k < r$ singular values (that is, by setting $\lambda_i = 0$ for $i = k+1, k+2, \ldots, r$), $\mathbf{R}$ can be approximated by

$$\mathbf{R}_k = \mathbf{P}\boldsymbol{\lambda}_k\mathbf{Q}^T. \tag{4.2}$$

The matrix $\mathbf{P}\boldsymbol{\lambda}_k$ in Equation 4.2 represents the reduced, original $n \times r$ ratings matrix $\mathbf{R}$ in the reduced basis $\mathbf{Q}$. Therefore, $\mathbf{R}_k$ is a lower-dimensional representation of $\mathbf{R}$, known as the truncated SVD or the

rank-k approximation of $\mathbf{R}$, providing the minimum Frobenius norm between $\mathbf{R}$ and $\mathbf{R}_k$ (Hastie *et al.*, 2015).

However, when the ratings matrix contains missing information, SVD is undefined. In an RS, each user will have rated only a limited number of all the available items, or new items will not yet have any ratings. Due to this, the ratings matrix is sparse, and SVD (as well as its truncated form) cannot directly be applied (Koren *et al.*, 2009).

A number of methods exist to alleviate the data sparsity problem, such as using information obtained from crowd-sourcing (Hwang *et al.*, 2014; and Jang *et al.*, 2016), from a trust network (Chang *et al.*, 2016; and Lee *et al.*, 2013), or by means of data imputation (Hwang *et al.*, 2016; and Hastie *et al.*, 2015). Since no additional information is needed for data imputation, it is considered a relatively simple way to address the sparsity problem. In this approach, the missing ratings are estimated and inserted into the ratings matrix. Once the missing values have been estimated, SVD can be applied.

A straightforward method of data imputation, suggested by Sarwar *et al.* (2001), is to first form a new matrix consisting of non-personalised ratings. This is done using the average rating of an item, normalised by subtracting the average user rating. Then, a complete matrix is obtained by adding together the original sparse matrix and the matrix of non-personalised ratings. More complex data imputation techniques include a smoothing-based approached proposed by Xue *et al.* (2005), and a method that uses both user and item information to obtain the values to impute (Ma *et al.*, 2007). The former approach uses a combination of memory-based and model-based techniques to group similar individuals into clusters. These clusters are used to predict the unrated items of users with similar interests to users in their cluster, thereby filling in the missing values.

The approach by Ma *et al.* (2007) is based upon a neighbourhood of similar users, and a neighbourhood of similar items. A neighbourhood $S(u)$ of similar users to user u is formed, containing users whose similarity to user u lies above a pre-specified threshold η. Similarly, a neighbourhood $S(i)$ of similar items to item i contains items whose similarity to item i lies above a pre-specified threshold θ. Then, when a missing rating r_{ui} is predicted, four conditions are considered. Depending on which condition is applicable to user u and item i, a different formula is used to calculate the predicted value of r_{ui}. The benefit of this approach is that missing values are only estimated if it is deemed that it will have a positive influence on the resulting recommendation. The interested reader should consult the paper by Ma *et al.* (2007) for details regarding the conditions and corresponding formulae.

While data imputation methods are advantageous in the sense that no additional information is required, the method has two significant drawbacks (Koren and Bell, 2011; Lee *et al.*, 2018). Firstly, since imputation increases the amount of data, it is memory intensive. Secondly, SVD is prone to over-fitting if the missing values are not carefully and accurately imputed, resulting in a loss of generality. Therefore, in-

stead of relying on data imputation, factorisation models can be used. These models, which are discussed in the following section, make use of only the observed ratings in order to directly estimate the missing values.

4.3 Factorisation Models

A users' rating pattern is indicative of their item preferences. Therefore, it can be said that the observed ratings matrix captures the correlation between users and items. By exploiting these correlations, factorisation models (or latent factor models) simultaneously describe users and items using k latent factors, where k is pre-specified. The latent factors can subsequently be used to make recommendations. In other words, in contrast to neighbourhood based methods that use statistical similarity measures to generate recommendations, latent factor models assume that the data can be represented in a lower dimensional space. This space contains hidden factors, indicative of user preferences, which facilitate recommendations (Srebro and Jaakkola, 2003).

Unlike SVD, data imputation is not required before applying factorisation models. Therefore, an advantage of LFMs is that they can be directly applied to sparse matrices. Matrix factorisation forms the basis of latent factor models. Two factorisation models that we consider are the basic matrix factorisation model and a tensor factorisation model known as SVD++. These discussions are largely based on the papers by Koren *et al.* (2009) and Koren and Bell (2011).

4.3.1 Matrix Factorisation

For the sake of simplicity, we first describe the basic matrix factorisation model, under the assumption that all ratings are present. In this case, matrix factorisation is comparable to SVD. We then show how one may modify the model in order to accommodate sparsity, as well as how to introduce bias into the model. Finally, in this section, we describe a commonly used method to learn latent factor model parameters, namely Stochastic Gradient Descent (SGD).

A key idea underlying matrix factorisation is that a user's preferences are influenced by only a small number of factors. Therefore, it is presumed that a preference vector for each user can be created, where each element describes the way in which a factor relates to the user. Suppose we have a ratings matrix $\mathbf{R} : n \times m$, then matrix factorisation maps $\mathbf{R}$ to a lower dimensional latent space. That is:

$$\mathbf{R} \quad = \quad \mathbf{P}\mathbf{Q}^T, \tag{4.3}$$

where $\mathbf{P} : d \times k$, where $\mathbf{Q} : n \times k$, and where k is an integer that needs to be pre-specified. The main difference between SVD and matrix factorisation is that the latter is unconstrained, which means that the columns of $\mathbf{P}$ and $\mathbf{Q}$ do not have to be orthogonal (Aggarwal, 2016).

Equation 4.3 indicates that inner products can be used to express the interactions between users and items. Each user u is related to an *user factor* $\mathbf{p}_u^T \in \mathbb{R}^k$, given by a row of $\mathbf{P}$, and each item i is related to an *item factor* $\mathbf{q}_i \in \mathbb{R}^k$, given by a column of $\mathbf{Q}$. The elements in vector $\mathbf{q}_i$ indicate the degree to which item i is associated with each of the k factors. Similarly, the elements in vector $\mathbf{p}_u^T$ indicate the preference of user u towards items that are related to the corresponding factors. From Equation 4.3, it is clear that the rating of item i by user u can be approximated by

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i. \tag{4.4}$$

That is, a user's interest in an item can be expressed as the dot product between latent vectors and latent factors.

Of course, Equations 4.3 and 4.4 are only comparable to SVD when there are no missing values in the ratings matrix. Fortunately, there is a way to learn $\mathbf{p}_u$ and $\mathbf{q}_i$ when the ratings matrix is sparse. In this case, we can apply matrix factorisation on the observed ratings only. Thus, if we have available a set of training observations $\mathcal{K}$ with known ratings, the model parameters $\mathbf{p}_u$ and $\mathbf{q}_i$ are given by the vectors that minimise the regularised squared error loss in Equation 4.5. That is, we need to find $\mathbf{p}^*$ and $\mathbf{q}^*$ that minimise

$$\frac{1}{2} \sum_{(u,i)\in\mathcal{K}} \left[(r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)^2 + \lambda(||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2) \right], \tag{4.5}$$

where $||\cdot||^2$ denotes the squared Frobenius norm. Two methods that are commonly used to minimise Equation 4.5 are Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS).

A regularisation parameter $\lambda > 0$ is included in Equation 4.5 to avoid over-fitting. The value of the regularisation parameter may be determined using cross-validation. Regularisation is important as the ratings matrix is typically very sparse, which means that there are only a small number of observed ratings and therefore over-fitting is highly likely. The regularisation parameter discourages large elements in $\mathbf{p}$ and $\mathbf{q}$. This means that less complex solutions are preferred since large elements are penalised. The larger the value of λ, the greater the penalisation for large elements in $\mathbf{p}$ and $\mathbf{q}$. Therefore, regularisation may produce a more stable model.

Given a specific user u, their interest in an item i is approximated by Equation 4.4. In other words, $\hat{r}_{ui}$ models the interaction between users and items. It is, however, possible that independent user or item biases have an effect on the observed rating. For example, consider a user u and a movie i, and let the average rating of all the movies in the system be denoted by μ. If i is a popular movie, it is likely to, on average, be rated higher than other movies. Let the deviation of item i from the average rating μ is denoted by b_i. Suppose also that user u is a critical thinker, therefore u tends to rate movies lower than average. Let the deviation of user u from the average rating be denoted by b_u. Thus, incorporating the

biases b_i and b_u, the predicted rating is given by

$$\begin{aligned} \hat{r}_{ui} &= b_{ui} + \mathbf{p}_u^T \mathbf{q}_i \\ &= \mu + b_u + b_i + \mathbf{p}_u^T \mathbf{q}_i, \end{aligned} \tag{4.6}$$

and the model parameters b_u, b_i, $\mathbf{p}_u$ and $\mathbf{q}_i$ can be learned by minimising the following regularised objective function:

$$\frac{1}{2} \sum_{(u,i) \in K} \left[(r_{ui} - \mu - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i)^2 + \lambda (b_u^2 + b_i^2 + ||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2) \right]. \tag{4.7}$$

As with Equation 4.5, Equation 4.7 can be minimised using either stochastic gradient descent or alternating least squares. SGD is generally faster and easier to implement than ALS, and therefore it is more commonly used (Koren *et al.*, 2009). Thus, in the remainder of this section, we describe how SGD can be used to solve Equation 4.7.

Towards this end, note that the error between the observed rating and the predicted rating is defined as

$$\begin{aligned} e_{ui} &= r_{ui} - \hat{r}_{ui} \\ &= r_{ui} - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i. \end{aligned} \tag{4.8}$$

Hence, the first term in Equation 4.7, denoted below by J, is the sum of squared errors. That is:

$$\begin{aligned} J &= \frac{1}{2} \sum_{(u,i) \in K} \left[(r_{ui} - \mu - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i)^2 + \lambda (b_u^2 + b_i^2 + ||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2) \right] \\ &= \frac{1}{2} \sum_{(u,i) \in K} \left[e_{ui}^2 + \lambda (b_u^2 + b_i^2 + ||\mathbf{p}_u||^2 + ||\mathbf{q}_i||^2) \right]. \end{aligned} \tag{4.9}$$

The first step of stochastic gradient descent is to compute the gradient of J with respect to the unknown parameters that need to be learned. Then, for each observed training case, the parameters are incrementally updated by moving in the opposite direction to the gradient. This process is terminated as soon as the objective function J converges to a small enough value. To illustrate, we show how the parameter b_u can be updated using SGD, noting that all other parameters can be updated similarly.

To update b_u, we use

$$b_u \leftarrow b_u - \alpha \frac{\delta J}{\delta b_u}, \tag{4.10}$$

where α is the rate of convergence (or the learning rate), typically chosen to be small. Taking the partial derivative of Equation 4.9 with respect to b_u and setting this to zero, we obtain

$$\begin{aligned} \frac{\delta J}{\delta b_u} &= -(r_{ui} - b_u - b_i - \mathbf{p}_u^T \mathbf{q}_i) + \lambda b_u \\ &= -e_{ui} + \lambda b_u. \end{aligned} \tag{4.11}$$

Thus, Equation 4.10 may now be rewritten as

$$b_u \leftarrow b_u + \alpha(e_{ui} - \lambda b_u). \tag{4.12}$$

Following a similar procedure, all the parameters values minimising the objective function in Equation 4.9 may be obtained as follows:

$$b_i \leftarrow b_i + \alpha(e_{ui} - \lambda b_i) \tag{4.13}$$

$$\mathbf{q}_i \leftarrow \mathbf{q}_i + \alpha(e_{ui}\mathbf{p}_u - \lambda \mathbf{q}_i) \tag{4.14}$$

$$\mathbf{p}_u \leftarrow \mathbf{p}_u + \alpha(e_{ui}\mathbf{q}_i - \lambda \mathbf{p}_u). \tag{4.15}$$

Before closing this section on matrix factorisation, we would like to mention a special instance of MF known as Non-Negative Matrix Factorisation (NMF) (Lee and Seung, 1999). In NMF, the ratings matrix $\mathbf{R}$ is also decomposed as in Equation 4.3. However, matrices $\mathbf{R}$, $\mathbf{P}$ and $\mathbf{Q}$ are constrained to have non-negative entries. This is useful in situations where the underlying latent vectors can be regarded as non-negative, for example in topic modelling. Xu *et al.* (2003) used an NMF based model in order to cluster documents according to topics. This method was shown to be an improvement over previous approaches used for topic modelling. The interested reader should consult Lee and Seung (1999), where they describe how NMFs can be used for the representation of facial images and for semantic analysis of text documents.

4.3.2 Tensor Factorisation

From the previous section, it is clear that the interaction between two categorical variables (users and items in the case of an RS) can be modelled by means of matrix factorisation. Tensor factorisation allows for the interaction of several categorical variables to be modelled, and therefore they are considered extensions to MF (Rendle, 2012). Two successful tensor factorisation methods are Tucker Decomposition (Malik and Becker, 2018) and Pairwise Interaction Tensor Factorisation (PITF) (Rendle and Lars, 2010) where the latter method is often used for tag recommendation.

The most well-known tensor factorisation model is SVD++ (Koren, 2008). Consequently, we discuss this approach in more detail. This method is used to introduce implicit feedback into the system. As implicit feedback provides additional information with regards to user preferences, it has been shown to offer greater accuracy than the traditional matrix factorisation approach.

Suppose we have access to unary implicit feedback. Then, for each user u there is a set of items $I^*(u)$ for which the user's preference is indicated in an implicit way. To incorporate this information into the model in Equation 4.6, an additional set of item factors is imported. That is, each item $j \in I^*(u)$ is associated with a factor vector $\mathbf{a}_j \in \mathbb{R}^k$. Based on the set of items $I^*(u)$ that a user u expressed interest in, this

user can be described with the new item factors in the vector $\sum_{j \in I^*(u)} \mathbf{a}_j$. The model for predicting the rating r_{ui} is now given by

$$\hat{r}_{ui} = \mu + b_i + b_u + (\mathbf{p}_u + |I^*(u)|^{-\frac{1}{2}} \sum_{j \in I^*(u)} \mathbf{a}_j)^T \mathbf{q_u}. \tag{4.16}$$

From Equation 4.16, it can be seen that by incorporating implicit feedback into the model, a user is now modelled by $\mathbf{p}_u + |I^*(u)|^{-\frac{1}{2}} \sum_{j \in I^*(u)} \mathbf{a}_j$. As was the case with traditional MF, the explicit ratings are used to determine the free vector $\mathbf{p}_u$. Note that the vector $\sum_{i \in I^*(u)} \mathbf{a}_i$ is normalised in order to decrease its variance. As with MF, the parameters can be learned using stochastic gradient descent. Additionally, following the above described procedure, further types of implicit feedback can be incorporated into the model.

Besides SVD++, there are also a number of specialised factorisation models, such as timeSVD++ and Scalable Probabilistic Tensor Factorisation (SPTF), that are able to incorporate non-categorical variables. The timeSVD++ model is a tensor factorisation model which takes the effects of time on ratings into account. If we consider a user's movie genre preferences, for example, his/her taste is likely to change over time. Therefore it is useful to take time into consideration to ensure that recommendations remain accurate.

A domain in which a large amount of implicit feedback is available, is that of e-commerce websites. Here, implicit feedback is available from at least four sources, namely whether or not a user purchased an item, whether or not an item was 'added-to-cart', 'added-to-favourite', or whether or not the item link was clicked on. This type of data is known as heterogeneous behaviour data, and is indicative of user interests. The SPTF model was proposed by Yin *et al.* (2017), which allows all user behaviours to be modelled. Here, the user, item and behaviour type are each represented as a latent vector, and shown to improve prediction accuracy.

4.4 Factorisation Machines

Proposed by Rendle (2010), factorisation machines belong to the class of supervised learning algorithms. They are not restricted to a specific domain, and can be applied in both regression and classification setups. Therefore, FMs are more general models than the typical factorisation models used in collaborative filtering. Furthermore, in contrast to other prediction algorithms, such as Support Vector Machines (SVMs), they are able to operate under sparsity. There are a number of libraries available that allow for the implementation of FMs, the earliest being LibFM, which was written by Rendle (2010). Other FM libraries include xLearn (Ma, 2017) and tffm (Trofimov and Novikov, 2016), where the latter allows for the implementation of arbitrary order FMs in Tensorflow. Also note that FMs have been implemented in the machine learning library environment known as Amazon Sagemaker[1] (Loni, 2018).

[1] https://docs.aws.amazon.com/sagemaker/latest/dg/fact-machines.html

Factorisation models used in collaborative filtering have proven to produce accurate predictions. A prime example is its successful application in the Netflix Prize competition (Bennett *et al.*, 2007; and Koren *et al.*, 2009). However, the drawback of factorisation models is the requirement of the input variables to be categorical variables (Rendle, 2012). In situations where the data cannot be described by categorical variables, the application of factorisation models will not yield a unique solution. Hence, for data with non-categorical input variables, new models and learning algorithms need to be developed. This is not only time consuming, but also requires sufficient knowledge of factorisation models and model development to ensure that the new models do not produce poor results (Rendle, 2012). FMs are more advantageous than factorisation models in the sense that they are able to take real-valued features as input. This is comparable to general predictive machine learning algorithms, such as linear regression and SVMs. For example, both FMs and SVMs are good at modelling interaction between pairs of input variables.

Despite the fact that there exists a connection between FMs and SVMs, importantly note that SVMs are known to fail in the face of sparse data. Therefore, SVMs cannot be used for collaborative filtering. In contrast, since FMs factorise parameters, they are suitable to be used when data are sparse.

From the above description, we see that factorisation machines are able to take real-valued input features (as with SVMs) and produce accurate predictions in sparse settings by factorising interactions between variables (as with factorisation models). Additionally, Rendle (2010) showed that by using feature engineering, FMs are able to imitate a number of factorisation models (including those discussed in Section 4.3.1 and Section 4.3.2), thereby alleviating the need for a proficiency of factorisation models.

Based on Rendle (2010) and Rendle (2012), in Sections 4.4.1 to 4.4.2 we describe the factorisation machine model, as well as the way in which the FM model parameters may be learned. We also compare factorisation machines to methods for matrix factorisation.

4.4.1 The Factorisation Machine Model

An $n \times m$ matrix is generally used to denote the input data for factorisation models, where n indicates the number of users, and where m indicates the number of items. For factorisation machines, however, each observation is represented as a real-valued feature vector $\mathbf{x}_i \in \mathbb{R}^p$, where $i = 1, 2, \ldots, n$. A feature vector specifies the user, and the item that the user interacted with, and every $\mathbf{x}_i$ has a corresponding target value y_i. For example, Figure 4.1 shows the input data to a factorisation machine when creating a movie recommender. Here we see that the blue block specifies the active user, whereas the red block specifies the active item. Note the one-hot encoded form of the user and item IDs. Moreover, for each observation, there is exactly one active user and one active item.

The advantage of representing the input data in terms of feature vectors is that if any additional information about the items or users becomes available, it may easily be incorporated. For example, from

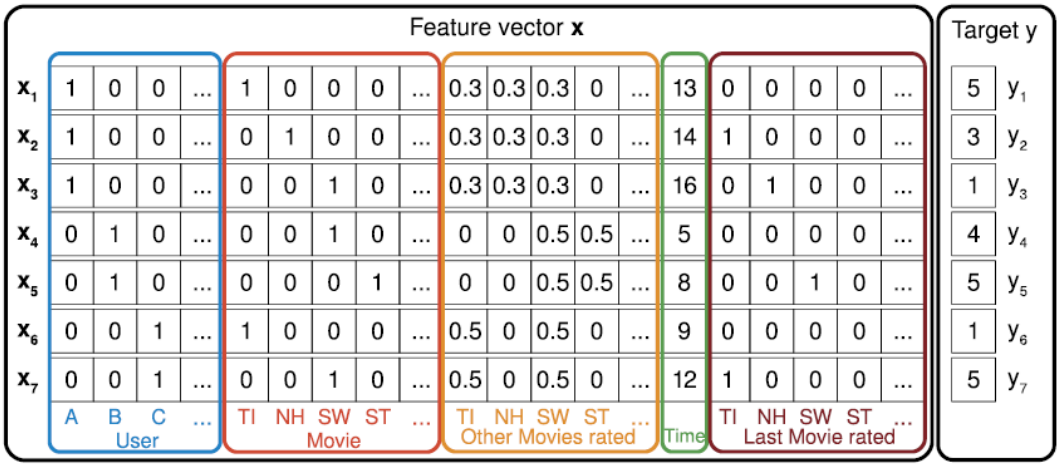


Figure 4.1: Example of movie input data for a factorisation machine.

Source: Rendle (2010)

Figure 4.1, the other movies that a user interacted with (orange block), the time (measured in months) since the rating was given (green block), as well as the last movie that the user rated (brown block), can easily be incorporated into the model. This property, facilitated by the form of the data matrix, of course renders factorisation machines to be highly flexible models.

Generalising the above, let the input data for a factorisation machine be described by the $n \times p$ matrix $\mathbf{X}$, where the i-th row $\mathbf{x}_i$ is the p-dimensional real-valued input vector associated with user i. Each row is associated with a corresponding target value y_i. Then the degree $d = 2$ factorisation machine model is given by

$$\hat{y}_{FM}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} w_{ij} x_i x_j, \tag{4.17}$$

where

$$\begin{aligned} w_{ij} &= \langle \mathbf{v}_i, \mathbf{v}_j \rangle \\ &= \sum_{f=1}^{k} v_{if} v_{jf}. \end{aligned} \tag{4.18}$$

Extending the notation in Equation 4.17, define $\mathbf{V} \in \mathbb{R}^{p \times k}$ as the matrix with rows equal to the latent vectors associated with each variable. That is, the rows of $\mathbf{V}$ are the k-dimensional vectors $\mathbf{v}_j$, $j = 1, 2, \ldots, p$. The hyper-parameter $k \in \mathbb{N}_0^+$ indicates the dimension of the factorisation. This leaves the bias $w_0 \in \mathbb{R}$, the weight vector $\mathbf{w} \in \mathbb{R}^p$, and the latent factor matrix $\mathbf{V} \in \mathbb{R}^{p \times k}$ to be estimated.

Clearly in the degree $d = 2$ model, $\mathbf{w} \in \mathbb{R}^p$ models all main effects, while pairwise interaction are modelled by the w_{ij} parameters. Importantly, the interaction between input variables X_i and X_j is captured by the inner product between two low-dimensional latent vectors $\mathbf{v}_i$ and $\mathbf{v}_j$. Consequently, interaction parameters are not independent, and data for one interaction is used to estimate the parameters for

other interactions. This allows the FM to obtain good parameter estimates even in sparse situations (Rendle *et al.*, 2011).

Rendle (2012) shows that given a large enough k, any pairwise interaction can be expressed by a factorisation machine. This is because for a symmetric, positive semi-definite matrix $\mathbf{W}$, there exists a matrix $\mathbf{V}$ such that $\mathbf{W}$ can be decomposed as $\mathbf{W} = \mathbf{V}\mathbf{V}^T$. In a factorisation machine, the pairwise interactions between variables are represented in the matrix $\mathbf{W}$. Thus, $\mathbf{W}$ is clearly symmetric. $\mathbf{W}$ is also positive definite since $i \neq j$ in the third term of Equation 4.17. Therefore the diagonal elements of $\mathbf{W}$ are not needed in the model, and may assume any values. Of course the expressiveness of the FM is controlled by k. In sparse settings Rendle (2010) notes that a small k should be used to ensure the the modelled can generalise well.

In a factorisation machine, all pairwise interactions need to be computed, therefore the complexity of the estimation process is $O(kp^2)$. Fortunately, via the reformulation of Equation 4.17, the estimation time can be shown to be linear (Rendle, 2012).

$$\begin{aligned}
&\sum_{i=1}^{p-1}\sum_{j=i+1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j \\
&= \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{p}\langle \mathbf{v}_i, \mathbf{v}_j\rangle x_i x_j - \frac{1}{2}\sum_{i=1}^{p}\langle \mathbf{v}_i, \mathbf{v}_i\rangle x_i x_i \\
&= \frac{1}{2}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\sum_{f=1}^{k} v_{if}v_{jf}x_i x_j - \sum_{i=1}^{p}\sum_{f=1}^{k} v_{if}v_{if}x_i x_i\right) \\
&= \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{i=1}^{p} v_{if}x_i\right)\left(\sum_{j=1}^{p} v_{jf}x_j\right) - \sum_{i=1}^{p} v_{if}^2 x_i^2\right] \\
&= \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{i=1}^{p} v_{if}x_i\right)^2 - \sum_{i=1}^{p} v_{if}^2 x_i^2\right]. \qquad (4.19)
\end{aligned}$$

Since Equation 4.19 is linear in both k and p, we see that the model complexity reduces to $O(kp)$. In sparse situations, there is of course a further decrease in model complexity: many of the elements in a row vector $\mathbf{x}$ will be zero, and the sums in Equation 4.19 need only to be computed over a much smaller set of non-zero elements. Therefore, if we let $\bar{m}$ be the average number of non-zero elements of all the vectors $\mathbf{x}$ in $\mathbf{X}$, then the model complexity of the resulting FM, given in Equation 4.20, is $O(k\bar{m})$. That is:

$$\hat{y}_{FM}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2}\sum_{f=1}^{k}\left[\left(\sum_{i=1}^{p} v_{if}x_i\right)^2 - \sum_{i=1}^{p} v_{if}^2 x_i^2\right]. \qquad (4.20)$$

Higher-order variable interactions can be factorised in order to extend the factorisation model in Equation 4.17 to a higher-order model. We discuss this extension of a degree $d = 2$ FM to a higher-order FMs in Section 4.5.1.

4.4.2 Learning Factorisation Machines

In Section 4.4.1 we have seen that the FM model may be estimated in linear time. Therefore, the parameters w_0, $\mathbf{w}$ and $\mathbf{V}$ in Equation 4.17 can efficiently be estimated by means of stochastic gradient descent. It is, however, also possible to use other learning methods, such as alternating least squares (Rendle *et al.*, 2011) or Markov Chain Monte Carlo inference (Freudenthaler *et al.*, 2011). The original implementation of FMs by Rendle (2010) in the library LibFM (Python) contains these three methods. Common loss functions that are minimised when determining the optimal model parameters include squared-, logit- or hinge loss functions.

We proceed with a brief discussion of the use of stochastic gradient descent to optimise the FM parameter values. We start by restating Equation 4.20:

$$\hat{y}_{FM}(\mathbf{x}) \quad = \quad w_0 + \sum_{i=1}^{p} w_i x_i + \frac{1}{2} \sum_{f=1}^{k} \left[\left(\sum_{i=1}^{p} v_{if} x_i \right)^2 - \sum_{i=1}^{p} v_{if}^2 x_i^2 \right].$$

It is easy to see that the gradient of the factorisation machine is given by

$$\frac{\delta}{\delta\theta} \hat{y}_{FM}(\mathbf{x}) = \begin{cases} 1 & \text{if } \theta = w_0 \\ x_i & \text{if } \theta = w_i \\ x_i \sum_{j=1}^{p} v_{j,f} x_j - v_{if} x_i^2 & \text{if } \theta = v_{if}, \end{cases} \tag{4.21}$$

where each gradient can generally be computed in constant time. Let

$$\frac{1}{n} \sum_{i=1}^{n} l(\hat{y}_{FM}(\mathbf{x_i}), y_i) + 2(\beta_1 ||\mathbf{w}||^2 + \beta_2 ||\mathbf{V}||^2) \tag{4.22}$$

be the objective function to be minimised, where $\beta_1 > 0$ and $\beta_2 > 0$ are regularisation parameters learned via cross-validation. Using stochastic gradient descent, the model parameters are incrementally updated until the objective function is minimised. If we denote the learning rate by α, then, at each step of the SGD estimation process, the updates of the model parameters can be computed as follows:

$$\hat{\theta} \quad \leftarrow \quad \hat{\theta} - \alpha \left(\frac{\delta}{\delta\hat{\theta}} l(\hat{y}(\mathbf{x}), y) + 2(\beta_1 ||\mathbf{w}||^2 + \beta_2 ||\mathbf{V}||^2) \right). \tag{4.23}$$

A more detailed description of using SGD or ALS to learn the model parameters of a factorisation machine can be found in Rendle (2012).

4.4.3 Matrix Factorisation versus Factorisation Machines

It is easy to show how factorisation machines are related to matrix factorisation. Indeed, with specific inputs, the model equations of an FM and of MF are exactly the same.

The standard input for the basic matrix factorisation model are the ratings given by a set of users $\mathcal{U}$ to the set of available items $\mathcal{I}$. In the previous section we have seen that instead of letting the users

and items form the rows and columns of the ratings matrix respectively, they can be encoded as binary variables, where each variable represents one level of the users and items. There will now be a total of $|\mathcal{U} \cup \mathcal{I}|$ features, depicted by the blue and red blocks in Figure 4.1. Each observation will have two non-zero entries that correspond to the user and the item in question. Thus, the factorisation model in Equation 4.17 simplifies to

$$\hat{y}_{FM}(\mathbf{x}) = w_0 + w_u + w_i + \langle \mathbf{v}_u, \mathbf{v}_i \rangle. \tag{4.24}$$

Recall from Section 4.3.1 that the matrix factorisation model for predicting the rating of user u for item i was given by

$$\hat{r}_{ui} \quad = \quad \mu + b_u + b_i + \mathbf{p}_u^T \mathbf{q}_i.$$

Relating Equation 4.6 to Equation 4.25, clearly the parameters w_0 and μ are used to the model global bias, w_u and b_u are used to the model user bias, w_i and b_i are used to model the item bias, and $\langle \mathbf{v}_u, \mathbf{v}_i \rangle$ and $\mathbf{p}_u^T \mathbf{q}_i$ are used to model the interaction between users and items in terms of latent vectors.

4.5 Advances in Factorisation Machines

Several enhancements of FMs have been proposed in the literature. The main focus of research in this line is to improve the underlying FM learning model (Loni, 2018). The purpose of this section is to provide an overview of work that has been done in order to propose advanced FM models. We start with an outline of the main directions of research in this context, and proceed with a discussion of key methods from each direction.

In terms of the way in which interaction terms are modelled, Nguyen *et al.* (2014) propose using Gaussian kernels, whereas Blondel *et al.* (2016*a*) introduce the first efficient algorithm for training Higher-Order Factorisation Machines (HOFMs). A third contribution in this regard is the interesting proposal by Tay *et al.* (2019), called holographic factorisation machines, where the inner products in FMs are replaced with so-called 'holographic reduced representations', *viz.* circular convolution and correlation. These operators were first proposed by Plate (1995), as encoding and decoding operations used to 'emulate storage and retrieval in holography'.

In terms of grouping together features in the input matrix, Juan *et al.* (2016) propose learning interactions depending on the group (or field) to which a feature belongs to. Their proposal is called a Field-Aware Factorisation Machine (FFM). A more efficient algorithm, called Field-Weighted Factorisation Machines (FwFMs), is proposed in the paper by Pan *et al.* (2018). Along similar lines, Loni (2018) proposes the use of weighted factorisation machines, where each group of features is assigned a weight parameter that needs to be learned.

Other contributions include extending the use of FMs to RS ranking (Loni *et al.*, 2019), speeding up recommendation with FMs (Loni *et al.*, 2015), ways of enhancing FMs by making fuller use of the available data (Loni *et al.*, 2014), the use of neural networks to filter out relevant FM interactions in so-called Attentional Factorisation Machines (AFMs) (Xiao *et al.*, 2017), and ensemble learning with FMs (Yuan *et al.*, 2017).

With regard to an alternative approach to handling interaction terms, we focus in this section on HOFMs, whereas in terms of modelling interactions between groups of features, we discuss FFMs and FwFMs. We also single out AFMs as a recent state-of-the-art procedure.

4.5.1 Higher Order Factorisation Machines

In Section 4.4, we discussed how a second-order FM models the matrix containing the pairwise interactions between variables by a low-rank matrix. The consequence of this lower-dimensional representation is two-fold. Rendle *et al.* (2011) showed that FMs are not only able to produce results that are equal to those produced by SVMs, but are able to do so with much more efficiency. This is particularly true in highly sparse situations where SVMs fail. The other advantage is that in sparse settings, weights can be determined even for feature combinations not occurring in the training data since the weights are not independent.

Rendle *et al.* (2011) extend the second-order factorisation model given in Equation 4.17 to an arbitrary-order factorisation model that allows for higher-order feature combinations. The d-order FM model is given by

$$\hat{y}_{HOFM}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{l=2}^{d} \sum_{i_1=1}^{p} \dots \sum_{i_l=i_{l-1}+1}^{p} \left(\prod_{j=1}^{l} x_{i_j} \right) \left(\sum_{f=1}^{k_l} \prod_{j=1}^{l} v_{i_j f}^{(l)} \right), \tag{4.25}$$

where the model parameters to be estimated are $w_0 \in \mathbb{R}$, $\mathbf{w} \in \mathbb{R}^p$ and $\mathbf{V}^{(l)} \in \mathbb{R}^{p \times k_l}$, and where $l \in \{2, 3, \dots, d\}$ is the degree of feature combinations considered. Since a unique matrix containing the latent vectors associated with each variable is used for each degree, a large number of parameters need to be estimated.

The objective function of an HOFM, defined similarly to the objective function of a second-order FM, is given by

$$\frac{1}{n} \sum_{i=1}^{n} l(\hat{y}_{HOFM}(\mathbf{x}_i), y_i) + 2(\beta_1 ||\mathbf{w}||^2 + \sum_{l=2}^{d} \beta_l ||\mathbf{V}^{(l)}||^2), \tag{4.26}$$

where $\beta_1, \beta_2, \dots, \beta_d > 0$ are hyper-parameters. Blondel *et al.* (2016*a*) suggests setting $\beta_1 = \beta_2 = \dots = \beta_d$ and $k_2 = k_3 = \dots = k_d$ in order to reduce the computation time needed to search all possible hyper-parameter combinations.

In the original paper by Rendle *et al.* (2011), no training algorithm was given for HOFMs. Due to this, Blondel *et al.* (2016*a*) proposed two linear-time learning algorithm for HOFMs, one using stocatishic gradient descent, the other using coordinate descent. To use these algorithms, Equation 4.25 needs to be expressed in terms of the so-called ANOVA kernel (Blondel *et al.*, 2016*b*). The multi-linearity of the ANOVA kernel is exploited in the derivation of the proposed learning algorithms. The interested reader should consult Blondel *et al.* (2016*a*) for the full derivation of the linear-time learning algorithms.

The ANOVA kernel of degree $2 \leq d \leq p$ is given by

$$\mathcal{A}^d(\mathbf{v}, \mathbf{x}) = \sum_{i_d > \ldots > i_1} \prod_{l=1}^{d} v_{i_l} x_{i_l}. \tag{4.27}$$

In order to see how the HOFM model can be expressed in terms of Equation 4.27, we first reformulate Equation 4.25. Let $\bar{\mathbf{v}}_i^{(l)}$ be the ith row of $\mathbf{V}^{(l)}$ and let $\langle \bar{\mathbf{v}}_{i_1}^{(l)}, \ldots, \bar{\mathbf{v}}_{i_l}^{(l)} \rangle$ be the sum of element-wise products. Then the HOFM model can expressed as

$$\hat{y}_{HOFM}(\mathbf{x}) = w_0 + \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{i' > i} \langle \bar{\mathbf{v}}_i^{(2)}, \bar{\mathbf{v}}_{i'}^{(2)} \rangle x_i x_{i'} + \ldots + \sum_{i_d > \ldots > i_1} \langle \bar{\mathbf{v}}_{i_1}^{(d)}, \ldots, \bar{\mathbf{v}}_{i_l}^{(d)} \rangle x_{i_1} x_{i_2} \ldots x_{i_d}. \tag{4.28}$$

Following this, the HOFM can be written in terms of the ANOVA kernel as follows

$$\hat{y}_{HOFM}(\mathbf{x}) = w_0 + \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{f=1}^{k_2} \mathcal{A}^2(\mathbf{v}_f{}^{(2)}, \mathbf{x}) + \ldots + \sum_{f=1}^{k_d} \mathcal{A}^d(\mathbf{v}_f{}^{(d)}, \mathbf{x}). \tag{4.29}$$

As mentioned previously, different matrices $\mathbf{V}^2, \mathbf{V}^3, \ldots, \mathbf{V}^d$ are used to model each degree, resulting in a complex model that is time-consuming to evaluate. In their paper, Blondel *et al.* (2016*a*) also propose two new kernels that allow parameters to be shared between degrees, thereby reducing computation cost.

4.5.2 Field-Aware Factorisation Machines

A recent variant of factorisation machines are field-aware factorisation machines, proposed by Juan *et al.* (2016). In their paper, FFMs where shown to be successful in the prediction of Click Through Rate (CTR), which is essential for determining whether or not an online advertisement will be successful. Zhang *et al.* (2018) reformulate FFMs in terms of recommender systems, and we follow a similar procedure in our discussion of FFMs.

Pan *et al.* (2018) note three challenges that need to be addressed when designing a model for predicting CTR, which can be carried over to the prediction of ratings by recommender systems. Firstly, the interactions between features need to be accounted for by the model. Secondly, it is possible that the feature interactions across fields differ. And thirdly, because there may be a large number of features, model complexity needs to be minimised. From our discussion in Section 4.4, the first challenge is easily addressed by factorisation machines. In this section, we review FFMs in the context of recommender systems, and show how they are able to address the second problem.

For our discussion, consider the artificial movie dataset depicted in Table 4.1, adapted from Figure 4.1. There are a total of eight categorical features in this dataset, namely Titanic (T), Notting Hill (NH), Star Wars (SW), Star Trek (ST), Home Alone (AL), Romance (R), Sci-Fi (SF) and Comedy (C). These features can be grouped according to which field they belong to, namely User, Item or Genre. Therefore, this type of data is known as multi-field categorical data. Since each feature belongs to only one field, multi-filed categorical data are typically very sparse.

Table 4.1: An artificial movie dataset.

User	Item	Genre	Rating
A	Titanic	Romance	5
A	Notting Hill	Romance	3
A	Star Wars	Sci-Fi	1
B	Star Wars	Sci-Fi	4
B	Home Alone	Comedy	5
C	Titanic	Romance	1
C	Star Wars	Sci-Fi	5
D	Star Trek	Sci-Fi	4
D	Home Alone	Comedy	4

Referring to the dataset described in Table 4.1, suppose that we obtain a new observation, depicted in Table 4.2. The vector representation of this observation is given in Equation 4.30.

Table 4.2: Observation from an artificial movie dataset.

User	Movie	Genre
D	Star Wars	Sci-Fi

$$\begin{aligned}\mathbf{x} &= (x_A, x_B, x_C, x_D, x_T, x_{NH}, x_{SW}, x_{HA}, x_{ST}, x_R, x_{SF}, x_C) \\ &= (0,0,0,1,0,0,1,0,0,0,1,0). \end{aligned} \tag{4.30}$$

The difference between FMs and FFMs is the number of latent vectors used to describe the interaction between features. In FMs, a single latent vector is used to represent a feature and learn the interaction effects between other features. Referring to Table 4.2, suppose we wanted to predict the rating that user D would give to the movie Star Wars using a $d = 2$ factorisation machine. The effect of the interactions between features would be described by the FM model as follows:

$$\begin{aligned}\phi_{FM} &= \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} w_{ij} x_i x_j \\ &= \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\ &= \langle \mathbf{v}_D, \mathbf{v}_{SW} \rangle + \langle \mathbf{v}_D, \mathbf{v}_{SF} \rangle + \langle \mathbf{v}_{SW}, \mathbf{v}_{SF} \rangle. \end{aligned} \tag{4.31}$$

If features belong to different fields, the latent effects of the features may in fact be different. Therefore, the interaction between features may not be accurately captured by the factorisation model in Equation 4.31. Here, the same latent vector $\mathbf{v}_D$ is used to learn the effects of the interaction between the features D and Star Wars, and between D and Sci-Fi, even though these effects may be different.

FFMs aim to rectify this problem by using the fact that features can be grouped into fields. Instead of learning a single latent vector for a feature, several latent vectors with k factors are learned for each feature. The latent vectors will correspond to the field into which the feature falls. This is shown in Equation 4.32, where f_i indicates the field to which feature i belongs. Thus, when the latent effects with other features are learned, different latent vectors are used.

$$\begin{aligned} \phi_{FFM} &= \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \langle \mathbf{v}_{i,f_j}, \mathbf{v}_{j,f_i} \rangle x_i x_j \\ &= \langle \mathbf{v}_{D,Item}, \mathbf{v}_{SW,User} \rangle + \langle \mathbf{v}_{D,Genre}, \mathbf{v}_{SF,User} \rangle + \langle \mathbf{v}_{SW,Genre}, \mathbf{v}_{SF,Movie} \rangle. \end{aligned} \tag{4.32}$$

From the above example, the degree $d = 2$ field-aware factorisation machine model is as follows:

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \langle \mathbf{v}_{i,f_j}, \mathbf{v}_{j,f_i} \rangle x_i x_j, \tag{4.33}$$

where w_0 is the global bias and w_i indicates the strength of the i-th feature. Also, the interaction between the i-th and the j-th variable is represented by $\langle \mathbf{v}_{i,f_j} \mathbf{v}_{j,f_i} \rangle$. The interaction between the feature x_i and the feature x_j (belonging to field f_j) is given by the latent vector $\mathbf{v}_{i,f_j}$. As with FMs, stochastic gradient decent can be used to learn the model parameters of FFMs.

Suppose that there are total of p features and F different fields, and the latent vectors have k factors. Then the number of parameters to be learned by the FFM is $p + kp(F-1)$, which is more than the $p + kp$ parameters of the FM model. If $\bar{m}$ is the average number of non-zero elements of all the observations $\mathbf{x}$, then it can be shown that the complexity of an FFM is $O(k\bar{m})$.

4.5.3 Field-Weighted Factorisation Machines

In Section 4.5.2, we saw how multiple latent vectors are learned for each feature in order to model the interaction effects of features belonging to different fields. Therefore, FFMs resolve the second challenge noted by Pan *et al.* (2018). The third challenge is rectified by FwFMs, by determining the degree to which different field pairs are associated with one another.

In FMs, there is no discrimination between the importance of field interactions. This means that interactions that are in fact not very useful to the prediction of ratings are treated the same as the interactions that are important. This lack of differentiation between the importance of field interactions in FMs may be detrimental to the resulting prediction. Therefore, Pan *et al.* (2018) proposed the introduction of weights into the FM model in order to capture the importance of field interactions. Bare in mind that

FwFMs are applicable only to data containing categorical variables, and therefore have been used for improved CTR prediction, rather than for general recommendation problems.

The FwFM model extends the FM model with the addition of a weight, and is given by

$$\hat{y}(\mathbf{x}) = w_0 + \sum_{i=1}^{p} w_i x_i + \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j r_{f_i, f_j}, \tag{4.34}$$

where w_0 is the global bias and w_i indicates the strength of the i-th feature. The field of feature i is denoted by f_i. The various strengths of the different field pairs interactions is expressed by the weight r_{f_i,f_j}. Pan *et al.* (2018) suggest using mutual information to validate the importance of different field pair interactions, thereby determining the weights r_{f_i,f_j}.

Table 4.3 shows the number of parameters in the three factorisation machine models discussed in this chapter. The number of features and number of fields are denoted by p and F respectively, while the number of latent factors used is k. Generally we have that $F \ll p$, which means that the number of parameters in an FwFM is smaller than the number of parameters in an FFM. Therefore, FwFMs have a model complexity that is smaller than that of FMs.

Table 4.3: Number of parameters in various factorisation machines models.

Model	**Number of Parameters**
FM	$p + kp$
FFM	$p + kp(F-1)$
FwFM	$p + pk + \frac{F(F-1)}{2}$

4.5.4 Attentional Factorisation Machines

Attentional factorisation machines are similar models to FwFMs, allowing for the interaction between features to have different levels of importance in the prediction of a rating. However, there are two key differences between the models. Firstly, AFMs are not restricted to binary inputs, and can therefore be applied to a wider range of problems. Secondly, since a neural network model is used for AFMs, the interaction weights are learned in an automatic way, unlike FwFMs where they are determined using mutual information (Pan *et al.*, 2018). We briefly discuss the architecture of the neural network for AFMs here, though a detailed description can be found in the paper by Xiao *et al.* (2017).

The input to the neural network is a sparse p-dimensional vector $\mathbf{x}$. This vector is passed to the embedding layer, where each non-zero element in $\mathbf{x}$ is projected onto an embedding vector $\mathbf{v}$. Therefore, letting $\mathcal{X}$ denote the set of non-zero features in $\mathbf{x}$, the output of the embedding layer is $\mathcal{E} = \{\mathbf{v}_i x_i\}_{i \in \mathcal{X}}$.

Each of the embedding vectors are then passed to a pairwise interaction layer, denoted by PI, where the element-wise product $(\mathbf{v}_i \odot \mathbf{v}_j)$ between all non-zero embedded feature vectors are computed. The

output of this layer is

$$f_{PI}(\mathcal{E}) \quad = \quad \{(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j\}_{(i,j)\in\mathcal{R}_x}, \tag{4.35}$$

where $\mathcal{R}_x = \{(i,j)_{i\in\mathcal{X}, j\in\mathcal{X}, j>i}\}$. The computed interaction features are then passed to the attention-based pooling layer, denoted by Att, in which the attention score a_{ij} for each interaction feature $(\mathbf{v}_i \odot \mathbf{v}_j)\rangle$ is computed. This attention score signifies the importance of the interaction feature. The output of this layer is given by

$$f_{Att}(f_{PI}(\mathcal{E})) \quad = \quad \sum_{(i,j)\in\mathcal{R}_x} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j. \tag{4.36}$$

Finally, the prediction layer consists of a fully connect layer which projects the output of the attention-based pooling layer to the predicted score. The AFM model is given by

$$\hat{y}_{AFM}(\mathbf{x}) \quad = \quad w_0 + \sum_{i=1}^{p} w_i x_i + \mathbf{p}^T \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} a_{ij}(\mathbf{v}_i \odot \mathbf{v}_j)x_i x_j, \tag{4.37}$$

where w_0 is the global bias, w_i indicates the strength of the i-th feature, and $\mathbf{p} \in \mathbb{R}^k$ represents the weights computed in the prediction layer. The model parameters of an AFM can be learned by minimising an objective function using stochastic gradient descent.

4.6 Summary

In this chapter we explored various techniques of representing sparse data in a low-dimensional space. The original approach used to do this was singular value decomposition. However, due to the fact that missing values first needed to be imputed, a variant of SVD, known as matrix factorisation, was developed. This allowed for the correlation between users and items to be represented by latent factors. From this, tensor factorisation models where proposed that permit the inclusion of other sources of information into the model.

Following these techniques, the factorisation machine model was described. We saw that the FM model improved on the basic MF models by incorporating feature interactions. Further modifications of the FM gave way to state-of-the-art models that allow for higher-order interactions, for allocating different weights to features belonging to different fields, or for the incorporation of feature importance into the model.

Chapter 5

Recommendations using Multi-Label Classification

5.1 Introduction

Our topic of the preceding chapters was traditional recommender systems, where the aim was to recommend the most relevant item(s) to a specific user. In this chapter, in addition to traditional recommender systems, we also turn our attention to the problem of context recommendation, which may be regarded as a smaller theme in research on recommender systems. In context recommendation, the aim is to recommend the most relevant context in which particular items are expected to be appreciated by the active user. Context recommendation should not be confused with context-aware recommendation, which constitutes a much broader area of research than context recommendation.

In order to appreciate the differences between context-aware recommendation and context recommendation, we start with a brief overview of the former. Recall that traditional recommender systems only consider user and item data, ignoring any detail providing context to a user or event (such as location or time). Intuitively, the exploitation of such contextual user information may improve the performance of recommender systems. For example, during weekend evenings, users might prefer watching movies, while during week nights, users might favour watching the news. It is clear that users' preferences depend on their contexts, and in this example, if an RS is aware of the type of day on which a user watches TV, more appropriate items may be suggested. Therefore, briefly, a Context-Aware Recommender System (CARS) aims to address a short-coming of traditional recommender systems by taking users' contexts into account when making a recommendation.

Context-aware recommender systems may be applied in a broad area of domains. Rosenberger and Gerhard (2018) explore a use case of CARS in an industrial application. Here they consider the daily tasks of workers in a company that produces automotive parts. Workers can be divided into different groups,

depending on the information that they require relating to the products, machinery and other workers. A CARS system is proposed that will assist the workers by giving them the appropriate information, depending on the workers' context. In a multi-dimensional approach by Adomavicius *et al.* (2005), it is shown that the use of contextual information, coupled with the typical user and item information, improves recommendations in e-commerce personalisation. The incorporation of contextual information in recommender systems may also be advantageous in the case of location-based services on mobile phones. For example, tourists may share their travels with remote users through photographs, location and voice feedback using the system George Square (established by Brown *et al.* 2005).

From the above findings it is clear that the inclusion of contextual information into the recommendation process allows for recommended items to be more applicable. In contrast, and in order to introduce the context recommendation problem, often recommendation of the contexts in which an item is expected to be appreciated by a particular user, is of primary interest. For example, when a user is shopping online for a jacket, he or she might not know if it is more appropriate for everyday wear or rather to be worn in the snow. Thus, recommending contexts in which the jacket can be worn, may aid the user's final decision. Or, a user might be unsure with regard to the best situation in which to watch a particular movie. A context recommender could then suggest settings most suited to the movie. These could include the best location (home or theatre), companion (child or partner), or time (weekday or weekend), among other possible contexts.

In contrast to context-aware recommendation, little attention has been paid to context recommendation in the literature. In order to predict the best context for a user to listen to a music track, a kNN-based model was suggested by Baltrunas *et al.* (2010). Benmansour *et al.* (2015) use context prediction in pervasive computing for smart home systems, where the difference between context prediction and context recommendation is that in the former, the aim is to predict the upcoming context.

The objective of our work in this chapter, and of the accompanying empirical study in Chapter 7, is to explore the potential of MLC procedures to address the context recommendation problem. This topic has been investigated for the recommendation of movies and travel destinations in the paper by Zheng *et al.* (2014). For the broader application of recommendation, but not specifically for context recommendation, the use of MLC was also studied in the papers by Rivolli *et al.* (2017 and 2018). Our unique contribution is extending the exploration of the use of MLC in recommender systems to include the recently proposed use of regression approaches for MLC (Bierman, 2019).

The remainder of the chapter is structured as follows. We provide further information regarding the context recommendation problem in Section 5.2, which is largely based on the paper by Zheng *et al.* (2014). Since our focus is on context recommendation by means of multi-label classification, the MLC framework and established MLC models are discussed in Sections 5.3 and 5.4, respectively. The more

recently proposed use of multivariate linear regression in an MLC context is discussed in Section 5.5. We conclude the chapter with a discussion of frequently used performance measures in a multi-label classification framework in Section 5.6, and with a summary in Section 5.7.

5.2 The Context Recommendation Problem

Time, location, companion(s) and weather are some attributes that may be thought of when referring to context. With regard to context recommender systems, these variables are known as contextual dimensions. The values that contextual dimensions may assume are called contextual conditions. For example, the contextual dimension 'companion(s)' may have the contextual conditions 'alone', 'family', 'friends', 'partner', or 'other'.

In a traditional RS, users U and items I are taken as input and the ratings R are predicted. Therefore, we have a two-dimensional rating function h_R given by

$$h_R : U \times I \to R. \tag{5.1}$$

As in traditional systems, we also have a two-dimensional function in a context recommender. However, instead of predicting ratings, the outputs to be estimated are contexts. That is, we have the mapping

$$h_C : U \times I \to C. \tag{5.2}$$

Note that contexts may be predicted in either a direct, or in an indirect manner. For indirect context recommendation, customary algorithms proposed for context-aware recommender systems, may be used to obtain a rating. The procedure involves fixing the user and item pair, changing the context values and predicting the corresponding rating. In this way, all possible contexts for a user and item pair are ranked based on the predicted rating. The context values that achieved the highest predicted rating are deemed to be the most relevant, and are then recommended to the user.

For direct context recommendation, the rating function defined in Equation 5.3 is used. Here each unique combination of contexts are represented as context labels. Therefore this method may be implemented using classification methods.

$$h_C : U \times I \times R \to C. \tag{5.3}$$

In context recommendation, the contexts also need to be filtered in order to determine the relevant contexts. Contexts are often domain-specific, and therefore some contexts may be more valuable in certain scenarios. Determining the appropriate contexts to use is important, since it may influence the final results from the model (Adomavicius and Tuzhilin, 2011). For example, consider a system recommending the best contexts for a restaurant. Contexts such as 'time' or 'companion' would be useful, while a context such as the current price of petrol might be less useful in recommending a restaurant.

Determining relevant contexts can be done either before or after predicting the contexts, respectively know as pre- or post-filtering.

Pre-filtering entails selecting particular contexts prior to prediction, and only considering those when making predictions. Different pre-filtering methodologies exist in order to determine relevant contexts, such as consulting domain experts (Adomavicius *et al.*, 2005). Other methods that may assist in selecting contexts include user studies and statistical analyses. For example, Baltrunas *et al.* (2012) conducted a user study in order to deduce which contextual features influence the decisions users make with regard to particular items. Examples of statistical analyses toward selecting contexts are χ^2-tests (Liu *et al.*, 2010), while Adomavicius *et al.* (2005) use t-tests to determine the most appropriate context variables. Briefly, we consider how a t-test may be carried out.

Consider items with the contextual dimension 'Time', which has two contextual conditions 'Weekend' and 'Weekday'. In order to establish whether or not this dimension affects the ratings, we would need to determine if 'Time = Weekend' and 'Time = Weekday' have the same distribution of ratings. In other words, we have a hypothesis test with the null hypothesis that the distribution of ratings for 'Time = Weekend' is the same as the distribution of ratings for 'Time = Weekday'. To conduct a t-test, the item ratings are partitioned according to these contextual conditions, and the average rating received by each condition is computed. The difference in mean ratings between the two conditions is used to calculate the t-statistic and the associated p-value.

In post-filtering, all available contexts are considered when making predictions. After the predictions have been made for all contexts, the contexts that are deemed irrelevant are removed, or a list of the top-N contexts are selected. To implement this, it is necessary to evaluate the ranking of these predicted contexts. Users' preferences, obtained either implicitly or explicitly, with regard to the contextual dimensions are needed in order to conduct the evaluation.

5.3 Multi-Label Classification

Since our objective in this chapter (and in Chapter 7) is to explore the use of multi-label classification methods as part of recommendation systems, in this section we introduce the multi-label classification setup.

Consider a set of training data $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, 2, \ldots, n\}$, where $\mathbf{x}_i$, $i = 1, 2, \ldots, n$, denotes a p-dimensional vector of observations of the input variables X_1 up to X_p, and where the K-dimensional vector $\mathbf{y}_i$, $i = 1, 2, \ldots, n$ represents the observations of a set of output variables Y_1 up to Y_K. Here the output variables (or labels) may assume a value of 0 or 1. In order to contrast multi-label classification with the better known problems of binary and multi-class classification, recall that in binary classification, all data observations may be categorised into one of two disjoint groups (or classes). This implies that

$Y \in \{0, 1\}$, and $K = 1$. In multi-class classification, we have $K > 1$, and each data observation may belong to one of the K possible classes presented by the binary labels Y_1 up to Y_K. That is, the sum of the observed values for Y_1 up to Y_K has to equal 1.

It is clear that in both binary and multi-class classification, data observations are allowed to only belong to a single class. In contrast, in multi-label classification, observations may simultaneously belong to several categories. That is, we have $K > 1$ and $Y_k \in \{0, 1\}$, $k = 1, 2, \ldots, K$, where the sum of the observed values for Y_1 up to Y_K may be be greater than 1. Here the different response classes are referred to as labels, and each observation may be associated with multiple labels. Note also that the collection of unique label combinations is known as the *labelset*.

A multi-label dataset may be represented in matrix notation as $[\mathbf{X}\,\mathbf{Y}]$, where $\mathbf{X}$ denotes the predictors and where $\mathbf{Y}$ denotes the responses, or labels. A typical multi-label dataset of size $n \times (p + K)$ is depicted in Table 5.1 below.

Table 5.1: Multi-label dataset represented as a matrix.

	X_1	X_2	$\ldots$	X_p	Y_1	Y_2	$\ldots$	Y_K
1	$x_{1,1}$	$x_{1,2}$	$\ldots$	$x_{1,p}$	1	0	$\ldots$	0
2	$x_{2,1}$	$x_{2,2}$	$\ldots$	$x_{2,p}$	0	1	$\ldots$	1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	$x_{n,1}$	$x_{n,2}$	$\ldots$	$x_{n,p}$	0	0	$\ldots$	1

Given an observation $\mathbf{x}$, which was unseen during training, the goal of multi-label classification is to assign the corresponding biniary outcomes of the K labels as accurately as possible. MLC has a wide scope of applications, including tag annotation (Katakis *et al.*, 2008), the classification of music by the emotion it evokes (Trohidis *et al.*, 2011), and image annotation (Nasierding *et al.* 2009; and Chong *et al.* 2009).

5.4 Established MLC Learning Methods

There are three main types of methods available for performing multi-label classification, *viz.* problem transformation-, algorithm adaptation-, and ensemble methods. We provide a brief overview of these three approaches here. The interested reader may also want to refer to a comprehensive exposition and comparative study of them, as provided in the paper by Madjarov *et al.* (2012). Since an important part of our work in this chapter is the exploration of the use of regression approaches to MLC in recommender systems, note that regression for MLC (which may be categorised as an algorithm adaptation approach) will also be discussed.

5.4.1 Problem Transformation Methods

Problem transformation methods work by converting the multi-label classification problem into several single-label problems. This is done by transforming the data in such a way that a single-label classifier can be implemented. Thus, any classification algorithm can be applied to the transformed data in order to predict the binary outcomes of the K labels. Three well-known problem transformation methods are Binary Relevance (BR) (Tsoumakas and Katakis, 2007), Label Powerset (LP) (Modi and Panchal, 2012; Sorower, 2010) and Classifier Chains (CC) (Read *et al.*, 2011).

Binary relevance is the simplest problem transformation method. Each label is considered to represent a separate binary classification problem. Therefore in binary relevance, K binary classifiers are learned - one for each of the labels. A potential disadvantage of binary relevance stems from the fact that the K classifiers are learned separately: thereby informative dependencies among labels cannot be taken into account.

Label powerset methods attempt to rectify the potential disadvantage of binary relevance mentioned above. This is done by viewing each unique set of labels as a single class, whereby any multi-class classifier may then be applied. In other words, the multi-label classification problem is transformed into a multi-class classification problem. A disadvantage of label powerset methods is that the transformation of the MLC problem typically leads to a large number of classes, and only a small number of observations per class. In order to alleviate this problem, a pruned problem transformation method was proposed by Read (2008). In this approach, observations corresponding to labels that appear less than a predetermined threshold are removed from the training data. To prevent the loss of too much information, some of these observations are reintroduced under a new label. A further complication of the LP method is that by combing the label combinations into a single class, no new label combinations are allowed. This means that a new, unseen observation will not be correctly classified if the label combination never appeared in the training set.

Classifier chains are similar to binary relevance in the sense that K binary classifiers are also learned. However, in the CC approach, a way is found to exploit potential information in the form of label dependencies. By finding a way to link the binary classifiers, correlations among labels are incorporated in the learning process. Therefore classifier chains take the form of a sequence of classifiers, where the labels predicted by the previous classifier in the sequence are integrated into the feature space of the current classifier.

5.4.2 Algorithm Adaptation Methods

In a sense, algorithm adaptation methods take an opposite approach to that of problem transformation methods. Instead of transforming the data, algorithm adaptation methods transform the algorithms for

single-label classifiers in order to be applied in the MLC setting.

Examples of algorithms that have been extended to multi-label classification, are decision trees and nearest neighbour averaging. Clare and King (2001) adapted the C4.5 tree algorithm to allow multiple labels in the leaves of the trees. Two common extensions of the nearest neighbour algorithm are Multi-label k-nearest Neighbours (ML-kNN) (Zhang and Zhou, 2007), and Binary Relevance k-nearest Neighbours (BR-kNN) (Spyromitros *et al.*, 2008). In ML-kNN, to classify a new observation to a labelset, a neighbourhood consisting of the new observation's k-nearest neighbours in the training set is first identified. The frequency of each label in this neighbourhood is then determined, as well as the prior and posterior probability corresponding to the frequency of each label. Using the frequencies and the corresponding probabilities for each label, the maximum *a posterior* principle is utilised to determine the labelset of the new observation. In other words, ML-kNN combines the kNN algorithm and Bayesian inference to decide on the labelset of an unseen instance. In BR-kNN, the k-nearest neighbours of a new observation are also first identified. Following this, a classifier is learned for each label in the neighbourhood, as in binary relevance.

Adaptations have also been made to neural networks in order to handle multi-label data. Zhang and Zhou (2006) modify the back propagation algorithm to make it applicable to mutli-label problems. Their proposed algorithm, Back-Propagation Multi-Label Learning (BP-MLL), is able to ascertain that labels associated with an observation should be deemed more important than the labels not associated with the observation. This is done with a unique error function. Another algorithm, named Multi-Label Radial Basis Function (ML-RBF), was introduced by Zhang (2009). The first step in the algorithm is to apply k-means clustering so that observations that correspond to the same label are grouped together. The centroids of these clusters, known as prototype vectors, form the first layer of the neural network. These prototype vectors are used as input to the radial basis functions. Then, the sum-of-squared-errors loss function is minimised in order to learn the weights of the second layer.

5.4.3 Ensemble Methods

Ensemble methods combine the predictions of several classifiers into a single prediction. Typically the predictions from each classifier is averaged, leading to an ensemble classifier with smaller variance than that of the individual classifiers. The individual classifiers may be obtained using either problem transformation or algorithm adaptation algorithms. Well-known ensemble methods for MLC are Random k-labelsets (RAkEL) (Tsoumakas and Vlahavas, 2007) and ensembles of classifier chains (Read *et al.*, 2011).

5.5 Regression Approaches to MLC

A more recent proposal toward MLC is to make use of multivariate linear regression. Whereas it is well-known that ordinary multivariate linear regression of K labels onto p predictors simply fits K separate linear functions, there are extensions to multivariate linear regression which take potential dependence structures among the labels into account. The use of these regression approaches in an MLC context was proposed by Bierman (2019). While in the past, most MLC research focused on advancements in terms of classification performance alone, there has recently been a shift toward obtaining interpretable MLC models. This was also the main focus in the Bierman (2019) paper. In an empirical study on a set of benchmark datasets it was found that the regression approaches are competitive with current state-of-the-art MLC methods. The regression approaches have the additional advantage of yielding interpretable output in terms of estimated regression coefficients. Developed further, such output may facilitate variable selection for MLC, and additional improvements in classification performance. The following discussion is largely based on the work in Bierman (2019).

5.5.1 Extensions to Multivariate Linear Regression

Suppose that we have a set of training data as defined in the beginning of Section 5.3. The multivariate linear regression model is well-known, *viz.*

$$\mathbf{Y} \quad = \quad \mathbf{XB} + \mathcal{E}, \tag{5.4}$$

where $\mathbf{X}$ is the $n \times p$ matrix containing the observations on the input variables, and $\mathbf{Y}$ is the $n \times K$ matrix of observations on the K labels, as defined earlier on in Section 5.3. $\mathcal{E}$ is the $n \times K$ matrix of error terms, and $\mathbf{B}$ is the $p \times K$ matrix of regression coefficients which needs to be estimated. We may use Ordinary Least Squares (OLS) to estimate $\mathbf{B}$, yielding

$$\hat{\mathbf{B}}_{OLS} \quad = \quad (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{5.5}$$

As stated before, the $\hat{\mathbf{B}}_{OLS}$ matrix will contain the estimated regression coefficients when each label is modelled separately. Therefore, in predicting each label, information contained in the other labels cannot be used. To address this problem, extensions to ordinary multivariate linear regression have been proposed. Note that these proposals have more general applications in mind, and were not developed for MLC. The following discussion summarises the fundamental design of these extensions to multivariate regression. One starts by transforming the data to canonical coordinates. The first pair of canonical coordinates is the linear combinations $\sum_{k=1}^{K} a_k Y_k$ and $\sum_{j=1}^{p} b_j X_j$, maximising the correlation between any two such linear combinations. Further coordinates, up to a maximum of $q = \min(p, K)$ are defined similarly, subject to orthogonality restrictions. Coordinates with small canonical correlations are then

shrunk towards zero, whereafter the canonical coordinates are transformed back to the original coordinate system. The above steps may be achieved by means of the following matrix multiplication:

$$\hat{\mathbf{B}} = \hat{\mathbf{B}}_{OLS}\mathbf{A}\mathbf{D}\mathbf{A}^{-1}, \tag{5.6}$$

where the columns of the $q \times q$ matrix $\mathbf{A}$ contain the response canonical coordinates and $\mathbf{D}$ is a diagonal matrix. Different methods to shrink the canonical coordinates imply different $\mathbf{D}$ matrices, and lead to different regression approaches. Izenman (1975) proposed Reduced Rank (RR) regression, where the diagonal entries in $\mathbf{D}$ are specified as follows:

$$d_i = \begin{cases} 1 & \text{if } i \leq m \\ 0 & \text{otherwise,} \end{cases} \tag{5.7}$$

where m is a tuning parameter that needs to be specified. Van Der Merwe and Zidek (1980) proposed Filtered Canonical Y-variate Regression (FICYREG). In FICYREG, the diagonal entries in $\mathbf{D}$ are given by

$$d_i = \frac{c_i^2 - w/n}{c_i^2(1 - w/n)}, \tag{5.8}$$

where $c_1^2 \geq c_2^2 \geq \ldots \geq c_q^2$ denote the squared canonical correlations, and where $w = p - K - 1$. Finally in this section, (Breiman and Friedman, 1997) proposed so-called Curds-and-Whey (CW) multivariate regression procedure, where

$$d_i = \frac{(1-r)(c_i^2 - r)}{(1-r)^2 c_i^2 + r^2(1 - c_i^2)}, \tag{5.9}$$

and where $r = \frac{p}{n}$.

5.5.2 Thresholding the Regression Output

In the regression approaches discussed in the previous section, of course we obtain real valued predictions via the function $\hat{f}(\mathbf{x}) = \hat{\mathbf{B}}^T\mathbf{x}$. Given an input vector $\mathbf{x}$, in order to predict the corresponding binary label vector $\mathbf{y}(\mathbf{x})$, we need to threshold the prediction output by the estimated regression function. Let $t_1, t_2, \ldots, t_K$ denote values such that

$$\mathbf{y}_k(\mathbf{x}) = I(\hat{f}_k(\mathbf{x}) > t_k), \tag{5.10}$$

where $k = 1, 2, \ldots, K$ and where $I(.)$ denotes the indicator function. That is, for every label, a different threshold value t_k is allowed.

In Bierman (2019), three approaches for data-dependent specification of label-specific thresholds are considered. The first approach is based upon the idea that the distribution of 0's and 1's in the training data should be similar to the distribution of 0's and 1's in the test data. The other two approaches

are both based upon the optimisation of a performance measure. The performance measure considered for this purpose during training, is the same as the performance measure used during test time. More specifically, in one of these two approaches, threshold values are found that optimise a performance measure on the training data. Therefore, the threshold values used to convert the real-valued output obtained for the test data are the same values that was used to convert the real-valued output obtained for the training data. In the other approach (also based upon the optimisation of a performance measure on the training data), the threshold value obtained during training is first converted to a quantile value. The corresponding quantile is then found on the test data. Hence, the threshold values used to convert the real-valued output obtained for the test data are not the same as the threshold values used for the training data.

In the empirical study in the Bierman (2019) paper, the performance of the above three thresholding approaches were also compared. Although differences among the separate approaches were not always very large, overall, the third approach was found to perform the best.

5.5.3 Interpretation of Predictions

The multivariate regression output includes a $p \times K$ matrix consisting of estimated regression coefficient values. An interpretation of the multi-label classifier can be achieved with the assistance of the information contained in the estimated coefficients. In the case of OLS regression, it is straightforward to obtain the standard errors of these coefficients. In the Bierman (2019) paper, heatmaps of standardised OLS coefficients were displayed, and interpreted in terms of the importance of features to the different labels. By means of resampling, it should be possible to also obtain estimated standard errors of the CW, FICYREG and RR coefficients, and to in a similar fashion, be able to interpret the regression coefficients obtained from these methods.

5.6 MLC Performance Measures

The multi-label data structure in MLC causes the usual classification performance measures, such as $0-1$ loss, not to be applicable in this context. Therefore, in MLC, a unique set of performance measures may be used in order to evaluate MLC models (Tsoumakas *et al.*, 2010). These measures may be divided into so-called example-based and label-based measures. Example-based measures apply a performance measure to each data observation (therefore row-wise in the $\mathbf{Y}$ matrix). The average performance over the n data cases is then reported as the final performance measure. In contrast, label-based measures apply a metric to each label (therefore column-wise in the $\mathbf{Y}$ matrix), and then obtains an average of these K performance values. Thus, these two kinds of performance metrics yield different information.

There are many proposed performance metrics for MLC since the performance of a learning algorithm

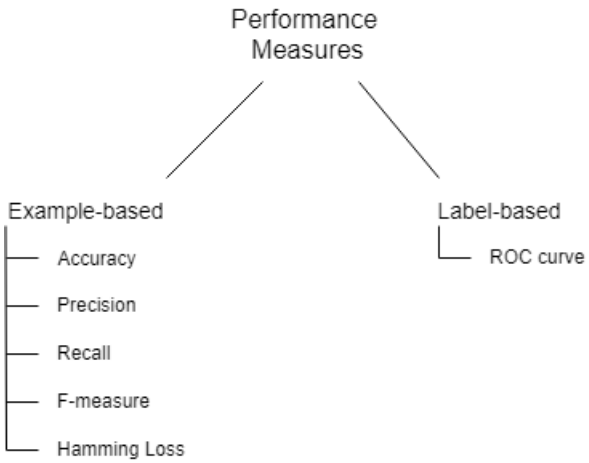


Figure 5.1: Performance measures for multi-label classification.

will vary from metric to metric (Wu and Zhou, 2017). Figure 5.1 gives examples of common performance measures used in multi-label classification. In the remainder of this section, we provide the formulae for five of the most frequently used example-based MLC performance measures. We also show how to extend binary classification evaluation measures to a multi-label setup. The label-based classification metrics, namely the macro- and micro-averaging metrics, are also defined.

Consider a test dataset, given by $\{(\tilde{\mathbf{x}}_i, \tilde{\mathbf{y}}_i), i = 1, 2, \ldots, \tilde{n}\}$, where $\tilde{\mathbf{x}}_i$ denotes the unseen observations of the inputs $X_1, X_2, \ldots, X_p$ and where $\tilde{\mathbf{y}}_i$ are the respective test labels of $Y_1, Y_2, \ldots Y_k$. This gives an $\tilde{n} \times (p + K)$ matrix $[\tilde{\mathbf{X}}\, \tilde{\mathbf{Y}}]$. Using a multi-label classification procedure, denote the obtained predicted label vectors for the test data by $\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \ldots, \hat{\mathbf{y}}_{\tilde{n}}$. Then the accuracy measure is given by

$$Accuracy \quad = \quad \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \left\{ \frac{\sum_{k=1}^{K} \tilde{y}_{ik} \hat{y}_{ik}}{K - \sum_{k=1}^{K} (1 - \tilde{y}_{ik})(1 - \hat{y}_{ik})} \right\}. \tag{5.11}$$

Precision is the average proportion of predicted labels that are correct. That is,

$$Precision \quad = \quad \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \left\{ \frac{\sum_{k=1}^{K} \tilde{y}_{ik} \hat{y}_{ik}}{\sum_{k=1}^{K} \hat{y}_{ik}} \right\}. \tag{5.12}$$

The average proportion of true values ($Y = 1$) that are correctly predicted is known as the recall, and is given by

$$Recall \quad = \quad \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \left\{ \frac{\sum_{k=1}^{K} \tilde{y}_{ik} \hat{y}_{ik}}{\sum_{k=1}^{K} \tilde{y}_{ik}} \right\}. \tag{5.13}$$

The harmonic mean of recall and precision is the F-measure, defined by

$$F\text{-}measure \quad = \quad \frac{2 \times precision \times recall}{precision + recall}. \tag{5.14}$$

Finally in this section, the fraction of incorrectly predicted labels is computed using the Hamming Loss measure, defined as

$$HL \quad = \quad \frac{1}{\tilde{n}K} \sum_{i=1}^{\tilde{n}} \sum_{k=1}^{K} I(\tilde{y}_{ik} \neq \hat{y}_{ik}). \tag{5.15}$$

		Predicted	
		$\hat{Y}=1$	$\hat{Y}=0$
Actual	$Y=1$	tp	fn
	$Y=0$	fp	tn

Figure 5.2: Confusion Matrix.

In a multi-label classification problem, it is possible to make two types of incorrect classifications, namely false positive (fp) and false negative (fn). A false positive classification occurs when the label is not present ($Y=0$), but the classifier predicts the label to be present ($\hat{Y}=1$). A false negative classification occurs when the label is present ($Y=1$), however the classifier predicts that the label is not present ($\hat{Y}=0$). A multi-label classifier can also make two types of correct classifications which are known as true positive (tp) and true negative (tn) classifications. If a label is present ($Y=1$) and the classifier correctly predicts that the label is present ($\hat{Y}=1$), a true positive classification is made. Similarly, a true negative classification is made when the label is not present ($Y=0$) and the classifier predicts that it is not present ($\hat{Y}=0$). Figure 5.2 depicts these errors in a confusion matrix. The receiver operating characteristic or ROC curve can be constructed using the tp and fp values. The larger the area under the curve, the better the performance of the classifier.

The evaluation measures that are used for binary classification can be extended for multi-label classification, and make use of the four types of classifications defined in Figure 5.2 (Giraldo-Forero *et al.*, 2015). Let $B(.)$ be a binary classification metric, and let tp_k, tn_k, fp_k and fn_k be the number of true positives, true negatives, false positives and false negatives, respectively, for a label k after a binary evaluation. Then, we can define the so-called macro- and micro-averaged label-based measures. The macro-averaged metric is given by

$$B_{macro} = B\left(\sum_{k=1}^{K} tp_k, \sum_{k=1}^{K} fp_k, \sum_{k=1}^{K} tn_k, \sum_{k=1}^{K} fn_k\right), \tag{5.16}$$

and the micro-averaged metric is given by

$$B_{micro} = \frac{1}{K}\sum_{k=1}^{K} B\left(tp_k, fp_k, tn_k, fn_k\right). \tag{5.17}$$

From Equation 5.16 it can be seen that the macro-averaged metric is calculated locally over each label. Therefore, it is more influenced by the performance of the classifier on minority labels. On the other hand, the micro-averaged metric is more influenced by the performance on commonly occurring labels. This is because it is calculated globally over all labels, as seen in Equation 5.17. Thus, label-based metrics are well suited to imbalanced MLC problems. However, as noted by Giraldo-Forero *et al.* (2015), the label dependencies are not represented by label-based metrics since the performance of each label is determined separately. Therefore, in our empirical analysis we will only make use of the example-based measures.

5.7 Summary

In this chapter, the idea of context recommendation was explored. We considered both pre- and post-filtering methods in order to determine which contexts are relevant for a domain in question. We focused on direct context recommendation, whereby combinations of contexts are represented as labels. Therefore, traditional recommendation methods were not considered for the recommendation of context, but rather multi-label classification procedures. Consequently, a large portion of this chapter was dedicated to a discussion regarding multi-label classification techniques. Three established MLC methods where described, as well as the recent multivariate regression approaches to MLC. The advantage of the multivariate regression approaches is that they assist with the interpretation of the multi-label classifier. Finally in this chapter, we described the two types of performance measures used in MLC problems, namely exampled-based and label-based measures.

Chapter 6

The Impact of Data Sparsity on Recommender Systems

6.1 Introduction

One of the biggest issues in recommender systems is that only a small proportion of all available items have been rated by each user. Therefore the ratings matrix contains a large number of missing ratings, which causes a number of problems, and which potentially may have a very negative impact of the quality of recommendations. More specially, the cold-start problem, the reduced coverage problem and the neighbour transitivity problem are all consequences of sparse ratings matrices in recommender systems. As discussed in Chapter 3, CF systems suffer from the cold-start problem as they cannot recommend an item to a user if it has never received a rating before. Furthermore, a large number of unrated items diminishes the coverage of the system, implying that the RS is only able to produce predictions for a very small number of items. In sparse settings, memory-based CF suffers severely from the neighbour transitivity problem. This problem arises since the neighbourhood of common items between users is small, therefore the system often fails to recognise similar users.

In this chapter, we consider the problem of data sparsity and the effect it has on various recommendation techniques. We compare the performance of some of the traditional collaborative filtering methods (discussed in Chapter 3), with the performance of some of the more advanced latent factor models (discussed in Chapter 4) using simulated datasets with varying levels of sparsity. We hereby aim to show that in sparse settings, latent factor models (in particular factorisation machines) produce higher quality predictions than the simpler collaborative filtering approaches (Rendle, 2012). Moreover, we provide a guide as to which recommender models to use, depending on the level of sparsity of the dataset.

The remainder of this chapter is organised as follows. In Section 6.2, we provide an overview of related work. In Section 6.3, an overview of a brief exploratory analysis of the data to be used, is given. The

experimental design of our study is discussed in Section 6.4, followed by a discussion of the results in Section 6.5. We close the chapter with a summary in Section 6.6.

6.2 Related Work

Three recent contributions with respect to specifically investigating the impact of data sparsity on recommender systems are the papers by Hwangbo and Kim (2017) and da Silva *et al.* (2018), as well as the master's study by Strömqvist (2018). In the paper by da Silva *et al.* (2018), it is stated that the effects of sparsity on recent recommendation methods are still unclear. Focusing on non-negative matrix factorisation, singular value decomposition and stacked autoencoders in deep learning, the authors investigate the effects of data sparsity on the RMSE and MAE measures. The empirical work in this study is based upon the MovieLens 100K dataset. In the paper by Hwangbo and Kim (2017), the authors investigate the impact of various levels of data sparsity and degrees of data overlap on the performance of recommender systems based upon cross-domain collaborative filtering. In their paper, the performance measures considered were precision and recall. The authors make use of Korean fashion retail datasets in order to conduct their analyses. In the Strömqvist (2018) study, the focus is specifically on the impact of data sparsity in the case of matrix factorisation methods for recommender systems. In the analysis, Strömqvist makes use of the MovieLens 100K dataset, and of the RMSE as a performance metric.

In the literature, a few papers may be found where the aim is to propose recommender techniques that are not prone to suffer the negative consequences of data sparsity (*cf.* for example Huang *et al.*, 2004; and Alqadah *et al.*, 2015). From the nature of the objective of these papers, they include empirical evaluations of the effects of sparsity on the proposed methods, and on the methods with which they are compared. Note that these contributions mainly focus on remedies to collaborative filtering methods. Finally, in this section, in the paper by Lee *et al.* (2012), a comprehensive comparative study of collaborative filtering algorithms is reported. The CF algorithms included in the study are matrix factorisation-based and memory-based CF, and the CF proposals by Lemire and Maclachlan (2005), Yu *et al.* (2009), and Sun *et al.* (2011). Importantly, note that in their study, Lee *et al.* (2012) take cognisance of the effects of the number of users and items, as well as of the effects of sparsity on each CF algorithm considered.

Motivated by the above studies, and our discussions in Chapter 4, we recognise that there is scope for an extended investigation of the effects of data sparsity on recommender systems. That is, the evaluation of CF algorithms on varying levels of sparsity, as well as the evaluation of the more recent factorisation machines.

6.3 Exploratory Data Analysis

We consider two datasets in our analysis, *viz.* the MovieLens dataset (Harper and Konstan, 2015) and the LDOS-CoMoDa dataset (Kosir *et al.*, 2011). Table 6.1 provides a comparison of the descriptive statistics of both datasets.

Table 6.1: Description of the two datasets: MovieLens and LDOS-CoMoDa.

	MovieLens	**LDOS-CoMoDa**
Number of users	610	121
Number of items	9742	1232
Number of ratings	100836	2296
Sparsity	98.3%	98.4%
Rating scale	0.5 - 5	1 - 5

6.3.1 The MovieLens Dataset

The Department of Computer Science and Engineering at the University of Minnesota has a research lab called GroupLens. Recommender systems, among other areas of research, have been intensively studied by GroupLens data scientists. The MovieLens website is a site run by GroupLens, offering movie recommendations to its users. The MovieLens data is publicly available from the website, and contains both user ratings and user tags. Note that there are four different sizes of MovieLens datasets available, *viz.* 100K, 1M, 10M and 20M datasets.

In order to limit computation time, we focus on the MovieLens 100K dataset, which was collected between 29 March 1996 and 24 September 2018, and was generated on 26 September 2018. As indicated in Table 6.1, it contains 100836 ratings and 3683 tags provided by $n = 610$ users, related to $m = 9742$ movies. The data are contained in four files, and in this chapter, we focus only on one of them, that is on the `ratings.csv` file. Table 6.2 shows the format of the ratings file. The users represented in the dataset were randomly selected, however note that a user had to have rated at least 20 movies. Each user is represented by a unique ID, and no other information (age, gender, race etc.) regarding the user is available. A movie was included in the dataset if it had at least one rating or at least one tag. Each movie is also assigned a unique ID. Each row represents a user's rating of a movie. The ratings are based on a 10-star scale, ranging from 0.5 stars to 5 stars, in increments of 0.5, with a higher value indicating a better rating. The ratings are ordered by User ID, and and within each user, by Movie ID. A timestamp of the rating indicates the number of seconds since midnight of 1 January 1970 (Coordinated Universal Time or UTC).

A frequency barplot of the ratings in the MovieLens 100K dataset is given in Figure 6.1. Clearly, the most common ratings are 3 and 4 stars.

Table 6.2: Format of the MovieLens rating file.

User ID	Movie ID	Rating	Timestamp
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
1	47	5.0	964983815
1	50	5.0	964982931

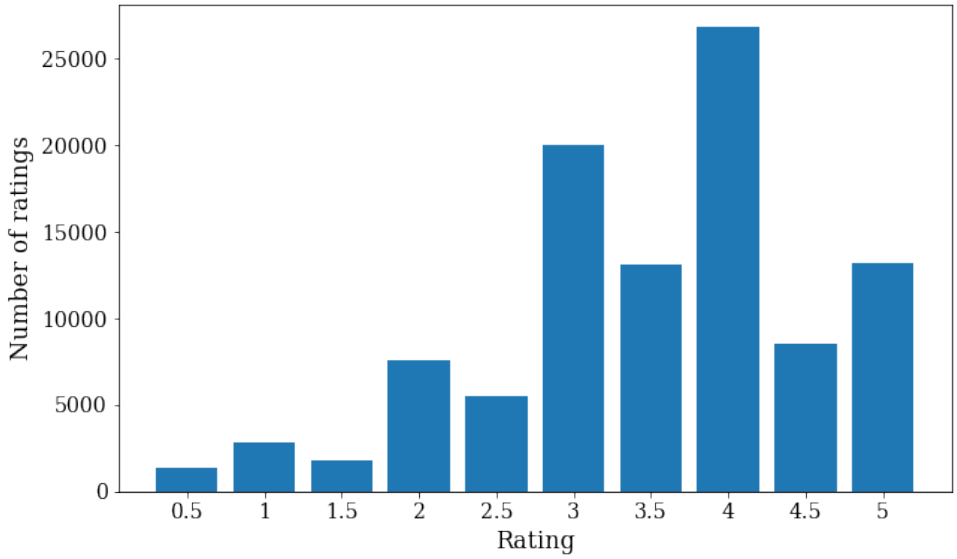


Figure 6.1: Frequency barplot of the MovieLens 100K ratings.

6.3.2 The LDOS-CoMoDa Dataset

The LDOS-CoMoDa dataset also contains user ratings of movies. From Table 6.1 it can be seen that this dataset, with 2296 ratings from $n = 121$ users on $m = 1232$ items, is much smaller than the MovieLens dataset. However, the level of sparsity of the two dataset is very similar, *viz.* 98.3% and 98.4%. As in the case of the MovieLens data, we will focus on the LDSO-CoMoDa ratings, although it should be noted that the LDOS-CoMoDa dataset also contains additional features capturing contextual information. The LDOS-CoMoDa dataset will feature again in Chapter 7.

Table 6.3: Format of the LDOS-COMODA rating file.

User ID	Movie ID	Rating
15	178	4
15	4101	5
15	250	5
15	249	5
15	248	5

Users were included in the LDOS-CoMoDa dataset if they made use of the web application designed by Kosir *et al.* (2011), and were subsequently assigned a unique ID. Movies where included if they received a

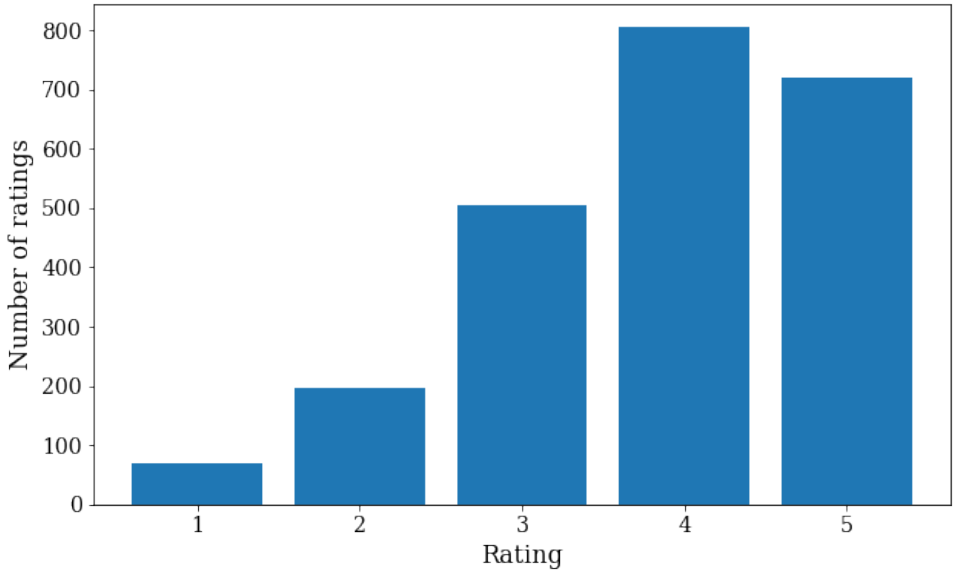


Figure 6.2: Frequency barplot of the LDOS-CoMoDa ratings.

rating from a user and each movie was also assigned a unique ID. The ratings are based on a 5-star scale, ranging from 1 stars to 5 stars, in increments of 1, with a higher value indicating a better rating. The format of the data is given in Table 6.3, while the distribution of the ratings is displayed in Figure 6.2, showing that the most common rating given to a movie is a 4, followed by a rating of 5.

6.4 Experimental Design

Parts of our experimental design is based upon the settings chosen in Lee *et al.* (2012) and Strömqvist (2018). Similarities to their experimental design, and departures from it, will be pointed out where appropriate. Note that we discuss flow of our empirical work in four parts. We start with a discussion of the creation of dense matrices from existing datasets, followed by the way in which to sample matrices to have varying pre-specified levels of sparsity. The algorithms to be evaluated and compared are then given, and finally, their predictive performance on the varying levels of sparsity is evaluated. Also note that all programming code relating to this chapter may be found in Appendix A.

We proceed with a more detailed description of our experimental setup in Sections 6.4.1-6.4.4. The generation of dense matrices is detailed in Section 6.4.1, while the sampling step is discussed in Section 6.4.2. The algorithms to be considered, and their evaluation, is described in Sections 6.4.3 and 6.4.4 respectively.

6.4.1 Generating a Dense Ratings Matrix

In order to compare the performance of different algorithms on varying levels of data sparsity, a dense rating matrix is first required. The datasets that we consider in this study (and in general in recommender systems) are of course far from dense. The MovieLens and LDOS-CoMoDa datasets have sparsity levels

98.3% and 98.4%, respectively, where the sparsity level of an $n \times m$ ratings matrix $\mathbf{R}$ is defined as

$$sparsity\ level \quad = \quad 1 - \frac{r}{nm}. \tag{6.1}$$

Here r is the number of observed ratings in $\mathbf{R}$, n is the total number of users and m is the total number of items (Lee *et al.*, 2012). Therefore, prior to our experimental analysis, we need to first transform these sparse datasets into dense datasets.

To create a dense matrix, we use a similar method as in Strömqvist (2018). The matrix factorisation model, given in Equation 4.6, is used to estimate the missing ratings. To ensure that as much information is retained in the approximation, we chose the number of latent factors to be $k = 100$. The dense matrices may subsequently be used to generate sparse datasets by means of sampling user-item pairs, which is described in Section 6.4.2.

6.4.2 Sampling a Sparse Ratings Matrix

Strömqvist (2018) considers four sparsity levels (95%, 97%, 98% and 99%) on the MovieLens 100K dataset, while Lee *et al.* (2012) considers a range of sparsity levels from 95% to 99.5% on the Netflix dataset. From this, we chose to consider two levels of sparsity for the MovieLens data, namely 98% and 99%, and three levels of sparsity for the LDOS-CoMoDa data, namely 95%, 98% and 99%. For each sparsity level, and for each of the created dense matrices, five sparse matrices are obtained by sampling the required number of user and item pairs from the corresponding dense matrix.

As noted by Strömqvist (2018), when creating sparse matrices, an important consideration is the underlying structure of the original dataset. For example, popular items are more likely to have a higher number of ratings than less popular items. There also may be some users who rate many more items than others. If we randomly sample user-item pairs from the dense matrix, these characteristics of the original will not be preserved. Therefore, we need to sample user-item pairs in such as way as to capture these properties.

One way in which the underlying properties of the original data can be preserved is to consider the proportion of ratings given by each user, as well as the proportions of ratings received by each item, as observed in the original dataset. Sampling user and item pairs from the created dense matrix may subsequently be guided by these proportions, by using them as sampling probabilities.

For each sparsity level, five sparse matrices are generated. Each time a new sparse matrix is generated, the proportions are permuted. This is done to ensure that the same user and item are not selected with the same probability for each sparse matrix. In other words, we are concerned with preserving the rating *proportions* of the users and items as observed in the original data. We definitely do not want to select the

same user and item pairs for each matrix. For illustration purposes, we consider the following example based upon the MovieLens 100K dataset.

Example 6.1. As noted previously, the original MovieLens 100K dataset, consists of $n = 610$ users, $m = 9742$ items and 100836 observed ratings. Using matrix factorisation, a dense matrix may be created. This matrix consists of $610 \times 9724 = 5931640$ ratings. Suppose that we wish to generate two matrices, both having a sparsity level of 99%. This implies that we need to sample $5931640 \times (1 - 0.99) \approx 59316$ user-items pairs from the dense matrix. Before sampling, we need to record the user rating proportions observed in the original MovieLens matrix. These are given in Table 6.4.

Table 6.4: Proportions of user ratings in the original MovieLens dataset.

User ID	Number of ratings	Rating proportion
0	323	0.0023
1	29	0.0003
2	39	0.0004
⋮	⋮	⋮
608	37	0.0004
609	1302	0.0129

The proportions given in Table 6.4 are used to sample users for the sparse matrices. Table 6.5 demonstrates how the user proportions are permuted before a new matrix is generated. It can be seen that for the first matrix, the original user proportions given in Table 6.4 may be used. User 0 will be selected for the sparse matrix, according to the rating proportion, with probability 0.0023. For the second matrix, the rating proportion of User 0 is 0.0013, and therefore the probability of selecting this user for the second matrix has decreased. Additionally, since the rating proportions are permuted, User 608 will be selected for the second matrix with probability 0.0023, which is the same probability of selecting User 0 for the first matrix.

Table 6.5: Selecting users for different sparse matrices.

Matrix 1		Matrix 2	
User ID	**Rating proportion**	**User ID**	**Rating proportion**
0	0.0023	0	0.0013
1	0.0003	1	0.0002
2	0.0004	2	0.0005
⋮	⋮	⋮	⋮
608	0.0004	608	0.0023
609	0.0129	609	0.0003

△

6.4.3 Algorithms

We consider a total of ten algorithms in our study. These are presented in Table 6.6, grouped according to category. Some of the algorithms in the categories 'memory-based' and 'factorisation models' were also considered by Lee *et al.* (2012). We take into consideration an additional category, namely 'advanced methods', which include factorisation machines and field-aware factorisation machines. The third column indicates the Python library used for the algorithm implementation. The names given to the algorithms in Table 6.6 are based on the names given in the respective libraries. Therefore, the reader should keep in mind that the SVD (unbiased) algorithm and the SVD (biased) algorithm are in fact the unbiased and biased matrix factorisation models discussed in Section 4.3.1.

Although not indicated in Table 6.6, both memory-based models and the NMF model did not include user and item bias for computational purposes. Additionally, since we are only making use of the ratings as inputs to the algorithms in this study, the FM and FFM models are mimicking the SVD models, as noted by Rendle *et al.* (2011).

Table 6.6: Algorithms used in experiments.

Category	**Algorithm**	**Library**
Baseline	Baised rating	Surprise
Memory-based	CF (user-based)	
	CF (item-based)	
Factorisation Models	SVD (unbiased)	
	SVD (biased)	
	SVD++	
	NMF	
Advanced Methods	FM	xLearn
	FFM	

6.4.4 Evaluation

The evaluation of the algorithms in Table 6.6 comprises three steps: splitting a sparse matrix into training and test sets, hyper-parameter tuning via cross-validation, and the evaluation of the predictive performance of the trained model. This process is discussed below.

As previously mentioned, five sparse matrices, which may be denoted by $M_i, i = 0, 1, \ldots, 4$, are generated for each sparsity level. To explain our method of evaluation, we consider the CF (user-based) algorithm on sparsity level 99%.

Consider the first sparse matrix M_0. This matrix is first split into a training set Tr_0 and test set Te_0, which contain 90% and 10% of the observations in M_0, respectively. Next, the training set Tr_0 is used to tune the hyper-parameter of the algorithm. The only hyper-parameter of the CF algorithm is the number of neighbours (k). The values considered for k are $k = 5\,, 15\,, 25$ and 50.

Starting with $k = 5$, we evaluate the CF model by means of three-fold cross-validation using Tr_0. For each fold, the algorithm is evaluated based on the RMSE. The predictive performance of the CF algorithm with $k = 5$ is the average RMSE over the three folds. This process is repeated for all values of k. The value of k that gave the lowest average RMSE during cross-validation is the chosen number of neighbours.

The algorithm is then refit with the chosen k value on the entire training set Tr_0. The test set Te_0 is used to compute the predictive performance of the algorithm. We make use of RMSE and MAE to evaluate the predictions on the test set.

The above process is repeated for the remaining sparse matrices $M_i, i = 1, 2, 3, 4$. The reported metrics for the sparsity level are the averaged RMSE and MAE across the five matrices. The hyper-parameters that achieved the best RMSE during cross-validation are reported in Appendix B. We repeat the procedures described for each level of sparsity, as well as for each of the algorithms in Table 6.6.

6.5 Results

6.5.1 The MovieLens Dataset

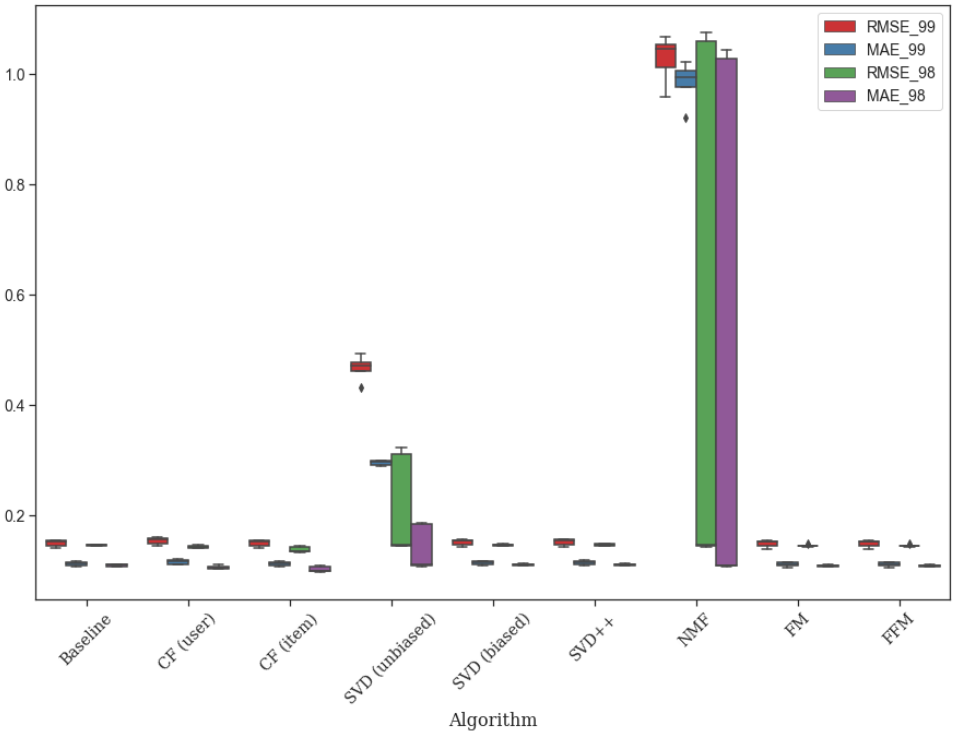


Figure 6.3: Boxplots of the RMSE and the MAE obtained by different algorithms on sparsity level 99% and 98% on the MovieLens dataset.

Consider Figure 6.3, which displays the boxplots of the RMSE and MAE under varying levels of sparsity, grouped by algorithm, for the MovieLens dataset. Due to the unbiased SVD and NMF algorithms attaining significantly different results compared to the remaining algorithms under evaluation, the spread

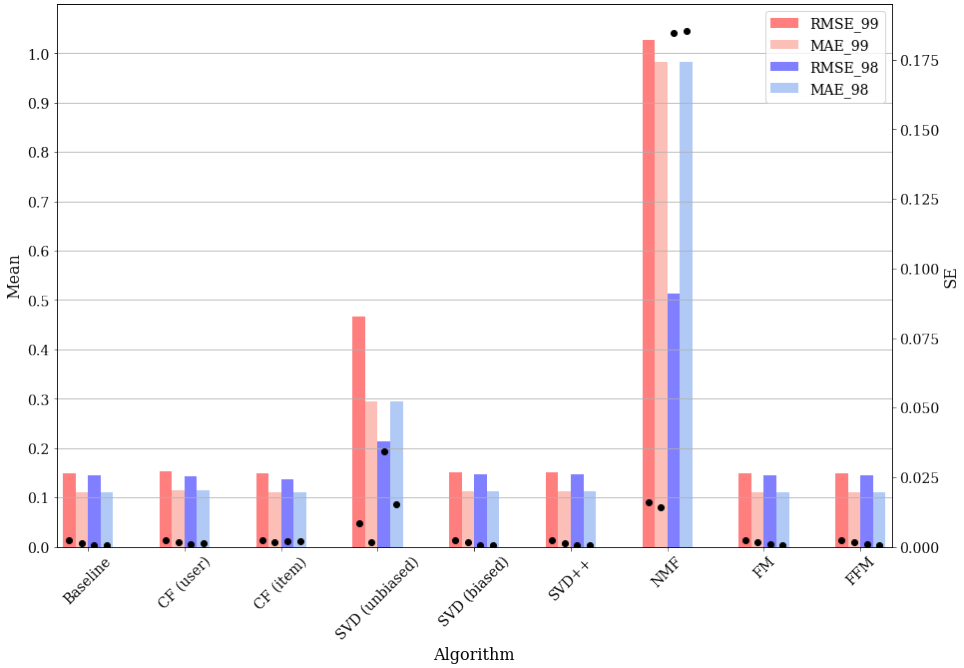


Figure 6.4: The mean and the standard error of the RMSE and the MAE obtained by different algorithms on sparsity level 99% and 98% on the MovieLens dataset. The bars indicate the mean while the dots indicate the standard error.

of these remaining algorithms cannot clearly be seen in Figure 6.3. Therefore, while still displaying all algorithms for comparative purposes, we include Figure 6.4. This shows the mean attained for the metrics under different levels of sparsity (bars) and the standard error thereof (dots). Clearly, apart from the unbiased SVD algorithm and the NMF algorithm, all the algorithms perform very similarly. The unbiased SVD algorithm and the NMF algorithm are the two worst performing algorithms in terms of both metrics (and both levels of sparsity). Interestingly, the spread of the RMSE and MAE for both these models under a sparsity level of 98% is significantly larger compared to the a sparsity level of 99%. This may indicate unstable models. Since both the unbiased SVD and NMF algorithms perform worse than the simple baseline algorithm, an important conclusion can be drawn: incorporating user and item bias does play a large role in producing accurate results. This is made evident by the fact that the biased SVD attains a much lower average RMSE and average MAE value than the former two models.

Table 6.7 contains the mean and standard error values of the RMSE and MAE obtained by each algorithm under the different levels of sparsity. It also includes the average times (in seconds) taken for the models to run (including hyper-parameter tuning, training and testing). The bold values indicate the lowest mean RMSE and mean MAE. We also highlight the algorithm with the shortest run time.

Based on Table 6.7, we observe that all models perform better at a lower level of sparsity. This is to be expected since at a lower level of sparsity, there are more ratings present and therefore more information is available for the models to learn from. The simple models, such as the baseline and CF models, perform surprisingly well at high sparsity levels. Based on the average RMSE, the best performing model at a

Table 6.7: The mean and standard error of the RMSE and MAE obtained by different algorithms on sparsity level 99% and 98% on the MovieLens data. The average time (in seconds) it took the algorithms to run is given in the last column.

Sparsity: 99%					
	RMSE		MAE		
Algorithm	**Mean**	**SE**	**Mean**	**SE**	**Time**
Baseline	0.1483	0.0024	0.1112	0.0015	29.0973
CF (user)	0.1529	0.0024	0.1146	0.0015	25.4242
CF (item)	0.1484	0.0024	0.1113	0.0015	90.4452
SVD (unbiased)	0.4662	0.0083	0.2945	0.0018	204.2183
SVD (biased)	0.1504	0.0024	0.1131	0.0015	208.6893
SVD++	0.1504	0.0024	0.1132	0.0014	7290.0472
NMF	1.0264	0.0159	0.9829	0.0141	337.8787
FM	**0.1480**	0.0024	**0.1106**	0.0015	16.3066
FFM	**0.1480**	0.0024	0.1480	0.0015	**15.9468**
Sparsity: 98%					
	RMSE		MAE		
Algorithm	**Mean**	**SE**	**Mean**	**SE**	**Time**
Baseline	0.1449	0.0006	0.1088	0.0005	55.7826
CF (user)	0.1419	0.0010	0.1048	0.0011	62.1334
CF (item)	**0.1370**	0.0020	**0.1012**	0.0020	256.8961
SVD (unbiased)	0.2132	0.0344	0.1390	0.0153	412.0770
SVD (biased)	0.1459	0.0006	0.1097	0.0005	421.1836
SVD++	0.1460	0.0006	0.1098	0.0006	29686.9596
NMF	0.5130	0.1845	0.4787	0.1853	625.5833
FM	0.1446	0.0009	0.1085	0.0006	33.3736
FFM	0.1446	0.0009	0.1085	0.0006	**31.9338**

sparsity level of 98% is the item-based collaborative filtering model, with an average RMSE of 0.1370, closely followed by the user-based CF model having an average RMSE of 0.1419. The FM model and FFM model are tied for third best model, both having an average RMSE of 0.1446. The same conclusions are drawn from the average MAE. At a sparsity level of 99%, according to the average RMSE, the best performing models are the FM and the FFM models, both having an average RMSE of 0.1480. Studying the average MAE, we see that the best performing model is the FM. For both levels of sparsity, the fastest algorithm to implement was the FFM, followed by the FM.

Consider Figure 6.5, which displays the four best performing algorithms together with the baseline for comparative purposes. Clearly all models perform similarly, even in terms of variability. However, while the user- and item-based models perform better at a lower level of sparsity, they take considerably longer to run than the latent factor models, with the item-based model taking over four minutes to complete. The RMSE and MAE of the FM and FFM are only marginally worse than those obtained by the item-based CF at a sparsity level of 98%. Furthermore, as the sparsity of the matrix increases, the FM and FFM model outperforms the simpler models, both in terms of accuracy and run time. Therefore, the latent factor models are clearly the top performers.

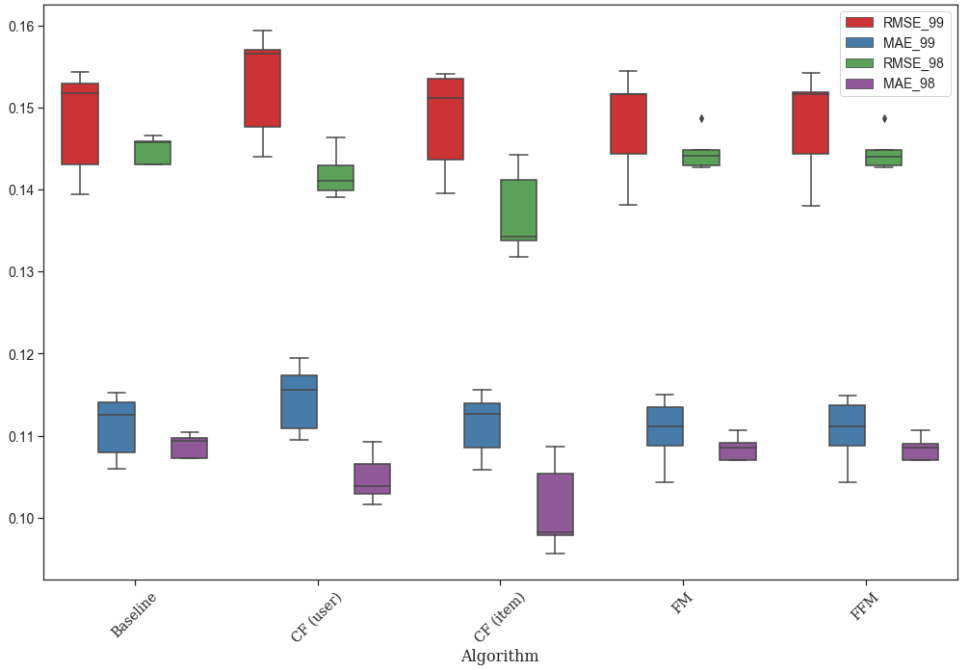


Figure 6.5: Boxplots of the RMSE and the MAE for algorithms having either the lowest RMSE, MAE or run time on sparsity level 99% and 98% on the MovieLens dataset.

6.5.2 The LDOS-CoMoDa Dataset

The LDOS-CoMoDa dataset is significantly smaller than the MovieLens dataset. Due to this, the run times of the algorithms are significantly reduced and we were therefore able to apply the algorithms to an additional level of sparsity.

We first consider Figure 6.6, which displays the boxplots of the RMSE and MAE under varying levels of sparsity, grouped by algorithm, for the LDOS-CoMoDa dataset. We can draw similar conclusions as we did from the results on the MovieLens dataset. The unbiased SVD algorithm and the NMF algorithm are the two worst performing algorithms in terms of both metrics (and at all levels of sparsity). On the LDOS-CoMoDa dataset, however, the spread of the RMSE and MAE for the unbiased SVD under a sparsity level of 99% is significantly larger compared to the lower levels of sparsity. This reinforces the suggestion that this algorithm has high variability. Again we see that incorporating user and item bias does play a large role in producing accurate results, since the biased SVD attains a much lower average RMSE and average MAE value than the former two models.

For all algorithms, we see from Figure 6.7 that as sparsity decreases, there is a small decrease in the mean RMSE and mean MAE (as well as the standard error) indicating that there is less variability in accuracy as sparsity decreases. For its simplicity, the baseline algorithm achieves surprisingly good results at a high level of sparsity, however the standard errors are marginally larger than is the case with the FM and the FFM.

Despite not taking biases into account, both user- and item-based collaborative filtering algorithms pro-

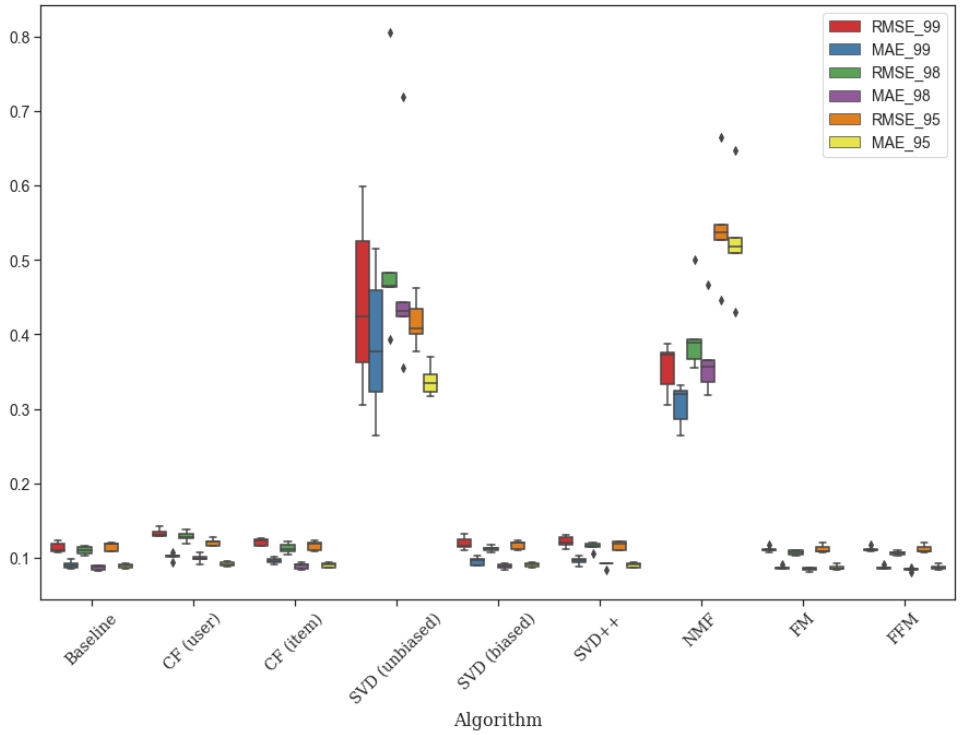


Figure 6.6: Boxplots of the RMSE and the MAE obtained by different algorithms on sparsity level 99%, 98% and 95% on the LDOS-CoMoDa dataset.

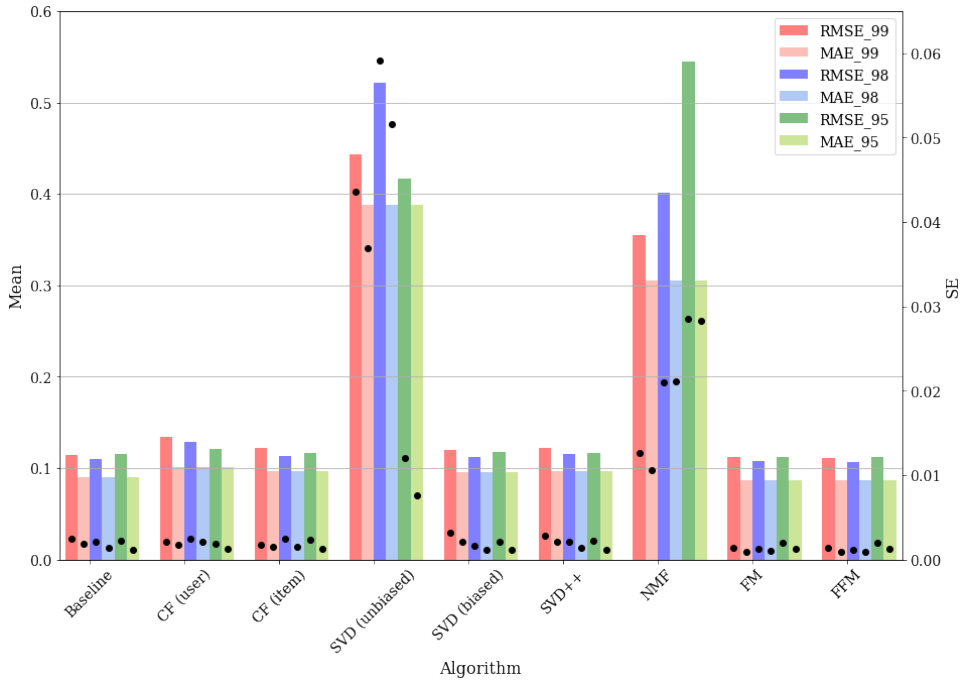


Figure 6.7: The mean and the standard error of the RMSE and the MAE obtained by different algorithms on sparsity level 99%, 98% and 95% on the LDOS-CoMoDa dataset. The bars indicate the mean while the dots indicate the standard error.

duce relatively small RMSE and MAE values. However, the top performing algorithms are the FM and FFM, having a lower mean for both metrics compared to remaining algorithms, suggesting that they are able to perform better under high levels of sparsity.

Table 6.8: The mean and standard deviation of the RMSE and MAE obtained by different algorithms on sparsity level 99%, 98% and 95% on the LDOS-CoMoDa dataset. The average time (in seconds) it took the algorithms to run is given in the last column.

Sparsity: 99%					
	RMSE		MAE		
Algorithm	**Mean**	**SE**	**Mean**	**SE**	**Time**
Baseline	0.1139	0.0025	0.0905	0.0019	0.8938
CF (user)	0.1337	0.0022	0.1015	0.0018	**0.6369**
CF (item)	0.1218	0.0018	0.0965	0.0015	1.1211
SVD (unbiased)	0.4434	0.0436	0.3877	0.0369	5.2274
SVD (biased)	0.1196	0.0032	0.0955	0.0021	5.1171
SVD++	0.1220	0.0029	0.0962	0.0021	36.6128
NMF	0.3548	0.0126	0.3054	0.0106	14.9847
FM	0.1116	0.0014	0.0872	0.0009	3.5420
FFM	**0.1116**	0.0014	**0.0871**	0.0010	1.3415
Sparsity: 98%					
	RMSE		MAE		
Algorithm	**Mean**	**SE**	**Mean**	**SE**	**Time**
Baseline	0.1100	0.0021	0.0868	0.0014	1.8313
CF (user)	0.1286	0.0025	0.0998	0.0022	**1.5300**
CF (item)	0.1132	0.0025	0.0892	0.0015	2.8243
SVD (unbiased)	0.5218	0.0591	0.4745	0.0516	10.7751
SVD (biased)	0.1125	0.0016	0.0887	0.0012	11.7862
SVD++	0.1153	0.0021	0.0911	0.0013	146.2609
NMF	0.4008	0.0210	0.3691	0.0211	29.1378
FM	0.1072	0.0013	0.0849	0.0010	1.8558
FFM	**0.1067**	0.0011	**0.0843**	0.0009	1.7620
Sparsity: 95%					
	RMSE		MAE		
Algorithm	**Mean**	**SE**	**Mean**	**SE**	**Time**
Baseline	0.1152	0.0022	0.0894	0.0011	3.2661
CF (user)	0.1206	0.0018	0.0917	0.0013	3.1894
CF (item)	0.1164	0.0024	0.0903	0.0012	8.4866
SVD (unbiased)	0.4166	0.0120	0.3380	0.0077	19.0467
SVD (biased)	0.1176	0.0021	0.0910	0.0011	19.2490
SVD++	0.1168	0.0022	0.0906	0.0012	627.7266
NMF	0.5445	0.0285	0.5270	0.0283	32.4978
FM	0.1124	0.0020	0.0872	0.0013	3.8477
FFM	**0.1124**	0.0020	**0.0872**	0.0013	**3.1105**

Table 6.8 contains the mean and standard error values of the RMSE and MAE obtained by each algorithm under the different levels of sparsity. It also includes the average times (in seconds) taken for the models to run. The bold values indicate the lowest mean RMSE and mean MAE. We also highlight the algorithm with the shortest run time. Based on this table, we deduce that the top performing algorithms in terms of the mean metrics are the FFM, FM and the baseline models. Based on time, the best performing models are a mixture of the FFM, both user and item CF models, as well as the baseline model. Therefore, based on these results, and for further comparisons, we constructed Figure 6.8.

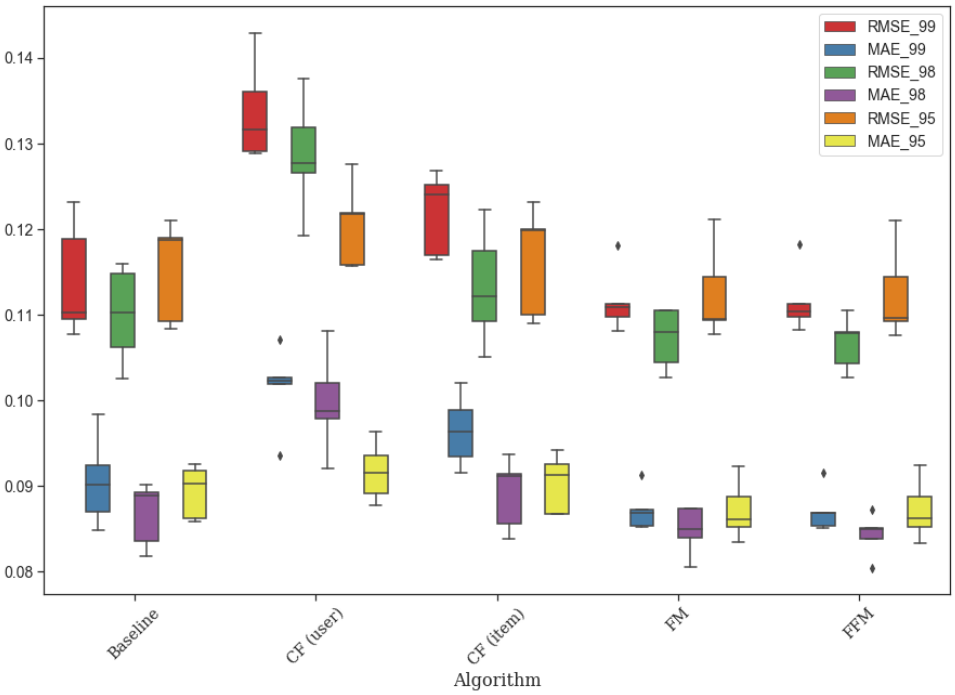


Figure 6.8: Boxplots of the RMSE and MAE for algorithms having either the lowest RMSE, MAE or run time on sparsity level 99%, 98% and 95% on the LDOS-CoMoDa dataset.

While both the collaborative filtering models have a short run time, from Figure 6.8 we see that they perform worse in terms of both metrics compared to the other three models. The baseline model outperforms the CF models, however the spread is larger than that of the FM and the FFM. Between the FM and the FFM, the latter achieves better results. In fact, based on the spread of both metrics, it performs better at a sparsity level of 99% than 95%, indicating that this model is ideal for high levels of sparsity.

From Table 6.8, at a sparsity of 99% the FFM does takes longer to run than the simple baseline, which achieves similar values for the metrics. If we consider the run times of the algorithms on the MovieLens data in Table 6.7, we note that at both levels of sparsity, the FFM and FM models have the shortest run time. Since these datasets vary in size, these results suggest that the size of the matrix also should be considered when selecting a model. When the ratings matrix is sparse and very large, the factorisation machine models are an obvious choice. However, when the ratings matrix is sparse but relatively small in size, the simpler models should not be ignored.

6.6 Summary

One of the key problems when designing a recommender system is that the rating matrix is typically very sparse. Therefore, in order to aid in the selection process of algorithms to use in an RS, in this chapter we conducted a comparative study of the effects of sparsity on different algorithms. To do this, we considered the MovieLens and LDOS-CoMoDa datasets, from which we generated matrices of varying levels of sparsity. Using the generated sparse matrices, we applied simple collaborative filtering

algorithms as well as the more complex latent factors models, the latter of which included the more recent factorisation machines. The predictive performance of these algorithms were compared using the accuracy metrics RMSE and MAE, as well as the run time of the algorithm.

According to our results, we can conclude that the FFM and the FM models are the better choice at a higher level of sparsity. Simpler models perform competitively with these more advanced models as more information becomes available, as well as when the size of the dataset is small. Therefore, at lower levels of sparsity, simpler models should not be ignored.

Chapter 7

Multi-Label Classification for Context Recommendation

7.1 Introduction

This chapter is devoted to empirical work in a rather niche area of research on recommender systems. As noted in Chapter 5, both context recommendation and the use of multi-label classification approaches in an RS setup are topics which, in our opinion, have not yet fully been explored in the literature. For example, with regard to work on context recommendation, we could find only the contributions by Baltrunas *et al.* (2010) and Benmansour *et al.* (2015). In terms of work on the use of MLC approaches in general recommender problems, the reader may refer to the papers by Rivolli *et al.* (2017 and 2018). Most relevant to our work, however, is the paper by Zheng *et al.* (2014), wherein the authors investigate the use of MLC for recommending appropriate contexts for movies and travel destinations.

The objective of the empirical study reported in this chapter, is to expand upon the work in the latter paper. Specifically, we extend the algorithms considered to also include modifications to multivariate linear regression. In these modifications, potential dependencies among the labels are exploited, and typically leads to improvements in prediction accuracy. An important advantage of the regression approaches is that their output includes estimated regression coefficients which may provide valuable information for feature selection, or for more explainable recommendations. Since in the literature, some work is being done on finding recommender systems that are easily explainable to users, we regard investigation of the use of regression for MLC in an RS context to be a worthwhile endeavour.

The performances of the MLC algorithms in Chapter 5 are compared on two real world datasets, *viz.* the already encountered LDOS-CoMoDa dataset (Kosir *et al.*, 2011), and the so-called TripAdvisor dataset (Zheng *et al.*, 2012). Note that both these datasets may be considered as so-called context-aware datasets.

The remainder of the chapter is structured as follows. A description of the LDOS-CoMoDa and TripAd-

visor datasets is provided in Section 7.2. A discussion of the experimental design of our study follows in Section 7.3. Our empirical results are reported and briefly discussed in Section 7.4. We also compare our results to those obtained in the paper by Zheng *et al.* (2014). We conclude the chapter in Section 7.5, with a summary of our main findings.

7.2 Datasets

7.2.1 The LDOS-CoMoDa Dataset

The LDOS-CoMoDa dataset is a relatively small context-aware dataset relating to movies rated by users. The data was collected by Kosir *et al.* (2011) using web surveys, and is freely available[1]. Several versions of the LDOS-CoMoDa dataset may be found on the internet. The version of the dataset that was used in our study contains 121 users, 1232 items (movies) and 2296 ratings. The ratings range from 1 to 5, in one step increments. There are 30 variables in total, consisting of user information, movie content, as well as contextual information. There are a total of 12 contextual variables which describe the context in which the user watched the movie. A detailed description of all available variables can be found in Kosir *et al.* (2011).

7.2.2 The TripAdvisor Dataset

This is a context-aware dataset containing information of hotel reviews. The data was complied by Zheng *et al.* (2012) after scraping the travel website TripAdvisor. There are a total of 14175 hotel ratings. These ratings also range from 1 to 5, in one step increments. The dataset contains ratings on 7269 hotels, provided by 2371 users. There are 6 variables, describing user location (state and timezone) and the hotel location (state, city and timezone). The remaining variable is the contextual variable, TripType, which describes the type of trip the user was on. As users tend not to stay in the same hotel under different contexts, or leave a review each time they stay in a hotel, note that the dataset is sparse with respect to the context.

7.3 Experimental Design

In our analysis, not all variables in the LDOS-CoMoDa dataset were considered. We based the selection of variables on those chosen by Zheng *et al.* (2014). Therefore, four context variables are used, namely time, location, daytype and social, together with a number of content features. The content features are user gender, movie country, movie year, genre and movie language. For the TripAdvisor dataset, however, we do make use of all available content and context variables.

[1] http://212.235.187.145/spletnastran/raziskave/um/comoda/comoda.php

In the LDOS-CoMoDa dataset, a number of observations had missing values for the context variables. Since we are trying to predict the appropriate context for a user to watch a movie, these observations are not useful and were removed. One observation had missing values for both genre and movie language. There were three other observations that had either missing values for genre or for movie language. These four observations were removed. The TripAdvisor dataset did not contain any missing values, and therefore no observations were removed.

Recommendation of appropriate contexts for an item is based on the assumption that the user would enjoy the item. Therefore, the datasets were split into positive and negative ratings. Ratings above 3 were considered positive, while ratings less than and equal to 3 were considered negative. Furthermore, in both datasets, there were nominal input variables. It was necessary to label-encode these variables.

Table 7.1 provides a description of the datasets. The values in brackets next to the context variables indicate the number of dimensions that each context has. For example, for the TripAdvisor dataset, there are five possible trip types that a user can take, namely 'solo', 'business', 'family', 'couples', and 'friends'. Note that also the context variables were one-hot encoded. Therefore, each dimension of a context variable represents a label that the MLC algorithm is required to predict.

Table 7.1: Description of the context-aware datasets: LDOS-CoMoDa and TripAdvisor. The values in brackets next to the context variables indicate the number of dimensions that each context has.

	LDOS-CoMoDa	**TripAdvisor**
Number of users	118	2371
Number of items	1205	7269
Number of ratings	2150	12175
Number of positives	1419	11264
Contexts	time (4), location (3), day (3) social (7)	trip type (5)
Contents	user gender, movie country, movie year, genre movie language	user country, user timezone, hotel city, hotel state, hotel country, hotel timezone

Following Zheng *et al.* (2014), we evaluate the performance of the respective MLC algorithms using five-fold cross-validation. The five sets of training observations, and the five sets of validation observations are obtained in the following manner. All observations corresponding to positive ratings are split into five folds of roughly equal size. Each of these positive folds are in turn used as a validation set. The remaining four positive folds, together with all observations corresponding to negative ratings, are in turn used as training set. Therefore note that the data observations corresponding to negative ratings form part of all five training sets.

In order to compare the results of the MLC algorithms, we make use of three baseline recommendation methods, *viz.* Global Popular (GP), Item Popular (IP) and User Popular (UP). These methods are based on popularity. The most popular contexts that occur throughout the training set are recommended in

the global popular recommendation. The contexts that occur most frequently for a specific item are recommended by the item popular baseline. Finally, the user popular baseline recommends the context that occurs the most frequently for a given user. Since the GP method does not provide personalised recommendations, it represents by far the simplest form of recommendation. The other two baselines, IP and UP, provide simple personalised recommendations to users.

Note that we evaluate and compare the performance of six MLC algorithms in our experiments. Under MLC adaptation algorithms we compare ML-kNN and BR-kNN, while CCs, LP and BR represent the class of MLC transformation algorithms. We also include RAkEL as MLC ensemble method. After transforming the data, of course any traditional classifier may be implemented by each of the MLC transformation algorithms. In this regard, note that we considered the use of Bayesian nets, decision trees and kNN classifiers.

The multi-label classification was preformed in Java, using the MLC library *Mulan* (version 1.5) (Tsoumakas *et al.*, 2011). For the classifiers used by the transformation MLC algorithms, we made use of Weka (version 3.7.10), which is a Java based platform containing machine learning algorithms (Hall *et al.*, 2009). The classifiers used from Weka are `Bayesnet` for Bayesian nets, `J48` for decision tress, and `Ibk` for kNN classifier. As in Zheng *et al.* (2014), we used unpruned trees. Cross-validation was used in order to determine the number of neighbours to use for the kNN classifiers.

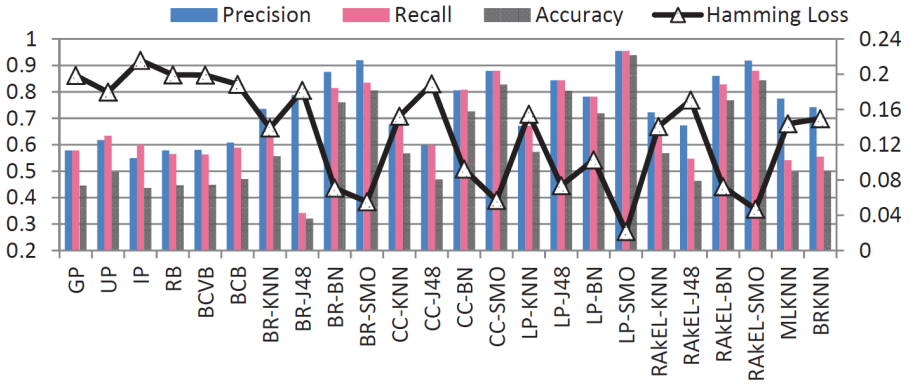
The format of the data used by Weka is an ARFF file. In this file format, the user may specify the type of data used. We found that for the LDOS-CoMoDa dataset, the MLC algorithms performed better when the data was treated as nominal, while the algorithms on the TripAdvisor dataset performed better when the data was treated as numeric.

The code relating to this chapter may be found in Appendix C.

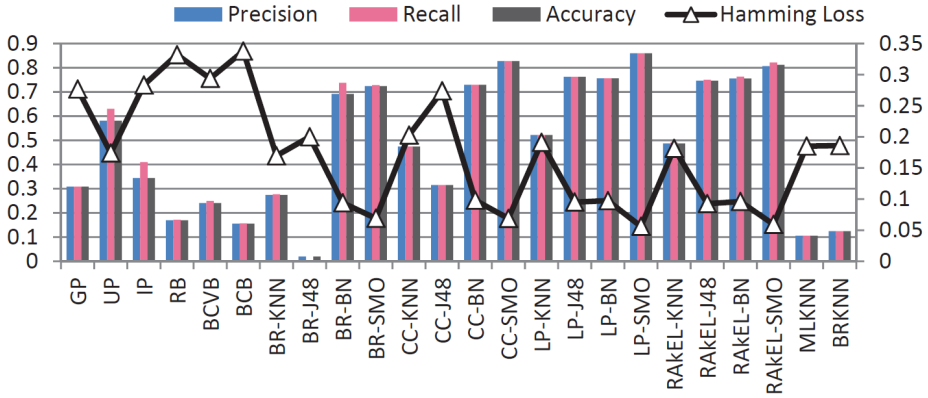
7.4 Results

Since we follow a similar experimental setup to the setup described in Zheng *et al.* (2014), we start by presenting the results from their study in Figure 7.1 below.

With respect to the results displayed in Figure 7.1, first note that we chose not to include the three baseline kNN algorithms (proposed by Baltrunas *et al.* (2010)) in our analysis. These baselines are called the Ratings-based (RB), Best Context Vector-based (BCVB), and Best Context-based (BCB) procedures in the study by Zheng *et al.* (2014). For computation reasons, the SVM classifier (`Ibk`) was also omitted. Note also that in Figure 7.1, and in our further discussion, the transformation algorithms will be denoted by 'Algorithm-Classifier'. Thus 'BR-kNN', for example, refers to the binary relevance algorithm used in combination with a kNN classifier.



(a) LDOS-CoMoDa



(b) TripAdvisor

Figure 7.1: Prediction performance of the MLC algorithms on LDOS-CoMoDa and TripAdvisor datasets. Source: Zheng *et al.* (2014).

Zheng *et al.* (2014) evaluate all baseline and MLC procedures using precision, recall, accuracy and Hamming Loss. These measures were obtained using five-fold cross validation and a very specific way of obtaining the five training- and test folds (described in the previous section). Note that the average precision, recall and accuracy values are summarised using bar plots, whereas the average Hamming Loss values are depicted using line graphs.

Considering the results in Figure 7.1, it is evident that for both datasets, the MLC algorithms generally outperform the baseline algorithms. Exceptions, however, occur in the case of BR-J48 (for both datasets), as well as in the case of BR-kNN and the two adaptation algorithms (for the TripAdvisor data). Zheng *et al.* (2014) note that GP would perform better than a personalised algorithm in cases where a user visited more than one hotel on a single trip, meaning that the trip type for those ratings would be the same.

Since the F-measure combines recall and precision, we chose to also report the F-measure in our empirical work. We proceed with a more detailed comparison between the results in Zheng *et al.* (2014), and the results from our empirical evaluation. This is done first for the set of baseline algorithms (in Section 7.4.1),

subsequently also for the standard MLC algorithms (in Section 7.4.2), and finally for the MLC regression approaches (in Section 7.4.3).

7.4.1 Baseline Algorithms

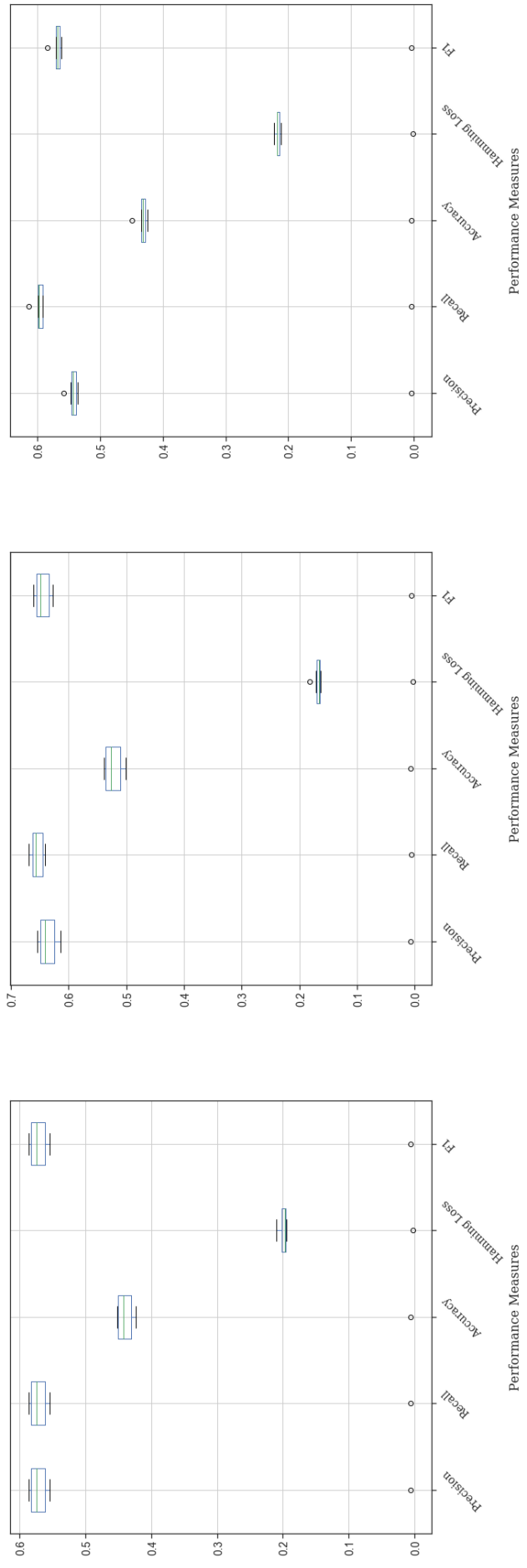
The performance measures of the baseline algorithms, as obtained in our empirical analysis, are summarised in terms of boxplots in Figures 7.2 and 7.3. The former figure displays the LDOS-CoMoDa results, whereas the latter figure is used to depict the TripAdvisor results. Note that the values of the performance measures on each fold were used to produce the boxplots. The narrowness of the boxes indicates that there is not much variation from one fold to the next with regards to the obtained metrics.

First consider the results for the LDOS-CoMoDa dataset in Figure 7.2. Clearly, GP and IP perform very similarly. For both, the average F-measure value is about 0.58, and the average Hamming Loss value is close to 0.20. In terms of average performance measures, we see that UP does slightly better, with an average F-measure of approximately 0.65, and with an average Hamming Loss value of about 0.18. Interestingly, in terms of the variance of performance measures, note that the IP approach yields measures with a much smaller variance than in the case of GP and UP.

Next consider the results for the TripAdvisor dataset in Figure 7.3. Here GP and IP perform much worse than UP. The average F-measure values for GP and IP are about 0.32 and 0.4, respectively, compared to about 0.6 for UP. In terms of Hamming Loss, differences between the baseline algorithms are far less pronounced: UP achieves about 0.18, compared to approximately 0.27, achieved by both GP and IP.

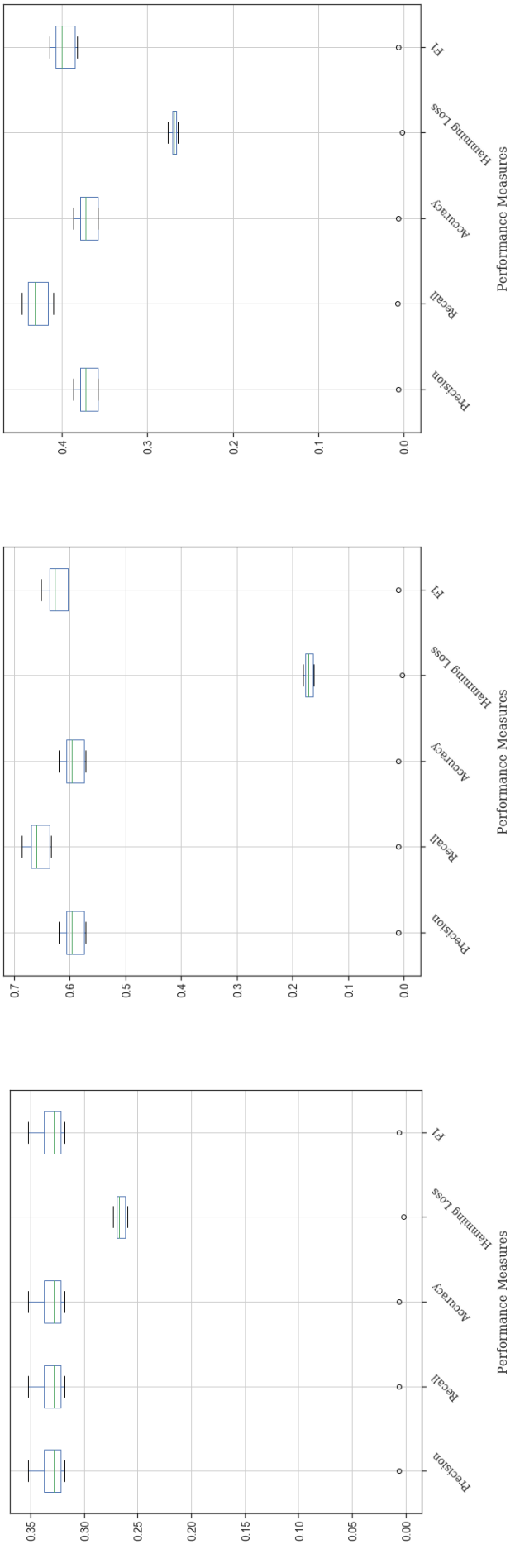
The baseline results reported in Zheng *et al.* (2014), and those obtained in our empirical analysis (*cf.* Figures 7.1 and 7.4) are found to be very similar. Zheng *et al.* (2014) also finds that UP outperforms both GP and IP. For the TripAdvisor dataset in particular, our baseline algorithms achieve slightly better results. For example, the average accuracies for GP, UP and IP in Zheng *et al.* (2014) are 0.3, 0.58, and 0.34. In our analysis, we obtained average accuracies of 0.33, 0.6 and 0.37.

In summary in this section, if one takes the simplicity of the baseline algorithms into account, their performance seems to be very reasonable. Also, since in both applications, UP is shown to outperform GP and IP, user personalisation seems to be an important aspect.



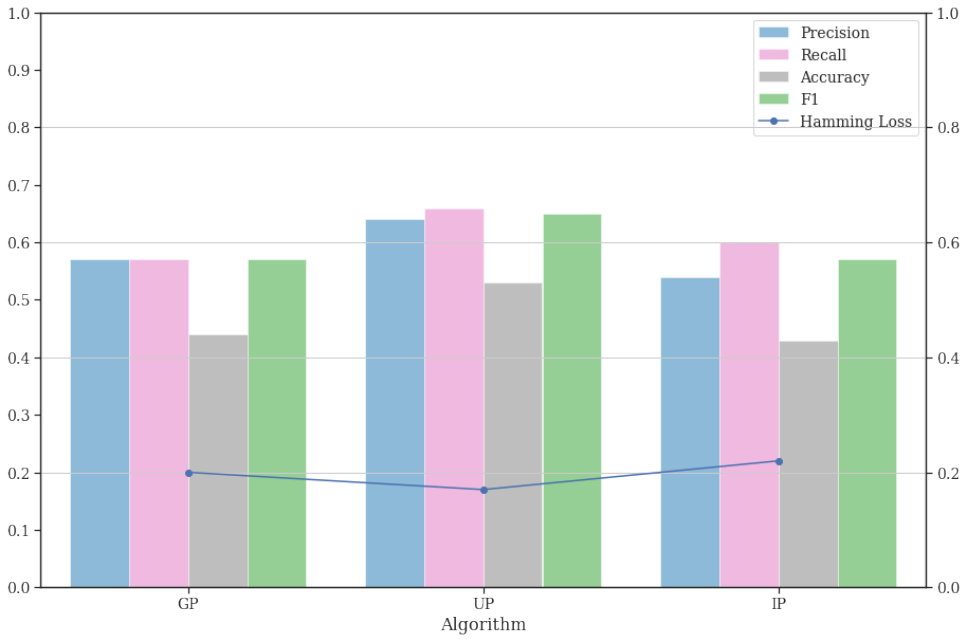
(a) GP baseline (b) UP baseline (c) IP baseline

Figure 7.2: LDOS-CoMoDa: Boxplot of performance measures for the different baseline recommenders.

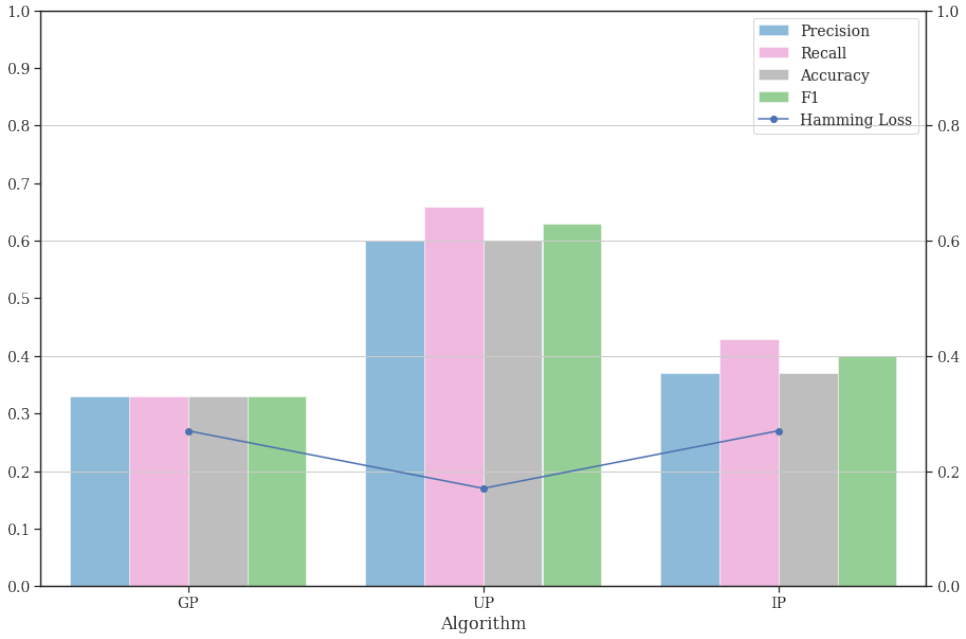


(a) GP baseline (b) UP baseline (c) IP baseline

Figure 7.3: TripAdvisor: Boxplot of performance measures for the different baseline recommenders.



(a) LDOS-CoMoDa



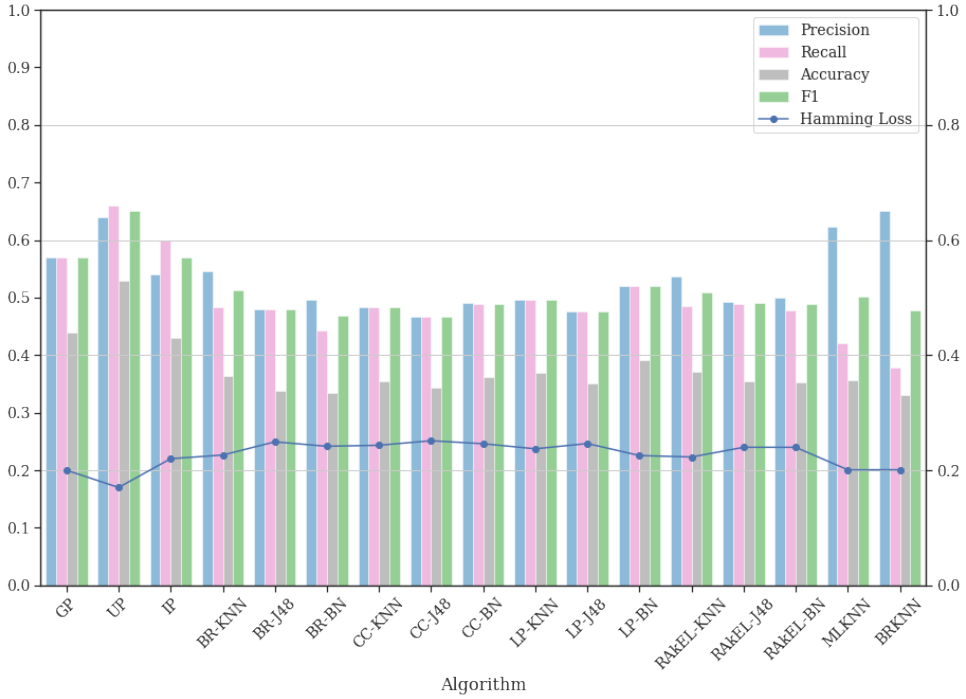
(b) TripAdvisor

Figure 7.4: Prediction performance of baseline algorithms. Precision, recall, accuracy and F1 values are read off the left axis, while Hamming loss is read off the right axis.

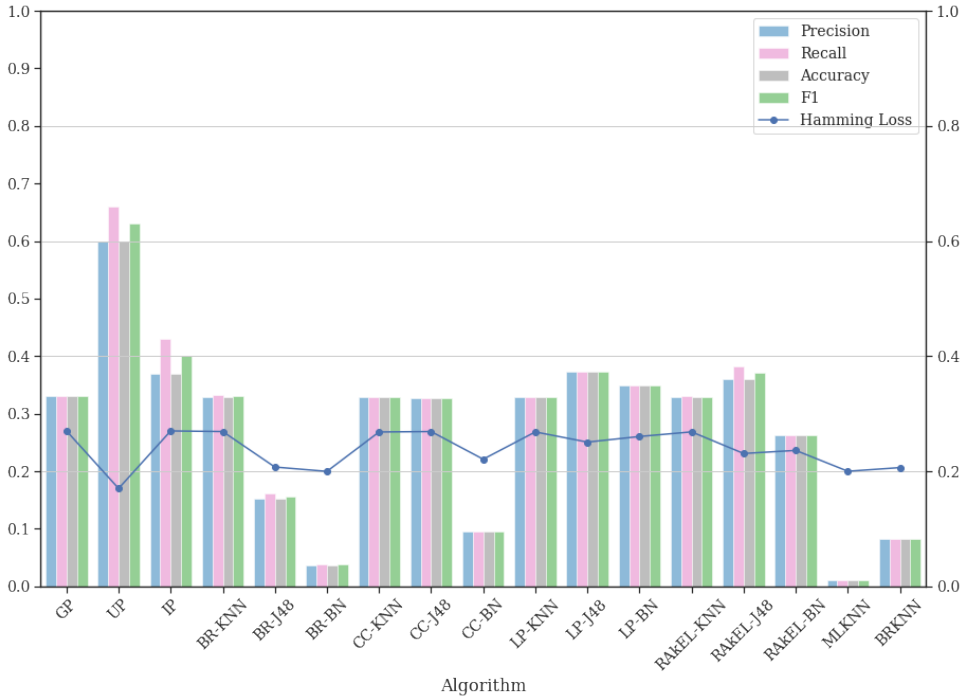
7.4.2 MLC Algorithms

The performance measures of the MLC algorithms, as obtained in our empirical analysis, are summarised in terms of bar plots in Figure 7.5. In the case of both datasets, the baseline UP method is found to outperform all MLC approaches.

For the LDOS-CoMoDa dataset, the MLC approaches perform very similarly. Only two adaption algorithms (ML-kNN and BR-kNN) are found to show some potential: we obtain an average Hamming



(a) LDOS-CoMoDa



(b) TripAdvisor

Figure 7.5: Prediction performance of all algorithms. Precision, recall, accuracy and F1 values are read off the left axis, while Hamming loss is read off the right axis.

Loss value of approximately 0.2 in the case of both these algorithms. Additionally, the average precision values are above 0.6, which is significantly higher than that obtained by the other algorithms. However, in terms of the other measures reported, these two approaches do not perform well.

Next consider the MLC results for the TripAdvisor dataset. Here we do observe some variation between

performances. In terms of Hamming Loss, BR-J48, BR-BN, CC-BN, and once again ML-kNN and BR-kNN seem to have the most potential. For these methods, the average Hamming Loss values are once again close to 0.2. However, as was the case for the LDOS-CoMoDa dataset, in terms of the other performance measures, these approaches do not perform well.

We observe significant differences between our MLC results, and the MLC results reported in Zheng *et al.* (2014) (*cf.* Figures 7.1 and 7.5). Across all performance measures and in the case of both datasets, our MLC results are generally 20% to 30% lower than the reported results. There is, however, an exception of this phenomenon that is perhaps worth mentioning. For the TripAdvisor data, in the case of the BR-J48, CC-J48 and BR-kNN algorithms, we achieve very similar values to all reported performance values.

Several potential reasons for inconsistencies between our results and the reported results may further be explored. Firstly, the format of the data, as well as the chosen variables, could be influential. It may well be that we have used different encoding techniques, or selected different variables to form part of the models considered. Another discrepancies might occur with hyper-parameter specifications. For example, the number of nearest neighbours used in the kNN classifiers may differ from the parameter value used in the study by Zheng *et al.* (2014).

7.4.3 Regression Algorithms

Recall from Section 5.5 that the regression approaches to MLC involve a method to specify label-specific threshold values. Therefore note that on each fold, we obtain the Hamming Loss, F-measure and accuracy measures for each combination of regression- and threshold methods. We number and label the threshold methods as follows: 1 indicates the quantile method, 2 indicates thresholding on the performance measure, and 3 indicates thresholding on the performance measure and subsequently converting the value to a quantile. Thus, 'CW-1' indicates the curds-and-whey regression approach used in combination with the quantile threshold method. Note that the other regression approaches are named similarly. The average results from the five-fold cross-validation procedure are presented in Tables 7.2 to 7.4. Note that all best performing methods are indicated in bold font.

Consider first the Hamming Loss values in Table 7.2. It is interesting to note that the results pertaining to the LDOS-CoMoDa data are far less variable than in the case of the TripAdvisor data. Overall, it is very difficult to identify an overall winner in terms of the regression approaches. Indeed, depending on the application, the best performing regression approach is found to differ. In the case of the LDOS-CoMoDa data, FICYREG performs the best, while in the case of the TripAdvisor data, RR outperforms the other approaches. We also see that exploiting label dependencies has a greater positive impact in the case of the TripAdvisor data than in the case of the LDOS-CoMoDa dataset. In terms of the performance of the various threshold methods, the second threshold method seems to be the best choice.

Table 7.2: Hamming Loss values for each regression algorithm.

Approach	LDOS-CoMoDa	TripAdvisor
CW-1	0.2380	0.3198
CW-2	0.2246	0.3292
CW-3	0.2259	0.3292
FICY-1	0.2335	0.3171
FICY-2	**0.2221**	0.3387
FICY-3	0.2224	0.3387
OLS-1	0.2352	0.3173
OLS-2	0.2244	0.3756
OLS-3	0.2253	0.3756
RR-1	0.2355	**0.3055**
RR-2	0.2260	0.3114
RR-3	0.2257	0.3114

Consider next the F-measure values in Table 7.3. For both datasets, these values are very similar. For the LDOS-CoMoDa data, FICYREG performs the best. For the TripAdvisor data, the performances of CW and RR are tied for first place. In terms of a choice between threshold methods, in the case of the LDOS-CoMoDa dataset, the first threshold method seems to be the best, while the second threshold method is the best choice in the case of the TripAdvisor dataset.

Table 7.3: F-measure values for each regression algorithm.

Approach	LDOS-CoMoDa	TripAdvisor
CW-1	0.5133	0.3139
CW-2	0.4663	**0.3553**
CW-3	0.4650	0.3503
FICY-1	**0.5201**	0.3213
FICY-2	0.4529	0.3547
FICY-3	0.4553	0.3494
OLS-1	0.5178	0.3431
OLS-2	0.4705	0.3440
OLS-3	0.4694	0.3464
RR-1	0.5173	0.3154
RR-2	0.4662	**0.3553**
RR-3	0.4649	0.3499

The average accuracy values are reported in Table 7.4. For the LDOS-CoMoDa data, we observe the same pattern as in the case of the Hamming Loss and F-measure, with FICYREG emerging as the best regression approach. As was the case with results based on the F-measure, the first threshold procedure is preferred in the case of all regression approaches. For the TripAdvisor dataset, the best accuracy across the regression approaches is achieved using the first threshold technique. Somewhat surprisingly, here OLS outperforms the other approaches.

A comparison between the performances of the regression approaches and the standard MLC algorithms is of particular interest to us. We base our comparison on all three performance measures considered, starting with the F-measure. Therefore, first consider Table 7.3 and Figure 7.5. For the LDOS-CoMoDa

Table 7.4: Accuracy values for each regression algorithm.

Approach	LDOS-CoMoDa	TripAdvisor
CW-1	0.3829	0.2743
CW-2	0.3229	0.2309
CW-3	0.3220	0.2266
FICY-1	**0.3912**	0.2814
FICY-2	0.3116	0.2300
FICY-3	0.3135	0.2256
OLS-1	0.3903	**0.3004**
OLS-2	0.3273	0.2204
OLS-3	0.3135	0.2256
RR-1	0.3895	0.2889
RR-2	0.3234	0.2315
RR-3	0.3135	0.2256

data, the regression approaches is found to perform similarly to the standard MLC algorithms. FICY-1 yields an F-measure of approximately 0.52, which is almost identical to the F-measure corresponding to the best performing MLC algorithm (LP-BN). This is also true in the for the TripAdvisor data: the average F-measure corresponding to the best regression approach (RR-2) is approximately 0.35, while the average F-measure corresponding to the best MLC algorithms (LP-J48 and RAkEL-J48) is approximately 0.37.

In terms of average Hamming Loss values, we reconsider Table 7.2 and Figure 7.5. In the case of the LDOS-CoMoDa data, once again the regression results are very similar to the MLC results. In the case of the TripAdvisor data, the regression results are slightly worse than the MLC results. The average Hamming Loss value is approximately 0.3 across all approaches, whereas the MLC algorithms produce an average Hamming Loss below 0.3.

In terms of average accuracy values, we compare Table 7.4 to Figure 7.5. For the LDOS-CoMoDa dataset, with exception of the first threshold method, the regression approaches perform consistently worse than the MLC algorithms. The best regression approach is found to be FICY-1, achieving an accuracy of 0.39, which is the same accuracy obtained by LP-BN. The regression approaches also perform significantly worse in the case of the TripAdvisor dataset. The best accuracy of 0.30 (OLS-1) is approximately 7% lower than the best performing MLC method (LP-J48).

In summary of this section, while the regression methods do not necessarily provide better results than the MLC methods, they are at least somewhat similar. The best performing regression approach on the LDOS-CoMoDa dataset across all performance measures is FICYREG. On the TripAdvisor dataset, the best Hamming Loss and F-score is achieved by RR, whereas OLS yields the best accuracy.

7.5 Summary

In this chapter, an empirical study was conducted in order to evaluate the predictive performance of MLC algorithms for context recommendation against simple baseline methods. We also considered various regression approaches to multi-label classification.

Compared to the standard MLC algorithms that we considered, the simple baseline methods perform much better than the more complex methods. In particular, the User Popular baseline achieves the best results on both datasets. This indicates that user personalistion is necessary in order to recommend the best contexts associated with an item. The fact that the simple baseline algorithms have a better predictive performance than the more complex algorithms, and that our results differ from that of Zheng *et al.* (2014) is reason to do additional future empirical studies on this topic. Alternative designs were also explored, however these did not yield improved results.

In our experiments, it was shown that the MLC regression approaches are unable to outperform the standard MLC methods. However, most of the results were found to be comparable, thereby not entirely discouraging an interest in the topic.

Chapter 8

Conclusion

The main objective of this thesis was to investigate various algorithms for recommender systems. We focused on two traditional types of recommender systems, as well as on recent improvements thereof. An overview of the algorithms that typically form part of recommender systems was provided. Towards explainable recommender systems, we considered the connection between multi-label classification and recommender systems.

We conducted two empirical studies. In the first study, our aim was to evaluate the performance of latent factor models and factorisation machines using varying levels of data sparsity. In the second study, in addition to standard MLC approaches, we also evaluated the use of extensions to multivariate linear regression in an MLC context. In the following two sections, we proceed with a summary of the thesis, and conclude with suggested avenues for further research.

8.1 Summary

Chapters 2 to 5 were devoted to a literature review of related RS algorithms. We started by describing the more standard approaches of content-based recommendation and collaborative filtering in Chapters 2 and 3, and proceeded with a discussion of more recent collaborative filtering approaches in Chapter 4. A different perspective to the design of an RS was explored in Chapter 5. This entails casting recommendation in terms of a multi-label classification problem. Chapters 6 and 7 were devoted to the two empirical studies that formed part of the research. We proceed with a more detailed description of each chapter below.

In Chapter 2 the focus was on content-based recommender systems. The design of these systems was described, which included using vector space models to represent items in a usable format, as well as using machine learning algorithms to learn user profiles. While these systems do not suffer from the cold-start problem, without well described items, their performance is generally poor. Therefore, collaborative filtering systems, discussed in Chapter 3, are typically preferred. The two types of CF algorithms, *viz.*

memory- and model-based, were discussed. Memory-based algorithms are simpler than model-based algorithms and can perform well on small datasets, however they do not scale well. Recommendations from both types of CF methods are severely impacted in sparse data settings. Since model-based CF methods scale reasonably well, extensions which aim to alleviate the impact of data sparsity on the recommendations were discussed in Chapter 4. These extensions included well-known latent factor models, such as matrix and tensor factorisation, as well as the more state-of-the-art factorisation machines. We concluded our literature review in Chapter 5, with an overview of recommendation by means of multi-label classification. More specifically, we considered the use of well-known MLC approaches, and extensions to multivariate linear regression in an MLC context, in order to recommend the context in which to use an item.

Our first empirical study was discussed in Chapter 6. The aim of this study was to investigate the impact of data sparsity on the predictive performance of recommender systems. The experimental design of the study was based upon the experimental frameworks considered in related work, although necessary modifications were made. We compared the performance of memory-based CF models, latent factor models and factorisation machines on two recommendation datasets, using generated datasets of varying levels of sparsity. In summary, the results showed that factorisation machines are able to provide improved recommendations at high levels of data sparsity. Furthermore, in the case of large datasets, factorisation machines are faster to implement than the simpler methods.

Chapter 7 was devoted to our second empirical study. The purpose of this study was to explore the use of multi-label classification algorithms in recommender problems. Since related work focused on the use of MLC algorithms in context recommendations, we also attempted to make use of MLC methods in order to recommend the contexts in which users are more likely to appreciate items of interest. With a view to a contribution in terms of explainable recommender systems, we extended our study to include the more recently proposed MLC regression approaches. Our results indicated that an MLC approach is suboptimal in an RS setting, however it is important to note that our results differ from those of Zheng *et al.* (2014), therefore further investigation is necessary.

8.2 Future Research

Several options for further research in the recommender systems field are worth mentioning. Since Zheng *et al.* (2014) report promising results for the use of MLC methods in an RS setting, there is some evidence that improvement of the performance of MLC methods in this context is within reach. Several pre-processing steps (such as using sampling to correct class imbalances, and considering different data formats) may be worth investigating.

Recommender systems that produce explainable output can improve user satisfaction as well as trust

in the system. Recommendations produced by simple memory-based CF models are easy to explain, however they typically have a low accuracy. On the other hand, latent factor models produce highly accurate results, but these are difficult to interpret and explain. Therefore, it is desirable to have a recommender system that is not only accurate, but explainable. A recent review of the currently used methods towards explainable recommendation may be found in the survey by Zhang and Chen (2019). If the regression approaches to MLC can be shown to perform as well as the MLC methods investigated in Zheng *et al.* (2014), we believe that there is a contribution to be made in this research field.

In our discussion in Chapter 4, we have touched upon the use of factorisation machines in the hidden layers of neural networks. Studies investigating the use of neural networks towards improving the predictive performance of recommender systems can be found in the literature. For example, Covington *et al.* (2016) make use of deep learning to improve YouTube recommendations, and van den Oord *et al.* (2013) investigate the use of convolutional neural networks for music recommendation. A review of the use of deep learning for recommender systems may be found in the paper by Batmaz *et al.* (2018). From this paper it is clear that deep learning for recommender systems is a specialised field, and one which seems to be a natural direction for further study.

Appendices

Appendix A

Source Code: Chapter 6

A.1 Required Python Packages

```
import pandas as pd
import os
import matplotlib.pyplot as plt
from scipy import sparse
import numpy as np
import random
import time
from pprint import pprint
import xlearn as xl

from surprise import NMF, SVD, KNNBasic
from surprise import prediction_algorithms
from surprise import Reader, Dataset
from surprise.model_selection import train_test_split
from surprise.model_selection import GridSearchCV
from surprise import accuracy
from surprise.model_selection import KFold
```

A.2 Functions Needed To Generate A Sparse Matrix

```
def calculate_dataframe_sparsity(ratings_df):
    n_users = len(ratings_df['userId'].unique())
    n_movies = len(ratings_df['movieId'].unique())
    n_ratings = len(ratings_df)
    sparsity = 100.0 - (100.0 * n_ratings / (n_users * n_movies))

    return("users: ", n_users,
           "movies: ", n_movies,
           "ratings: ", n_ratings,
```

```
          "sparsity: ", sparsity)

def sparsity_of_an_array(a):
    n_elements = np.prod(a.shape)
    return((n_elements - np.count_nonzero(a))/n_elements)

def label_encode_ids(ratings_df):

    from sklearn import preprocessing
    le = preprocessing.LabelEncoder()

    # encode userids
    le.fit(ratings_df['userId'].unique())
    ratings_df['userId'] = le.transform(ratings_df['userId'])

    # encode movieids
    le.fit(ratings_df['movieId'].unique())
    ratings_df['movieId'] = le.transform(ratings_df['movieId'])

    return(ratings_df)

def user_and_item_proportions(df):
    '''
    Parameters:
        - df: dataframe with columns "userId", "movieId", "rating".
    Returns:
        Dataframe specifying user proportions and a dataframe specifiying item
    proportions.
    '''
    import numpy as np

    unique_userid = np.array(df['userId'].unique())
    unique_itemid = np.array(df['movieId'].unique())
    total_number_of_ratings = len(df)

    # number of ratings per user
    n_user_ratings = df.drop(columns=["movieId"])
    n_user_ratings = n_user_ratings.groupby(["userId"]).count().reset_index()
    n_user_ratings = n_user_ratings.rename(columns={"rating": "n_ratings"})

    # proportion of ratings for each user
    n_user_ratings['user_prop'] = n_user_ratings['n_ratings']/total_number_of_ratings

    # number of ratings per item
```

```
    n_item_ratings = df.drop(columns=["userId"])
    n_item_ratings = n_item_ratings.groupby(["movieId"]).count().reset_index()
    n_item_ratings = n_item_ratings.rename(columns={"rating": "n_ratings"})

    # proportion of ratings for each item
    n_item_ratings['item_prop'] = n_item_ratings['n_ratings']/total_number_of_ratings

    return(n_user_ratings, n_item_ratings)

def create_dense_matrix(df, mf_technique = 'SVD', k = 100, bias = 'False', scale = (0.5, 5)):
    '''
    Paramters:
        - df: dataframe with columns "userId", "movieId", "rating".
        - mf_technique: matrix factorisation technique to be used, either "SVD" or "NMF".
        - k: number of latent factors.
        - bias: whether to use baselines (boolean).
        - scale: min and max rating values.
    Returns:
        Dense matrix, user latent vector, item latent vector.
    '''

    from surprise import NMF, SVD
    from surprise import prediction_algorithms
    from surprise import Reader, Dataset
    import numpy as np

    reader = Reader(rating_scale = scale)
    data = Dataset.load_from_df(df, reader)
    temp = data.build_full_trainset()

    if mf_technique == "SVD":
        alg = SVD(n_factors = k, biased = bias, random_state = 1)
    else:
        alg = NMF(n_factors = k, biased = bias, random_state = 1)

    alg.fit(temp)
    pu = alg.pu
    qi = alg.qi
    R_hat = np.dot(pu, qi.T)

    return(R_hat, pu, qi)

def number_of_ratings_needed_for_a_level_of_spasrsity(total_number_of_ratings,
```

```
    sparsity_level):
    return(int(total_number_of_ratings*(1-sparsity_level)))

def create_sparse_matrix(mat, user_prop, item_prop, sparsity_level):
    import random
    import pandas as pd
    import numpy as np

    np.random.seed(2)
    i, u = mat.shape
    x = number_of_ratings_needed_for_a_level_of_spasrsity(u*i, sparsity_level)

    unique_userid = np.array(user_prop['userId'].unique())
    unique_itemid = np.array(item_prop['movieId'].unique())

    random.shuffle(unique_userid)

    sparse_df = pd.DataFrame(columns = ['userId', 'movieId', 'rating'])

    for j in range(0, x):
        selected_user = int(np.random.choice(unique_userid, 1, p = user_prop['user_prop
    '], replace = True))
        selected_item = int(np.random.choice(unique_itemid, 1, p = item_prop['item_prop
    '], replace = True))
        selected_rating = mat[selected_user, selected_item]
        sparse_df = sparse_df.append({'userId': selected_user,
                                      'movieId': selected_item, 'rating': selected_rating
    }, ignore_index =True)


    convert_dict = {'userId': int, 'movieId':int, 'rating': float}
    sparse_df = sparse_df.astype(convert_dict)
    return(sparse_df)
```

A.3 Functions Needed To Run Collaborative Filtering Algorithms

```
def get_data(file_name):
    import pandas as pd
    from surprise import Reader, Dataset
    import numpy as np
    import random
    import csv

    R_100_s = pd.DataFrame(np.loadtxt(file_name, dtype=float), columns = ['userId', '
    itemId', 'rating'])
```

```
    R_100_s = R_100_s.astype({'userId': int,'itemId':int, 'rating':float})

    # Load the full dataset.
    reader = Reader(rating_scale=(min(R_100_s['rating']), max(R_100_s['rating'])))
    data = Dataset.load_from_df(R_100_s[['userId', 'itemId', 'rating']], reader)
    raw_ratings = data.raw_ratings

    # shuffle ratings if you want
    random.shuffle(raw_ratings)

    with open(file_name + '_raw','w') as out:
        csv_out=csv.writer(out)
        for row in raw_ratings:
            csv_out.writerow(row)

    # A = 90% of the data, B = 10% of the data
    threshold = int(.9 * len(raw_ratings))
    print(threshold)
    A_raw_ratings = raw_ratings[:threshold]
    B_raw_ratings = raw_ratings[threshold:]

    data.raw_ratings = A_raw_ratings  # data is now the set A
    trainset = data.build_full_trainset()
    testset = data.construct_testset(B_raw_ratings)  # testset is now the set B

    return(data, trainset, testset, raw_ratings, threshold)

######### BASIC METHODS

def get_baseline_only_model(gs_data, train_data):
    from surprise.model_selection import GridSearchCV
    from surprise import accuracy
    from surprise import BaselineOnly
    from surprise import prediction_algorithms

    bsl_options = {'method': ['sgd'], 'learning_rate': [0.007, 0.009, 0.01], 'n_epochs':
    [5,10]}
    param_grid = {'bsl_options': bsl_options}

    baseline_gs = GridSearchCV(BaselineOnly, param_grid, measures=['rmse'], cv=3, n_jobs
    =5)
    baseline_gs.fit(gs_data)

    best_params_baselineonly = baseline_gs.best_params
```

```
    print(best_params_baselineonly)
    baselineonly = baseline_gs.best_estimator['rmse']

    # retrain on the whole trainset
    model_BaselineOnly = baselineonly.fit(train_data)

    return(model_BaselineOnly)

########## KNN METHODS

def get_best_CF_userbased_model(gs_data, train_data):
    from surprise.model_selection import GridSearchCV
    from surprise import accuracy
    from surprise import KNNBasic
    from surprise import prediction_algorithms

    sim_options = {'name': ['cosine'],
                   'user_based': [True]
                  }
    param_grid = {'k': [5, 15, 25, 50], 'sim_options': sim_options}

    knnbasic_gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse'], cv=3, n_jobs=5)
    knnbasic_gs.fit(gs_data)

    best_params_CF_user = knnbasic_gs.best_params
    print(best_params_CF_user)
    knn_users = knnbasic_gs.best_estimator['rmse']

    # retrain on the whole trainset
    model_CF_userbased = knn_users.fit(train_data)

    return(model_CF_userbased)

def get_best_CF_itembased_model(gs_data, train_data):
    from surprise.model_selection import GridSearchCV
    from surprise import accuracy
    from surprise import KNNBasic
    from surprise import prediction_algorithms

    sim_options = {'name': ['cosine'],
                   'user_based': [False]
                  }
    param_grid = {'k': [5, 15, 25, 50], 'sim_options': sim_options}

```

```
95      knnbasic_gs = GridSearchCV(KNNBasic, param_grid, measures=['rmse'], cv=3, n_jobs=5)
96      knnbasic_gs.fit(gs_data)
97
98      best_params_CF_item = knnbasic_gs.best_params
99      print(best_params_CF_item)
100     knn_item = knnbasic_gs.best_estimator['rmse']
101
102     # retrain on the whole trainset
103     model_CF_itembased = knn_item.fit(train_data)
104
105     return(model_CF_itembased)
106
107 ########## MATRIX FACTORISATION METHODS
108
109 def get_best_svd_model(gs_data, train_data):
110     '''
111     Does a grid search to determining the best parameters (based on rmse), and returns
        the model fitted on training data.
112
113     Parameters:
114         - data: gridseach data
115         - train: data to train model
116     Returns:
117         - model: trained model
118     '''
119
120     from surprise.model_selection import GridSearchCV
121     from surprise import accuracy
122     from surprise import SVD
123     from surprise import prediction_algorithms
124
125     param_grid = {'n_factors': [20, 50, 100], 'n_epochs': [5, 10], 'lr_all': [0.007,
        0.009, 0.01],
126                   'reg_all':[0.02, 0.04, 0.06]}
127
128     grid_search = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=3)
129     grid_search.fit(gs_data)
130
131     best_params_svd = grid_search.best_params
132     print(best_params_svd)
133     algo_svd = grid_search.best_estimator['rmse']
134
135     # retrain on the whole trainset
136     model_svd = algo_svd.fit(train_data)
```

```
137
138     return(model_svd)
139
140 def get_best_svd_unbiased_model(gs_data, train_data):
141     '''
142     Does a grid search to determining the best parameters (based on rmse), and returns
        the model fitted on training data.
143
144     Parameters:
145         - data: gridseach data
146         - train: data to train model
147     Returns:
148         - model: trained model
149     '''
150
151     from surprise.model_selection import GridSearchCV
152     from surprise import accuracy
153     from surprise import SVD
154     from surprise import prediction_algorithms
155
156     param_grid = {'n_factors': [20, 50, 100], 'n_epochs': [5, 10], 'lr_all': [0.007,
        0.009, 0.01],
157                   'reg_all':[0.02, 0.04, 0.06], 'biased': [False]}
158
159     grid_search = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=3)
160     grid_search.fit(gs_data)
161
162     best_params_svd = grid_search.best_params
163     print(best_params_svd)
164     algo_svd = grid_search.best_estimator['rmse']
165
166     # retrain on the whole trainset
167     model_svd = algo_svd.fit(train_data)
168
169     return(model_svd)
170
171 def get_best_svdpp_model(gs_data, train_data):
172     '''
173     Does a grid search to determining the best parameters (based on rmse), and returns
        the model fitted on training data.
174
175     Parameters:
176         - data: gridseach data
177         - train: data to train model
```

```
178     Returns:
179         - model: trained model
180     '''
181
182     from surprise.model_selection import GridSearchCV
183     from surprise import accuracy
184     from surprise import SVDpp
185     from surprise import prediction_algorithms
186
187     param_grid = {'n_factors': [20, 50], 'n_epochs': [5, 10], 'lr_all': [0.007, 0.009,
        0.01],
188                   'reg_all':[0.04, 0.06]}
189
190     grid_search = GridSearchCV(SVDpp, param_grid, measures=['rmse'], cv=3)
191     grid_search.fit(gs_data)
192
193     best_params_svdpp = grid_search.best_params
194     print(best_params_svdpp)
195     algo_svdpp = grid_search.best_estimator['rmse']
196
197     # retrain on the whole trainset
198     model_svdpp = algo_svdpp.fit(train_data)
199
200     return(model_svdpp)
201
202 def get_best_nmf_model(gs_data, train_data):
203     '''
204     Does a grid search to determining the best parameters (based on rmse), and returns
        the model fitted on training data.
205
206     Parameters:
207         - data: gridseach data
208         - train: data to train model
209     Returns:
210         - model: trained model
211     '''
212
213     from surprise.model_selection import GridSearchCV
214     from surprise import accuracy
215     from surprise import NMF
216     from surprise import prediction_algorithms
217     param_grid = {'n_factors': [20, 50, 100], 'n_epochs': [5, 10], 'reg_qi':[0.02, 0.04,
        0.06],
218                   'reg_pu':[0.02, 0.04, 0.06]}
```

```
219
220     grid_search = GridSearchCV(NMF, param_grid, measures=['rmse'], cv=3)
221     grid_search.fit(gs_data)
222
223     best_params_nmf = grid_search.best_params
224     print(best_params_nmf)
225     algo_nmf = grid_search.best_estimator['rmse']
226
227     # retrain on the whole trainset
228     model_nmf = algo_nmf.fit(train_data)
229
230     return(model_nmf)
231
232 ################ PREDICTION MODELS
233
234 def baseline_predict(model, train, test):
235     from surprise import accuracy
236     from surprise import prediction_algorithms
237
238     # Compute biased accuracy on train
239     predictions_bias = model.test(train.build_testset())
240     bias_rmse = accuracy.rmse(predictions_bias)
241     bias_mae = accuracy.mae(predictions_bias)
242
243     # Compute unbiased accuracy on test
244     predictions_unbias = model.test(test)
245     unbias_rmse = accuracy.rmse(predictions_unbias)
246     unbias_mae = accuracy.mae(predictions_unbias)
247
248     return(bias_rmse, bias_mae, unbias_rmse, unbias_mae)
249
250 def cf_users_predict(model, train, test):
251     from surprise import accuracy
252     from surprise import prediction_algorithms
253     from surprise import Reader, Dataset
254
255     # Compute biased accuracy on train
256     predictions_bias_knn_users = model.test(train.build_testset())
257     bias_rmse_knn_users = accuracy.rmse(predictions_bias_knn_users)
258     bias_mae_knn_users = accuracy.mae(predictions_bias_knn_users)
259
260     # Compute unbiased accuracy on test
261     predictions_unbias_knn_users = model.test(test)
262     unbias_rmse_knn_users = accuracy.rmse(predictions_unbias_knn_users)
```

```
263     unbias_mae_knn_users = accuracy.mae(predictions_unbias_knn_users)
264
265     return(bias_rmse_knn_users, bias_mae_knn_users, unbias_rmse_knn_users,
        unbias_mae_knn_users)
266
267 def cf_item_predict(model, train, test):
268     from surprise import accuracy
269     from surprise import prediction_algorithms
270
271     # Compute biased accuracy on train
272     predictions_bias_knn_item = model.test(train.build_testset())
273     bias_rmse_knn_item = accuracy.rmse(predictions_bias_knn_item)
274     bias_mae_knn_item = accuracy.mae(predictions_bias_knn_item)
275
276     # Compute unbiased accuracy on test
277     predictions_unbias_knn_item = model.test(test)
278     unbias_rmse_knn_item = accuracy.rmse(predictions_unbias_knn_item)
279     unbias_mae_knn_item = accuracy.mae(predictions_unbias_knn_item)
280
281     return(bias_rmse_knn_item, bias_mae_knn_item, unbias_rmse_knn_item,
        unbias_mae_knn_item)
282
283 def svd_predict(model, train, test):
284     from surprise import accuracy
285     from surprise import prediction_algorithms
286
287     # Compute biased accuracy on train
288     predictions_bias_svd = model.test(train.build_testset())
289     biased_rmse_svd = accuracy.rmse(predictions_bias_svd)
290     biased_mae_svd = accuracy.mae(predictions_bias_svd)
291
292     # Compute unbiased accuracy on test
293     predictions_unbias_svd = model.test(test)
294     unbiased_rmse_svd = accuracy.rmse(predictions_unbias_svd)
295     unbiased_mae_svd = accuracy.mae(predictions_unbias_svd)
296
297     return(biased_rmse_svd, biased_mae_svd, unbiased_rmse_svd, unbiased_mae_svd)
298
299 def svd_unbiased_predict(model, train, test):
300     from surprise import accuracy
301     from surprise import prediction_algorithms
302
303     # Compute biased accuracy on train
304     predictions_bias_svd = model.test(train.build_testset())
```

```
305     biased_rmse_svd = accuracy.rmse(predictions_bias_svd)
306     biased_mae_svd = accuracy.mae(predictions_bias_svd)
307 
308     # Compute unbiased accuracy on test
309     predictions_unbias_svd = model.test(test)
310     unbiased_rmse_svd = accuracy.rmse(predictions_unbias_svd)
311     unbiased_mae_svd = accuracy.mae(predictions_unbias_svd)
312 
313     return(biased_rmse_svd, biased_mae_svd, unbiased_rmse_svd, unbiased_mae_svd)
314 
315 def svpp_predict(model, train, test):
316     from surprise import accuracy
317     from surprise import prediction_algorithms
318 
319     # Compute biased accuracy on train
320     predictions_bias_svdpp = model.test(train.build_testset())
321     biased_rmse_svdpp = accuracy.rmse(predictions_bias_svdpp)
322     biased_mae_svdpp = accuracy.mae(predictions_bias_svdpp)
323 
324     # Compute unbiased accuracy on test
325     predictions_unbias_svdpp = model.test(test)
326     unbiased_rmse_svdpp = accuracy.rmse(predictions_unbias_svdpp)
327     unbiased_mae_svdpp = accuracy.mae(predictions_unbias_svdpp)
328 
329     return(biased_rmse_svdpp, biased_mae_svdpp, unbiased_rmse_svdpp, unbiased_mae_svdpp)
330 
331 def nmf_predict(model, train, test):
332     from surprise import accuracy
333     from surprise import prediction_algorithms
334 
335     # Compute biased accuracy on train
336     predictions_biased_nmf = model.test(train.build_testset())
337     biased_rmse_nmf = accuracy.rmse(predictions_biased_nmf)
338     biased_mae_nmf = accuracy.mae(predictions_biased_nmf)
339 
340     # Compute unbiased accuracy on test
341     predictions_unbias_nmf = model.test(test)
342     unbias_rmse_nmf = accuracy.rmse(predictions_unbias_nmf)
343     unbias_mae_nmf = accuracy.mae(predictions_unbias_nmf)
344 
345     return(biased_rmse_nmf, biased_mae_nmf, unbias_rmse_nmf, unbias_mae_nmf)
```

A.4 Functions Needed To Run Factorisation Machine Algorithms

The function `convert_ratings_to_fm` is modified from Punia (2018).

```
def convert_ratings_to_fm(fin,fout,feature_index,rating_index,_model = "fm"):
    '''
    Input : ratings file with columns in the following order
            1) user_id
            2) movie_id
            3) rating
            4) timestamp - Ignoring this column for now

    Arguments : fin : input ratings file
                fout : output file name - column indices to be included
                column index containing the rating
                _model : ffm/fm

    Output:
            ratings matrix transformed to libsvm

    '''
    rat_file = open(fin,'r')  #input file
    text_file = open(fout,'w') #output file

    if _model=="ffm":
        add_field = 1
    else:
        add_field = 0

    #Initialize ::
    lines=rat_file.readlines()
    val0 = lines[0]
    split_row0 = val0.split(',')
    datastring = ""  #stores the final string
    indx_cntr = 0
    d_field = {}

    ###User
    d_field[feature_index[0]] = {split_row0[0] : indx_cntr}
    indx_cntr = indx_cntr + 1
    ###movie
    d_field[feature_index[1]] = {split_row0[1] : indx_cntr}
    indx_cntr = indx_cntr + 1
    ###first string
    ###rating
```

```
42      datastring += split_row0[rating_index]
43      ###user
44      datastring += "," + ("0" + ":") * add_field + str(d_field[0][split_row0[0]]) + ":" +
        "1"
45      ###movie
46      datastring += "," + ("1" + ":") * add_field + str(d_field[1][split_row0[1]]) + ":" +
        "1"
47
48      datastring += "\n"
49      text_file.write(datastring)
50
51      #iterate over all the lines
52      for val in lines[1:len(lines)]:
53
54      #split each row
55          split_row = val.split(',')
56          #rating
57          datastring = split_row[rating_index].replace('\n', '')
58          for col in feature_index: #ignoring timestamp, rating
59
60              #if a new user/movie found, add it to dictionary
61              if d_field[col].get(split_row[col],None) == None:
62                  d_field[col][split_row[int(col)]] = indx_cntr
63                  indx_cntr += 1
64
65              datastring += "," + (str(col) + ":") * add_field + str(d_field[col][split_row
    [col]]) + ":" + "1"
66          datastring += "\n"
67          text_file.write(datastring)
68
69      text_file.close()
70
71  def train_test_split_for_fm(file_name):
72      # splits the sparse matrix into a train and test set. 90/10 split.
73      # set seed before running this to ensure threshold is the same
74
75      import csv
76
77      all_data = open(file_name,'r')
78      lines = all_data.readlines()
79
80      threshold = int(.9 * len(lines))
81      print(threshold)
82
```

```
    train = lines[:threshold]
    test = lines[threshold:]

    with open(file_name + '_train','w') as out:
        csv_out=csv.writer(out)
        for row in train:
            r = row.replace('\n', '')
            r = r.split(',')
            csv_out.writerow(r)

    with open(file_name + '_test','w') as out:
        csv_out=csv.writer(out)
        for row in test:
            r = row.replace('\n', '')
            r = r.split(',')
            csv_out.writerow(r)

def create_cv_train_test_splits_for_fm(file_name):
    # splits the training data into 3 folds. Each of the three folds is spilt into a
    # train and test set. These training and testing sets are saved.
    # set seed before running this to ensure threshold is the same

    import csv

    all_data = open(file_name,'r')
    lines = all_data.readlines()

    threshold = int(len(lines)/3)

    fold1 = lines[:threshold]
    fold2 = lines[threshold:threshold*2]
    fold3 = lines[threshold*2:]

    split_fold_into_train_test(fold1, "fold_1")
    split_fold_into_train_test(fold2, "fold_2")
    split_fold_into_train_test(fold3, "fold_3")

def split_fold_into_train_test(fold, file_name):
    # splits a fold into train and test set. 90/10 split.

    import csv

    threshold = int(.9 * len(fold))

```

```
127     train = fold[:threshold]
128     test = fold[threshold:]
129 
130     with open(file_name + '_train_cv','w') as out:
131         csv_out=csv.writer(out)
132         for row in train:
133             r = row.replace('\n', '')
134             r = r.split(',')
135             csv_out.writerow(r)
136 
137     with open(file_name + '_test_cv','w') as out:
138         csv_out=csv.writer(out)
139         for row in test:
140             r = row.replace('\n', '')
141             r = r.split(',')
142             csv_out.writerow(r)
143 
144 def cv_for_FM(fold_dict, params, is_fm = True):
145     '''
146     Performs 3 fold cross-validation .
147     Calculates the rmse on each fold, and average rmse across the folds
148 
149     Parameters
150     ----------
151         fold_dict: dictionary specifiying the file paths for the training and testing
    folds
152         params: parameters to be used for the model
153         is_fm: bool, FM or FFM
154 
155     Returns
156     -------
157     Average rmse of the three folds
158     '''
159 
160     import xlearn as xl
161 
162     aggr_rmse = 0
163     for fold in fold_dict:
164         if is_fm:
165             model = xl.create_fm()
166         else:
167             model = xl.create_ffm()
168 
169         model.setTrain(fold_dict[fold][0]) #  ('train1_file_name', "test1_file_name")
```

```
170         model.setTest(fold_dict[fold][1]) #  ('train1_file_name', "test1_file_name")
171         model.fit(params, "./save_model_path" + fold + ".out")
172         model.predict("./save_model_path" + fold + ".out", "./output_" +fold+".txt")
173 
174         rmse,_ = calculate_metrics(fold_dict[fold][1], "./output_" +fold+".txt")
175         aggr_rmse += rmse
176 
177         # aggr_mae += mae
178         # update aggregate
179     avg_rmse = aggr_rmse/len(fold_dict)
180 
181     return(avg_rmse)
182 
183 def GridSearchFM(train_filename, l_rates, regs, n_factors, fm=True):
184     '''
185     Performs a grid search across sepcified parameters by means of 3-fold cross-
        validation.
186     The parameters resulting in the lowest rmse are returned.
187 
188     Paramters
189     ----------
190         train_filename: string specifiying the train file path
191         l_rates: array, learning rates
192         regs: array, regularisation
193         n_factors: array, number of latent vectors
194     Returns
195         best_rmse: lowest obtained average rmse
196         best_params: paramters that resulted in the lowest rmse
197     '''
198 
199     import csv
200     import xlearn as xl
201     import numpy as np
202 
203     # splits the data into 3 folds, then splits each fold into a train and test set
204     create_cv_train_test_splits_for_fm(train_filename)
205 
206     # fold_dict: dictionary containing folds to use for cv
207     fold_dict = {'fold1': ('fold_1_train_cv', "fold_1_test_cv"),
208                  'fold2': ('fold_2_train_cv', "fold_2_test_cv"),
209                  'fold3': ('fold_3_train_cv', "fold_3_test_cv")}
210 
211     best_params = [0,0,0]
212     best_rmse = np.inf
```

```
213
214     # Grid Search
215     for lr in l_rates:
216         for reg in regs:
217             for k in  n_factors:
218                 print([lr, reg, k])
219                 param = {'task':'reg',
220                       'lr':lr,
221                       'lambda':reg,
222                       'metric':'rmse',
223                       'fold':3,
224                       'opt':'sgd',
225                       'k': k}
226
227                 # Cross validation
228                 avg_rmse = cv_for_FM(fold_dict, param, is_fm = fm)
229
230                 if avg_rmse < best_rmse:
231                     best_rmse = avg_rmse
232                     best_params = [lr,reg,k]
233     return(best_rmse, best_params)
234
235 def FM_predict(train_file_name, test_file_name, params, fm = True):
236     '''
237     Fits model to the training data. Returns the biased and unbiased rmse and mae
238
239     Parameters
240     ----------
241         train_file_name: string, train file path
242         test_file_name: sting, test file path,
243         params: dictionary, specify paramters to use
244         fm: bool, FM or FFM
245
246     Returns
247     -------
248         biased_rmse
249         biased_mae
250         unbiased_rmse
251         unbiased_mae
252
253     '''
254     import xlearn as xl
255
256     if fm:
```

```
257         model = xl.create_fm()
258     else:
259         model = xl.create_ffm()
260 
261     model.setTrain(train_file_name)
262     model.fit(params, "./model.out")
263 
264     model.setTest(train_file_name)
265     model.predict("./model.out", "./output_biased.txt")
266     biased_rmse, biased_mae = calculate_metrics(train_file_name, "./output_biased.txt")
267 
268     model.setTest(test_file_name)
269     model.predict("./model.out", "./output_unbiased.txt")
270     unbiased_rmse, unbiased_mae = calculate_metrics(test_file_name, "./output_unbiased.
        txt")
271 
272     return(biased_rmse, biased_mae, unbiased_rmse, unbiased_mae)
273 
274 def calculate_metrics(test_fname, predict_fname):
275     '''
276         Reads predictions and test data from file and calculates
277         RMSE and MAE.
278 
279         Parameters
280         ----------
281         test_fname - String
282             Test data file name
283         predict_fname - String
284             Predictions file name
285 
286         Returns
287         ----------
288         RMSE - float
289             Root mean squared error
290         MAE - float
291             Mean Absolute Error
292     '''
293     from sklearn.metrics import mean_squared_error, mean_absolute_error
294     import math
295     import csv
296     import numpy as np
297 
298     true = open(test_fname,'r')
299     lines_true = true.readlines()
```

```
300
301     y_true = np.zeros(len(lines_true))
302
303     for i in range(0, len(lines_true)):
304         ob = lines_true[i]
305         ob = ob.split(',')[0]
306         ob = float(ob)
307         y_true[i] = float(ob)
308
309     pred = open(predict_fname,'r')
310     lines_pred = pred.readlines()
311
312     y_pred = np.zeros(len(lines_pred))
313
314     for i in range(0, len(lines_pred)):
315         ob = lines_pred[i]
316         ob = ob.replace('\n', '')
317         ob = float(ob)
318         y_pred[i] = float(ob)
319
320     mse = mean_squared_error(y_true, y_pred)
321     rmse = math.sqrt(mse)
322     mae = mean_absolute_error(y_true, y_pred)
323
324     return rmse, mae
```

A.5 Example

For ease of use, we provide an example demonstrating the creation of a dense matrix and the subsequent generation of a matrix with a sparsity level of 99%. This is followed by applying the algorithms described in Chapter 6.

```
1  # Load data
2  dataset_path = "file_path"
3  ratings_df = pd.read_csv(os.path.join(dataset_path, "ratings.csv"), sep=",")
4  ratings_df.drop(columns=["timestamp"], inplace=True)
5
6  # Determine sparsity of matrix
7  calculate_dataframe_sparsity(ratings_df)
8
9  # Label encode users and items so that the IDs start at 0
10 ratings_df = label_encode_ids(ratings_df)
11
12 # User and item proportions of original matrix
```

```
user_props, item_props = user_and_item_proportions(ratings_df)

# Estimate missing ratings using Surprise package. That is, create a dense matrix
K = 100
R_hat, p,q = create_dense_matrix(ratings_df, k = K)
np.savetxt('R_hat_' + str(K), R_hat, fmt='%f')
np.savetxt('p_' + str(K), p, fmt='%f')
np.savetxt('q' + str(K), q, fmt='%f')

# Create 5 sparse matrices (Sparsity: 99\%)
R_100 = np.loadtxt('R_hat_100', dtype=float)
st = time.time()
for i in range(0,5):
    print(i)
    R_100_s99 = create_sparse_matrix(R_100, user_props, item_props, .99)
    np.savetxt('R_100_s99_' + str(i), R_100_s99, fmt='%f')
et = time.time()
print(et-st)

# Fit CF algorithms on first matrix of sparsity 99. Repeat for reaming matrices.
data0, trainset0, testset0 = get_data('R_100_s99_0')

models = {
    "baseline": {
        "train": get_baseline_only_model,
            "predict": baseline_predict
    },
    "CF_userbased_model": {
         "train": get_best_CF_userbased_model,
            "predict": cf_users_predict
    },
    "CF_itembased_model": {
         "train": get_best_CF_itembased_model,
            "predict": cf_item_predict
    },
    "svd": {
        "train": get_best_svd_model,
          "predict": svd_predict
     },
    "svd_unbiases": {
        "train": get_best_svd_unbiased_model,
            "predict": svd_unbiased_predict
    },
    "svdpp": {
```

```
        "train": get_best_svdpp_model,
          "predict": svpp_predict
     },
    "nmf": {
        "train": get_best_nmf_model,
        "predict": nmf_predict
    }

}

for model_name, model_functions in models.items():
    print("Working with",model_name)
    print("Finding best model + training:")
    model = model_functions["train"](data0, trainset0)

    print("Done training. Predicting:")
    model_functions["predict"](model, trainset0, testset0)
    print(model_name,"completed")

# Fit FM algorithms on first matrix of sparsity 99. Repeat for reaming matrices.
random.seed(2)

# get the data to work with
_, _, _,_, threshold0 = get_data('R_100_s99_0')

# transform the data into the required format for fms
convert_ratings_to_fm("R_100_s99_0_raw", "s99_0_fm", [0,1],2,_model = "fm")

# split the data into a train and test set
train_test_split_for_fm("s99_0_fm")

random.seed(2)
#create_cv_train_test_splits_for_fm("s99_0_fm_train")
s = time.time()
best_rmse, best_params = GridSearchFM("s99_0_fm_train", l_rates = np.linspace(0.001,
    0.002, 5),
            regs = np.linspace(0.5, 0.65, 5), n_factors = [20, 40, 50], fm =False)

param = {'task':'reg',
         'lr':best_params[0],
         'lambda':best_params[1],
         'metric':'rmse',
         'fold':3,
         'opt':'sgd',
```

```
100          'k': best_params[2]}
101 pred = FM_predict("s99_0_fm_train", "s99_0_fm_test", params = param, fm = True)
102
103 e = time.time()
104
105
106 # Fit FFM algorithms on first matrix of sparsity 99. Repeat for reaming matrices.
107 random.seed(2)
108 # get the data to work with
109 _, _, _,_, threshold0 = get_data('R_100_s99_0')
110
111 # transform the data into the required format for fms
112 convert_ratings_to_fm("R_100_s99_0_raw", "s99_0_ffm", [0,1],2,_model = "ffm")
113
114 # split the data into a train and test set
115 train_test_split_for_fm("s99_0_ffm")
116
117 random.seed(2)
118 #create_cv_train_test_splits_for_fm("s99_0_fm_train")
119 s = time.time()
120 best_rmse, best_params = GridSearchFM("s99_0_ffm_train", l_rates = np.linspace(0.001,
    0.002, 5),
121              regs = np.linspace(0.5, 0.65, 5), n_factors = [20, 40, 50], fm =False)
122
123 param = {'task':'reg',
124          'lr':best_params[0],
125          'lambda':best_params[1],
126          'metric':'rmse',
127          'fold':3,
128          'opt':'sgd',
129          'k': best_params[2]}
130 pred = FM_predict("s99_0_ffm_train", "s99_0_ffm_test", params = param, fm = True)
131 e = time.time()
```

Appendix B

Chapter 6: Model Parameters

The model parameters used in the models in Chapter 6 are given in this section. If not stated, the default parameters for the model where used. For the collaborative filtering models, a cosine similarity measure was used. Stochastic gradient descent is used to minimise objective functions. We use λ to denote a regularisation parameter and we use γ to denote the learning rate. The number of epochs is specified by n_{epochs} and the number of latent factors is denoted by $n_{factors}$.

Table B.1: Algorithm parameters used for the models on the MovieLens dataset at sparsity level 99% and 98%.

Algorithm	**Parameters at Sparsity 99%**	**Parameters at Sparsity 98%**
Baised rating	$\gamma = 0.007$ $n_{epochs} = 5$	$\gamma = 0.007$ $n_{epochs} = 5$
CF (user-based)	$k = 15$	$k = 25$
CF (item-based)	$k = 50$	$k = 50$
SVD (unbiased)	$\gamma_{all} = 0.01$ $\lambda_{all} = 0.02$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.009$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$
SVD (biased)	$\gamma_{all} = 0.01$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.009$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$
SVD++	$\gamma_{all} = 0.007$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.007$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$
NMF	$\lambda_{q_i} = 0.06$ $\lambda_{p_u} = 0.06$ $n_{epochs} = 20$ $n_{factors} = 10$	$\lambda_{q_i} = 0.04$ $\lambda_{p_u} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 50$
FM	$\gamma = 0.00125$ $\lambda = 0.5$ $n_{factors} = 20$	$\gamma = 0.001$ $\lambda = 0.5375$ $n_{factors} = 20$
FFM	$\gamma = 0.001$ $\lambda = 0.65$ $n_{factors} = 20$	$\gamma = 0.00175$ $\lambda = 0.65$ $n_{factors} = 50$

Table B.2: Algorithm parameters used for the models on the LDOS-CoMoDa dataset at sparsity level 99%, 98% and 95%.

Algorithm	**Parameters at Sparsity 99%**	**Parameters at Sparsity 98%**	**Parameters at Sparsity 95%**
Baised rating	$\gamma = 0.007$ $n_{epochs} = 5$	$\gamma = 0.007$ $n_{epochs} = 5$	$\gamma = 0.007$ $n_{epochs} = 5$
CF (user-based)	$k = 5$	$k = 5$	$k = 5$
CF (item-based)	$k = 15$	$k = 50$	$k = 50$
SVD (unbiased)	$\gamma_{all} = 0.007$ $\lambda_{all} = 0.02$ $n_{epochs} = 5$ $n_{factors} = 20$	$\gamma_{all} = 0.01$ $\lambda_{all} = 0.02$ $n_{epochs} = 10$ $n_{factors} = 100$	$\gamma_{all} = 0.01$ $\lambda_{all} = 0.02$ $n_{epochs} = 10$ $n_{factors} = 20$
SVD (biased)	$\gamma_{all} = 0.01$ $\lambda_{all} = 0.04$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.009$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.01$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$
SVD++	$\gamma_{all} = 0.009$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.009$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$	$\gamma_{all} = 0.007$ $\lambda_{all} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$
NMF	$\lambda_{q_i} = 0.02$ $\lambda_{p_u} = 0.02$ $n_{epochs} = 5$ $n_{factors} = 20$	$\lambda_{q_i} = 0.06$ $\lambda_{p_u} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$	$\lambda_{q_i} = 0.06$ $\lambda_{p_u} = 0.06$ $n_{epochs} = 10$ $n_{factors} = 20$
FM	$\gamma = 0.002$ $\lambda = 0.65$ $n_{factors} = 40$	$\gamma = 0.002$ $\lambda = 0.65$ $n_{factors} = 50$	$\gamma = 0.001$ $\lambda = 0.575$ $n_{factors} = 50$
FFM	$\gamma = 0.00175$ $\lambda = 0.575$ $n_{factors} = 50$	$\gamma = 0.002$ $\lambda = 0.65$ $n_{factors} = 40$	$\gamma = 0.002$ $\lambda = 0.6125$ $n_{factors} = 40$

Appendix C

Source Code: Chapter 7

C.1 Python Code For Pre-processing Data

C.1.1 LDOS-CoMoDa Pre-processing Functions

```
'''
Generate data
'''
def mulan_sparse():
    import pandas as pd
    import numpy as np
    import scipy
    num_folds = 5

    data = get_og_data()
    data = remove_rows_with_minus_one_missing_values(data)
    data = rating_to_pos_and_neg(data)
    cols = list(data.columns)
    cols.remove("rating")
    data = make_one_hot_specific_columns(data, cols = cols, drop = False)
    data["rating"] = data["rating"].astype("uint8")

    split = pos_neg_split(data, folds=num_folds, seed=2)

    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        train_labels_df = keep_columns(get_one_hot_labels(), train)
        train_data_df = train.drop(get_one_hot_labels(), axis = 1).reset_index(drop=True)
        test_labels_df = keep_columns(get_one_hot_labels(), test)
        test_data_df = test.drop(get_one_hot_labels(), axis = 1).reset_index(drop=True)
```

```
27          save_to_arff(scipy.sparse.csr_matrix(test_data_df.values), scipy.sparse.
        csr_matrix(test_labels_df.values), filename="test"+str(i)+".arff")
28          save_to_arff(scipy.sparse.csr_matrix(train_data_df.values), scipy.sparse.
        csr_matrix(train_labels_df.values), filename="train"+str(i)+".arff")
29
30  def com_generate_pos_neg_5_nominal_folds_for_mulan(save_type="arff", save = False):
31      if save_type != "arff" and save_type != "csv":
32          print("Save type must be 'arff' or 'csv'")
33          return
34      import pandas as pd
35      import numpy as np
36      num_folds = 5
37      data = get_og_data()
38      data = remove_rows_with_minus_one_missing_values(data)
39      data = rating_to_pos_and_neg(data)
40      data = make_one_hot_specific_columns(data, cols = get_og_labels(), drop = False)
41      split = pos_neg_split(data, folds=num_folds, seed=2)
42      for i in range(num_folds):
43          train = split[i][0]
44          test = split[i][1]
45          if save:
46              if save_type == "arff":
47                  tr = convert_df_to_arff_string(train, str(i)+"_train")
48                  with open("com_pos_neg_"+str(i)+"_train.arff", "w") as text_file:
49                      text_file.write(tr)
50
51                  te = convert_df_to_arff_string(test, str(i)+"_train")
52                  with open("com_pos_neg_"+str(i)+"_test.arff", "w") as text_file:
53                      text_file.write(te)
54
55              elif save_type == "csv":
56                  save_df_as_csv(train, "csv/com_pos_neg_"+str(i)+"_train.csv")
57                  save_df_as_csv(test, "csv/com_pos_neg_"+str(i)+"_test.csv")
58
59  def generate_og_data(save):
60      import pandas as pd
61      raw = get_og_raw_data()
62      df = keep_columns(get_og_cols(), raw)
63      if save:
64          save_df_as_csv(df, "baseline_processed_data.csv")
65      return df
66
67  def com_generate_pos_neg_5_folds_regression(save = False):
68      import pandas as pd
```

```
    import numpy as np
    num_folds = 5
    data = get_og_data()
    data = remove_rows_with_minus_one_missing_values(data)
    data = rating_to_pos_and_neg(data)
    data = make_one_hot_specific_columns(data, cols = get_og_labels(), drop = True)
    split = pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        X_train, y_train = get_X_y_one_hot_labels(train)
        X_test, y_test = get_X_y_one_hot_labels(test)
        if save:
            save_df_as_csv(X_train, "com_pos_neg_reg_"+str(i)+"_X_train.csv")
            save_df_as_csv(X_test, "com_pos_neg_reg_"+str(i)+"_X_test.csv")
            save_df_as_csv(y_train, "com_pos_neg_reg_"+str(i)+"_y_train.csv")
            save_df_as_csv(y_test, "com_pos_neg_reg_"+str(i)+"_y_test.csv")

def com_generate_pos_neg_5_folds(save = False):
    import pandas as pd
    import numpy as np
    num_folds = 5
    data = get_og_data()
    data = remove_rows_with_minus_one_missing_values(data)
    data = rating_to_pos_and_neg(data)
    data = make_one_hot_specific_columns(data, cols = get_og_labels())
    split = pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        X_train, y_train = get_X_y_one_hot_labels(train)
        X_test, y_test = get_X_y_one_hot_labels(test)
        if save:
            save_df_as_csv(X_train, "com_pos_neg_"+str(i)+"_X_train.csv")
            save_df_as_csv(X_test, "com_pos_neg_"+str(i)+"_X_test.csv")
            save_df_as_csv(y_train, "com_pos_neg_"+str(i)+"_y_train.csv")
            save_df_as_csv(y_test, "com_pos_neg_"+str(i)+"_y_test.csv")
#     return X_train, y_train, X_test, y_test

def com_generate_pos_neg_5_folds_one_hot(save = False):
    import pandas as pd
    import numpy as np
    num_folds = 5
    data = get_og_data()
```

```
    cols = set(data.columns)
    data = remove_rows_with_minus_one_missing_values(data)
    data = rating_to_pos_and_neg(data)
    data = make_one_hot_specific_columns(data, list(cols - set(['rating'])))
    split = pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        X_train, y_train = get_X_y_one_hot_labels(train)
        X_test, y_test = get_X_y_one_hot_labels(test)
        if save:
            save_df_as_csv(X_train, "com_pos_neg_one_hot_"+str(i)+"_X_train.csv")
            save_df_as_csv(X_test, "com_pos_neg_one_hot_"+str(i)+"_X_test.csv")
            save_df_as_csv(y_train, "com_pos_neg_one_hot_"+str(i)+"_y_train.csv")
            save_df_as_csv(y_test, "com_pos_neg_one_hot_"+str(i)+"_y_test.csv")
#     return X_train, y_train, X_test, y_test

def com_generate_pos_neg_5_folds_with_missing_values(save = False):
    import pandas as pd
    import numpy as np
    num_folds = 5
    data = get_og_data()
    data = rating_to_pos_and_neg(data)
    data = remove_obs_with_missing_values_in_labels(data)#
    data = replace_missing_values_with_question_mark(data)#
    data = make_one_hot_specific_columns(data, cols = get_og_labels())
    split = pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        X_train, y_train = get_X_y_one_hot_labels(train)
        X_test, y_test = get_X_y_one_hot_labels(test)
        if save:
            save_df_as_csv(X_train, "com_pos_neg_"+str(i)+"_X_train.csv")
            save_df_as_csv(X_test, "com_pos_neg_"+str(i)+"_X_test.csv")
            save_df_as_csv(y_train, "com_pos_neg_"+str(i)+"_y_train.csv")
            save_df_as_csv(y_test, "com_pos_neg_"+str(i)+"_y_test.csv")

def generate_entire_data_set_with_missing_values(save = False):
    # removes missing values from the response variables
    # converts missing values in predictors to '?'
    import pandas as pd
    import numpy as np
    num_folds = 5
```

```
157     data = get_og_data()
158     data = rating_to_pos_and_neg(data)
159     data = remove_obs_with_missing_values_in_labels(data)#
160     data = replace_missing_values_with_question_mark(data)#
161     data = make_one_hot_specific_columns(data, cols = get_og_labels())
162     if save:
163             save_df_as_csv(data, "all_data_with_question_marks.csv")
164
165 def generate_simple_one_hot_spilt(save = False):
166     import pandas as pd
167     import numpy as np
168     data = get_og_data()
169     data = remove_rows_with_minus_one_missing_values(data)
170     data = rating_to_pos_and_neg(data)
171     data = one_hot_all_cols(data)
172     train, test = simple_20_80_split(data)
173     X_train, y_train = get_X_y_one_hot_labels(train)
174     X_test, y_test = get_X_y_one_hot_labels(test)
175     if save:
176         save_df_as_csv(X_train, "simple_X_train.csv")
177         save_df_as_csv(X_test, "simple_X_test.csv")
178         save_df_as_csv(y_train, "simple_y_train.csv")
179         save_df_as_csv(y_test, "simple_y_test.csv")
180     return X_train, y_train, X_test, y_test
181
182 def generate_simple_split(save = False):
183     import pandas as pd
184     import numpy as np
185     data = get_og_data()
186     data = remove_rows_with_minus_one_missing_values(data)
187     data = rating_to_pos_and_neg(data)
188     data = make_one_hot_specific_columns(data, cols = get_og_labels())
189     train, test = simple_20_80_split(data)
190     X_train, y_train = get_X_y_one_hot_labels(train)
191     X_test, y_test = get_X_y_one_hot_labels(test)
192     if save:
193         save_df_as_csv(X_train, "simple_X_train.csv")
194         save_df_as_csv(X_test, "simple_X_test.csv")
195         save_df_as_csv(y_train, "simple_y_train.csv")
196         save_df_as_csv(y_test, "simple_y_test.csv")
197     return X_train, y_train, X_test, y_test
198
199 def save_df_as_csv(df, filename):
200     df.to_csv(filename, index = False)
```

```
201
202 def convert_df_to_numeric_arff_string(df, rel):
203     import arff
204     xor_dataset = {
205         'description': '',
206         'relation': rel,
207         'attributes': get_ta_numeric_attributes(),
208         'data': df.values
209     }
210     return arff.dumps(xor_dataset)
211
212 def convert_df_to_arff_string(df, rel):
213     import arff
214     xor_dataset = {
215         'description': '',
216         'relation': rel,
217         'attributes': get_nominal_attributes(df),
218         'data': df.values
219     }
220     return arff.dumps(xor_dataset)
221
222 '''
223 Get data and columns
224 '''
225
226 def get_og_data():
227     import pandas as pd
228     return pd.read_csv("baseline_processed_data.csv")
229
230 def get_og_raw_data():
231     import pandas as pd
232     return pd.read_csv("../Raw/CoMoDa data/LDOS-CoMoDa.csv")
233
234 def get_og_cols():
235     return ['userID', 'itemID', 'rating', 'sex', 'time', 'daytype', 'location', 'social',
236                                 'movieCountry', 'movieLanguage', 'movieYear', 'genre1']
237
238 def get_og_labels():
239     return ['time', 'daytype', 'location', 'social']
240
241 def get_one_hot_labels():
242     return ['time_4', 'time_1', 'time_2', 'time_3', 'daytype_3', 'daytype_1', 'daytype_2
    ', 'location_3', 'location_1', 'location_2', 'social_7', 'social_1', 'social_2', '
    social_3', 'social_4', 'social_5', 'social_6']
```

```
243
244 '''
245 Process data
246 '''
247 def keep_columns(keep_columns, df):
248     keep_cols = set(keep_columns)
249     cols = set(df.columns)
250     df = df.drop(list(cols - keep_cols), axis = 1).reset_index(drop=True)
251
252     if (len(df.columns) != len(keep_cols)):
253         print("Something went wrong. Not all keep cols are present in dataframe.")
254         print("Missmatch cols:",set(keep_columns)-set(df.columns))
255     return df
256
257 def remove_rows_with_minus_one_missing_values(df):
258     replaced = df.replace({-1: None})
259     return replaced.dropna().reset_index(drop = True)
260
261 def replace_missing_values_with_question_mark(df):
262     replaced = df.replace({-1: '?'})
263     return replaced.reset_index(drop = True)
264
265 def remove_obs_with_missing_values_in_labels(df):
266     for label in get_og_labels():
267         df[label] = df[label].replace({-1: None})
268
269     return df.dropna().reset_index(drop = True)
270
271 def make_one_hot_specific_columns(df, cols, drop = False):
272     import pandas as pd
273     return pd.get_dummies(df, columns = cols, drop_first=drop)
274
275 def rating_to_pos_and_neg(df):
276     df['rating'][df['rating'] < 4] = 0
277     df['rating'][df['rating'] > 3] = 1
278     return df
279
280 def one_hot_all_cols(df):
281     import pandas as pd
282     cols = list(df.columns)
283     return pd.get_dummies(df, columns = cols)
284
285 def get_X_y_one_hot_labels(df):
286     y_labels = get_og_labels()
```

```
    one_hot_cols = list(df.columns)
    y_one_hot_labels = [x for x in one_hot_cols if any(x.startswith(y) for y in y_labels)
    ]
    X_one_hot_labels = list(set(one_hot_cols) - set(y_one_hot_labels))
    if len(y_one_hot_labels) + len(X_one_hot_labels) != len(one_hot_cols):
        print("Something went wrong.")
    return keep_columns(X_one_hot_labels, df), keep_columns(y_one_hot_labels, df)

def join_dfs(df1, df2):
    import pandas as pd
    return pd.concat([df1,df2]).reset_index(drop=True)

def pos_neg_split(df, folds=5, seed=1):
    import pandas as pd
    import numpy as np
    from sklearn.utils import shuffle
    shuffled = shuffle(df, random_state = seed)
    pos = shuffled[shuffled["rating"] == 1]
    neg = shuffled[shuffled["rating"] == 0]
    fold_data = []
    N = pos.shape[0]
    fold_size = int(N/folds)
    for i in range(folds):
        start_ind = i*fold_size
        end_ind = (i+1)*fold_size
        test_fold = pos.iloc[start_ind:end_ind].reset_index(drop=True)
        train_fold = join_dfs(pos.iloc[0:start_ind], pos.iloc[end_ind:N])
        train_fold = join_dfs(train_fold, neg)
        fold_data.append((train_fold, test_fold))
    return fold_data
```

C.1.2 TripAdvisor Pre-processing Functions

```
'''
Generate data
'''
def ta_get_one_hot_labels():
    return ['TripType_BUSINESS','TripType_COUPLES', 'TripType_FAMILY', 'TripType_FRIENDS
    ', 'TripType_SOLO']

def ta_generate_pos_neg_5_numeric_folds_for_mulan(save_type="arff", save = False):
    if save_type != "arff" and save_type != "csv":
        print("Save type must be 'arff' or 'csv'")
        return
    import pandas as pd
```

```
    import numpy as np
    from utils import convert_df_to_numeric_arff_string, save_df_as_csv, keep_columns
    from sklearn.preprocessing import LabelEncoder
    num_folds = 5
    data = ta_get_raw_data()
    data = ta_rating_to_pos_and_neg(data)
    data = ta_make_one_hot_specific_columns(data, ["TripType"])
    for col in data.columns:
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col])
    split = ta_pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        if save:
            if save_type == "arff":
                tr = convert_df_to_numeric_arff_string(train, str(i)+"_train")
                with open("ta_pos_neg_"+str(i)+"_train.arff", "w") as text_file:
                    text_file.write(tr)

                te = convert_df_to_numeric_arff_string(test, str(i)+"_train")
                with open("ta_pos_neg_"+str(i)+"_test.arff", "w") as text_file:
                    text_file.write(te)

            elif save_type == "csv":
                save_df_as_csv(train, "csv/ta_pos_neg_"+str(i)+"_train.csv")
                save_df_as_csv(test, "csv/ta_pos_neg_"+str(i)+"_test.csv")


def ta_generate_pos_neg_5_nominal_folds_for_mulan(save_type="arff", save = False):
    if save_type != "arff" and save_type != "csv":
        print("Save type must be 'arff' or 'csv'")
        return
    import pandas as pd
    import numpy as np
    from utils import convert_df_to_arff_string, save_df_as_csv, keep_columns
    from sklearn.preprocessing import LabelEncoder
    num_folds = 5
    data = ta_get_raw_data()
    data = ta_rating_to_pos_and_neg(data)
    data = ta_make_one_hot_specific_columns(data, ["TripType"])
    for col in data.columns:
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col])
```

```
    split = ta_pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        if save:
            if save_type == "arff":
                tr = convert_df_to_arff_string(train, str(i)+"_train")
                with open("ta_pos_neg_"+str(i)+"_train.arff", "w") as text_file:
                    text_file.write(tr)

                te = convert_df_to_arff_string(test, str(i)+"_train")
                with open("ta_pos_neg_"+str(i)+"_test.arff", "w") as text_file:
                    text_file.write(te)

            elif save_type == "csv":
                save_df_as_csv(train, "csv/ta_pos_neg_"+str(i)+"_train.csv")
                save_df_as_csv(test, "csv/ta_pos_neg_"+str(i)+"_test.csv")

def ta_generate_pos_neg_5_folds_for_regression(save = False):
    import pandas as pd
    import numpy as np
    from utils import save_df_as_csv, keep_columns
    from sklearn.preprocessing import LabelEncoder
    num_folds = 5
    data = ta_get_raw_data()
    data = ta_rating_to_pos_and_neg(data)
    data = ta_make_one_hot_specific_columns(data, ["TripType"], drop=True)
    for col in data.columns:
        le = LabelEncoder()
        data[col] = le.fit_transform(data[col])
    split = ta_pos_neg_split(data, folds=num_folds, seed=2)
    for i in range(num_folds):
        train = split[i][0]
        test = split[i][1]
        X_train, y_train = ta_get_X_y_one_hot_labels(train)
        X_test, y_test = ta_get_X_y_one_hot_labels(test)

        if save:
            save_df_as_csv(X_train, "ta_pos_neg_reg_"+str(i)+"_X_train.csv")
            save_df_as_csv(X_test, "ta_pos_neg_reg_"+str(i)+"_X_test.csv")
            save_df_as_csv(y_train, "ta_pos_neg_reg_"+str(i)+"_y_train.csv")
            save_df_as_csv(y_test, "ta_pos_neg_reg_"+str(i)+"_y_test.csv")

'''
```

```
Get data and columns
'''
def ta_get_raw_data():
    import pandas as pd
    return pd.read_csv("../Raw/Data_TripAdvisor_v2.csv")
'''
Process data
'''
def remove_columns(remove_cols, df):
    df = df.drop(remove_cols, axis = 1).reset_index(drop=True)
    return df

def ta_rating_to_pos_and_neg(df):
    df['Rating'][df['Rating'] < 4] = 0
    df['Rating'][df['Rating'] > 3] = 1
    return df

def ta_make_one_hot_specific_columns(df, cols, drop=False):
    import pandas as pd
    return pd.get_dummies(df, columns = cols, drop_first=drop)

def ta_get_X_y_one_hot_labels(df):
    from utils import keep_columns
    y_labels = ['TripType']
    one_hot_cols = list(df.columns)
    y_one_hot_labels = [x for x in one_hot_cols if any(x.startswith(y) for y in y_labels)
    ]
    X_one_hot_labels = list(set(one_hot_cols) - set(y_one_hot_labels))
    if len(y_one_hot_labels) + len(X_one_hot_labels) != len(one_hot_cols):
        print("Something went wrong.")
    return keep_columns(X_one_hot_labels, df), keep_columns(y_one_hot_labels, df)

def join_dfs(df1, df2):
    import pandas as pd
    return pd.concat([df1,df2]).reset_index(drop=True)

def ta_pos_neg_split(df, folds=5, seed=1):
    import pandas as pd
    import numpy as np
    from sklearn.utils import shuffle
    shuffled = shuffle(df, random_state = seed)
    pos = shuffled[shuffled["Rating"] == 1]
    neg = shuffled[shuffled["Rating"] == 0]
    fold_data = []
```

```
143     N = pos.shape[0]
144     fold_size = int(N/folds)
145     for i in range(folds):
146         start_ind = i*fold_size
147         end_ind = (i+1)*fold_size
148         test_fold = pos.iloc[start_ind:end_ind].reset_index(drop=True)
149         train_fold = join_dfs(pos.iloc[0:start_ind], pos.iloc[end_ind:N])
150         train_fold = join_dfs(train_fold, neg)
151         fold_data.append((train_fold, test_fold))
152     return fold_data
```

C.2 Java Code For MLC Algorithms

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import mulan.classifier.MultiLabelLearner;
import mulan.classifier.lazy.BRkNN;
import mulan.classifier.lazy.MLkNN;
import mulan.classifier.meta.RAkEL;
import mulan.classifier.transformation.BinaryRelevance;
import mulan.classifier.transformation.ClassifierChain;
import mulan.classifier.transformation.LabelPowerset;
import mulan.data.InvalidDataFormatException;
import mulan.data.MultiLabelInstances;
import mulan.evaluation.Evaluation;
import mulan.evaluation.Evaluator;
import mulan.evaluation.measure.ExampleBasedAccuracy;
import mulan.evaluation.measure.ExampleBasedPrecision;
import mulan.evaluation.measure.ExampleBasedRecall;
import mulan.evaluation.measure.HammingLoss;
import mulan.evaluation.measure.Measure;
import weka.classifiers.bayes.BayesNet;
import weka.classifiers.functions.SMO;
import weka.classifiers.lazy.IBk;
import weka.classifiers.trees.J48;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;

public class Experiments2 {
  public static List<Measure> getMeasures(int numLabels) {
    List<Measure> measuresList = new ArrayList<Measure>();
    measuresList.add(new ExampleBasedPrecision());
    measuresList.add(new ExampleBasedRecall());
    measuresList.add(new ExampleBasedAccuracy());
    measuresList.add(new HammingLoss());
    return measuresList;
  }

  public static J48 getJ48() throws Exception {
    String[] options = { "-U" };
    J48 j48 = new J48();
```

```
44      j48.setOptions(options);
45      return j48;
46    }
47
48    public static IBk getKnn() {
49      IBk knn = new IBk();
50      knn.setCrossValidate(true);
51      return knn;
52    }
53
54    public static Map<String, MultiLabelLearner> getLearners() throws Exception {
55      Map<String, MultiLabelLearner> map = new HashMap<String, MultiLabelLearner>();
56
57      map.put("BR-KNN", new BinaryRelevance(getKnn()));
58      map.put("BR-J48", new BinaryRelevance(getJ48()));
59      map.put("BR-BN", new BinaryRelevance(new BayesNet()));
60
61      map.put("CC-KNN", new ClassifierChain(getKnn()));
62      map.put("CC-J48", new ClassifierChain(getJ48()));
63      map.put("CC-BN", new ClassifierChain(new BayesNet()));
64
65      map.put("LP-KNN", new LabelPowerset(getKnn()));
66      map.put("LP-J48", new LabelPowerset(getJ48()));
67      map.put("LP-BN", new LabelPowerset(new BayesNet()));
68
69      map.put("RAkEL-KNN", new RAkEL(new LabelPowerset(getKnn())));
70      map.put("RAkEL-J48", new RAkEL(new LabelPowerset(getJ48())));
71      map.put("RAkEL-BN", new RAkEL(new LabelPowerset(new BayesNet())));
72
73      map.put("MLkNN", new MLkNN());
74      map.put("BRkNN", new BRkNN());
75      return map;
76    }
77
78    public static void writeToCsv(String filename, List<String> headers, List<List<String>>
        rows) throws IOException {
79      FileWriter writer = new FileWriter(filename);
80      CSVUtils.writeLine(writer, headers);
81
82      for (List<String> row : rows) {
83        CSVUtils.writeLine(writer, row, ',', '"');
84      }
85      writer.flush();
86      writer.close();
```

```
87   }
88
89   public static void main(String[] args) throws Exception {
90     String xmlFilename = "ta.xml";
91     int num_folds = 5;
92     int numMeasures = 4;
93     String dataType = "nominal"; // nominal or numeric or OneHot
94     String dataSource = "TripAdvisor"; // COM or TripAdvisor
95
96 //     String xmlFilename = "comoda.xml";
97 //     int num_folds = 5;
98 //     int numMeasures = 4;
99 //     String dataType = "nominal"; // nominal or numeric or OneHot
100 //     String dataSource = "COM"; // COM or TripAdvisor
101
102     for (String key : getLearners().keySet()) {
103       try {
104         MultiLabelLearner learner = getLearners().get(key);
105         System.out.println(key);
106         List<String> headers = new ArrayList<String>();
107         List<List<String>> rows = new ArrayList<List<String>>();
108         List<Double> average = new ArrayList<Double>();
109         for (int i = 0; i < numMeasures; i++) {
110           average.add(0.0);
111         }
112         for (int i = 0; i < num_folds; i++) {
113 //           String testArffFilename = "com_nominal/com_pos_neg_" + i + "_test.arff";
114 //           String trainArffFilename = "com_nominal/com_pos_neg_" + i + "_train.arff";
115           String testArffFilename = "ta_arff_nominal/ta_pos_neg_"+i+"_test.arff";
116           String trainArffFilename = "ta_arff_nominal/ta_pos_neg_"+i+"_train.arff";
117
118           MultiLabelInstances trainDataset = new MultiLabelInstances(trainArffFilename,
    xmlFilename);
119           MultiLabelInstances testDataset = new MultiLabelInstances(testArffFilename,
    xmlFilename);
120
121           learner.build(trainDataset);
122
123           Evaluator eval = new Evaluator();
124           Evaluation results;
125           results = eval.evaluate(learner, testDataset, getMeasures(40));
126           if (i == 0) {
127             headers = new ArrayList<String>();
128             for (int j = 0; j < numMeasures; j++) {
```

```
129             headers.add(results.getMeasures().get(j).getName());
130           }
131
132           System.out.println(results.getMeasures().get(0).getName() + ","
133               + results.getMeasures().get(1).getName() + "," + results.getMeasures().
    get(2).getName()
134               + "," + results.getMeasures().get(3).getName());
135         }
136         List<String> row = new ArrayList<String>();
137         for (int j = 0; j < numMeasures; j++) {
138           row.add(results.getMeasures().get(j).getValue() + "");
139
140           average.set(j, (average.get(j)+results.getMeasures().get(j).getValue()));
141
142         }
143
144         rows.add(row);
145         System.out.println(results.getMeasures().get(0).getValue() + ","
146             + results.getMeasures().get(1).getValue() + "," + results.getMeasures().get
    (2).getValue()
147             + "," + results.getMeasures().get(3).getValue());
148       }
149       List<String> line = new ArrayList<String>();
150       for (int j = 0; j < numMeasures; j++) {
151         line.add("-");
152       }
153       System.out.println("- - - -");
154       List<String> averageAsString = new ArrayList<String>();
155       for (int j = 0; j < numMeasures; j++) {
156         averageAsString.add(average.get(j)/(num_folds) + "");
157       }
158       System.out.println(averageAsString.get(0) + ","
159           + averageAsString.get(1) + "," + averageAsString.get(2)
160           + "," + averageAsString.get(3));
161       rows.add(line);
162       rows.add(averageAsString);
163       writeToCsv("results/"+dataSource+"/" + dataType + "/" + key + ".csv", headers,
    rows);
164     } catch (Exception e) {
165       System.out.println("Failed for the following reason:");
166       System.out.println(e.getMessage());
167     }
168     System.out.println("\n\n");
169   }
```

```
170     }
171 }
```

C.3 R Code For Regression Techniques

C.3.1 Functions Needed To Apply Regression Techniques

```
fAcc <- function(ylabels,zlabels)
{
  ymaalz = ylabels*zlabels
  yinterseksiez = sum(ymaalz)
  nylabels=sum(ylabels)
  nzlabels=sum(zlabels)
  yverenigz = nylabels+nzlabels-yinterseksiez
  proportion5 = yinterseksiez/yverenigz
  return("acc"=proportion5)
}

fF12 <- function(ylabels,zlabels)
{
  ymaalz = ylabels*zlabels
  yinterseksiez = sum(ymaalz)
  nzlabels = sum(zlabels)
  proportion2 = yinterseksiez/nzlabels
  precision = proportion2
  nylabels = sum(ylabels)
  proportion3 = yinterseksiez/nylabels
  recall = proportion3
  F12 = 1/mean(c(1/precision,1/recall))
  output=list("F12"=F12)
  return(output)
}

fHloss <- function(ylabels,zlabels)
{
  yminz = ylabels-zlabels
  ydeltaz = sum(abs(yminz))
  return(ydeltaz)
}

Pmeasures_com <- function(ylabels,zlabels){


  # This program computes various measures of classification accuracy.
  # The input to the program is:
  #     1. ylabels: an indicator matrix containing the true labels for a set of Nnew new
    cases
  #     2. zlabels: an indicator matrix containing the predicted labels for the Nnew new
```

```
  cases
41
42 # The following measures are computed:
43
44 # Example-based measures:
45 #     1. Hamming loss (Hloss)
46 #     2. Classification accuracy (clacc)
47 #     3. Precision (precision)
48 #     4. Recall (recall)
49 #     5. F-one, first version (F11)
50 #     6. F-one, second version (F12)
51 #     7. Accuracy (accuracy)
52 #    ** Note that precision and recall was accidentally swopped
53
54 # Remove rows in the ylabels and zlabels matrices where either one of the matrices
  contain no ones
55
56 # somy=apply(ylabels,1,sum)
57 # somz=apply(zlabels,1,sum)
58 # indy=which(somy==0)
59 # indz=which(somz==0)
60 # indweg=c(indy,indz)
61 # ylabels=ylabels[-indweg,]
62 # zlabels=zlabels[-indweg,]
63
64
65 # We next do some bookkeeping.
66 M <- nrow(zlabels)
67 time_1 <- rep(0, M)
68 daytype_1 <- rep(0, M)
69 location_1 <- rep(0, M)
70 social_1 <- rep(0, M)
71
72 for (i in 1:M) {
73   if (sum(zlabels[i, 1:3]) == 0) {
74     time_1[i] <- 1
75   }
76   if (sum(zlabels[i, 4:5]) == 0) {
77     daytype_1[i] <- 1
78   }
79   if (sum(zlabels[i, 6:7]) == 0) {
80     location_1[i] <- 1
81   }
82   if (sum(zlabels[i, 8:13]) == 0) {
```

```
      social_1[i] <- 1
    }
  }

  zlabels <- cbind(zlabels, time_1, daytype_1, location_1, social_1)

  Nnew = nrow(ylabels)
  q = ncol(ylabels)

  # First, compute the example-based measures.


  yminz = ylabels-zlabels
  ymaalz = ylabels*zlabels
  ydeltaz = apply(yminz,1,function(x) sum(abs(x)))
  nylabels = apply(ylabels,1,sum)
  nzlabels = apply(zlabels,1,sum)
  #print(nylabels)
  #print(nzlabels)

  proportion1 = ydeltaz/q
  Hloss = mean(proportion1)
  clacc = sum(ydeltaz==0)/Nnew

  yinterseksiez = apply(ymaalz,1,sum)
  yverenigz = nylabels+nzlabels-yinterseksiez

  proportion2 = yinterseksiez/nzlabels
  proportion3 = yinterseksiez/nylabels
  proportion4 = yinterseksiez/(nylabels+nzlabels)
  proportion5 = yinterseksiez/yverenigz

  precision = mean(proportion2)
  recall = mean(proportion3)
  F11 = 2*mean(proportion4)
  F12 = 1/mean(c(1/precision,1/recall))
  accuracy = mean(proportion5)

  afvoer = list(Hloss=Hloss,clacc=clacc,precision=precision,recall=recall,F11=F11,F12=F12
    ,accuracy=accuracy)
  return(afvoer)
}

Pmeasures_ta <- function(ylabels,zlabels){
```

```
126
127 # This program computes various measures of classification accuracy.
128 # The input to the program is:
129 #     1. ylabels: an indicator matrix containing the true labels for a set of Nnew new
      cases
130 #     2. zlabels: an indicator matrix containing the predicted labels for the Nnew new
      cases
131
132 # The following measures are computed:
133 # Example-based measures:
134 #     1. Hamming loss (Hloss)
135 #     2. Classification accuracy (clacc)
136 #     3. Precision (precision)
137 #     4. Recall (recall)
138 #     5. F-one, first version (F11)
139 #     6. F-one, second version (F12)
140 #     7. Accuracy (accuracy)
141 #    ** Note that precision and recall was accidentally swopped
142
143
144 # Remove rows in the ylabels and zlabels matrices where either one of the matrices
      contain no ones
145
146 # somy=apply(ylabels,1,sum)
147 # somz=apply(zlabels,1,sum)
148 # indy=which(somy==0)
149 # indz=which(somz==0)
150 # indweg=c(indy,indz)
151 # ylabels=ylabels[-indweg,]
152 # zlabels=zlabels[-indweg,]
153
154 # We next do some bookkeeping.
155 M <- nrow(zlabels)
156
157 TripType_BUSINESS <- rep(0, M)
158
159 for (i in 1:M) {
160   if (sum(zlabels[i, ]) == 0) {
161     TripType_BUSINESS[i] <- 1
162   }
163 }
164 zlabels <- cbind(zlabels, TripType_BUSINESS)
165
166 Nnew = nrow(ylabels)
```

```
167   q = ncol(ylabels)
168
169   # First, compute the example-based measures.
170
171   yminz = ylabels-zlabels
172   ymaalz = ylabels*zlabels
173   ydeltaz = apply(yminz,1,function(x) sum(abs(x)))
174   nylabels = apply(ylabels,1,sum)
175   nzlabels = apply(zlabels,1,sum)
176   #print(nylabels)
177   #print(nzlabels)
178
179   proportion1 = ydeltaz/q
180   Hloss = mean(proportion1)
181   clacc = sum(ydeltaz==0)/Nnew
182
183   yinterseksiez = apply(ymaalz,1,sum)
184   yverenigz = nylabels+nzlabels-yinterseksiez
185
186   proportion2 = yinterseksiez/nzlabels
187   proportion3 = yinterseksiez/nylabels
188   proportion4 = yinterseksiez/(nylabels+nzlabels)
189   proportion5 = yinterseksiez/yverenigz
190
191   precision = mean(proportion2)
192   recall = mean(proportion3)
193   F11 = 2*mean(proportion4)
194   F12 = 1/mean(c(1/precision,1/recall))
195   accuracy = mean(proportion5)
196   afvoer = list(Hloss=Hloss,clacc=clacc,precision=precision,recall=recall,F11=F11,F12=F12
        ,accuracy=accuracy)
197
198   return(afvoer)
199 }
200
201 Pregression_enron <- function (ydata,ftraindata,ftestdata) {
202
203   # This program takes 2 matrices as input:
204   # 1. A matrix of training data labels.
205   # 2. A matrix of f-values to threshold.
206   # It then employs different thresholding approaches to transform the f-values to labels
        .
207
208   tyd1=proc.time()
```

```
ymat = as.matrix(ydata)
ftrainmat = as.matrix(ftraindata)
ftestmat = as.matrix(ftestdata)
Ntrain = nrow(ymat)
Ntest = nrow(ftestmat)
Kval = ncol(ftestmat)

# ylabels1 = matrix(0,Ntest,Kval)


# Quantile threshold procedure

# threshold1 = rep(0,Kval)
# densities = apply(ymat,2,mean)
# for (k in 1:Kval) {
#     vector = ftestdata[,k]
#     threshold1[k] = quantile(vector,1-densities[k],names=FALSE)
#     for (i in 1:Ntest) {
#         ylabels1[i,k] = as.numeric(ftestdata[i,k]>threshold1[k])
#         }
#     }
#
# vRowsNeed1 = (1:nrow(ylabels1))[apply(ylabels1,1,sum) == 0]
#
# for (i in vRowsNeed1) {
#     posNeed1 = which.max(ftraindata[i,])
#     ylabels1[i,posNeed1]=1
# }

# Minimum measure-quantile threshold procedure
waarde = 1500
ylabels2.hl = matrix(0,Ntest,Kval)
ylabels2.f = matrix(0,Ntest,Kval)
ylabels2.acc = matrix(0,Ntest,Kval)
threshold2.hl = rep(0,Kval)
threshold2.f = rep(0,Kval)
threshold2.acc = rep(0,Kval)
ylab = rep(0,Ntrain)
numvCut=ifelse(Ntrain*Kval>waarde,150,Ntrain-1)
onder = min(ftraindata)
bo = max(ftraindata)
stap = (bo-onder)/numvCut
vCut = seq(from=onder,to=bo,by=stap)

```

```
253 for (k in 1:Kval) {
254   vector = ftraindata[,k]
255   vHloss=NULL
256   vF12=NULL
257   vAcc=NULL
258   for (i in vCut) {
259     for (ii in 1:Ntrain) ylab[ii]=as.numeric(ftraindata[ii,k]>i)
260     hloss=fHloss(ylabels=ymat[,k],zlabels=ylab)
261     f12=fF12(ylabels=ymat[,k],zlabels=ylab)
262     acc=fAcc(ylabels=ymat[,k],zlabels=ylab)
263     vHloss=append(vHloss,hloss)
264     vF12=append(vF12,f12)
265     vAcc=append(vAcc,acc)
266   }
267
268   indexmin.hl = which.min(vHloss)
269   indexmax.f = which.max(vF12)
270   indexmax.acc = which.max(vAcc)
271
272   threshold2.hl[k] = vCut[indexmin.hl]
273   threshold2.f[k] = vCut[indexmax.f]
274   threshold2.acc[k] = vCut[indexmax.acc]
275
276   Fn=ecdf(ftraindata[,k])
277   prob.hl=Fn(threshold2.hl[k])
278   prob.f=Fn(threshold2.f[k])
279   prob.acc=Fn(threshold2.acc[k])
280
281   thr.hl=quantile(ftestdata[,k],probs=prob.hl)
282   thr.f=quantile(ftestdata[,k],probs=prob.f)
283   thr.acc=quantile(ftestdata[,k],probs=prob.acc)
284
285   for (i in 1:Ntest){
286     ylabels2.hl[i,k]=as.numeric(ftestdata[i,k]>thr.hl)
287     ylabels2.f[i,k]=as.numeric(ftestdata[i,k]>thr.f)
288     ylabels2.acc[i,k]=as.numeric(ftestdata[i,k]>thr.acc)
289   }
290 }
291
292 vRowsNeed1 = NULL
293 vRowsNeed1 = (1:nrow(ylabels2.hl))[apply(ylabels2.hl,1,sum) == 0]
294
295 for (i in vRowsNeed1) {
296   posNeed1 = which.min(abs(ftestdata[i,]-threshold2.hl))
```

```
297     ylabels2.hl[i,posNeed1]=1
298   }
299
300   vRowsNeed1 = NULL
301   vRowsNeed1 = (1:nrow(ylabels2.f))[apply(ylabels2.f,1,sum) == 0]
302
303   for (i in vRowsNeed1) {
304     posNeed1 = which.min(abs(ftestdata[i,]-threshold2.f))
305     ylabels2.f[i,posNeed1]=1
306   }
307
308   vRowsNeed1 = NULL
309   vRowsNeed1 = (1:nrow(ylabels2.acc))[apply(ylabels2.acc,1,sum) == 0]
310
311   for (i in vRowsNeed1) {
312     posNeed1 = which.min(abs(ftestdata[i,]-threshold2.acc))
313     ylabels2.acc[i,posNeed1]=1
314   }
315
316   tyd2=proc.time()
317
318   # Return the output.
319
320   Output = list("hl"=ylabels2.hl,"f12"=ylabels2.f,"acc"=ylabels2.acc,"t1"=tyd1,"t2"=tyd2)
321   return(Output)
322 }
323
324 Pregression1<-function(r,xdata,ydata,xdatatest)
325 {
326   # In this program various regression based approaches to ML classification are
        investigated.
327   # This is combined with different ways of thresholding the fitted values.
328   #
329   # 1. OLS; 2. CW; 3. RR; 4. FICYREG
330
331   xmat = xdata
332   ymat = ydata
333   xmattest = xdatatest
334
335   N = nrow(xmat)
336   Ntest = nrow(xmattest)
337   p = ncol(xmat)
338   Kval = ncol(ymat)
339
```

```
ftrain1 = matrix(0,N,Kval)
ftest1 = matrix(0,Ntest,Kval)
ytrain1 = matrix(0,N,Kval)
ytest1 = matrix(0,Ntest,Kval)

ftrain2 = matrix(0,N,Kval)
ftest2 = matrix(0,Ntest,Kval)
ytrain2 = matrix(0,N,Kval)
ytest2 = matrix(0,Ntest,Kval)

ftrain3 = matrix(0,N,Kval)
ftest3 = matrix(0,Ntest,Kval)
ytrain3 = matrix(0,N,Kval)
ytest3 = matrix(0,Ntest,Kval)

ftrain4a = matrix(0,N,Kval)
ftrain4 = matrix(0,N,Kval)
ftest4a = matrix(0,Ntest,Kval)
ftest4 = matrix(0,Ntest,Kval)
ytrain4 = matrix(0,N,Kval)
ytest4 = matrix(0,Ntest,Kval)

ftrain5 = matrix(0,N,Kval)
ftest5 = matrix(0,Ntest,Kval)
ytrain5 = matrix(0,N,Kval)
ytest5 = matrix(0,Ntest,Kval)

ftrain6 = matrix(0,N,Kval)
ftest6a = matrix(0,Ntest,Kval)
ftest6 = matrix(0,Ntest,Kval)
ytrain6 = matrix(0,N,Kval)
ytest6 = matrix(0,Ntest,Kval)

# We first scale the data.

# xstd = apply(xmat,2,sd)
# ystd = apply(ymat,2,sd)
#
# print("These predictors have a standard deviation of 0")
# zero_sd <- which(xstd == 0)
# print(zero_sd)
#
# print(dim(xmat))
# xmat <- xmat[, -as.vector(zero_sd)]
```

```
384 # xmattest <- xmattest[, -as.vector(zero_sd)]
385 # print(dim(xmat))
386
387 #xstd = apply(xmat,2,sd)
388 #ystd = apply(ymat,2,sd)
389 #xaverage = apply(xmat,2,mean)
390 #yaverage = apply(ymat,2,mean)
391
392 #xmatstd = scale(xmat,center=xaverage,scale=xstd)
393 #ymatstd = scale(ymat,center=yaverage,scale=ystd)
394 #xmatteststd = scale(xmattest,center=xaverage,scale=xstd)
395
396 xmatstd = as.matrix(xdata)
397 ymatstd = as.matrix(ydata)
398 xmatteststd = as.matrix(xdatatest)
399
400 # Ordinary multiple linear regression is used as base classifier.
401
402 fit1 = lm(ymatstd ~ xmatstd-1)
403 # print("coefs na")
404 #print(fit1$coefficients[1,])
405 # print(sum(is.na(fit1$coefficients)))
406 for (i in 1:N) ftrain1[i,] = xmatstd[i,]%*%fit1$coefficients
407 for (i in 1:Ntest) ftest1[i,] = xmatteststd[i,]%*%fit1$coeff
408 # print(ftrain1)
409 # print(ftest1)
410
411 # Curds and whey regression is used as classifier.
412 # Now we compute the canonical coefficients and transform the response data.
413 # Note that the columns of the output matrices from function cancor define the
      canonical variables.
414
415 canoncor = cancor(xmatstd,ymatstd)
416 Tmat = t(canoncor$ycoef)
417 ccor = canoncor$cor
418 ccor2 = ccor^2
419
420 # Now we fit the transformed response variables separately to the predictor data, using
      OLS.
421 # We also compute the shrinkage factors.
422
423 fitcw = lm(ymatstd ~ xmatstd-1)
424 for (i in 1:N) ftrain4a[i,] = xmatstd[i,]%*%fitcw$coefficients
425 for (i in 1:Ntest) ftest4a[i,] = xmatteststd[i,]%*%fitcw$coefficients
```

```
426
427   rvalue = p/N
428   dtilde = ((1-rvalue)*(ccor2-rvalue))/((1-rvalue)^2*ccor2+(rvalue^2)*(1-ccor2))
429   for (i in 1:length(dtilde)) if (dtilde[i]<0) dtilde[i]=0
430
431   if (nrow(Tmat)>length(dtilde)) dtilde=c(dtilde,rep(0,nrow(Tmat)-length(dtilde)))
432
433   Dmat = diag(dtilde)
434   Bmat = solve(Tmat)%*%Dmat%*%Tmat
435   ftrain4 = ftrain4a%*%t(Bmat)
436   ftest4 = ftest4a%*%t(Bmat)
437   betakap.CW=fit1$coefficients%*%t(Bmat)
438
439   # We now fit a reduced rank regression model to the data.
440   # The number of canonical variables to retain is a parameter, r, which is input to the
        function.
441
442   Dmat.RR = diag(c(rep(1,r),rep(0,Kval-r)))
443   Bmat.RR = solve(Tmat)%*%Dmat.RR%*%Tmat
444   ftrain5 = ftrain4a%*%t(Bmat.RR)
445   ftest5 = ftest4a%*%t(Bmat.RR)
446   betakap.RR=fit1$coefficients%*%t(Bmat.RR)
447
448   # We now fit the FICYREG procedure.
449   # We use the recipe described on page 15 of the Breiman and Friedman paper.
450
451   dtilde.FICYREG = (ccor2-(p-Kval-1)/N)/(ccor2*(1-(p-Kval-1)/N))
452   for (i in 1:length(dtilde.FICYREG)) if (dtilde.FICYREG[i]<0) dtilde.FICYREG[i]=0
453 if (nrow(Tmat)>length(dtilde.FICYREG)) dtilde.FICYREG=c(dtilde.FICYREG,rep(0,nrow(Tmat)-
        length(dtilde.FICYREG)))
454
455   Dmat.FICYREG = diag(dtilde.FICYREG)
456   Bmat.FICYREG = solve(Tmat)%*%Dmat.FICYREG%*%Tmat
457   ftrain6 = ftrain4a%*%t(Bmat.FICYREG)
458   for (i in 1:Ntest) ftest6a[i,] = xmatteststd[i,]%*%fitcw$coefficients
459   ftest6 = ftest6a%*%t(Bmat.FICYREG)
460   betakap.FICYREG=fit1$coefficients%*%t(Bmat.FICYREG)
461
462   # Return the output.
463
464   Output = list("ols.train"=ftrain1,"ridge.train"=ftrain2,"lasso.train"=ftrain3,"CW.train
        "=ftrain4,"RR.train"=ftrain5,
465                 "FICYREG.train"=ftrain6,"ols.test"=ftest1,"ridge.test"=ftest2,"lasso.test
        "=ftest3,"CW.test"=ftest4,"RR.test"=ftest5,
```

```
466                   "FICYREG.test"=ftest6,"ols.betakap"=fit1$coefficients,"cw.betakap"=
      betakap.CW,"rr.betakap"=betakap.RR,
467                   "ficyreg.betakap"=betakap.FICYREG)
468   return(Output)
469 }
470 Pregression2 <- function (ydata,ftraindata,ftestdata) {
471
472   # This program takes 2 matrices as input:
473   # 1. A matrix of training data labels.
474   # 2. A matrix of f-values to threshold.
475   # It then employs different thresholding approaches to transform the f-values to labels
        .
476
477   tyd1=proc.time()
478   ymat = as.matrix(ydata)
479   ftrainmat = as.matrix(ftraindata)
480   ftestmat = as.matrix(ftestdata)
481   Ntrain = nrow(ymat)
482   Ntest = nrow(ftestmat)
483   Kval = ncol(ftestmat)
484
485   ylabels1 = matrix(0,Ntest,Kval)
486
487   # Quantile threshold procedure
488
489   threshold1 = rep(0,Kval)
490   densities = apply(ymat,2,mean)
491
492   for (k in 1:Kval) {
493     vector = ftestdata[,k]
494     threshold1[k] = quantile(vector,1-densities[k],names=FALSE, na.rm = TRUE)
495     for (i in 1:Ntest) {
496       ylabels1[i,k] = as.numeric(ftestdata[i,k]>threshold1[k])
497     }
498   }
499   vRowsNeed1 = (1:nrow(ylabels1))[apply(ylabels1,1,sum) == 0]
500   for (i in vRowsNeed1) {
501     posNeed1 = which.max(ftraindata[i,])
502     ylabels1[i,posNeed1]=1
503   }
504
505   # Minimum measure threshold procedure
506
507   waarde = 1500
```

```
508  ylabels2.hl = matrix(0,Ntest,Kval)
509  ylabels2.f = matrix(0,Ntest,Kval)
510  ylabels2.acc = matrix(0,Ntest,Kval)
511  threshold2.hl = rep(0,Kval)
512  threshold2.f = rep(0,Kval)
513  threshold2.acc = rep(0,Kval)
514  ylab = rep(0,Ntrain)
515  numvCut=ifelse(Ntrain*Kval>waarde,150,Ntrain-1)
516  onder = min(ftraindata)
517  bo = max(ftraindata)
518  stap = (bo-onder)/numvCut
519  vCut = seq(from=onder,to=bo,by=stap)
520
521  for (k in 1:Kval) {
522    vector = ftraindata[,k]
523    #print(k)
524    vHloss=NULL
525    vF12=NULL
526    vAcc=NULL
527    for (i in vCut) {
528      for (ii in 1:Ntrain) ylab[ii] = as.numeric(ftraindata[ii,k]>i)
529      hloss=fHloss(ylabels=ymat[,k],zlabels=ylab)
530      f12=fF12(ylabels=ymat[,k],zlabels=ylab)
531      acc=fAcc(ylabels=ymat[,k],zlabels=ylab)
532      vHloss=append(vHloss,hloss)
533      vF12=append(vF12,f12)
534      vAcc=append(vAcc,acc)
535    }
536
537    indexmin.hl = which.min(vHloss)
538    indexmax.f = which.max(vF12)
539    indexmax.acc = which.max(vAcc)
540
541    threshold2.hl[k] = vCut[indexmin.hl]
542    threshold2.f[k] = vCut[indexmax.f]
543    threshold2.acc[k] = vCut[indexmax.acc]
544
545    for (i in 1:Ntest){
546      ylabels2.hl[i,k] = as.numeric(ftestdata[i,k]>threshold2.hl[k])
547      ylabels2.f[i,k] = as.numeric(ftestdata[i,k]>threshold2.f[k])
548      ylabels2.acc[i,k] = as.numeric(ftestdata[i,k]>threshold2.acc[k])
549    }
550  }
551
```

```
552   vRowsNeed1 = NULL
553   vRowsNeed1 = (1:nrow(ylabels2.hl))[apply(ylabels2.hl,1,sum) == 0]
554
555   for (i in vRowsNeed1) {
556     posNeed1 = which.min(abs(ftestdata[i,]-threshold2.hl))
557     ylabels2.hl[i,posNeed1]=1
558   }
559
560   shl=length(vRowsNeed1)
561
562   vRowsNeed1 = NULL
563   vRowsNeed1 = (1:nrow(ylabels2.f))[apply(ylabels2.f,1,sum) == 0]
564
565   for (i in vRowsNeed1) {
566     posNeed1 = which.min(abs(ftestdata[i,]-threshold2.f))
567     ylabels2.f[i,posNeed1]=1
568   }
569
570   sf=length(vRowsNeed1)
571
572   vRowsNeed1 = NULL
573   vRowsNeed1 = (1:nrow(ylabels2.acc))[apply(ylabels2.acc,1,sum) == 0]
574
575   for (i in vRowsNeed1) {
576     posNeed1 = which.min(abs(ftestdata[i,]-threshold2.acc))
577     ylabels2.acc[i,posNeed1]=1
578   }
579
580   sacc=length(vRowsNeed1)
581
582   tyd2=proc.time()
583
584  # Return the output.
585
586   Output = list("q"=ylabels1,"hl"=ylabels2.hl,"f12"=ylabels2.f,"acc"=ylabels2.acc,"shl"=
        shl,
587                 "sf"=sf,"sacc"=sacc,"vRN1"=vRowsNeed1,"t1"=tyd1,"t2"=tyd2)
588   return(Output)
589 }
```

C.4 Example

Example demonstrating how to apply regression techniques to LDOS-CoMoDa data. First generate data in Python as follows

```
com_generate_pos_neg_5_folds_regression(save = False)
```

In R, run the following.

```
CW_HL <- matrix(0, 3, 7)
colnames(CW_HL) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(CW_HL) <- c('q', 'hl', 'hl2')
CW_F <- matrix(0, 3, 7)
colnames(CW_F) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(CW_F) <- c('q', 'hl', 'hl2')
CW_A <- matrix(0, 3, 7)
colnames(CW_A) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(CW_A) <- c('q', 'hl', 'hl2')
FICY_HL <- matrix(0, 3, 7)
colnames(FICY_HL) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se
    ')
rownames(FICY_HL) <- c('q', 'hl', 'hl2')
FICY_F <- matrix(0, 3, 7)
colnames(FICY_F) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se
    ')
rownames(FICY_F) <- c('q', 'f', 'f2')
FICY_A <- matrix(0, 3, 7)
colnames(FICY_A) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se
    ')
rownames(FICY_A) <- c('q', 'acc', 'acc2')
OLS_HL <- matrix(0, 3, 7)
colnames(OLS_HL) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se
    ')
rownames(OLS_HL) <- c('q', 'hl', 'hl2')
OLS_F <- matrix(0, 3, 7)
colnames(OLS_F) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(OLS_F) <- c('q', 'f', 'f2')
OLS_A <- matrix(0, 3, 7)
colnames(OLS_A) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(OLS_A) <- c('q', 'acc', 'acc2')
RR_HL <- matrix(0, 3, 7)
colnames(RR_HL) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(RR_HL) <- c('q', 'hl', 'hl2')
RR_F <- matrix(0, 3, 7)
colnames(RR_F) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(RR_F) <- c('q', 'f', 'f2')
RR_A <- matrix(0, 3, 7)
colnames(RR_A) <- c('split_0', 'split_1', 'split_2', 'split_3', 'split_4', 'mean', 'se')
rownames(RR_A) <- c('q', 'acc', 'acc2')
LS_CW <- list()
```

```
LS_FICY <- list()
LS_OLS <- list()
LS_RR <- list()

for (split in 0:4) {

    print( split)
    #   Data
    data.train.x <- read.csv(paste("com_pos_neg_reg_",toString(split),"_X_train.csv", sep
    =""))
    data.train.y <- read.csv(paste(com_pos_neg_reg_",toString(split),"_y_train.csv", sep
    =""))
    data.test.x <- read.csv(paste(com_pos_neg_reg_",toString(split),"_X_test.csv", sep
    =""))
    data.test.y <- read.csv(paste("com_pos_neg_reg_",toString(split),"_y_test.csv", sep
    =""))

    M <- nrow(data.test.y)
    time_1 <- rep(0, M)
    daytype_1 <- rep(0, M)
    location_1 <- rep(0, M)
    social_1 <- rep(0, M)

    for (i in 1:M) {
      if (sum(data.test.y[i, 1:3]) == 0) {
        time_1[i] <- 1
      }
      if (sum(data.test.y[i, 4:5]) == 0) {
        daytype_1[i] <- 1
      }
      if (sum(data.test.y[i, 6:7]) == 0) {
        location_1[i] <- 1
      }
      if (sum(data.test.y[i, 8:13]) == 0) {
        social_1[i] <- 1
      }
    }

    data.test.y <- cbind(data.test.y, time_1, daytype_1, location_1, social_1)
    #   1. The regression step on Split s
    data_out <- Pregression1(r=6, data.train.x,
                             data.train.y, data.test.x)

    LS_CW[[split + 1]] <- data_out$cw.betakap
```

```
78    LS_FICY[[split + 1]] <- data_out$ficyreg.betakap
79    LS_OLS[[split + 1]] <- data_out$ols.betakap
80    LS_RR[[split + 1]] <- data_out$rr.betakap
81
82    #    Approach: CW
83
84    data_threshold_cw <- Pregression2(data.train.y, data_out$CW.train, data_out$CW.test)
85    data_threshold_cw2 <- Pregression_enron(data.train.y, data_out$CW.train, data_out$CW.
      test)
86    data_cw_measures_q <- Pmeasures_com(data.test.y, data_threshold_cw$q)
87
88    # Hamming Loss
89    print("CW: Hamming Loss")
90    data_cw_measures_hl <- Pmeasures_com(data.test.y, data_threshold_cw$hl)
91    data_cw_measures_hl2 <- Pmeasures_com(data.test.y, data_threshold_cw2$hl)
92
93    CW_HL[1, split+1] <- data_cw_measures_q$Hloss
94    CW_HL[2, split+1] <- data_cw_measures_hl$Hloss
95    CW_HL[3, split+1] <- data_cw_measures_hl2$Hloss
96
97    # F measure
98    print("CW: F meausre")
99    data_cw_measures_f <- Pmeasures_com(data.test.y, data_threshold_cw$f)
100   data_cw_measures_f2 <- Pmeasures_com(data.test.y, data_threshold_cw2$f)
101
102   CW_F[1, split+1] <- data_cw_measures_q$F12
103   CW_F[2, split+1] <- data_cw_measures_f$F12
104   CW_F[3, split+1] <- data_cw_measures_f2$F12
105
106   # Accuracy
107   print("CW: Accuracy")
108   data_cw_measures_acc <- Pmeasures_com(data.test.y, data_threshold_cw$acc)
109   data_cw_measures_acc2 <- Pmeasures_com(data.test.y, data_threshold_cw2$acc)
110
111   CW_A[1, split+1] <- data_cw_measures_q$acc
112   CW_A[2, split+1] <- data_cw_measures_acc$acc
113   CW_A[3, split+1] <- data_cw_measures_acc2$acc
114
115   #    Approach: FICYREG
116
117   data_threshold_ficy <- Pregression2(data.train.y, data_out$FICYREG.train,
      data_out$FICYREG.test)
118   data_threshold_ficy2 <- Pregression_enron(data.train.y, data_out$FICYREG.train,
      data_out$FICYREG.test)
```

```
119     data_ficy_measures_q <- Pmeasures_com(data.test.y, data_threshold_ficy$q)
120
121     # Hamming Loss
122     print("FICY: Hamming Loss")
123     data_ficy_measures_hl <- Pmeasures_com(data.test.y, data_threshold_ficy$hl)
124     data_ficy_measures_hl2 <- Pmeasures_com(data.test.y, data_threshold_ficy2$hl)
125
126     FICY_HL[1, split+1] <- data_ficy_measures_q$Hloss
127     FICY_HL[2, split+1] <- data_ficy_measures_hl$Hloss
128     FICY_HL[3, split+1] <- data_ficy_measures_hl2$Hloss
129
130     # F measure
131     print("FICY: F measure")
132     data_ficy_measures_f <- Pmeasures_com(data.test.y, data_threshold_ficy$f)
133     data_ficy_measures_f2 <- Pmeasures_com(data.test.y, data_threshold_ficy2$f)
134
135     FICY_F[1, split+1] <- data_ficy_measures_q$F12
136     FICY_F[2, split+1] <- data_ficy_measures_f$F12
137     FICY_F[3, split+1] <- data_ficy_measures_f2$F12
138
139     # Accuracy
140     print("FICY: Accuracy")
141     data_ficy_measures_acc <-Pmeasures_com(data.test.y, data_threshold_ficy$acc)
142     data_ficy_measures_acc2 <- Pmeasures_com(data.test.y, data_threshold_ficy2$acc)
143
144     FICY_A[1, split+1] <- data_ficy_measures_q$acc
145     FICY_A[2, split+1] <- data_ficy_measures_acc$acc
146     FICY_A[3, split+1] <- data_ficy_measures_acc2$acc
147
148     #    Approach: OLS
149
150     data_threshold_ols <- Pregression2(data.train.y, data_out$ols.train, data_out$ols.
        test)
151     data_threshold_ols2 <- Pregression_enron(data.train.y, data_out$ols.train,
        data_out$ols.test)
152     data_ols_measures_q <- Pmeasures_com(data.test.y, data_threshold_ols$q)
153
154     # Hamming Loss
155     print("OLS: Hamming Loss")
156     data_ols_measures_hl <- Pmeasures_com(data.test.y, data_threshold_ols$hl)
157     data_ols_measures_hl2 <- Pmeasures_com(data.test.y, data_threshold_ols2$hl)
158
159     OLS_HL[1, split+1] <- data_ols_measures_q$Hloss
160     OLS_HL[2, split+1] <- data_ols_measures_hl$Hloss
```

```
161     OLS_HL[3, split+1] <- data_ols_measures_hl2$Hloss
162
163     # F measure
164     print("OLS: F measure")
165     data_ols_measures_f <- Pmeasures_com(data.test.y, data_threshold_ols$f)
166     data_ols_measures_f2 <- Pmeasures_com(data.test.y, data_threshold_ols2$f)
167
168     OLS_F[1, split+1] <- data_ols_measures_q$F12
169     OLS_F[2, split+1] <- data_ols_measures_f$F12
170     OLS_F[3, split+1] <- data_ols_measures_f2$F12
171
172     # Accuracy
173     print("OLS: Accuracy")
174     data_ols_measures_acc <-Pmeasures_com(data.test.y, data_threshold_ols$acc)
175     data_ols_measures_acc2 <- Pmeasures_com(data.test.y, data_threshold_ols2$acc)
176
177     OLS_A[1, split+1] <- data_ols_measures_q$acc
178     OLS_A[2, split+1] <- data_ols_measures_acc$acc
179     OLS_A[3, split+1] <- data_ficy_measures_acc2$acc
180
181     #    Approach: RR
182
183     data_threshold_rr <- Pregression2(data.train.y, data_out$RR.train, data_out$RR.test)
184     data_threshold_rr2 <- Pregression_enron(data.train.y, data_out$RR.train, data_out$RR.
        test)
185     data_rr_measures_q <- Pmeasures_com(data.test.y, data_threshold_rr$q)
186
187     # Hamming Loss
188     print("RR: Hamming Loss")
189     data_rr_measures_hl <- Pmeasures_com(data.test.y, data_threshold_rr$hl)
190     data_rr_measures_hl2 <- Pmeasures_com(data.test.y, data_threshold_rr2$hl)
191
192     RR_HL[1, split+1] <- data_rr_measures_q$Hloss
193     RR_HL[2, split+1] <- data_rr_measures_hl$Hloss
194     RR_HL[3, split+1] <- data_rr_measures_hl2$Hloss
195
196     # F measure
197     print("RR: F measure")
198     data_rr_measures_f <- Pmeasures_com(data.test.y, data_threshold_rr$f)
199     data_rr_measures_f2 <- Pmeasures_com(data.test.y, data_threshold_rr2$f)
200
201     RR_F[1, split+1] <- data_rr_measures_q$F12
202     RR_F[2, split+1] <- data_rr_measures_f$F12
203     RR_F[3, split+1] <- data_rr_measures_f2$F12
```

```
204
205     # Accuracy
206     print("RR: Accuracy")
207     data_rr_measures_acc <-Pmeasures_com(data.test.y, data_threshold_rr$acc)
208     data_rr_measures_acc2 <- Pmeasures_com(data.test.y, data_threshold_rr2$acc)
209
210     RR_A[1, split+1] <- data_rr_measures_q$acc
211     RR_A[2, split+1] <- data_rr_measures_acc$acc
212     RR_A[3, split+1] <- data_ficy_measures_acc2$acc
213 }
214 CW_HL[, 6] <- t(apply(CW_HL[, 0:5], 1, mean))
215 CW_HL[, 7] <- t(apply(CW_HL[, 0:5], 1, sd))
216 CW_F[, 6] <- t(apply(CW_F[, 0:5], 1, mean))
217 CW_F[, 7] <- t(apply(CW_F[, 0:5], 1, sd))
218 CW_A[, 6] <- t(apply(CW_A[, 0:5], 1, mean))
219 CW_A[, 7] <- t(apply(CW_A[, 0:5], 1, sd))
220 FICY_HL[, 6] <- t(apply(FICY_HL[, 0:5], 1, mean))
221 FICY_HL[, 7] <- t(apply(FICY_HL[, 0:5], 1, sd))
222 FICY_F[, 6] <- t(apply(FICY_F[, 0:5], 1, mean))
223 FICY_F[, 7] <- t(apply(FICY_F[, 0:5], 1, sd))
224 FICY_A[, 6] <- t(apply(FICY_A[, 0:5], 1, mean))
225 FICY_A[, 7] <- t(apply(FICY_A[, 0:5], 1, sd))
226 OLS_HL[, 6] <- t(apply(OLS_HL[, 0:5], 1, mean))
227 OLS_HL[, 7] <- t(apply(OLS_HL[, 0:5], 1, sd))
228 OLS_F[, 6] <- t(apply(OLS_F[, 0:5], 1, mean))
229 OLS_F[, 7] <- t(apply(OLS_F[, 0:5], 1, sd))
230 OLS_A[, 6] <- t(apply(OLS_A[, 0:5], 1, mean))
231 OLS_A[, 7] <- t(apply(OLS_A[, 0:5], 1, sd))
232 RR_HL[, 6] <- t(apply(RR_HL[, 0:5], 1, mean))
233 RR_HL[, 7] <- t(apply(RR_HL[, 0:5], 1, sd))
234 RR_F[, 6] <- t(apply(RR_F[, 0:5], 1, mean))
235 RR_F[, 7] <- t(apply(RR_F[, 0:5], 1, sd))
236 RR_A[, 6] <- t(apply(RR_A[, 0:5], 1, mean))
237 RR_A[, 7] <- t(apply(RR_A[, 0:5], 1, sd))
```

Bibliography

Adomavicius, G., Sankaranarayanan, R., Sen, S. and Tuzhilin, A. (2005). Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, vol. 23, no. 1, pp. 103–145. ISSN 1046-8188.
Available at: http://doi.acm.org/10.1145/1055709.1055714

Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P.B. (eds.), *Recommender Systems Handbook*, chap. 7. Springer. ISBN 978-0-387-85819-7.

Aggarwal, C.C. (2016). *Recommender Systems: The Textbook*. Springer.

Alqadah, F., Reddy, C.K., Hu, J. and Alqadah, H.F. (2015). Biclustering neighborhood-based collaborative filtering method for top-n recommender systems. *Knowledge and Information Systems*, vol. 44, no. 2, pp. 475–491. ISSN 0219-3116.
Available at: https://doi.org/10.1007/s10115-014-0771-x

Amazon [Online] (2017). Available at: https://www.amazon.com/gp/customer-reviews/R3JOA2LIFQIBI6/ref=cm_cr_getr_d_rvw_ttl?ie=UTF8&ASIN=B071G3X2GF.

Balabanović, M. and Shoham, Y. (1997). Fab: Content-based, collaborative recommendation. *Commun. ACM*, vol. 40, no. 3, pp. 66–72. ISSN 0001-0782.
Available at: http://doi.acm.org/10.1145/245108.245124

Baltrunas, L., Ludwig, B., Peer, S. and Ricci, F. (2012). Context relevance assessment and exploitation in mobile recommender systems. *Personal Ubiquitous Comput.*, vol. 16, no. 5, pp. 507–526. ISSN 1617-4909.
Available at: http://dx.doi.org/10.1007/s00779-011-0417-x

Baltrunas, L., Rokach, L., Kaminskas, M., Shapira, B., Ricci, F., Luke, K.-H. and Ag, D.T. (2010). Best usage context prediction for music tracks. In: *Proceedings of the 2nd Workshop on Context Aware Recommender Systems*.

Batmaz, Z., Yürekli, A., Bilge, A. and Kaleli, C. (2018). A review on deep learning for recommender systems: challenges and remedies. *Artificial Intelligence Review*, vol. 52, pp. 1–37.

Benmansour, A., Bouchachia, A. and Feham, M. (2015). Multioccupant activity recognition in pervasive smart home environments. *ACM Comput. Surv.*, vol. 48, no. 3, pp. 1–34. ISSN 0360-0300.
Available at: http://doi.acm.org/10.1145/2835372

Bennett, J., Lanning, S. and Netflix, N. (2007). The netflix prize. In: *KDD Cup and Workshop in conjunction with KDD*.

Berners-Lee, T. (1989). Information management: A proposal. *W3C*.

Bierman, S. (2019). Interpretable multi-label classification by means of multivariate linear regression. *South African Statistical Journal*, vol. 53, no. 1, pp. 1–13.
Available at: https://journals.co.za/content/journal/10520/EJC-14af67f7cb

Blondel, M., Fujino, A., Ueda, N. and Ishihata, M. (2016*a*). Higher-order factorization machines. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3359–3367. Curran Associates Inc., USA. ISBN 978-1-5108-3881-9.

Blondel, M., Ishihata, M., Fujino, A. and Ueda, N. (2016*b*). Polynomial networks and factorization machines: New insights and efficient training algorithms. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, vol. 48 of *ICML'16*, pp. 850–858. JMLR.org.

Breiman, L. and Friedman, J. (1997). Predicting multivariate responses in multiple linear regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 59, pp. 3–54.

Brown, B., Chalmers, M., Bell, M., Hall, M., MacColl, I. and Rudman, P. (2005). Sharing the square: Collaborative leisure in the city streets. In: Gellersen, H., Schmidt, K., Beaudouin-Lafon, M. and Mackay, W. (eds.), *ECSCW 2005*, pp. 427–447. Springer Netherlands, Dordrecht. ISBN 978-1-4020-4023-8.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370. ISSN 0924-1868.
Available at: http://dx.doi.org/10.1023/A:1021240730564

Burke, R.D., Felfernig, A. and Göker, M.H. (2011). Recommender systems: An overview. *AI Magazine*, vol. 32, pp. 13–18.
Available at: https://www.aaai.org/ojs/index.php/aimagazine/article/view/2361

Chang, S., Harper, F.M. and Terveen, L.G. (2016). Crowd-based personalized natural language explanations for recommendations. In: *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 175–182. ACM, New York, NY, USA. ISBN 978-1-4503-4035-9.
Available at: http://doi.acm.org/10.1145/2959100.2959153

Chong, W., Blei, D. and Li, F. (2009). Simultaneous image classification and annotation. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1903–1910. ISSN 1063-6919.

Clare, A. and King, R.D. (2001). Knowledge discovery in multi-label phenotype data. In: De Raedt, L. and Siebes, A. (eds.), *Principles of Data Mining and Knowledge Discovery*, pp. 42–53. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-44794-8.

Covington, P., Adams, J. and Sargin, E. (2016). Deep neural networks for youtube recommendations. In: *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 191–198. ACM, New

York, NY, USA. ISBN 978-1-4503-4035-9.
Available at: http://doi.acm.org/10.1145/2959100.2959190

Cremonesi, P., Koren, Y. and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pp. 39–46. New York, NY, USA.
Available at: http://doi.acm.org/10.1145/1864708.1864721

da Silva, J.F.G., de Moura Junior, N.N. and Calôba, L.P. (2018). Effects of data sparsity on recommender systems based on collaborative filtering. In: *2018 International Joint Conference on Neural Networks, IJCNN*. IEEE. ISBN 978-1-5090-6014-6.
Available at: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8465565

Deshpande, M. and Karypis, G. (2004). Item-based top-N recommendation algorithms. *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177.
Available at: https://doi.org/10.1145/963770.963776

Desrosiers, C. and Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P.B. (eds.), *Recommender Systems Handbook*, chap. 4. Springer. ISBN 978-0-387-85819-7.

Dhumal, P. and Deshmukh, S. (2016). Survey on information retrieval and pattern matching for compressed data size using the svd technique on real audio dataset. *International Journal of Computer Applications*, vol. 133, no. 16, pp. 10–12.

Drosou, M. and Pitoura, E. (2013). Ymaldb: Exploring relational databases via result-driven recommendations. *The VLDB Journal*, vol. 22, no. 6, pp. 849–874. ISSN 1066-8888.
Available at: http://dx.doi.org/10.1007/s00778-013-0311-4

Ehsan, H., Sharaf, M.A. and Chrysanthis, P.K. (2016). Muve: Efficient multi-objective view recommendation for visual data exploration. In: *IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 731–742.

Felfernig, A. and Burke, R. (2008). Constraint-based recommender systems: Technologies and research issues. In: *Proceedings of the 10th International Conference on Electronic Commerce*, ICEC '08, pp. 3:1–3:10. ACM, New York, NY, USA. ISBN 978-1-60558-075-3.
Available at: http://doi.acm.org/10.1145/1409540.1409544

Freudenthaler, C., Schmidt-Thieme, L. and Rendle, S. (2011). Bayesian factorization machines. In: *Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation.*

Ge, M., Delgado-Battenfeld, C. and Jannach, D. (2010). Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys10, pp. 257–260. ACM, New York, NY, USA. ISBN 978-1-60558-906-0.
Available at: http://doi.acm.org/10.1145/1864708.1864761

Giraldo-Forero, A.F., Jaramillo-Garzón, J.A. and Castellanos-Domínguez, C.G. (2015). Evaluation of example-based measures for multi-label classification performance. In: Ortuño, F. and Rojas, I. (eds.), *Bioinformatics and Biomedical Engineering*, pp. 557–564. Springer International Publishing, Cham. ISBN 978-3-319-16483-0.

Goldberg, D., Nichols, D., Oki, B.M. and Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, vol. 35, no. 12, pp. 61–70. ISSN 0001-0782.
Available at: http://doi.acm.org/10.1145/138859.138867

Gomez-Uribe, C.A. and Hunt, N. (2015). The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, pp. 13:1–13:19. ISSN 2158-656X.
Available at: http://doi.acm.org/10.1145/2843948

Guo, Y., Huang, M. and Lou, T. (2015). A collaborative filtering algorithm of selecting neighbors based on user profiles and target item. In: *Proceedings of the 2015 12th Web Information System and Application Conference (WISA)*, WISA '15, pp. 9–14. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4673-9372-0.
Available at: http://dx.doi.org/10.1109/WISA.2015.51

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18.

Han, E.-H.S. and Karypis, G. (2005). Feature-based recommendation system. In: *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pp. 446–452. ACM, New York, NY, USA. ISBN 1-59593-140-6.
Available at: http://doi.acm.org/10.1145/1099554.1099683

Harper, F.M. and Konstan, J.A. (2015). The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19. ISSN 2160-6455.
Available at: http://doi.acm.org/10.1145/2827872

Hastie, T., Tibshirani, R. and Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman & Hall. ISBN 1498712169, 9781498712163.

Herlocker, J.L., Konstan, J.A., Terveen, L.G. and Riedl, J.T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 5–53. ISSN 1046-8188.
Available at: http://doi.acm.org/10.1145/963770.963772

Huang, Z., Chen, H. and Zeng, D. (2004). Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering. *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 116–142. ISSN 1046-8188.
Available at: http://doi.acm.org/10.1145/963770.963775

Hwang, W., Li, S., Kim, S. and Lee, K. (2014). Data imputation using a trust network for recommendation. In: *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14 Companion, pp. 299–300. ACM, New York, NY, USA. ISBN 978-1-4503-2745-9.
Available at: http://doi.acm.org/10.1145/2567948.2577363

Hwang, W., Parc, J., Kim, S., Lee, J. and Lee, D. (2016). 'told you i didn't like it': Exploiting uninteresting items for effective collaborative filtering. In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pp. 349–360.

Hwangbo, H. and Kim, Y. (2017). An empirical study on the effect of data sparsity and data overlap on cross domain collaborative filtering performance. *Expert Syst. Appl.*, vol. 89, pp. 254–265.

Iaquinta, L., de Gemmis, M., Lops, P., Semeraro, G., Filannino, M. and Molino, P. (2008). Introducing serendipity in a content-based recommender system. In: *Eighth International Conference on Hybrid Intelligent Systems*, pp. 168–173.

Izenman, A.J. (1975). Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, vol. 5, no. 2, pp. 248–264.
Available at: http://www.sciencedirect.com/science/article/pii/0047259X75900421

Jang, M., Faloutsos, C., Kim, S., Kang, U. and Ha, J. (2016). Pin-trust: Fast trust propagation exploiting positive, implicit, and negative information. In: *Proceedings of the 2016 ACM Conference on Information and Knowledge Management*, pp. 629–38.

Jannach, D., Jugovac, M. and Lerche, L. (2016). Supporting the design of machine learning workflows with a recommendation system. *ACM Trans. Interact. Intell. Syst.*, vol. 6, no. 1. ISSN 2160-6455.
Available at: http://doi.acm.org/10.1145/2852082

Jannach, D., Zanker, M., Felfernig, A. and Friedrich, G. (2010). *Recommender Systems: An Introduction*. 1st edn. Cambridge University Press, New York, NY, USA. ISBN 0521493366, 9780521493369.

Juan, Y., Zhuang, Y., Chin, W. and Lin, C. (2016). Field-aware factorization machines for ctr prediction. In: *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 43–50. ACM, New York, NY, USA. ISBN 978-1-4503-4035-9.
Available at: http://doi.acm.org/10.1145/2959100.2959134

Kaminskas, M. and Bridge, D. (2016). Diversity, serendipity, novelty, and coverage: A survey and empirical analysis of beyond-accuracy objectives in recommender systems. *ACM Trans. Interact. Intell. Syst.*, vol. 7, no. 1, pp. 2:1–2:42. ISSN 2160-6455.
Available at: http://doi.acm.org/10.1145/2926720

Katakis, I., Tsoumakas, G. and Vlahavas, I. (2008). Multilabel text classification for automated tag suggestion. In: *Proceedings of the ECML/PKDD 2008 Discovery Challenge*.

Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pp. 426–434. ACM, New York, NY, USA. ISBN 978-1-60558-193-4.
Available at: http://doi.acm.org/10.1145/1401890.1401944

Koren, Y. and Bell, R. (2011). Advances in collaborative filtering. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P.B. (eds.), *Recommender Systems Handbook*, chap. 8. Springer. ISBN 978-0-387-85819-7.

Koren, Y., Bell, R. and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, vol. 42, no. 8, pp. 30–37. ISSN 0018-9162.
Available at: http://dx.doi.org/10.1109/MC.2009.263

Kosir, A., Odić, A., Kunaver, M., Tkalvcivc, M. and F Tasivc, J. (2011 01). Database for contextual personalization. *Elektrotehnivski vestnik*, vol. 78, pp. 270–274.

Kotkov, D., Veijalainen, J. and Wang, S. (2016). Challenges of serendipity in recommender systems. In: *Proceedings of the 12th International conference on web information systems and technologies*, vol. 2, pp. 251–256.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In: *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, ICML'95, pp. 331–339. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-377-8.
Available at: http://dl.acm.org/citation.cfm?id=3091622.3091662

Lee, D. and Seung, H. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, vol. 401, no. 6755, pp. 788–791. ISSN 0028-0836.

Lee, J., Jang, M., Lee, D., Hwang, W., Hong, J. and Kim, S. (2013). Alleviating the sparsity in collaborative filtering using crowdsourcing. In: *Workshop on Crowdsourcing and Human Computation for Recommender Systems (CrowdRec)*.

Lee, J., Sun, M. and Lebanon, G. (2012). A comparative study of collaborative filtering algorithms. *CoRR*, vol. abs/1205.3193. 1205.3193.
Available at: http://arxiv.org/abs/1205.3193

Lee, Y., Kim, S., Park, S. and Xie, X. (2018). How to impute missing ratings?: Claims, solution, and its application to collaborative filtering. In: *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pp. 783–792. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland. ISBN 978-1-4503-5639-8.
Available at: https://doi.org/10.1145/3178876.3186159

Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. In: *Proceedings of the 2005 SIAM International Conference on Data Mining*, vol. 5, pp. 471–475.

Levinas, C.A. (2014). *An Analysis of Memory Based Collaborative Filtering Recommender Systems with Improvement Proposals*. Master's thesis, Polytechnic University of Catalonia.

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

Liu, L., Lecue, F., Mehandjiev, N. and Xu, L. (2010). Using context similarity for service recommendation. In: *2010 IEEE Fourth International Conference on Semantic Computing*, ICSC '10, pp. 277–284. IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-4154-9.
Available at: http://dx.doi.org/10.1109/ICSC.2010.39

Loni, B. (2018). *Advanced Factorization Models for Recommender Systems*. Master's thesis, Delft University of Technology.

Loni, B., Larson, M., Karatzoglou, A. and Hanjalic, A. (2015). Recommendation with the right slice: Speeding up collaborative filtering with factorization machines. In: *Poster Proceedings of the 9th ACM Conference on Recommender Systems, RecSys 2015*.

Loni, B., Pagano, R., Larson, M. and Hanjalic, A. (2019). Top-n recommendation with multi-channel positive feedback using factorization machines. *ACM Trans. Inf. Syst.*, vol. 37, no. 2. ISSN 1046-8188.

Loni, B., Said, A., Larson, M. and Hanjalic, A. (2014). Free lunch enhancement for collaborative filtering with factorization machines. In: *RecSys 2014 - Proceedings of the 8th ACM Conference on Recommender Systems*, pp. 281–284.

Lops, P., de Gemmis, M. and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P.B. (eds.), *Recommender Systems Handbook*, chap. 3. Springer. ISBN 978-0-387-85819-77.

Ma, C. (2017). xlearn. https://github.com/aksnzhy/xlearn.

Ma, H., King, I. and Lyu, M.R. (2007). Effective missing data prediction for collaborative filtering. In: *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pp. 39–46. ACM, New York, NY, USA. ISBN 978-1-59593-597-7.
Available at: http://doi.acm.org/10.1145/1277741.1277751

MacKenzie, I., Meyer, C. and Noble, S. (2013). How retailers can keep up with consumers.
Available at: https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers

Madjarov, G., Kocev, D., Gjorgjevikj, D. and Deroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recogn.*, vol. 45, no. 9, pp. 3084–3104. ISSN 0031-3203.
Available at: http://dx.doi.org/10.1016/j.patcog.2012.03.004

Magnini, B. and Strapparava, C. (2001). Improving user modelling with content-based techniques. In: *Proceedings of the 8th International Conference on User Modeling 2001*, UM '01, pp. 74–83. Springer-Verlag, Berlin, Heidelberg. ISBN 3-540-42325-7.
Available at: http://dl.acm.org/citation.cfm?id=647664.733419

Mak, H., Koprinska, I. and Poon, J. (2003). Intimate: a web-based movie recommender using text categorization. In: *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pp. 602–605.

Malik, O.A. and Becker, S. (2018). Low-rank tucker decomposition of large tensors using tensorsketch. In: *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 10117–10127. Curran Associates Inc., USA.
Available at: http://dl.acm.org/citation.cfm?id=3327546.3327674

Modi, H. and Panchal, M. (2012). Experimental comparison of different problem transformation methods for multi-label classification using meka. *International Journal of Computer Applications*, vol. 59, no. 15, pp. 10–15.

Montaner, M., López, B. and de la Rosa, J.L. (2003). A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, vol. 19, no. 4, pp. 285–330.

Mooney, R.J. and Roy, L. (2000). Content-based book recommending using learning for text categorization. In: *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, pp. 195–204. ACM, New York, NY, USA. ISBN 1-58113-231-X.
Available at: http://doi.acm.org/10.1145/336597.336662

Nasierding, G., Tsoumakas, G. and Kouzani, A.Z. (2009). Clustering based multi-label classification for image annotation and retrieval. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*, pp. 4514–4519. ISSN 1062-922X.

Nguyen, T.V., Karatzoglou, A. and Baltrunas, L. (2014). Gaussian process factorization machines for context-aware recommendations. In: *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pp. 63–72. ACM, New York, NY, USA. ISBN 978-1-4503-2257-7.

Pan, J., Xu, J., Ruiz, A.L., Zhao, W., Pan, S., Sun, Y. and Lu, Q. (2018). Field-weighted factorization machines for click-through rate prediction in display advertising. In: *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pp. 1349–1357. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland. ISBN 978-1-4503-5639-8.
Available at: https://doi.org/10.1145/3178876.3186040

Plate, T.A. (1995). Holographic reduced representations. *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 623–641. ISSN 1045-9227.

Punia, A. (2018). Movielens recommender system. https://github.com/AkhilPunia/Movielens-Recommender-System.

Read, J. (2008). A pruned problem transformation method for multi-label classification. In: *In: Proc. 2008 New Zealand Computer Science Research Student Conference (NZCSRS*, pp. 143–150.

Read, J., Pfahringer, B., Holmes, G. and Frank, E. (2011). Classifier chains for multi-label classification. *Mach. Learn.*, vol. 85, no. 3, pp. 333–359. ISSN 0885-6125.
Available at: http://dx.doi.org/10.1007/s10994-011-5256-5

Rendle, S. (2010). Factorization machines. In: *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pp. 995–1000. IEEE Computer Society, Washington, DC, USA. ISBN 978-0-7695-4256-0.
Available at: http://dx.doi.org/10.1109/ICDM.2010.127

Rendle, S. (2012). Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.*, vol. 3, no. 3, pp. 57:1–57:22. ISSN 2157-6904.
Available at: http://doi.acm.org/10.1145/2168752.2168771

Rendle, S., Gantner, Z., Freudenthaler, C. and Lars, S. (2011). Fast context-aware recommendations with factorization machines. In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pp. 635–644. ACM, New York, NY, USA. ISBN 978-1-4503-0757-4.
Available at: http://doi.acm.org/10.1145/2009916.2010002

Rendle, S. and Lars, S. (2010). Pairwise interaction tensor factorization for personalized tag recommendation. In: *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pp. 81–90. ACM, New York, NY, USA. ISBN 978-1-60558-889-6.
Available at: http://doi.acm.org/10.1145/1718487.1718498

Resnick, P. and Varian, H.R. (1997). Recommender systems. *Communications of the ACM*, vol. 40, no. 3, pp. 56–58. ISSN 0001-0782.
Available at: http://doi.acm.org/10.1145/245108.245121

Ricci, F., Rokach, L. and Shapira, B. (2011). Introduction to recommender systems handbook. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P.B. (eds.), *Recommender Systems Handbook*, chap. 1. Springer. ISBN 978-0-387-85819-77.

Rivolli, A., Parker, L.C. and de Carvalho, A.C.P.L.F. (2017). Food truck recommendation using multi-label classification. In: Oliveira, E., Gama, J., Vale, Z. and Lopes Cardoso, H. (eds.), *Progress in Artificial Intelligence*, vol. 10423, pp. 585–596. Springer International Publishing. ISBN 978-3-319-65340-2.

Rivolli, A., Soares, C. and de Carvalho, A.C.P.L.F. (2018). Enhancing multilabel classification for food truck recommendation. *Expert Systems*, vol. 35, no. 4.
Available at: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.12304

Rosenberger, P. and Gerhard, D. (2018). Context-awareness in industrial applications: definition, classification and use case. *Procedia CIRP*, vol. 72, pp. 1172 – 1177. ISSN 2212-8271.
Available at: http://www.sciencedirect.com/science/article/pii/S2212827118304128

Sammut, C. and Webb, G.I. (eds.) (2010). *Latent Factor Models and Matrix Factorizations*, pp. 571–571. Springer US, Boston, MA. ISBN 978-0-387-30164-8.
Available at: https://doi.org/10.1007/978-0-387-30164-8_887

Sarwar, B.M., Karypis, G., Konstan, J.A. and Riedl, J. (2001). Application of dimensionality reduction in recommender system - a case study. In: *10th International Conference on the World Wide Web*, pp. 285–295.

Sarwar, B.M., Karypis, G., Konstan, J.A. and Riedl, J.T. (2002). Recommender systems for large-scale e-commerce : Scalable neighborhood formation using clustering. In: *5th International Conference on Computer Information Technology (ICCIT)*.

Schafer, J., Frankowski, D., Herlocker, J. and Sen, S. (2007). *Collaborative Filtering Recommender Systems*. Springer Berlin.

Shani, G. and Gunawardana, A. (2011). Advances in collaborative filtering. In: Ricci, F., Rokach, L., Shapira, B. and Kantor, P.B. (eds.), *Recommender Systems Handbook*, chap. 5. Springer. ISBN 978-0-387-85819-7.

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating “word of mouth”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pp. 210–217. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0-201-84705-1.
Available at: http://dx.doi.org/10.1145/223904.223931

Sorower, M.S. (2010). A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, vol. 18, pp. 1–25.

Spyromitros, E., Tsoumakas, G. and Vlahavas, I. (2008). An empirical study of lazy multilabel classification algorithms. In: *Proc. 5th Hellenic Conference on Artificial Intelligence (SETN 2008)*.

Srebro, N. and Jaakkola, T. (2003). Weighted low-rank approximations. In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 720–727. AAAI Press. ISBN 1-57735-189-4.
Available at: http://dl.acm.org/citation.cfm?id=3041838.3041929

Stomberg, J.C. (2014). *A Comparative Study and Evaluation of Collaborative Recommendation Systems*. Master's thesis, Michigan Technological University.

Strömqvist, Z. (2018). *Matrix factorization in recommender systems*. Master's thesis, Uppsala University.

Su, X. and Khoshgoftaar, T.M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, vol. 2009, pp. 4:2–4:2. ISSN 1687-7470.
Available at: http://dx.doi.org/10.1155/2009/421425

Sun, M., Lebanon, G. and Kidwell, P. (2011). Estimating probabilities in recommendation systems. In: Gordon, G., Dunson, D. and DudÃk, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15 of *Proceedings of Machine Learning Research*, pp. 734–742. PMLR, Fort Lauderdale, FL, USA.
Available at: http://proceedings.mlr.press/v15/sun11a.html

Tay, Y., Zhang, S., Luu, A.T., Hui, S.C., Yao, L. and Vinh, T.D.Q. (2019). Holographic factorization machines for recommendation. In: *The Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 5143–5150.

Trofimov, M. and Novikov, A. (2016). tffm: Tensorflow implementation of an arbitrary order factorization machine. https://github.com/geffy/tffm.

Trohidis, K., Tsoumakas, G., Kalliris, G. and Vlahavas, I. (2011). Multi-label classification of music by emotion. *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2011, no. 1, p. 4. ISSN 1687-4722.
Available at: https://doi.org/10.1186/1687-4722-2011-426793

Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 2007, pp. 1–13.

Tsoumakas, G., Katakis, I. and Vlahavas, I. (2010). *Mining Multi-label Data*, pp. 667–685. Springer US, Boston, MA. ISBN 978-0-387-09823-4.
Available at: https://doi.org/10.1007/978-0-387-09823-4_34

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J. and Vlahavas, I. (2011). Mulan: A java library for multi-label learning. *Journal of Machine Learning Research"*, vol. 12, pp. 2411–2414.
Available at: http://mulan.sourceforge.net/

Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In: *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, pp. 406–417. Springer-Verlag, Berlin, Heidelberg. ISBN 978-3-540-74957-8.
Available at: http://dx.doi.org/10.1007/978-3-540-74958-5_38

Turney, P.D. and Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, vol. 37, no. 1, pp. 141–188. ISSN 1076-9757.

van den Oord, A., Dieleman, S. and Schrauwen, B. (2013). Deep content-based music recommendation. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z. and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 2643–2651. Curran Associates, Inc.
Available at: http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf

Van Der Merwe, A. and Zidek, J.V. (1980). Multivariate regression analysis and canonical variates. *Canadian Journal of Statistics*, vol. 8, no. 1, pp. 27–39.
Available at: https://onlinelibrary.wiley.com/doi/abs/10.2307/3314667

Wang, Y., Chan, S.C. and Ngai, G. (2012). Applicability of demographic recommender system to tourist attractions: A case study on trip advisor. In: *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, vol. 3, pp. 97–101.

Wu, X. and Zhou, Z. (2017). A unified view of multi-label performance measures. In: *Proceedings of the 34th International Conference on Machine Learning*, vol. 70.

Xiao, J., Ye, H., He, X., Zhang, H., Wu, F. and Chua, T. (2017). Attentional factorization machines: Learning the weight of feature interactions via attention networks. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pp. 3119–3125.

Xu, W., Liu, X. and Gong, Y. (2003). Document clustering based on non-negative matrix factorization. In: *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pp. 267–273. ACM, New York, NY, USA. ISBN 1-58113-646-3.
Available at: http://doi.acm.org/10.1145/860435.860485

Xue, G., Lin, C., Yang, Q., Xi, W., Zeng, H., Yu, Y. and Chen, Z. (2005). Scalable collaborative filtering using cluster-based smoothing. In: *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pp. 114–121. ACM, New York, NY, USA. ISBN 1-59593-034-5.
Available at: http://doi.acm.org/10.1145/1076034.1076056

Yin, H., Chen, H., Sun, X., Wang, H., Wang, Y. and Nguyen, Q.V.H. (2017). Sptf: A scalable probabilistic tensor factorization model for semantic-aware behavior prediction. In: *2017 IEEE International Conference on Data Mining (ICDM)*, pp. 585–594. ISSN 2374-8486.

Yu, K., Zhu, S., Lafferty, J. and Gong, Y. (2009). Fast nonparametric matrix factorization for large-scale collaborative filtering. In: *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, pp. 211–218. ACM, New York, NY, USA. ISBN 978-1-60558-483-6.
Available at: http://doi.acm.org/10.1145/1571941.1571979

Yuan, F., Guo, G., Jose, J.M., Chen, L., Yu, H. and Zhang, W. (2017). Boostfm: Boosted factorization machines for top-n feature-based recommendation. In: *Proceedings of the 22Nd International Conference on Intelligent User Interfaces*, IUI '17, pp. 45–54. ACM, New York, NY, USA.

Zhang, M. and Zhou, Z. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1338–1351.

Zhang, M.-L. (2009 04). Ml-rbf: Rbf neural networks for multi-label learning. *Neural Processing Letters*, vol. 29, pp. 61–74.

Zhang, M.-L. and Zhou, Z.-H. (2007). Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, vol. 40, pp. 2038–2048.

Zhang, Y. and Chen, X. (2019). Explainable recommendation: A survey and new perspectives. *CoRR*.

Zhang, Z., Liu, Y. and Zhang, Z. (2018). Field-aware matrix factorization for recommender systems. *IEEE Access*, vol. 6, pp. 45690–45698.

Zheng, Y., Burke, R. and Mobasher, B. (2012). Differential context relaxation for context-aware travel recommendation. In: *E-Commerce and Web Technologies*, vol. 123, pp. 88–99. ISBN 9783642322723.

Zheng, Y., Mobasher, B. and Burke, R. (2014). Context recommendation using multi-label classification. In: *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, vol. 2, pp. 288–295. IEEE Computer Society, Washington, DC, USA. ISBN 978-1-4799-4143-8.
Available at: http://dx.doi.org/10.1109/WI-IAT.2014.110