UNIVERSITY OF SOUTHAMPTON

FACULTY OF PHYSICAL AND APPLIED SCIENCES

Electronics and Computer Science

Robust, Scalable, and Practical Algorithms for Recommender Systems

by

Mustansar Ali Ghazanfar

Thesis for the degree of Doctor of Philosophy

May 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
Electronics and Computer Science

Doctor of Philosophy

ROBUST, SCALABLE, AND PRACTICAL ALGORITHMS FOR RECOMMENDER SYSTEMS

by Mustansar Ali Ghazanfar

The purpose of *recommender systems* is to filter information unseen by a user to predict whether a user would like a given item. Making effective recommendations from a domain consisting of millions of ratings is a major research challenge in the application of machine learning and data mining. A number of approaches have been proposed to solve the recommendation problem, where the main motivation is to increase the accuracy of the recommendations while ignoring other design objectives such as scalability, sparsity and imbalanced dataset problems, cold-start problems, and long tail problems. The aim of this thesis is to develop recommendation algorithms that satisfy the aforementioned design objectives making the recommendation generation techniques applicable to a wider range of practical situations and real-world scenarios.

With this in mind, in the first half of the thesis, we propose novel hybrid recommendation algorithms that give accurate results and eliminate some of the known problems with recommender systems. More specifically, we propose a novel switching hybrid recommendation framework that combines Collaborative Filtering (CF) with a content-based filtering algorithm. Our experiments show that the performance of our algorithm is better than (or comparable to) the other hybrid recommendation approaches available in the literature. While reducing the dimensions of the dataset by Singular Value Decomposition (SVD), prior to applying CF, we discover that the SVD-based CF fails to produce reliable recommendations for some datasets. After further investigation, we find out that the SVD-based recommendations depend on the imputation methods used to approximate the missing values in the user-item rating matrix. We propose various missing value imputation methods, which exhibit much superior accuracy and performance compared to the traditional missing value imputation method - item average. Furthermore, we show how the gray-sheep users problem associated with a recommender system can effectively be solved using the K-means clustering algorithm. After analysing the effect of different centroid selection approaches and distance measures in the K-means clustering algorithm, we demonstrate how the gray-sheep users in a recommender system can be identified by treating them as an outlier problem. We demonstrate that the

performance (accuracy and coverage) of the CF-based algorithms suffers in the case of gray-sheep users. We propose a hybrid recommendation algorithm to solve the gray-sheep users problem.

In the second half of the thesis, we propose a new class of kernel mapping recommender system methods that we call KMR for solving the recommendation problem. The proposed methods find the multi-linear mapping between two vector spaces based on the structure-learning technique. We propose the user- and item-based versions of the KMR algorithms and offer various ways to combine them. We report results of an extensive evaluation conducted on five different datasets under various recommendation conditions. Our empirical study shows that the proposed algorithms offer a state-of-the-art performance and provide robust performance under all conditions. Furthermore, our algorithms are quite flexible as they can incorporate more information—ratings, demographics, features, and contextual information—easily into the forms of kernels and moreover, these kernels can be added/multiplied. We then adapt the KMR algorithm to incorporate new data incrementally. We offer a new heuristic namely KMR^{incr} that can build the model without retraining the whole model from scratch when new data are added to the recommender system, providing significant computation savings. Our final contribution involves adapting the KMR algorithms to build the model on-line. More specifically, we propose a perceptron-type algorithm namely $KMR^{percept}$ which is a novel, fast, on-line algorithm for building the model that maintains good accuracy and scales well with the data. We provide the temporal analysis of the $KMR^{percept}$ algorithm. The empirical results reveal that the performance of the $KMR^{percept}$ is comparable to the KMR, and furthermore, it overcomes some of the conventional problems with recommender systems.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Mustansar Ali Ghazanfar , declare that the thesis entitled *Robust, Scalable, and Practical Algorithms for Recommender Systems* and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- parts of this work have been published as: (Ghazanfar and Prügel-Bennett, 2010a), (Ghazanfar and Prügel-Bennett, 2010b), (Ghazanfar and Prügel-Bennett, 2010c), (Ghazanfar and Prügel-Bennett, 2010d), (Ghazanfar and Prügel-Bennett, 2010e), (Ghazanfar and Prügel-Bennett, 2011a), (Ghazanfar and Prügel-Bennett, 2011b), (Ghazanfar et al., 2011), and (Ghazanfar et al., 2012)

Signed:.....

Date:.....

Acknowledgements

First of all, I would like to express my heartfelt gratitude to Dr. Adam Prugel-Bennett for his supervision, help, support, concern, and valuable suggestions from the day I started working with him until the completion of this project. He has been a driving force behind this project. I really appreciate his assistance, guidance and sincerity, and for always being there with valuable feedback that put flesh on the bones of the work. I would also like to extend my grateful thanks to Dr. Sandor Szedmak who gave me a plethora of ideas. I am grateful to Dr. Craig Saunder who helped me start this PhD in the first place. I am also deeply indebted to Prof. Paul Lewis for providing me the valuable feedback in the transfer report, which helped shape this thesis. I am also very thankful to Prof. Steve R. Gunn and the PASCAL Network of Excellence[1] for the travel grant given to me during this time.

I am very grateful for the prayers and love of my parents who have always been a source of inspiration and help for me. I thank my family members: Mudassar, Alnasar, Mubashir, Zamurd, Shakira, Nusrat, Mazia, Rozina, Esba, Hamza, Zaeem, Hazik, and Arham for their constant support and unconditional love.

I enjoyed the company of a number of friends who were with me through the final stages of this thesis. It is hard to list all them; however, I would like to mention some of them - thanks guys. In alphabetical order: Ali, Bassam, Daisy, Darko, Fahime, Gondal, Imran, Kei, Qassem, Ramanan, Sung, Tayarani, Tayyaba, Umer, and Wei.

This work has been supported by the Instant Knowledge project by MobileVCE (the Virtual Center of Excellence in Mobile & Personal Communications, www.mobielvce.com), jointly funded by the UK Technology Strategy Boards Collaborative Research and Development Programme. This work has also been partially funded by UET (University of Engineering and Technology) Taxila, Pakistan.

[1] http://www.pascal-network.org/

Nomenclature

$\mathcal{U}$	List of users
$\mathcal{I}$	List of items
u_a	The active user
i_t	The target item
M	The total number of users in the system
N	The total number of items in the system
R	The user-item rating matrix
$\mathcal{D}$	The set of user-item pairs that have been rated
T	The total number of ratings in the system
$\mathbb{R}$	The rating scale of a recommender system
$\mathcal{I}_u$	List of items rated by u
$\mathcal{U}_i$	List of users who rated item i
$\mathcal{I}_{u_a,u_b}$	The subset of items that have been rated by both users u_a and u_b
$\mathcal{U}_{i_x,i_y}$	The subset of users that have rated both items i_x and i_y
r_{iu}	The rating given by user u on item i (for simplicity we use r_{iu})
$\check{r}_{i,u}$	The prediction for user u on item i (for simplicity we use $\check{r}_{iu}$)
$f(i,u)$	A utility function
$\check{f}(i,u)$	A model that can predict the rating for user u on item i
$\bar{r}_i$	Arithmetic mean rating for item i
$\bar{r}_u$	Arithmetic mean rating for user u
$\bar{r}$	Arithmetic overall mean rating
$\tilde{r}_i$	Geometric mean rating for item i
$\tilde{r}_u$	Geometric mean rating for user u
$\tilde{r}$	Geometric overall mean rating
$\mu_{i,u}$	$\mu_{i,u} = r_{i,u} - \overline{r}_u$
$\hat{r}_{iu}$	The (arithmetic) residual value of rank r_{iu}. It is equal to $\hat{r}_{iu} = r_{iu} - \bar{r}_i - \bar{r}_u + \bar{r}$
Sim(x, y)	The similarity between two objects x and y (can be two users or items)
l	The number of neighbouring users (or items) against an active user (or item)
A	Word-by-document matrix
$\mathfrak{D}$	Collection of document vectors
$a_{w,d}$	The weight of word w in document d
n_d	The total number of documents in a collection

n_w	The total number of words in a collection
TF	Term Frequency
IDF	Inverse Document Frequency
DF	Document Frequency
$C_1, \cdots C_z$	The number of classes (categories) for classification algorithms
$\mathcal{T}$	A target concept $\mathcal{T} : \mathfrak{D} \rightarrow C$, which maps given documents to a class
n_{sv}	The number of support vectors in the Support Vector Machine (SVM) classifier
$P(C_j\|d)$	The posterior probability in the Naive Bayes (NB) classifier
$P(C_j)$	The prior probability in the NB classifier
$P(d\|C_j)$	The likelihood in the NB classifier
$P(d)$	The evidence in the NB classifier
$\widehat{P}(C_j)$	An estimate for $P(C_j)$
$Pr(C_j)$	The posterior probability of class j computed by the NB classifier or SVM's estimated probability for class j
L	A list containing the posterior probabilities of all classes
$d(i, j)$	The absolute difference between the posterior probabilities of two classes, i and j, computed by the NB classifier (i.e. $d(i, j) = \|L(i) - L(j)\| = \|Pr(C_i) - Pr(C_j)\|$)
$\mathcal{X}$	The input space
$\mathcal{F}$	The feature space
ν	The parameter that stores the difference between the posterior probabilities of two classes, i and j (i.e. $\nu = d(i, j)$)
λ	The parameter that stores the difference between the predictions computed by the CF and a classifier
$SwitchRec_{CF}^{sup}$	Proposed switching hybrid recommender system made up of the item-based CF (IBCF) and a classifier. The superscript can be nb for the NB and svm for the SVM classifier
$SVD(A)$	Singular Value Decomposition (SVD) of matrix A
S	The singular matrix
U, V	The orthogonal matrices obtained after applying SVD
EM	The expectation maximisation algorithm
k	Represents the number of dimensions for SVD in Chapter 5 and represents the number of clusters in Chapter 6
k^*	The optimal number of dimensions for SVD
$S_{k^*}, U_{k^*}, V_{k^*}$	S, U, V matrices reduced to dimensions k^*
$neigh^*$	The optimal number of neighbours in the SVD-based CF algorithm
$error^*$	The minimum MAE observed
r'_{iu}	A rating assigned by a pseudo-user u on item i (in the SVD-based recommender system)

α, β	Parameters that measure the relative weights of the SVD-based User-Based CF (UBCF) and Item-Based CF (IBCF) in the $ImpSvd_{CF}^{hybrid}$ algorithm respectively
$\mathcal{M}$	The optimised SVD model
ϑ	A threshold parameter to terminate the EM algorithm
Θ_{sparse}	A parameter that shows when to do imputation
Θ_{dense}	A parameter that shows how much imputation is enough
$ImpSvd$	Imputed SVD algorithm (SVD enhanced by an imputation method)
$ItrSvd$	Iterative SVD algorithm (SVD combined with the EM algorithm)
$ImpSvd_{CF}$	The SVD-based CF
$ImpSvd_{CF}^{hybrid}$	A hybrid algorithm that linearly combines the SVD-based UBCF and SVD-based IBCF
u_p	The user, which has rated the maximum number of items
i_p	The item, which has been rated by the maximum number of users
dist(u)	A variable representing the shortest distance from a user u to the closest centroid we have already chosen
c_j	A centroid of K-means clustering algorithm
$\mathcal{D}_{c_j}$	The set of user-item rating pairs in a cluster represented with centroid c_j
r_{i,c_j}	The rating given on item i by centroid c_j
g_j	A cluster with centroid c_j
G	Total number of clusters ($G = \{g_1, \cdots, g_k\}$)
TotalSim	The within-cluster total similarity
$\mathbb{P}(u)$	The number of ratings provided by a user, normalised by the number of ratings given by u_p
ω	The similarity threshold to detect the gray-sheep users
itr	The maximum number of iteration in the K-means clustering algorithm
pow_{thr}	Threshold parameter to detect the power users in the system
Υ	A parameter that linearly combines the CF with the Content-Based Filtering (CBF) for the gray-sheep users
KMR	Kernel Mapping Recommender system algorithms
KMR_{ib}	The item-based version of the KMR algorithms
KMR_{ub}	The user-based version of the KMR algorithms
KMR_{hybrid}^{sup}	A hybrid recommender system made up of KMR_{ib} and KMR_{ub}. The subscript can be *linear*, *cnt*, and *var* representing the different types of hybrid KMR as defined in Chapter 7
θ_{sub}	Parameter to combine the user- and item-based KMR in the case of KMR_{hybrid}^{sup}. The subscript can be *linear*, *cnt*, and *var*.

KMR^{sup}_{sub}	Different variants of the KMR algorithms. The subscript can be the item-based (ib), the user-based (ub), demographic (D), and feature (F). The superfix can be M^4 representing the corresponding version of the KMR algorithms, where we take into account the max, mean, mode, and median of the output probability distribution.
$\check{r}^{sup}_{i,u}$	The prediction for user u on item i by an algorithm. The subscript represents the algorithm (e.g. ib for the item-based CF, nb for the Naive Bayes classifier etc.)
$\mathbf{q}_i$	The information about item i
$\mathbf{q}_u$	The information about user u
$\mathcal{H}_r$	Feature space of object class $\mathcal{X}_r$
ϕ_x, ψ_x	Functions mapping object $\mathcal{X}_x$ into $\mathcal{H}_x$
$\boldsymbol{\psi}(\hat{r}_{iu})$	The residue ranks (ratings) information mapped in some Hilbert space ($\mathcal{H}$)
$\boldsymbol{\phi}(\boldsymbol{q}_i)$	The item information, $\boldsymbol{q}_i$, mapped in some Hilbert space ($\mathcal{H}$)
$\mathcal{N}(x\|\hat{r}, \sigma)$	Normal distribution with mean $\hat{r}$ and variance σ^2
K_{sub}	A Kernel function. The subscript can be rat for rating, $feat$ for feature and $demo$ for demographic kernel
$K_{\boldsymbol{q}}$	The input feature kernel
$K_{\hat{r}}$	The residual kernel
$\mathbf{W}$	Linear mapping
$\mathcal{L}(f)$	Lagrangian of a function
α_{iu}	Lagrangian multiplier. The design variable to be updated and optimised in the algorithm, for all $i \in \mathcal{I}, u \in \mathcal{U}$
λ_i	Lagrangian multiplier
ξ_i	Slack variable belonging to movie i
$\frac{\partial f(x,y,z)}{\partial x}$	Partial derivative of a function w.r.t variable x
C	Penalty parameter
σ	Variance parameter
$\otimes$	Tensor product
$\langle\ \rangle$	Inner product
$\|\ \|_F$	Frobenius norm
KMR^{incr}	The incremental version of the KMR algorithms
KMR^{percet}	The perceptron-type (on-line) version of the KMR algorithms
t	Time counter of the observed rank items
$(i, u, r_{iu})_t$	The tuple of an observation at time t ($i(t)$,$u(t)$ and $r_{iu}(t)$ denote the components)
$\mathcal{I}(t)$	The set of all movies observed by time t
$\mathcal{U}(t)$	The set of all users observed by time t
$\mathcal{R}(t)$	The set of all ranks arrived by time t. It is equal to $\{r_{iu}\|i \in \mathcal{I}(t), u \in \mathcal{U}(t)\}$
$\mathcal{I}_u(t)$	The set of movies observed by time t, which have been by user u
$\mathcal{U}_i(t)$	The set of users observed by time t, which have seen movie i

$E(t)$	The overall average rating at time t
$E_i(t)$	The average rating of item i at time t
$E_u(t)$	The average rating of user u at time t
$E_\alpha(t)$	The design variables's average at time t
$\hat{r}_{iu}(t)$	The residual value of rank r_{iu} at time t and is equal to $\hat{r}_{iu}(t) = r_{iu}(t) - E_i(t) - E_u(t) + E(t)$.
$\alpha_{iu}(t)$	The value of a design variable α_{iu} at time t
$\xi_i(t)$	The value of slack variable, ξ_i, at time t
β	The discounting factor. It is in $(0, 1)$
s	The step function. It is in $(0, 1)$
β_k	The accumulated discounting normaliser up to k updates
t_k	Time at update k
MAE	Mean Absolute Error
NMAE	Normalized Mean Absolute Error
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristic
KNN	K-Nearest Neighbours
IMDB	Internet Movie Database
CF	Collaborative Filtering
CBF	Content-Based Filtering
KB	Knowledge-Based
DM	Demographic-Based
RS	Recommender System
1V1	One Versus One
1VR	One Versus Rest
DAG	Directed Acyclic Graph
UBCF	The User-Based CF
IBCF	The Item-Based CF
RBF	Radial Basis Function
PCC	The Pearson Correlation
PCCDV	The Pearson Correlation with Default Votes
VS	The Vector Similarity
VSDV	The Vector similarity with Default Votes
w.r.t.	With Respect To

To my parents, Ch. Ghazanfar Ali and Sughra Begum, for their love, affection, and prayers!

Chapter 1

Introduction

This chapter discusses the problem under investigation in this thesis, the motivations and the design objectives, the contributions made to the field, and outlines the structure of the remaining chapters.

1.1 Recommender Systems

> "*Every day, approximately 20 million words of technical information are recorded. A reader capable of reading 1000 words per minute would require 1.5 months, reading eight hours every day, to get through one day's output, and at the end of that period would have fallen 5.5 years behind in his reading*" (Murray, 1966).

There has been an exponential increase in the volume of available digital information (e.g. videos in Youtube (youtube.com) and Netflix (netflix.com), music in LastFm (last.fm)), electronic resources (e.g. research papers in CiteULike (citeulike.org)), and online services (e.g. Flicker (flickr.com), Delicious (delicious.com), Amazon (amazon.com)) in recent years. This information overload has created a potential problem, which is how to filter and efficiently deliver relevant information to a user. Furthermore, information needs to be prioritised for a user rather than just filtering the right information; otherwise, it could become overwhelming. Search engines help Internet users by filtering pages to match explicit queries, but it is very difficult to specify what a user wants by using simple keywords. The Semantic Web also provides some help to find useful information by allowing intelligent search queries; however, it depends on the extent to which the web pages are annotated. These problems highlight a need for information filtering systems that can filter unseen information and can predict whether a user would like a given resource. Such systems are called *recommender systems*, and they mitigate the aforementioned problems to a great extent.

1.2 Research Objectives

Recommender systems have been a very active topic of research for around twenty years. This, in part, has been spurred on by the Netflix prize competition (Bennett and Lanning, 2007) to improve the performance of a baseline algorithm by 10%. A number of approaches have been proposed to solve the recommendation problem including: content-based filtering, Ontology-based approaches, supervised classification techniques, unsupervised clustering techniques, memory-based Collaborative Filtering (CF), model-based approaches spanning a number of algorithms such as Singular Value Decomposition (SVD), Matrix Factorisation (MF) techniques, and principal component analysis, all of which suffer from some problems (see below) in one way or the other. By careful examination of the literature, we find that the current state-of-the-art algorithms (especially the ones proposed in the Netflix prize competition) attain an increased accuracy rate by using a specific dataset's peculiar characteristics or by blending dozens (or hundreds) of matrix factorisation-based predictors trained on a static dataset. Although, such systems are interesting, they are not very flexible, practical or ideal for the real world applications. The reason is that the recommendation generation is a complex process and the quality of a recommendation algorithm depends on a number of factors. For instance, the accuracy of a recommendation algorithm might be very good given a dense dataset and it may suffer under sparse settings. There are a number of design objectives to be satisfied in order to make a recommendation algorithm to be effectively used in real world scenarios as follows:

1. *Accurate:* An algorithm should be able to provide accurate recommendations for a user. If a user trusts and leverages a recommender system, and then discovers that they are unable to find what they want then it is unlikely that they will continue with that system. Consequently, the most important task for a recommender system is to accurately predict the rating of the non-rated user-item combination and recommend items based on these predictions.

2. *Robustness with sparsity:* The performance of an algorithm should not degrade badly with sparsity. In many commercial recommender systems like Amazon, it is not unusual, even for an active user, to provide ratings for well under 1% of all the available items. Besides, an increase in the number of items in the database will decrease the density of each user with these items. Furthermore, most of the recommender systems have imbalanced data, i.e. a user may have provided one rating and others may have provided hundreds of ratings and the same is true for items as well, which might result in a performance bottleneck.

3. *Long tail problems:* Newly introduced or unpopular items having only a few ratings can create a potential problem for a recommender system. Many recommender systems, for example, the CF ones, ignore these items or cannot produce reliable

recommendations for these items. This problem is called the long tail problem (Park and Tuzhilin, 2008). As the majority of the items in a recommender system generally falls into this category (Park and Tuzhilin, 2008), there is a need to develop algorithms which can filter, personalise, and accurately recommend from the huge amount of items available in the long tail.

4. *Cold-start scenarios:* Generally, while testing recommender systems, a dataset is used where some sets of ratings are treated as unseen while the other ratings are used for learning. The unseen data are then used to test the performance of the algorithm. To obtain accurate results, datasets are usually selected with users that have made a relatively high number of ratings. However, in real applications, the datasets are often highly skewed; for example, a large number of users may have made only a small number of ratings and a large number of items may have received very few ratings. These are important scenarios in practical systems as making reasonable recommendations to new users can be crucial in attracting more users. There are two important cold-start scenarios as described below (Schein et al., 2002):

   - *New user cold-start problem:* When a new user enters the system, initially the system does not have enough data for that user, and hence the quality of the recommendations would suffer, a potential problem called the new user cold-start problem (Adomavicius and Tuzhilin, 2005).
   - *New item cold-start problem:* When a new item is added to a system, then initially it is not possible to get a rating for that item from a significant number of users, and consequently the CF recommender systems would not be able to recommend that item effectively. This problem is called the new item cold-start problem (Adomavicius and Tuzhilin, 2005).

   Often, recommendation algorithms that have been optimised to give good recommendations on dense datasets perform poorly under these scenarios. Hence, there is a need to devise algorithms that give sensible and accurate recommendations under these scenarios.

5. *Scalable:* An algorithm should be designed to scale well with large datasets. Since the search space grows very quickly as the number of users and items increases, it is extremely difficult to perform an exhaustive search (a brute-force search) where every possible candidate for the solution is examined (e.g. given 4000 users and 4000 items, the number of comparisons required to produce recommendations in case of the conventional CF algorithm (Shardanand and Maes, 1995) are $4000^2(4000) = 64 \times 10^9$). It must be noted that there is a conflicting trade-off between the accuracy and scalability of a system.

6. *Practical:* The recommendation algorithm should be practical—producing recommendation in real-time with minimum off-line and on-line cost. With the current

state-of-the-art algorithms proposed in the Netflix prize competition (Bennett and Lanning, 2007), we consider the recommendation generation process to be theoretically applicable, but practically infeasible, because the final solution is generated by blending dozens (or even hundreds) of recommendation algorithms. For instance; the winner of the Netflix prize team states that: "*Moreover, the solution is based on a huge amount of models and predictors which would not be practical as part of a commercial recommender system. However, this result is a direct consequence of the nature and goal of the competition: obtain the highest possible accuracy at any cost, disregarding completely the complexity of the solution and the execution performance*" (Piotte and Chabbert, 2009).

7. *Flexible:* It is desirable for an algorithm to be flexible, i.e. it should be able to easily incorporate additional information—ratings, demographics, features, and context information—when available. It must be noted that the complexity of an algorithm must not increase significantly when given more information.

8. *Incorporating new users and items efficiently:* Since most of the recommender system domains are dynamic—data (new users and items) are being added continuously in the system—an algorithm must be able to incorporate new users and items effectively. It must be noted that retraining the whole model from scratch (given millions of ratings) upon the arrival of new data is not pragmatic due to the tremendous cost related to the execution time and memory required.

9. *On-line model building:* Since most of the recommender system datasets are huge, typical batch processing algorithms that require multiple passes through the datasets are not ideal for this situation. Hence, the on-line model building algorithms are called for.

10. *Maximum coverage:* The coverage of an algorithm should be maximum, i.e. it should be able to make recommendations for all the existing items and the coverage should not degrade under cold-start and sparse scenarios.

11. *Gray-sheep users problem:* In the CF domain, the gray-sheep users—users who partially agree/disagree with other users and have low correlation coefficients with almost all users—pose some potential problems. It is desirable for an algorithm to detect and satisfy their needs, as they might not get useful recommendations from the opinions of the user community.

12. *Overcome conventional problems:* It should overcome conventional problems with the recommender systems, such as *stability vs. plasticity* (Burke, 2002), *confidence* in a prediction (Mcnee et al., 2003), and *over-specialisation* (Burke, 2002) problems.

Therefore, by developing recommendation algorithms that satisfy the above mentioned design objectives, we aim at making recommender system techniques applicable to a wider range of practical situations and real-world scenarios.

1.3 Research Contributions

Against the aforementioned research objectives, we first propose hybrid recommender system algorithms (Chapters 4, 5, and 6) that can be used to make reliable recommendations under different conditions. We then propose a new class of recommender system algorithm based on the structure-learning technique (Chapters 7 and 8) that gives state-of-the-art performance on high dimensional datasets.

This thesis makes significant contributions to the state-of-the-art recommender system algorithms in the following ways:

1. We propose a novel *switching hybrid* recommender system framework using Collaborative Filtering (CF) and classification approaches (Naive Bayes and support vector machines classifiers) trained on the content profiles of users. We show empirically that the proposed hybrid recommender system gives more accurate results than the conventional hybrid recommender systems and the individual ones and, moreover it helps in overcoming some of the recorded problems with the recommender systems.

2. We propose an imputed version of the Singular Value Decomposition (SVD)-based recommender systems, to overcome the sparsity and other problems associated with recommender systems. We show how a careful selection of imputation methods in the SVD-based recommender systems can provide potential benefits ranging from cost saving to performance enhancement. The proposed missing value imputation methods have the ability to exploit any underlying correlations in the data and are proven to exhibit superior performance compared to the traditional missing value imputation strategy—item average—that has been the preferred approach in the literature to resolve this problem. Furthermore, we show that the convergence and accuracy of the traditional approach suffer when SVD is combined with the Expectation Maximisation (EM) algorithm. Finally, we present a trade-off between the accuracy and scalability by showing when and how much imputation is required.

3. We propose various centroid selection approaches and distance measures for the K-means clustering algorithm. We employ the best approach to partition the recommender system dataset into clusters and offer a simple strategy that separates the so-called gray-sheep users into a distinct group. We show that the performance of the CF-based algorithms suffer in the case of gray-sheep users. We offer a hybrid

recommendation algorithm to overcome the gray-sheep users problem. To the best of our knowledge, this is the first attempt to solve the gray-sheep users problem associated with a recommender system.

4. We propose a new class of Kernel Mapping Recommender (KMR) system methods for solving the recommendation problem[1]. The proposed methods are based on a structure-learning technique, where the main idea is to exploit the relationship between two data sources that can be expressed by real-value functions. These functions can be approximated by multi-linear functions and the missing values for items can be estimated by turning this problem into a convex one-class classification problem. The data sources are represented in the inner-product space, and hence the flexibility of the kernel-based learning can be employed. The estimation of the inference is based on the well-known maximum margin principle (Joachims, 2006). We propose the user- and item-based versions of the KMR and offer various ways to combine them. We empirically show on five different datasets that the proposed algorithms outperform (or give comparable results to) the state-of-the-art algorithms. We demonstrate that the proposed algorithms maintain robust performance under cold-start, long tail, sparse, skewed, and imbalanced datasets. Furthermore, they are quite flexible: (1) more information—ratings, demographics, features, and contextual information—can be added in the forms of kernels, (2) the residues in the ratings can be mapped onto a density function that helps to overcome the cold-start and imbalanced dataset problems, and (3) they can switch between the user- and item-based versions depending on the reliability of the predictions as measured by the uncertainty in the prediction of the algorithms.

5. The KMR algorithms are batch processing algorithms, so they cannot accommodate the incremental update with the arrival of new data, thus making them unsuitable for the dynamic environments. From this line of research, we propose a heuristic method that we call KMR^{incr}, which can effectively incorporate new data, providing significant computation savings compared to the case where we retrain the model from scratch upon the arrival of new training data. Finally, we introduce an on-line version of the KMR algorithms that we call $KMR^{percept}$, which is a novel, fast incremental method for building the model that maintains a good level of accuracy and scales well with the data. The proposed algorithm implements an incremental subgradient descent step. The implemented version of the algorithm follows the dual perceptron schema where only the knowledge of the corresponding kernels is required. We present a simple solution to eliminate the stability vs. plasticity problem associated with a recommender system. We

[1]This work is done in collaboration with Dr. Szedmak Sandor, when he was a research fellow in the School of Electronics and Computer Science, University of Southampton UK. He has proposed a general framework to handle missing data sources. We applied this framework to the recommender system's domain after some modifications and presented various refinements and improvements.

also provide the temporal analysis of the proposed algorithm and show that it can effectively incorporate new data when available.

1.4 Thesis Outline

This thesis has been organised as follows:

Chapter 2 gives an overview of recommender systems and existing algorithms. It describes the taxonomy of the existing recommendation algorithms and highlights their advantages and disadvantages. It also describes the limitations of the current state-of-the-art algorithms.

Chapter 3 describes the datasets and metrics used in this work. It also sheds light on the feature extraction and selection algorithms that have been used in this work. Moreover, it demonstrates how the content-based and demographic recommender systems are built.

Chapter 4 introduces and evaluates a switching hybrid framework that combines Collaborative Filtering (CF) with the content-based filtering algorithm. After giving the details of the existing hybrid recommender systems, it proposes a simple strategy that can be used effectively to combine the individual systems.

Chapter 5 presents an imputed version of the Singular Value Decomposition (SVD)-based recommender systems. It shows by thorough evaluation that the baseline imputation strategy—item average—fails to produce effective recommendations under cold-start, long tail, and sparse settings. It proposes various imputation strategies that can effectively be used under all scenarios.

Chapter 6 proposes a clustering algorithm to detect the gray-sheep users. After describing extensive experiments conducted to improve the quality of K-means clustering algorithms using different centroids selections approaches and distance measures, it shows that the CF algorithms fail to produce effective and accurate recommendations for the gray-sheep users. It proposes a hybrid recommender system to provide effective recommendations for the gray-sheep users.

Chapter 7 introduces a new class of Kernel Mapping Recommender (KMR) system algorithms for solving the recommendation problem. It shows that the proposed algorithms outperform (or give comparable performance to) the state-of-the-art algorithms. Furthermore, it shows that the proposed algorithms are quite flexible and produce reliable recommendations under all recommender system scenarios.

Chapter 8, the final contributory chapter, presents a heuristic method based on the KMR algorithms that we call KMR^{incr} that can update the model effectively on the arrival of new data. Furthermore, it proposes an on-line version of the KMR algorithms,

namely $KMR^{percept}$, that can build the model by sequentially processing one data point at a time.

Chapter 9 concludes the thesis by highlighting the most significant contributions and outlines the directions for future research.

1.5 Publications

In this section, we outline the papers that have been peer reviewed and published in support of these contributions:

1. Ghazanfar M.A. and Prügel-Bennett A. (2010). An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering. *In Lecture Notes in Engineering and Computer Science: Proceedings of The International Multi Conference of Engineers and Computer Scientists 2010, (pp. 493502). IMECS 2010*, 1719 March, 2010, Hong Kong.

2. Ghazanfar M.A. and Prügel-Bennett A. (2010). Building Switching Hybrid Recommender System Using Machine Learning Classifiers and Collaborative Filtering. In *IAENG International Journal of Computer Science,* 37(3), 272 287.

3. Ghazanfar M.A. and Prügel-Bennett A. (2011). Fulfilling the Needs of Gray-Sheep Users in Recommender Systems, A Clustering Solution. *In 2011 International Conference on Information Systems and Computational Intelligence.*

4. Ghazanfar M.A. and Prügel-Bennett A. (2011). The Advantage of Careful Imputation Sources in Sparse Data-Environment of Recommender Systems: Generating Improved SVD-based Recommendations. In *IADIS European Conference on Data Mining,* 24-26 July 2011, Rome Italy 2011. (**Was granted the best student paper award**)

5. Ghazanfar M.A., Szedmak S., and Prügel-Bennett A. (2011). Incremental Kernel Mapping Algorithms for Scalable Recommender Systems, *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Special Session on Recommender Systems in e-Commerce (RSEC)*, Nov 2011, USA.

6. Ghazanfar M.A., Prügel-Bennett A., and Szedmak S. (2011). Kernel Mapping Recommender System Algorithms, *In Information Sciences Journal*, Dec 2011 (**Accepted**).

The following paper is under review (**going through second round**):

1. Ghazanfar M.A. and Prügel-Bennett A. (2011). The Advantage of Careful Imputation Sources in Sparse Data-Environment of Recommender Systems: Generating Improved SVD-based Recommendations, In *Informatica Journal*, Jan 2011.

Furthermore, during this time period, we have completed some other publications, relating to the broad area of recommender systems (hybrid recommender systems and distributed recommender systems); however, they have not been addressed by this thesis:

1. Ghazanfar M.A. and Prügel-Bennett A. (2010). A Scalable, Accurate Hybrid Recommender System. *In The 3rd International Conference on Knowledge Discovery and Data Mining (WKDD 2010). IEEE,* 910 January, 2010, Thailand.

2. Ghazanfar M.A. and Prügel-Bennett A. (2010). Novel Significance Weighting Schemes for Collaborative Filtering: Generating Improved Recommendations in Sparse Environments. In *DMIN10, the 2010 International Conference on Data Mining. WORLDCOMP10, 1215 July, 2010, USA.*

3. Ghazanfar M.A. and Prügel-Bennett A. (2010). Novel Heuristics for Coalition Structure Generation in Multi-Agent Systems, In *The 2010 International Conference of Computational Intelligence and Intelligent Systems, ICCIIS10,* 30 June2 July 2010, London, U.K., 2010.

1.6 Summary

In this chapter, after defining the problem statement, we present various design objectives that have laid the foundations of our work. We present various contributions of the thesis and a list of publications in support of these contributions, and outline the content of the following chapters.

Chapter 2

Recommender Systems: Background and Existing Algorithms

In this chapter, we discuss background information about recommender systems. After giving some definitions and examples of application, we formalise the recommendation problem. Then we discuss how users' and items' profiles can be defined. After that, we discuss various types of recommender systems, starting from the most widely used collaborative filtering and content-based filtering systems that this work (mainly) focuses on, and continue to describe different types that have been used in the literature. We illustrate how these approaches differ from each other, bring to light their merits and drawbacks, and describe some of their applications. At the end, we describe the state-of-the-art recommendation algorithms and the algorithms we have used to benchmark our results.

2.1 What are Recommender Systems?

Recommender systems are information filtering systems, which suggest interesting resources[1] (i.e. movies, books, music, people, etc.) to users based on their preferences—what they like or dislike about a particular resource—with the goal that these resources are likely to be of interest to users. They process the historical data about users' preferences using machine learning algorithms and learn a model that can compile a ranked list of all resources available for recommendation for each user based on the information encoded in their profile. The highly ranked resources are then recommended to the corresponding user based on the rationale that these resources are most likely to be consumed next by this user.

[1]The terms resource and item are used interchangeably in this work.

Nowadays, a number of recommender systems have been built that help people to find useful resources, spanning a number of areas such as movies (MovieLens (movielens.org), Netflix (netflix.com), FilmTrust (trust.mindswap.org/FilmTrust), Moviefinder (moviefinderonline.com), reel.com); music (CDNOW (CDNOW.com), Ringo (ringo.com), LastFm (last.fm), Pandora (pandora.com)); pictures (flickr.com); e-commerce (Amazon (amazon.com), Ebay (ebay.com), Dietorecs (Dietorecs.com), choicestream.com, Entree (Burke, 2002)); expertise finder (Referral Web (referralweb.net), Linkedin (linkedin.com)); news filtering (GroupLens (Konstan et al., 1997), PHOAKS (Terveen et al., 1997), P-Tango (Claypool et al., 1999), Google news (Das et al., 2007)); email filtering (Tapestry (Goldberg et al., 1992)); web (citeseer (citeseerx.ist.psu.edu), Fab (Balabanović and Shoham, 1997), QuickStep (Middleton, 2002), Foxtrot (Middleton, 2002)); books (whichbook.net, WhatShouldIReadNext.com, librarything.com, Libra (Mooney and Roy, 2000)); electronic program guides (Barragáns-Martínez et al., 2010); and holidays and travel (tripadvisor.co.uk).

Recommender systems are now considered a salient part of any modern e-commerce system because they help increase the e-commerce systems sales by making useful recommendations—items a customer/user would be most likely to consume. The statement, given by Greg Linden, who implemented the first recommendation system for Amazon, shows how the recommender systems help industry to make profits:

> "*(Amazon.com) recommendations generated a couple orders of magnitude more sales than just showing top sellers*"[2]

Next, we set out the formalisation of the recommendation problem.

2.2 Formalisation of the Recommendation Problem

A Recommender System (RS) consists of two basic entities: users and items, where users provide their opinions (ratings) about items. We denote these users by $\mathcal{U} = \{u_1, u_2, \cdots, u_M\}$, where the number of users using the system is $|\mathcal{U}| = M$, and denote the set of items being recommended by $\mathcal{I} = \{ i_1,i_2, \cdots, i_N\}$, with $|\mathcal{I}| = N$. We can represent each element of user space $\mathcal{U}$ and item space $\mathcal{I}$ with a profile. We usually represent a user's profile by defining their characteristics like age, gender, geographical location, etc.; however, in simple cases we represent it by a unique user Identifier (ID). Similarly, we represent each item by defining some characteristic; for example in a book recommender system, each book can be represented by author, topic, year of release, etc.

Recommender systems store the history of the user's interactions with the system; for example, user purchase history, types of items they purchase together, their ratings, etc.

[2] http://glinden.blogspot.com/2007/05/google-news-personalization-paper.html

Most of the recommender systems require users to rate some item, in order to recommend unknown items; for example, in the MovieLens movie recommender system, when a new user registers they have to rate some movies in order to get proper recommendations from the system. The users will have given ratings of some but not all of the items. We denote these ratings by $(r_{i,u}|(i,u) \in \mathcal{D})$, where $\mathcal{D} \subset \mathcal{I} \times \mathcal{U}$ is the set of user-item pairs that have been rated. We denote the total number of ratings made by $|\mathcal{D}| = T$. Typically each user rates only a small number of the possible items, so that $|\mathcal{D}| = T \ll |\mathcal{I} \times \mathcal{U}| = N \times M$. It is not unusual in practical systems to have $T/(N \times M) \cong 0.01$. The set of possible ratings made by users can be thought of as elements of an $M \times N$ rating matrix R. This matrix is called the user-item rating matrix, an example of which is shown in Table 2.1.

	Troy (Action)	The Godfather (Crime)	Titanic (Romantic)	Forrest Gump (Comedy)
Fahime	5	⊘	5	1
Musi	5	⊘	⊘	1
Hamza	4	4	5	1
Paul	4	⊘	5	5
Adam	1	2	⊘	5

Table 2.1: Example: a subset of the user-item rating matrix in a movie recommender system. We have five users (rows) and four movies (columns). The case where a user has not rated a particular movie is shown by the ⊘ symbol. The rating scale consisting of integer values between 1 and 5, captures the extreme likes (5) and extreme dislikes (1) behaviour of a user.

We denote the items for which there are ratings by user u as $\mathcal{I}_u$ (i.e. $\mathcal{I}_u \subseteq \{\mathcal{I}|\forall_{i \in \mathcal{I}} r_{i,u} \neq \oslash\}$), and the users who have rated an item i by $\mathcal{U}_i$ (i.e. $\mathcal{U}_i \subseteq \{\mathcal{U}|\forall_{u \in \mathcal{U}} r_{i,u} \neq \oslash\}$). We use the term $\mathcal{I}_{u_a,u_b}$ to denote the subset of items that have been rated by both users u_a and u_b (i.e. $\mathcal{I}_{u_a,u_b} = \mathcal{I}_{u_a} \cap \mathcal{I}_{u_b}$). Likewise, the term $\mathcal{U}_{i_x,i_y}$ denotes the subset of users that have rated both items i_x and i_y (i.e. $\mathcal{U}_{i_x,i_y} = \mathcal{U}_{i_x} \cap \mathcal{U}_{i_y}$).

Let f be a utility function that measures the utility of item i to user u, i.e.

$$f : \mathcal{I} \times \mathcal{U} \rightarrow \mathbb{R},$$

where $\mathbb{R}$ is a totally ordered set. Now for each user $u \in \mathcal{U}$, the aim of a recommender system is to choose certain items $I_u \in \mathcal{I}$ that maximise the user's utility (Adomavicius and Tuzhilin, 2005). We can specify this as follows:

$$I_u = \arg\max_{i \in \mathcal{I} \setminus \mathcal{I}_u} f(i,u) \; \forall_{u \in \mathcal{U}},$$

where the utility of an item is application-dependent; for example in a movie recommender system, this can be represented by a rating in some numeric scale that indicates to what extent a particular user liked a specific movie.

Typically, the utility, f, is defined on a subset of $\mathcal{I} \times \mathcal{U}$ and not on the whole space. For instance, in the case of a movie recommender system, the utility is defined over items previously rated by the users. The task of the recommender systems then becomes to extrapolate utility, f, to the whole space $\mathcal{I} \times \mathcal{U}$ in order to make recommendations. There are different ways to extrapolate the utility function over the whole $\mathcal{I} \times \mathcal{U}$ space (Adomavicius and Tuzhilin, 2005). In the simplest case, utility function can be defined by specifying some heuristics and its performance can be validated empirically. Alternatively, utility function can be estimated by optimising certain performance criteria, such as the mean absolute error.

The utility function, f, essentially explains the mapping of a user $u \in \mathcal{U}$ and an item $i \in \mathcal{I}$ to a rating $r_{i,u}$, i.e. $f(i,u) = r_{i,u}$. This mapping can be estimated by a model $\check{f}$ that predicts the rating of a non-rated user-item combination. Formally:

$$\check{f}(i,u) = f(i,u) + e \; \forall_{i \in \mathcal{I}\, u \in \mathcal{U}},$$

where e is a small error between prediction made by the model and the actual rating assigned by a user. Once a model has been built, the unknown ratings can be predicted and recommended to users. We can use machine learning algorithms, approximation theory, and some heuristics for prediction. Next, we discuss how users' and items' profiles can be built.

2.3 Users' and Items' Profiles

The main building elements of the recommender systems, i.e. users and items, need to be modelled in such a way that recommendation algorithms can exploit them. Recommender systems usually get initial information about users when they first register with the system. The simplest way is to create an empty user's profile, which is updated as the system gathers the user's feedback. This method, however, would not be able to recommend any items unless it gathers some information about the user's preferences. An alternative approach is where the user manually creates a profile. The user might need to give their interests (e.g. types of domain they are interested in), demographic (e.g. age, genre, etc.) information, and geographical (e.g. country) information. Another approach, used by the MovieLens video recommender system and iLike music recommender system (ilike.com), requires user to provide ratings on a predefined set of items. For example, when a new user registers with the iLike web-site, the system presents them a list of artists they need to rate before getting the recommendations.

After getting the initial information, the system maintains the user's profile, as they provide feedback. The feedback can be explicit or implicit (Mobasher, 2007). Explicit feedback, where the user provides their opinions about certain items, can be positive or negative and usually comes in the forms of ratings. Rating scales can be discrete

(real values from 0 to N) or binary (likes and dislikes of a user), although most of the recommender systems use discrete scales. Explicit feedback can also be gathered by allowing users to write comments and opinions about certain items. In implicit feedback the user's interaction with the item is observed; for example, web usage mining (e.g. time spent in a web page), analysing the listening/watching habits in media player (e.g. in YouTube the system might store how a user plays, re-plays, skips, and stops videos), and observing the history of the transactions in the e-commerce website (e.g. items purchased or returned by a user). Like explicit feedback, implicit feedback can be positive and negative, although the negative feedback is not reliable. Explicit feedback is noise free although the user is unlikely to rate many items, whereas implicit feedback is noisy (error prone), but can collect a lot of training data (Alag, October, 2008). In general, a trade-off between implicit and explicit user feedback is used.

The different techniques to gather the information about users, called knowledge acquisition techniques (Middleton et al., 2004), are beyond the scope of this thesis. We assume that we have some ratings provided by users about items and the task is to make useful recommendations.

An item's profile can be defined in different ways: (1) by getting features (or meta data) about the item (Mooney and Roy, 2000), (2) by using the ratings provided by users on that item (Sarwar et al., 2001), (3) by using the domain-specific Ontologies (Maidel et al., 2008), and (4) by using demographic information (category) about items (Vozalis and Margaritis, 2007). The vector space model (van Meteren and van Someren, 2000) (described in the next chapter) is the most widely used method to represent the item's profiles. Next, we give the classification of the existing recommender system's algorithms.

2.4 Classification of Recommender Systems

Recommender systems fall into five main classes: collaborative filtering, content-based filtering, demographic-based, knowledge-based, and hybrid recommender systems. In this thesis, we have focused (mainly) on collaborative filtering, content-based filtering, and demographic filtering. Next, we discuss the approaches they use for recommendation, their merits and drawbacks, and some domains where they have been successfully applied.

2.4.1 Collaborative Filtering (CF) recommender systems

Collaborative Filtering (CF) recommender systems (Goldberg et al., 1992; Resnick et al., 1994; Shardanand and Maes, 1995; Terveen et al., 1997; Konstan et al., 1997; Ghazanfar and Prügel-Bennett, 2010d) recommend items by taking into account the taste (in terms

of preferences of items) of users, under the assumption that users will be interested in items that users similar to them have rated highly. Examples of these systems include GroupLens system (Konstan et al., 1997) and Ringo (ringo.com). Collaborating filtering recommender systems are based on the assumption that people who agreed in the past will agree in the future too. In these systems, the utility $f(i, u)$ of item i for user u is estimated based on the utilities $f(i, u'|u' \in U)$ assigned to item i by those users $U \subset \mathcal{U}$ who have similar taste to user u (also called neighbours of user u).

These systems take into account the ratings provided by users on items and build the user-item rating matrix, where each row of the matrix represents a user profile and the column represents an item profile. The following example, based on the user-item rating matrix given in Table 2.1, illustrates how these systems make predictions.

Example 1: *User Musi has not seen the movie "The Godfather" and he is in a dilemma—whether or not to rent this movie. Only two users, Hamza and Adam have already seen this movie. He knows that Hamza has the same taste in movies as he has, as both of them have liked "Troy" and disliked "Forest Gump" movies. Furthermore, he knows that Adam has quite opposite tastes to his, as Adam has liked the movies he disliked (i.e. "Forest Gump") and vice versa. Considering this he asks Hamza's opinion and discards (or acts opposite to) Adam's opinion and makes the decision accordingly. It must be noted that Fahime has exactly the same taste as Musi; however, her opinion cannot be taken into account, as she has not rated the "The Godfather".*

These systems can be classified into two sub-categories: memory-based and model-based CF, which are discussed next.

2.4.1.1 Memory-based CF

Memory-based approaches (Goldberg et al., 1992; Resnick et al., 1994; Konstan et al., 1997; Breese et al., 1998) make a prediction by taking into account the entire collection of previous rated items by a user. There are three main steps in this approach:

- In the first step, users rate some items they have experienced previously.
- In the second step, an *active user* (the user for whom the recommendations are computed)'s profile is matched with other users' profiles in the system. A set of similar users also called *neighbours* of the active user are found.
- In the last step, predictions are made for items that the active user has not rated based on the ratings provided by its nearest neighbours. Finally, these items are presented to the active user in a suitable order.

There are several methods to define the nearest neighbours: (1) choosing the top l-nearest neighbours, (2) choosing all neighbours whose similarity is greater than a given

similarity threshold, and (3) filtering out the neighbours with negative similarities. The approach that uses the l most similar neighbours has been widely used (Breese et al., 1998; Ma et al., 2007; Vozalis and Margaritis, 2007). The l nearest neighbour approach gave us the best results and hence this work is based on this approach[3]. In the l-nearest neighbour approach, the value of an unknown rating r_{i,u_a} for item i and user u_a is computed by aggregating the ratings of other l similar users for the same item i:

$$\check{r}_{i,u_a} = aggr_{u \in \mathcal{U}^{neigh}} r_{i,u} \,. \tag{2.1}$$

Where $\mathcal{U}^{neigh} \subset \mathcal{U}$ represent the set of l users ($l < M$) that are the most similar to user u_a and who have rated item i, and $aggr$ represents an aggregate function. Aggregate functions include:

$$\check{r}_{i,u_a} = \frac{1}{l} \sum_{u \in \mathcal{U}^{neigh}} r_{i,u}, \tag{2.2}$$

$$\check{r}_{i,u_a} = \frac{1}{\sum_{u \in \mathcal{U}^{neigh}} |sim(u_a, u)|} \sum_{u \in \mathcal{U}^{neigh}} sim(u_a, u) \times r_{i,u}, \tag{2.3}$$

$$\check{r}_{i,u_a} = \bar{r}_{u_a} + \frac{1}{\sum_{u \in \mathcal{U}^{neigh}} |sim(u_a, u)|} \sum_{u \in \mathcal{U}^{neigh}} sim(u_a, u) \times (r_{i,u} - \bar{r}_u), \tag{2.4}$$

where $sim(u_a, u)$ is the similarity between user u_a and u, and $\bar{r}_u$ is the average rating of user u. The average rating of user u is computed as follows:

$$\bar{r}_u = \frac{1}{|\mathcal{I}_u|} \sum_{i \in \mathcal{I}_u} r_{i,u}, \tag{2.5}$$

where $\mathcal{I}_u$ represents the set of items that have been rated by user u. In equation 2.2, the aggregate function is a simple *average* function, whereas in equation 2.3, it is a *weighted sum* of the ratings. The prediction of r_{i,u_a} depends on the $sim(u_a, u)$ which is used as a weight here. The function $sim(u_a, u)$ is a heuristic that gives more weight to similar users than to dissimilar ones, while making predictions. It is worth noting that the similarity measure is application-dependent, i.e. different applications can use different similarity measures that suit their requirements. Equation 2.4 is called the *adjusted weighted sum* that considers the deviation of ratings from the average rating of the corresponding user. This aggregate function overcomes the dissimilar rating scale used by different users.

Several approaches can be used for measuring the similarity between two users. Two famous approaches are *correlation-based* and *cosine-based* similarity measures. In the correlation-based approach, the similarity between two users is measured by the *Pearson*

[3]We have not shown the results comparing these approaches. Interested candidates can refer to Herlocker et al. (2002).

correlation. The Pearson correlation (Breese et al., 1998) between two users u_a and u_b is computed as follows:

$$sim(u_a, u_b) = \frac{\sum_{i \in \mathcal{I}_{u_a,u_b}} \mu_{i,u_a} \mu_{i,u_b}}{\sqrt{\sum_{i \in \mathcal{I}_{u_a,u_b}} \mu_{i,u_a}^2 \sum_{i \in \mathcal{I}_{u_a,u_b}} \mu_{i,u_b}^2}}, \tag{2.6}$$

where $\mu_{i,u} = r_{i,u} - \overline{r}_u$. The output of the Pearson coefficient is 1 when two users are perfectly similar, 0 when they are not similar, and -1 if they are totally dissimilar. Another famous similarity function is the *cosine-based approach* (Breese et al., 1998). In the cosine-based approach, the profiles of two users u_a and u_b are represented by vectors in the X-dimensional space, where $X = |\mathcal{I}_{u_a,u_b}|$. The cosine of the angle between two vectors gives the similarity measure and is computed as follows:

$$\begin{aligned} sim(u_a, u_b) &= \cos(\vec{u_a}, \vec{u_b}) \\ &= \frac{(\vec{u_a}.\vec{u_b})}{(||\vec{u_a}||_2 \times ||\vec{u_b}||_2)} \\ &= \frac{\sum_{i \in \mathcal{I}_{u_a,u_b}} r_{i,u_a} r_{i,u_b}}{\sqrt{\sum_{i \in \mathcal{I}_{u_a}} r_{i,u_a}^2 \sum_{i \in \mathcal{I}_{u_b}} r_{i,u_b}^2}}, \end{aligned} \tag{2.7}$$

where $\vec{u_a}.\vec{u_b}$ represents the dot product between $\vec{u_a}$ and $\vec{u_b}$. The output of the cosine-based approach is 1 when two users are similar, and is 0 when they are not similar. Furthermore, Ahn (2008) proposed a new heuristic similarity measure to overcome the cold-start problems.

2.4.1.2 Model-based CF

Model-based approaches (Sarwar et al., 2000b, 2002b; Vozalis and Margaritis, 2006a; Rendle and Lars, 2008; Park and Tuzhilin, 2008) learn a model from a collection of ratings and use this model for making predictions. A well-known example of these approaches is the item-based CF (Sarwar et al., 2001). It builds a model of item similarities using an off-line stage. Let us assume that we want to make prediction on item i_t for user u. There are three main steps in this approach as follows:

- In the first step, all items rated by an active user are retrieved.
- In the second step, the target item's similarity is computed with the set of retrieved items. A set of l most similar items $i_1, i_2 \cdots i_l$ with their similarities $\{sim(i_t, i_1), sim(i_t, i_2), \cdots sim(i_t, i_l)\}$ are selected. Similarity $sim(i_x, i_y)$, between two items i_x and i_y, is computed by first isolating all the users who have rated these items

(i.e. $\mathcal{U}_{i_x i_y}$), and then applying the Adjusted Cosine similarity (Sarwar et al., 2001) as follows:

$$sim(i_x, i_y) = \frac{\sum_{u \in \mathcal{U}_{i_x,i_y}} \mu_{i_x,u}\, \mu_{i_y,u}}{\sqrt{\sum_{u \in \mathcal{U}_{i_x,i_y}} \mu^2_{i_x,u} \sum_{u \in \mathcal{U}_{i_x,i_y}} \mu^2_{i_y,u}}}. \tag{2.8}$$

Where, $\mu_{i,u} = r_{i,u} - \overline{r}_u$, i.e. normalising a rating by subtracting the respective user's average from the rating, which is helpful in overcoming the discrepancies in the user's rating scale. We used the significance weighting schemes (Ghazanfar and Prügel-Bennett, 2010d) while measuring the similarities.

- In the last step, prediction for the target item is made by computing the weighted average of the active user's rating on the l most similar items. Using the weighted sum, the prediction $r_{i_t,u}$ on item i_t for user u is computed as follows:

$$\check{r}_{i_t,u} = \frac{\sum_{j=1}^{l} (sim(i_t, j) \times r_{j,u})}{\sum_{j=1}^{l} (|sim(i_t, j)|)}. \tag{2.9}$$

Equation 2.9 cannot be generalised to all datasets. If most of the item-item similarities are negative, then it would result in negative prediction, which is not correct. This formula can be corrected, by using the *adjusted weighted sum* that considers the deviation of ratings from the average rating ($\overline{r}_u$) of a user.

$$\check{r}_{i_t,u} = \overline{r}_u + \frac{\sum_{j=1}^{l} (sim(i_t, j) \times \mu_{j,u})}{\sum_{j=1}^{l} (|sim(i_t, j)|)}, \tag{2.10}$$

where $\mu_{i,u} = r_{i,u} - \overline{r}_u$. Next, we highlight the advantages and disadvantages of CF recommender systems.

2.4.1.3 Advantages and disadvantages of CF recommender systems

Their advantages include:

1. *Can identify cross-genre niches:* They can make recommendations outside the preferences ("outside the box" (Burke, 2002)) of an individual, for instance, a user who loves watching action movies can also enjoy getting a good romantic movie.

2. *Domain knowledge is not needed:* These systems do not require domain knowledge as required in knowledge-based recommender systems.

3. *Adaptive:* They capture more information about users' preferences over time, which results in improved recommendations.

4. *Produce high quality recommendations:* They produce high quality recommendations compared to the other types of recommender systems. Furthermore, they work well for complex objects such as music and movies.

5. *Implicit feedback is sufficient:* They can generate recommendations by only taking a user's implicit feedback into consideration.

Their disadvantages include:

1. *New user/item cold-start problem:* The performance of these systems suffer under new user and item cold-start problems. The new item problem is also known as the *early-rated* problem (Burke, 2002), since the first user to rate the new item gets little reward.

2. *Sparsity:* In most of these systems, the percentage of ratings assigned by users is very small compared to the percentage of ratings the system has to predict; hence prediction accuracy of a recommender system suffers in this case.

3. *Coverage:* Due to the sparsity problem, the coverage of a typical CF recommender system is typically very low.

4. *Scalability:* Memory-based CF approaches do not scale well with the number of users/items and ratings. Some dimensionality reduction techniques, such as Sarwar et al. (2002b) and Xue et al. (2005) have been proposed to overcome this problem.

5. *Users with unique taste:* There can be users in the system that have unusual taste compared to the rest of the community, so the CF recommender systems would produce poor recommendations for these users.

6. *Stability vs. Plasticity problem:* Once a detailed user's profile has been built, then it becomes very difficult for these systems to change this profile.

7. *Long tail problem:* The performance of these systems suffers under the long tail scenario.

2.4.2 Content-Based Filtering (CBF) recommender systems

Content-Based Filtering (CBF) recommender systems (Lang, 1995; Mooney and Roy, 2000; van Meteren and van Someren, 2000; Pazzani and Billsus, 2007) recommend items

based on the description information of an item, under the assumption that users will like similar items to the ones they liked before. The description of the items can be automatic, where the feature extraction algorithms are used to extract features from the description of the item, or manual, where the domain experts annotate the items. Furthermore, recent social tagging websites (e.g. Flicker) allow the user to tag certain items that can be used to describe an item. In Chapter 3, we discuss how features can be extracted from the description of items. The content-based filtering systems estimate the utility $f(i, u)$ of item i for user u based on the utilities $f(i'|i' \in I, u)$ assigned by user u to items $I \subset \mathcal{I}$ that are similar to item i.

These approaches have their roots in Information Retrieval (IR) (Berry et al., 1995) research. The IR approaches focus on answering the ephemeral interest queries of a user, for instance, finding all the movies that involve the James Bond character. As these approaches only store the specific user's queries and not the long-term user's interests, hence they are less valuable for actual recommendation process. The CBF approaches differ from IR in a sense because they store and update the user's profile—tastes, preferences, and needs—in the system which can be used to give personalised results.

There are four main steps in these approaches, as described below:

- In the first step, the system gathers information about items; for example, in a movie recommender system, this would be on movie title, genre, actors, producers, etc.
- In the second step, a user is asked to rate some items. Binary scale (in terms of their likes/dislikes) or some numeric scale (e.g. 1 to 5) are used for capturing the user's ratings.
- In the third step, a user's profile is built based on the information gathered in the first step and the rating provided in the second step. Different machine learning or information retrieval techniques are used for this purpose. Users' profiles (which are long-term models) update as more information about users' preferences is observed and are highly dependent on the learning method employed.
- In the last step, the system matches the content of un-rated items with the active user's profile and assigns a score to items based on the quality of match.

For example, in a movie recommender system, the system finds movies similar to the ones a user has rated highly in the past based on a specific actor/actress, director, subject, etc. Different similarity measures can be used for measuring the similarity between the item and user profiles. A frequently used similarity measure is the cosine similarity. Furthermore, different machine learning techniques, such as classification (Mooney and

Roy, 2000; Melville et al., 2002), regression and clustering (Steinbach et al., 2000) can be used for content-based recommendations. Next, we highlight the main advantages and disadvantages of CBF recommender systems (Adomavicius and Tuzhilin, 2005; Pazzani and Billsus, 2007).

2.4.2.1 Advantages and disadvantages of CBF recommender systems

The advantages of the content-based filtering systems are essentially the same as those of CF: they do not need domain knowledge, they are adaptive, and can operate with implicit feedback. Furthermore, they do not have the new item problem. Their disadvantages include:

1. *Limited content analysis:* These systems depend on the features that are explicitly associated with items; hence there should be a sufficient number of features. For this purpose, the features should be machine-readable as manual assignment of features to items is not pragmatic due to limited resources.
2. *Over-specialisation:* These systems only recommend items that are the most similar to a user's profile. In this way, a user cannot find any recommendation that is different from the ones they have already rated or seen.
3. *New user problem:* In order to build the model of user preferences, a content-based filtering system requires users to rate a large number of items, which is not possible for a newly registered user. Hence, the system would produce poor quality recommendations.

2.4.3 Knowledge-Based (KB) recommender systems

2.4.3.1 Utility-based recommender systems

Utility-based recommender (Burke, 2002, 1999) systems do not attempt to build long-term users' profiles, but rather attempt to suggest items based on inferences about users' needs and preferences. The users' profiles can be any knowledge structure that endorses this inference. Mainly, these systems have catalogue, functional and user knowledge.

2.4.3.2 Ontology-based recommender systems

Ontology-based recommender systems (Middleton, 2002; Middleton et al., 2002; Buriano et al., 2006; Cantador et al., 2007; Weng and Chang, 2008; Shoval et al., 2008) use Ontologies to define the users' and items' profiles. These systems generate recommendations by assessing the similarities between the instances associated with the users and

all other instances in the system's knowledge bases. Different heuristics are used for assigning the weights to super- and sub-instances. In the next chapter, we show how we can exploit this idea to define the genre vector of a movie. Next, we discuss the main advantages and disadvantages of KB recommender systems.

2.4.3.3 Advantages and disadvantages of KB recommender systems

They have no new user and sparsity problems. They are sensitive to the change of preferences of users and can map users' needs to products. Furthermore, it has been claimed that that Ontology-based representation of context information can give us benefits; for instance, information can be augmented, enriched, and synthesised using suitable reasoning mechanisms (Buriano et al., 2006).

The disadvantages of these systems are: they require labour-intensive knowledge engineering techniques to capture the catalogue and user knowledge, and they (utility-based) are static in a sense that they do not learn users' profiles over time.

2.4.4 Demographic-Based (DM) recommender systems

Demographic-Based (DM) recommender systems categorise users based on their personal attributes (e.g. age, gender, etc.) and make recommendations based on these categorisations. Furthermore, items can be categorised based on their attributes; for example, a movie can be categorised into different groups based on its genre information (Vozalis and Margaritis, 2007), and hence recommendations can be generated based on this categorisation. Next, we discuss the advantages and disadvantages of DM recommender systems.

2.4.4.1 Advantages and disadvantages of DM recommender systems

Like CF recommender systems, they do not need domain knowledge, they are adaptive, and can identify cross-genre niches. They share a number of common problems with CF recommender systems, such as sparsity, stability, plasticity, poor recommendations for the gray-sheep users and in some cases, they do not have enough demographic data against a user. Furthermore, with an increase in the sensitivity to on-line privacy, users are reluctant to supply demographic information, which is a potential problem for these systems.

2.4.5 Hybrid recommender systems

Hybrid recommender systems combine CBF, CF, KB and DM recommenders to avoid certain aforementioned limitations of the individual systems. Several hybrid recommender systems have been proposed (Pazzani, 1999; Claypool et al., 1999; Burke, 1999; Melville et al., 2002; Burke, 2002, 2007). We show in Chapter 4 how we can combine individual systems systematically to produce effective recommendations under different scenarios.

2.4.6 Other types of recommender systems

We briefly outline other recommender systems.

2.4.6.1 Context-aware recommender systems

Context-aware recommender systems (Hayes and Cunningham, 2004; Baltrunas, 2008) are relatively new and little work has been done in this area. The basic idea is that the appropriateness of recommendations is highly dependent on the context in certain scenarios. The basic recommendation algorithms cannot fulfill the user's immediate interests or needs so any recommendation made may not be appropriate for the current context. Certain approaches are used to incorporate the context information into the recommendation generation process; for example *feature selection*, exploiting the most relevant items based on the current context, and *weighting scheme*, using all the items and assigning higher weights to items that are relevant to the current context. A multi-dimensional approach to incorporate the context information into collaborative filtering has been proposed in Adomavicius et al. (2005). We show in Appendix D how our Kernel Mapping Recommender (KMR) system algorithms can exploit the context information.

2.4.6.2 Rule filtering recommender systems

Rule filtering recommender systems define some rules either based on a user's history or require them to explicitly formulate rules (e.g. *I never watch horror movies*), and recommend unknown items based on these rules. An example of these systems is the Tapestry (Goldberg et al., 1992) recommender system. These systems can also be categorised under utility-based systems. Their drawbacks include (1) inferring rules can become very complicated as users rate more and more items and inferred rules might conflict with one another, (2) users might find defining rules in formal languages an awkward process, and (3) to precisely define the rule, the user needs to know exactly what they would like to be recommended, which is somewhat in conflict with the notion of a recommender system. We have not focused on these systems in this thesis.

2.5 State-of-the-art Recommendation Algorithms

Over the last 20 years, a number of recommender system algorithms have been proposed, which use different techniques to solve the recommendation problem, including neighbourhood Collaborative Filtering (CF) (Sarwar et al., 2001), clustering-based approaches (Connor and Herlocker, 2001; Sarwar et al., 2002b; Xue et al., 2005; Rashid et al., 2006; Park and Tuzhilin, 2008; Shepitsen et al., 2008), Bayesian network (Stern et al., 2009), Singular Value Decomposition (SVD)-based approaches (Sarwar et al., 2000b, 2002a; Vozalis and Margaritis, 2007; Barragáns-Martínez et al., 2010), and various matrix factorisation techniques (Srebro et al., 2005; Rennie and Srebro, 2005; Bell et al., 2007; Wu, 2007; Takács et al., 2008; Salakhutdinov and Mnih, 2008; Lawrence and Urtasun, 2009; Koren et al., 2009; Takács et al., 2009).

Several Matrix Factorisation (MF)-based approaches have been devised and applied in the CF domain. The idea of MF is to approximate the original user-item rating matrix with a low-rank one. There are several ways to achieve this goal; for example, a well-known approach, Maximum Margin MF (MMMF) (Srebro et al., 2005), minimises the sum of squared error between the observed and the predicted ratings. The complexity of the model is controlled by penalising the trace/nuclear norm (sum of singular values) of the matrix. Other approaches extending the MMMF approach have been proposed; for example, Salakhutdinov and Mnih (2008) offer the Bayesian treatment of the problem and Lawrence and Urtasun (2009) offer the non-linear MF using the Gaussian latent variable model. Some other well-known MF techniques are expectation maximisation for MF (Kurucz et al., 2007), alternative least square (Bell and Koren, 2007b), mixed membership matrix factorisation (Mackey et al., 2010), and ensembles of MF techniques (Takács et al., 2008).

Different MF-based approaches have been combined together for improving the prediction accuracy; for example, Bell et al. (2007) proposed a solution for the Netflix prize (Bennett and Lanning, 2007) by blending 107 individual predictors, and won the Netflix 2007 prize. A similar approach is presented in Paterek (2007), where the author proposed a linear combination of SVD-based predictor, K-means clustering, SVD combined with K Nearest Neighbours (KNN), SVD post processed with ridge regression, and others (total of 72 predictors) and claimed that it gave 7.04% improvement in terms of Root Mean Square Error (RMSE) over the Netflix's Cinematch[4] on the Netflix prize competition. Wu (2007) combined (using ensemble methods) different variants of matrix factorisation, such as regular matrix factorisation, and non-negative matrix factorisation, and claimed that the combined approach gave 7% improvement, in terms of RMSE, over the Netflix's CineMatch recommender system. Furthermore, hybrid approaches combining the neighbourhood-based methods with MF have been devised (Koren, 2008; Takács et al.,

[4]Cinematch is the Netflix proprietary recommender system.

2009). Though, theoretically, we can increase the accuracy of a recommender system by these methods; however, it is not pragmatic (Piotte and Chabbert, 2009).

Most of the aforementioned algorithms used the rating information ignoring the feature and demographic information about users/items. Some of the algorithms employed the side-information; for example, Stern et al. (2009) used meta-data about users/items; Lawrence and Urtasun (2009) used the meta-data about items, Koren (2008) used implicit feedback provided by users (information about which items have been rated by users, even if we do not know the actual ratings). However, none of them incorporated more general forms of the side-information (e.g. features and demographics).

These algorithms offer state-of-the-art performance in terms of accuracy on static datasets. However, our goal is very different from the aforementioned algorithms, i.e. to satisfy a broader set of design objectives discussed in the previous chapter (Section 1.2) and not just improving the accuracy of recommendations. Furthermore, most of these algorithms perform well using a dataset's particular peculiarities; hence their performance cannot be generalised to other datasets.

Next, we briefly describe the algorithms that we have used for benchmarking our proposed algorithms. We chose several other algorithms based on the number of citations given in the literature, the algorithm classification space (i.e. memory-based or model-based approaches), and whether the algorithm claims to give state-of-the-art results (over the datasets described in Chapter 3, Section 3.1). We have used the following algorithms to benchmark our proposed Kernel Mapping Recommender (KMR) system algorithms:

UBCF_{DV} : Breese et al. (1998) proposed a variant of the user-based CF (refer to Section 2.4.1.1), where the main idea is to use some default votes to decrease the sparsity of the user-item rating matrix. The author claimed that it outperformed the conventional user-based CF algorithm.

IBCF: The item-based CF proposed by Sarwar et al. (2001) has been described in Section 2.4.1.2. The author claimed that it is more accurate and scalable than the conventional user-based CF.

Hybrid CF: A naive approach which combines the user- and item-based CF by taking the average of their results.

Baseline SVD: Baseline SVD, proposed by Sarwar et al. (2000b), is the conventional SVD-based approach for solving the recommendation problem. The steps to make predictions using this approach are given in Chapter 5 (Section 5.3).

MatchBox: The MatchBox recommender system (Stern et al., 2009) is based on the fully Bayesian matrix factorisation (similar to one proposed by Salakhutdinov and Mnih (2008)). It employs users' features, items' features, and binary feedback as

input and learns the model by mapping these features into a shared latent space, where the correlation between users and items is learned in order to predict the users' preferences for unknown items. It is an on-line learning algorithm which can incrementally add new data. The authors claimed that the meta-data about users and items (they used the demographic data provided with the MovieLens dataset) can help overcoming some of the cold-start problems.

MMMF: Maximum Margin MF (MMMF) proposed by Srebro et al. (2005) is a variant of MF-based techniques as described above. The author compared the results with several other algorithms proposed by Marlin (2004) over the MovieLens dataset and claimed that the MMMF outperforms them in terms of the MAE.

E-MMF: The E-MMF (DeCoste, 2006) makes predictions using the ensembles of maximum margin MF technique (Srebro et al., 2005). The authors combined different variants of the MMMF using ensemble methods, such as voting by averaging, voting by confidence, and bagging and claimed that ensembles of the MMMF provide better results than a single MMMF over the MovieLens dataset.

NLMF: The non-linear matrix factorisation technique, NLMF, offered by Lawrence and Urtasun (2009), extend the Maximum Margin MF (Srebro et al., 2005) using a Bayesian framework. They tested their algorithm over the MovieLens dataset and claimed that it outperforms several other state-of-the-art algorithms. The authors also suggested to use the meta-data about items to overcome the new item cold-start problem; however, no results were provided to support this claim.

M^{3}F-TIB: The M^3F-TIB proposed by Mackey et al. (2010) integrates two complementary algorithms—discrete mixed membership modelling and continuous latent factor modelling (i.e. matrix factorisation)—into a common framework using the Bayesian approach, which illustrates the power of carefully combining different algorithms. The authors trained the model by performing Bayesian posterior inference with Gibbs sampling. They tested their algorithm over the MovieLens and Netflix dataset and claimed that the algorithm gives state-of-the-art performance outperforming Lawrence and Urtasun (2009)'s results. It must be noted that the Gibbs sampling in a rich Bayesian model is more computationally expensive than some alternative approaches (like maximum a posteriori). Furthermore, this paper totally ignores other design objectives as discussed in the previous chapter (Section 1.2).

To benchmark our hybrid recommendation algorithms, we have used the IBCF, $UBCF_{DV}$, Baseline SVD, and the following algorithms:

cBoosted: Melville et al. (2002) offered a hybrid recommender algorithm to recommend movies to users. In the content-based filtering part, the authors train a Naive

Bayes classifier based on a user's profile. The Naive Bayes classifier is used to approximate the missing entries in the user-item rating matrix, and a user-based CF is applied over this dense matrix. They claimed that the proposed approach outperformed the conventional user-based CF in terms of accuracy.

Baseline SVD-based IBCF: The baseline SVD-based IBCF algorithm, proposed by Vozalis and Margaritis (2006a), extends the Sarwar et al. (2000b)'s approach to item-based CF. The authors reduce the dimensions of the original user-item rating matrix by applying SVD. They claimed that applying item-based CF over the reduced dimensions outperforms the conventional item-based CF in terms of MAE.

Idemsvd − 2svd: $Idemsvd - 2svd$ algorithm, proposed by Vozalis and Margaritis (2007), combines the SVD-based IBCF approach with demographic data. The authors applied SVD over the user-item rating matrix and demographic data of users and items, and claimed that a system consisting of a linear combination of SVD-based demographic correlation and SVD-based (item-based) CF increases the accuracy of the recommender system.

2.6 Summary

In this chapter we discuss and formalise the recommender system problem. The recommender systems consist of two main entities: users and items, where users provide their opinion about the items. The users' profiles are defined based on the their opinions about items (e.g. ratings, comments, etc.) and information (e.g. demographic information); whereas items' profiles are defined based on the item's description (e.g. keywords describing the item). We show how, by exploiting the users' and items' profiles, a recommendation algorithm can solve the problem of recommending items to users. We provide a classification of the existing recommendation algorithms based on their principal characteristics. We discuss their main benefits as well as their pitfalls by highlighting several factors under which their performance would suffer. At the end, we discuss the limitations of the state-of-the-art algorithms and the algorithms that we have used to benchmark our work.

Chapter 3

Experimental Methodology

In this chapter, we discuss in detail the experimental set up of our system. We start with the description of the datasets used in this work. Then we give an overview of the different metrics that we have used to evaluate this work. We present the feature extraction and selection algorithms and show how they are used to extract the features from the content descriptions of movies. After that, we discuss how these features are used to train the classification techniques that we will use in this work. At the end, we explain how we employ the demographic information to build the demographic-based recommender systems.

3.1 Datasets

As is common in the field of recommender systems, we used data from film recommendation sites to test our algorithms. These provide some of the largest available datasets allowing us to test the scaling performance of the algorithms. In addition, as these datasets are very commonly used in the literature it allows us to benchmark our algorithm against the state-of-the-art. The datasets we have used in our work are described as below.

- **MovieLens 100K ratings:** This dataset (denoted by "SML" in this work) contains 943 users, 1 682 movies, and 100 000 ratings on an integer scale from 1 (bad) to 5 (excellent). It has been used in many research projects, such as Sarwar et al. (2000b), Sarwar et al. (2001), Sarwar et al. (2002a), Vozalis and Margaritis (2006b), Vozalis and Margaritis (2007) and Barragáns-Martínez et al. (2010).
- **MovieLens 1M ratings:** This dataset (denoted by "ML" in this work) contains 6 040 users, 3 900 movies, and 1 000 000 ratings. It has been used in projects such as Melville et al. (2002), Lawrence and Urtasun (2009) and Takács et al. (2009). This dataset has the same rating scale as the 100K one.

- **MovieLens 10M ratings:** This dataset (denoted by "ML10" in this work) contains 71 567 users, 10 681 movies, and 10 000 054 ratings on a floating point scale from 1.0 to 5.0 (with a difference 0.25). It has been used in projects such as Melville et al. (2002), Lawrence and Urtasun (2009) and Mackey et al. (2010).

- **FilmTrust:** We created this dataset by crawling the FilmTrust website. The dataset retrieved (on 10th March 2009) contains 1 214 users, 1 922 movies, and 28 645 ratings on a floating point scale of 1 (bad) to 10 (excellent) (with a difference of 0.25). The FilmTrust dataset adequately captures the new user and new item cold-start problems. It has imbalanced data, i.e. one user may have provided one rating and others may have provided hundreds of ratings and the same is true for items as well. We also created a subset of this dataset by filtering all users and movies which have less than 5 ratings. The resulting dataset contains 1 016 users, 314 movies, and 25 730 ratings. In this work, we have used the terms "FT1" and "FT5" to denote the original and the filtered FilmTrust datasets respectively. It is worth noting that the FT5 dataset is relatively denser than the FT1 dataset (refer to Table 3.1).

- **Netflix:** Random sub-sample of 20 000 users from the Netflix dataset (denoted by "NF" in this work). The sub-sampled dataset contains 20 000 users, 17 766 movies, and 4 260 735. It has the same rating scale as that of the SML dataset. It has been very widely used (Bell et al., 2007; Bell and Koren, 2007a; Wu, 2007; Koren, 2008; Piotte and Chabbert, 2009), in part because of the prize (Bennett and Lanning, 2007) offered for achieving a level of improvement over a benchmark. We have not attempted to compare our algorithm against the state-of-the-art Netflix algorithms for three reasons. First, they have been highly tuned to that particular dataset, while we have concentrated on developing general purpose recommender algorithms. Second, the full Netflix dataset is so large (training dataset consists of 100 480 507 ratings provided by 480 189 users on 17 770 movies) that it is difficult to process on a normal desktop machine without spending significant time on optimising memory management. Third, the performance of those algorithms was evaluated over a sub-set of the dataset called the "qualifying set" and only the jury (Netflix organisers) knows the actual ratings of the qualifying set. After the completion of the Netflix prize, there is no longer any support for this.

The characteristics of the datasets described earlier are given in Table 3.1. The sparsity of a dataset is calculated as $\left(1 - \frac{non\ zero\ entries}{all\ possible\ entries}\right)$; for instance for the SML dataset it is: $1 - \frac{100000}{943 \times 1682} = 0.937$. This means that only 6.3% of the total user-item pairs have been rated.

Table 3.1: Characteristics of the datasets used in this work. FT5, FT1, SML, ML, ML10, and NF represent the FilmTrust filtered, FilmTrust original, MovieLens $100K$, MovieLens $1M$, MovieLens $10M$, and Netflix datasets respectively. Average rating represents the average rating given by all users in the dataset.

Characteristics	**Dataset**					
	FT5	**FT1**	**SML**	**ML**	**ML10**	**NF**
Number of users	1 016	1 214	943	6 040	71 567	20 000
Number of movies	314	1 922	1 682	3 706	10 681	17 766
Number of ratings	25 730	28 645	100 000	1 000 209	10 000 054	4 260 735
Rating scale	1.0 to 10.0	−−	1 to 5	1 to 5	1.0 to 5.0	1 to 5
Sparsity	0.919	0.988	0.934	0.955	0.987	0.988
Max number of ratings given by a user	133	244	737	2 314	7 359	17 653
Max number of ratings given to a movie	842	880	583	3 428	34 864	9 667
Average rating	7.601	7.607	3.529	3.581	3.512	3.591

3.2 Getting Additional Features About Movies

In this work, we have used content information about movies in addition to the rating information for the FT, SML, and ML10 datasets. For the ML10 dataset, we used the tag information provided with the dataset (http://www.grouplens.org/node/12). For the FT and SML datasets, we obtained the additional information by crawling the Internet Movie Database (IMDB) web site (www.imdb.com). Specifically, we matched the titles and URLs provided in the dataset with those given in IMDB, using the jmdb (www.jmdb.com). The information crawled from the IMDB against a movie is given in Table 3.2.

We found keywords, tags, and cast (e.g. actors, actresses, etc.) as the most important information about movies. We did not take into account the critics or user reviews which might also be helpful.

3.3 Evaluation Metrics

Several metrics have been used to evaluate the performance of recommender systems; however, there is a lack of standardisation, which makes it hard to compare published results. Herlocker et al. (2004) give an overview of the different metrics that have been used along with their merits/demerits. To date, the majority of the published work has focused on the accuracy metrics, which can broadly be categorised into three categories: (1) *predictive accuracy metrics*, (2) *classification accuracy metrics*, and (3) *rank accuracy metrics* (Herlocker et al., 2004).

Table 3.2: The information crawled from the IMDB against a movie.

Crawled Information	**Description of the Information**
Keywords	Keywords given to a movie (variable length)
Plot summary/synopsis	Summary or synopsis of the movie (variable length)
Tags	Tags given to a movie (variable length)
Movie links	Links between movie e.g. followed by, series of, version of a movie
Actors/Actresses	Top five actors/actresses
Directors	Top five directors
Producers	Top three producers
Editors	Top three editors
Writers	Top three writers
Production companies	Top three companies
Technical	Sound mix (e.g. DTS), colour info, film negative format e.g. 35 mm film
Soundtracks	Music by and lyric by information
MPAA (The Motion Picture Association of America) ratings	Ratings that provide parents with advance information about the content of films. Ratings can be G, PG, PG-13, R and NC-17
AKA-Titles	Movie is also known as
Language	Original language of the movie (e.g. English, French, etc.)
Ratings	Global rating given by the community of users (rounded to the nearest integer)
Votes	Number of votes (v) given to a movie by the community of users. We divide the movies into 10 clusters (C) based on the number of votes they received as follows: $\{C_1 \mid v \leq 100$, $C_2 \mid (v > 100\, AND\, v \leq 500)$, $C_3 \mid (v > 500\, AND\, v \leq 1000)$, $\cdots$, $C_{10} \mid v > 5\,000\ \}$. Any two movies residing in the same cluster are considered similar

- *Predictive accuracy metrics* measure how close is the recommender system's predicted value of a rating, with the true value of that rating assigned by the user. They are used in cases where the task is to display the predicted ratings to users; for instance the MovieLens recommender system displays the number of stars (from 1 to 5) a user would give to an unknown movie. These metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Normalised Mean Absolute Error (NMAE) and have been used extensively in research projects such as Breese et al. (1998), Sarwar et al. (2000a), Sarwar et al. (2000b), Sarwar et al. (2001), Sarwar et al. (2002b) and Xue et al. (2005).

- *Classification accuracy metrics* (sometimes referred to as the decision support metrics) determine the frequency of decisions made by a recommender system, for finding and recommending a high quality item (the item the user would like to consume) to a user. These are used in cases where the exact rating prediction is not required, rather the task is to help the user to classify high quality items from the available ones; for instance presenting the user with a list of top-N recommended

items. These metrics include Receiver Operating Characteristic (ROC)-sensitivity, precision, recall, and $F1$ measure, and have been used in Sarwar et al. (2000b,a) and Zanardi (2011).

- *Rank accuracy metrics* measure the proximity between the ordering predicted by a recommender system to the ordering given by the actual user, for the same set of items. These metrics present users with a ranked list of recommendations with the assumption that they are unlikely to browse every recommendation. These metrics include half-life utility metric proposed by Breese et al. (1998) and have not been used widely.

We have focused on predictive accuracy metrics and classification accuracy metrics because our specific task in this work is to predict scores for items that have already been rated by actual users, and to check how well this prediction helps users in selecting high quality items. Furthermore, these metrics allow us to benchmark our results with other state-of-the art algorithms. Specifically, we have used the MAE, RMSE, NMAE, ROC-sensitivity, precision, recall, and F1 measure. In addition to the accuracy metrics, we used coverage. We also showed how learning rate and confidence can be defined.

3.3.1 Mean Absolute Error (MAE) and related metrics

The Mean Absolute Error (MAE) measures the average absolute deviation between the rating predicted by a recommendation algorithm and the true rating assigned by the user. It is computed as follows:

$$MAE = \frac{1}{|\mathcal{D}^{test}|} \sum_{r_{i,u} \in \mathcal{D}^{test}} |\check{r}_{i,u} - r_{i,u}|,$$

where $r_{i,u}$ and $\check{r}_{i,u}$ are the actual and predicted values of a rating respectively, and $\mathcal{D}^{test}$ is the set of rating records in the test set. A rating record is a tuple consisting of a user ID (Identifier), movie ID, and rating, $< uid, mid, r >$, where r is the rating a recommender system has to predict. It has been used in Breese et al. (1998), Sarwar et al. (2000b), Sarwar et al. (2001), Sarwar et al. (2002a), Vozalis and Margaritis (2006a), Ma et al. (2007), Zhang and Pu (2007), Vozalis and Margaritis (2007), Ghazanfar and Prügel-Bennett (2010e) and Ghazanfar and Prügel-Bennett (2010a). The aim of a recommender system is to minimise the MAE score.

The Normalised Mean Absolute Error (NMAE) has been used in Marlin (2004), DeCoste (2006), and Lawrence and Urtasun (2009), and is computed by normalising the MAE by a factor. The value of the factor depends on the range of the ratings; for example, for the MovieLens dataset, it is 1.6. For further information, refer to Lawrence and Urtasun (2009).

A closely related measure to the MAE is the Root Mean Squared Error (RMSE), which is calculated as follows:

$$RMSE = \sqrt{\frac{1}{|\mathcal{D}^{test}|} \sum_{r_{i,u} \in \mathcal{D}^{test}} (\check{r}_{i,u} - r_{i,u})^2}.$$

The RMSE has been used in Bell and Koren (2007b), Rendle and Lars (2008), Lawrence and Urtasun (2009), and Piotte and Chabbert (2009). It will be slightly more sensitive to large outliers than MAE.

3.3.2 Receiver Operating Characteristic (ROC)-sensitivity

The Receiver Operating Characteristic (ROC) model assumes that there are two classes for the items: relevant or good items (positive) and irrelevant or bad items (negative). *Sensitivity* (also called recall rate or true positive rate) determines a classifier's performance on classifying a relevant item correctly among all relevant items available during the test. It measures the proportion of the actually relevant items which are correctly identified by the filter. *specificity* measures the proportion of the actually irrelevant items which are correctly identified by the filter. *1-specificity* (also called the false positive rate or false alarm rates) measures how many bad items are returned by the filter.

The ROC curve is generated by plotting, for each predicted item, the sensitivity (true positive rate) vs. *1-specificity* (false positive rate) against the threshold values. It shows how the number of correctly classified relevant (positive) items varies with the number of incorrectly classified irrelevant (negative) items. The Area Under the Curve (AUC) called ROC-sensitivity, increases if the filter classifies more relevant examples correctly. The AUC varies between 1 for a perfect filter to 0 for an imperfect filter, with 0.5 for a random filter.

To use this metric for recommender systems, we must first determine which items are relevant or good (signal) and which are irrelevant or bad (noise). In Melville et al. (2002) the authors consider a movie "good" if the user awarded it a rating of 4 or higher and "bad" otherwise. The flaw with this approach is that it does not take into account the inherent difference in the user rating scale—a user may consider a rating of 3 in a 5-point scale to be good, while another may consider it bad. We consider an item good if a user rated it with a score higher than their average (in the training set) and bad otherwise.

3.3.3 Precision, recall, and F1 measure

Precision, recall, and F1 measure evaluate the effectiveness of a recommender system by measuring the frequency with which it helps users in selecting/recommending a good item. The most appropriate way (Herlocker et al., 2004) to measure the precision and

Table 3.3: Confusion matrix: each row represents the instance in an actual class, while each column represents the instance in the predicted class.

	Selected	Not Selected	Total
Relevant	I_{rs}	I_{rn}	I_r
Irrelevant	I_{is}	I_{in}	I_i
Total	I_s	I_n	I

recall in the context of recommender systems, is to predict the top-N items for the known ratings, which can be done by splitting each user's ratings into the training and test set, training the model on the training set, and then predicting the top-N items from the test set. This has been used in Billsus and Pazzani (1998). Here the underlying assumption is that the distribution of relevant and irrelevant items in each user's test set is the same as the true distribution for that user across all items.

The information retrieval (Berry et al., 1995) area defines an "objective" measure for precision, recall and related metric, where the relevance is independent of the user, and is only associated with the query. However in the context of recommender systems, the term "objective relevance" does not fit well—as every user has different tastes, opinions, and reasons to rate an item, hence, relevance is inherently "subjective" in recommender systems (Herlocker et al., 2004). The first step in computing the precision and recall is to divide items into two classes: relevant and irrelevant (refer to Table 3.3), which is the same as in ROC-sensitivity.

Precision gives us the probability that a selected item is relevant (Herlocker et al., 2004). Mathematically, it is defined as follows:

$$Precision = \frac{I_{rs}}{I_s}.$$

Recall gives us the probability that a relevant item is selected (Herlocker et al., 2004). Mathematically, it is defined as follows:

$$Recall = \frac{I_{rs}}{I_r}.$$

Example 2: *Precision and Recall: Let us assume that a recommender system's database has* 100 *movies,* 50 *of which are starring James Bond. Suppose a user initiates a query to watch a movie starring James Bond. Let us further assume that the recommendation algorithm returns* 10 *movies in response to the query. If* 9 *of the movies in the list actually star James Bond, then the precision of the system is* 9/10*. This means the precision of the system is very high because it contains many relevant items from the retrieved ones. On the other hand, the recall of the system is* 9/50*, which is very low, because it is unable to retrieve all the relevant movies (which are* 50*). A simple, solution to increase the recall is to retrieve all the items in the database, i.e.* 100*, and send them*

to the user. In this case, the recall would increase to 1*; however, the precision would decrease to* 1/2*.*

From Example 2, we see that precision and recall are inversely proportional to each other, and furthermore, they depend on the size of the resultant vector returned to the user. Hence, they must be measured together. $F1$ measure (Herlocker et al., 2004) combines the precision and recall into a single metric and has been used in many research projects, such as Sarwar et al. (2000b,a). F1 is computed as follows:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}.$$

We calculated precision, recall, and F1 measures for each user over the top-20 recommendations, and reported the average results over all users.

3.3.4 Coverage

Coverage measures how many items a recommender system can make recommendation for. Coverage is an important metric, as many modern e-commerce services contain millions of items in the catalogue, which should be recommended to customers. In this work, we did not take coverage as the percentage of items that can be recommended/predicted from all available ones. The reason is, a recommendation algorithm can increase coverage by making bogus predictions, hence coverage and accuracy must be measured simultaneously. We selected only those items that have already been rated by the actual users. Herlocker et al. (2004) have used the term *prediction coverage* for this metric. It can be defined as follows:

$$Coverage = \frac{\sum_{u \in \mathcal{U}^{test}} \sum_{i \in \mathcal{I}_u^{test}} \mathbf{1}_{\mathbb{R}_{>0}}(\check{r}_{i,u})}{|\mathcal{D}^{test}|}, \tag{3.1}$$

where $\mathcal{U}^{test}$ denotes all users in the test set, $\mathcal{I}_u^{test}$ denotes the items rated by user u in the test set, $\mathbb{R}_{>0}$ denotes the set of real numbers which are greater than zero, $|\mathcal{D}^{test}|$ is the total number of rating records in the test set, and $\mathbf{1}_{\mathbb{R}_{>0}}(\check{r}_{i,u})$ is an indicator function, which is defined as:

$$\mathbf{1}_{\mathbb{R}_{>0}}(\check{r}_{i,u}) = \begin{cases} 1 & \text{if } \check{r}_{i,u} \in \mathbb{R}_{>0}, \\ 0 & \text{otherwise.} \end{cases}$$

3.3.5 Other metrics

3.3.5.1 Learning rate

Learning rate is usually a parameter in an iterative learning model. This is the performance as a function of the number of learning examples. Learning rate has not been

studied widely in the recommender systems literature and hence there is no well documented metric to report the results. Learning rates are non-linear (Herlocker et al., 2004), and are concerned with the performance of a recommendation algorithm against the available ratings to learn the model. Three different learning rates are (Herlocker et al., 2004) (1) overall learning rate—the recommendation quality as a function of overall ratings in the systems; (2) per-item learning rate—the recommendation quality for an item as a function of available ratings for that item; and (3) per-user learning rate—the recommendation quality for a user as a function of available ratings given by that user. A recommendation algorithm should produce robust and "acceptable" recommendations at different rates. The term acceptable is application-dependent; for example when a new user enters the system, it is highly desirable to give them quite accurate recommendations to build their trust in the system. The learning rate metric can be thought of as checking the performance of an algorithm under artificially created new user, new item, and sparse dataset. In a sense, if an algorithm produces consistently good performance under different (learning) conditions, then its learning rate is much higher than others.

3.3.5.2 Confidence in a prediction

Confidence is concerned with the assurance a recommendation algorithm has in a prediction to be accurate. The importance of the confidence metric becomes visible when we consider a recommender system as a part of a decision-support system—a tool which helps people to make the best possible decisions about what to buy or what to watch, by guiding them in a personalised way to interesting resources in a large space of possible resources. It has been claimed in Mcnee et al. (2003) that showing a confidence display along with recommendations can influence the user's decision making. Showing the best confidence display increases the user's *trust* in the system, while showing the worst confidence display worsens the decision making. It is worth noting that trust is a major factor which influences the user's decision (McNee, 2006).

Again, there is no standard metric to measure the confidence. High confidence implies that the corresponding recommendation should be accurate, which can be checked in the test set. The authors in Mcnee et al. (2003) and McNee (2006) used the number of ratings given by a user and to an item as a confidence measure, where the recommendations made by a few ratings were considered "risky". The problem with this approach is that this scheme is "non-personalised", measuring the same confidence for all users/items who have the same number of ratings in the system.

We show how confidence can effectively be measured by our Kernel Mapping Recommender (KMR) system algorithms (refer to Chapter 7). We take into account the variance in the output probability distribution of a prediction. We show through experiments that if we have less variance in the output probability distribution, then the

prediction is more accurate and vice versa. This variance in the output probability distribution can directly be mapped to confidence, where a low variance means high confidence and vice versa.

3.3.6 Evaluation from the user's point of view

In this thesis, we have not focused on the metrics which measure the performance of a recommendation algorithm from purely human-computer interaction theory point of view (McNee, 2006; Loizou, 2009). This approach requires user intervention in the evaluation process; for instance conducting online system surveys which are beyond the scope of this thesis.

3.4 Presenting Recommendations to Users

Recommendations can be presented to a user in the following two ways: by predicting ratings of items a user has not seen before and by constructing a list of items ordered by their preferences. In the former case, an active user provides the prediction engine with the list of items to be predicted; the prediction engine uses other users' (or items') ratings or content information, and then predicts how much the user would like the given item in some numeric or binary scale. In the latter case, different heuristics are used for producing an ordered list of items, sometimes termed as *top-N recommendations* (Sarwar et al., 2000b; Rashid et al., 2006). For example, in the collaborative filtering recommender system, this list is produced by making the rating predictions of all items that an active user has not yet rated, sorting the list, and then keeping the top-N items the active user would like the most. In this work, we have focused on both of these approaches.

3.5 Evaluation Methodology

We performed 5-fold cross validation by randomly dividing the dataset into a test and training set and reported the average results. We further subdivided our training set into a test and training set for measuring the parameters' sensitivity. For learning the parameters, we conducted k-fold (where $k = 2$ for the KMR algorithms, and 5 for remaining algorithms) cross validation on the training set. We show the average of the results with error-bars over 5-folds.

3.6 Feature Extraction

Information extraction techniques search for specific pieces of data in natural language documents and extract structured information (Cardie, 1997; Nahm and Mooney, 2002). By extracting information from a corpus of textual data, they construct a structured, searchable database, thus making data more easily accessible. We downloaded textual descriptions of each movie from IMDB. We then built items' profiles based on the textual description of items. There are two main techniques for building a user's profile as follows:

- A model of the user's preferences is built using the descriptions and types of the items they are interested in.
- A history of the user's interactions with the system is stored. The history of a user can be gathered by explicit feedback (e.g. their ratings) or implicit feedback (e.g. time spent in a web page).

We focused on both techniques where we use the explicit feedback (i.e. ratings) only for the second approach. Creating and learning a user profile is a form of classification problem, where training data can be divided into two categories: items liked by a user and items disliked by a user.

For the remainder of this chapter, we view each item as a text document, since an item's textual description can be thought as a text document. There are certain steps involved to get the features from a text document as discussed next.

3.6.1 Pre-processing

In the pre-processing step, documents, which typically are strings of characters, are transformed into a representation suitable for the machine learning algorithms. The documents are first converted into *tokens*, which are sequences of letters and digits, and then usually the following modifications are performed (Aas and Eikvil, 1999):

- HTML (and others) tags are removed
- Stop words are removed
- Stemming is performed

Stop words are frequently occurring words that carry little information. They have the following syntactical classes: conjunctions, articles, particles, prepositions, pronouns,

anomalous verbs, adjectives, and adverbs (Witten et al., 1999). We customised Google's stop word list (ranks.nl/resources/stopwords.html) for this task.

Stemming removes the case and inflections information from a word and maps it to the same stem. For example, the words *recommender*, *recommending*, *recommendation*, and *recommended* are all mapped to the same stem *recommend*. We used the Porter stemmer (Alag, October, 2008) algorithm for this task.

3.6.2 Indexing

Each document is usually represented by a vector of weighted *index terms*. A **Vector Space Model** is the most commonly used document representation technique, in which documents are represented by vectors of words. A *word-by-document matrix*, **A**, is used to represent a collection of documents, where each entry symbolises the occurrence of a word in a document,

$$\mathbf{A} = a_{w,d}. \tag{3.2}$$

In equation 3.2, $a_{w,d}$ is the weight of word w in document d. This matrix is typically very sparse, as not every word appears in every document.

Let n_d be the number of documents in a collection, n_w be the total number of words (after stop word removal and stemming) in the collection, $DF(w)$ be the number of times word w occurs in the whole collection, and $TF(w,d)$ be the frequency of word w in document d. Different approaches are used for determining the weight $a_{w,d}$ of word w in document d; for example, boolean weighting, word frequency weighting, TF-IDF weighting, and entropy weighting (Aas and Eikvil, 1999). We used the TF-IDF approach due to its simplicity and wide use in the literature (Joachims, 1998; Mooney and Roy, 2000)

Term Frequency-Inverse Document Frequency (TF-IDF) is a well-known approach that uses the frequency of a word in a document as well as in the collection of documents for computing weights. The weight $a_{w,d}$ of word w in document d is computed as a combination of $TF(w,d)$ and $IDF(w)$. Term Frequency, $TF(w,d)$, treats all words as equally important when it comes to assessing the relevance of a query. It is a potential problem, as in most of the cases, certain terms have little or no discriminating power in determining relevance. For example, a collection of documents relating to the *software* industry is likely to have the term *'software'* in almost every document. Hence, there is a need for a mechanism which attenuates the effect of frequently occurring terms in a collection of documents. Inverse Document Frequency, $IDF(w)$, is used for this purpose and is calculated from Document Frequency, $DF(w)$, as follows:

$$IDF(w) = \log\left(\frac{n_d}{DF(w)}\right). \tag{3.3}$$

Intuitively, the IDF of a word is high if it occurs in one document and is low otherwise. A composite weight $a_{w,d}$ for word w in document d is calculated by combining the TF and IDF as follows:

$$a_{w,d} = TF(w,d) \times IDF(w). \tag{3.4}$$

The TF-IDF approach does not take the length of document into account, which could be a problem in certain situations where documents have different lengths. We can eliminate this problem by normalising the weights:

$$a_{w,d} = \frac{TF(w,d) \times IDF(w)}{\sqrt{\sum_{j=1}^{n_w} \Big[TF(j,d) \times IDF(j)\Big]^2}}. \tag{3.5}$$

The indexing step leads to a *bag-of-words* representation of documents, which is equivalent to *attribute-value* representation in machine learning (Witten and Frank, 1999). We represent tokens as attributes and corresponding weights as values.

3.6.3 Dimensionality reduction techniques

The feature space in a typical attribute-value representation can be very large (there is one dimension for each unique word found in the collection of documents, after removing stop words and stemming.). In some cases, where we have a large number of documents, machine learning techniques cannot deal with this high dimensional matrix due to limited memory and processing power. The dimensionality reduction techniques overcome this problem by reducing the feature set without significantly sacrificing the information. In this way, the conventional learning methods can be used to improve the generalisation accuracy and to avoid over-fitting[1]. These techniques usually fall into two categories as follows:

1. **Feature Selection:** Feature selection process reduces the feature space by eliminating useless noise words—words having little (or no) discriminating power in a classifier, or having low signal-to-noise ratio. Several approaches are used for feature selection; for example, Document Frequency (DF) thresholding, information gain, χ^2, and mutual information gain. We used DF thresholding and χ^2 which are effective in reducing the dimensions without loss of accuracy (Sebastiani, 2002).

    - *DF thresholding:* This approach computes the Document Frequency (DF) for each word in the training set and removes words having DF value less than a predetermined threshold. The assumption behind this is that these

[1] Over-fitting is a problem with machine learning algorithms, where an algorithm becomes too specific to a dataset, and cannot be generalised to other datasets or domains.

rare words neither have the discriminating power for a category prediction nor do they influence the global performance.

- χ^2 *statistic:* This approach measures how independent word w and class C_j are:

$$\chi^2(w, C_j) = \frac{n_d \times (AD - CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)}. \tag{3.6}$$

Where A is the number of documents from class C_j that contain word w, B is the number of documents that contain word w but are not from class C_j, C is the number of documents from class C_j that do not contain word w, and D is the number of documents that neither contain word w nor are they from class C_j. This approach computes χ^2 for each word. The lower the χ^2, the more independent a word will be from a class. The words having the lowest value for χ^2 are removed because we are interested in words which are not independent from C_j (Sebastiani, 2002).

2. **Re-Parametrisation:** Re-Parametrisation process transforms the original features and constructs new ones. *Latent Semantic Indexing* (LSI) is a well-known technique used for this purpose. LSI assumes that words' usage across documents has some latent structure that can be estimated by using statistical techniques. LSI uses *Singular Value Decomposition* (SVD) which in turn uses factor analysis and eigenvector decomposition. We show in Chapter 5 how LSI can be helpful in reducing the dimensions of the feature space.

3.7 Building the Classification/Regression Approaches Based on Features

We trained the text categorisation approaches based on the features information to predict an unknown rating. It have been claimed that the text categorisation and recommender system share a number of characteristics (Zhang and Iyengar, 2002); for example high dimensions of the matrix, sparsity, etc. Zhang and Iyengar (2002) argue that each user can be viewed as a document and each item rated by a user can be represented by a word appearing in a document. Another approach using the content features of an item has been proposed in Mooney and Roy (2000), for a book recommender system. In this approach, each item was considered as a document represented by a vector of bags of words and a user's rating as one of the class labels. A Naive Bayes classification approach was used to learn a user' profile, from a set of movies the user have rated (i.e. labelled documents). This approach has been used by many other researchers such as Melville et al. (2002). We used this approach for building the classification and regression approaches. Next, we show how a text categorisation algorithm can be trained using the content features.

3.7.1 Training the model using the content features

Text categorisation is the process of automatically assigning one or more predefined categories to text documents (Sebastiani, 2002). To this end, let $\mathfrak{D}$ be the collection of document vectors, $\mathfrak{D}^{test} = \{ d'_1, \cdots, d'_n \}$ be the n document vectors to be classified, $C = \{C_1, \cdots, C_z\}$ be the z possible categories (classes), $\mathfrak{D}^{training} = \{ d_1, \cdots, d_m \}$ be the training set consisting of m document vectors with corresponding class labels $\{y_1, \cdots, y_m\}$, and $\mathcal{T}$ be a target concept $\mathcal{T} : \mathfrak{D} \rightarrow C$, which maps given documents to a class. We assume that each document is assigned to exactly one category. We use information contained in the training examples to find a model $\check{\mathcal{T}} : \mathfrak{D} \rightarrow C$, which approximates $\mathcal{T}$. The function $\check{\mathcal{T}}(d)$ defines the class to which the learned model assigns the document d and is used for classification of new documents. The objective here is to find a model which maximises the accuracy, i.e. assigns a new document to the most appropriate class.

In recommender system settings, we build a multi-class classifier for each user, where the number of classes are equal to the rating scale of the corresponding system. In the MovieLens dataset, we have 5 classes, whereas in the case of the FilmTrust dataset we have 8 classes (refer to Appendix A for the histogram of the FilmTrust dataset). A user's profile is learned from the movies in $\mathfrak{D}^{training}$. Specifically, for each user, the feature vectors consisting of *TF-IDF* weights are constructed against each class. A model can easily be trained over these feature vectors, which can classify any test movie, into one of the classes. We give more details about training a classifier in Chapter 4.

3.8 Demographic Information

The term "demographic" primarily refers to users' attributes that can be used to categorise users into different groups (Pazzani, 1999; Burke, 2002). Some researchers have claimed that items can also be categorised based on certain information such as genre vector, which can be termed as items' demographic information. This analogy is arguable, because genre information can be classified as the feature information of a movie. Regardless of the use of the term "genre" as a distinct feature or demographic information of an item, the same genre vector is used to generate recommendations. In our work, the term "demographic" information about a movie refers to the genre information about that movie.

To construct the demographic vector of items we used the hierarchy of genre as shown in Figure 3.1. To determine the weight of a genre in the genre vector, we used a simple weighting scheme as employed in QuickStep, an Ontology-based recommender system (Middleton et al., 2009). The main idea is that the immediate super class is assigned 50% of a subject's value, the next super class is assigned 25%, and so on until

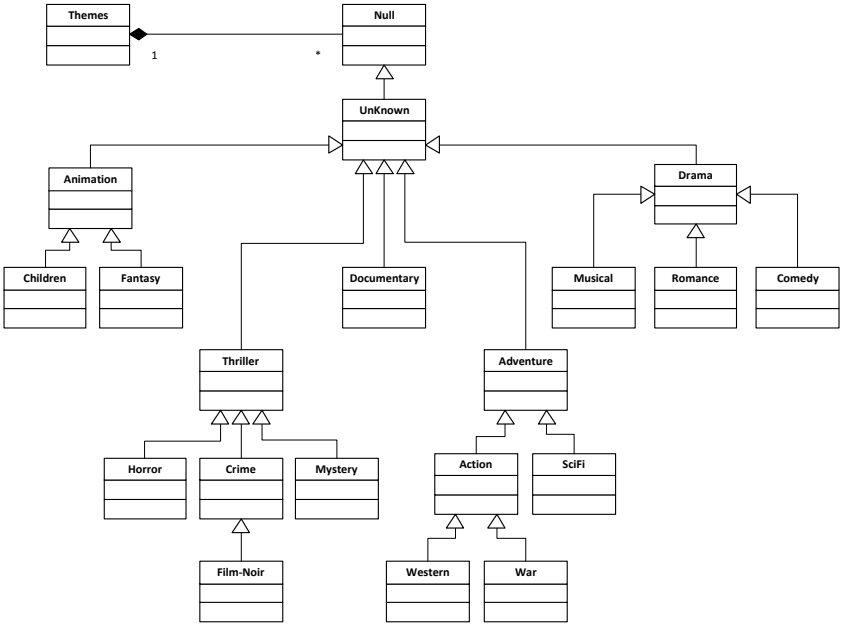


Figure 3.1: Hierarchy of genres modified from Schickel-Zuber and Faltings (2006). All the super classes of a genre get a share when a genre receives some interest. For instance if a rated movie falls into the "Crime" genre, then the "Crime" subject will get weight q, the immediate super class, the "Thriller" will get weight of $q/2$; the next super class "Unknown" will get a weight of $q/4$.

the most general subject in the Ontology is reached. By making a hierarchy of the genre and assigning different weights to sub and super classes, we hope to enrich an item's profile.

3.9 Summary

In this chapter, we give the details of our methodology. We discuss the characteristics of various datasets and shed light on the evaluation metrics that are used in this work. We provide justification for using these datasets and evaluation metrics. We discuss the types of information that are crawled from the IMDB against each movie. Then we discuss the steps in the feature extraction and selection algorithms that we have employed to extract and select features from the content information. We illustrate how these features are used to build the classification and regression approaches used in this work. Finally, we explain how the genre information about movies is used to build the demographic-based recommender systems.

Chapter 4

Switching Hybrid Recommender Systems

4.1 Introduction

Collaborative Filtering (CF) and Content-Based Filtering (CBF) recommender systems suffer from potential problems, such as sparsity, reduced coverage, cold-start, and over-specialisation, which reduce the effectiveness of these systems. Hybrid recommender systems combine individual recommendation approaches to overcome some of the aforementioned problems. In this chapter, we propose novel *switching hybrid recommendation algorithms* using classification approaches trained on the content profiles of users and item-based CF. A switching hybrid recommender system is intelligent in the sense that it can switch between recommendation approaches using some criteria. The benefit of a switching hybrid hybrid recommender is that it can make efficient use of strengths and weaknesses of its constitutional recommender systems. We show empirically that the proposed algorithms outperform (or give comparable results to) other recommender system algorithms in terms of the MAE, ROC-Sensitivity, and coverage; while at the same time eliminate some of the recorded problems with recommender systems. We evaluate our algorithm over the MovieLens (SML) and FilmTrust (FT1) datasets.

The rest of the chapter has been organised as follows. Section 4.2 discusses the related work. Section 4.3 presents some background concepts relating to the Naive Bayes and SVM classifiers. Section 4.4 outlines the proposed algorithms. Section 4.5 compares the performance of the proposed algorithms with others. Section 4.6 offers a variant of the proposed algorithms based on the singular value decomposition. Finally, Section 4.7 concludes and outlines the future work.

4.2 Related Work

A significant part of research in recommender systems concerns the techniques to combine the individual recommendation algorithms. A number of hybrid recommender systems have been proposed, a vast majority of which combines the collaborative filtering with content-based filtering (Balabanović and Shoham, 1997; Sarwar et al., 1998; Good et al., 1999; Claypool et al., 1999; Melville et al., 2002; Uchyigit and Clark, 2002; Li and Kim, 2003; Das et al., 2007; Barragáns-Martínez et al., 2010; Gemmell et al., 2010), while a few combine the collaborative filtering with the knowledge-based techniques (Burke, 1999; Tran and Cohen, 2000; Burke, 2002; Middleton et al., 2004) or demographic filtering (Pazzani, 1999; Vozalis and Margaritis, 2006b, 2007).

Based on Burke (2007)'s taxonomy of hybrid recommender systems, hybrid recommender systems can be categorised into the following seven classes: (1) *weighted*—where the score of a recommended item is computed by employing some weighting scheme to combine the results from all of the available recommendation techniques present in the system. Examples include Pazzani (1999) and Claypool et al. (1999); (2) *switching*—which can switch between the individual techniques using some switching criteria. Examples include Billsus and Pazzani (2000); (3) *mixed*—which presents recommendations from several different recommenders at the same time. Examples include Cotter and Smyth (2000) and McNee (2006); (4) *feature combination*—which augments the feature data associated with each example by adding collaborative information into them, and then uses content-based technique over this data set. Examples include Basu et al. (1998); (5) *cascade*—where a recommendation technique is applied to produce a coarse recommendation list of items that are refined by applying another recommendation technique. Examples include EntreeC (Burke, 2002); (6) *feature augmentation*—where one recommendation technique is applied to produce a rating or classification of an item and then a second recommendation technique incorporates that information. Examples include Melville et al. (2002) and McNee (2006); and (7) *meta-level*—where one recommendation technique is applied to generate a model, which is then given as an input to a second recommendation technique. Examples include Fab (Balabanović and Shoham, 1997).

Sarwar et al. (1998), Good et al. (1999) and Park et al. (2006) all proposed a (feature augmentation hybrid) scheme to combine the content-based filtering with CF by adding FilterBots or information filtering agents. For example, in Sarwar et al. (1998), the authors used simple agents, such as spell-checking, which analyse a new document in the news domain and rate it to reduce the sparsity of the dataset. These agents behave like users and can be correlated with the actual users. They claimed that the integration of information filtering agents with CF outperformed the simple CF in terms of accuracy. The problem with these approaches is that the recommendation quality would heavily depend on the training of individual agents, which may not be desired in certain cases, especially given limited resources.

Pazzani (1999) introduced a hybrid recommendation approach called "collaboration via content" in which a content profile of each user is used to find the similar users that are used for making predictions. The author used Winnow to extract features from users' home pages to build the content profile of users. The problem with this approach is that if the content profile of a user is erroneous (maybe due to synonyms problems or others), then it will result in poor recommendations. Furthermore, they proposed a consensus scheme to combine the predictions generated by CF, collaborative via content, and demographic approaches and claim that the combined approach outperformed the simple ones.

Another way of combining the different recommender systems has been presented in Vozalis and Margaritis (2006b, 2007). The authors applied Singular Value Decomposition (SVD) over the user-item rating matrix and items' demographic data and claimed that a linear hybrid recommender system consisting of item-based CF and demographic recommender system give more accurate results than the individual ones. A related example is given in Ghazanfar and Prügel-Bennett (2010e), where the authors used a linear combination of collaborative filtering, content-based filtering, and demographic recommenders and claimed that the combined version give more accurate results than the conventional hybrid recommender systems. Another example is the P-Tango system (Claypool et al., 1999), which uses the weighted average of collaborative filtering and content-based filtering recommender systems for news recommendations. The downside of these approaches is that they assume that the relative weight of different techniques is, more or less, uniform across the space of possible items, which is not true.

Commercial systems relying on the hybrid recommender systems have been proposed; for example, the Google news recommender system (Das et al., 2007) combines several approaches to produce scalable and real time recommendations. Specifically, it is a linear combination of collaborative filtering using clustering, probabilistic latent semantic indexing, and covisitation count. A personalised TV recommender system has been brought forward by Cotter and Smyth (2000). A mixed hybrid recommendation approach was proposed where CF was used to overcome over-specialisation problems and content-based filtering was used to overcome new item problems. Another example of a mixed hybrid TV recommender is proposed in Barragáns-Martínez et al. (2010), where the authors combined content-based filtering with collaborative filtering coupled with SVD. A user is given a list of top ranked items by employing some sort of combination technique.

Various hybrid recommender systems have been proposed using Ontology and CF (Mobasher et al., 2003; Middleton et al., 2004; Szomszor et al., 2007; Cantador et al., 2008; Weng and Chang, 2008) to overcome the sparsity problem of the user-item rating matrix. For example Mobasher et al. (2003) used domain-specific Ontologies to enhance the similarity between the items in the item-based CF. They linearly combine the similarities between items based on the user-item rating matrix and structure semantic knowledge

about items to generate recommendations, and claimed that this semantically enhanced approach outperforms the conventional item-based CF particularly given the sparse dataset. The problems with these approaches is that they require time-consuming knowledge engineering techniques to capture the domain-specific knowledge, which may not be pragmatic given millions of items that is common with e-commerce domains.

Various classification approaches have been employed for solving the recommendation problem, for example Billsus and Pazzani (1998); Basu et al. (1998); Mooney and Roy (2000); Zhang and Iyengar (2002); Melville et al. (2002). Hybrid recommender systems combining the classification technique with collaborative filtering have been proposed, for example, in Melville et al. (2002), the authors offered a hybrid recommender framework that combines collaborative filtering with a Naive Bayes classifier to recommend movies to users. The problem with this approach is that it is not very scalable (see section 4.5.2.3 for details). Another example is given in McNee (2006), where the author used a Naive Bayes classifier and collaborative filtering for research paper recommendations. He combined the individual recommenders by a mixed and a feature augmentation technique and claimed that the combined approach gives more accurate results than the individual ones.

In this work, we have focused on switching hybrid recommender systems. The literature specifically focusing on switching hybrid recommender systems includes news recommender systems (Billsus and Pazzani, 2000), recommender systems for e-commerce (Tran, 2007), case-based reasoning systems (Cheetham and Price, 2004), movies recommender system (Lekakos and Caravelas, 2008), and others (Nakagawa and Mobasher, 2003; Van Setten, 2005). For example, Tran (2007) proposes a strategy for top-N recommendations, which chooses the collaborative filtering approach as the main recommender and triggers the knowledge-based approach if the collaborative filtering cannot classify an item as good. An item is classified as good if its predicted value is more than a predefined threshold; for example, user average on certain group of items in the training set. The value of the threshold was changed based on the user behaviour. The problem with this approach is that knowledge-based approaches are computationally expensive.

Switching hybrid recommender systems differ in the selection of switching criteria; for example, some of them have used the confidence value in a recommendation component as a switching criteria (e.g. the similarity of a new case with the existing ones, where a high value of the similarity implies high confidence) (Billsus and Pazzani, 2000; Nakagawa and Mobasher, 2003; Cheetham and Price, 2004; Van Setten, 2005), while others have focused on some external criteria (e.g. Web site topology and the degree of connectivity in a Web personalisation system) (Nakagawa and Mobasher, 2003). Switching criteria can also choose the different implemented versions of a recommendation approach; for example in *News Dude* (Billsus and Pazzani, 2000), a news recommender system, the authors used different content-based filtering algorithms based on the short-term and long-term interest profiles of a user. They used a nearest neighbourhood approach to

recommend news stories based on users' short-term profiles and a Naive Bayes classifier to recommend news stories based on users' long-term profiles.

4.3 Background

In this section, we give an overview of the Naive Bayes and SVM classifiers. We explain how we use and modify them for building the switching hybrid recommender systems.

4.3.1 Naive Bayes classifier

The Naive Bayes classifier is based on the *Bayes theorem* with strong (Naive) independence assumption, and is suitable for the cases having high input dimensions. Using the Bayes theorem, the probability of a document d being in class C_j is calculated as follows:

$$P(C_j|d) = \frac{P(C_j)P(d|C_j)}{P(d)}, \tag{4.1}$$

where $P(C_j|d)$, $P(C_j)$, $P(d|C_j)$, and $P(d)$ are called the *posterior, prior, likelihood,* and *evidence* respectively.

The Naive assumption is that features are conditionally independent; for instance, in a document the occurrence of words (features) do not depend upon each other (Witten and Frank, 1999). Formally, if a document has a set of features $w_1, \cdots, w_h$ then we can express the numerator of equation 4.1 as follows:

$$\begin{aligned} P(C_j)P(d|C_j) &= P(C_j)P(w_1, \cdots, w_h|C_j) \\ &= P(C_j)P(w_1|C_j)P(w_2, \cdots, w_h|C_j) \\ &= P(C_j)P(w_1|C_j)P(w_2|C_j, w_1)P(w_3, \cdots, w_h|C_j) \\ &= P(C_j)P(w_1|C_j)P(w_2|C_j, w_1)\cdots P(w_h|C_j, w_1 \cdots, w_{h-1}). \end{aligned} \tag{4.2}$$

Now the naive assumption says that each feature w_i is conditionally independent of every other feature w_j for $j \neq i$, i.e. $P(w_i|C_j, w_j) = P(w_i|C_j)$. Hence, we can simplify equation 4.2 as follows:

$$\begin{aligned} P(C_j)P(d|C_j) &= P(C_j)P(w_1|C_j)P(w_2|C_j, w_1)\cdots P(w_h|C_j, w_1 \cdots, w_{h-1}) \\ &= P(C_j)P(w_1|C_j)P(w_2|C_j)P(w_3|C_j)\cdots \\ &= P(C_j)\prod_{x=1}^{h} P(w_x|C_j). \end{aligned} \tag{4.3}$$

After substituting equation 4.3 into 4.1, we have:

$$P(C_j|d) = \frac{P(C_j)\prod_{x=1}^{h} P(w_x|C_j)}{P(w_1, \cdots, w_h)}. \tag{4.4}$$

An estimate $\widehat{P}(C_j)$ for $P(C_j)$ can be calculated as:

$$\widehat{P}(C_j) = \frac{|\mathfrak{D}_j^{training}|}{|\mathfrak{D}^{training}|}, \tag{4.5}$$

where $|\mathfrak{D}_j^{training}|$ is the number of training documents that belongs to category C_j and $|\mathfrak{D}^{training}|$ is the total number of training documents. To classify a new document, Naive Bayes calculates posteriors for each class, and assigns the document to the class having the highest posterior.

In our case, we used the approach employed in Mooney and Roy (2000) for a book recommender system and in Melville et al. (2002) for a movie recommender system, with the exception that we used *DF thresholding feature selection* scheme for selecting the most relevant features. We assume we have z possible classes, i.e. $C = \{C_1, C_2, \cdots, C_z\}$, where $z = 5$ for the MovieLens dataset and $z = 8$ for the FilmTrust dataset. We have H types of information about a movie—keywords, tags, actors/actress, etc. (refer to Section 3.2). We constructed a vector of bags-of-words (Aas and Eikvil, 1999), d_t, against each type. The posterior probability of a movie, i, is calculated as follows:

$$P(C_j|i) = \frac{P(C_j)\prod_{t=1}^{H}\prod_{x=1}^{|d_t|} P(w_{tx}|C_j, H_t)}{P_i}, \tag{4.6}$$

where $P(w_{tx}|C_j, H_t)$ is the probability of a word w_{tx} (xth word in slot t) given class C_j and type H_t. We used Laplace smoothing (Witten and Frank, 1999) to avoid the zero probabilities and log probabilities to avoid underflow.

4.3.2 Support Vector Machines (SVM)

Support Vector Machines (SVM) are a set of related supervised learning methods with a special property in that they simultaneously minimise the empirical classification error and maximise the geometric margin; hence they are also known as *maximum margin classifiers*. SVM works well for classification and especially for text categorisation (Witten and Frank, 1999; Joachims, 1998). Joachims (1998) compared different text categorisation algorithms under the same experimental conditions, and showed that SVM perform better than conventional methods like Rocchio, decision tree, K-NN, etc. SVM work well for text categorisation problems, because (1) they can cope with high dimensional

input space, (2) they assume that the input space contains few irrelevant features, and (3) they are suited for problems with sparse instances.

If we consider a two-class, linearly separable classification problem we can have many decision boundaries. In SVM, the decision boundary should be as far away from the data of both classes as possible. The training of SVM tries to maximise the distance between the training samples of the two classes. The (binary class) SVM classifies a new vector d' into a class by a following decision rule:

$$\sum_{j=1}^{n_{sv}} \alpha_j y_j \mathbf{d}_j \mathbf{d}' + b, \tag{4.7}$$

where n_{sv} is the number of support vectors, α_j are the support vectors (which determines the decision boundary), $y_i \in \{ +1, -1 \}$ are the class labels, and d_j are the training vectors. This decision rule classifies d' as class +1 if the sum is positive and class -1 otherwise.

SVM can also handle the non-linear decision boundary using the *kernel trick* (Burges, 1998). The key idea is to transform the input space into a high dimensional feature space. After applying this transformation, the linear operation in the feature space becomes equivalent to a non-linear operation in the input space. Hence it reduces complexity and classification task becomes relatively easy. This transformation is denoted as:

$$\underline{\phi} : \mathcal{X} \mapsto \mathcal{F},$$

where $\mathcal{X}$ is the input space and $\mathcal{F}$ is the feature space. An example of polynomial kernel transformation is:

$$\underline{\phi}(x_1, x_2) \mapsto ({x_1}^2, {x_2}^2, \sqrt{x_1 x_2}, x_1, x_2, 1).$$

After the transformation, equation 4.7 can be written as follows:

$$\sum_{j=1}^{n_{sv}} \alpha_j y_j \underbrace{\underline{\phi}^T(\mathbf{d}_j)\underline{\phi}(\mathbf{d}')}_{K(\mathbf{d}_j, \mathbf{d}')} + b, \tag{4.8}$$

where $K(\mathbf{d}_j, \mathbf{d}') = \underline{\phi}^T(\mathbf{d}_j)\underline{\phi}(\mathbf{d}')$ is a kernel function. A kernel function is a symmetric positive semi-definite function of two variables. Many kernel functions can be used; for example, linear kernel, polynomial kernel, and radial basis kernel (Hsu et al., 2003).

For recommender system settings, vectors of features (i.e. words) consisting of *TF-IDF* weights, are constructed against each class. Like the Naive Bayes classifier, we have 5 classes for the MovieLens dataset and 8 classes for the FilmTrust dataset respectively. We normalised the data in the scale of $0 - 1$ and used LibSVM (Chang and Lin, 2011) for binary classification. We used linear kernel and trained the cost parameter C using the validation set. We used linear kernel rather than radial basis function (RBF), as

other researchers have found that if the number of features are very large compared to the number of instances, there is no significant benefit of using RBF over linear kernel (Hsu et al., 2003). Furthermore, parameters tuning in RBF and polynomial kernels is computationally intensive given a large feature size. For the multi class problem, several methods have been proposed, such as one-verse-one (1v1), one-verse-all (1vR), Directed Acyclic Graph (DAG) (Witten and Frank, 1999). We did not find any significant difference between the results obtained by 1v1, 1vR, and DAG; hence we used 1v1 for this work.

4.4 Combining the Item-based CF and Classification Approaches for Improved Recommendations

In this work, we have used classification techniques trained on the content profiles of users as a content-based filtering approach. Our framework[1] is based on the intuition that the classification approaches can accurately predict an unknown rating, if they have sufficient evidence for doing so. We use the classification approach as the main predictor in case we have sufficient evidence that the prediction made is correct and trigger the Item-Based Collaborative Filtering (IBCF) approach otherwise. We provide a simple generalised algorithm for combining the classification approach with the IBCF. We first show how the Naive Bayes classification approach can be combined with the IBCF, and then show how the SVM (or any other) classification approach can be combined with the IBCF.

4.4.1 Combining the item-based CF and the Naive Bayes classifier ($SwitchRec^{NB}_{CF}$)

The basic idea is to use the prediction computed by the Naive Bayes classifier if we have sufficient confidence in the Naive Bayes's prediction; otherwise, the prediction computed by the item-based CF is used. We propose a simple approach for determining the confidence in the Naive Bayes's prediction.

Let $\check{r}^{nb}_{i,u}$, $\check{r}^{cf}_{i,u}$, and $\check{r}^{final}_{i,u}$ represent the predictions generated by the Naive Bayes classifier, item-based CF, and the prediction we are confident will be accurate. Let $Pr(C_j)$ be the posterior probability of class j computed by the Naive Bayes classifier, L be a list containing the posterior probabilities of each class, and diff(i, j) be the absolute difference between the posterior probabilities of class i and j, i.e. diff(i, j) $= |L(i) - L(j)| = |Pr(C_i) - Pr(C_j)|$ where $i \neq j$. The proposed hybrid approach is outlined in Algorithm 1. Steps 2 to 5 represent the case, where the IBCF fails to make a prediction or where very few users have rated the target item. This can happen under the new item cold-start

[1] Appendix A gives details of packages and libraries used for the implementation.

Algorithm 1 *SwitchRec*$^{NB}_{CF}$; Combines the IBCF and the NB classifier
Input: $\check{r}^{nb}_{i,u}$, NB's prediction; $\check{r}^{cf}_{i,u}$, IBCF's prediction; L, a list containing posterior probabilities of each class; $|\mathcal{U}_i|$, the number of users who have rated the target item
Output: $\check{r}^{final}_{i,u}$, Final prediction

procedure SwitchRec($\check{r}^{cf}_{i,u}$, $\check{r}^{nb}_{i,u}$, L, $|\mathcal{U}_i|$)
  if ($\check{r}^{cf}_{i,u} = \emptyset$) OR $|\mathcal{U}_i| < \nu$ **then**
    $\check{r}^{final}_{i,u} \leftarrow \check{r}^{nb}_{i,u}$
    return $\check{r}^{final}_{i,u}$
  end if
  Sort the list L in ascending order, so that $L(1)$ contains the lowest value and $L(z)$ contains the highest value.
  if ($L(z) \neq L(z-1)$) **then**
    if diff$(z, z-1) > \delta$ **then**
      $\check{r}^{final}_{i,u} \leftarrow \check{r}^{nb}_{i,u}$
      return $\check{r}^{final}_{i,u}$
    else
      if ($|\check{r}^{nb}_{i,u} - \check{r}^{cf}_{i,u}| < \lambda$) **then**
        $\check{r}^{final}_{i,u} \leftarrow \check{r}^{nb}_{i,u}$
        return $\check{r}^{final}_{i,u}$
      end if
    end if
  else (i.e. $L(z) = L(z-1)$)
    for $t \leftarrow z-1, 1$ **do**
      if ($L(z) = L(t)$) **then**
        if ($|\check{r}^{cf}_{i,u} - t| < \lambda$) **then**
          $\check{r}^{final}_{i,u} \leftarrow t$
          return $\check{r}^{final}_{i,u}$
        end if
      else
        Break **for**
      end if
    end for
  end if
  $\check{r}^{final}_{i,u} \leftarrow \check{r}^{cf}_{i,u}$
  return $\check{r}^{final}_{i,u}$
end procedure

scenario. As the prediction quality of the IBCF depends heavily on the available data, it would suffer under this scenario. In this case, we use the prediction computed by the Naive Bayes classifier. Steps 7 to 16 determine the confidence in the Naive Bayes's prediction. The confidence in the Naive Bayes's prediction is high when the posterior probability of the predicted class is sufficiently larger than the others. If diff$(z, z-1)$ is sufficiently large, then we can assume that the actual value of an unknown rating has been predicted. The parameter δ represents this difference and can be found empirically over the validation set. The parameter λ tells us if the difference between the predictions made by the individual recommender systems is small; then again we are confident that the Naive Bayes is able to predict a rating correctly. This is a kind of heuristic learned from the prediction behaviour of the IBCF and the Naive Bayes. The IBCF gives predictions in a floating point scale, and the Naive Bayes gives predictions in an integer point scale. The IBCF recommender systems typically give accurate recommendation, but mostly they do not predict actual value; for example, if the actual value of an unknown rating is 4, then the IBCF's prediction might be 3.9 (or 4.1, or some other value). On the other hand, the Naive Bayes can predict the actual value; for example, in the aforementioned case, it might result in 4. However, if the Naive Bayes is not very confident, then it might result in a prediction that is not close to the actual one, e.g. 3, 2, etc. We take the difference between the individual recommender's predictions, and if it is less than a threshold (λ), then we use the Naive Bayes's prediction, assuming that it has been predicted correctly. Steps 17 to 28 represent the case, where we have tie cases in the posterior probabilities of some classes. In this scenario, we take the difference of each tie class with the IBCF's prediction and use that class as the final prediction, if the difference is less than λ. Steps 29 to 30 describe the case where we do not have sufficient trust in the Naive Bayes's prediction, so we use the prediction made by the IBCF.

4.4.2 Combining the item-Based CF and the SVM classifier ($SwitchRec_{CF}^{SVM}$)

Algorithm 1 can be used to combine the item-based CF and the SVM classifier. The methodology is the same, except that $Pr(C_j)$ represents the SVM's estimated probability for the class j. Similarly any other classifier can be combined with the collaborative filtering.

4.5 Results and Discussion

4.5.1 Learning the optimal system parameters

The purpose of these experiments is to determine which of the parameters affects the prediction quality of the proposed algorithms, and to determine their optimal values.

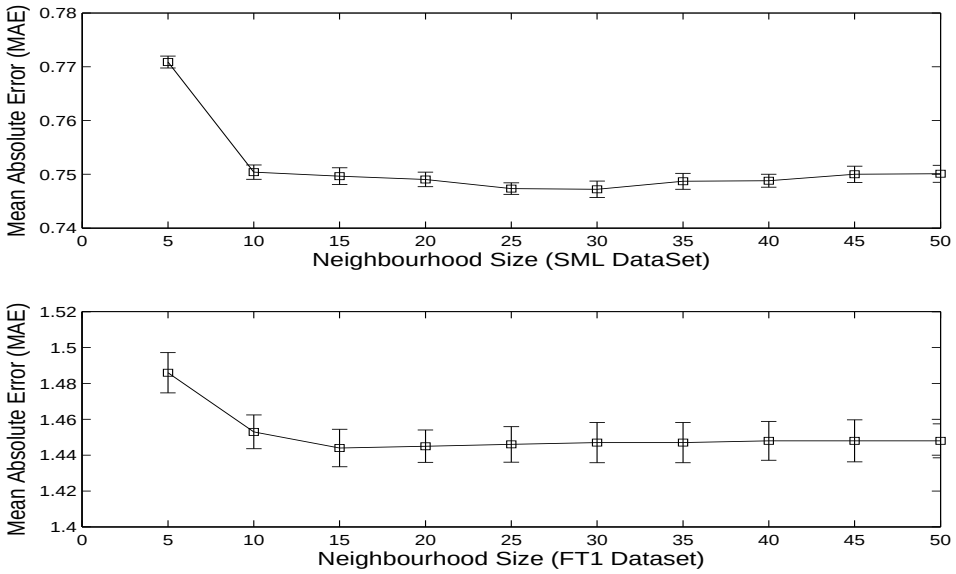


Figure 4.1: Determining the optimal value of neighbourhood size (l) for the MovieLens (SML) and FilmTrust (FT1) datasets over the validation set.

We describe the tuning of the important parameters. The tuning of other parameters is given in Appendix A.

4.5.1.1 Finding the optimal number of neighbours (l) in the item-based CF

To measure the optimal number of neighbours, we changed the neighbourhood size from 5 to 50 with a difference of 5, and observed the corresponding MAE. Figure 4.1 shows that the MAE decreases in general with an increase in the neighbourhood size. This is in contrast with the conventional item-based CF proposed in Sarwar et al. (2001), where the MAE is minimum for a small neighbourhood size (< 10 for the SML dataset) and then starts increasing as the neighbourhood size increases. The reason is that Sarwar et al. (2001) did not use any significance weighting scheme and used the weighted sum prediction generation formula, whereas we are using a significance weighting scheme and adjusted weighted sum prediction generation formula[2]. Figure 4.1 shows that the MAE keeps on decreasing with the increase in the number of neighbours, reaches its minimum for $l = 25$ for the SML dataset and $l = 15$ for the FT1 dataset, and then either starts increasing (although the increment is very small) or stays constant. For the subsequent experiments, we choose $l = 25$ for the SML and $l = 15$ for the FT1 dataset as the optimal neighbourhood size.

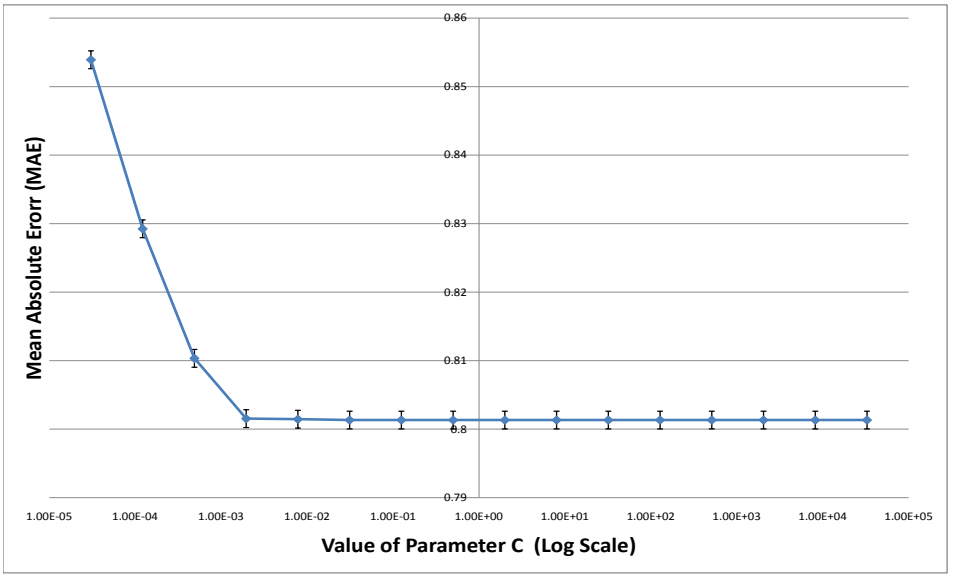


Figure 4.2: Determining the optimal value of parameter C for the SVM (SML dataset). X-axis shows the value of C in log scale. The corresponding MAE is shown on y-axis. The MAE decreases with an increase in the value of C, and becomes stable after $C = 2^{-9}$ (1×10^{-3}).

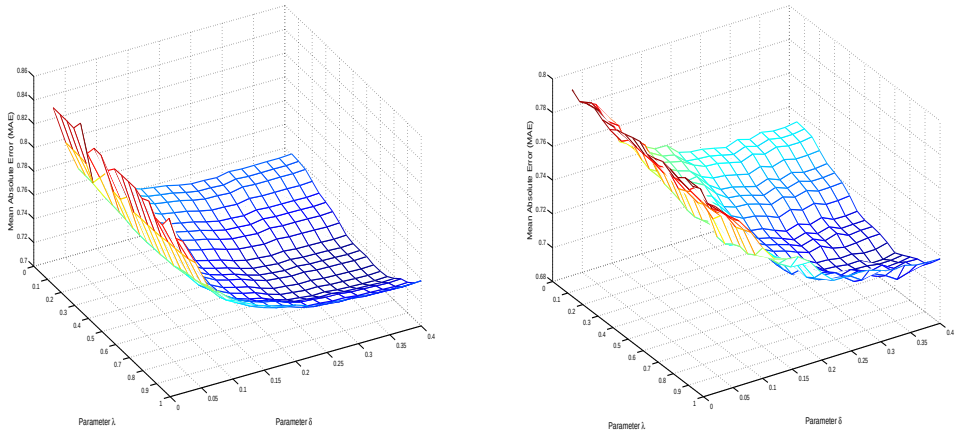
4.5.1.2 Finding the optimal value of C for the SVM classifier

The cost parameter, C, controls the trade-off between permitting training errors and forcing rigid margins. It allows some misclassification by creating soft margins. A more accurate model can be created by increasing the value of C that increases the cost of misclassification; however, the resulting model may over-fit. Similarly, a small value of C may under-fit the model. Figure 4.2 shows how the MAE varies with a change in the value of C. We changed the value of C from 2^{-15} to 2^{15} by increasing the power by 2. Figure 4.2 shows that the MAE is large for the small value of C, which may be due to under-fitting. The MAE decreases with the increase in the value of C, reaches its minimum for $C = 2^{-9}$, and then becomes stable for $C > 2^{-9}$ (between 1×10^{-3} and $1 \times 10^{+5}$ in log scale). We choose $C = 2$ for the MovieLens dataset to avoid any over-fitting and under-fitting of the model. The FilmTrust dataset shows similar results (not shown); hence we choose $C = 2$ as an optimal value for the FilmTrust dataset.

4.5.1.3 Finding the optimal values of δ and λ

For the MovieLens dataset, we performed a series of experiments by changing the value of δ from 0.02 to 0.4 with a difference of 0.02. For each experiment, we changed the value of λ from 0.05 to 1.0 with a difference of 0.05, keeping the δ parameter fixed, and observed the corresponding MAE. The grid coordinates giving the lowest MAE are recorded to be the optimal parameters. Figure 4.3(a) shows how the MAE changes with

[2]The details are not in the scope of this work; please refer to Ghazanfar and Prügel-Bennett (2010d).



(a) $SwitchRec_{CF}^{NB}$ (b) $SwitchRec_{CF}^{SVM}$

Figure 4.3: Finding the optimal values of δ and λ in the $SwitchRec_{CF}^{NB}$ and $SwitchRec_{CF}^{SVM}$, through grid search (SML dataset).

these parameters in the case of $SwitchRec_{CF}^{NB}$ algorithm. Figure 4.3(a) shows that the MAE decreases with an increase in the value of δ, reaches its peak at $\delta = 0.34$, and after that it either increases or stays constant. We note (keeping $\delta = 0.34$) that the MAE is minimum between $\lambda = 0.70$ to $\lambda = 0.80$. The grid coordinates $\{\delta, \lambda\}$, which gave the lowest MAE, are found to be $\{0.34, 0.70\}$. Similarly, we tuned these parameters for the FilmTrust dataset, which are found to be $\{0.20, 0.90\}$.

Figure 4.3(b) shows how the MAE varies with these parameters in the case of $SwitchRec_{CF}^{SVM}$ algorithm. Figure 4.3(b) shows that the MAE decreases with an increase in the value of δ, reaches its peak at $\delta = 0.36$, and after that it either increases or stays constant. We note (keeping $\delta = 0.36$) that the MAE is minimum at $\lambda = 0.70$. Considering the results, we choose the optimal value of $\{\delta, \lambda\}$ to be $\{0.36,0.70\}$. For the FilmTrust dataset, the optimal values of parameters are found to be $\{0.20,0.85\}$.

4.5.2 Performance evaluation with other algorithms

We compared our algorithm with seven different algorithms: user-based CF using Pearson correlation with default voting ($UBCF_{DV}$) proposed in Breese et al. (1998), item-based CF (IBCF) using Adjusted cosine similarity proposed in Sarwar et al. (2001), a Naive Bayes classification approach (NB) using item content information, a SVM classification approach using item content information, two naive hybrid approaches (NBIBCF, SVMIBCF) by taking the average of the predictions generated by the NB and the item-based CF, and the SVM and the item-based CF, and the content-boosted algorithm (*cBoosted*) proposed in Melville et al. (2002).

We are more interested to compare our algorithm $SwitchRec_{CF}^{NB}$ with a well-known *cBoosted* algorithm because both of them use the collaborative filtering and Naive Bayes classifier, although in different ways. Furthermore, we tuned all algorithms for the best parameters.

Table 4.1: A comparison of the proposed algorithms with others in terms of accuracy metrics and coverage. $IBCF_{SW}$ represents the item-based CF (IBCF) with significance weights applied over the rating similarities. The best results are shown in bold font.

Algorithm	**On-line Cost**	**Best MAE**		**ROC-Sensitivity**		**Coverage**	
		SML	**FT1**	**SML**	**FT1**	**SML**	**FT1**
$UBCF_{DV}$	$O(M^2N) + O(NM)$	0.746 ± 0.001	1.462 ± 0.008	0.714 ± 0.004	0.502 ± 0.008	**100**	99.981 ± 0.006
IBCF	$O(N^2)$	0.764 ± 0.001	1.449 ± 0.007	0.654 ± 0.003	0.534 ± 0.007	99.867 ± 0.011	95.262 ± 0.008
$IBCF_{SW}$	$O(N^2)$	0.744 ± 0.001	1.433 ± 0.007	0.753 ± 0.002	0.540 ± 0.006	99.867 ± 0.011	95.262 ± 0.008
$SwitchRec_{CF}^{NB}$	$O(N^2) + O(Nn_w)$	0.704 ± 0.002	1.398 ± 0.008	0.785 ± 0.002	0.544 ± 0.007	**100**	99.990 ± 0.007
$SwitchRec_{CF}^{SVM}$	$O(N^2) + O(Nn_{sv})$	$\mathbf{0.701 \pm 0.002}$	$\mathbf{1.392 \pm 0.006}$	$\mathbf{0.793 \pm 0.002}$	$\mathbf{0.548 \pm 0.007}$	**100**	99.990 ± 0.007
NB	$O(Nn_w)$	0.815 ± 0.003	1.471 ± 0.009	0.685 ± 0.006	0.512 ± 0.011	**100**	99.990 ± 0.007
SVM	$O(Nn_{sv})$	0.779 ± 0.003	1.463 ± 0.009	0.687 ± 0.004	0.513 ± 0.010	**100**	99.990 ± 0.007
NBIBCF	$O(N^2) + O(Nn_w)$	0.768 ± 0.003	1.458 ± 0.008	0.717 ± 0.004	0.526 ± 0.008	**100**	99.990 ± 0.007
SVMIBCF	$O(N^2) + O(Nn_{sv})$	0.759 ± 0.002	1.445 ± 0.008	0.723 ± 0.004	0.534 ± 0.008	**100**	99.990 ± 0.007
cBoosted	$O(M^2N) + O(NM)$ $+O(Nn_w)$	0.711 ± 0.002	1.412 ± 0.006	0.748 ± 0.003	0.539 ± 0.006	**100**	$\mathbf{99.994 \pm 0.006}$

Table 4.2: Performance evaluation under new item cold-start problem. We observe that the proposed algorithms produce more accurate results than the conventional ones. The $SwitchRec_{CF}^{SVM}$ algorithm produces the best results shown in bold font.

Algo.	**MAE**2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
$UBCF_{DV}$	0.925 ± 0.020	0.887 ± 0.011	0.860 ± 0.006	0.844 ± 0.005	0.835 ± 0.005
$IBCF_{SW}$	0.884 ± 0.022	0.877 ± 0.007	0.875 ± 0.007	0.873 ± 0.009	0.868 ± 0.005
cBoosted	0.831 ± 0.018	0.819 ± 0.009	0.816 ± 0.006	0.812 ± 0.006	0.804 ± 0.004
$SwitchRec_{CF}^{NB}$	0.806 ± 0.016	0.806 ± 0.013	0.808 ± 0.009	0.806 ± 0.005	0.805 ± 0.004
$SwitchRec_{CF}^{SVM}$	$\mathbf{0.777 \pm 0.013}$	$\mathbf{0.777 \pm 0.011}$	$\mathbf{0.776 \pm 0.007}$	$\mathbf{0.775 \pm 0.004}$	$\mathbf{0.775 \pm 0.004}$

4.5.2.1 Performance evaluation in terms of MAE, ROC-Sensitivity, and coverage

The MAE, ROC-sensitivity, and coverage of different algorithms are shown in Table 4.1. Table 4.1 shows that the proposed algorithms outperform others significantly in terms of the MAE and ROC-sensitivity; whereas they give comparable results to others in terms of coverage metric. The percentage decrease in MAE, in the case of $SwitchRec_{CF}^{NB}$ over the NBIBCF, is found to be 8.33% and 4.11% for the SML and FT1 dataset respectively. The percentage decrease in MAE, in the case of $SwitchRec_{CF}^{SVM}$ over the SVMIBCF is found to be 7.64% and 3.67% for the SML and FT1 datasets respectively.

Table 4.1 shows that the proposed algorithms outperform the *cBoosted* algorithm in terms of MAE and ROC-sensitivity; however the coverage of the *cBoosted* algorithm is better than the proposed ones for the FT1 dataset. The percentage improvement over the *cBoosted* algorithm in terms of MAE is found to be (1) 0.98% and 1.0% for the SML and FT1 datasets respectively in the case of $SwitchRec_{CF}^{NB}$, and (2) 1.41% and 1.42% for the SML and FT1 datasets respectively in the case of $SwitchRec_{CF}^{SVM}$.

It is worth noting that for the FilmTrust dataset, the ROC-sensitivity is lower, for all algorithms in general, as compared to the MovieLens dataset. We believe that this is due to the rating distribution—in FilmTrust, the majority of the users have rated the popular set of movies and their rating tends to match the average rating of the movies. Furthermore, the coverage of the algorithms is much lower in the case of the FilmTrust dataset, which is due to the reason that it is very sparse (98.8%).

4.5.2.2 Performance evaluation under cold-start scenarios

We checked the performance of different algorithms under the new item cold-start scenario. When an item is rated by only a few users, then item- and user-based CF will not give good results. Our proposed scheme works well in the new item cold-start scenario, as it does not depend solely on the number of users who have rated the target item for finding the similarity.

For testing our algorithm in this scenario, we selected 100 random items. While making prediction for the target item, the number of users in the training set who have rated the target item were kept 2, 5, 10, 15, and 20. The corresponding *MAE*, represented by $MAE2$, $MAE5$, $MAE10$, $MAE15$, and $MAE20$, is shown in Table 4.2. Table 4.2 shows that the CF approaches give inaccurate predictions. The poor performance of user-based CF is due to the reason that we have less neighbours against an active user, hence performance degrades. The reason in the case of item-based CF is that the similarity computed between two items is not reliable. As, while finding similarity, we isolate all users who have rated both target item and the item we are finding similarity with. In

this case, we have very few users who have rated both items; as a result, similarity found by adjusted cosine measure will be misleading. Both *cBoosted* and the proposed approaches give good results as they make effective use of a user's content profile that can be used by a classifier for making predictions.

4.5.2.3 Performance evaluation in terms of cost

Table 4.1 shows the on-line cost[3] of different algorithms used in this work. Here, N, M, n_w, and n_{sv} represent the number of items, number of users, number of features/words in the dictionary (used in a classifier), and the number of support vectors in the case of the SVM classifier. The training computation complexity of the SVM and NB classifiers for one user are $O(N^3)$ and $O(Nn_w)$ respectively. We train M classifiers, so total training computation complexity becomes $O(MN^3)$ for the SVM and $O(MNn_w)$ for the NB. The classifying computation complexity for one sample (rating) is $O(n_{sv})$ for the SVM and $O(n_w)$ for the NB. If we classify N items then it becomes $O(Nn_{sv})$ for the SVM and $O(Nn_w)$ for the NB.

Table 4.1 shows that the proposed algorithms are scalable and practical as their on-line cost is less than or equal to the cost of other algorithms. We are using the item-based CF, whose on-line cost is less than that of the user-based CF used in Melville et al. (2002)[4]. Even if we consider using the Naive Bayes classifier to fill the user-item rating matrix and then apply the item-based CF over this filled matrix, our cost will be less than that. The reason is, in the filled matrix case, one has to go through all the filled rows of the matrix for finding the similar items. For a large e-commerce system like Amazon, where we already have millions of neighbours against an active user/item, filling the matrix and then going through all the users/items to find the similar users/items is not practical due to limited memory and other constraints on the execution time of the recommender system.

4.5.3 Eliminating over-specialisation problem

Content-based filtering recommender systems recommend items that are the most similar to a user's profile. In this way, a user cannot find recommendations that are different from the ones it has already rated or seen. The proposed algorithms can overcome the over-specialisation problem caused by the pure content-based filtering recommender systems. The reason is that they do not totally depend on classification algorithms trained on the content information. By switching between the machine learning classifiers and

[3]It is the cost for generating predictions for N items. We assume that we compute item similarities and train classifiers in off-line fashion.

[4]It is because we can build expensive and less volatile item similarity model in off-line fashion. Hence on-line cost becomes $O(N^2)$ in worst case, and in practice it is $O(lN)$, where l is the number of top l most similar items against a target item ($l < N$).

the CF approach, our algorithms can balance the accuracy and diversity (Herlocker et al., 2004) of recommendations. If we construct a list of top-N recommendations for an active user, then our algorithms would introduce some sort of randomness in the recommendation list, resulting in a range of alternatives to be recommended rather than a homogeneous set of items.

4.6 Variant of the Proposed Algorithms

We investigated whether we could improve the performance of proposed algorithms by reducing the dimensions of the data matrix. our intuition being that the resulting system would become scalable without sacrificing any performance. We applied Singular Value Decomposition (SVD) over the user-item rating matrix and then used the item-based CF to generate recommendations (refer to the next chapter for the steps involved in applying the item-based CF over the reduced matrix).

The results of the item-based CF over the reduced dataset were rather surprising. For the FilmTrust dataset, the performance of the item-based CF applied over the reduced dataset degraded. It might be due to the reason that the FilmTrust dataset is very sparse. In the next chapter, we will analyse how sparsity affects the performance of the SVD-based recommender systems.

4.7 Conclusion and Future Work

In this chapter, we propose switching hybrid recommendation algorithms by combining the item-based Collaborative Filtering (CF) with classification approaches trained on the content profiles of users. We empirically show that our recommendation algorithms give more accurate results than the conventional hybrid approaches. They also outperform (or give comparable results to) the well-known content-boosted algorithm (Melville et al., 2002). Furthermore, they maintain robust performance under the cold-start scenarios.

As a future work, we would like to use over sampling and under sampling (Witten and Frank, 1999) schemes in classifiers to overcome the imbalanced dataset problem. Moreover, feature selection algorithms such as singular value decomposition can be applied to remove the useless features, which might increase the performance of the classifiers.

In this work, we have used a simple approach by taking the difference between the posterior probabilities as the measure of uncertainty in the prediction computed by a classifier. Other techniques to measure the uncertainty can be used; for example, entropy measure, the mean square difference, etc. Using these techniques might further increase the performance of the proposed algorithms. Furthermore, an aggregate confidence measure might give better results than the single confidence measure.

Finally, individual predictions computed by the user- and item-based CF can be combined in switching hybrid way. We hope that combining these two approaches will result in an increase in accuracy, as both of them focus on different kinds of relationship. Combining these approaches with classification ones can further increase the performance of the proposed algorithms.

Chapter 5

Exploiting Imputation in Singular Value Decomposition-Based Recommender Systems

5.1 Introduction

Singular Value Decomposition (SVD)-based approaches have been proposed to solve the recommendation problem (Sarwar et al., 2000b, 2002a; Barragáns-Martínez et al., 2010); however, approximating the missing values in the user-item rating matrix by the item average prior to applying SVD, which has been heavily used in the literature, is not a reasonable approach. It can lead to poor quality recommendations especially under cold-start and sparse scenarios. We proposed various missing value imputation methods, which exhibited much superior accuracy and performance compared to the traditional missing value imputation method - item average.

We performed extensive experiments over the SML (MovieLens 100K ratings dataset), ML (MovieLens 1M ratings dataset), FT1 (FilmTrust original dataset), and FT5 (FilmTrust filtered dataset) datasets under different conditions. Our empirical study shows that the results are dataset-dependent; however, rather than using the traditional approach to approximate the missing values or merely ignoring the missing values, robust and advanced approaches can provide considerable performance benefits in the (1) SVD-based recommendations, (2) SVD applied in the Expectation Maximisation (EM) fashion, (3) SVD-based CF (CF applied over the dataset reduced by employing SVD), and (4) cold-start, long tail, and sparse scenarios.

The rest of the chapter has been organised as follows. Section 5.2 discusses the related work in detail. Section 5.3 sheds light on the background concepts related to SVD. Section 5.4 outlines SVD algorithms used for recommendations. Section 5.5 describes

the proposed approaches used to approximate the missing values in the sparse user-item rating matrix. Section 5.6 presents results comparing the performance of the proposed approaches with the traditional one. Section 5.7 discusses when and how much imputation is sufficient to achieve good performance. Section 5.8 gives a discussion of the work, and Section 5.9 concludes the work.

5.2 Related Work

The Singular Value Decomposition (SVD)-based approach for solving the recommendation problem was first introduced by Billsus and Pazzani (1998). Sarwar et al. (2000b) presented a detailed analysis of the behaviour of SVD-based recommender systems. Various algorithms combining the SVD-based approach with the item-based CF have been advocated (Vozalis and Margaritis, 2005, 2006a; Martinez et al., 2009; Barragáns-Martínez et al., 2010); for example, Vozalis and Margaritis (2006a) combined SVD with the item-based CF and claimed that their approach outperformed the conventional item-based CF. An example of using the SVD-based approach with demographic data has been presented in Vozalis and Margaritis (2007), where the authors applied SVD over the user-item rating matrix and demographic data of users and items, and claimed that a system consisting of a linear combination of SVD-based demographic correlation and SVD-based (item-based) CF, increases the accuracy of the recommender system. Sarwar et al. (2002a) suggested an incremental SVD model building approach and claimed that it is more scalable than the conventional SVD-based recommender systems, while producing recommendations with same accuracy. All of the aforementioned approaches used the item average of the data matrix to approximate the missing values, which may destroy the covariance structure of the data, resulting in inaccurate recommendations.

Another way of applying SVD is presented in Goldberg et al. (2001), where the authors applied Principal Component Analysis (PCA)[1] over a so-called 'gauge-set' of items—set of items rated by every user in the system. Although it may reduce sparsity, getting this dataset is hard in real-life scenarios, and also it may lead to potential loss of information as we are ignoring ratings not in the gauge-set. Sometimes, case deletion strategy (Kim and Yum, 2005) is used for dealing with the missing values where all variables with missing values are omitted in the data matrix resulting in loss of information, which is not desirable. Some other well-known approaches to approximate the missing values are filling by zero and scaling the known entries as suggested by Azar et al. (2001).

The missing values have been handled by the Expectation Maximisation (EM) algorithm (Do and Batzoglou, 2008) by Canny (2002), Srebro and Jaakkola (2003), Zhang et al. (2005), Kim and Yum (2005), Zhang et al. (2006) and Kurucz et al. (2007). In this

[1] PCA is a closely related concept to SVD, which reduces the dimensionality by projecting high dimensional data along a smaller number of orthogonal dimensions.

approach, the predictions generated by the current model are replaced by the previous one and the procedure is repeated until some stopping criteria are reached; for example, the error between two successive models becomes less than a threshold. The problem with this approach is that the final error and the convergence is highly dependent on the method used to approximate the initial values. Srebro and Jaakkola (2003) showed the convergence behaviour of the EM algorithm by approximating the missing entries by zeros (Azar et al., 2001) and using the gauge-set (Goldberg et al., 2001). Furthermore, they proposed an approach by starting with a large rank approximation and gradually reducing the rank of SVD in each iteration of EM.

The most similar work with ours is that undertaken by Kurucz et al. (2007), where the authors used an item-item imputation technique in addition to the user-average over the Netflix dataset. Our work, however, differs from theirs in a number of areas, as follows: (1) they only used an item-item imputation while we are using 17 different approaches to analyse the behaviour of SVD and EM algorithms; (2) they only used one dataset; however, we are using three different datasets and furthermore, we find out that the results are highly dataset-dependent; (3) they claimed that the item-item imputation scheme is outperformed by the average, which is in contrast with our findings; and (4) we are applying CF over the reduced dataset; however, they did not apply it. In summary, their focus was on the efficient implementation of Lanczos, power iteration, and other algorithms rather than imputation; however, we are analysing the behaviour of the SVD-based algorithms under different recommender system conditions—cold-start, long tail, and sparsity problems.

Various matrix factorisation techniques, such as Srebro et al. (2005), Bell et al. (2007), Wu (2007), Takács et al. (2008), Salakhutdinov and Mnih (2008) and Takács et al. (2009) have been proposed to solve the recommendation problem. In this work, we have focused on the SVD-based recommender systems (Sarwar et al., 2000b; Vozalis and Margaritis, 2007; Kurucz et al., 2007; Martinez et al., 2009; Barragáns-Martínez et al., 2010) rather than matrix factorisation techniques.

The imputation has been used in collaborative filtering domain. The idea of using imputation in the collaborative filtering domain was proposed by Breese et al. (1998), where the authors used some default votes to decrease the sparsity of the user-item rating matrix. The author claimed that using the default votes in the user-based CF outperforms the conventional user-based CF in terms of accuracy. This idea has further been used by many researchers in various ways to approximate the missing values in the user-item rating matrix; for example, Melville et al. (2002) used a Naive Bayes classifier trained on the content profiles of users, Good et al. (1999) and Park et al. (2006) used information filtering agents or "Filterbot", Ma et al. (2007) used a linear combination of user- and item-based CF, Zhang and Pu (2007) used a recursive CF algorithm, and Su et al. (2008b,a) used several methods. The problem with these approaches is that they are not very scalable (refer to previous chapter). Our approach is different from these

because we are doing imputation in the SVD domain and CF is applied over the dataset reduced by employing SVD. Furthermore, imputation has been used in other domains; for example, for Epistatic miniarray profiles (Ryan et al., 2010).

5.3 Background: Singular Value Decomposition

Singular Value Decomposition (SVD) (Berry et al., 1995; Scott C. et al., 1990) is a matrix factorisation technique that takes an $m \times n$ matrix A, with rank r and decomposes it into three component matrices as follows:

$$\text{SVD}(A) = U \times S \times V^{\mathsf{T}}. \tag{5.1}$$

U and V (V^{T} is for the transpose of V) are orthogonal matrices with dimensions $m \times m$, and $n \times n$ respectively, and S, called the *singular matrix*, is a $m \times n$ diagonal matrix consisting of non-negative real numbers. These matrices reflect the decomposition of the original matrix into linearly independent vectors (factor values). The set of initial r diagonal values of S $(s_1, s_2, \cdots, s_r)$ are all positive with $s_1 \geq s_2 \geq s_3, \cdots, \geq s_r$. The first r columns of U are eigenvectors of AA^{T} and represent the left singular vectors of A. Similarly, the first r columns of V are eigenvectors of $A^{\mathsf{T}}A$ and represent the right singular vectors of A. The best low-rank approximation of matrix A is obtained by retaining the first k diagonal values of S, by removing $r - k$ columns from U, and by removing $r - k$ rows from V, which can be represented as follows:

$$A_k = U_k \times S_k \times V_k^{\mathsf{T}}. \tag{5.2}$$

By keeping only the k largest singular values of S, the effective dimensions of the SVD matrices U, S, and V become $m \times k$, $k \times k$, and $k \times n$ respectively. The best-k rank approximation of matrix A with respect to the Frobenius norm can be represented by:

$$||A - A_k||_F^2 = \sum_{i,u}(a_{iu} - \sum_{k} U_{uk} \times S_k \times {V^{\mathsf{T}}}_{ki}) \tag{5.3}$$

SVD can be applied over the user-item rating matrix, of dimensions $M \times N$, generated by a recommender system. It assumes that there is some latent structure—overall structure that relates to all or most items (or users)—in the matrix that is partially obscured by variability in ratings assigned to items (or assigned by users). This latent structure can be captured by transforming the matrix in low dimensions. After transformation, users and items can be represented by a vector in the k-dimensional space. The matrix product $U_k.\sqrt{S_k}^{\mathsf{T}}$ represents M users and $\sqrt{S_k}.V_k^{\mathsf{T}}$ represents N items in the k-dimensional space. For example, in a movie domain, each element of $\sqrt{S_k}.V_k^{\mathsf{T}}(i)$ $(1 \leq i \leq N)$ can be a feature of movie i, such as whether it is a horror movie, whether it is rated PG-13 or not, etc. Similarly, the corresponding element of $U_k.\sqrt{S_k}^{\mathsf{T}}(u)$ $(1 \leq u \leq M)$ shows whether the

user likes these features in movies. A rating assigned by a pseudo-users u on item i is denoted by $r'_{i,u}$. The prediction $\check{r}_{i,u}$ for the uth user on the ith item can be computed by the following equation:

$$\check{r}_{i,u} = U_k.\sqrt{S_k}^{\mathsf{T}}(u).\sqrt{S_k}.{V_k}^{\mathsf{T}}(i). \tag{5.4}$$

If we normalise the user-item rating matrix by subtracting the respective user average ($\overline{r}_i$) from a rating, then a prediction is given by the equation:

$$\check{r}_{i,u} = \overline{r}_i + U_k.\sqrt{S_k}^{\mathsf{T}}(u).\sqrt{S_k}.{V_k}^{\mathsf{T}}(i). \tag{5.5}$$

5.4 SVD-Based Recommendations

5.4.1 Using imputation in SVD

We used various imputation methods, F (discussed in the next section), for approximating the missing values in the user-item rating matrix R and then applied SVD for reducing the dimensions of the matrix. The pseudo code to generate improved recommendations is given in Algorithm 2[2]. In step 7, which serves as a pre-processing step, we fill in the missing values in the initial sparse user-item rating matrix by an imputation source. In step 8, we normalise the filled rating matrix by subtracting the respective user average from the filled rating matrix. In step 9, we reduce the dimensions of the filled normalised rating matrix by applying SVD. In the ImputedError procedure, from steps 12 to 26, we find the optimal number of dimensions (k) by changing the dimension from 1 to 50 and observing the corresponding MAE.

5.4.2 SVD-based collaborative filtering

We can apply the user- and item-based CF over the matrix components generated by the Impute procedure. Algorithm 3 outlines the steps required to apply CF over the reduced data matrix. The similarity between two items can be found by Adjusted cosine or cosine measure (Adomavicius and Tuzhilin, 2005). We used Adjusted cosine similarity because it gave us more accurate results. The similarity between two items i_x and i_y can be found by measuring the cosine of angle computed over k users as follows:

$$\text{sim}(i_x, i_y) = \frac{\sum_{u=1}^{k} {r'}_{i_x,u}.{r'}_{i_y,u}}{\sqrt{\sum_{u=1}^{k} {r'}^2_{i_x,u} \sum_{u=1}^{k} {r'}^2_{i_y,u}}}, \tag{5.6}$$

[2] Appendix B gives details of packages and libraries used for the implementation.

Algorithm 2 : **ImpSvd**; Impute the matrix, compute SVD, and generate recommendations
Input: R, the user-item rating matrix; f, an imputation method
Output: $error^*$, the minimum MAE; k^*, the optimal number of dimensions for SVD

procedure SVDRecommendation(R, f)
    (U, S, V)=Impute(R, f)
    ($error^*$, k^*)=ImputedError(U, S, V)
    return ($error^*$, k^*)
end procedure

procedure Impute(R, f)
    Fill in the missing values in the user-item rating matrix R by an imputation method f. Call the resulting dense matrix R_f.
    Normalise the dense matrix (R_f) and call it R_N.
    Apply SVD over the normalised matrix R_N and find three components of the matrix as shown in equation 5.1. Call these matrices U, S, and V.
    return (U, S, V)
end procedure

procedure ImputedError(U, S, V)
    $error^* \leftarrow 10$
    $k^* \leftarrow 1$
    for $k \leftarrow 1, 50$ **do**
        (U_k, S_k, V_k)=DimReduce(U, S, V, k)
        Compute $U_k.\sqrt{S_k}^\intercal$ and $\sqrt{S_k}.{V_k}^\intercal$
        Make predictions using equation 5.5
        Compute MAE for all predictions, call this $error_{new}$
        if $error_{new} < error^*$ **then**
            $error^* \leftarrow error_{new}$
            $k^* \leftarrow k$
        end if
    end for
    return ($error^*$, k^*)
end procedure

procedure DimReduce(U, S, V, k)
Perform dimensionality reduction step:
    Find S_k by setting $S_{i,i} = 0$ for $i > k$
    Find U_k by removing $r - k$ columns from U
    Find V_k by removing $r - k$ rows from V
    return (U_k, S_k, V_k)
end procedure

Algorithm 3 : **ImpSvd$_{\mathbf{CF}}$**; Apply SVD over the reduced dataset
Input: R, the user-item rating matrix; f, an imputation method; $flag$, a variable to decide between the user- and item-based CF
Output: $error^*$, the minimum MAE; k^* and $neigh^*$, the optimal number of dimensions and neighbours for CF

procedure CFRecommendation(R, f, flag)
  (U, S, V)=Impute(R,f)
  Start grid search over dimensions, k and neighbourhood size, $neigh$ to find the optimal number of dimensions, k^* and neighbourhood size, $neigh^*$
  (U_k, S_k, V_k)=DimReduce(U, S, V, k)
  if flag $= 1$ **then**
    $\check{r}^{ib}_{i,u} = ImpSvd^{ib}_{CF}(U_k, S_k, V_k, neigh)$
  else
    $\check{r}^{ub}_{i,u} = ImpSvd^{ub}_{CF}(U_k, S_k, V_k, neigh)$
  end if
  Store the minimum MAE, $error^*$; the optimal number of dimensions, k^*; and the optimal number of neighbours, $neigh^*$
  End grid search
  return ($error^*$, k^*, $neigh^*$)
end procedure

procedure $ImpSvd^{ib}_{CF}(U_k, V_k, S_k, l)$
  Find the matrix product $\sqrt{S_k}.V_k{}^{\mathsf{T}}$
  Find the similarity between two items using equation 5.6
  Isolate l most similar items to the target item (neighbours of the target item) found using equation 5.6
  Make a prediction, $\check{r}^{ib}_{i,u}$, using equation 5.8
  return $\check{r}^{ib}_{i,u}$
end procedure

procedure $ImpSvd^{ub}_{CF}(U_k, V_k, S_k, l)$
  Find the matrix product $U_k.\sqrt{S_k}^{\mathsf{T}}$
  Find the similarity between two users using equation 5.7
  Isolate l most similar users to the active user (neighbours of the active user) found using equation 5.7
  Make a prediction, $\check{r}^{ub}_{i,u}$, using equation 5.9
  return $\check{r}^{ub}_{i,u}$
end procedure

where ${r'}_{i_x,u}$ and ${r'}_{i_y,u}$ are the ratings assigned by user u on items i_x and i_y respectively. The ratings shown by r' are obtained from the matrix product $\sqrt{S_k}^{\mathsf{T}}.V_k$, which represents the rating given by k (pseudo) users on N items[3]. We used the significance weighting schemes as proposed by Ghazanfar and Prügel-Bennett (2010d) while measuring the similarity between users (or items).

The similarity between two users can be found by the Pearson correlation or the cosine of angle (Herlocker et al., 2002). We used the cosine of angle, which gave us more accurate results than the Pearson correlation. The similarity between two users can be found by the cosine of angle, computed over k items, as follows[4]:

$$\text{sim}(u_a, u_b) = \frac{\sum_{i=1}^{k} {r'}_{i,u_a}.{r'}_{i,u_b}}{\sqrt{\sum_{i=1}^{k} {r'}^2_{i,u_a} \sum_{i=1}^{k} {r'}^2_{i,u_b}}}, \tag{5.7}$$

where ${r'}_{i,u_a}$ and ${r'}_{i,u_b}$ are the ratings assigned on item i by users u_a and u_b respectively. The ratings shown by r' are obtained from the matrix product $U_k\sqrt{S_k}^{\mathsf{T}}$, which represents the ratings given by M users on k (pseudo) items.

In the case of item-based CF, the prediction for an active user u_a on target item i_t is made by using the adjusted weighted sum formula as follows:

$$\check{r}_{i_t,u_a} = \overline{r}_{u_a} + \frac{\sum_{i=1}^{l} sim(i, i_t) \times {r'}_{i,u_a}}{\sum_{i=1}^{l} |sim(i, i_t)|}, \tag{5.8}$$

where l represents the l most similar items against a target item, found after applying equation 5.6.

In the case of the user-based CF, the prediction for an active user u_a on target item i_t is made by using the adjusted weighted sum formula as follows:

$$\check{r}_{i_t,u_a} = \overline{r}_{u_a} + \frac{\sum_{u=1}^{l} sim(u, u_a) \times ({r'}_{i_t,u} - \bar{r}_u)}{\sum_{i=u}^{l} |sim(u, u_a)|}, \tag{5.9}$$

where l represents the l most similar users against an active user, found after applying equation 5.7.

[3]We do not need to subtract the respect user average while measuring the similarity as the matrix has already been normalised prior to applying SVD.

[4]In this case, we do not normalise the user-item rating matrix prior to applying SVD.

Individual predictions made by the user- and item-based CF can be combined linearly. We expect that combining these two approaches will result in an increase in the accuracy, as both of them focus on different kinds of relationships. Let $\check{r}_{i,u}^{ub}$ and $\check{r}_{i,u}^{ib}$ represent the prediction generated by the user- and item-based CF respectively. The final prediction is a linear combination of these predictions as follows:

$$\check{r}_{i,u} = \alpha \times \check{r}_{i,u}^{ub} + \beta \times \check{r}_{i,u}^{ib}, \tag{5.10}$$

where parameters α and β can be found over the validation set. We call this algorithm $ImpSvd_{CF}^{hybrid}$.

5.4.3 Applying SVD combined with EM algorithm

Algorithm 4 : **ItrSvd**; Apply SVD in EM fashion
Input: R, the user-item rating matrix; f, an imputation method; ϑ, threshold value to terminate the EM algorithm
Output: t, the number of iterations in the EM algorithm; $error^*$, the MAE observed after the EM algorithm converges

procedure IterativeRecommendation(R, f, ϑ)
  $t \leftarrow 0$
  $error^{(t)} \leftarrow 0$
  repeat
    (U, S, V) = Impute(R, f)
    $(U_{k^*}, V_{k^*}, S_{k^*})$=DimReduce($U, S, V, k^*$) ## k^* is the optimal number of dimensions learned through the validation set
    Compute $U_{k^*}\sqrt{S_{k^*}}^{\intercal}$, $\sqrt{S_{k^*}}{V_{k^*}}^{\intercal}$
    Call the current SVD model $\mathcal{M}_{k^*}$
    Make predictions using equation 5.5
    Compute the MAE for $\mathcal{M}_{k^*}$, call it $error_{new}$
    $t \leftarrow t + 1$
    $error^{(t)} \leftarrow error_{new}$
    $f \leftarrow \mathcal{M}_{k^*}$
  until $|error^{(t)} - error^{(t-1)}| < \vartheta$
  $error^* \leftarrow error^{(t)}$
  return t, $error^*$
end procedure

The *ItrSvd* (Algorithm 4) uses the combination of SVD and Expectation Maximisation (EM) (Do and Batzoglou, 2008) to estimate the missing values. As SVD calculations require the filled matrix, missing values are replaced by an imputation method prior to the k most effective eigenvalues being selected. In each iteration of the EM algorithm, the missing values are replaced by the corresponding values in the previous estimated model in the expectation step, i.e.

$$R_{iu}^{(t)} = \begin{cases} R_{iu} & if\ iu \in \mathcal{D}\,, \\ [\sum_k U_k \times S_k \times {V^{\intercal}}_k]_{iu}^{(t-1)} & otherwise, \end{cases} \tag{5.11}$$

and in the maximisation step the aim is to find the model ($\mathcal{M}^{(t)}$) parameters that minimises

$$\sum_{iu}(R_{iu}^{(t)} - \mathcal{M}_{iu})^2, \tag{5.12}$$

where $\mathcal{M}_{iu} = [\sum_k U_k \times S_k \times V^{\mathsf{T}}{}_k]_{iu}$. The algorithm keeps alternating between expectation and maximisation (SVD computation) steps, until it converges (the change in the MAE between two iterations becomes less than a pre-determined threshold (0.001)). This algorithm usually gives more accurate results after convergence; however, its drawback is that it is highly sensitive to the noise in the dataset and it only considers the global data correlation, which means that in a locally correlated dataset, it will lead to higher estimation error.

5.5 Proposed Approaches to Approximate the Missing Values in the User-item Rating Matrix

The imputation approaches we have used are discussed below:

1- *Filling by zero (Zeros)*: In this approach, we replace an unknown rating in the user-item rating matrix by zero. This approach is very simple, and computationally efficient, which makes it attractive. It does not take into account the underlying correlation structure of the data affecting the data variance that is generally high. Subsequently, if we have a large number of missing values, then this imputation approach can result in inaccurate recommendations.

2- *Filling by random number (Rand)*: In this approach, we replace an unknown rating in the user-item rating matrix by a random number generator function that generates a random number in the range of 1 to 5 in the case of MovieLens and 1 to 10 in the case of FilmTrust dataset. Its advantages and disadvantages are the same as those of Zeros.

3- *Filling by normal distribution (Nor_U, Nor_I)*: In this approach, we replace an unknown rating in the user-item rating matrix by normal distribution $\mathcal{N}(\mu, \sigma^2)$. Here we use Nor_U to represent the case where the corresponding user average and standard deviation of ratings are used as μ and σ respectively. Similarly, we use Nor_I to represent the case where the corresponding item average and standard deviation of ratings (given by other users) are used as μ and σ respectively.

4- *Filling by uniform distribution (UniformDist)*: In this approach, we replace an unknown rating in the user-item rating matrix by uniform distribution $\mathcal{U}(a, b)$, where $(a, b) = (1, 5)$ for the MovieLens dataset and $(a, b) = (1, 10)$ for the FilmTrust dataset.

5- *Filling by items' averages (ItemAvg)*: In this approach, we replace an unknown rating in the user-item rating matrix by the average rating given by all the users in the training set. If no one has rated that item it is replaced by zero. This approach serves as a **baseline** for our experimental evaluation, as it has been the preferred approach in the literature to resolve this problem.

6- *Filling by users' averages (UserAvg)*: In this approach, we replace an unknown rating in the user-item rating matrix by the average rating given by the active user in the training set. If the active user has rated no item, then it is replaced by zero. This approach is very simple; however, it can distort the shape of the distribution and can reduce the variance of the data. We use the term **conventional methods** for the ItemAvg and UserAvg imputation methods.

7- *Filling by the average of users' and items' averages (UserItemAvg)*: In this approach, we replace an unknown rating in the user-item rating matrix by averaging the user's average rating and the item's average rating.

8- *Filling by the user-based CF (UBCF)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the user-based CF[5].

9- *Filling by the item-based CF (IBCF)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the item-based CF.

10- *Filling by the average of user- and item-based CF (UBIBCF)*: In this approach, we replace an unknown rating in the user-item rating matrix by averaging the predictions generated by the user- and item-based CF.

11- *Filling by SVM classifier (SVMClass)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the SVM classifier over the training set. The details of building and using the SVM have been given in the previous chapter.

12- *Filling by the Naive Bayes classifier (NBClass)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the Naive Bayes classifier over the training set. The details of building and using the Naive Bayes classifier for recommender system have been given in the previous chapter.

13- *Filling by the K Nearest Neighbours (KNN)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the K Nearest Neighbours (KNN) using the Weka collection of machine learning algorithms (Hall et al., 2009). KNN estimates missing values by searching for the K nearest neighbours (users) and then taking the weighted average of these

[5]If the algorithm fails to predict a rating, then it is replaced by the average of users' and items' average ratings. The same is true for the item-based CF.

K neighbours' ratings. In our work, the proposed scheme is similar to the KNN; however, it differs in that the contribution of each neighbour is weighted by its similarity to the active user. As the degree of contribution will be determined by the choice of weighting system, hence we tested our scheme with two weighting systems. In the first approach (shown by KNN in the results) we weight neighbours by $1-dist$, where $dist$ is the distance between two neighbours. In the second approach (shown by WKNN in the results), we weight neighbours according to the following scheme employed by Ryan et al. (2010):

$$weight(i,j) = \Big(\frac{dist^2}{1-dist^2+\epsilon}\Big)^2,$$

where $\epsilon = 10^{-6}$ is added to avoid dividing by zero. This function is similar to the Gaussian kernel function, which gives more weight to closer neighbours than distant neighbours. WKNN has proven to give good results in Ryan et al. (2010).

14- *Filling by the decision tree (C4.5)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the decision tree (C4.5) using the Weka library. Although the process of constructing the tree tries to minimise the error rate using the training data for evaluation, it will probably not perform well while classifying the test data. The reason is that it can easily be *over-fitted* to the training data (Witten and Frank, 1999). Therefore, in order to generalise its performance, we pruned the tree by learning the pruning confidence over the validation set.

15- *Filling by the SVM regression (SVMReg)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the SVM regression over the training set. We used the linear kernel and trained the cost parameters. We used the nu-SVR version of the SVM regression using the LibSVM (Chang and Lin, 2011) library.

16- *Filling by the linear regression (LinearReg)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the linear regression using the Weka library. This method tries to lower the data variance of missing value estimates by exploiting the underlying localised or global correlation structure of the data.

17- *Filling by the logistic regression (LogisticReg)*: In this approach, we replace an unknown rating in the user-item rating matrix by using the results obtained by applying the logistic regression using the Weka library.

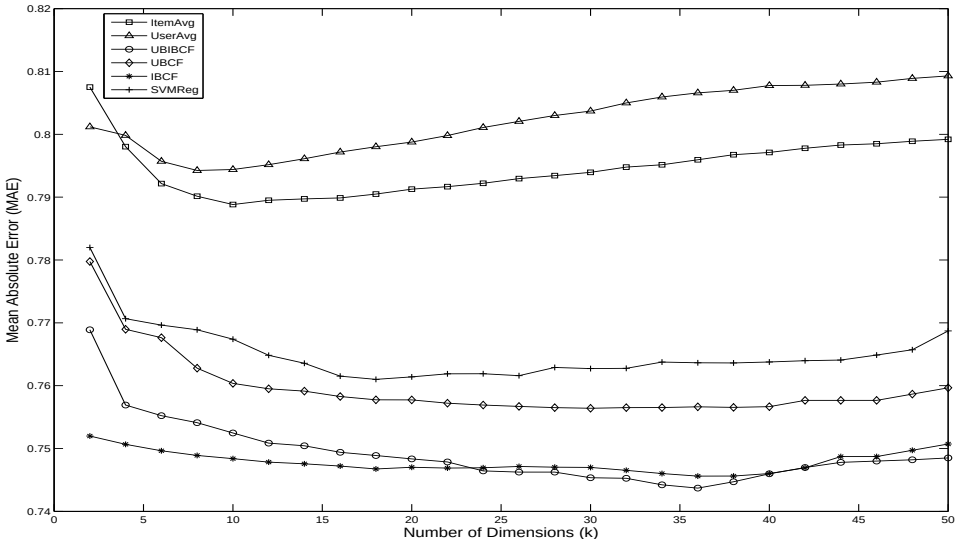


Figure 5.1: Determining the optimal number of dimensions in the *ImpSvd* over the validation set (SML dataset). The error bars, lying between 0.001 and 0.004 for all approaches, are not shown for reasons of clarity.

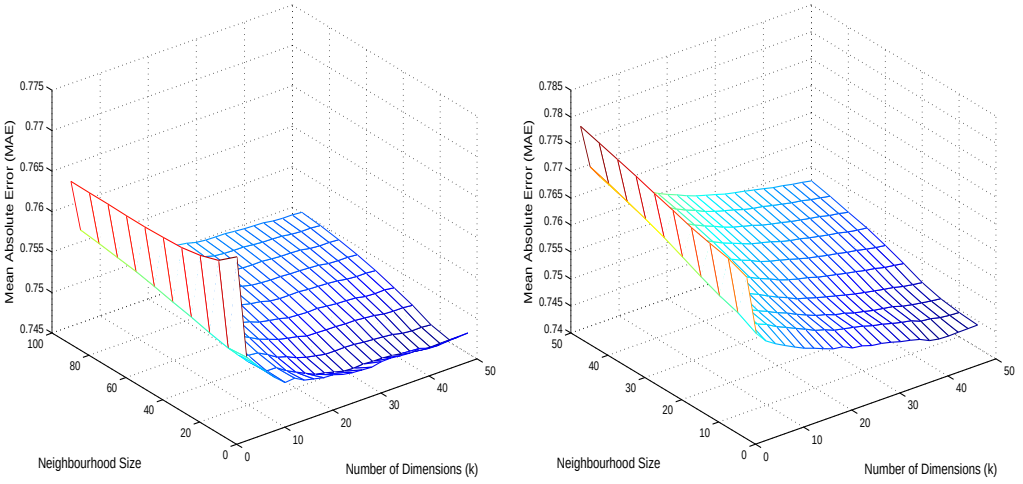
5.6 Results and Discussion

5.6.1 Learning the optimal system parameters

The purpose of these experiments is to determine which of the parameters affects the prediction quality of the proposed algorithms, and to determine their optimal values. We show the tuning of important parameters. The tuning of other parameters is given in Appendix B.

5.6.1.1 Finding the optimal number of dimensions for SVD

Two factors are important while finding the optimal number of dimensions. First, the number of dimensions must be small enough to make the resulting system scalable and second it must be big enough to capture the important latent information between the users or items. Figure 5.1 shows how the MAE changes as a function of the number of dimensions (k) in the case of SML dataset. We show results only for the conventional approaches and the ones giving us good results. We observe that, in the case of UBCF, IBCF, and UBIBCF, the MAE keeps on decreasing, reaches its minimum between $k = \{30 - 40\}$, and then starts increasing again. We choose $k = 36$ for these imputation methods. We further observe that the MAE is minimum at $k = 18$, $k = 8$, and $k = 10$ in the case of SVMReg, UserAvg, and ItemAvg respectively. Similarly, we tuned all approaches for the optimal dimensions for other datasets.



(a) User-based CF (b) Item-based CF

Figure 5.2: Determining the optimal parameters in the SVD-based CF for the SML dataset (with the IBUBCF imputation), through grid search over the validation set. The "Number of Dimensions (k)" represents the number of dimensions in the reduced space and "Neighbourhood Size" represents the number of most similar items against the target item in the case of the item-based CF and the number of most similar users against the active user in the case of the user-based CF.

5.6.1.2 Finding the optimal number of neighbours and dimensions in CF

The neighbourhood size is dataset-dependent and furthermore, a change in the distribution and sparsity of the dataset will change the neighbourhood size. We performed a series of experiments by changing the dimension each time from 2 to 50 with a difference of 2. For each experiment (keeping the dimension parameter fixed), we changed the neighbourhood size from 5 to 100 with a difference of 10 for the user-based CF and from 5 to 50 with a difference of 5 for the item-based CF, and observed the corresponding MAE. Figure 5.2 shows how the MAE changes with these parameters in the cases of the user- and item-based CF. Figure 5.2(a) shows that, in the case of the user-based CF, the MAE is minimum at the neighbourhood size of 15. We observe in the dimension scale, keeping the neighbourhood size fixed to 15, that the MAE decreases with an increase in the rank of the lower dimension space, reaches its peak at $k = 46$, and after that it either increases or stays constant. The grid coordinates $\{$*neighbours, dimensions*$\}$, which gave the lowest MAE, are found to be $\{15, 46\}$. Figure 5.2(b) shows that the MAE is minimum at $\{5, 44\}$. Considering these results, we choose the optimal parameters ($\{$*neighbours, dimensions*$\}$) to be $\{15, 46\}$ and $\{5, 44\}$ for the user- and item-based CF respectively. Similarly, we tuned the parameters for all approaches for other datasets.

5.6.2 Performance evaluation of different imputation methods

The results obtained by the *ImpSvd* algorithm (Algorithm 2) under different imputation methods are shown in Table 5.1. Note that we only show the best results obtained by varying k from 1 to 50. The table shows that the SVMReg, UBCF, IBCF, and UBIBCF imputation approaches give more accurate results than others. The % decrease in MAE over the baseline approach ItemAvg is found to be (1) 4.79%, 5.62%, and 6.58% in the case of UBCF, IBCF, and UBIBCF respectively for the ML dataset; (2) 5.17%, 5.56%, 5.94%, and 7.23% in the case of SVMReg, UBCF, IBCF, and UBIBCF respectively for the SML dataset; (3) 17.0%, 14.71%, 14.53%, and 15.53% in the case of SVMReg, UBCF, IBCF, and UBIBCF respectively for the FT1 dataset; and (4) 5.87%, 2.76%, 2.56%, and 4.59% in the case of SVMReg, UBCF, IBCF, and UBIBCF respectively for the FT5 dataset. The ranking of different approaches (with respect to the MAE) with the respective p-value in the case of pair t test is found to be: (1) **UBIBCF** ($p < 0.001$) > **IBCF** ($p < 0.05$) > **UBCF** ($p < 0.05$) for the ML dataset; (2) **UBIBCF** ($p < 0.001$) > **IBCF** ($p < 0.001$) > **UBCF** ($p < 0.001$) >**SVMReg** ($p < 0.05$) for the SML dataset; (3) **SVMReg** ($p < 0.001$) > **UBIBCF** ($p < 0.001$) > **IBCF** ($p < 0.001$) >**UBCF** ($p < 0.001$) for the FT1 dataset; and (4) **SVMReg** ($p < 0.001$) > **UBIBCF** ($p < 0.005$) > **IBCF** ($p < 0.001$) >**UBCF** ($p < 0.005$) for the FT5 dataset. Furthermore, the proposed imputation approaches give 5% to 10% improvement over the baseline approach, in terms of ROC-sensitivity, precision, recall, and F1 (refer to Appendix B). In the following, we will concentrate on the conventional approaches and the ones which gave good results.

The FilmTrust dataset effectively demonstrates the real world recommender system's characteristics. It adequately captures the *new user* and *item cold-start problems*. What is evident from Table 5.1 is that the baseline approach gives the worst results in this case. We observe that the SVMReg approach outperforms others in the case of the FilmTrust dataset. This is because the FilmTrust dataset is very sparse, and hence we do not have comprehensive users' (or items') rating profiles that can be used to make predictions for other unknown items. However, we can capture users' profiles in terms of the important features in which they are interested, resulting in improved users' profiles and predictions.

In the case of the FT5 dataset, the performance of different approaches, even the conventional ones, improves simply because we have removed users and items with less clear profiles. Again, the baseline approach gives the worse results.

Table 5.1: Best MAE observed in different imputation methods. k represents the number of dimensions, which gave the most accurate results. The best results are shown in bold font.

Imp. Method	**Best *MAE***				**Number of dimensions (k)**			
	ML	**SML**	**FT**1	**FT**5	**ML**	**SML**	**FT**1	**FT**5
Zeros	2.425 ± 0.001	2.321 ± 0.001	4.354 ± 0.009	3.898 ± 0.013	26	12	2	2
Rand	1.092 ± 0.001	1.072 ± 0.002	2.214 ± 0.005	2.064 ± 0.012	14	4	2	2
ItemAvg	0.730 ± 0.001	0.774 ± 0.001	1.700 ± 0.005	1.483 ± 0.005	22	10	10	4
UserAvg	0.759 ± 0.001	0.778 ± 0.001	1.452 ± 0.007	1.433 ± 0.002	22	8	4	4
UserItemAvg	0.724 ± 0.001	0.754 ± 0.001	1.527 ± 0.008	1.442 ± 0.006	30	12	14	4
UniformDist	0.911 ± 0.001	0.905 ± 0.001	2.061 ± 0.012	1.933 ± 0.011	10	4	2	2
Nor_U	0.790 ± 0.001	0.810 ± 0.001	1.505 ± 0.008	1.491 ± 0.005	4	2	2	2
Nor_I	0.766 ± 0.001	0.800 ± 0.001	1.796 ± 0.008	1.562 ± 0.010	2	2	2	2
UBCF	0.695 ± 0.001	0.731 ± 0.001	1.450 ± 0.008	1.442 ± 0.006	40	36	4	10
IBCF	0.689 ± 0.001	0.728 ± 0.001	1.453 ± 0.005	1.445 ± 0.005	40	36	6	8
UBIBCF	$\mathbf{0.682 \pm 0.001}$	$\mathbf{0.718 \pm 0.001}$	1.436 ± 0.006	1.415 ± 0.006	40	36	6	10
KNN	$--$	0.804 ± 0.002	1.485 ± 0.008	1.479 ± 0.008	$--$	18	4	4
WKNN	$--$	0.793 ± 0.002	1.481 ± 0.007	1.474 ± 0.008	$--$	18	4	4
NBClass	$--$	0.775 ± 0.002	1.475 ± 0.008	1.468 ± 0.007	$--$	26	8	6
SVMClass	$--$	0.763 ± 0.002	1.455 ± 0.007	1.445 ± 0.007	$--$	18	6	4
C4.5	$--$	0.781 ± 0.002	1.495 ± 0.005	1.485 ± 0.005	$--$	22	10	12
SVMReg	$--$	0.734 ± 0.002	$\mathbf{1.411 \pm 0.005}$	$\mathbf{1.396 \pm 0.005}$	$--$	18	6	6
LinearReg	$--$	0.783 ± 0.002	1.447 ± 0.005	1.437 ± 0.005	$--$	16	4	2
LogisticReg	$--$	0.781 ± 0.002	1.443 ± 0.007	1.434 ± 0.007	$--$	14	4	4

Table 5.2: The MAE observed in different imputation methods in the case of $ImpSvd^{ib}_{CF}$. The best results are shown in bold font.

Imp. Method	**Best *MAE***			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.741 ± 0.001	0.781 ± 0.001	1.702 ± 0.005	1.475 ± 0.006
UserAvg	0.767 ± 0.001	0.788 ± 0.002	1.496 ± 0.005	1.442 ± 0.005
UBCF	0.721 ± 0.001	0.739 ± 0.001	1.483 ± 0.005	1.434 ± 0.005
IBCF	0.701 ± 0.001	0.738 ± 0.001	1.459 ± 0.004	1.462 ± 0.004
UBIBCF	$\mathbf{0.691 \pm 0.000}$	$\mathbf{0.723 \pm 0.001}$	1.432 ± 0.005	1.418 ± 0.007
SVMReg	$--$	0.744 ± 0.001	$\mathbf{1.417 \pm 0.006}$	$\mathbf{1.404 \pm 0.006}$

Table 5.3: The MAE observed in different imputation methods in the case of $ImpSvd^{ub}_{CF}$. The best results are shown in bold font.

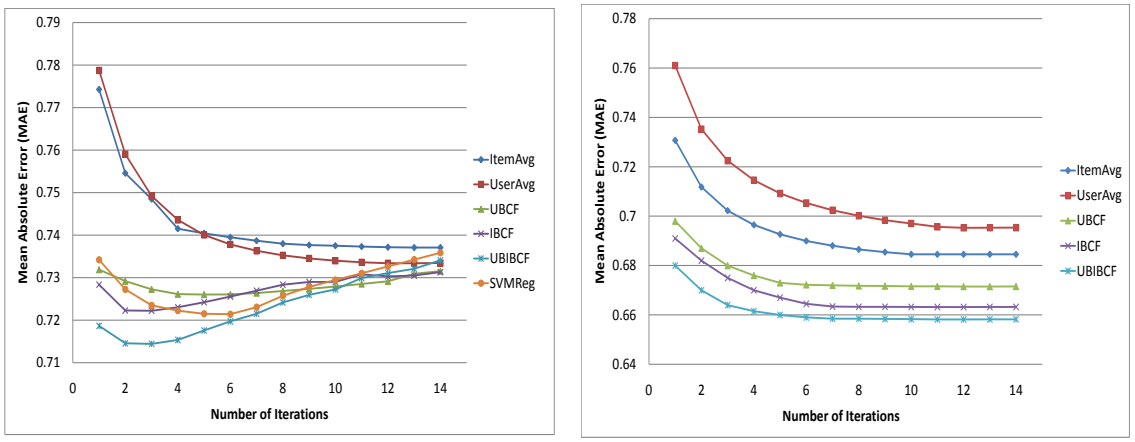
Imp. Method	**Best *MAE***			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.742 ± 0.001	0.776 ± 0.001	1.731 ± 0.005	1.465 ± 0.006
UserAvg	0.773 ± 0.001	0.786 ± 0.001	1.483 ± 0.004	1.439 ± 0.005
UBCF	0.709 ± 0.001	0.734 ± 0.001	1.465 ± 0.004	1.422 ± 0.006
IBCF	0.706 ± 0.001	0.732 ± 0.001	1.446 ± 0.005	1.445 ± 0.007
UBIBCF	$\mathbf{0.692 \pm 0.000}$	$\mathbf{0.722 \pm 0.001}$	1.445 ± 0.005	1.419 ± 0.007
SVMReg	$--$	0.743 ± 0.001	$\mathbf{1.416 \pm 0.005}$	$\mathbf{1.401 \pm 0.006}$

5.6.3 Performance evaluation of the SVD-based CF

Tables 5.2 and 5.3 show that the proposed approaches give more accurate results than the conventional ones, when we apply CF over the reduced dataset. It is worth noting that the results (in general) obtained by applying CF over the reduced dataset do not have any advantages over the results obtained by applying SVD. However, in the case of the FilmTrust dataset, some of the proposed approaches (UBCF, IBCF, UBIBCF) give (insignificantly) better results when CF is applied over the reduced dataset. It is due to the reason that the FilmTrust dataset is very sparse, which implies the latent structure between movies and users might not be captured by applying SVD, and can be found by applying CF over the reduced dataset. Another factor to note is that the results obtained in the case of the proposed approaches are (almost) equivalent to the

Table 5.4: The MAE, ROC-sensitivity, precision, recall, and F1 observed in the case of hybrid recommender system ($ImpSvd^{hybrid}_{CF}$) proposed in Section 5.4.2. The SMVReg is used for the FilmTrust dataset and the UBIBCF is used for the remaining datasets as an imputation method, prior to applying SVD.

DataSet	**MAE**	**ROC**	**Precision**	**Recall**	**F**1
ML	0.686 ± 0.000	0.790 ± 0.001	0.518 ± 0.002	0.595 ± 0.001	0.524 ± 0.002
SML	0.719 ± 0.001	0.695 ± 0.002	0.543 ± 0.002	0.555 ± 0.001	0.513 ± 0.002
FT1	1.409 ± 0.005	0.566 ± 0.003	0.591 ± 0.003	0.568 ± 0.003	0.549 ± 0.003
FT5	1.394 ± 0.005	0.578 ± 0.003	0.598 ± 0.005	0.574 ± 0.003	0.556 ± 0.005



(a) *ItrSvd* (SML dataset)

(b) *ItrSvd* (ML dataset)

Figure 5.3: Comparing the performance (in terms of the MAE) of the proposed approaches with others in the case of *ItrSvd* (fixed dimension case) over the SML dataset. X-axis shows the number of iterations of the EM algorithm and y-axis shows the corresponding MAE observed. The proposed approaches converge much quicker than the conventional ones. The error bars (< 0.001 for all approaches) are not shown for reasons of clarity.

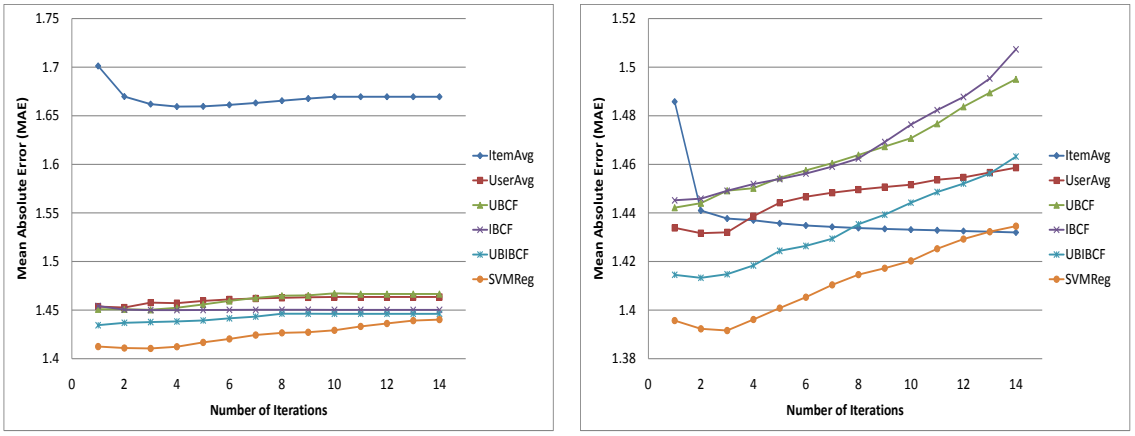
ones obtained in the *ImpSvd* algorithm. Furthermore, in general, the user-based CF performs better than the item-based CF in the case of FT1, FT5, and SML dataset, whereas the item-based CF performs better than the user-based CF in the case of the ML dataset.

The user- and item-based CF can be combined linearly as discussed in Section 5.4.2. Table 5.4 shows that linearly combining the user- and item-based CF gives the improved results with MAE equal to 0.686, 0.719, 1.409, and 1.394 in the case of SML, ML, FT1, and FT5 datasets respectively. The reason of improvements in the results is that the user- and item-based CF focus on different kinds of relationship in the dataset.

5.6.4 Performance evaluation of SVD combined with the EM algorithm (*ItrSvd*)

There are two options to find the optimal number of dimensions in the *ItrSvd* algorithm; (1) learning the optimal number of dimensions in the first iteration using the validation set and keeping them fixed for all the iterations, and (2) learning the optimal number of dimensions in each iteration using the validation. In the following, we represent the former case with *fixed dimension* and the latter one with *variable dimension*. We first show results for the fixed dimension and then proceed to the variable dimension case.

Figure 5.3(a) and 5.3(b) show how the MAE changes with the number of iterations for the SML and ML datasets respectively. We observe that the conventional approaches



(a) *ItrSvd* (FT1 dataset)

(b) *ItrSvd* (FT5 dataset)

Figure 5.4: Comparing the performance (in terms of the MAE) of the proposed approaches with others in the case of *ItrSvd* (fixed dimension case), over the FilmTrust dataset. X-axis shows the number of iterations and y-axis shows the corresponding MAE observed. The proposed approaches give better results than the conventional ones. The error bars (lying between 0.001 and 0.004 for all approaches) are not shown for reasons of clarity.

converge much slower compared to the proposed ones. Figure 5.3(a) shows that in the case of the baseline approach, the MAE keeps on decreasing until it converges after 10 iterations. The minimum MAE observed after 10 iterations is 0.738. We observe that the MAE for the proposed approaches is much lower (about 5% less) compared to the conventional ones, and they converge much faster than the conventional ones. The IBCF and UBIBCF converge after $2-3$, whereas the UBCF and SVMReg converge after $5-6$ iterations, and then the MAE starts increasing, which may be due to the over-fitting. We further observe that the CF imputation method gives better results than the SVMReg. Furthermore, Figure 5.3(b) shows that the results are the same for the ML dataset.

The baseline approach gives the worst results in the case of the FT1 dataset. Figure 5.4(a) shows that for the baseline approach, the MAE keeps on decreasing until it converges after $4-5$ iterations. The IBCF and SVMReg approaches show similar behaviour, where the MAE reaches its minimum after $3-4$ iterations, and then starts increasing again. The remaining approaches do not show any improvement in the MAE with an increase in the number of iterations.

The results in the case of FT5 dataset are shown in Figure 5.4(b). The results of the baseline approach are surprisingly good where the MAE keeps on decreasing, until it converges after $12-14$ iterations. The lowest MAE observed after 12 iterations is still higher than the ones obtained in the first iteration of the proposed approaches. In the case of the UBCF and IBCF approaches, the MAE increases with a corresponding increase in the number of iterations. The MAE in the case of the remaining approaches

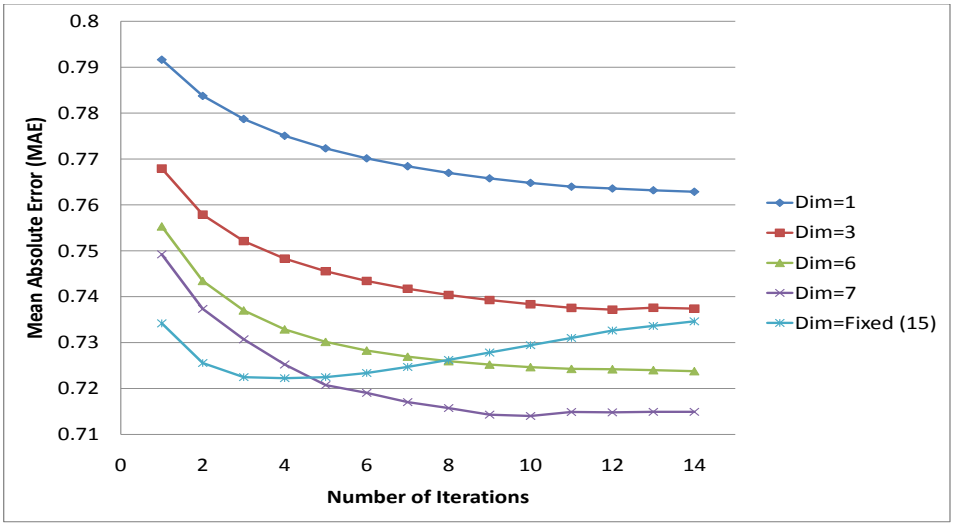


Figure 5.5: How the MAE changes with an increase in the number of dimensions for the SVMReg approach, over the SML dataset. X-axis shows the number of iterations and y-axis shows the corresponding MAE observed. Fixed dimensions represents the case where the optimal numbers of dimensions are learned in the first iteration through the validation set and kept fixed for all iterations. We observe that the results are highly dependent on the dimension parameter.

decreases with an increase in the number of iterations, reaches its minimum at $2-3$ iterations and then starts increasing. Again, the SVMReg imputation approach gives more accurate results.

The optimal number of dimensions can be learned at each iteration using the validation set; although this would be expensive, it may increase accuracy. To check how the MAE changes with the dimension parameter, we show results in the case of SVMReg imputation approach over the SML dataset for 1, 3, 5, 7, and 15 dimensions. Figure 5.5 shows that the MAE is highly dependent on the dimension parameter. To further investigate the results, we perform experiments where the optimal numbers of dimensions are learned at each iteration. We only show results in the case of the SML dataset, although similar results were observed for other datasets as well.

The results[6] for the SML dataset are shown in Figure 5.6. Figure 5.6 shows that learning the optimal number of dimensions at each iteration decreases the MAE of all approaches in general. Furthermore, all approaches except the SVMReg show similar behaviour as shown by the fixed dimension case. The MAE in the case of the SVMReg approach keeps on decreasing with an increase in the number of iterations, reaches its minimum when the number of iterations is 6, and then either stays stable or increases again. We further observe that the SVMReg outperforms others, which is not true in the fixed dimension case.

[6]Note that we used the training data to estimate the best parameters and used an independent test set to give the unbiased estimate of the generalisation error.

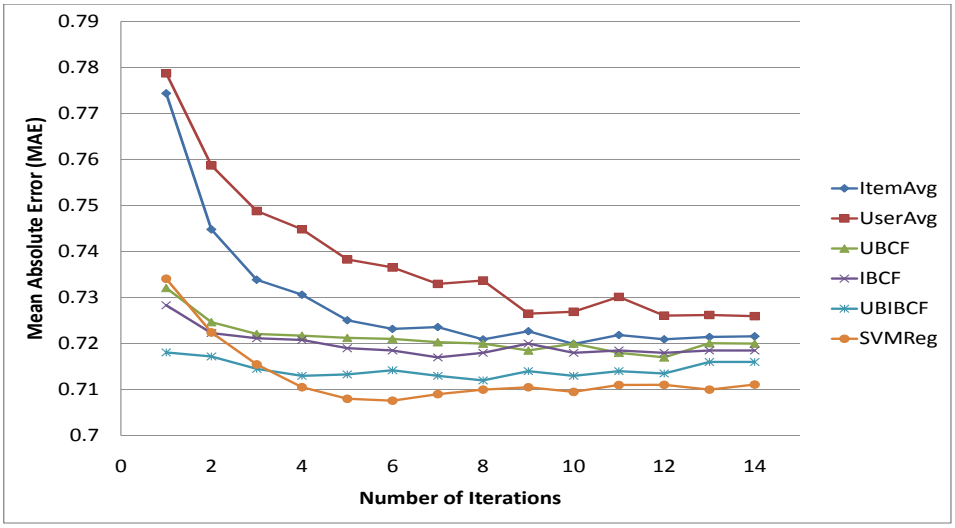


Figure 5.6: Comparing the proposed approaches with others in the case of *ItrSvd* (variable dimension case) over the SML dataset. X-axis shows the number of iterations and y-axis shows the corresponding MAE observed. The proposed approaches converge much quicker compared to the conventional ones. The error bars (< 0.001) are not shown for reasons of clarity.

Table 5.5: Comparing the MAE observed in different imputation methods in the *ItrSvd* (fixed dimension case). The best results are shown in bold font.

Imp. Method	**Best MAE**			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.685 ± 0.001	0.738 ± 0.001	1.661 ± 0.002	1.438 ± 0.005
UserAvg	0.697 ± 0.001	0.734 ± 0.001	1.451 ± 0.002	1.430 ± 0.002
UBCF	0.672 ± 0.001	0.723 ± 0.001	1.450 ± 0.002	1.442 ± 0.005
IBCF	0.664 ± 0.001	0.722 ± 0.001	1.448 ± 0.002	1.442 ± 0.005
UBIBCF	$\mathbf{0.659 \pm 0.001}$	$\mathbf{0.715 \pm 0.001}$	1.436 ± 0.002	1.418 ± 0.004
SVMReg	$--$	0.721 ± 0.001	$\mathbf{1.401 \pm 0.001}$	$\mathbf{1.390 \pm 0.004}$

Table 5.5 compares the performance in terms of MAE of different approaches under the *ItrSvd* (fixed iteration case). We observe that the proposed approaches produce better results than the baseline approach. We further observe that, in the case of the MovieLens dataset, the UBIBCF and IBCF approaches outperform others; whereas in the case of the FilmTrust dataset, the SVMReg and UBCF perform the best. Similar results were observed in terms of ROC-sensitivity and top-N metrics (refer to Appendix B).

5.6.5 Performance evaluation under different sparsity levels

To check the performance of the proposed approaches under sparsity, we increased the sparsity level of the training set by removing some randomly selected rating records, whereas, we kept the test set the same for each sparse training set. We used the *ImpSvd* algorithm to make recommendations. Figure 5.7 shows how different approaches perform

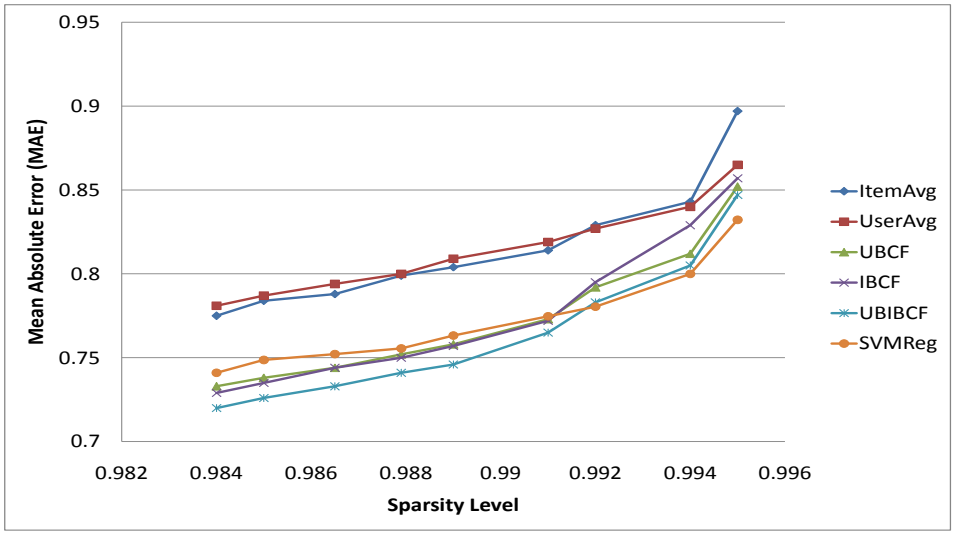


Figure 5.7: How sparsity affects the performance of different approaches for the SML dataset. Algorithm 2 was used to make recommendations.

under sparse conditions. The figure shows that the performance of the conventional approaches suffer more than the proposed ones. The reason is that under sparse conditions, the items' and users' averages can be misleading resulting in erroneous recommendations.

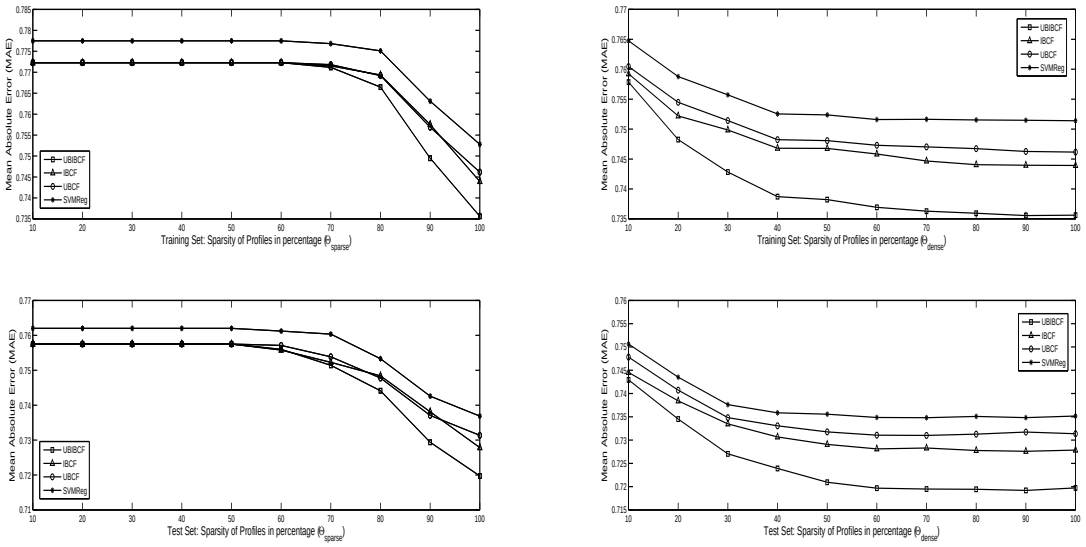
It must be noted that under very sparse conditions (sparsity ≥ 0.994), SVMReg outperforms the rest. The reason is the same as discussed in Section 5.6.2. We also note that the remaining approaches give the equivalent results. Hence, under very sparse conditions, the SVMReg can be used provided sufficient resources (e.g. content features) are available, and the conventional approaches can be used otherwise.

We also performed experiments with different test and training sizes and again observed similar results (refer to Appendix B for details.).

5.6.6 Performance evaluation under cold-start and long tail scenarios

We tested the proposed *ImpSvd* algorithm for cold-start and long tail scenarios. As an example, we present results in the case of a new item scenario. Similar results were observed for other scenarios as well, which are given in Appendix B.

For testing the performance of different approaches under the new item cold-start scenario, we selected 100 random items, and kept the number of users in the training set who have rated these item to 2, 5, 10, 15, and 20. The corresponding *MAE*, represented by $MAE2$, $MAE5$, $MAE10$, $MAE15$, and $MAE20$, is shown in Table 5.7. Table 5.7 shows that the SVMReg gives the best performance when an item has been rated by less than (or equal) to 10 items, and UBCF gives the best performance otherwise. We note that the IBCF does not perform very well compared to the UBCF. The reason is that we do not have the comprehensive items' rating profiles.



(a) When is imputation required?

(b) How much imputation is required?

Figure 5.8: Figures showing when and how much imputation is required for the SML dataset. Θ_{sparse} shows the sparsity of users' (or items') profiles in percentage. Θ_{dense} shows the percentage up to which users' (or items') profiles are filled using the proposed approaches. The optimal number of dimensions have been kept the same as shown in Table 5.1.

5.6.7 A comparison of the proposed algorithms with others

Table 5.6 gives a comparison of different algorithms in terms of MAE. We compared the proposed algorithms with others as discussed in Chapter 2, Section 2.5. Furthermore, we tuned all algorithms for the optimal parameters. For the proposed algorithms, we used UBIBCF and SVMReg as imputation methods in the MovieLens and FilmTrust datasets respectively. Table 5.6 shows that the proposed algorithms are scalable and practical as they have on-line cost less than or equal to the cost of other algorithms; however, they are more accurate. It must be noted that the baseline SVD-based CF algorithms do not perform very well compared to the user- and item-based CF applied over the original user-item rating matrix, which is in contrast with the work proposed by Vozalis and Margaritis (2007)[7]. The proposed *ItrSvd* algorithm performs the best out of all of them; however, it would incur the biggest off-line cost (depending on the number of iterations required to converge), and must be used given the availability of sufficient resources. The same is true for the $ImpSvd_{CF}^{hybrid}$, which gives more accurate results compared to baseline or simple CF; however, it would incur the greater cost. The

[7]It might be due to the reason that Vozalis and Margaritis (2007) did not use any significance weighting schemes, and used weighted sum prediction formula (Ghazanfar and Prügel-Bennett, 2010d) in the item-based CF.

ImpSvd algorithm comes the next, and can be used if we want the lowest off-line cost (as SVD is applied only once), fast on-line performance, and prefer (good) accuracy.

5.7 When and How Much Imputation is Required

As it is costly to do imputation by the proposed approaches, hence we investigate when it is beneficial to switch to the conventional approaches, which are cheap to compute. Next, we shed light on the following two questions: (1) when is imputation required? and (2) how much imputation is required?

5.7.1 When to do imputation by the proposed approaches

To answer this question, we look into the sparsity of users' and items' profiles. We only do imputation by the proposed approaches when a user's (or item's) profile is $\Theta_{sparse}\%$ sparse, where $\Theta_{sparse} = \{10, 20, \cdots, 100\}$. A value of $\Theta_{Sparse} = 10$ shows that the proposed approaches are used to fill in the missing values if the sparsity of a profile is less than $10\% = 0.1$, and the UserItemAvg approach is used otherwise. Figure 5.8(a) shows that the MAE is minimum at $\Theta_{sparse} = 100$. For the subsequent experiments, we choose to do imputation when $\Theta_{sparse} = 100$.

5.7.2 How much imputation is required

Users' (or items') profiles can be filled up to $\Theta_{dense}\%$ of the missing values in their profiles. To investigate how much imputation is necessary, we performed experiments with different values of Θ_{dense} and observed the corresponding MAE. A value of $\Theta_{dense} = 10$ shows that 10% missing values of a profile are filled using the proposed approaches and UserItemAvg is used for the remaining 90% missing values. Figure 5.8(b) shows that after $\Theta_{desne} = 60$, the change in the MAE becomes very small. Hence, 60% imputation is sufficient to achieve good accuracy.

Table 5.6: A comparison of the proposed algorithms with the existing ones in terms of cost and accuracy metrics. The SMVReg is used for the FilmTrust dataset and the UBIBCF is used for the remaining datasets as an imputation method prior to applying SVD. The best results are shown in bold font.

Algorithm	**On-line Cost**	**Best *MAE***			
		ML	**SML**	**FT**1	**FT**5
User-based CF with DV	$O(N^2)$	0.706 ± 0.000	0.746 ± 0.001	1.462 ± 0.008	1.419 ± 0.008
Item-based CF	$O(NM)$	0.705 ± 0.000	0.744 ± 0.001	1.433 ± 0.007	0.418 ± 0.006
Baseline SVD	$O(1)$	0.730 ± 0.001	0.774 ± 0.001	1.700 ± 0.005	1.483 ± 0.005
Baseline SVD-based IBCF	$O(N^2)$	0.741 ± 0.001	0.781 ± 0.001	1.702 ± 0.005	1.475 ± 0.006
$Idemsvd-2svd$	$O(N^2)$	0.738 ± 0.001	0.775 ± 0.001	1.682 ± 0.005	1.469 ± 0.006
$ImpSvd$	$O(1)$	0.682 ± 0.001	0.718 ± 0.001	1.411 ± 0.005	1.396 ± 0.005
$ImpSvd_{CF}^{ib}$	$O(N^2)$	0.691 ± 0.000	0.723 ± 0.001	1.417 ± 0.006	0.404 ± 0.006
$ImpSvd_{CF}^{ub}$	$O(NM)$	0.692 ± 0.000	0.722 ± 0.001	1.416 ± 0.005	0.401 ± 0.006
$ImpSvd_{CF}^{hybrid}$	$O(N^2)+O(NM)$	0.686 ± 0.000	0.719 ± 0.001	1.409 ± 0.005	1.394 ± 0.005
$ItrSvd$	$O(1)$	$\mathbf{0.659 \pm 0.001}$	$\mathbf{0.715 \pm 0.001}$	$\mathbf{1.401 \pm 0.001}$	$\mathbf{1.390 \pm 0.004}$

Table 5.7: Comparing the MAE observed in different imputation methods under **new item cold-start scenario**, for the SML dataset. The best results are shown in bold font

Imp. Method	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
ItemAvg	1.010 ± 0.003	0.876 ± 0.003	0.854 ± 0.003	0.840 ± 0.002	0.838 ± 0.002
UserAvg	0.876 ± 0.003	0.874 ± 0.003	0.872 ± 0.003	0.870 ± 0.002	0.867 ± 0.002
UserItemAvg	0.865 ± 0.003	0.833 ± 0.003	0.832 ± 0.003	0.824 ± 0.002	0.822 ± 0.002
UBCF	0.911 ± 0.003	0.829 ± 0.003	0.810 ± 0.003	$\mathbf{0.800 \pm 0.002}$	$\mathbf{0.790 \pm 0.002}$
IBCF	0.850 ± 0.003	0.834 ± 0.003	0.829 ± 0.003	0.818 ± 0.002	0.813 ± 0.002
UBIBCF	0.858 ± 0.003	0.826 ± 0.003	0.812 ± 0.003	0.802 ± 0.002	0.795 ± 0.002
SVMReg	$\mathbf{0.846 \pm 0.003}$	$\mathbf{0.824 \pm 0.003}$	$\mathbf{0.809 \pm 0.002}$	0.804 ± 0.002	0.802 ± 0.002

5.8 Discussion

What is evident from the experimental results is that the approximation of missing values in the sparse user-item rating matrix, prior to applying SVD, plays an important role in SVD-based recommendations. The literature proposes using item average to approximate the missing values in the sparse user-item rating matrix. We find out that this is not a feasible solution in terms of accuracy. Moreover, the convergence of the baseline approach is very slow in the case of the *ItrSvd* algorithm. It is worth noting that the SVD computation is an expensive task (despite the fact that it is done off-line) which implies that the conventional approaches are not pragmatic, and hence the proposed approaches should be used to save resources.

We note that the imputation approaches based on the content-based filtering (except the SVMReg) are not very accurate compared to the collaborative filtering ones in the recommender system domain, although content-based filtering has successfully been applied to text categorisation and it gives accurate results as well (Sebastiani, 2002). The reason is that the text categorisation and recommender system problems are quite different from each other. First, a user rates the same item differently under different contexts (Baltrunas, 2008) and the reason for the rating might be complex. Similarly, the positive feedback (Oard and Kim, 1998) given by a user, e.g. *purchased an item*, is dependent on the context; for example, a user might purchase an item as a gift, hence we cannot predict that they will purchase other similar items. Second, the user feedback (Pazzani and Billsus, 2007) in a recommender system is noisy, the observations, *did not buy an item*, or *did not watch a movie*, do not necessarily mean that the user is not interested in that item or movie. It may be the case that the user likes that item or movie but has not purchased or watched it. Third, the evaluation criteria for both are different; the recommender system usually provides a list of top items a user would like to consume, whereas text categorisation classifies a given document to a set of pre-defined categorisations. Furthermore, in text categorisation a document belongs to a single or a very few categories, whereas a user in a recommender system might be interested in a large number of different items (Zhang and Iyengar, 2002). Fourth, a user might change their taste over time and this temporal change in the profile is not shared by the text categorisation tasks. Making accurate recommendation given the noisy input is different and more difficult compared to the text categorisation task.

Based on the experimental results, we can underline five interesting points: (1) the results of different imputation approaches are dataset-dependent and no approach is a panacea. Due to dataset characteristics—data distribution, scale, and sparsity—one approach might be very good for one dataset while it may fail to produce good results for the second dataset; (2) collaborative filtering and SVM provide more accurate and much more computationally tractable results under all experiments; (3) although the conventional approaches are straightforward to implement, they do not provide good

results. The same is true for many classification and regression approaches; (4) the hybrid recommender system algorithms provide more accurate recommendations than the individual ones. Different recommendation algorithms, if combined in a systemic way, have complementary roles for recommendation generation; and (5) different imputation methods can be chosen depending the different circumstances and priorities—time and frequency of running the off-line computation, required accuracy, required recommendation time—and available resources (e.g. the content features and memory).

5.9 Conclusion and Future Work

This chapter makes the following contributions to the state-of-the-art in recommender systems:

1- We show that SVD-based recommendations highly depend on the imputation methods used to approximate the missing values in the user-item rating matrix. We provide the best imputation methods and empirically show that they significantly outperform the traditional approach—item average—used extensively in the literature. We show that the traditional approach fails to produce accurate recommendations under cold-start scenarios and sparse datasets.

2- We show that the results obtained by applying collaborative filtering over the reduced user-item rating matrix vary with the imputation methods used to approximate the missing values in the user-item rating matrix, and moreover point out under what conditions these results are significant over SVD-based results.

3- We show that in the case of *ItrSvd*—an algorithm that combines SVD with the expectation maximisation algorithm to estimate the missing values—the proposed approaches converge more quickly and produce better recommendations compared to the traditional one.

As a future work, we would like to explore in detail the questions discussed in Section 5.7. Another avenue for future work would be to incorporate external sources of information, such as Ontology of items, which might improve the results, particularly under sparse conditions. Furthermore, Boosting algorithms, such as AdaBoost algorithm (Witten and Frank, 1999) can be used for increasing the performance of classifiers, such as C4.5, which is a subject of future research.

Chapter 6

Using Clustering Algorithms to Solve the Gray-Sheep Users Problem

6.1 Introduction

Two of the important design objectives of a recommender system (refer to Chapter 1) are accuracy and scalability. In the Collaborative Filtering (CF) domain, they are in conflict, since the less time an algorithm spends searching for neighbours, the more scalable it will be, but produces worse quality recommendations. The CF approaches based on K-means clustering algorithms have been proposed to increase the scalability of recommender systems. We investigate how to improve the quality of clusters and recommendations focusing on the following key issues:

1- How do different centroid selection approaches affect the quality of clusters/recommendations?

2- How does the choice of distance metric affect the quality of clusters/recommendations?

Humans typically do not have predictable simple taste—they rate items differently and the reasons for rating an item are likely to be complex. In the CF domain, the correlation coefficient (in the case of Pearson correlation) between two users varies between 1, indicting absolute agreement, to -1, indicating absolute disagreement between two users. Based on the correlation coefficient, we can categorise users into two main classes[1]: (1)

[1]Some authors have used another class "black-sheep" for the users having no (or very few) other users with whom they correlate. The CF-based algorithms cannot make predictions for these users (Su and Khoshgoftaar, 2009).

white sheep—the users who have high correlation value with many other users; and (2) gray-sheep—the users who partially agree/disagree with other users and have low correlation coefficient with almost all users.

In this chapter, we systematically explore the gray-sheep users problem. Specifically, we look at four key questions:

1- How can the gray-sheep users be effectively detected in a recommender system?

2- Does the presence of the gray-sheep users affect the recommendation quality of the community?

3- How do the CF algorithms perform over these users?

4- How do the text categorisation algorithms trained on the content profiles perform over these users?

We proposed a clustering solution to detect the gray-sheep users in off-line fashion. We offered a *switching hybrid recommender system* (Burke, 2002) and showed that the proposed approach reduces the recommendation error rate for the gray-sheep users while maintaining reasonable computational performance. To the best of our knowledge, this is the first attempt to propose a formal solution to satisfy the needs of gray-sheep users. We evaluate our algorithm over the MovieLens and FilmTrust datasets.

The rest of the chapter has been organised as follows. In Section 6.2, we present the related work by giving an overview of different clustering algorithms and shed light on the gray-sheep users problem. In Section 6.3, we present various centroid selection algorithms. In Section 6.4, we discuss various distance measures that we have used in this work. We outline our algorithm to detect the gray-sheep users in Section 6.5. In Section 6.6, we discuss the results in detail, followed by the conclusion in Section 6.7.

6.2 Related Work

In this section, we give a brief overview of clustering algorithms that have been used in recommender systems. We then discuss the gray-sheep users problem and describe how this problem has been overlooked by the recommender system's community.

6.2.1 Clustering in recommender systems

Clustering belongs to unsupervised classification algorithms, whose goal is to discover natural grouping (clustering) of patterns (observations, data points, etc.) (Witten and Frank, 1999). There are two main types of clustering algorithms (Jain, 2010; Berkhin,

2002): hierarchical and partitional. Hierarchical clustering algorithms produce a nested series of partitions either in agglomerative mode—starting with each pattern in a distinct cluster and merging the most similar pairs of clusters successively to form a cluster hierarchy—or divisive mode—starting with each pattern into a single cluster and splitting each cluster into smaller clusters until some stopping criteria are met. Partitional clustering algorithms do not impose a clustering hierarchy and find all clusters once as a partition. Examples of hierarchical clustering algorithms include single link, complete link, and average link and those of partitional clustering algorithms include K-means, graph theoretic, and expectation maximisation.

Several CF recommendation approaches based on the partition clustering algorithms have been proposed (Sarwar et al., 2002b; Xue et al., 2005; Rashid et al., 2006). In Sarwar et al. (2002b) the authors proposed an approach, which divides the user-item rating matrix into k non-overlapping partitions, using a variant of K-means clustering algorithm called Bisecting K-means clustering. To find neighbours for an active user, it scans the cluster where the active user belongs, and generates recommendation by picking the top most similar users (i.e. neighbours) from that cluster. This approach has been extended in Xue et al. (2005), where the authors used the active user's average rating and the average deviated rating given to a target item in a cluster to approximate the missing values in the user-item rating matrix. They assigned different weights to items that have been rated by the user and items that were predicted using approximation function. They claimed that their approach increases the accuracy and efficiency of recommendations by overcoming the problems of data sparsity and scalability.

The proposed evaluation criteria in Xue et al. (2005) is somewhat biased. For example, for the MovieLens dataset, the authors used the last 200 users as the test and remaining 300 as the training users. In our opinion, a cross validation scheme (e.g. k-fold cross validation, leave one out cross validation, etc.) or at least a random selection of users (e.g. 20% randomly selected users as test users) should have been used for this purpose. The reason is that, using this approach, it is possible that one might select users with less ratings in the test set and with more ratings in the training set.

A highly scalable algorithm using a variant of K-means clustering algorithm and CF has been proposed by Rashid et al. (2006). This algorithm makes predictions using the following three steps: (1) the similarity between an active user and k other centroids is computed using the Pearson correlation, (2) the l ($l \leq k$) most similar centroids, called neighbours of the active user, are selected, and (3) the prediction on target item is made using the ratings provided by neighbours. They claimed that the proposed approach gives results comparable to the conventional CF approaches.

Several hierarchical clustering algorithms have been employed to solve the recommendation problem (Kohrs and Merialdo, 1999; Kelleher and Bridge, 2003; Uchyigit and Clark, 2004; Haruechaiyasak et al., 2005; Shepitsen et al., 2008). In Kelleher and Bridge

(2003), the authors proposed a clustering approach which recursively splits successive datasets into child clusters by building a binary tree of clusters using K-means clustering algorithm, where the root represents the whole dataset. The average rating of the cluster for an item, to which the active user belongs, is used as recommendation. If no one in a cluster has rated a target item, then the algorithm continues climbing up to the parent cluster until it finds a cluster which has rated that item or it reaches the root node. This algorithm has been used in Bridge and Kelleher (2002), where the author determine how the conventional CF, users, and items clusters are affected by the sparsity. Another example of the hierarchical clustering is given in Shepitsen et al. (2008), where the authors proposed a personalisation algorithm for recommendation of resources in folksonomies, which relies on hierarchical tag clusters. They applied K-means and hierarchical agglomerative clustering algorithms over the last.fm (www.last.fm/home) and delicious (www.delicious.com) datasets and claimed that a later clustering algorithm gave better results. They represented resources and user profiles with vectors of tags and used the cosine-similarity measure for computing the similarity. Furthermore, they modified the clustering algorithm to choose clusters according to the user context, and claimed that this technique is suited to folksonomies which are sparse and where tags can take on a range of meanings across different topic areas, like delicious.

Various hybrid recommender systems employing the clustering techniques have been proposed (Clerkin et al., 2003; Puntheeranurak and Tsuji, 2007); for example, Puntheeranurak and Tsuji (2007) offered a hybrid recommender system using fuzzy K-means clustering. They linearly combined the results of CF applied over the original and clustered data and claimed that it outperforms the conventional CF. Furthermore, several other clustering algorithms have been employed in the recommender system domain, for example, Connor and Herlocker (2001) applied random partition, genre partition, average link hierarchical agglomerative, k-Metis, h-Metis, and multilevel k-way graph partitioning over the MovieLens dataset using Pearson correlation as a similarity measure, and showed that k-Metis produced better results than others in terms of MAE and coverage.

We find out that the literature of the clustering algorithms is very rich ranging from partitioning-based clustering to hierarchical clustering spanning a number of algorithms. We have used the K-means clustering algorithm to partition the dataset into different clusters, as it has effectively been applied in different domains (Berkhin, 2002; Jain, 2010). In the recommender system domain, different authors, for instance Rashid et al. (2006), have claimed that K-means-based collaborative filtering yields results comparable to several other recommendation algorithms. Furthermore, the K-means clustering algorithm has small computation complexity $\big(O(M \times k \times itr)\big)$, which is linear in the number of users being clustered (M), number of clusters (k), and number of iterations (itr).

To make predictions we can use the technique proposed in Sarwar et al. (2002b) and Xue et al. (2005) or in Rashid et al. (2006). The approaches proposed in Sarwar et al. (2002b) and Xue et al. (2005) are not very scalable as there might be many potential neighbours for an active user in a cluster. Moreover, their coverage is arguable, as they only find the neighbours against an active user in the cluster where the active user belongs. If the quality of the clusters are not very good, it would result in increased error and reduced coverage. The approach proposed in Rashid et al. (2006) claimed to be more scalable and accurate, and yields results comparable to several other algorithms such as conventional CF, singular value decomposition, and Personality diagnosis (Pennock et al., 2000). We have employed this algorithm (denoted by CCF in the results section) for making predictions, with the exception that we use the conventional K-means clustering algorithm for partitioning the user-item rating matrix. Furthermore, we choose centroids using the centroid selection algorithms as described in Section 6.3.

6.2.2 Gray-sheep users problem in recommender systems

The gray-sheep users problem was highlighted in Claypool et al. (1999), where the authors proposed an on-line hybrid recommender system for news recommendation. They used a weighted average approach to combine the CF approach with the content-based filtering (CBF) approach, where the weights are learned per user basis. This approach is very expensive both in terms of memory requirement and time. In Cantador et al. (2008), the authors offered a hybrid recommender system which builds a multi-layered community of interests by dividing the user profiles, defined in domain Ontologies, into different areas of interest. The authors divided the users' and items' profiles into groups based on the cohesive interests, and claimed that taking into account the profiles at different semantic interest layers, while establishing the similarities between two users, can provide better correlation measure and hence might help in overcoming the gray-sheep users problem. The problem with this approach is that it require labour-intensive domain Ontology building stage. Neither of these approaches proposes any formal solution focussing specifically on gray-sheep users, nor provides any results.

There has been some work on the gray-sheep users in other domain; for example, Wurst (2005) presented the agent-based simulation for distributed knowledge management. They instantiated this framework for the CF domain using the MovieLens dataset. They claimed that the gray-sheep agents—agents belonging to several communities—do not affect the regular agents—agents belonging to one community, which is in contrast with our results (refer to Section 6.6). As it was a simulation, they did not describe a method to identify these users and satisfy their needs. Furthermore, none of the aforementioned approaches provides any benchmarks, making it unclear how to compare the proposed algorithm with them.

A closely related problem is the *long tail problem*. Park and Tuzhilin (2008) proposed a scheme for dividing the items into heads and tails sections and applying clustering algorithms over the tail part only for overcoming the long tail problem associated with a recommender system.

6.3 Centroid Selection Approaches

As the conventional K-means clustering algorithm (Jain, 2010) randomly selects the k initial centroids, an important research question would be, *"how does the quality of clusters vary with the choice of different initial centroids"*? In Arthur and Vassilvitskii (2007), the authors proposed an algorithm, namely K-means++, that uses a probabilistic approach for choosing the centroids and claimed that it yields much finer clusters than the K-means clustering algorithm.

We applied a modified version of the K-means++ clustering algorithm for clustering the user-item rating matrix. We used the K-means++ concept over the so-called *power users*—users that have rated a large number of items in a recommender system (Herlocker et al., 2004). The concept of the power user has been used in Amatriain et al. (2009), where the authors employed the power users as neighbours of an active user for producing the scalable CF-based recommendations. The authors used the term "experts" to emphasise the importance of these users. Other researchers (Zanardi, 2011; Zanardi and Capra, 2011) have claimed that in the tag-based social tagging website such as CiteULike (www.citeulike.org), there are a relatively small number of users, called *leaders*, which provide a large number of tags, and other users in the system, known as *followers*, who just follow these tags. They claimed that taking into account the opinion of only the leaders, one can provide scalable and accurate recommendations. We propose to use the power users as the initial centroids and the rationale behind this is that it might speed up the convergence and improve the quality of the resulting clusters.

Let u_p denote the user who has rated the maximum number of items in the training set. Rather than using the raw rating count, we normalise the number of ratings provided by user u by dividing it by the number of ratings provided by u_p, i.e.

$$\mathbb{P}(u) = \frac{|\mathcal{I}_u|}{|\mathcal{I}_{u_p}|}, \tag{6.1}$$

where $|\mathcal{I}_u|$ and $|\mathcal{I}_{u_p}|$ denote the number of items rated by users u and u_p respectively. Let $\mathcal{U}^{power} = \{ u_1, u_2, \cdots, u_z \}$ be the set of z power users having the highest value for $\mathbb{P}(u)$ (i.e. $\mathbb{P}(u_m) > \mathbb{P}(u_n) : \forall_{u_m \in \mathcal{U}^{power} AND\, u_n \notin \mathcal{U}^{power}}$); $G = \{g_1, g_2, \cdots g_k\}$ represents the k clusters and $C = \{c_1, c_2, \cdots c_k\}$ denotes the k centroids of the corresponding clusters; $\mathcal{D}_{c_j}$ denotes the set of user-item pairs that have been rated in cluster g_j represented with centroid c_j (i.e. $(r_{i,u}|(i,u) \in \mathcal{D}_{c_j})$); r_{i,c_j} denotes the rating given by centroid c_j on item

i, i.e. $r_{i,c_j} = \frac{1}{|\mathcal{D}_{c_j}|} \sum_{i,u \in \mathcal{D}_{c_j}} r_{i,u}$; and $\text{dist}(u)$ denotes the shortest distance from user u to the closest centroid we have already chosen.

Algorithm 5 : **CentroidSelect**, Selects k users as centroids from the dataset
Input: $\mathcal{U}$, training users; k, the number of clusters; $\mathbb{P}(u)$, the normalised rating count of a user; pow_{thr}, a threshold to detect the power users
Output: $\{ c_1, c_2, \cdots c_k \}$, k centroids

procedure $KMeans(\mathcal{U}, k)$
  Choose k centroids, $\{ c_1, c_2, \cdots c_k \}$, at random without repeating.
  return $\{ c_1, c_2, \cdots c_k \}$ ▷ k centroids
end procedure

procedure $KMeansPlus(\mathcal{U}, k)$
  repeat
    Choose the initial centroid c_1 uniformly at random from $\mathcal{U}$.
    Choose the next centroid c_i by selecting $c_i = u' \in \mathcal{U}$ with probability:

$$\text{Prob} = \frac{\text{dist}(u')^2}{\sum_{u \in \mathcal{U}} \text{dist}(u)^2}.$$

  until k centroids are found
  return $\{ c_1, c_2, \cdots c_k \}$ ▷ k centroids
end procedure

procedure $KMeansPlus^{power}(\mathcal{U}, k)$
  $\mathcal{U}^{power} = \emptyset$
  for all $u \in \mathcal{U}$ **do**
    if $\mathbb{P}(u) > \text{pow}_{\text{thr}}$ **then**
      $\mathcal{U}^{power} = \mathcal{U}^{power} \cup u$
    end if
  end for
  $\{ c_1, c_2, \cdots c_k \} = KMeansPlus\ (\mathcal{U}^{power}, k)$
  return $\{ c_1, c_2, \cdots c_k \}$ ▷ k centroids
end procedure

procedure $KMeansPlus^{ProbPower}(\mathcal{U}, k)$
  repeat
    Choose the initial centroid c_1 to be u_p.
    Choose the next centroid c_i by selecting $c_i = u' \in \mathcal{U}$ with the probability:

$$\text{Prob} = \frac{\left(\frac{\text{dist}(u')^2}{\sum_{u \in \mathcal{U}} \text{dist}(u)^2} + \frac{\mathbb{P}(u')^2}{\sum_{u \in \mathcal{U}} \mathbb{P}(u)^2} \right)}{2}.$$

  until k centroids are found
  return $\{ c_1, c_2, \cdots c_k \}$ ▷ k centroids
end procedure

Algorithm 5 outlines the pseudo code of the different centroid selection algorithms. In Algorithm 5, the variant of the centroid selection algorithm denoted by $KMeans$ uses the

randomly selected k users from the training set as centroids. The variant of the centroid selection algorithm denoted by $KMeansPlus$ implements the K-means++ algorithm. Specifically, it uses a randomly selected user as an initial centroid and then keeps on choosing the next centroid with a probability which is proportional to the shortest distance of the candidate centroids with the existing one(s) until all k centroids have been chosen. The variant of the centroid selection algorithm, denoted by $KMeansPlu^{power}$, applies the K-means++'s centroid selection concept over the power users only. From steps 13 to 18, we identify power users, $\mathcal{U}^{power}$—users having $\mathbb{P}(u) > \text{pow}_{\text{thr}}$. We then call $KMeansPlus$ procedure with these users as candidate centroids, which chooses centroids using the K-means++'s centroid selection algorithm. The last variant of the centroid selection algorithm, denoted by $KMeansPlus^{ProbPower}$, aims at finding the centroids with probability proportional to distance and the number of ratings (see step 25 in Algorithm 5). This variant is based on the rationale that users who have the maximum distance (minimum similarity) with the current ones(s) and who have rated a large number of items might provide potential benefits, such as reduced error and increased coverage.

6.4 Distance Measure

In Algorithm 5, the dist function measures the distance between a centroid and a user. In our case, we are measuring the similarity (sim) between a user and a centroid. As the distance between two points is maximum when the similarity is zero and vice versa; we can use a simple equation to model the distance function as follows:

$$\text{dist} = \begin{cases} \frac{1}{\text{sim}} & if\ \text{sim} \neq 0\,, \\ \text{MAX}_{\text{DIST}} & otherwise, \end{cases} \tag{6.2}$$

where MAX_{DIST} (chosen as 1000 in our case) represents the maximum distance between two points. In the case of the Pearson correlation, the similarity between two points can be negative, which cannot be modeled by equation 6.2. To avoid this, we add 1 to all similarities returned by the Pearson correlation (i.e. $sim(u) \leftarrow sim(u)+1\,, \forall_{u \in \mathcal{U}}$), before applying equation 6.2, while experimenting with the centroid selection algorithms.

A common problem with the CF approach is that if there are less common items between an active user and the centroid, then it will not perform well. The reason is that, it only takes into account the intersection of the items that both the active user and the centroid have voted on. If we assume some default votes for items a user or centroid has not voted on, then we can extend the correlation over the union of items, which can increase the coverage and accuracy of the system. This concept is similar to the one used by Xue et al. (2005) to smooth the missing values in a user's profiles. The difference is that Xue et al. (2005) find the similarity between a user and each cluster and then use the

average of most similar cluster to smooth the missing values in a user's profile; however, we are using a user's and cluster's average to smooth the missing values in a user's and cluster's profile respectively. Xue et al. (2005) used the Pearson correlation as a distance to measure the similarity between a cluster and the user; nevertheless as we will see in the results section, this similarity measure fails under sparse datasets (e.g. FilmTrust). We assume the users' and clusters' averages for the missing values as follows:

$$r = \begin{cases} \frac{1}{|\mathcal{D}_u|} \sum_{i \in \mathcal{D}_u} r_{i,u}, & if\ r_{i,u} = \varnothing\,, \\ \frac{1}{|\mathcal{D}_{c_j}|} \sum_{i,u \in \mathcal{D}_{c_j}} r_{i,u}, & if\ r_{i,c_j} = \varnothing. \end{cases} \tag{6.3}$$

In this work, we used the following distance measures: (1) Pearson Correlation (PCC), (2) Pearson Correlation with Default Votes (PCCDV), (3) Vector Similarity (VS), and (4) Vector Similarity with Default Votes (VSDV). The results of using these distances are shown in the results section.

6.5 Detecting Gray-Sheep Users: A Clustering Solution

This concept has been inspired from the shape detection in the image processing domain. While generating clusters for shape detection, a separate cluster is made for the features that are not very similar with the current clusters[2]. Other researchers have found that this can result in a decrease of error in shape detection (Ramanan, 2010). Gray-sheep users can be handled in the same way, and can be separated into a distinct cluster using our proposed algorithm, Algorithm 6.

In step 2, we choose k initial centroids using the CentroidSelect centroid selection algorithm. Then we initialise the loop counter, which counts the number of iterations for which the algorithm will be executed. In step 5, we assign each user to the most similar cluster found measuring the similarity between the user and the corresponding centroid. In step 6, we update centroids which now contain the set of user-item pairs rated by all users belonging to that cluster. We then increment the loop counter and check its value with itr, which is the maximum number of iterations the algorithm will be executed. If the loop counter is less than itr, then we keep executing the steps from 5 to 7, which are essentially the same as those of the K-means clustering algorithm (Arthur and Vassilvitskii, 2007). Nevertheless, if the value of the loop counter is equal to itr, then the steps from 9 to 15 are executed, which detect the gray-sheep users by grouping users having similarity with the most similar cluster, less than a pre-defined threshold ω, into a separate cluster. In step 17, we return the clustered data.

We can think of each centroid as a vector of length N, the number of items in the system. Any centroid c_j can be represented as: $c_j = \{r_{i_1,c_j}, r_{i_2,c_j}, \cdots, r_{i_N,c_j}\}$, where r_{i,c_j} denotes

[2]For example, a separate cluster is made for a distinct pimple on the face. The details are beyond the scope of this thesis. Please refer to Ramanan (2010).

the rating given by centroid c_j on item i, i.e. $r_{i,c_j} = \frac{1}{|\mathcal{D}_{c_j}|}\sum_{i,u\in\mathcal{D}_{c_j}} r_{i,u}$. If no one has rated the item in that cluster then the rating value is essentially zero in the case that we use a distance measure without default votes, and is estimated using equation 6.3 otherwise.

Algorithm 6 : **ClustAndDetect**, Clusters the user-item rating matrix into $k+1$ clusters, and groups the gray-sheep users into a separate cluster
Input: $\mathcal{U}$, training users; k, the number of clusters; itr, the number of iteration for the K-means clustering algorithm; ω, a similarity threshold to detect the gray-sheep users
Output: G, C; the clusters with corresponding centroids

procedure GSUDETECTOR($\mathcal{U}, k, \text{itr}, \omega$)
  $C =$ **CentroidSelect**
  $t = 0$
  repeat
    Set the cluster g_j, for each $j \in 1, ..., k$, to be the set of users in $\mathcal{U}$ that are closer to c_j than they are to c_l for all $l \neq j$.
    Set c_j, for each $j \in 1, ..., k$, to be the centre of mass of all users in g_j, i.e.

$$c_j = \frac{1}{|g_j|}\sum_{u\in g_j} u\,.$$

    $t = t + 1$
    if $t =$ itr **then**
      Create a new centroid c_{k+1}
      for all $u \in \mathcal{U}$ **do**
        if $\text{sim}(u) < \omega$ **then**
          Assign user u to cluster g_{k+1}
        end if
      end for
    end if
  until $(t =$ itr$)$
  return (G, C)
end procedure

6.6 Results and Discussion

In this section, we show the results of recommendation algorithms over different kinds of users. We assume that the CF-based algorithms would not perform well over gray-sheep users, because they have partial agreement with the rest of the community. This assumption argues that the recommendations can be generated based on the content profile of these users (by training the machine learning classifiers) and ignoring the contributions of the community (neighbours). We trained (using the Weka library) the following Text Categorisation (TC) algorithms over the content profiles of users: K Nearest Neighbours (KNN), Naive Bayes classifier (NB), decision tree (C4.5), Support Vector Machines Classification (SVMClass), and Support Vector Machines Regression

Table 6.1: Checking the effect of different distance measures in the K-means clustering algorithm over the validation set. The number of clusters (k) and iteration (itr) have been fixed to 50 and 5 respectively. The randomly selected 50 users have been selected as initial centroids. The distance measure denoted by PCCDV (Pearson Correlation with Default Votes) gives the best results shown in bold font.

	MAE		**Coverage**	
Distance Measure	**FT1**	**SML**	**FT1**	**SML**
PCC	1.518 ± 0.019	$\mathbf{0.784 \pm 0.005}$	94.010 ± 0.180	99.953 ± 0.025
PCCDV	$\mathbf{1.515 \pm 0.020}$	0.785 ± 0.006	$\mathbf{95.519 \pm 0.180}$	**100**
VS	1.525 ± 0.034	0.786 ± 0.004	95.401 ± 0.180	99.967 ± 0.017
VSDV	1.533 ± 0.030	0.788 ± 0.006	95.510 ± 0.190	**100**

(SVMReg). Furthermore, we tuned them for the optimal parameters over the validation set.

6.6.1 Learning the optimal system parameters

We give the tuning of important parameters. The tuning of other parameters (the optimal number of clusters, the optimal number of iterations in the K-means clustering algorithm, and the optimal number of neighbours in CCF algorithm) is given in Appendix C.

6.6.1.1 Distance measure

The results of the Clustering-based Collaborative Filtering (CCF) algorithm for different distance measures are shown in Table 6.1. Table 6.1 shows that (in general) the PCC with default votes gives insignificantly better results than others in terms of MAE and coverage. We observe that for the FilmTrust dataset, the coverage degrades in the case of the PCC and VS distance similarity measures. It is because the FilmTrust dataset is very sparse and it is not possible to find reliable similarities between a user and the cluster centroid. Considering the results, we choose the PCCDV similarity measure for the subsequent experiments.

6.6.1.2 Centroid selection approaches

We experimented with different centroid selection algorithms under various cluster sizes and number of iterations. As an example, we show results at $k = 60$ and $itr = 10$ in Table 6.2; however, similar results were observed at other cluster sizes and iterations. We observe that the $KMeansPlus^{ProbPower}$ algorithm gives better results—in terms of

the MAE and coverage—than others; however, the improvement is not significant. The same was true for the convergence rate (results are not shown).

Table 6.2: Checking the effect of different centroid selection algorithms over the validation set. The number of clusters (k) and iterations (itr) have been fixed to 60 and 10 respectively. The PCCDV distance measure has been used for measuring the similarity between a centroid and a user. The best results are shown in bold font.

Centroid Selection	**MAE**		**Coverage**	
	FT1	**SML**	**FT1**	**SML**
$KMeans$	1.518 ± 0.020	0.783 ± 0.006	95.519 ± 0.175	**100**
$KMeansPlus$	1.515 ± 0.020	0.784 ± 0.007	95.518 ± 0.179	**100**
$KMeansPlus^{power}$	1.513 ± 0.022	$\mathbf{0.782 \pm 0.005}$	95.523 ± 0.171	**100**
$KMeansPlus^{ProbPower}$	$\mathbf{1.512 \pm 0.022}$	0.783 ± 0.006	$\mathbf{95.524 \pm 0.168}$	**100**

Table 6.3: Checking the within-cluster similarity of different centroid selection algorithms at a cluster size of 60 over the validation set. The PCCDV distance measure has been used for measuring the similarity between a centroid and a user. "TotalSim" represents the total inter-cluster similarity between each user and the cluster it belongs to. The best results are shown in bold font.

Centroid Selection	**Dataset**	**TotalSim observed at different number of iterations (Itr)**				
		Itr: 2	Itr: 4	Itr: 6	Itr: 8	Itr: 10
$KMeans$	**FT1**	863.465 ± 13.66	1876.580 ± 16.06	2905.367 ± 17.02	3936.42 ± 18.80	4969.706 ± 23.19
$KMeansPlus$		859.966 ± 11.06	1877.867 ± 15.37	2913.732 ± 27.05	3955.408 ± 40.06	4999.748 ± 52.64
$KMeansPlus^{power}$		861.975 ± 37.84	1886.001 ± 73.03	2926.699 ± 108.98	3973.713 ± 142.38	5022.389 ± 174.99
$KMeansPlus^{ProbPower}$		$\mathbf{1022.715 \pm 11.78}$	$\mathbf{2132.155 \pm 21.20}$	$\mathbf{3252.587 \pm 28.64}$	$\mathbf{4373.327 \pm 36.91}$	$\mathbf{5495.665 \pm 44.61}$
$KMeans$	**SML**	339.501 ± 23.50	724.634 ± 35.03	1120.735 ± 32.67	1517.206 ± 30.99	1795.767 ± 161.79
$KMeansPlus$		333.344 ± 12.03	720.172 ± 9.52	1113.141 ± 10.48	1506.744 ± 13.68	1820.095 ± 163.87
$KMeansPlus^{power}$		353.691 ± 15.25	$\mathbf{746.031 \pm 9.50}$	1140.712 ± 26.25	1535.806 ± 32.43	1814.008 ± 202.41
$KMeansPlus^{ProbPower}$		$\mathbf{354.426 \pm 15.96}$	745.214 ± 9.36	$\mathbf{1143.308 \pm 12.16}$	$\mathbf{1538.061 \pm 15.91}$	$\mathbf{1855.608 \pm 172.96}$

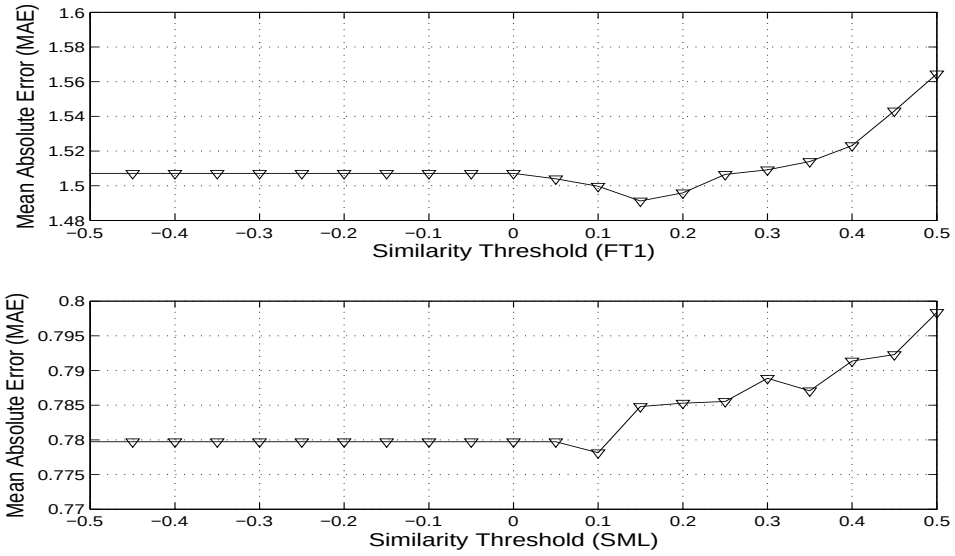


Figure 6.1: Finding the optimal similarity threshold (ω) for the MovieLens (SML) and FilmTrust (FT1) datasets through the validation set.

Table 6.3 shows the total within-cluster similarity between each user and the cluster it belongs to. Formally, for each user $u \in \mathcal{U}$ in the training set, we measure the within-cluster total similarity, i.e.

$$\text{TotalSim}(\mathcal{U}, G) = \sum_{g_j \in G} \sum_{u \in \mathcal{U}^{g_j}} \text{sim}(u, c_j), \tag{6.4}$$

where $\mathcal{U}^{g_j}$ represents the number of users in cluster g_j and c_j is the centroid of the corresponding cluster g_j. We note that the within-cluster similarity of $KMeansPlus^{ProbPower}$ is the highest amongst the rest for each iteration. Considering the results, we choose $KMeansPlus^{ProbPower}$ centroid selection algorithm for the subsequent experiments.

6.6.1.3 Optimal similarity threshold to detect the gray-sheep users

The similarity threshold ω parameter determines and controls the number of gray-sheep users. A large value of ω (e.g. $\omega = 1$) would results in all users being gray-sheep users, whereas a smaller value of ω (e.g. $\omega = -1$) would result in no user being a gray-sheep user. We changed the value of ω from 1.0 to -1.0 with a difference of 0.05 and measured the corresponding MAE of the users not identified as gray-sheep users. The value of ω that gives the minimum MAE is termed as the optimal value of ω. Figure 6.1 shows how the MAE changes with a change in ω. We observe that the MAE is minimum at $\omega = 0.1$ and $\omega = 0.15$ for the SML and FT1 datasets respectively. A further increase in the similarity threshold increases the MAE. We chose these optimal values for the subsequent experiments. Hence the answer to the question: *"How can the gray-sheep users be effectively detected in a recommender system?"* is that these users can be

detected using a K-means clustering algorithm, where the similarity threshold to isolate these users from the rest of clusters can be found empirically.

6.6.2 Results of CF-based algorithms for different types of users

We show the results of the Clustering-based Collaborative Filtering (CCF) algorithm over (1) all users, (2) the gray-sheep users[3], and (3) the users not classified as gray-sheep users. Taking the results of Table 6.4 into account, the answer to the question *"Does the presence of the gray-sheep users affect the recommendation quality of the community?"* is that it is dataset-dependent. Their presence does not make any difference to the recommendation quality, in the case of the MovieLens dataset, because only 25 users are detected as gray-sheep users. However, it does make some difference in the case of the FilmTrust dataset (with 1.47% improvement in MAE), as a greater number of users are detected as the gray-sheep users (283). Considering these results, we claim that the presence of a large number of gray-sheep users might significantly affect the recommendations quality of the community (i.e. the MAE for all users).

We observe that the performance of the CCF algorithm suffers the most for gray-sheep users, because they rely on the similar users (neighbours). In this case, the correlation coefficient is poorly approximated, and thus less reliable recommendations are produced. The percentage increase in the MAE for the gray-sheep users, compared with the remaining users, is 2.29% for the MovieLens dataset and 8.75% for the FilmTrust dataset. Taking these results into account, the answer to the question: *"How do the CF algorithms perform over these users?"* is that the performance of the CF algorithms suffer for these users[4].

6.6.3 Results of TC-based algorithms for the gray-sheep users

To answer the question: *"How do the text categorisation algorithms trained on the content profiles perform over these users?"*, we perform experiments with different Text Categorisation (TC) algorithms and the results are shown in Table 6.5. The results show that these algorithms improve the recommendation quality compared to the CF ones. We note that the SVMReg outperforms the rest, with 3.68% and 8.32% improvement over the CCF's result in the case of the MovieLens and FilmTrust datasets respectively.

[3]After detecting the gray-sheep users, the steps of algorithm 1 from 8 to 15 are not executed. This ensures that a user groups with the most similar clusters.

[4]The results were the same for other conventional CF algorithms. Refer to Appendix C for details.

Table 6.4: The performance of the Clustering-based CF (CCF) algorithm over different types of users. "All" represents all users, "GS" represents the gray-sheep users, and "Remaining" represents the users not identified as gray-sheep.

Metric	Dataset	Users		
		All	**GS**	**Remaining**
MAE	SML	0.772 ± 0.001	0.787 ± 0.012	0.769 ± 0.001
	FT1	1.492 ± 0.021	1.611 ± 0.042	1.470 ± 0.022
ROC-Sensitivity	SML	0.737 ± 0.002	0.701 ± 0.015	0.746 ± 0.002
	FT1	0.492 ± 0.008	0.412 ± 0.021	0.513 ± 0.008
Coverage	SML	100	100	100
	FT1	95.539 ± 0.080	91.243 ± 0.901	95.770 ± 0.071

Table 6.5: The performance in terms of MAE, ROC-sensitivity, and coverage of different algorithms computed over the gray-sheep users. The best results are shown in bold font.

Dataset	Approach	MAE	ROC-Sensitivity	Coverage
SML	SVMReg	$\mathbf{0.758 \pm 0.006}$	0.702 ± 0.004	**100**
	SVMCalss	0.781 ± 0.005	$\mathbf{0.704 \pm 0.004}$	**100**
	NB	0.811 ± 0.006	0.698 ± 0.005	**100**
	KNN	0.816 ± 0.006	0.666 ± 0.004	**100**
	C4.5	0.819 ± 0.005	0.642 ± 0.005	**100**
	CCF	0.787 ± 0.012	0.701 ± 0.015	**100**
	CBFCF	0.761 ± 0.009	0.698 ± 0.008	**100**
FT1	SVMReg	$\mathbf{1.477 \pm 0.008}$	$\mathbf{0.518 \pm 0.009}$	$\mathbf{99.993 \pm 0.005}$
	SVMCalss	1.483 ± 0.008	0.514 ± 0.010	$\mathbf{99.993 \pm 0.005}$
	NB	1.501 ± 0.009	0.511 ± 0.010	$\mathbf{99.993 \pm 0.005}$
	KNN	1.515 ± 0.010	0.504 ± 0.009	$\mathbf{99.993 \pm 0.005}$
	C4.5	1.522 ± 0.014	0.491 ± 0.011	$\mathbf{99.993 \pm 0.005}$
	CCF	1.611 ± 0.042	0.412 ± 0.021	91.243 ± 0.901
	CBFCF	1.497 ± 0.010	0.478 ± 0.012	$\mathbf{99.993 \pm 0.005}$

6.6.4 Combining the CF with CBF for the gray-sheep users

As described above, the accuracy of the recommender system increases if we use Content-Based Filtering (CBF) for the gray-sheep users; however, these users will not benefit from the individual advantages offered by the CF. The CF and CBF can be combined linearly and weights can be learned per user; nevertheless it would be computationally expensive. We propose a very simple scheme, which combines the CF and CBF for these users, by taking into account the similarity of a user with the most similar cluster (found after running Algorithm 6) and the number of ratings provided by a user

$$\check{r}_{i,u} = \Upsilon \times CCF + (1 - \Upsilon) \times CBF, \tag{6.5}$$

where $\Upsilon = \min\Big(1, \max\big(0, sim(u) + \mathbb{P}(u)\big)\Big)$. The intuition is based on the analysis that the CBF was better for users having relatively less ratings or having less similarity with the community of users. For the gray-sheep users the CBF has more weight initially

Table 6.6: The performance in terms of MAE, ROC-sensitivity, and coverage of different algorithms computed over all users. The best results are shown in bold font.

Dataset	Approach	MAE	ROC-Sensitivity	Coverage
SML	SVMReg	0.788 ± 0.003	0.689 ± 0.004	**100**
	SVMCalss	0.806 ± 0.003	0.687 ± 0.004	**100**
	NB	0.826 ± 0.004	0.685 ± 0.006	**100**
	KNN	0.832 ± 0.004	0.669 ± 0.005	**100**
	C4.5	0.847 ± 0.004	0.631 ± 0.006	**100**
	CCF	$\mathbf{0.772 \pm 0.001}$	$\mathbf{0.737 \pm 0.002}$	**100**
FT1	SVMReg	$\mathbf{1.485 \pm 0.009}$	$\mathbf{0.515 \pm 0.008}$	$\mathbf{99.990 \pm 0.007}$
	SVMCalss	1.489 ± 0.009	0.513 ± 0.010	$\mathbf{99.990 \pm 0.007}$
	NB	1.507 ± 0.012	0.512 ± 0.011	$\mathbf{99.990 \pm 0.007}$
	KNN	1.520 ± 0.011	0.499 ± 0.012	$\mathbf{99.990 \pm 0.007}$
	C4.5	1.527 ± 0.014	0.488 ± 0.014	$\mathbf{99.990 \pm 0.007}$
	CCF	1.492 ± 0.021	0.492 ± 0.008	95.539 ± 0.080

and the CF acquires more weight as these users rate more items, and their similarity increases with the community of users. Table 6.6 shows that the proposed approach (CBFCF) gives quite good results despite its simplicity.

6.6.5 A comparison of different algorithms for all users

Table 6.5 shows how different algorithms perform over the MovieLens and FilmTrust datasets. We observe that, for the MovieLens dataset, the CF-based algorithm performs the best, whereas for the FilmTrust dataset, the SVM regression performs the best. This is because the FilmTrust dataset is relatively sparse as compared to the MovieLens dataset.

We observe from Tables 6.5 and 6.6 that the CCF algorithm gives good results for the MovieLens and FilmTrust datasets; however, the performance degrades for the gray-sheep users. The reason is that gray-sheep users have unclear rating profiles, and in the worse case, we might find very few or no similar users (neighbours) for a gray-sheep user. The text categorisation approaches give good results, for these users, as they make effective use of users' content profiles that are used for making predictions.

6.6.6 Rating distribution of different kinds of users

We argue that a user cannot be identified as a gray-sheep user based solely on the number of ratings provided by the user. Figure 6.2 shows the rating distribution of all kinds of users for the FilmTrust dataset. We observe that 97% of the users identified as gray-sheep users ($0.97 \times 283 = 274$) have provided less than or equal to 5 ratings. Figure 6.2 further shows that 56% of users not identified as gray-sheep users ($0.56 \times 931 = 521$)

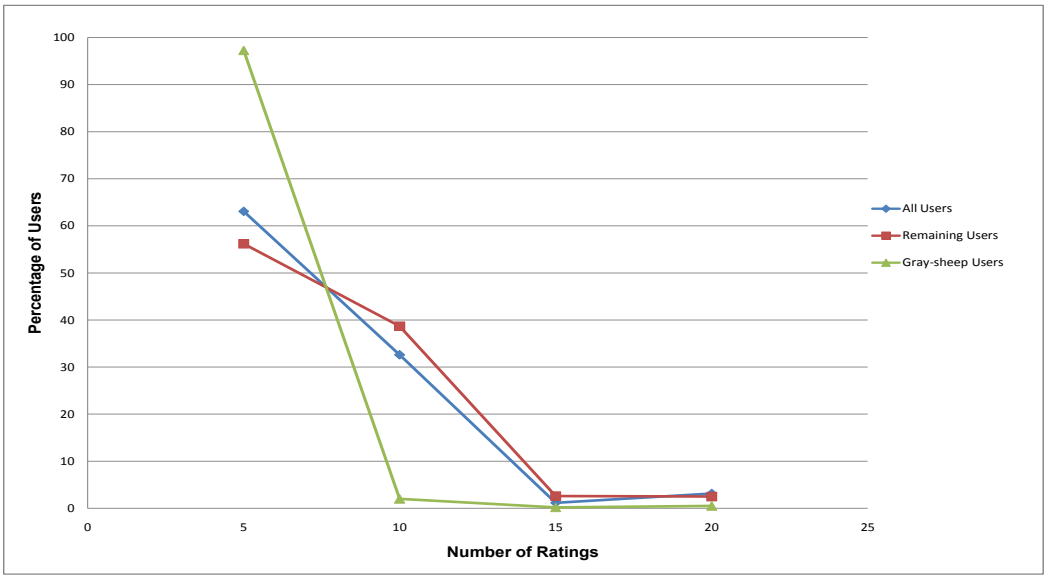


Figure 6.2: Rating distribution of FT1 datasets for different types of users.

have provided less than or equal to 5 ratings. Hence, we discover that there are roughly twice as many users as gray-sheep users who have provided less than or equal to 5 ratings and yet they are not identified as gray-sheep users by the algorithm. This analysis shows that there is no potential correlation between the rating count of a user and its being the gray-sheep user.

6.6.7 Complexity of the proposed solution

The training of SVM can be done off-line and it has a time complexity of $O(N^3)$. To make a prediction, SVM takes $O(n_{sv})$, where n_{sv} is the number of support vectors. The training cost[5] of the proposed clustering algorithm is $O(MN)$. To make a prediction for a user using the clustering technique, we find the active user's similarity with k other clusters, which takes $O(k)$ calculations. It must be noted that the on-line cost is less than the conventional hybrid algorithm; for example, the one offered by Claypool et al. (1999).

6.7 Conclusion

In this chapter, we focus on K-means clustering-based collaborative filtering algorithms. We propose new centroid selection approaches for the K-means clustering algorithm and analyse how they affect the quality of clusters and recommendations. We also check how the choice of distance metric affects the performance of recommendations. Our analysis shows that different centroid selection algorithms do not significantly affect the cluster

[5]In addition to this, we incur the cost of choosing the cluster centroids, which is $O(kM)$.

quality; however, the performance of the clustering algorithm varies with the distance measure.

Furthermore, we point out the gray-sheep users problem associated with a recommender system. We find out that the presence of gray-sheep users in a community of users poses two problems: (1) after the initial start-up phase of a system, as more and more users enter the system, for obvious statistical reasons the chances of finding other users with similar tastes increases and hence better recommendations can be made. The gray-sheep users are a potential problem for the CF-based systems because they do not get useful recommendations due to their idiosyncratic tastes, even after the initial start-up phase; and (2) they can negatively affect the recommendations of the rest of the community. A clustering solution is proposed to detect these users and the recommendations for these users are generated based on the SVM regression trained on the content profiles of the users, whereas for other users based on the clustering-based collaborative filtering algorithm.

Although the K-means++ algorithm proposed by Arthur and Vassilvitskii (2007) claims to give significant computation savings in terms of speed and accuracy compared to the K-means clustering algorithms, it did not perform competitively in our experiments. The performance of different centroid selection algorithms can be checked over the NetFlix dataset, which is a subject of future research.

Chapter 7

Kernel Mapping Recommender (KMR) System Algorithms

7.1 Introduction

In this chapter, we propose a new class of kernel-mapping methods for solving the recommender system problem that gives state-of-the-art performance. The main idea is to find a multi-linear mapping between two vector spaces. The first vector space might, for example, have vectors encoding information about the items that we wish to rate, while the second vector space may contain a probability density describing how a particular user will rate an item. Learning an appropriate mapping can be expressed as a quadratic optimisation problem. As the problem involves a linear mapping the solution to the optimisation problem involves inner products in the two vector spaces. This allows us to use the kernel trick. Directly solving the optimisation problem using quadratic programming would be too slow for most recommendation datasets. Instead, we find an approximate solution iteratively, following an idea first developed by Joachims (2006). This allows us to train the recommender in linear time. The method described here is a specialisation of a general structured-learning framework developed by Szedmak and used in Szedmak et al. (2010) for handling incomplete data sources.

The approach we have adopted is easily adapted to different sources of information. We can, for example, use either rating information from other users or textual information about the items. Similarly, we are able to build either an item- or a user-based version of the algorithm. Because we have chosen to build a mapping to a space of functions approximating the probability density of the ratings, we have an intuitive interpretation of the recommendations produced by the algorithm. This gives us flexibility in how we make our final recommendation, which we can exploit to improve the final prediction for different datasets.

Our algorithm relies on a single coherent method (albeit with several variants) that has not been designed for a specific dataset. We have thus compared our approach with other general purpose recommenders. To the best of our knowledge the state-of-the-art algorithms are by Lawrence and Urtasun (2009) and Mackey et al. (2010). These achieve a considerable gap in performance advantage over the older algorithms. Our algorithm achieves similar performance to these approaches, although it is outperformed by Lawrence and Urtasun (2009) on a dataset with 1 000 000 ratings and by Mackey et al. (2010) on a dataset of 10 000 000 ratings. Our approach is however very different. The other two approaches are based on matrix factorisation, although Lawrence and Urtasun (2009) also uses kernel functions. There has been considerable work on developing matrix factorisation techniques which are at the heart of many of the most competitive algorithms for this problem. Part of the interest of our algorithm is that it takes a very different viewpoint from the matrix factorisation approaches, yet still has very competitive performance.

The main contributions of this chapter are highlighted in the following:

1. We present a new class of Kernel Mapping Recommender (KMR) system algorithms for solving the recommendation problem. We describe both the user- and item-based versions and show empirically that they outperform (or give comparable results to) the state-of-the-art algorithms.
2. We propose various ways of combining the user- and item-based versions. We show that both versions are complementary as they focus on different types of relationship in the dataset.
3. We show how more information can be added (linearly and non-linearly) and how it affects the recommendation quality.
4. We show how the cold-start, long tail, sparsity, and imbalanced datasets problems can be solved effectively.

The rest of the chapter has been organised as follows. In the next section we briefly outline related work. Section 7.3 outlines the proposed algorithm using an item-based approach. In Section 7.4 we describe extensions to the basic algorithm. Section 7.5 presents the tuning of parameters. This is followed in Section 7.6 by a presentation of results from our experimental evaluation. We conclude in Section 7.7.

7.2 Related Work

A large number of approaches have been proposed to remedy the accuracy, sparsity, and scalability problems associated with a recommender system, ranging from supervised classification techniques (Billsus and Pazzani, 1998) to unsupervised clustering

techniques (Park and Tuzhilin, 2008; Ghazanfar and Prügel-Bennett, 2011b), to dimensionality reduction techniques like Singular Value Decomposition (SVD) or matrix factorisation (Srebro et al., 2005; Rennie and Srebro, 2005; Bell et al., 2007; Wu, 2007; Takács et al., 2008; Salakhutdinov and Mnih, 2008; Lawrence and Urtasun, 2009; Koren et al., 2009; Takács et al., 2009). Classification techniques do not scale well with the dataset and furthermore they do not give accurate results. The clustering methods are effective at reducing the dimensionality of the datasets; however, they do not give accurate results. Singular value decomposition or matrix factorisation techniques give reasonably accurate results; however, they are expensive in terms of training and memory requirements, and often lead to the over-fitting. The proposed algorithms can be trained in linear time and provide accurate recommendations under all datasets.

In this work, we propose to use the rating data effectively to overcome the cold-start problems. Some other well-known approaches to overcome the cold-start problems using only the rating data have been offered by Ahn (2008) and Kim et al. (2011). For example, Kim et al. (2011) proposed to add an error correction term in the predictions computed by the collaborative filtering algorithm for the cold-start users or items. The error correction term, computed over all items rated by a user, is the difference between the predicted value of a rating and the actual value assigned by the user. This is very expensive heuristic, as it has to build the model for all user-item pairs that have been rated in the system making it unrealistic for large scale systems and dynamic environments. Our approach to overcome the cold-start (and related) problems is very simple and effective.

Hybrid recommender systems have been proposed elsewhere (Melville et al., 2002; Burke, 2002; Pazzani, 1999; Claypool et al., 1999; Burke, 1999; Ghazanfar and Prügel-Bennett, 2010a,e,b), which combine individual recommender systems to avoid certain limitations of individual recommender systems. In our approach we can add more information (about items) in the forms of additional kernels, which can be thought of combining collaborative filtering with content-based filtering. A related approach has been proposed in Basilico and Hofmann (2004), where the authors employed a unified approach for integrating the user-item ratings information with user/item attributes using kernels. They learned a prediction function using an on-line perceptron learning algorithm. They claimed that adding more kernels increases the performance, which is in contrast with our findings[1].

In Szedmak et al. (2010), the authors proposed a structured-learning algorithm for learning from incomplete datasets. The idea of the structure-learning has been used in Astikainen et al. (2008), where the authors employed the structured output prediction for enzyme prediction. We show how the structure-learning approach can be used to solve the recommender system problem effectively.

[1] It might be due to the reasons that they used very simple kernels, such as correlation, identity, etc.; however, we used polynomial kernels, which in turn are addition of correlation, identity, etc.

7.3 Item-based KMR

In this work, we only consider the ratings provided by actual users and assume all other ratings to be missing values, i.e.

$$r_{iu} = \begin{cases} r_{iu} & \text{if } r_{iu} \text{ is given,} \\ \varnothing & \text{otherwise.} \end{cases}$$

To perform the recommendation task we consider building the additive and multiplicative models for the *residual ratings*. The additive model is shown in equation 7.1

$$\hat{r}_{iu} = r_{iu} - \bar{r}_i - \bar{r}_u + \bar{r}, \tag{7.1}$$

where $\bar{r}_i$, $\bar{r}_u$ and $\bar{r}$ are respectively the mean rating for the item, of the user, and the overall mean. The multiplicative model can be expressed as follows:

$$\hat{r}_{iu} = \frac{r_{iu}\tilde{r}}{\tilde{r}_i\tilde{r}_u}, \tag{7.2}$$

where $\tilde{r}$, $\tilde{r}_i$ and $\tilde{r}_u$ are the geometric means for all the ratings, the ratings for item i, and the rating of user u respectively. We found the additive model to be (marginally) better than the multiplicative one, and hence this work is based on the additive model.

We use a technique developed by Szedmak and co-workers for learning structured data (Szedmak et al., 2010). In the following, we outline how this approach is adapted for solving the collaborative filtering problem. We assume that we have some information about the items which we denote by $\boldsymbol{q}_i$. This may, for example, be the set of ratings r_{iu} for $u \in \mathcal{D}_i$, or it could be text describing the item i. We map the information to some vector $\boldsymbol{\phi}(\boldsymbol{q}_i)$ in some extended feature (Hilbert) space. Similarly, we map the rating residues, $\hat{r}_{iu}$, to 'vectors' in some other Hilbert space. In this work, we consider these objects to lie in the function space $L_2(\mathbb{R})$. In particular we represent each residual $\hat{r}_{iu}$, by a normal distribution with mean $\hat{r}_{iu}$ and variance σ^2. That is,

$$\boldsymbol{\psi}(\hat{r}_{iu}) = \mathcal{N}(x|\hat{r}_{iu}, \sigma). \tag{7.3}$$

The motivation of this choice is to model possible errors in the rating either due to the discretisation of the rating scale or the variability in assigning a rating (e.g. due to the mood of the user on the day they made the rating).

The method developed by Szedmak is to seek a linear mapping between these two spaces which can be used for making predictions. More specifically, in our application, we look for a linear mapping $\mathbf{W}_u$ from the space of $\boldsymbol{\phi}$ vectors to the space of $\boldsymbol{\psi}$ vectors, such that the inner product satisfies the inequality

$$\langle \boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle \geq 1 - \zeta_i,$$

where $\zeta_i \geq 0$ is a slack variable. We then seek to minimise the Frobinius norm of $\mathbf{W}_u$ and the slack variables ζ_i. We can describe the optimisation problem succinctly as

$$\begin{array}{ll} \text{min} & \frac{1}{2}\sum_{u\in\mathcal{U}}\|\mathbf{W}_u\|^2 + C\sum_{i\in\mathcal{I}}\zeta_i \\ \text{with respect to} & \mathbf{W}_u,\ u\in\mathcal{U},\ \zeta_i,\ i\in\mathcal{I} \\ \text{subject to} & \langle\boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle \geq 1-\zeta_i \\ & \zeta_i \geq 0,\ i\in\mathcal{I},\ u\in\mathcal{D}_i. \end{array} \tag{7.4}$$

Note that minimisation will be achieved when the vectors $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ are as uniformly aligned as possible with the vector $\boldsymbol{\psi}(\hat{r}_{iu})$. Having learned the mappings $\mathbf{W}_u$ we can then make predictions for a new item j using $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$. This outputs a function which informally we can think of as an estimate for the probability density of the residue $\hat{r}_{ju}$. However, $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$ does not need to be, and typically is not, positive everywhere or normalised. Thus, it is not itself a probability density. We discuss later different methods for interpreting $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_j)$.

To solve this constrained optimisation problem, we define the Lagrangian

$$\mathcal{L} = \frac{1}{2}\sum_{u\in\mathcal{U}}\|\mathbf{W}_u\|^2 + C\sum_{i\in\mathcal{I}}\zeta_i - \sum_{(i,u)\in\mathcal{D}}\alpha_{iu}\left(\langle\boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle - 1 + \zeta_i\right) - \sum_{i\in\mathcal{I}}\lambda_i\zeta_i,$$

where $\alpha_{iu} \geq 0$ are Lagrange multipliers introduced to ensure that $\langle\boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle \geq 1-\zeta_i$ and $\lambda_i \geq 0$ are Lagrange multipliers introduced to ensure that $\zeta_i \geq 0$. The optimum mapping is found by solving

$$\min_{\{\mathbf{W}_u\},\{\zeta_i\}}\ \max_{\{\alpha_{iu}\},\{\lambda_i\}}\ \mathcal{L},$$

subject to the constraints that $\alpha_{iu} \geq 0$ for all $(i,u) \in \mathcal{D}$ and $\lambda_i \geq 0$ for all $i \in \mathcal{I}$. For a general linear mapping, $\mathbf{W}_u$, we have that

$$\frac{\partial}{\partial\mathbf{W}_u}\langle\boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)\rangle = \boldsymbol{\psi}(\hat{r}_{iu}) \otimes \boldsymbol{\phi}(\boldsymbol{q}_i),$$

where $\otimes$ is the tensor-product of the two vectors. This is clearly the case when the Hilbert spaces are finite dimensions so that the mapping $\mathbf{W}_u$ can be represented by a matrix, but this can be extended for linear mappings between more general Hilbert spaces. Using this result we find

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}_u} = \mathbf{W}_u - \sum_{i\in\mathcal{D}_u}\alpha_{iu}\boldsymbol{\psi}(\hat{r}_{iu}) \otimes \boldsymbol{\phi}(\boldsymbol{q}_i).$$

The Lagrangian is minimised with respect to $\mathbf{W}_u$ when $\mathbf{W}_u = \sum_{i\in\mathcal{D}_u}\alpha_{iu}\boldsymbol{\psi}(\hat{r}_{iu}) \otimes \boldsymbol{\phi}(\boldsymbol{q}_i)$. Taking derivatives with respect to ζ_i we find

$$\frac{\partial\mathcal{L}}{\partial\zeta_i} = C - \sum_{u\in\mathcal{D}_i}\alpha_{iu} - \lambda_i.$$

Setting these derivatives to 0 we find that the Lagrangian is minimised with respect to ζ_i when

$$\sum_{u \in \mathcal{D}_i} \alpha_{iu} = C - \lambda_i \leq C$$

where the inequality arises because $\lambda_i \geq 0$.

After substituting back the expressions containing only the Lagrange multipliers into the Lagrangian we obtain the dual problem of (7.4) which is a maximisation problem with respect to the variables α_{iu}

$$f(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{u \in \mathcal{U}} \sum_{i,i' \in \mathcal{D}_u} \alpha_{iu} \alpha_{i'u} \langle \boldsymbol{\psi}(\hat{r}_{iu}), \boldsymbol{\psi}(\hat{r}_{i'u}) \rangle \langle \boldsymbol{\phi}(\boldsymbol{q}_i), \boldsymbol{\phi}(\boldsymbol{q}_{i'}) \rangle + \sum_{(i,u) \in \mathcal{D}} \alpha_{iu}$$

subject to the constraint that $\boldsymbol{\alpha} \in Z(\boldsymbol{\alpha})$ where

$$Z(\boldsymbol{\alpha}) = \left\{ \boldsymbol{\alpha} \middle| \forall u \in \mathcal{U}, \sum_{u \in \mathcal{D}_i} \alpha_{iu} \leq C \wedge \forall (i,u) \in \mathcal{D}, \alpha_{iu} \geq 0 \right\}.$$

We are now in the position where we can apply the usual kernel trick. The kernel functions can be defined by

$$K_{\hat{r}}(\hat{r}_{iu}, \hat{r}_{i'u}) = \langle \boldsymbol{\psi}(\hat{r}_{iu}), \boldsymbol{\psi}(\hat{r}_{i'u}) \rangle$$
$$K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_{i'}) = \langle \boldsymbol{\phi}(\boldsymbol{q}_i), \boldsymbol{\phi}(\boldsymbol{q}_{i'}) \rangle,$$

and then we can write $f(\boldsymbol{\alpha})$ as

$$f(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{u \in \mathcal{U}} \sum_{i,i' \in \mathcal{D}_u} \alpha_{iu} \alpha_{i'u} K_{\hat{r}}(\hat{r}_{iu}, \hat{r}_{i'u}) K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_{i'}) + \sum_{(i,u) \in \mathcal{D}} \alpha_{iu}$$

where we are free to choose any pair of positive definite kernel functions. With our choice of mapping the rating residual, $\hat{r}$, to $\psi(\hat{r}) = \mathcal{N}(x|\hat{r}, \sigma)$, we note that

$$K_{\hat{r}}(\hat{r}, \hat{r}') = \langle \boldsymbol{\psi}(\hat{r}), \boldsymbol{\psi}(\hat{r}') \rangle = \mathcal{N}(\hat{r} - \hat{r}' | \sqrt{2}\sigma),$$

which is inexpensive to compute. We could build more complex kernels for $K_{\hat{r}}(\hat{r}, \hat{r}')$, by mapping $\psi(\hat{r})$ into another extended feature space, although we would then lose the interpretation of $\mathbf{W}_u \boldsymbol{\phi}(\boldsymbol{q}_i)$ as an approximation to the density function for $\hat{r}_{iu}$.

7.3.1 Learning the Lagrange multipliers

For large-scale recommender systems, solving this quadratic programming problem using a general quadratic programming solver would be impractical due to the large number of data points. However, we can find an approximate solution iteratively using the

conditional gradient method. To understand this method it is helpful to write $f(\boldsymbol{\alpha})$ in matrix form

$$f(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\mathbf{M}\boldsymbol{\alpha} + \boldsymbol{b}^\mathsf{T}\boldsymbol{\alpha}$$

with $\boldsymbol{\alpha} \in Z(\boldsymbol{\alpha})$. We obtain a series of approximations $\boldsymbol{\alpha}_t$ for the optimal parameters starting from some initial guess $\boldsymbol{\alpha}_0 \in Z(\boldsymbol{\alpha})$. At each step we use a linear approximation for $f(\boldsymbol{\alpha})$

$$f(\boldsymbol{\alpha}) \approx \hat{f}_{\boldsymbol{\alpha}_t}(\boldsymbol{\alpha}) = f(\boldsymbol{\alpha}_t) + (\boldsymbol{\alpha} - \boldsymbol{\alpha}_t)\boldsymbol{\nabla} f(\boldsymbol{\alpha}_t).$$

We compute the next approximation using two stages. We first solve the linear programming problem

$$\begin{aligned}\boldsymbol{\alpha}^* &= \operatorname*{argmax}_{\boldsymbol{\alpha} \in Z(\boldsymbol{\alpha})} \hat{f}_{\boldsymbol{\alpha}_t}(\boldsymbol{\alpha}) \\ &= \operatorname*{argmax}_{\boldsymbol{\alpha} \in Z(\boldsymbol{\alpha})} -\boldsymbol{\alpha}^\mathsf{T}(\mathbf{M}\boldsymbol{\alpha}_t - \boldsymbol{b}) + \text{const.}\end{aligned}$$

We then find the new approximation $\boldsymbol{\alpha}_{t+1}$ to be

$$\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \tau(\boldsymbol{\alpha} - \boldsymbol{\alpha}_t)$$

where we choose τ to be

$$\tau = \operatorname*{argmax}_{\tau} f(\boldsymbol{\alpha}_{t+1}) = \frac{(\boldsymbol{b} - \mathbf{M}\boldsymbol{\alpha}_t)^\mathsf{T}(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}_t)}{(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}_t)^\mathsf{T}\mathbf{M}(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}_t)}.$$

This guarantees that no step increases the objective function.

We note that in the linear programming problem we have an objective function of the form

$$\sum_{i \in \mathcal{I}} \sum_{u \in \mathcal{D}_i} \alpha_{iu} \frac{\partial f(\boldsymbol{\alpha}_t)}{\partial \alpha_{iu}} + \text{const},$$

which decouples for every set of Lagrange multipliers $\mathcal{A}_i = \{\alpha_{ui} | u \in \mathcal{D}_i\}$. The linear constraints $Z(\boldsymbol{\alpha})$ also decouple into a set of constraints for each set of Lagrange multipliers $\mathcal{A}_i$. Thus we can perform the linear programming independently for each set of variables $\mathcal{A}_i$. Furthermore, due to the simplicity of the constraint, it turns out that the linear programming problem can be solved in linear time (as opposed to cubic time for a general linear programming problem). The training is stopped after a fixed number of iterations which is a parameter of the training algorithm. The total complexity of this step for all users is $O(|D|)$. This leads to an algorithm with linear complexity in the number of available ranks. Note that the non-zero elements of the matrix $\mathbf{M}$ is equal to $|\mathcal{D}_u|$ which tends to be constant, i.e. it does not increase with the number of users.

Table 7.1: Example: a subset of the user-item rating matrix in a movie recommender system. We have four users (rows) and three movies (columns). The case, where a user has not rated a particular movie is shown by the ⊘ symbol. The rating scale, consisting of integer values between 1 and 5, captures the extreme likes (5) and extreme dislikes (1) behaviour of a user. The rating we want to predict is shown by "?" symbol.

	i_1	i_2	i_3
u_1	5	⊘	1
u_2	5	5	1
u_3	5	4	2
u_4	1	3	?

7.3.2 Predicting unseen ratings

To make a prediction for the rating r_{iu} where $(i, u) \notin \mathcal{D}$, we estimate the residue $\hat{r}_{iu} = r_{iu} - \bar{r}_i - \bar{r}_u + \bar{r}$ using the function

$$\begin{aligned} p_{iu}(\hat{r}) &= \langle \psi(\hat{r}), \mathbf{W}_u \phi(\boldsymbol{q}_i) \rangle \\ &= \sum_{i' \in \mathcal{D}_u} \alpha_{i'u} K_{\hat{r}}(\hat{r}, \hat{r}_{i'u}) K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_{i'}), \end{aligned}$$

where $\psi(\hat{r}) = \mathcal{N}(\hat{r}, \sigma)$. We have a choice in how to obtain a single prediction from this function. Our *standard max predictor* will be to find the maximum argument of $p_{iu}(\hat{r})$

$$\check{r}_{iu} = \underset{\hat{r}}{\operatorname{argmax}}\, p_{iu}(\hat{r}).$$

To make it clear how the algorithm makes predictions, next, we provide a small scale example.

7.3.3 A small scale example

Suppose a recommender system has four users (i.e. $\mathcal{U} = \{u_1, u_2, u_3, u_4\}$) and three items (i.e. $\mathcal{I} = \{i_1, i_2, i_3\}$). The information about each item is a column vector of the user-item rating matrix, shown in Table 7.1. The users', items', and overall averages are:

$$\begin{aligned} \bar{r}_{u_1} &= 3.000, & \bar{r}_{u_2} &= 3.666, & \bar{r}_{u_3} &= 3.666, & \bar{r}_{u_4} &= 2.000, \\ \bar{r}_{i_1} &= 4.000, & \bar{r}_{i_2} &= 4.000, & \bar{r}_{i_3} &= 1.333, & \bar{r} &= 3.200, \end{aligned}$$

After applying the additive model (equation 7.1), the user-item rating matrix can be represented in the residual form as shown in Table 7.2. The input feature kernel, $K_{\boldsymbol{q}}$, using the polynomial kernel (refer to Section 7.5.2) is shown in equation 7.5.

Table 7.2: Example: a subset of the user-item rating matrix in a movie recommender system after normalisation.

	i_1	i_2	i_3
u_1	1.200	$\oslash$	-0.1333
u_2	0.533	0.533	-0.800
u_3	0.533	-0.466	0.200
u_4	-1.800	0.200	?

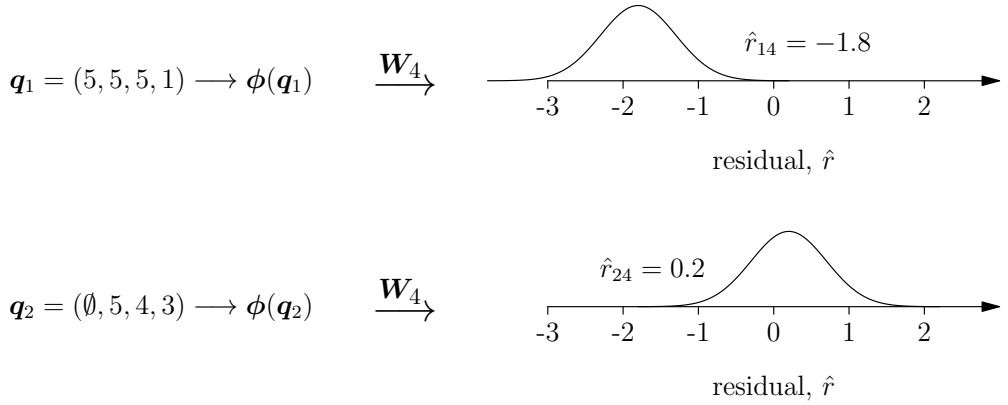


Figure 7.1: Schematic showing the aim of the algorithm. Information, $\boldsymbol{q}_i$ (in this case a rating vector) about an item i, is first mapped to a vector in an extended feature space $\boldsymbol{\phi}(\boldsymbol{q}_i)$. We then try to find the best linear mapping, $\mathbf{W}_4$, for user u_4, to the vector, $\psi(\hat{r}_{iu_4})$, describing the residual.

$$K_{\boldsymbol{q}} = \begin{bmatrix} 1.000 & 0.067 & 0.003 \\ 0.067 & 1.000 & 0.300 \\ 0.003 & 0.300 & 1.000 \end{bmatrix} \tag{7.5}$$

We can compute the residual kernel, $K_{\hat{r}}$, based on the inner products between Gaussian densities functions with expected values $\hat{r}$ and $\hat{r}'$, and sharing the common standard deviation σ.

$$K_{\hat{r}}(\hat{r}, \hat{r}') = \langle \boldsymbol{\psi}(\hat{r}), \boldsymbol{\psi}(\hat{r}') \rangle = \frac{1}{2\sigma\sqrt{\pi}} e^{-(\hat{r}-\hat{r}')^2/4\sigma^2}$$

Assume that $\sigma = 0.5$ then we have

$$K_{\text{residual}} = \begin{bmatrix} K_{\hat{r},u_1} & & & \\ & K_{\hat{r},u_2} & & \\ & & K_{\hat{r},u_3} & \\ & & & K_{\hat{r},u_4} \end{bmatrix},$$

where

$$K_{\hat{r},u_2} = \begin{bmatrix} 0.564 & 0.564 & 0.149 \\ 0.564 & 0.564 & 0.149 \\ 2.140 & 2.140 & 0.564 \end{bmatrix} \quad K_{\hat{r},u_1} = \begin{bmatrix} 0.564 & 0.149 \\ 2.140 & 0.564 \end{bmatrix}$$
$$K_{\hat{r},u_3} = \begin{bmatrix} 0.564 & 0.208 & 0.404 \\ 1.534 & 0.564 & 1.099 \\ 0.788 & 0.290 & 0.564 \end{bmatrix} \quad K_{\hat{r},u_4} = \begin{bmatrix} 0.564 & 0.149 \\ 2.140 & 0.564 \end{bmatrix}.$$

Table 7.3: The optimal values of design variable, α, for each user-item pair.

	i_1	i_2	i_3
u_1	1.000	$\oslash$	1.000
u_2	0.993	0.993	0.999
u_3	0.993	0.768	0.769
u_4	1.000	1.000	?

The optimal values for the design variable, α, are learned using the conditional gradient method, and are shown in Table 7.3. After learning α parameters, the mapping $\mathbf{W}_u$, can be defined for each user (recall $\mathbf{W}_u = \sum_{i \in \mathcal{D}_i} \alpha_{iu} \boldsymbol{\psi}(\hat{r}_{iu}) \otimes \boldsymbol{\phi}(\boldsymbol{q}_i)$). To make a prediction for the rating r_{iu}, where $(i,u) \notin \mathcal{D}$

$$\begin{aligned} \mathbf{W}_u \phi(\boldsymbol{q}_i) &= \sum_{i' \in \mathcal{D}_u} \alpha_{i'u} \boldsymbol{\psi}(\hat{r}_{iu}) < \boldsymbol{\phi}(\boldsymbol{q}_{i'}), \boldsymbol{\phi}(\boldsymbol{q}_i) >, \\ &= \sum_{i' \in \mathcal{D}_u} \alpha_{i'u} \boldsymbol{\psi}(\hat{r}_{iu}) K_{\boldsymbol{q}}(\boldsymbol{q}_{i'}, \boldsymbol{q}_i). \end{aligned}$$

In our case, we have $u = u_4$ and $i = i_3$, so

$$\begin{aligned} \mathbf{W}_{u_4} \phi(\boldsymbol{q}_{i_3}) &= \alpha_{i_1 u_4} \boldsymbol{\psi}(\hat{r}_{i_1 u_4}) K_{\boldsymbol{q}}(\boldsymbol{q}_{i_1}, \boldsymbol{q}_{i_3}) + \alpha_{i_2 u_4} \boldsymbol{\psi}(\hat{r}_{i_2 u_4}) K_{\boldsymbol{q}}(\boldsymbol{q}_{i_2}, \boldsymbol{q}_{i_3}) \\ &= 1.000 \boldsymbol{\psi}(\hat{r}_{i_1 u_4}) 0.003 + 1.000 \boldsymbol{\psi}(\hat{r}_{i_2 u_4}) 0.300, \\ &= 0.003 \boldsymbol{\psi}(\hat{r}_{i_1 u_4}) + 0.300 \boldsymbol{\psi}(\hat{r}_{i_2 u_4}), \\ &= 0.003 \mathcal{N}(\hat{r}_{i_1 u_4}, \sigma) + 0.300 \mathcal{N}(\hat{r}_{i_2 u_4}, \sigma). \end{aligned}$$

It is an unnormalized probability density function of mixture of two Gaussians. The optimal rating then can be derived by

$$\begin{aligned} p_{i_3 u_4}(\hat{r}) &= \langle \psi(\hat{r}), \mathbf{W}_{u_4} \phi(\boldsymbol{q}_{i_3}) \rangle \\ &= \arg\max_{\hat{r}} \langle \psi(\hat{r}), 0.003 \boldsymbol{\psi}(\hat{r}_{i_1 u_4}) + 0.300 \boldsymbol{\psi}(\hat{r}_{i_2 u_4}) \rangle, \\ &= \arg\max_{\hat{r}} \langle 0.003 K_{\hat{r}}(\hat{r}, \hat{r}_{i_1 u_4}) + 0.300 K_{\hat{r}}(\hat{r}, \hat{r}_{i_2 u_4}) \rangle, \\ &= \arg\max_{\hat{r}} \langle 0.003 \mathcal{N}(\hat{r} | \hat{r}_{i_1 u_4}, \sqrt{2}\sigma) + 0.300 \mathcal{N}(\hat{r} | \hat{r}_{i_2 u_4}, \sqrt{2}\sigma) \rangle \end{aligned}$$

Taking the optimum solution (refer to Figure 7.2), $r_{i_3u_4}$, the prediction for the residual is 0.2. Hence, user u_4 would rate item i_3 with rating of $\hat{r}_{i_3u_4} + \hat{r}_i + \hat{r}_u - \hat{r} = 0.2 + 2.0 + 1.333 - 3.2 = 0.33$.

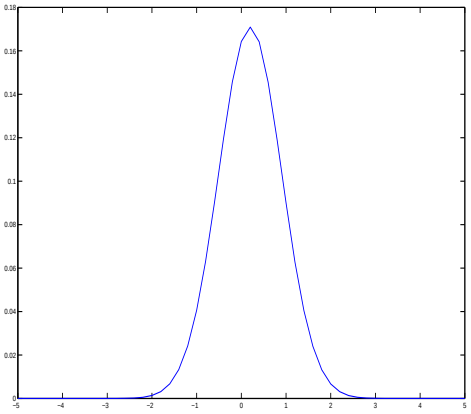


Figure 7.2: Plotting the probability density function of mixture of two Gaussians with $\hat{r}$={−5:0.2:5}. The optimal solution is found to be 0.2.

7.4 Extensions to the Basic Algorithm

In this section we describe extensions to the basic algorithm which are relevant to practical recommender systems.

7.4.1 User-based KMR

Depending on the dataset characteristics (e.g. number of items rated by the active user, number of users which have rated the target item, etc.) different models can be trained along the rows or columns of the data matrix. A related algorithm is proposed, which solves the problem from the user point of view, hence it is named as user-based KMR (KMR_{ub}). To perform a user-based recommendation, we use information $\boldsymbol{q}_u$ about users u and try to find a linear mapping $\mathcal{W}_i$ to align some extended feature vectors $\boldsymbol{\phi}(\boldsymbol{q}_u)$ to the residue vector $\boldsymbol{\psi}(\hat{r}_{iu})$. The derivation is identical to that for the item-based recommender when we interchange the subscripts i and u.

7.4.2 Combining the user- and item-based KMR

The user- and item-based versions provide complementary roles in generating predictions as they focus on different types of relationships in a dataset. Let $\check{r}_{iu}^{ub}$ and $\check{r}_{iu}^{ib}$ be the predictions made by the user- and item-based versions respectively. We have considered three different ways of combining the user- and item-based predictions.

- *Using the simple linear combination:* In this approach, the user- and item-based versions are linearly combined, where the parameter θ_{linear} is learned from a validation set.

$$\check{r}_{iu} = \theta_{linear}\check{r}^{ub}_{iu} + (1 - \theta_{linear})\check{r}^{ib}_{iu} \tag{7.6}$$

  We denote the resulting hybrid recommender system by KMR^{linear}_{hybrid}.

- *Switching on number of ratings:* Here, we take into account the information about user and item profiles. The rationale behind this approach is the intuition that if we have a large number of ratings for an item compared to the number of ratings made by the active user, then the user-based version is likely to give better results than the item-based version and vice-versa. Rather than using the raw number of ratings, we normalise by the number of ratings given by the power user, u_p (i.e. the user that has rated the most number of items) and by the power item i_p (i.e. the item with the maximum number of ratings). That is, we used

$$\check{r}_{iu} = \begin{cases} \check{r}^{ub}_{iu}: & \text{if } \frac{|\mathcal{U}_i|}{|\mathcal{U}_{i_p}|} - \frac{|\mathcal{I}_u|}{|\mathcal{I}_{u_p}|} > \theta_{cnt} \\ \check{r}^{ib}_{iu}: & \text{otherwise.} \end{cases} \tag{7.7}$$

  We denote the resulting hybrid recommender system by KMR^{cnt}_{hybrid}.

- *Switching on uncertainty in prediction:* Here we use a different strategy for switching between the user- and item-based predictors. We try to estimate the uncertainty in the prediction by examining the "variance" in $\boldsymbol{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ and $\boldsymbol{W}_i\boldsymbol{\phi}(\boldsymbol{q}_u)$. Since they are not real probability distributions, we must first exclude the regions where the functions go negative and normalise the output so that we can treat them as densities and compute their variance. We denote the variance by var_{ub} and var_{ib} for the user- and item-based versions, respectively. We then switch the recommendation we use according to

$$\check{r}_{iu} = \begin{cases} \check{r}^{ub}_{iu}: & \text{if } \text{var}_{ub} - \text{var}_{ib} > \theta_{var} \\ \check{r}^{ib}_{iu}: & \text{otherwise.} \end{cases} \tag{7.8}$$

  We denote the resulting hybrid recommender system by KMR^{var}_{hybrid}.

7.4.3 Combining different kernels

In many applications there are multiple sources of information that can be used to make a recommendation. We can easily accommodate different sources of information by combining kernels. To illustrate this we will test our algorithm on datasets consisting of film ratings where we have three types of information available (see chapter 3 for details)

- The ratings of other users from which we can construct a kernel K_{rat}

- "Demographic" information obtained from genre about the films from which we can construct a kernel K_{demo}
- "Feature" information obtained form a textual description of the films from which we construct a kernel K_{feat}.

These kernels can be combined *linearly*

$$K = \beta_{rat}K_{rat} + \beta_{demo}K_{demo} + \beta_{feat}K_{feat}, \tag{7.9}$$

where the parameters β_{rat}, β_{demo} and $\beta_{feat} = 1 - \beta_{rat} - \beta_{demo}$ can be tunn by measuring the generalisation performance on a validation set. This way of combining kernels can be viewed as a concatenation of the feature vectors

$$\begin{aligned}\boldsymbol{\phi} &= (\sqrt{\beta_{rat}}\boldsymbol{\phi}_{rat},\ \sqrt{\beta_{demo}}\boldsymbol{\phi}_{demo},\ \sqrt{\beta_{feat}}\boldsymbol{\phi}_{feat}) \\ &= \sqrt{\beta_{rat}}\boldsymbol{\phi}_{rat} \oplus \sqrt{\beta_{demo}}\boldsymbol{\phi}_{demo} \oplus \sqrt{\beta_{feat}}\boldsymbol{\phi}_{feat},\end{aligned}$$

where $\oplus$ represents the direct sum. Alternatively we can combine the kernels *non-linearly*

$$K = K_{rat} \cdot K_{demo} \cdot K_{feat}, \tag{7.10}$$

where the $\cdot$ denotes the point-wise product of the kernel matrices. This corresponds to taking a tensor product of the feature vectors

$$\boldsymbol{\phi} = \boldsymbol{\phi}_{rat} \otimes \boldsymbol{\phi}_{demo} \otimes \boldsymbol{\phi}_{feat}. \tag{7.11}$$

7.4.4 Cold-start, long tail, and imbalanced datasets

The standard max predictor works well when we have a sufficient number of ratings for the user and the item. However as we will see it gives poor predictions in scenarios where we have a small amount of training data. Since we argued earlier $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as an approximation for the probability density of $\hat{r}_{iu}$. It will not generally be positive everywhere, but by removing the negative part of the function we can treat the remaining function as a probability density. In this case we can consider the *mean*, *mode*, or *median* as approximations for the most likely value of $\hat{r}_{iu}$. Under conditions where we lack sufficient data we find that using a combination of the mean, mode and median together with the standard (max) prediction gives a considerable improvement in accuracy. In particular, we consider a predictor

$$\hat{r}_{M^4} = w_{max}\hat{r}_{max} + w_{mean}\hat{r}_{mean} + w_{mode}\hat{r}_{mode} + w_{median}\hat{r}_{median}$$

$\hat{r}_j$ for $j \in \{max, mean, mode, median\}$ are the standard predictors and the predictors using the mean, mode and median, while w_j are a set of weights that are learned from a validation set. We consider the weights to be constrained so that $w_j \geq 0$ and they sum to 1. In the results shown later we denote those that use this predictor by the superscript M^4.

7.4.5 Two-way clustering

In an attempt to increase the performance, we tried clustering both the users and items. The optimal clustering is intractable; however, fast approximate clustering algorithms are abundant. We propose a simple sorting algorithm for clustering the dataset. We used an iterative process of sorting first the columns and then the rows of the user-item rating matrix. This is repeated in an attempt to derive the matrix into a block structure[2]. While normalising the dataset (see equation 7.1), the user or item averages are taken from the cluster in which a user or item resides. This ensures that the average values are taken from the most similar group of users or items. The clustered dataset is shown in Figure 7.3. We observe that the ratings are in the block structure.

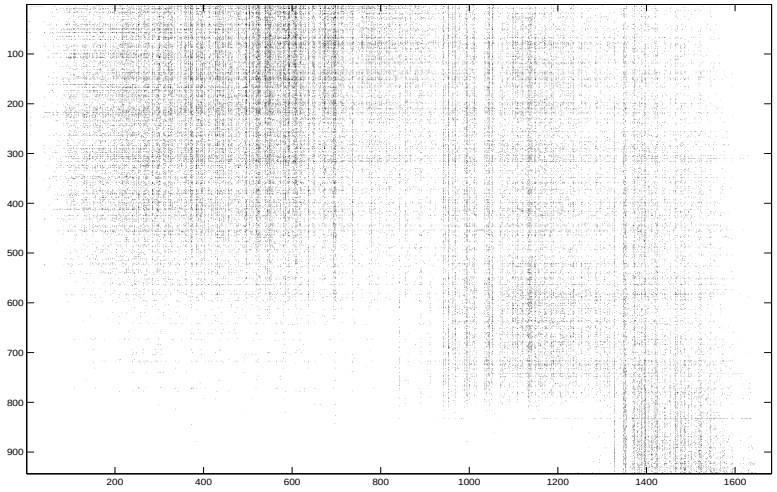


Figure 7.3: Clustering the rows and columns for the SML dataset for 2000 number of iterations. Note that the user-item rating matrix has been derived into a block structure.

7.4.6 Standard deviation in the output Gaussian kernel

The parameter, σ, is used in mapping $\psi(\hat{r}) = \mathcal{N}(x|\hat{r}, \sigma)$. It models the uncertainty or the fluctuation in a user's rating. The simplest solution is to use a single value of σ for

[2]The process is terminated when either there is no change in the sorting process or maximum number of iterations are repeated (10 000).

the dataset. However, as different users rate items differently (for example some rate items systematically, some rate arbitrary, depending on the context), hence, changing the value of σ per user is conceivable. Nevertheless, having more parameters in the system increases the complexity of the system and the system might suffer from over-fitting problem. Changing the value of σ per group is less ideal compared to changing the value of σ per user, yet it offers a good trade-off between the complexity of the system and modeling the uncertainty in the user's rating.

7.5 Learning the Optimal System Parameters

There are number of parameters that need to be learned. In this section, we discuss the training of important parameters.

7.5.1 Number of iterations

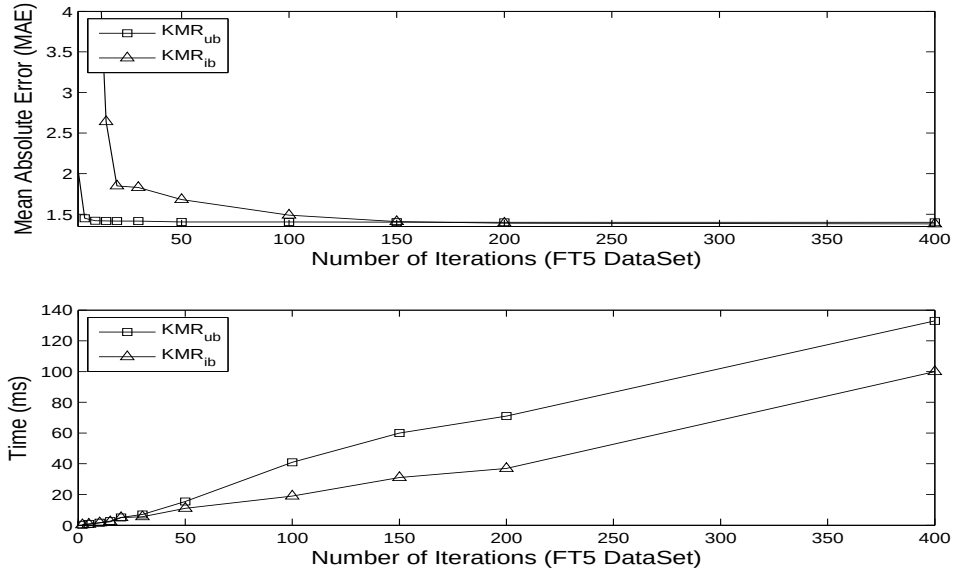


Figure 7.4: The number of iterations and time required to converge the KMR algorithms (FT5 dataset).

The algorithm we develop uses an iterative technique to learn the Lagrange multipliers. As we increase the number of iterations the mean absolute error improves. The speed of convergence will depend on the dataset and the type of information we are using (e.g. the user- or item-based). Figure 7.4 and 7.5, show the mean absolute error and the time taken to learn the Lagrange multipliers versus the number of iterations for the FT5 and SML datasets respectively.

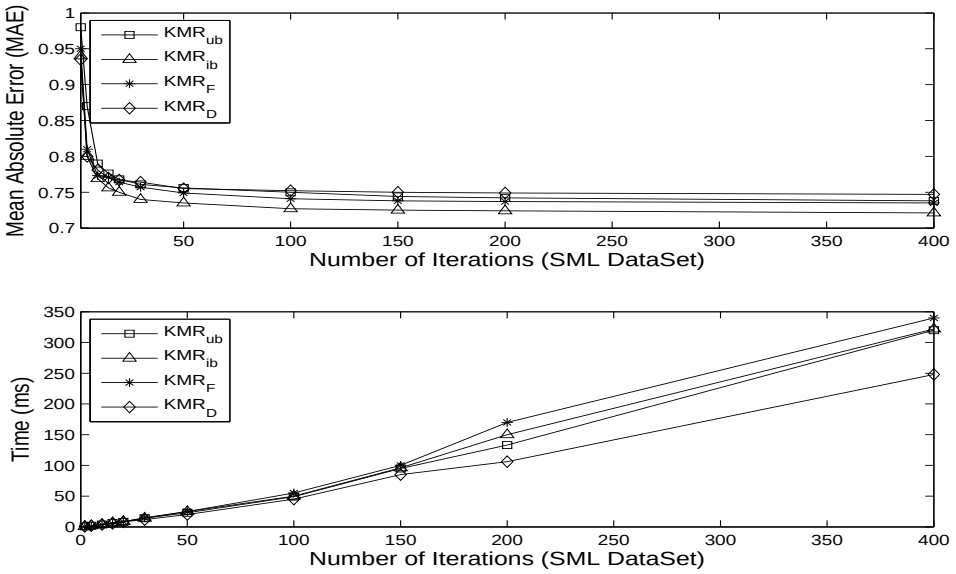


Figure 7.5: The number of iterations and time required to converge the KMR algorithms (SML dataset).

We note that for the FT5 dataset, the performance of the item-based version suffers badly when the number of iterations are very small. However, the performance of the user-based version is quite good even after a few iterations. Hence, if one has a constraint on the time required to build the model, then it is better to switch to the user-based version rather than the item-based version for the dataset. In contrast in the SML dataset the convergence of all the methods was relatively quick. The convergence clearly depends on the number of users/items and the user/item profile length (e.g. rating profile, feature profile length etc.). It is not obvious *a priori* how many iterations are needed to get good rating predictions. For the consequent experiments, we chose the number of iterations to be 400 for the SML dataset, 300 for FT5, 400 for ML, and 600 for ML10 and NF.

7.5.2 The optimal kernel parameters

We trained linear, polynomial, and poly-Gaussian kernels and chose the one giving the most accurate results. The polynomial kernel is of the form

$$K(\boldsymbol{x}, \boldsymbol{y}) = \Big(\langle \boldsymbol{x}, \boldsymbol{y} \rangle + R\Big)^d.$$

For the rating based version, the best polynomial kernel parameters (d, R) are found to be, for the user- and item-based versions respectively: $(3, 0.5)$ and $(4, 0.5)$ for the SML dataset; $(6, 0.4)$ and $(6, 0.4)$ for the FT5 dataset; and $(6, 0.1)$ and $(9, 0.1)$ for the ML dataset. For the feature based version, the best polynomial kernel parameters were found to be $(5, 0.5)$ for the SML dataset and; $(5, 0.1)$ for the FT5 dataset.

We did not tune the parameters for ML10 and NF dataset, as it was computationally expensive. We fixed them to $(14, 0.5)$ for user and item-based versions for both datasets; and $(12, 0.5)$ for the feature-based version for the ML10 dataset.

For the demographic based version, we found the best kernel was the poly-Gaussian kernel (which is a simple extension of the Gaussian one) given by

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^q}{\tau}\right), \tag{7.12}$$

where the best parameter (q, τ) were found to be $(0.1, 0.1)$ for the SML dataset; and $(0.2, 0.1)$ for the FT5 dataset. Again we did not tune parameters for ML10 dataset and they were fixed to $(0.5, 0.1)$.

7.5.3 Parameters β_{rat}, β_{feat}, and β_{demo}

Parameters β_{rat}, β_{feat}, and $\beta_{demo} = \beta_{rat} - \beta_{feat}$ determine the relative weights of rating, feature, and demographic kernels in the final prediction. The 66 parameter sets were generated by producing all possible combination of parameters values, ranging from 0 to 1.0 with differences of 0.1. We assume that $\beta_{rat} + \beta_{feat} + \beta_{demo} = 1$ without the loss of generalization. The parameters sets $\beta_{rat} = 1$ and $\beta_{feat} = 0$ gave the lowest MAE for all the datasets.

7.5.4 Parameter θ_{linear}

Parameters θ_{linear} and $(1 - \theta_{linear})$ determine the relative weights of the user- and item-based KMR in the final prediction respectively. We changed the value of θ_{linear} from 0 to 1 with a difference of 0.1 and the resulting MAE has been shown in Figure 7.6 (for Case 1, as discussed in Section 7.6.4). Figure 7.6 shows that for the SML dataset, the MAE is minimum at $\theta_{linear} = 0.3$, after which it starts increasing again; whereas, for the FT5 dataset, the MAE keeps on decreasing, reaches its minimum at $\theta_{linear} = 0.9$, an then increases again. For this reason, we choose the optimal value of θ_{linear} to be 0.3 and 0.9 for SML and the FT5 dataset respectively. Similarly, the value of θ_{linear} was trained for other datasets.

7.5.5 Threshold θ_{cnt}

In the hybrid variant, KMR^{cnt}_{hybrid} the threshold θ_{cnt} determines the switching point between using the item-based and user-based algorithms depending on the number of ratings of the item and user. We determine the best value of θ_{cnt} by varying it between 0 and 1 in steps of 0.04. Figure 7.7 shows the parameter θ_{cnt} learned (for Case 1,

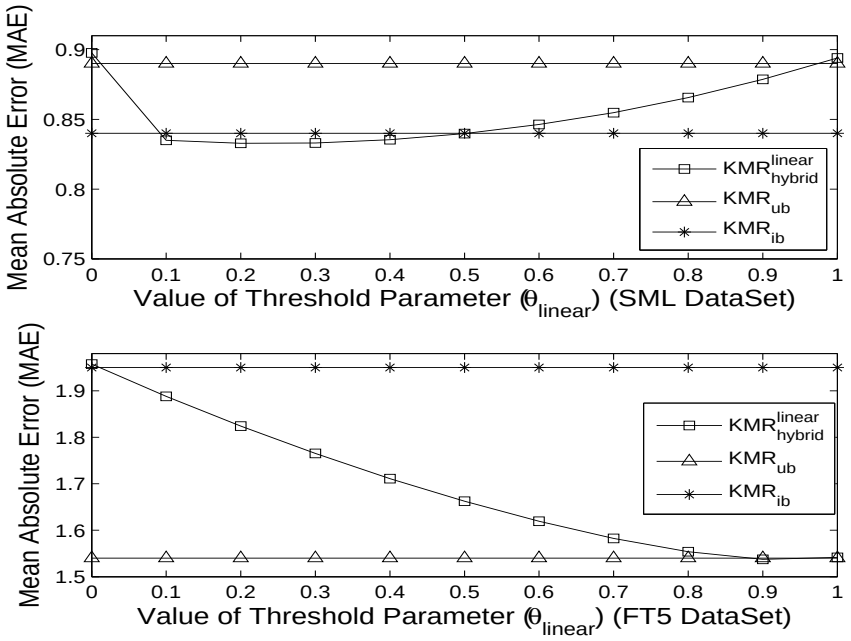


Figure 7.6: Learning the optimal value of threshold parameter θ_{linear}, over the validation set.

as discussed in Section 7.6.4) over the validation set. We observe that for the SML dataset, the MAE keeps on decreasing with the increase in the value of θ_{cnt}, reaches at its minimum between $\theta_{cnt} \in [0.64 - 0.68]$ and then either stays stable or starts increasing again. For the FT5 dataset, the MAE decreases initially, when the value of θ_{cnt} changes from 0 to 0.04 and then starts increasing when the value of θ_{cnt} increases beyond 0.04. For this reason, we choose the value θ_{cnt} to be 0.68 and 0.04 for SML and the FT5 datasets respectively. Similarly, the value of θ_{cnt} was trained for other datasets.

7.5.6 Threshold θ_{var}

In the hybrid algorithm, KMR^{var}_{hybrid}, the parameter θ_{var} controls the switching from the user-based prediction to the item-based prediction depending on the uncertainty in the predictions measured by the variance in the $\mathbf{W}_u \boldsymbol{\phi}(\boldsymbol{q}_i)$. To learn this parameter we changed its value from 0 to 1 in steps of 0.04 and observed the corresponding MAE. Figure 7.8 shows the parameter θ_{var} learned (for Case 1, as discussed in Section 7.6.4) over the validation. We observe that for the SML dataset, the MAE keeps on decreasing with the increase in the value of θ_{var}, reaches a minimum at 0.24, and then starts increasing again. For the FT5 dataset, the decrease in the MAE is not very significant, when $\theta_{var} < 0.44$, however, afterwards, a sharp decrease in the MAE is observed. The MAE keeps on decreasing, reaches its minimum at 0.64, and then either it stays stable or starts increasing again. For this reason we choose the optimal value θ_{var} to be 0.24

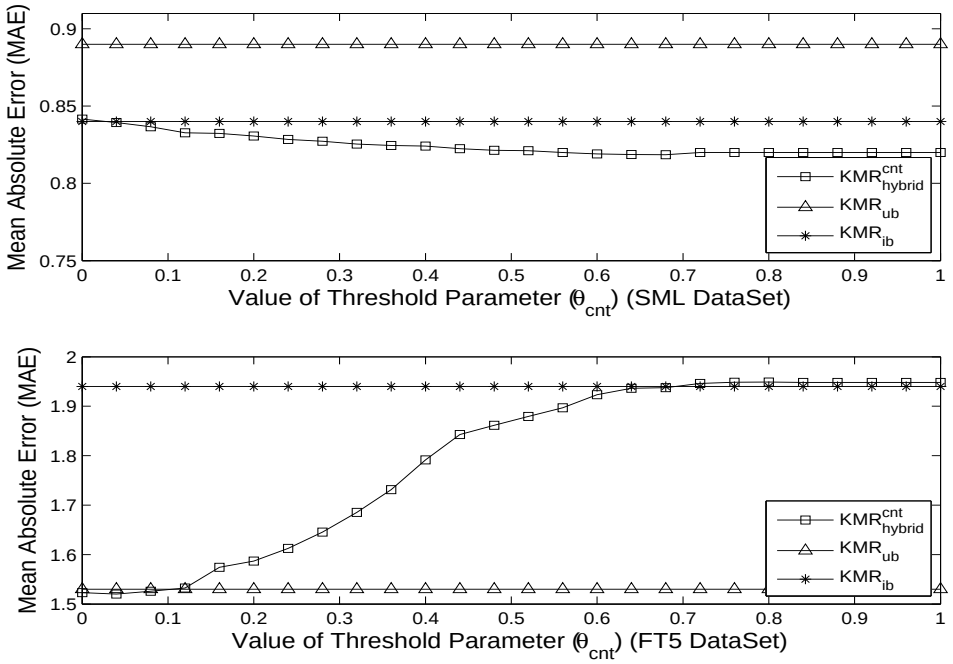


Figure 7.7: Learning the optimal value of threshold parameter θ_{cnt}, over the validation set.

and 0.64 for SML and the FT5 datasets respectively. Similarly, the value of θ_{var} were trained for other datasets.

7.5.7 Parameter σ and other parameters

We experimented with learning this parameter for each user, but found this computationally very expensive. We then tried grouping the users according to the variance in their ratings into 100 groups and tuned σ for each group. Although this gave improved performance, it was not found to be statistically significant. We therefore just used a single parameter $\sigma = 12$, which we tuned using a validation set.

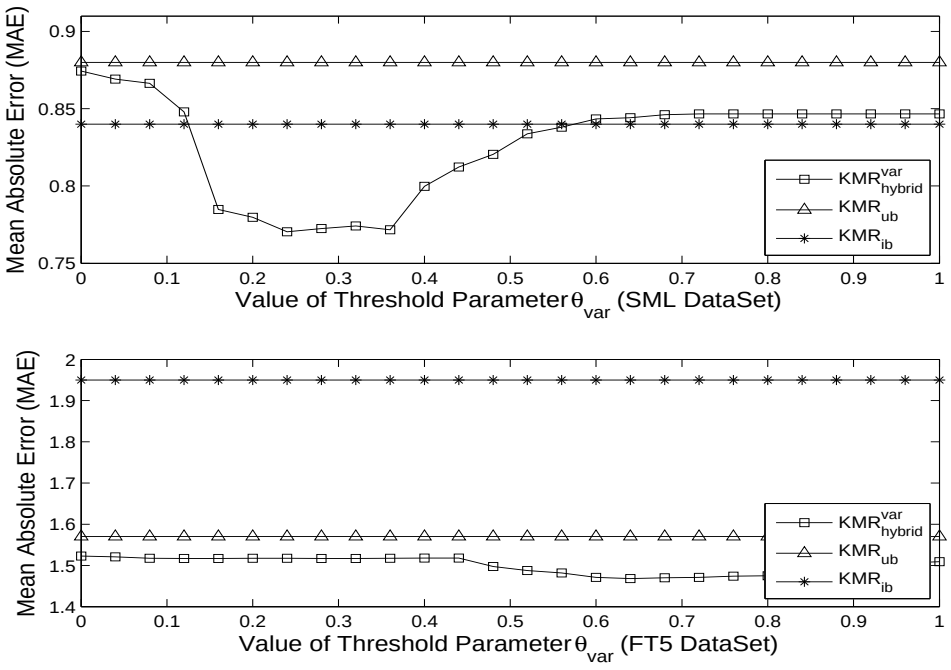


Figure 7.8: Learning the optimal value of threshold parameter θ_{var}, over the validation set.

Table 7.4: A comparison of the KMR algorithms with others in terms of the MAE. The best results are shown in bold font.

Algo.	**Best *MAE***				
	SML	**ML**	**ML**10	**FT5**	**NF**
$UBCF_{DV}$	0.746 ± 0.001	0.706 ± 0.000	0.678 ± 0.000	1.419 ± 0.008	0.713 ± 0.001
IBCF	0.764 ± 0.001	0.715 ± 0.000	0.675 ± 0.001	1.429 ± 0.006	0.719 ± 0.001
Hybrid CF	0.752 ± 0.001	0.702 ± 0.001	0.667 ± 0.000	1.427 ± 0.002	0.717 ± 0.001
SVD	0.774 ± 0.001	0.730 ± 0.001	0.691 ± 0.001	1.483 ± 0.005	0.725 ± 0.001
KMR_{ib}	0.715 ± 0.001	0.663 ± 0.001	0.638 ± 0.001	1.381 ± 0.002	0.684 ± 0.001
KMR_{ub}	0.731 ± 0.001	0.686 ± 0.001	0.649 ± 0.001	1.398 ± 0.002	0.687 ± 0.001
KMR^{var}_{hybrid}	$\mathbf{0.711 \pm 0.001}$	$\mathbf{0.658 \pm 0.001}$	$\mathbf{0.633 \pm 0.001}$	$\mathbf{1.377 \pm 0.002}$	$\mathbf{0.682 \pm 0.001}$

The parameter C (that punishes the slack variables in the Lagrange formulation) was fixed to 20, after initial experimentation.

7.6 Results and Discussion

In this section, we describe the results obtained from our experiments. In the following, we have denoted our proposed algorithm by KMR^{sup}_{sub}, where the subscript denotes the variant of the algorithm and the occasional superscript describes the variant in more detail where necessary. The main variants are item-based (ib), user-based (ub), feature-based (F) that use feature vectors rather than rating vectors, demographic (D) that use demographic vectors rather than rating vectors, and hybrid ($hybrid$) that uses a mixture of the user- and item-based predictions. For the hybrid algorithm we use the superscript to denote the different mechanisms for combining the user- and item-based predictions. When we use combinations of information, e.g. item-based ratings and features, we use KMR_{ib+F} to denote the case when we add the kernels and $KMR_{ib\otimes F}$ when we multiply the kernels. Finally, for the datasets with limited amount of ratings, instead of using the standard approach to predicting a new rating, we combined the standard approach (value of $\hat{r}$ that maximises the predictor $p(\hat{r})$) with the mean, mode and median of $\boldsymbol{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ (for the item-based approach). We denote this version of the algorithm with a superscript M^4.

7.6.1 Direct comparison

We compared our algorithms with three different algorithms: the user-based CF with default votes (Breese et al., 1998) (shown by $UBCF_{DV}$, which provides a useful baseline for comparing algorithms), the item-based CF (Sarwar et al., 2001) (shown by IBCF), and a SVD-based approach (Sarwar et al., 2000b) (shown by SVD). To provide as fair comparison as possible, we tuned all parameters of the algorithms.

Table 7.4 shows that the KMR algorithms outperforms all the aforementioned algorithms. The percentage decrease in error of KMR_{ib}, KMR_{ub}, and KMR^{var}_{hybrid} over the baseline approach is found to be 2.68%, 1.48%, and 2.96% for the FT5 dataset; 4.16%, 2.01%, and 4.69% for the SML dataset; 6.09%, 2.83%, and 6.80% for the ML dataset; 5.90%, 4.28%, and 6.64% for the ML10 dataset; and 4.07%, 3.65%, and 4.35% for the NF dataset. Appendix D compare the KMR with others in terms of ROC-sensitivity and F1 measure.

Table 7.5: A comparison of different algorithms in terms of the NMAE (Normalised MAE) for the ML dataset. The proposed algorithms outperform MatchBox (Stern et al., 2009), ImputedSVD (Ghazanfar and Prügel-Bennett, 2011a), and MMMF (Rennie and Srebro, 2005). They give the comparable results to E-MMF (DeCoste, 2006) and NLMF (Lawrence and Urtasun, 2009). Our results and the best results are shown in bold font.

Algorithm	**NMAE**
MatchBox	0.4206 ± 0.0055
ImputedSVD	0.4192 ± 0.0025
MMMF	0.4156 ± 0.0037
E-MMF	$\mathbf{0.4029 \pm 0.0027}$
NLMF Linear	0.4052 ± 0.0011
NLMF RBF	$\mathbf{0.4026 \pm 0.0020}$
KMR_{ib}	0.4125 ± 0.0034
KMR_{ub}	0.4251 ± 0.0032
KMR^{var}_{hybrid}	$\mathbf{0.4065 \pm 0.0021}$

7.6.2 Indirect comparison

In this section, we compare our results with other algorithms indirectly, i.e. we take the result from the respective papers without re-implementing them, which might make the comparison less than ideal. We conducted the weak generalization test procedures of Marlin (2004) using the All-But-One protocol—for each user in the training set a single rating is withheld for the test set. We averaged the results over the 3 random train-test splits as used in Lawrence and Urtasun (2009), Marlin (2004) and Rennie and Srebro (2005).

A comparison in terms of the Normalized MAE (NMAE) of the algorithms is given in Table 7.5. In Table 7.5, the MatchBox[3] is proposed in Stern et al. (2009), ImputedSVD is proposed in (Ghazanfar and Prügel-Bennett, 2011a), MMMF represents the Maximum Margin Matrix Factorisation algorithm proposed in (Rennie and Srebro, 2005), E-MMF represents the Ensemble Maximum Margin Matrix Factorisation technique proposed in (DeCoste, 2006), and NLMF represents the Non-Linear Matrix Factorisation technique (with linear and RBF versions) as proposed in Lawrence and Urtasun (2009).

Table 7.5 shows that the NLMF and E-MMF perform better than the rest. The proposed hybrid algorithm gives slightly poorer results to them with NMAE = 0.4065. It is worth mentioning that the E-MMF is an ensemble of about 100 predictors, which makes this algorithm unattractive. From this table, we may conclude that the proposed algorithm is comparable to the state-of-the-art algorithm for the MovieLens ($1M$) dataset.

[3]The authors did not provide any numeric value, only a graph is presented showing the minimum value approximately to 0.673. Furthermore, the test procedures of this paper were different from the rest.

Table 7.6: A comparison of different algorithms in terms of the RMSE for the ML10 dataset. NLMF represents the Non-Linear Matrix Factorisation technique as proposed in Lawrence and Urtasun (2009) and M^3F-TIB represents the Mixed Membership Matrix Factorisation model as proposed in Mackey et al. (2010). Our results and the best results are shown in bold.

Algorithm	**RMSE**
NLMF	0.8740 ± 0.02
M^3F-TIB	$\mathbf{0.8447 \pm 0.009}$
KMR_{ib}	0.8721 ± 0.011
KMR_{ub}	0.8994 ± 0.015
KMR^{var}_{hybrid}	$\mathbf{0.8612 \pm 0.011}$

To the best of our knowledge, the best results for the MovieLens $10M$ dataset that has been reported in the literature are those proposed in Lawrence and Urtasun (2009) and Mackey et al. (2010). They claimed their proposed algorithm gives RMSE accuracy of 0.8740 ± 0.02 and 0.8447 ± 0.009, respectively. We followed their experimental setup and the results have been shown in Table 7.6. Table 7.6 shows that the proposed algorithms outperform Lawrence and Urtasun (2009)'s results. The percentage improvement is found to be 1.46% in the case of KMR^{var}_{Hybrib}. The M^3F-TIB algorithm gave the best results outperforming our best algorithm KMR^{var}_{hybrid} with 1.92% decrease in error.

Unfortunately, no NMAE (or MAE) was provided for M^3F-TIB technique (Mackey et al., 2010) over the MovieLens 1M dataset, which makes it harder to compare different algorithms results with M^3F-TIB. Considering these result, we conclude that our approach appears to be competitive with the current state-of-the-art.

7.6.3 Combining different kernels

As discussed in Section 7.4.3, there can be different sources of information that can be used for making recommendations. Our framework allows these different sources to be exploited by combining different kernels built from different information vectors. In particular, we consider the rating information, feature information, and demographics information as described in Section 7.4.3.

Table 7.7 shows the performance of the different combinations of kernels on the SML dataset. We have shown not only the Mean Absolute Error (MAE), but also a number of measures of the ability to classify films as either highly rated or poorly rated. We observe reasonable performance using just rating information, demographic information and feature information. Interestingly, for this dataset, combining kernels does not give significantly better performance than using a kernel based on a single source of information. A plausible explanation of this observation is that our error rates are close to the optimum that can be achieved (there is a limit on the performance of any system due to the fickleness of the users making the ratings). Or, at least, we are close to the

optimum given the way we have represented the problem. On other datasets where, for example, ratings for some users are very sparse, demographic and feature information can be much more significant. The other striking feature of Table 7.7 is that multiplying kernels together seem to be more successful than adding different kernels.

Similar results (not shown) were observed in the case of FilmTrust and MovieLens ML10 datasets. We also attempted to linearly combine the predictions from different kernels, but again this gave no improvement.

7.6.4 Combining the user- and item-based versions

The methods of combining the user- and item-based versions (mentioned in Section 7.4.2) did not give any significant improvement over the individual results for the whole dataset. To check the performance for imbalanced datasets, we (randomly) selected 200 users and 300 movies from the SML dataset, and 200 users and 50 movies from the FT5 dataset; and randomly withheld their $x\%$ ratings. We checked the performance for two cases: for Case 1, the value of x lies between 0 to 50 (i.e. $x \in \{0, 50\}$), whereas for Case 2, the value of x lies between 0 to 100 (i.e. $x \in \{0, 100\}$). The latter case creates a relatively imbalanced subset of the dataset as compared to the former one.

Table 7.8 shows the performance of user-based, item-based, and different methods used to combine the individual versions. We use the average of the user- and item-based versions as a baseline. We observe that linearly combining the individual recommender systems does not give significant improvement over the baseline and the same is true for the second method (discussed in Section 7.4.2). However, KMR^{var}_{hybrid} does significantly improve the performance, with p-value in the case of pair-t test compared with the baseline recommender (max) found to be less than 10^{-6} for both datasets. Similar results were observed for other datasets as well. What is evident from Table 7.8 is that the user- and item-based versions of the algorithm are complementary and can improve the performance, if combined in a systematical way, for the imbalanced dataset.

Furthermore, we combined different kernel's results linearly; however, again it did not give any significant improvement in the results.

Table 7.7: Comparing the performance of KMR algorithms found with different combinations of kernel for the SML dataset. The best results are shown in bold font.

Algorithm	**MAE**	**ROC**	**Precision**	**Recall**	**F**1
KMR_{ib}	0.715 ± 0.001	$\mathbf{0.708 \pm 0.002}$	$\mathbf{0.562 \pm 0.002}$	$\mathbf{0.546 \pm 0.005}$	$\mathbf{0.533 \pm 0.003}$
KMR_{D}	0.748 ± 0.001	0.692 ± 0.002	0.546 ± 0.003	0.532 ± 0.004	0.505 ± 0.004
KMR_{F}	0.729 ± 0.001	0.693 ± 0.002	0.552 ± 0.003	0.526 ± 0.005	0.506 ± 0.003
KMR_{ib+F+D}	0.733 ± 0.001	0.705 ± 0.002	0.550 ± 0.002	0.540 ± 0.003	0.517 ± 0.002
KMR_{ib+F}	0.721 ± 0.001	0.706 ± 0.003	0.561 ± 0.002	0.545 ± 0.005	0.522 ± 0.002
KMR_{ib+D}	0.732 ± 0.001	0.705 ± 0.002	0.556 ± 0.003	0.542 ± 0.005	0.517 ± 0.003
KMR_{F+D}	0.735 ± 0.001	0.699 ± 0.002	0.544 ± 0.002	0.516 ± 0.005	0.501 ± 0.002
$KMR_{ib \otimes F \otimes D}$	0.736 ± 0.001	0.697 ± 0.003	0.551 ± 0.002	0.542 ± 0.016	0.510 ± 0.003
$KMR_{ib \otimes F}$	$\mathbf{0.714 \pm 0.001}$	0.698 ± 0.002	0.555 ± 0.004	0.532 ± 0.005	0.510 ± 0.003
$KMR_{ib \otimes D}$	0.727 ± 0.001	0.705 ± 0.002	0.554 ± 0.002	0.542 ± 0.003	0.518 ± 0.003
$KMR_{F \otimes D}$	0.739 ± 0.001	0.695 ± 0.002	0.551 ± 0.002	0.540 ± 0.003	0.509 ± 0.003

Table 7.8: Combining the user-based and item-based versions under imbalanced datasets. The Case 2 produces a relatively sparse subset of the dataset compared to Case 1. The best results are shown in bold font.

Approach	**MAE**			
	Case1		**Case**2	
	FT5	**SML**	**FT5**	**SML**
KMR_{ib}	1.969 ± 0.002	0.882 ± 0.002	1.996 ± 0.002	0.941 ± 0.002
KMR_{ub}	1.525 ± 0.001	0.831 ± 0.002	1.751 ± 0.001	0.903 ± 0.002
$(KMR_{ib} + KMR_{ub})/2$	1.675 ± 0.002	0.829 ± 0.002	1.763 ± 0.002	0.901 ± 0.002
KMR^{linear}_{hybrid}	1.524 ± 0.002	0.826 ± 0.002	1.715 ± 0.002	0.895 ± 0.002
KMR^{cnt}_{hybrid}	1.516 ± 0.002	0.825 ± 0.001	1.704 ± 0.002	0.903 ± 0.001
KMR^{var}_{hybrid}	$\mathbf{1.463 \pm 0.002}$	$\mathbf{0.765 \pm 0.001}$	$\mathbf{1.545 \pm 0.002}$	$\mathbf{0.802 \pm 0.001}$

Table 7.9: Comparing the MAE observed in different approaches under the **new user cold-start scenario**, for the SML dataset. The superfix M^4 represents the corresponding version of the KMR algorithms, where we take into account the max, mean, mode, and median of the output probability distribution. Average represents the average rating given by all users in the dataset. The best results are shown in bold font.

Approach	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
KMR_{ib}	3.841 ± 0.002	3.542 ± 0.002	2.872 ± 0.002	2.683 ± 0.002	2.504 ± 0.002
KMR_{ub}	2.102 ± 0.002	1.984 ± 0.002	1.672 ± 0.002	1.547 ± 0.002	1.374 ± 0.001
KMR_{D}	3.623 ± 0.002	3.321 ± 0.002	2.091 ± 0.002	1.955 ± 0.002	1.896 ± 0.002
KMR_{F}	3.652 ± 0.002	3.452 ± 0.002	1.944 ± 0.002	1.836 ± 0.002	1.757 ± 0.002
$KMR_{ib}^{M^4}$	0.858 ± 0.002	0.851 ± 0.002	0.809 ± 0.001	0.790 ± 0.001	0.784 ± 0.001
$KMR_{ub}^{M^4}$	$\mathbf{0.843 \pm 0.002}$	$\mathbf{0.841 \pm 0.002}$	$\mathbf{0.795 \pm 0.001}$	$\mathbf{0.776 \pm 0.001}$	$\mathbf{0.774 \pm 0.001}$
$KMR_{F}^{M^4}$	0.860 ± 0.002	0.856 ± 0.002	0.814 ± 0.002	0.801 ± 0.002	0.783 ± 0.002
$KMR_{D}^{M^4}$	0.866 ± 0.002	0.865 ± 0.002	0.815 ± 0.002	0.795 ± 0.002	0.786 ± 0.002
$KMR_{ib+F}^{M^4}$	0.859 ± 0.002	0.857 ± 0.002	0.810 ± 0.002	0.786 ± 0.002	0.779 ± 0.002
Average	0.887 ± 0.002	0.883 ± 0.002	0.863 ± 0.002	0.851 ± 0.002	0.829 ± 0.002

7.6.5 Sparse, skewed, and imbalanced datasets

In practical applications recommender systems often have access to limited and highly skewed information. Examples of these are

New user cold-start scenario where new users have relatively few ratings.

New item cold-start scenario where new items have relatively few ratings.

Long tail scenario where the majority of items have only a few ratings.

Imbalanced sparse datasets where the majority of users/items have only a few ratings.

In the datasets that we have used so far our test set consists of randomly chosen ratings and these are overwhelming in the dense region of the rating matrix. That is, the users that we tested, typically have rated many items and the items have been rated by many users. Thus, the results we have described so far are not strongly influenced by problems of limited and skewed information. However, these problems are often vital for a recommender system to prosper. For example, to attract new users it is highly beneficial to be able to give them good quality recommendations before they have made many ratings. Similarly, to introduce new items into the system it is useful to make sensible recommendations even if the item has only gained a few ratings.

We have tested the four scenarios outlined above by modifying the datasets we have been using to exaggerate the sparseness or skewness of the data. We found that in all cases the standard predictor that we have been using up to now gives very poor performance. However, we could very substantially improve the performance by combining the standard predictor with predictions using the mean, median and mode of $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as described in Section 7.3.2. In the tables shown below we denote the modified predictor with a superscript M^4.

We concentrate on the new user cold-start scenario as the results are representative of all four scenarios. The only major difference is in the new-item cold-start scenario where the feature-based and demographic-based recommenders also perform well as they are less influenced by a lack of ratings. Results for the new item cold-start, long tail, and sparse data scenarios are given in Appendix D.

7.6.5.1 New user cold-start scenario

To test the performance of the proposed algorithms under the new user cold-start scenario, we selected 100 random users, and kept their number of ratings in the training set to 2, 5, 10, 15, and 20. Keeping the number of ratings less than 20 ensures that a user

is new and it captures well the new user cold-start problem. The corresponding MAE, represented by $MAE2$, $MAE5$, $MAE10$, $MAE15$, and $MAE20$ is shown in Table 7.9. Using the standard predictor provides very poor performance. We can substantially improve the performance by combining the standard predictor with predictions using the mean, median and mode of $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as described in Section 7.3.2.

Recall that we learn the weights for combining the standard predictor with the predictor using the mean, mode and median. The value of the weights depend on the dataset. Figure 7.9 shows how the weights that have been learned change in the new user cold-start scenario as we increase the number of ratings in the training set[4]. The x-axis shows the number of ratings given by users (selected as cold-start users) and the y-axis shows the weights associated with different predictors. We observe that the contribution of the mode, mean, and median predictors decreases with the increase in the number of ratings, and finally become zero when the maximum number of ratings are available. Whereas, the contribution of the standard (ratings-based) predictor increases with the increase in the number of ratings, and becomes 1 when the maximum number of ratings are available.

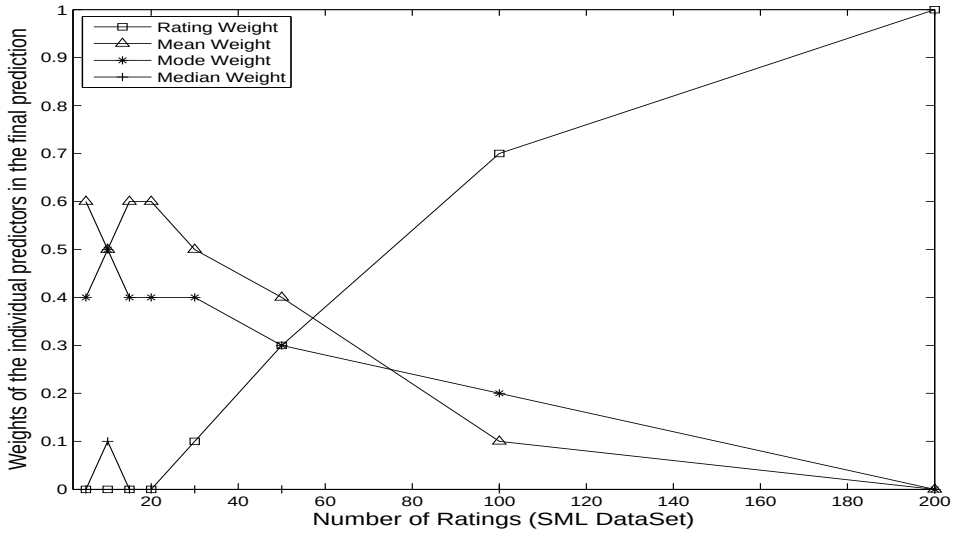


Figure 7.9: Weight learning over the validation set for the new user cold-start problem (SML dataset). "Number of Ratings" represents the number of ratings given by an active user in the training set.

7.6.6 Two-way clustering

Our intuition in the clustering algorithm being that a particular group of users might share their taste on whole group of items. To test the hypothesis, we clustered both the users and items using the clustering techniques described earlier. Although there was

[4]The new user cold-start scenario is taken as an example; similar results were observed in both the new item cold-start and long tail scenarios.

a slight improvement in the performance it was not statistically significant. It remains unclear whether the reasons for not getting a larger improvement is due to the fact that users and items do not neatly cluster, the proposed algorithms successfully exploit this information anyways, or if our clustering algorithm was too naive.

7.7 Conclusion

In this chapter, we propose a new class of Kernel Mapping Recommender (KMR) system algorithms that give state-of-the-art performance and eliminates the recorded problems with the recommender systems making the recommendation generation techniques applicable to a wider range of practical situations and real-world scenarios. The proposed kernel-mapping methods is competitive with what we believe to be the recommender with the best performance proposed by Lawrence and Urtasun (2009) and Mackey et al. (2010). Interestingly both the proposed algorithm and Lawrence and Urtasun (2009) recommender use kernel-based methods though in a very different way. Although, kernel-based techniques are known to give excellent performance, recommender systems are challenging because of the size of the datasets. By carefully choosing the constraints we have been able to create a kernel-based learning machine that can be trained in linear time in the number of data points.

The algorithm we have developed is very flexible, thus we can easily adapt it so that it is either user-based or item-based. In addition it can use other information such as text-based features and these features can be easily combined. Furthermore, context information can easily be added in the form of kernels[5]. The best algorithm on the large datasets switches between the user- and item-based information depending on the reliability of the predictions as measured by the spread in the output probability distribution of the algorithms.

One interesting feature of our approach is that we map the residues in the ratings onto a density function which encodes the uncertainty in the residue. For unseen residues we have interpreted the mapping $\mathbf{W}_u\boldsymbol{\phi}(\boldsymbol{q}_i)$ as an approximation to a density function for the residue. Even though this function is not itself a density function (it becomes negative in some regions and is not normalised), nevertheless, it is very useful to consider the positive part of the function as a density function from which we can measure the mean, mode, median and variance. These measurements help in improving the performance, particularly in the case of sparse data.

Based on the experimental results, we can underline five interesting points: (1) the user- and item-based KMR are complementary and can be combined in a systematical way to improve the performance of a recommender system; (2) a hybrid recommender system

[5]Refer to Appendix D for an example of adding context information.

that combines the user- and item-based KMR, by taking into account the variance in the output probability distribution, provides more accurate recommendations than the individual ones under the sparse and imbalanced datasets; (3) adding more information in the form of kernel does not give any significant improvements in the results, however it does help in the cold-start scenarios; (4) the *max* predictor fails under cold-start, long tail, and sparse dataset scenarios, and the performance can be improved by taking into account the *max*, *mean*, *mode*, and *median* of the output probability distribution; and (5) two-way clustering of the dataset does improve the performance; however, it is statistically insignificant.

Chapter 8

Incremental and On-line Kernel Mapping Recommender (KMR) System Algorithms

8.1 Introduction

The issue of large scale learning is becoming an increasingly important topic of interest in data mining and machine learning communities. As most of the recommender systems consist of millions of ratings and moreover they are dynamic—new data (users or items) are being continuously added to the system—estimating a reasonably efficient, compact, and accurate prediction function is a difficult problem, that has attracted a number of researchers, and a range of algorithms have been proposed.

The batch-processing algorithms, which require multiple passes through the dataset, are not pragmatic for these scenarios. Certain tricks (e.g. updating certain rows or columns of the data matrix) in the batch-processing algorithms can help saving computation time and memory. In contrast to the batch processing algorithms, the on-line algorithms construct a hypothesis by processing data points one at a time as they arrive and update it whenever the new data are available. These algorithms are typically fast, memory-efficient and simple to implement, and have the ability to adapt and learn in difficult situations (Bassam, 2010). An important practical advantage of the on-line algorithms is that they can accommodate the incremental update of the model upon the arrival of new data, without re-training from scratch. They are well suited to (1) large scale datasets where the off-line model building is the most computationally expensive task, and (2) to situations where the data arrive continuously, for example in e-commerce websites.

The Kernel Mapping Recommender (KMR) system algorithms introduced in the previous chapters build the model using an off-line stage and hence are not well suited to dynamic environments. For practical recommender systems this is a significant problem, as with the incremental and gradual arrival of the new data, it is desirable that the updates are performed on such data. It is unrealistic to recompute the model from scratch, based on these updates, due to the tremendous cost related to computation time and storage capacity. From this line of research, first, we have introduced a heuristic method that we call KMR^{incr}, which can be used to update the model effectively upon the arrival of new data. Second, we have proposed a perceptron-like algorithm namely $KMR^{percept}$, which is a novel, fast, on-line algorithm for learning on the large recommender system datasets. The proposed algorithm implements an incremental subgradient descent step (Bertsekas, 1999; Cristianini and Shawe-Taylor, 2000) to minimise a utility function similar to the one described in the previous chapter. The implemented version of the algorithm follows the dual perceptron schema where only the knowledge of the corresponding kernels is required.

Both of the proposed algorithms overcome the *accuracy* and *scalability* problems associated with a recommender system. Furthermore, the proposed algorithm, $KMR^{percept}$, overcomes the *stability vs. plasticity problem* (Burke, 2002). Once a detailed user profile has been established over a period of time, it becomes hard to change it. This is a potential problem with many of the recommender systems, which is sometimes referred to as the user-interest drifting problem. For example, in a movie recommender system, if a user was interested in action movies last year, but their taste then changed to romantic movies, then they would not receive useful recommendations. This is because, up to this point, their profile has been heavily shaped by action movies. Hybrid recommender systems employing the knowledge-based approaches (Burke, 1999) are less affected by this problem; however, it is desirable to solve this problem using only the rating information. This has motivated us to use the temporal discount concept—giving less weight to the old ratings—that can solve this problem effectively.

The rest of the chapter has been organised as follows: Section 8.2 outlines the proposed algorithms, namely KMR^{incr} and $KMR^{percept}$. Section 8.3 describes the experimental setup of the work. Section 8.4 shows results comparing the performance of the proposed algorithms with the baseline ones. Finally Section 8.5 concludes the work.

8.2 Proposed Algorithms

In the following, the base dataset, denoted by $\mathcal{D}^{base}$, represents the dataset used to train the initial model and the resulting model is called the base model. Similarly, the dataset added afterwards, denoted by $\mathcal{D}^{new}$, represents the new dataset and the resulting model is called the updated model. Let $\mathcal{I}$, $\mathcal{U}$, E, E_i, E_u, α, and E_α represent the model

items, users, overall average, items' average, users' average, design variables, and design variables' average respectively. Furthermore, let $\tilde{\mathcal{I}}$, $\tilde{\mathcal{U}}$, $\tilde{E}$, $\tilde{E}_i$, $\tilde{E}_u$, $\tilde{\alpha}$, and $\tilde{E}_{\tilde{\alpha}}$ denote the corresponding base model parameters.

8.2.1 KMR^{incr}

The pseudo-code of the proposed algorithm is given in Algorithm 7. From steps 3 to 4, we initialise the model parameters: items' vector, users' vector, overall average, users' average, items' average, design variable, and design variable's average. We then build the base model with $\mathcal{D}^{base}$. In step 6, we find the mean of the design variables ($\tilde{\alpha}$) computed while building the base model. We update the model by adding $\mathcal{D}^{new}$ to the existing dataset and initialise the new model parameters by the base model parameters.

In the solver procedure, from steps 10 to 12, we read the data. From steps 13 to 18, we initialise the design variables which are different for $\mathcal{D}^{base}$ and $\mathcal{D}^{new}$. Formally, the initialisation of the design variables for two different type of datasets is as follows:

$$\alpha_{iu} = \begin{cases} \frac{\sum_{i',u' \in \mathcal{D}^{base}} \tilde{\alpha}_{i'u'}}{|\mathcal{D}^{base}|}, & \text{if } (i,u) \in \mathcal{D}^{new}, \\ \tilde{\alpha}_{iu} & \text{otherwise.} \end{cases} \tag{8.1}$$

From steps 19 to 28, we keep track of the users and items entering the system by updating the users' and items' vector. From steps 30 to 34, we update the total, users', and items' averages. Then from steps 36 to 37, we compute the residual ranks and the feature vectors. Next, we compute the kernel function as shown in step 38. Afterwards, we can find the optimal design variables by feeding the above found parameters to the optimiser (see Chapter 7, Section 7.3.1), as shown in step 40.

8.2.2 $KMR^{percept}$

The on-line implementation of the KMR recommender system is realised via a perceptron-type algorithm. The problem behind the perceptron algorithm is derived from the following optimisation problem (described in Chapter 7, Section 7.4);

$$\begin{array}{ll} \text{min} & \frac{1}{2}\sum_{u \in \mathcal{U}} \|\mathbf{W}_u\|^2 + C\sum_{i \in \mathcal{I}} \zeta_i \\ \text{with respect to} & \mathbf{W}_u,\ u \in \mathcal{U},\ \zeta_i,\ i \in \mathcal{I} \\ \text{subject to} & \langle \boldsymbol{\psi}(\hat{r}_{iu}), \mathbf{W}_u \boldsymbol{\phi}(\boldsymbol{q}_i) \rangle \geq 1 - \zeta_i \\ & \zeta_i \geq 0,\ i \in \mathcal{I},\ u \in \mathcal{U}_i, \end{array} \tag{8.2}$$

by ignoring the regularisation term and only minimising the sum of the slack variables, i.e. $\sum_{i \in \mathcal{I}} \zeta_i$ which measures the value of the overall loss. The implemented version of

Algorithm 7 :KMR^{incr}; Build and update the model
Input: $\mathcal{D}^{base}$, base dataset; $\mathcal{D}^{new}$, new dataset
Output: $\mathcal{I}$, items; $\mathcal{U}$, users; $E(t)$, overall average; $E_u(t)$, users' average; $E_i(t)$, items' average; α, design variables

procedure BUILDMODEL($\mathcal{D}^{base}, \mathcal{D}^{new}$)
  ## Initialise the model parameters
  $\mathcal{I} = \emptyset$; $\mathcal{U} = \emptyset$;
  $E = 0$, $E_u = 0$, $E_i = 0$, $\alpha = 0$ $E_\alpha = 0$
  ## Build the base model
  ($\tilde{\mathcal{I}}$, $\tilde{\mathcal{U}}$, $\tilde{E}$, $\tilde{E}_u$, $\tilde{E}_i$, $\tilde{\alpha}$) = SOLVER($\mathcal{I}$, $\mathcal{U}$, E, E_u, E_i, E_α,α, $\mathcal{D}^{base}$, $\emptyset$)
  $\tilde{E}_{\tilde{\alpha}} = \dfrac{\sum_{i,u \in \mathcal{D}^{base}} \tilde{\alpha}_{iu}}{|\mathcal{D}^{base}|}$
  ## Update the base model
  ($\mathcal{I}$, $\mathcal{U}$, E, E_u, E_i, α) = SOLVER ($\tilde{\mathcal{I}}$, $\tilde{\mathcal{U}}$, $\tilde{E}$, $\tilde{E}_u$, $\tilde{E}_i$, $\tilde{E}_{\tilde{\alpha}}$,$\tilde{\alpha}$,$\mathcal{D}^{base}$,$\mathcal{D}^{new}$)
end procedure

Solve the optimisation problem and find the design variables
procedure SOLVER($\mathcal{I}$, $\mathcal{U}$, $E(t-1)$, $E_u(t-1)$, $E_i(t-1)$, E_α, α, $\mathcal{D}^{base}$, $\mathcal{D}^{new}$)
  for all $t \in \{1, \cdots, \mathcal{D}^{base} \cup \mathcal{D}^{new}\}$ **do**
    read ($u(t)$,$i(t)$,$r_{iu}(t)$) ## user, item, rating
    $u = u(t)$;$i = i(t)$;$r_{iu} = r_{iu}(t)$
    ## Initialization of design variables
    if $i \notin \mathcal{I}$ || $u \notin \mathcal{U}$ **then** ## for new, earlier unseen items and users
      $\alpha_{iu}(t) = E_\alpha$
    else
      $\alpha_{iu}(t) = \alpha_{iu}(t)$ ## for earlier seen items
    end if
    if $i \notin \mathcal{I}$ **then**
      $\mathcal{I} = \mathcal{I} \cup \{i\}$; $\mathcal{U}_i = \emptyset$
      $E_i(t-1) = 0$
    end if
    if $u \notin \mathcal{U}$ **then**
      $\mathcal{U} = \mathcal{U} \cup \{u\}$; $\mathcal{I}_u = \emptyset$
      $E_u(t-1) = 0$
      $\mathcal{U}_i = \mathcal{U}_i \cup \{u\}$
      $\mathcal{I}_u = \mathcal{I}_u \cup \{i\}$
    end if
    ## Update average values
    if $\mathcal{D}^{new} = \emptyset$ || $i, u \in \mathcal{D}^{new}$ **then**
      $E(t) = \dfrac{(t-1)E(t-1) + r_{iu}(t)}{t}$
      $E_u(t) = \dfrac{(|\mathcal{I}_u| - 1)E_u(t-1) + r_{iu}(t)}{|\mathcal{I}_u|}$
      $E_i(t) = \dfrac{(|\mathcal{U}_i| - 1)E_i(t-1) + r_{iu}(t)}{|\mathcal{U}_i|}$
    end if
    ## Compute residual ranks and inner products
    $\hat{r}_{iu} = \hat{r}_{iu}(t) = r_{iu}(t) - E_i(t) - E_u(t) + E(t)$
    $\mathbf{q}_i(t) = (\hat{r}_{iu} \; | u \in \mathcal{U}_i)$
    $\gamma^i_{nu}(t) \stackrel{\text{def}}{=} K_{\hat{r}}(\hat{r}_{iu}(t), \hat{r}_{nu}(t)) K_{\mathbf{q}}(\mathbf{q}_i(t), \mathbf{q}_n(t))$
  end for
  Solve the optimisation problem given $\mathcal{I}$, $\mathcal{U}$, $E(t)$, $E_u(t)$, $E_i(t)$, $\gamma_{nu}(t)$,α
  return ($\mathcal{I}$, $\mathcal{U}$, $E(t)$, $E_u(t)$, $E_i(t)$, α)
end procedure

Algorithm 8 :$KMR^{percept}$; Sequentially process the data and build the model
Input: $\mathcal{D}^{percept}$, dataset
Output: $\mathcal{I}$, items; $\mathcal{U}$, users; $E(t)$, overall average; $E_u(t)$, users' average; $E_i(t)$, items' average; α, design variables

procedure BUILDMODEL($\mathcal{D}^{percept}$)
  ##Initialise model parameters
  $k = 0$; $\mathcal{I} = \emptyset$; $\mathcal{U} = \emptyset$; $E(t) = 0$; $s > 0$; ##step size
  $0 < \beta < 1$; $\beta_k = 1$ ##discount factor and discount initialisation
  for all $t \in \{1, \cdots, \mathcal{D}^{percept}\}$ **do**
    read $(i(t),u(t),r_{iu}(t))$
    $i = i(t)$;$u = u(t)$;$r_{iu} = r_{iu}(t)$
   ## Initialisation for new, earlier unseen items and users
    if $i \notin \mathcal{I}$ || $u \notin \mathcal{U}$ **then**
      $\alpha_{iu}(t) = 0$
    end if
    if $i \notin \mathcal{I}$ **then**
      $\mathcal{I} = \mathcal{I} \cup \{i\}$; $\mathcal{U}_i = \emptyset$
      $E_i(t-1) = 0$
      $\zeta_i(t) = 0$
    end if
    if $u \notin \mathcal{U}$ **then**
      $\mathcal{U} = \mathcal{U} \cup \{u\}$; $\mathcal{I}_u = \emptyset$
      $E_u(t-1) = 0$
      $\mathcal{U}_i = \mathcal{U}_i \cup \{u\}$
      $\mathcal{I}_u = \mathcal{I}_u \cup \{i\}$
    end if
    ## Update average values
    $E(t) = \dfrac{(t-1)E(t-1) + r_{iu}(t)}{t}$
    $E_u(t) = \dfrac{(|\mathcal{I}_u| - 1)E_u(t-1) + r_{iu}(t)}{|\mathcal{I}_u|}$
    $E_i(t) = \dfrac{(|\mathcal{U}_i| - 1)E_i(t-1) + r_{iu}(t)}{|\mathcal{U}_i|}$
    ## Compute residual ranks
    $\hat{r}_{iu} = \hat{r}_{iu}(t) = r_{iu}(t) - E_i(t) - E_u(t) + E(t)$
    $\mathbf{q}_i(t) = (\hat{r}_{iu} \; | u \in \mathcal{U}_i)$
    ## Test the constraint belonging to $(i(t), u(t))$
    if $\sum_{n \in \mathcal{I}_u} \alpha_{nu}(t)\gamma^i_{nu}(t) < 1 - \zeta_i(t)$ **then**
      ## Discounted update of the variables
      $\beta_{k+1} = 1 + \beta\beta_k$
      $k = k + 1$; ## update counter
      for all $n \in \mathcal{I}_u$ **do**
        $\alpha_{nu}(t+1) = \frac{1}{\beta_k}[\beta\alpha_{nu}(t) + s\gamma^i_{nu}(t)]$
      end for
      if $\zeta_i(t) > 1 - \sum_{n \in \mathcal{I}_u} \alpha_{nu}(t+1)\gamma^i_{nu}(t)$ **then**
        $\zeta_i(t+1) = 1 - \sum_{n \in \mathcal{I}_u} \alpha_{nu}(t+1)\gamma^i_{nu}(t)$
      end if
      ## Where we used the shorthand notation
      $\gamma^i_{nu}(t) \stackrel{\text{def}}{=} K_{\hat{r}}(\hat{r}_{iu}(t), \hat{r}_{nu}(t))K_{\mathbf{q}}(\mathbf{q}_i(t), \mathbf{q}_n(t))$
    end if
  end for
  return $(\mathcal{I}, \mathcal{U}, E(t), E_u(t), E_i(t), \alpha)$
end procedure

the perceptron algorithm follows the dual perceptron schema where only the knowledge of the corresponding kernels is required.

The pseudo-code of the proposed perceptron-like algorithm, $KMR^{percept}$, is given in Algorithm 8. From steps 3 to 4, we initialise the model parameters[1]: counter (that counts the number of updates of the design variable), items' vector, users' vector, overall average, step size, discount factor, and discount parameter. From steps 5 to 7, we read the arriving data. From steps 8 to 21, we initialise the design variable of the rating made by user u on item i to zero if either the user or item is new; initialise the item slack variable to zero if the arriving item has not been rated by anyone in the system; and keep track of how many users and items have entered the system by adding them to users' and items' vector. From steps 22 to 25, we update the total, users', and items' average. Then we compute the residual ranks and feature vectors from steps 26 to 28.

The part of the algorithm from steps 29 to 42 implements an incremental subgradient descent step (Bertsekas, 1999; Nocedal and Wright, 1999; Cristianini and Shawe-Taylor, 2000) to minimise a problem similar to the one described in equation 8.2. After omitting the regularisation term $\frac{1}{2}\|\mathbf{W}\|^2$, the objective function in equation 8.2 becomes:

$$\min_{\mathbf{W}} \sum_{i\in\mathcal{I}} \max\left(0, 1 - \min_{u\in\mathcal{U}_i} \langle \psi(\hat{r}_{iu}), \mathbf{W}_u \phi(q_i)\rangle\right), \tag{8.3}$$

with the substitution

$$\zeta_i = \max\left(0, 1 - \min_{u\in\mathcal{U}_i} \langle \psi(\hat{r}_{iu}), \mathbf{W}_u \phi(q_i)\rangle\right). \tag{8.4}$$

Based on the Lagrangian of problem (8.2), assume that there are non-negative real numbers $\{\alpha_{iu}\}$, $(i,u) \in \mathcal{D}$, such that $\mathbf{W}_u$ for any u can be expressed by:

$$\mathbf{W}_u = \sum_{n\in\mathcal{I}_u} \alpha_{nu} \psi(\hat{r}_{nu}) \otimes \phi(q_n). \tag{8.5}$$

Equation 8.5 tells us that $\mathbf{W}_u$ can be represented in the tensor product space of the Hilbert spaces $\mathcal{H}_\phi$ and $\mathcal{H}_\psi$ with non-negative coefficients.

After substituting the value of $\mathbf{W}_u$ from equation 8.5 for all u into equation 8.3 we have

$$\begin{array}{ll} \min & \sum_{i\in\mathcal{I}} \max\left(0, 1 - \min_{u\in\mathcal{U}_i} \sum_{n\in\mathcal{I}_u} \alpha_{nu}\gamma^i_{nu}\right), \\ \text{w.r.t.} & \boldsymbol{\alpha} \in \mathbb{R}^{|\mathcal{D}|} \\ \text{s.t.} & \boldsymbol{\alpha} \geq \mathbf{0}, \end{array} \tag{8.6}$$

where

$$\gamma^i_{nu} \stackrel{\text{def}}{=} \underbrace{\langle \boldsymbol{\psi}(\hat{r}_{iu}), \boldsymbol{\psi}(\hat{r}_{nu})\rangle}_{K_{\hat{r}}(\hat{r}_{iu}, \hat{r}_{nu})} \underbrace{\langle \boldsymbol{\phi}(\mathbf{q}_i), \boldsymbol{\phi}(\mathbf{q}_n)\rangle}_{K_{\boldsymbol{q}}(\boldsymbol{q}_i, \boldsymbol{q}_n)}. \tag{8.7}$$

[1] In this work, we choose $\beta = 0.5$ and $s = 0.5$.

The optimisation problem of (8.6) has a convex, non-differentiable, piecewise linear objective function. In minimising an objective function similar to this we can use some variants of the subgradient descent method. In this case, we can follow three basic strategies:

- Computing the subgradient of the entire objective function.
- Using the coordinate descent algorithm (Cristianini and Shawe-Taylor, 2000) which optimises only one variable in each of the steps.
- Exploiting the structure of the objective function, which can be partitioned into blocks relating to the users, we can apply a block descent technique optimising only one block in each of the steps.

The last two methods are sub-cases of so-called incremental subgradient methods (Bertsekas, 1999; Cristianini and Shawe-Taylor, 2000) since only a subset of the elements of the subgradient is used in one iteration step. The first strategy requires time consuming computation of the full subgradient and a relatively large amount of memory. The second strategy, the coordinate descent algorithm, is efficient in the sense that it requires only a small computation effort in each step of the iteration; however, it improves the objective function very slowly. The third strategy, the block descent, is relatively faster and requires small storage requirement. In this work, we have used the block descent approach.

The user-specific blocks can be found in the second summation of the objective function (i.e. $\sum_{n\in\mathcal{I}_u} \alpha_{nu}\gamma^i_{nu}$ in equation 8.6). Note that one can make a descending step if this inequality

$$\min_{u\in\mathcal{U}_i} \sum_{n\in\mathcal{I}_u} \alpha_{nu}\gamma^i_{nu} < 1 \tag{8.8}$$

holds; otherwise, the corresponding ζ_i is zero and is not to be further decreased.

To this end, we assume that the index t counts the arriving data blocks in Algorithm 8 and the index k counts the updates of the variables when the condition (8.8) is fulfilled. The subgradient relating to the user block is equal to the vector $\gamma^i_{nu}(t),\ n \in \mathcal{I}_u$; thus the update with a fixed step size can be carried out by

$$\alpha_{nu}(t+1) = \alpha_{nu}(t) + s\gamma^i_{nu}(t),\ \forall n \in \mathcal{I}_u. \tag{8.9}$$

To make the algorithm more robust, a discounting factor-based averaging is carried out when the design variables of the underlying optimisation problem are updated. Refer to the steps in Algorithm 8 after line 30, where β is the discounting factor and is chosen from the open interval $(0, 1)$. The discounting can reduce the effect of the earlier observations on the most recent estimation of the variables, since at the beginning of

the algorithm the values of averages can have high variance caused by the small samples used to estimate them.

The variable β_k is used to normalise the accumulated values of the discounted variables. The subscript k refers to the number of updates of the design variable presented in Algorithm 8. This normalisation gives a convex combination of values of all updates of variables, where the weights diminish more if the update happened earlier. The current values of the β_k are computed by a recursive formula:

$$\beta_k = \begin{cases} 1 & \text{if } k = 0, \\ 1 + \beta\beta_{k-1} & \text{otherwise.} \end{cases}$$

Thus its accumulated value is equal to

$$\beta_k = \sum_{j=0}^{k} \beta^j. \tag{8.10}$$

Based on β_k the discounted value of the variables $\{\alpha_{iu}\}$ can be computed for a fixed pair of i and u. Let t_k be the index of the observation in update step k, then we can write up the following recursive formula for all $n \in \mathcal{I}_u$

$$\alpha_{nu}(t+1) = \frac{\beta\alpha_{nu}(t) + s\gamma_{nu}(t)}{\beta_k}|_{t=t_k}. \tag{8.11}$$

This update is applied whenever the constraint

$$\sum_{n\in\mathcal{I}_u} \alpha_{nu}(t)\gamma^i_{nu}(t) \geq 1 - \zeta_i(t) \tag{8.12}$$

is violated.

After updating the variables $\{\alpha_{nu}\}, n \in \mathcal{I}_u$, we can revise the estimation of the slack variable ζ_i used in step 30 of Algorithm 8. Since

$$\zeta_i(t) = \max\left(0, 1 - \min_{u\in\mathcal{U}_i} \sum_{n\in\mathcal{I}_u} \alpha_{nu}\gamma^i_{nu}(t)\right), \tag{8.13}$$

and assuming that

$$\min_{u\in\mathcal{U}_i} \sum_{n\in\mathcal{I}_u} \alpha_{nu}(t)\gamma^i_{nu}(t) < 1, \tag{8.14}$$

then we have

$$\zeta_i(t) = 1 - \min_{u\in\mathcal{U}_i} \sum_{n\in\mathcal{I}_u} \alpha_{nu}(t)\gamma^i_{nu}(t). \tag{8.15}$$

Now if

$$\sum_{n\in\mathcal{I}_u} \alpha_{nu}(t+1)\gamma^i_{nu}(t) > \min_{u\in\mathcal{U}_i} \sum_{n\in\mathcal{I}_u} \alpha_{nu}(t)\gamma^i_{nu}(t), \tag{8.16}$$

and (8.15) holds then we can decrease the value of the slack by

$$\zeta_i(t+1) = 1 - \sum_{n \in \mathcal{I}_u} \alpha_{nu}(t+1)\gamma^i_{nu}(t) < \zeta_i(t). \tag{8.17}$$

In this way, we decrease the value of the objective function in equation (8.3) directly, as shown in lines 37 and 38 in Algorithm 8.

8.3 Experimental Setup

To check the performance of the KMR^{incr} algorithm, we describe the results for two related scenarios: (1) when new users are introduced in the system, and (2) when new movies are introduced in the system. For the MovieLens (SML) dataset, we used the time information present in the dataset and sorted the users in the order in which they appear in the system (i.e. in the order in which they made ratings). We then used first X users ($X < M$) to train the base model, which we call $\mathcal{U}^{base}$ ($\mathcal{U}^{base} \subset \mathcal{U}$), and added the remaining users ($\mathcal{U}^{new} = \mathcal{U} \backslash \mathcal{U}^{base}$) to update the model. Similarly, we sorted the movies in the order in which they appear in the system (i.e. in the order in which they were rated by the users) and used first Y movies ($Y < N$) to train the base model, which we call $\mathcal{I}^{base}$ ($\mathcal{I}^{base} \subset \mathcal{I}$), and added the remaining movies ($\mathcal{I}^{new} = \mathcal{I} \backslash \mathcal{I}^{base}$) to update the model. We trained the base model using the optimal kernel parameters found using the validation set. For the FilmTrust (FT5) dataset, the test procedures were the same; however, we did not sort the users or movies as no time information was available against each rating. We conducted 5-fold cross validation and reported the average results.

To check the performance of the $KMR^{percept}$ algorithm, we built the model incrementally for $\mathcal{D}^{Percept} \subset \mathcal{D}$ data points, where 20% randomly selected data points from $\mathcal{D}^{Percept}$ were classified as the test set and the remaining as the training set. Again, we sorted the data points in $\mathcal{D}^{Percept}$ based on the time information for the MovieLens dataset.

In this way, we checked the performance of the algorithms by simulating the real world behaviour of recommender systems.

8.4 Results and Discussion

In the following, we denote the algorithm by KMR^{super}_{sub}, where the subscript denotes the variants of the algorithm, which can be item-based (ib) and user-based (ub). The superscript can be $full$ representing the baseline algorithm, where a full iterative model is used to build the model as presented in the previous chapter, $incr$ representing the proposed incremental algorithm, and $percept$ representing the proposed perceptron algorithm.

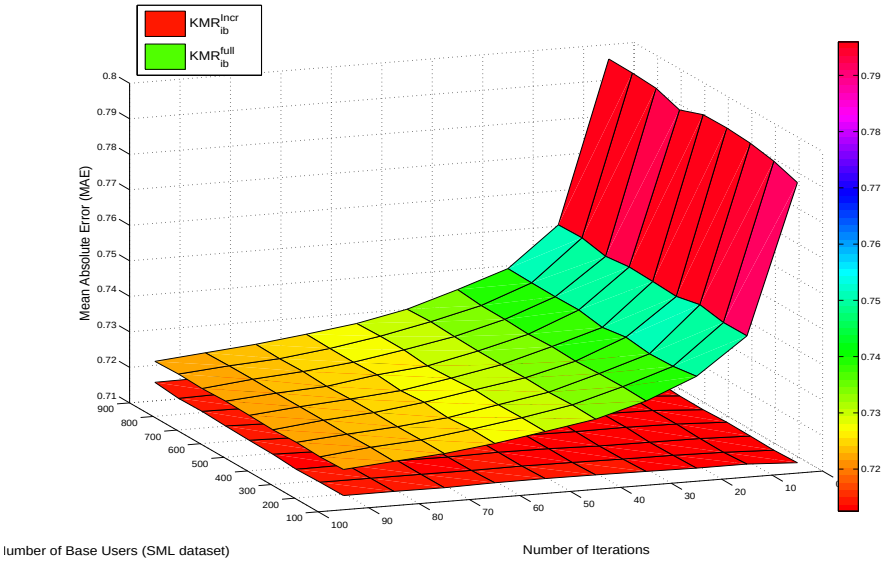


Figure 8.1: Comparing the performance of the proposed algorithm, KMR_{ib}^{incr}, with the baseline one, KMR_{ib}^{full}, for the MovieLens dataset, when new users are added in the system. "Number of Base Users" represents the number of users used to build the base model (i.e. $\mathcal{U}^{base}$). The model was updated by adding the remaining users (i.e. $\mathcal{U}^{new}$). "Number of Iterations" represents the number of iterations used to train the updated model.

8.4.1 Results of the KMR^{incr} algorithms

We compare our algorithm with the baseline algorithm where we retrain the model from scratch using some fixed number of iterations. The percentage decrease in the number of iterations required to update the model is calculated as $(itr_{full} - itr_{incr})/itr_{full}$, where itr_{full} and itr_{incr} represents the number of iterations used to update the model in the case of the baseline (KMR^{full}) and proposed (KMR^{incr}) algorithms respectively.

8.4.1.1 New users are added in the system

To check the behaviour of the proposed algorithm when new users enter the system, we performed a series of experiments by changing the base model size from 100 to 943 with a difference of 100 for the SML dataset and from 200 to 1 412 with a difference of 200 for the FT5 dataset. The base users, $\mathcal{U}^{base}$, are trained using optimal parameters. The remaining users ($\mathcal{U} \backslash \mathcal{U}^{base}$) are added afterwards in the system and the model was updated. For each experiment, we changed the number of iterations from 5 to 95 with a difference of 10, keeping the base model size fixed, and observed the corresponding MAE.

Figures 8.1 and 8.2 compare the performance of the proposed algorithm, KMR_{ib}^{incr}, with the baseline one, KMR_{ib}^{full} for the SML and FT5 datasets respectively. They show that

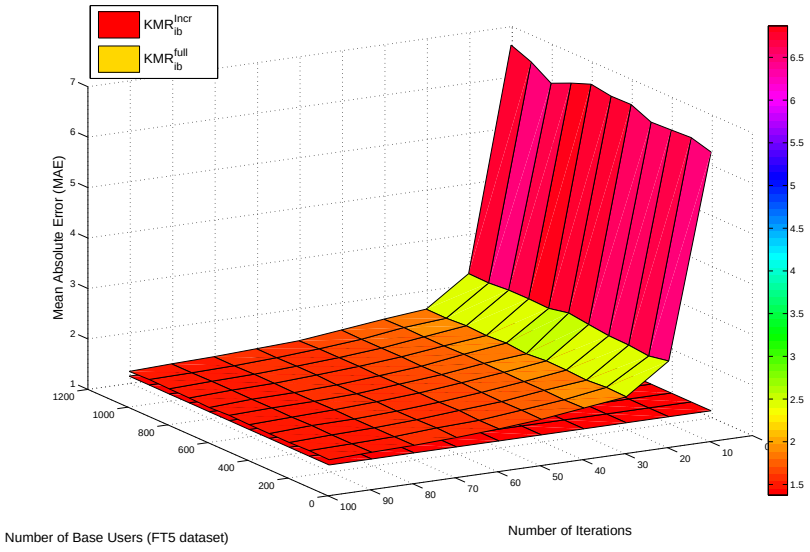


Figure 8.2: Comparing the performance of the proposed algorithm, KMR_{ib}^{incr}, with the baseline one, KMR_{ib}^{full}, for the FilmTrust dataset, when new users are added in the system. "Number of Base Users" represents the number of users used to build the base model (i.e. $\mathcal{U}^{base}$). The model was updated by adding the remaining users (i.e. $\mathcal{U}^{new}$). "Number of Iterations" represents the number of iterations used to train the updated model.

the proposed algorithm outperforms the baseline algorithm at every combination of base model size and number of iterations. We observe that for a small number of iterations, say less than 20, the baseline algorithm gives poor performance; however, the proposed algorithm maintains a good level of accuracy. Furthermore, the performance of the baseline algorithm degrades more in the case of the FilmTrust dataset compared to the MovieLens dataset. This is because, for the FilmTrust dataset, the baseline algorithm requires a significant number of iterations to converge and make reliable predictions (refer to previous chapter, Section 7.5.1). The percentage decrease in the MAE in the case of the proposed algorithm compared to the baseline one, keeping the number of base users and iterations fixed to 500 and 5 respectively, is found to be 6% for the SML and 70% for the FT5 dataset. Similar results were observed for the user-based KMR (refer to Appendix E).

Table 8.1 compares the performance of the proposed algorithm at a model size[2] of 500 with the baseline one. The table shows that the proposed algorithm gives better or comparable results to the baseline one. It must be noted that for the baseline algorithm shown in the table, the solution is found using the optimal number of iterations, which is expensive compared to the proposed algorithm. The percentage decrease in the number of iterations, required to update the model, in the case of the proposed algorithm compared to the baseline one is found to be 98% for both datasets.

[2]This model size is chosen as an example. Similar results were observed for the other sizes as well.

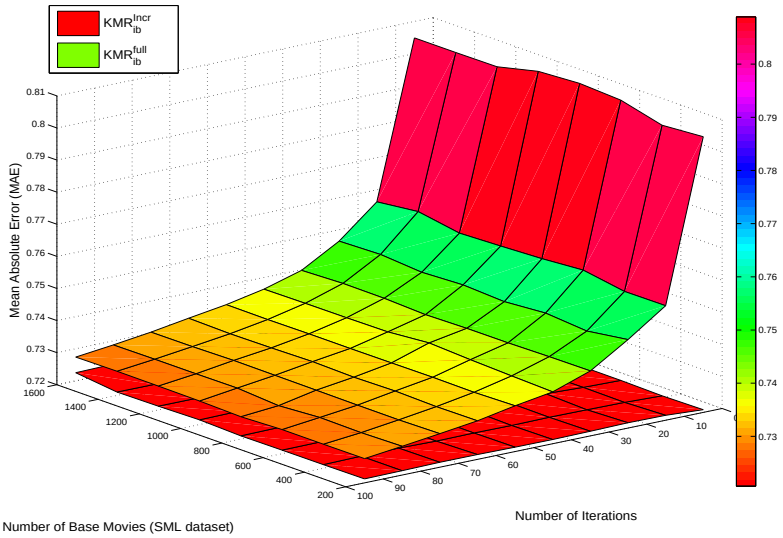


Figure 8.3: Comparing the performance of the proposed algorithm, KMR_{ib}^{incr}, with the baseline one, KMR_{ib}^{full}, for the MovieLens dataset, when new movies are added in the system. "Number of Base Movies" represents the number of movies used to build the base model (i.e. $\mathcal{I}^{base}$). The model was updated by adding the remaining movies (i.e. $\mathcal{I}^{new}$). "Number of Iterations" represents the number of iterations used to train the updated model.

8.4.1.2 New movies are added in the system

To check the behaviour of the proposed algorithm when new movies enter the system, we performed a series of experiments by changing the base model size from 200 to 1 682 with a difference of 200 for the SML dataset and from 50 to 314 with a difference of 50 for the FT5 dataset. The base movies, $\mathcal{I}^{base}$, were trained using the optimal parameters. The remaining movies ($\mathcal{I} \backslash \mathcal{I}^{base}$) were added afterwards in the system and the model was updated. For each experiment, we changed the number of iterations from 5 to 95 with a difference of 10, keeping the base model size fixed, and observed the corresponding MAE.

Figures 8.3 and 8.4 compare the performance of the proposed algorithm, KMR_{ib}^{incr}, with the baseline, KMR_{ib}^{full}, for the SML and FT5 datasets respectively. Again, we observe results similar to those discussed in Section 8.4.1.1. The percentage decrease in the MAE in case of the proposed algorithm compared to the baseline one, keeping the number of base movies and iterations fixed to 200 and 5 respectively, is found to be 6.6% for the SML and 58% for the FT5 dataset. Similar results were observed for the user-based KMR (refer to Appendix E).

Table 8.2 compares the performance of the proposed algorithm at a model size of 1 000 for the MovieLens dataset and 200 for the FilmTrust dataset with the baseline algorithm

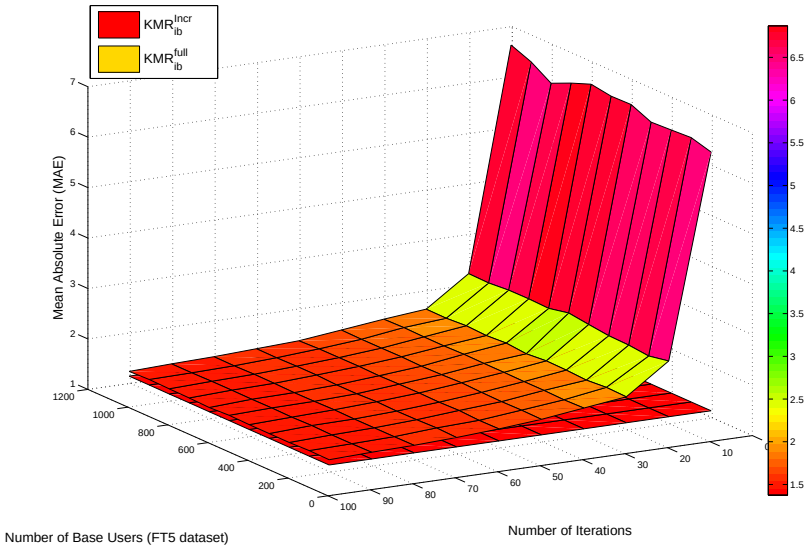


Figure 8.4: Comparing the performance of the proposed algorithm, KMR_{ib}^{incr}, with the baseline one, KMR_{ib}^{full}, for the FilmTrust dataset, when new movies are added in the system. "Number of Base Movies" represents the number of movies used to build the base model (i.e. $\mathcal{I}^{base}$). The model was updated by adding the remaining movies (i.e. $\mathcal{I}^{new}$). "Number of Iterations" represents the number of iterations used to train the updated model.

tuned using the optimal number of iterations. Again, we observe results similar to those in the new user case (Section 8.4.1.1).

8.4.2 Results of the $KMR^{percept}$ algorithms

We show results by building models at varying sizes (i.e. number of samples) of the datasets. Specifically, we used the following values of $\mathcal{D}^{percept}$: (1) {1 000, 2 000, 5 000, 10 000, 25 000, 50 000, 100 000 } for the SML dataset and (2) {1 000, 2 000, 5 000, 10 000, 15 000, 20 000, 25 730 } for the FT5 dataset. For each size, we used 80% of the samples for building the model and the remaining 20% for testing (refer to Section 8.3). We compare the performance of the proposed algorithm $KMR^{percept}$ with the baseline, KMR^{full} (trained using the optimal parameters) under varying dataset sizes. Figure 8.5 shows that the proposed algorithm gives MAE comparable to the baseline algorithm.

Table 8.3 compares the performance of $KMR^{percept}$ with KMR^{full} at sample size[3] of 10 000 (i.e. $\mathcal{D}^{percept} = 10\,000$). We observe that the proposed algorithm gives comparable results to the baseline one. Note that checking the performance of the proposed algorithm at different dataset sizes is analogous to cases where new users/movies are introduced in the system. Nevertheless, we performed experiments for the scenarios discussed in Section 8.3, and the results for the SML dataset are given in Appendix E.

[3]This value is taken as an example; however, similar results were observed for the other values as well.

Table 8.1: Comparing the performance of the proposed algorithm, KMR^{incr}, with the baseline one, KMR^{full}, at $|\mathcal{U}^{base}|$=500, when new users are added in the system. The performance of the proposed algorithm is better than (or comparable to) the baseline one.

Algorithm	**Itr**		**MAE**		**ROC-Sensitivity**		**F**1	
	FT5	**SML**	**FT5**	**SML**	**FT5**	**SML**	**FT5**	**SML**
KMR_{ib}^{incr}	5	5	1.379 ± 0.001	0.712 ± 0.000	0.627 ± 0.001	0.715 ± 0.000	0.533 ± 0.000	0.558 ± 0.000
KMR_{ib}^{full}	300	400	1.380 ± 0.000	0.713 ± 0.001	0.629 ± 0.001	0.715 ± 0.000	0.531 ± 0.001	0.564 ± 0.001
KMR_{ub}^{incr}	5	5	1.387 ± 0.001	0.735 ± 0.000	0.666 ± 0.001	0.725 ± 0.000	0.529 ± 0.000	0.595 ± 0.001
KMR_{ub}^{full}	300	400	1.384 ± 0.001	0.737 ± 0.001	0.652 ± 0.001	0.718 ± 0.000	0.524 ± 0.000	0.587 ± 0.001

Table 8.2: Comparing the performance of the proposed algorithm, KMR^{incr}, with the baseline one, KMR^{full}, at $|\mathcal{I}^{base}|$=1 000 for the SML dataset and $|\mathcal{I}^{base}|$=200 for the FT5 dataset, when new movies are added in the system. The performance of the proposed algorithm is comparable to the baseline one.

Algorithm	**Itr**		**MAE**		**ROC-Sensitivity**		**F**1	
	FT5	**SML**	**FT5**	**SML**	**FT5**	**SML**	**FT5**	**SML**
KMR_{ib}^{incr}	5	5	1.417 ± 0.005	0.721 ± 0.000	0.562 ± 0.002	0.683 ± 0.000	0.498 ± 0.001	0.514 ± 0.002
KMR_{ib}^{full}	300	400	1.381 ± 0.001	0.720 ± 0.000	0.628 ± 0.002	0.687 ± 0.001	0.504 ± 0.000	0.564 ± 0.002
KMR_{ub}^{incr}	5	5	1.397 ± 0.000	0.745 ± 0.000	0.592 ± 0.005	0.702 ± 0.001	0.506 ± 0.000	0.550 ± 0.003
KMR_{ub}^{full}	300	400	1.382 ± 0.000	0.742 ± 0.000	0.651 ± 0.001	0.705 ± 0.000	0.507 ± 0.000	0.588 ± 0.000

Table 8.3: Comparing the performance of the proposed algorithm, $KMR^{percept}$, with the baseline one, KMR^{full}, at a sample size of 10 000 ($\mathcal{D}^{Percept}$ = 10 000) for the SML and FT5 datasets. The performance of the proposed algorithm is comparable to the baseline one.

Algorithm	**MAE**		**ROC-Sensitivity**		**F**1	
	FT5	**SML**	**FT5**	**SML**	**FT5**	**SML**
KMR_{ib}^{full}	1.468 ± 0.004	0.826 ± 0.002	0.549 ± 0.002	0.607 ± 0.002	0.471 ± 0.002	0.512 ± 0.002
$KMR_{ib}^{percept}$	1.466 ± 0.004	0.828 ± 0.002	0.547 ± 0.002	0.599 ± 0.002	0.468 ± 0.002	0.510 ± 0.002
KMR_{ub}^{full}	1.480 ± 0.004	0.831 ± 0.002	0.613 ± 0.001	0.626 ± 0.002	0.472 ± 0.002	0.556 ± 0.002
$KMR_{ub}^{percept}$	1.479 ± 0.004	0.834 ± 0.002	0.596 ± 0.002	0.615 ± 0.003	0.467 ± 0.003	0.548 ± 0.002

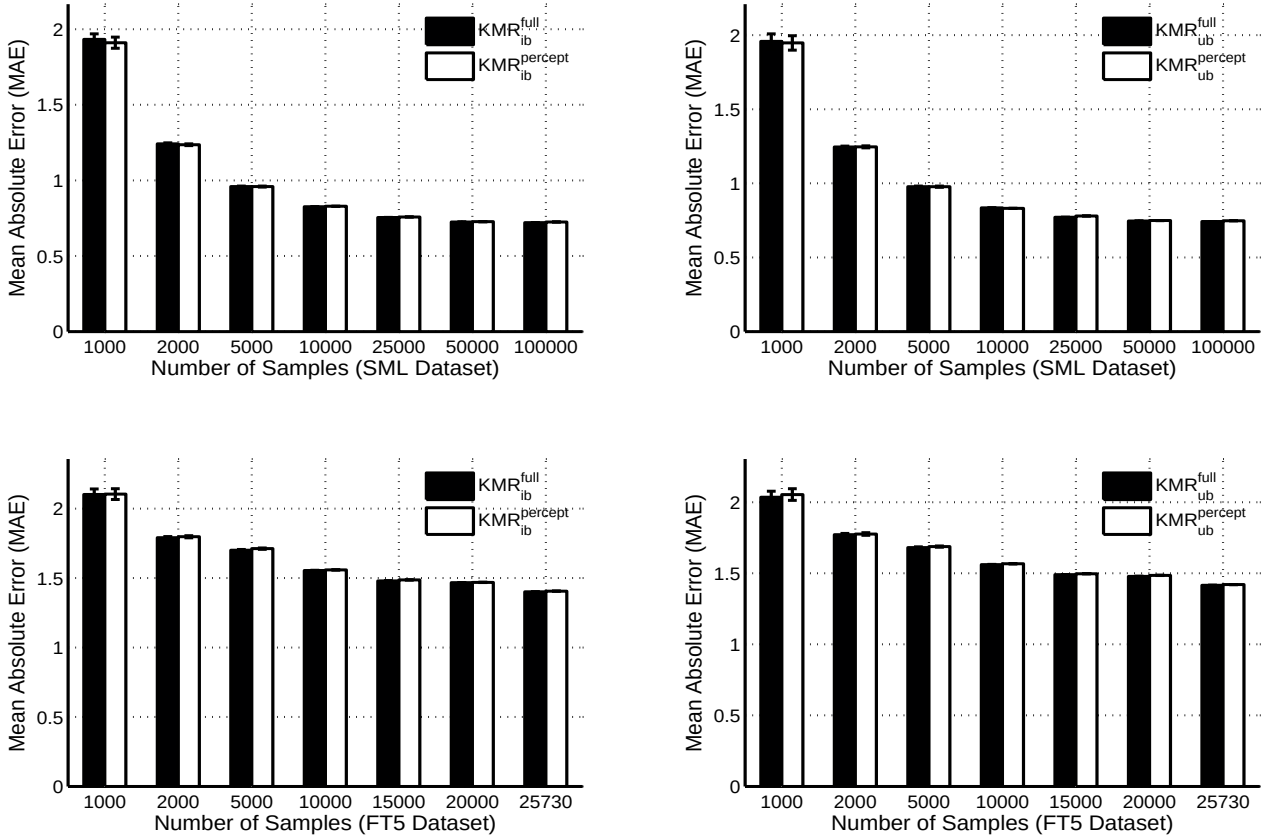


Figure 8.5: Comparing the performance of the proposed algorithm, $KMR^{percept}$, with the baseline one, KMR^{full}, under various values of $\mathcal{D}^{Percept}$. The baseline algorithm is trained using 400 and 300 iterations for the MovieLens and FilmTrust datasets respectively. "Number of Samples" represents the number of samples (i.e. data points) taken from the corresponding dataset to train the models. 80% of the total samples are used to train the model whereas the remaining samples are used for testing.

Table 8.4: Comparing the performance of the proposed algorithm, $KMR^{percept}$, with the baseline one, KMR^{full}, at a sample size of 400 000 ($\mathcal{D}^{Percept} = 400\,000$) for the Netflix dataset. The performance of the proposed algorithm is comparable to the baseline one.

Algorithm	**MAE**	**ROC-Sensitivity**	**F**1
KMR_{ib}^{full}	0.670 ± 0.001	0.649 ± 0.004	0.424 ± 0.003
$KMR_{ib}^{percept}$	0.681 ± 0.001	0.628 ± 0.005	0.418 ± 0.003
KMR_{ub}^{full}	0.677 ± 0.001	0.652 ± 0.004	0.429 ± 0.003
$KMR_{ub}^{percept}$	0.689 ± 0.001	0.644 ± 0.005	0.409 ± 0.003

To check the scalability of the proposed on-line algorithm, we performed experiments over the Netflix dataset. The results under varying sizes of $\mathcal{D}^{percept}$ showed similar behaviour to that shown by the MovieLens and the FilmTrust datasets. Table 8.4 compares the performance of the proposed $KMR^{percept}$ algorithm with KMR^{full} at a sample size of 400 000. Table 8.4 shows that the proposed algorithm gives comparable results to the baseline one.

The space complexity of the proposed algorithms is linear in the number of observations (i.e. user, movie, and rating). The complexity can be reduced if we use sparse structure for storage. The time complexity of the proposed algorithm is $O(nm)$, where $n < |\mathcal{U}|$ is the average number of ratings given to a movie and $m < |\mathcal{I}|$ is the average number of ratings by a user.

8.5 Conclusion

The Kernel Mapping Recommender (KMR) system algorithms introduced in the previous chapter are a new class of kernel-based methods for solving the recommendation problem that offer state-of-the-art performance. Although the KMR algorithms have the potential to build the model in the off-line stage, they have to recompute the model upon the arrival of new data, which is costly both in terms of computation time and storage. This problem makes this class of algorithms unsuitable for modern e-commerce systems where data are being added continuously in the system. In this chapter, we introduce two variants of KMR, namely KMR^{incr} and $KMR^{percept}$, that solve the aforementioned problem.

We have demonstrated by empirical results that the KMR^{incr} algorithm has the following important advantages:

- It can effectively incorporate the additional training data, when it is available, and maintains a good level of accuracy.
- It provides significant computation savings compared to the case where we retrain the model from scratch upon the arrival of new training data.

Likewise, we have shown that the $KMR^{percept}$ algorithm provides the following important advantages:

- It builds the model on-line by sequentially processing the dataset.
- It is able to scale well to large scale problems.
- It provides state-of-the-art performance.
- It can overcome some conventional problems with the collaborative filtering approaches; for instance, the stability vs. plasticity problem.

One plausible example of scenarios where our algorithms can be applied is Amazon's recommender engine. In this scenario, the data are huge, dynamic, and real time recommendations are required. $KMR^{percept}$ is well suited to this scenario.

Chapter 9

Conclusion and Future Work

9.1 Summary of the Work

The aim of this thesis has been to propose novel robust, scalable, and practical recommendation algorithms that can effectively be used to make accurate recommendations under different scenarios. We argue that although the current state-of-the-art algorithms (and especially the ones proposed in the Netflix prize competition) are quite accurate, they have certain drawbacks; for example, they get an increased accuracy rate by using a specific dataset's peculiar characteristics or by blending dozens (or hundreds in certain cases) of techniques trained on a static dataset. Furthermore, they ignore other design objectives, such as sparsity, cold-start, long tail, and dynamic updates, which makes them impractical for the real-world recommender system applications.

With this in mind, in the first half of the thesis, we proposed hybrid recommendation algorithms to overcome the conventional problems with the recommender systems. More specifically, we proposed a switching hybrid recommender system framework, which combines Collaborative Filtering (CF) with Content-Based Filtering (CBF). We showed empirically that the proposed framework produces accurate recommendations and moreover maintains good performance under the cold-start scenario. We also shed light on how Singular Value Decomposition (SVD)-based recommender systems are affected by the imputation methods used to approximate the missing values in the user-item rating matrix prior to applying SVD. We provided various imputation methods and empirically showed that they provide better recommendations than the literature approach under various recommendation scenarios. We also pointed out the gray-sheep users problem, associated with a recommender system, responsible for the increased error rate in the CF-based recommender systems. We demonstrated how the K-means clustering algorithm can be used to detect these users by treating them as outliers. We offered a hybrid recommender system to make recommendations and showed that the proposed approach

reduces the recommendation error rate for the gray-sheep users while maintaining reasonable computational performance.

In the second half of the thesis, we introduced a new class of kernel mapping recommender systems algorithms, namely KMR, based on the structure-learning. Comparing our algorithm with other current state-of-the-art algorithms, we observe that the performance of the proposed algorithms was better than (or comparable to) them. Experimental results on five different datasets support our claim. These experiments demonstrate the generality of our approach and the flexibility that makes this class of algorithm run on large-scale datasets. It is worth mentioning that the proposed algorithms give small improvement in results compared to the state-of-the-art algorithms; however, they are very practical and robust, and give consistently good performance for all datasets under all recommender system scenarios. Finally, we adapted the KMR algorithms to effectively incorporate the new arriving data and to build the model on-line. More specifically, we proposed a heuristic method, namely KMR^{incr}, that can incorporate additional data without retraining the whole model from scratch. We compared the performance of the KMR^{incr} with the KMR when new users and movies are added in the system and find out that the proposed method maintains good accuracy while providing significant computation savings. Furthermore, we introduced a novel on-line perceptron-like algorithm that we call $KMR^{percept}$, which can incrementally build the model by sequentially processing the data points. We show that the proposed algorithm is highly scalable and maintains a good level of accuracy. We provide the temporal analysis of the performance of the $KMR^{percept}$ algorithm.

An important lesson learned is that systematically combining different recommendation algorithms gives a robust performance. For example, in Chapter 4, a linear combination of CF and CBF did not give any improvement in the results; however, their non-linear combination increased the performance. The findings in Chapter 6 further support this idea. We propose to use a switching hybrid recommender system and the switching criterion is based on the type of user. Certain users do not get useful recommendations from a single algorithm and moreover their presence might negatively affect the recommendations of the rest of the community. Hence, it is beneficial to use different algorithms for different types of users. Furthermore, in Chapter 7, we note that a hybrid recommender system that switches between the user- and item-based KMR, depending on the reliability of the predictions as measured by the spread in the prediction of the algorithms, increases the performance of the system particularly under sparse and imbalanced datasets. Another finding is that different sources of information, e.g. rating, feature, and demographic, can help in cold-start, long tail, and sparse settings. We observe this behaviour especially in Chapter 7, where a linear or non-linear combination of different kernels (constructed using different sources of information) did not give any improvement in the results; however, they helped under cold-start, long tail, and sparse settings.

We believe that the KMR algorithms can effectively be used in domains, such as Amazon's and Netflix's recommender system. The proposed algorithms will bring considerable benefits to these systems. First, they provide high quality recommendations, which can increase the sale of a system. Second, they can overcome system start-up problems, where we have very few ratings and the resulting dataset is very sparse and imbalanced. Third, as they overcome cold-start and long tail problems, they would help the system in attracting more customers and recommending niche items. Finally, they can lead to significant savings, while updating the model upon the arrival of new data. Taking the results into account, we conclude that the algorithms proposed in this thesis make an important contribution towards improving current thinking on the subject.

9.2 Future Work

In future work, we would like to explore the following areas:

- **Combining the KMR with matrix factorisation techniques:** The KMR technique maps the vectors encoding information about the items (e.g. rating or text information) and the rating residual to vectors in some extended feature (Hilbert) space, where the main idea is to find the multi-linear mapping between these two vectors. The Matrix Factorisation (MF) technique is useful for finding the overall structure which is related to all (or most) of the users or items. Hence, these techniques deal with two different kinds of information—KMR deals with the local effects in the dataset, whereas the MF technique deals with the global (overall) effects in the dataset. We expect these two techniques to be complementary, and by capitalising the strengths of both we can increase the performance of the resulting system. Various approaches can be used to combine these two techniques. One simple approach is to measure the confidence in the prediction computed by the KMR (refer to Chapter 7, Section 7.4.2). We might switch to the MF technique when we have low confidence in the prediction computed by both the user- and item-based KMR.

- **Generalising the ranking, feedback, and adding context to KMR:** In certain cases, the ranks are not expressed by numbers but by a complex object; for example, an intelligent mobile phone, while scanning a user's activities, can observe the force of the touch, the speed and the direction, which give some clues to ranking. Moreover, the implicit feedback provided by users can be used; for example, individual downloads, viewing records, etc. One example of such a system is the MyExperiment (www.myexperiment.com) recommender system, which allows users to provide ratings and records their implicit feedback. Certain heuristics can be used for converting implicit feedback into explicit feedback. The users'

profiles can be represented in a highly complex way, where a separate kernel can be computed for each user action.

Furthermore, the context information can be employed. Adomavicius et al. (2005) provide various ways to add the context information in the recommendation process. We have shown in Appendix D how the context information can be added to the KMR algorithms; however, further experiments are needed to understand and adapt these algorithms for the context information.

- **Temporal shift of users' profiles:** We have claimed that the $KMR^{percept}$ algorithm can be used to overcome the stability vs. plasticity problem associated with a recommender system; however, further experiments are needed to analyse the behaviour of the algorithm. We would like to evaluate the performance of the $KMR^{percept}$ algorithm in scenarios where users' tastes are dynamic; for example, news personalisation (Das et al., 2007). In this domain, users can have short-term and long-term profiles and hence recommendations should be tailored according to the needs and interests of a particular user. The proposed $KMR^{percept}$ can be used to learn under this situation, where the earlier ratings provided by users can be deleted if their discount factors goes beyond a certain threshold. In this way the complexities can be reduced to a great extent as well. Furthermore, we can investigate better ways to set the value of the variable β. Ideally, a mechanism should be devised which can automatically set different values of β for different users depending on their rating behaviour. It might increase the complexity; however, it can potentially provide better recommendations.

- **Detecting gray-sheep users:** In this work, we have considered the gray-sheep users problem as an outlier detection problem, where the similarity between a user and the closest centroid is used to isolate the gray-sheep users. This similarity will vary with the number of clusters in the system. An important avenue for future research is to check the performance by relaxing the constraint of keeping a fixed number of centroids in applying the clustering algorithm. In the first iteration, a separate centroid can be made for a data point having a similarity (with existing centroids) less than a pre-defined threshold.

  In our work, the centroid selection approaches did not make any significant difference in increasing the performance of the K-means clustering algorithm. An alternative approach is to employ more than one user as the initial centroid, which might speed up the convergence rate of the K-means clustering algorithm. There are several issues to look at; for example, how will different users be selected as candidates for a centroid? What will be the initial size of each centroid? Should each centroid have the same size? Further experiments are needed to analyse this behaviour.

Another appealing area is to use the one-pass clustering algorithms (Bassam, 2010) to cluster the dataset. These algorithms will decrease the off-line cost of clustering the dataset; however, the accuracy of the resulting system might suffer.

9.3 Challenges in Practical Recommender Systems Algorithms

There are a number of challenges for practical recommender system algorithms. A few of these are discussed below:

- **Hybrid recommender systems:** We argue that different recommender systems taking into account different kinds of information, e.g. rating, demographic, feature, tags, social, etc. can be systematically combined to make reliable recommendations. The success of commercial recommender systems based on the hybrid technique; for example, Google news (Das et al., 2007), further supports this claim. It is an open research question for the recommender system's community to devise efficient mechanisms combining different recommender systems.

- **Scalable, flexible, and robust recommendation algorithms:** In order to exploit the recommender system algorithms effectively in the real-world recommender engines, the cost of model building should be small. Moreover, an algorithm should be designed to overcome potential problems, such as sparsity, cold-start, and long tail, in addition to the accuracy and scalability. It is an open challenge to the field to devise algorithms that are flexible and robust to give consistent performance under all recommender system scenarios.

- **Consistent performance over dynamic rather than static datasets:** Generally, while evaluating the performance of a recommendation algorithm, a random partition of a static dataset is used. The real-world recommender systems; however, have different characteristics, i.e. imbalanced dataset where dynamic updates occur frequently. Furthermore, users' tastes can change over time. The datasets describing such scenarios should be standardised and the performance of a recommendation algorithm should be evaluated over such datasets.

Appendix A

Switching Hybrid Recommender Systems

A.1 Rating Distribution of the FilmTrust Dataset

Figure A.1 shows the rating distribution of the FilmTrust dataset. We observe that there are total 8 classes against which the users have provided the ratings.

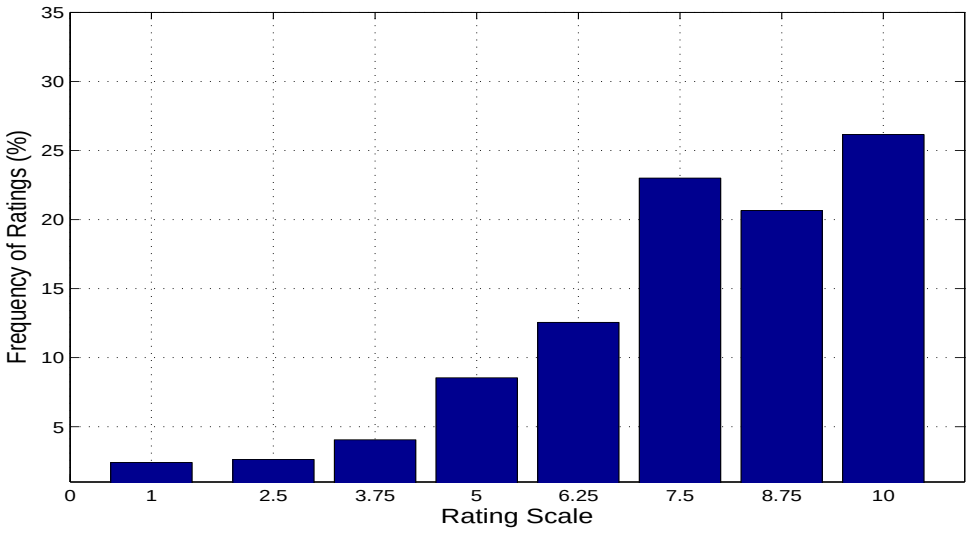


Figure A.1: Rating distribution of the FilmTrust dataset.

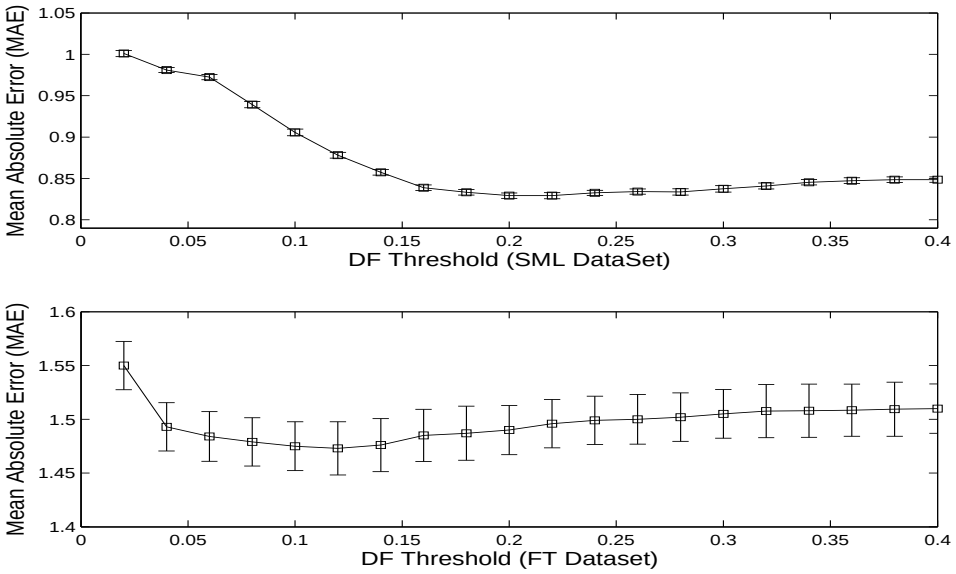


Figure A.2: Finding the optimal value of DF threshold for the Naive Bayes classifier over the validation set.

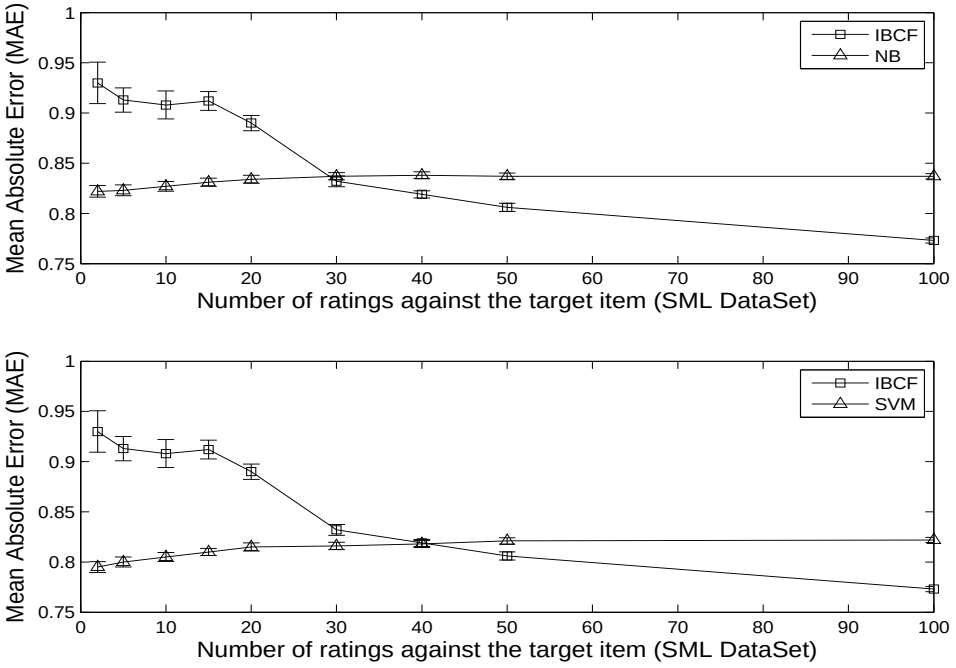


Figure A.3: Finding the optimal value of parameter ν over the validation set (SML dataset). X-axis represents the number of users who have rated the target item.

A.2 Learning the Optimal System Parameters

A.2.1 Finding the optimal value of DF thresholding

To determine the optimal value of the Document Frequency (DF) thresholding[1] (refer to Section 3.6.3), we varied the value of DF from 0.02 to 0.4 with a difference of 0.02. The results for the Naive Bayes classifier are shown in Figure A.2. Figure A.2 shows that $DF = 0.20$ and $DF = 0.12$ gave the lowest MAE for the MovieLens and FilmTrust dataset respectively. We choose these values of DF threshold for the subsequent experiments.

To determine the optimal value of DF threshold for the SVM classifier, we repeated the same experiment. The results (not shown) did not show any improvement in the results, hence we did not perform any feature selection for the SVM classifier.

A.2.2 Learning the optimal value for parameter ν

Figure A.3 shows how the MAE changes with the available number of ratings for the target item in the case of SML dataset. We observe that the CF fails to produce good results when we have fewer users who have rated the target item, whereas; the performance of the Naive Bayes and the SVM classifier does not suffer. Taking these results into account, we choose $\nu = 30$ and $\nu = 40$ in the case of $SwitchRec_{CF}^{NB}$ and $SwitchRec_{CF}^{SVM}$ respectively. Similarly, we tunned the optimal value of ν for the FilmTrust dataset, which are found to be 20 and 25 in the case of $SwitchRec_{CF}^{NB}$ and $SwitchRec_{CF}^{SVM}$ respectively.

A.3 Implementation

We used the Weka collection of machine learning algorithms (Hall et al., 2009) for building the Naive Bayes classifier. For the SVM classifier, we normalised the data in the scale of $0 - 1$ and used LibSVM (Chang and Lin, 2011) for binary classification. We used the Lucene (`lucene.apache.org/core/`) for text processing tasks (e.g. Stemming). We reused and extended the code from Carleton College, Northfield, MN (`http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/index.html`) for building the Collaborative Filtering algorithms (and clustering algorithms in Chapter 6). Java (Jre 6) language was used for implementation. A copy of the code can be obtained on request (eng.musi@gmail.com).

[1] We also experimented with χ^2 feature selection method; however, the DF thresholding gave us better results.

Appendix B

Imputation in SVD-based Recommender Systems

B.1 Learning the Optimal System Parameters

- *K Nearest Neighbour (KNN)*: The optimal number of neighbour are found to be for $\{KNN, WKNN\}$ (1) $\{300, 250\}$ for the SML dataset, (2) $\{150, 150\}$ for the FT1 dataset, and (3) $\{150, 100\}$ for the FT5 dataset.

- *Decision tree (C4.5)*: The pruning confidence was found to be 0.9 for the SML dataset and 0.5 for the FT dataset. We used Laplace smoothing for predicted probabilities.

- *SVM regression (SVMReg)*: The optimal value of the cost parameters C is found to be 2, 1, and 1 for the SML, FT1, and FT5 datasets respectively. Furthermore, the loss parameter (nu) is found to be 0.9 for the SML and 0.2 for the FT dataset.

- *Linear and logistic regression (LinearReg and LogisticReg)*: We found the parameter tuning in linear and logistic regression very expensive and hence we used the default parameters, given in the Weka library, for these methods.

- *Finding the optimal number of neighbours for the user-based CF (UBCF)*: The optimal number of neighbours are found to be 90, 60, 30, 25 for the ML, SML, FT1, and FT5 datasets respectively.

- *Finding the optimal values of neighbours for the item-based CF (IBCF)*: The optimal number of neighbours are found to be 35, 25, 15, 10 for the ML, SML, FT1, and FT5 datasets respectively.

- *Finding the optimal values of parameters α and β*: Parameters α and β determine the relative weights of user-based and item-based CF in the final prediction. The 9 parameter sets were generated by producing all possible combination of parameters

Table B.1: Learning parameter sets α and β over the validation set through cross validation. α and β show the relative impact of user- and item-based CF in a prediction respectively.

Parameters		**MAE**			
α	β	**ML**	**SML**	**FT**1	**FT**5
0.1	0.9	0.7101	0.7407	1.4836	1.4499
0.2	0.8	0.7098	0.7398	1.4763	1.4408
0.3	0.7	0.7086	0.7386	1.4717	1.4344
0.4	0.6	**0.7060**	**0.7362**	**1.4701**	1.4308
0.5	0.5	0.7076	0.7376	1.4715	**1.4295**
0.6	0.4	0.7082	0.7379	1.4752	1.4312
0.7	0.3	0.7078	0.7386	1.4810	1.4355
0.8	0.2	0.7097	0.7398	1.4891	1.4429
0.9	0.1	0.7121	0.7414	1.4995	1.4522

values, ranging from 0.1 to 1.0 with differences of 0.1. Table B.1 presents the parameter sets learned. The parameters sets $\alpha = 0.4, \beta = 0.6$; $\alpha = 0.4, \beta = 0.6$; $\alpha = 0.4, \beta = 0.6$; and $\alpha = 0.5, \beta = 0.5$ gave the lowest *MAE* in the case of ML, SML, FT1 and FT5 dataset respectively. It is worth noting that the values of parameters are found different for the MovieLens and FilmTrust dataset. We note that the item-based CF has more weight in the final prediction.

B.2 Performance Evaluation of the *ImpSvd* in Terms of ROC-Sensitivity and Top-N Metrics

Tables B.2, B.3, B.4, and B.5 compare the performance—in terms of ROC-sensitivity, F1, precision, and recall—of different approaches in the *ImpSvd* algorithm. The performance improvement of the proposed approaches over the baseline one is found to be 5% to 10%.

B.3 Performance Evaluation of the *ItrSvd* in Terms of ROC-Sensitivity and Top-N Metrics

Tables B.6, B.7, B.8, and B.9 compare the performance—in terms of ROC-sensitivity, F1, precision, and recall—of different approaches in the *ItrSvd*. We observe that the performance of the proposed approaches is better than the baseline one.

Table B.2: The ROC-Sensitivity observed in different imputation methods in the *ImpSvd* algorithm. The optimal number of dimensions have been kept the same as shown in Table 5.1. The best results are shown in bold font.

Imp. Method	**Best ROC-sensitivity**			
	ML	**SML**	**FT**1	**FT**5
Zeros	0.048 ± 0.002	0.017 ± 0.001	0.004 ± 0.002	0.005 ± 0.001
Rand	0.051 ± 0.002	0.056 ± 0.001	0.048 ± 0.003	0.045 ± 0.002
ItemAvg	0.634 ± 0.002	0.607 ± 0.002	0.472 ± 0.004	0.469 ± 0.012
UserAvg	0.732 ± 0.001	0.691 ± 0.002	0.560 ± 0.004	0.555 ± 0.011
UserItemAvg	0.652 ± 0.002	0.620 ± 0.002	0.477 ± 0.004	0.491 ± 0.012
UniformDist	0.131 ± 0.002	0.167 ± 0.002	0.065 ± 0.040	0.071 ± 0.003
Nor_U	0.651 ± 0.002	0.585 ± 0.010	0.471 ± 0.001	0.482 ± 0.010
Nor_I	0.611 ± 0.002	0.588 ± 0.002	0.438 ± 0.003	0.452 ± 0.013
UBCF	0.741 ± 0.001	0.696 ± 0.002	0.548 ± 0.011	0.539 ± 0.003
IBCF	$\mathbf{0.801 \pm 0.001}$	0.722 ± 0.011	0.541 ± 0.011	0.549 ± 0.003
UBIBCF	0.791 ± 0.001	0.661 ± 0.002	0.540 ± 0.004	0.555 ± 0.004
KNN	$--$	0.730 ± 0.002	0.498 ± 0.012	0.512 ± 0.013
WKNN	$--$	$\mathbf{0.739 \pm 0.002}$	0.498 ± 0.012	0.512 ± 0.013
NBClass	$--$	0.731 ± 0.002	0.512 ± 0.013	0.528 ± 0.013
SVMClass	$--$	$\mathbf{0.739 \pm 0.002}$	0.520 ± 0.014	0.539 ± 0.014
C4.5	$--$	0.723 ± 0.002	0.502 ± 0.014	0.514 ± 0.014
SVMReg	$--$	0.685 ± 0.002	$\mathbf{0.566 \pm 0.013}$	$\mathbf{0.567 \pm 0.014}$
LinearReg	$--$	0.662 ± 0.002	0.531 ± 0.012	0.542 ± 0.014
LogisticReg	$--$	0.667 ± 0.002	0.532 ± 0.013	0.540 ± 0.014

Table B.3: The F1 observed in different imputation methods in the *ImpSvd* algorithm. The optimal number of dimensions have been kept the same as shown in Table 5.1. The best results are shown in bold font.

Imp. Method	**Best F**1			
	ML	**SML**	**FT**1	**FT**5
Zeros	0.001 ± 0.004	0.002 ± 0.002	0.017 ± 0.002	0.009 ± 0.002
Rand	0.010 ± 0.001	0.018 ± 0.001	0.047 ± 0.004	0.045 ± 0.002
ItemAvg	0.407 ± 0.003	0.441 ± 0.002	0.444 ± 0.004	0.445 ± 0.003
UserAvg	0.471 ± 0.002	0.512 ± 0.004	0.534 ± 0.011	0.529 ± 0.014
UserItemAvg	0.419 ± 0.002	0.453 ± 0.004	0.454 ± 0.012	0.475 ± 0.003
UniformDist	0.053 ± 0.001	0.091 ± 0.001	0.060 ± 0.003	0.067 ± 0.003
Nor_U	0.407 ± 0.002	0.429 ± 0.011	0.451 ± 0.012	0.467 ± 0.004
Nor_I	0.398 ± 0.002	0.430 ± 0.002	0.395 ± 0.003	0.414 ± 0.013
UBCF	0.478 ± 0.002	0.515 ± 0.002	0.530 ± 0.011	0.527 ± 0.002
IBCF	$\mathbf{0.513 \pm 0.002}$	$\mathbf{0.537 \pm 0.003}$	0.517 ± 0.012	0.523 ± 0.012
UBIBCF	0.506 ± 0.002	0.491 ± 0.001	0.534 ± 0.004	0.543 ± 0.012
KNN	$--$	0.492 ± 0.003	0.502 ± 0.012	0.508 ± 0.016
NBClass	$--$	0.508 ± 0.003	0.514 ± 0.014	0.518 ± 0.015
SVMClass	$--$	0.511 ± 0.003	0.521 ± 0.012	0.531 ± 0.013
C4.5	$--$	0.492 ± 0.003	0.499 ± 0.011	0.506 ± 0.012
SVMReg	$--$	0.492 ± 0.003	$\mathbf{0.541 \pm 0.013}$	$\mathbf{0.544 \pm 0.014}$
LinearReg	$--$	0.483 ± 0.003	0.512 ± 0.012	0.519 ± 0.013
LogisticReg	$--$	0.485 ± 0.003	0.521 ± 0.012	0.522 ± 0.013

Table B.4: The precision observed in different imputation methods in the *ImpSvd* algorithm. The optimal number of dimensions have been kept the same as shown in Table 5.1. The best results are shown in bold font.

Imp. Method	**Best Precision**			
	ML	**SML**	**FT**1	**FT**5
Zeros	0.005 ± 0.001	0.009 ± 0.001	0.049 ± 0.003	0.025 ± 0.005
Rand	0.029 ± 0.001	0.030 ± 0.001	0.072 ± 0.001	0.074 ± 0.004
ItemAvg	0.468 ± 0.002	0.504 ± 0.004	0.495 ± 0.004	0.502 ± 0.004
UserAvg	0.492 ± 0.002	0.549 ± 0.004	0.569 ± 0.004	0.567 ± 0.006
UserItemAvg	0.480 ± 0.002	0.514 ± 0.002	0.505 ± 0.004	0.542 ± 0.003
UniformDist	0.075 ± 0.002	0.119 ± 0.003	0.116 ± 0.004	0.097 ± 0.003
Nor_U	0.447 ± 0.002	0.483 ± 0.012	0.493 ± 0.004	0.516 ± 0.006
Nor_I	0.455 ± 0.002	0.485 ± 0.003	0.425 ± 0.004	0.459 ± 0.006
UBCF	0.498 ± 0.002	0.549 ± 0.004	0.574 ± 0.005	0.584 ± 0.005
IBCF	0.506 ± 0.002	$\mathbf{0.560 \pm 0.003}$	0.566 ± 0.005	0.526 ± 0.004
UBIBCF	$\mathbf{0.507 \pm 0.002}$	0.535 ± 0.003	0.568 ± 0.004	0.573 ± 0.005
KNN	$--$	0.441 ± 0.005	0.514 ± 0.006	0.519 ± 0.006
WKNN	$--$	0.452 ± 0.005	0.525 ± 0.006	0.528 ± 0.006
NBClass	$--$	0.501 ± 0.004	0.532 ± 0.006	0.544 ± 0.006
SVMClass	$--$	0.503 ± 0.003	0.546 ± 0.005	0.552 ± 0.006
C4.5	$--$	0.492 ± 0.004	0.518 ± 0.005	0.524 ± 0.005
SVMReg	$--$	0.531 ± 0.004	$\mathbf{0.580 \pm 0.005}$	$\mathbf{0.594 \pm 0.006}$
LinearReg	$--$	0.509 ± 0.004	0.543 ± 0.005	0.553 ± 0.005
LogisticReg	$--$	0.512 ± 0.004	0.544 ± 0.005	0.553 ± 0.005
AdaBoost	$--$	0.506 ± 0.004	0.529 ± 0.006	0.541 ± 0.006

B.4 Performance Evaluation of *ImpSvd* Under Different Training and Test Sizes

We performed experiments with different sizes of the test and training set by randomly dividing the rating records into $X\%$ training set and $(100 - X)\%$ test set. A value of $X = 20\%$ for the SML dataset indicates that 100 000 ratings have been divided into 20 000 training cases and 80 000 test cases. Table B.10 shows that the proposed approaches outperform others at each value of X. We note that the SVMReg gives the best performance for smaller training set sizes. The reason is the same as discussed in Section 5.6.5.

B.5 Performance Evaluation of *ImpSvd* Under Cold-Start and Long Tail Scenarios

For testing the performance of approaches under new user cold-start scenario, we selected 100 random users, and kept their number of ratings in the training set to 2, 5, 10, 15, and 20. The corresponding *MAE*, represented by *MAE*2, *MAE*5, *MAE*10, *MAE*15, and

Table B.5: The recall observed in different imputation methods in the *ImpSvd* algorithm. The optimal number of dimensions have been kept the same as shown in Table 5.1. The best results are shown in bold font.

Imp. Method	**Best Recall**			
	ML	**SML**	**FT**1	**FT**5
Zeros	0.006 ± 0.004	0.002 ± 0.003	0.011 ± 0.001	0.009 ± 0.002
Rand	0.019 ± 0.001	0.020 ± 0.001	0.050 ± 0.002	0.045 ± 0.002
ItemAvg	0.043 ± 0.002	0.473 ± 0.002	0.471 ± 0.004	0.468 ± 0.005
UserAvg	0.510 ± 0.002	0.545 ± 0.003	0.560 ± 0.004	0.554 ± 0.006
UserItemAvg	0.490 ± 0.002	0.481 ± 0.002	0.474 ± 0.004	0.491 ± 0.004
UniformDist	0.069 ± 0.002	0.102 ± 0.002	0.080 ± 0.003	0.071 ± 0.003
Nor_U	0.434 ± 0.002	0.448 ± 0.003	0.470 ± 0.005	0.481 ± 0.004
Nor_I	0.426 ± 0.002	0.467 ± 0.001	0.434 ± 0.003	0.452 ± 0.006
UBCF	0.528 ± 0.002	0.555 ± 0.002	0.548 ± 0.005	0.534 ± 0.003
IBCF	$\mathbf{0.594 \pm 0.002}$	0.592 ± 0.005	0.553 ± 0.005	0.550 ± 0.001
UBIBCF	0.577 ± 0.002	0.520 ± 0.002	0.537 ± 0.004	0.555 ± 0.004
KNN	$--$	0.623 ± 0.004	0.482 ± 0.006	0.491 ± 0.006
WKNN	$--$	$\mathbf{0.653 \pm 0.004}$	0.491 ± 0.006	0.501 ± 0.006
NBClass	$--$	0.585 ± 0.004	0.512 ± 0.005	0.518 ± 0.007
SVMClass	$--$	0.616 ± 0.003	0.524 ± 0.005	0.531 ± 0.006
C4.5	$--$	0.534 ± 0.003	0.501 ± 0.005	0.508 ± 0.006
SVMReg	$--$	0.528 ± 0.003	$\mathbf{0.581 \pm 0.005}$	$\mathbf{0.576 \pm 0.007}$
LinearReg	$--$	0.512 ± 0.003	0.551 ± 0.005	0.544 ± 0.007
LogisticReg	$--$	0.515 ± 0.003	0.553 ± 0.005	0.545 ± 0.007
AdaBoost	$--$	0.551 ± 0.004	0.522 ± 0.005	0.530 ± 0.007

Table B.6: Comparing the ROC-sensitivity observed in different imputation methods in the *ItrSvd* (fixed iteration case). The best results are shown in bold font.

Imp. Method	**Best ROC-sensitivity**			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.685 ± 0.001	0.651 ± 0.002	0.504 ± 0.006	0.569 ± 0.005
UserAvg	0.721 ± 0.001	0.683 ± 0.004	0.572 ± 0.005	0.571 ± 0.006
UBCF	0.724 ± 0.001	0.691 ± 0.002	0.546 ± 0.005	0.530 ± 0.005
IBCF	$\mathbf{0.759 \pm 0.001}$	$\mathbf{0.724 \pm 0.006}$	0.534 ± 0.005	0.544 ± 0.006
UBIBCF	0.747 ± 0.001	0.711 ± 0.002	0.517 ± 0.005	0.563 ± 0.004
SVMReg	$--$	0.695 ± 0.003	$\mathbf{0.574 \pm 0.006}$	$\mathbf{0.583 \pm 0.006}$

Table B.7: Comparing the F1 observed in different imputation methods in the *ItrSvd* (fixed iteration case). The best results are shown in bold font.

Imp. Method	**Best F**1			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.445 ± 0.002	0.481 ± 0.002	0.486 ± 0.005	0.547 ± 0.004
UserAvg	0.463 ± 0.002	0.503 ± 0.003	0.540 ± 0.005	0.538 ± 0.006
UBCF	0.468 ± 0.002	0.514 ± 0.002	0.531 ± 0.005	0.520 ± 0.004
IBCF	$\mathbf{0.487 \pm 0.002}$	$\mathbf{0.531 \pm 0.003}$	0.505 ± 0.005	0.519 ± 0.005
UBIBCF	0.481 ± 0.002	0.528 ± 0.001	0.507 ± 0.005	0.534 ± 0.004
SVMReg	$--$	0.508 ± 0.003	$\mathbf{0.556 \pm 0.005}$	$\mathbf{0.563 \pm 0.006}$

Table B.8: Comparing the precision observed in different imputation methods in the *ItrSvd* (fixed iteration case). The best results are shown in bold font.

Imp. Method	**Best Precision**			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.502 ± 0.002	0.548 ± 0.004	0.533 ± 0.005	0.591 ± 0.004
UserAvg	0.493 ± 0.002	0.547 ± 0.003	0.571 ± 0.005	0.571 ± 0.006
UBCF	0.502 ± 0.002	0.556 ± 0.003	0.578 ± 0.005	0.576 ± 0.005
IBCF	0.501 ± 0.002	0.551 ± 0.003	0.542 ± 0.005	0.565 ± 0.003
UBIBCF	$\mathbf{0.503 \pm 0.002}$	$\mathbf{0.557 \pm 0.003}$	0.578 ± 0.005	$\mathbf{0.595 \pm 0.005}$
SVMReg	$--$	0.551 ± 0.004	$\mathbf{0.586 \pm 0.006}$	0.589 ± 0.006

Table B.9: Comparing the recall observed in different imputation methods in the *ItrSvd* (fixed iteration case). The best results are shown in bold font.

Imp. Method	**Best Recall**			
	ML	**SML**	**FT**1	**FT**5
ItemAvg	0.454 ± 0.002	0.502 ± 0.002	0.502 ± 0.005	0.569 ± 0.005
UserAvg	0.493 ± 0.002	0.531 ± 0.003	0.570 ± 0.005	0.564 ± 0.005
UBCF	0.496 ± 0.002	0.546 ± 0.003	0.543 ± 0.003	0.532 ± 0.005
IBCF	$\mathbf{0.532 \pm 0.002}$	$\mathbf{0.580 \pm 0.005}$	0.532 ± 0.005	0.543 ± 0.007
UBIBCF	0.520 ± 0.002	0.565 ± 0.002	0.514 ± 0.005	0.551 ± 0.004
SVMReg	$--$	0.537 ± 0.0074	$\mathbf{0.588 \pm 0.006}$	$\mathbf{0.590 \pm 0.006}$

Table B.10: Comparing the MAE observed in different imputation methods under **varying training set sizes**, for the SML dataset. The best results are shown in bold font.

Imp. Method	**Best MAE**			
	$X = 20\%$	$X = 40\%$	$X = 60\%$	$X = 80\%$
ItemAvg	0.838 ± 0.002	0.809 ± 0.002	0.788 ± 0.002	0.774 ± 0.001
UserAvg	0.839 ± 0.002	0.818 ± 0.002	0.792 ± 0.002	0.778 ± 0.001
UserItemAvg	0.798 ± 0.002	0.784 ± 0.002	0.767 ± 0.002	0.754 ± 0.001
UBCF	0.807 ± 0.002	0.766 ± 0.002	0.746 ± 0.003	0.732 ± 0.001
IBCF	0.804 ± 0.002	0.762 ± 0.002	0.740 ± 0.003	0.730 ± 0.001
UBIBCF	0.802 ± 0.002	0.760 ± 0.002	$\mathbf{0.733 \pm 0.003}$	$\mathbf{0.721 \pm 0.001}$
SVMReg	$\mathbf{0.796 \pm 0.002}$	$\mathbf{0.756 \pm 0.003}$	0.748 ± 0.003	0.736 ± 0.001

*MAE*20 is shown in Table B.11. Table B.11 shows that the conventional approaches suffer the most under this scenario. It is worth noting that, when a user has rated less than (or equal to) 10 movies, then UserItemAvg gives the best results; however, as a user rates more items, the UBIBCF gives reliable recommendations.

To test the performance of the proposed algorithms under long tail scenario, we created the artificial long tail scenario by randomly selecting the 80% of items in the tail. The number of ratings given in the tail part were varied between 2, 4, 6, 8, 10 and 15. The results, shown in Table B.12, demonstrated the similar behaviour as in the case of new item case (refer to Chapter 5, Section 5.6.6).

Table B.11: Comparing the MAE observed in different imputation methods under the **new user cold-start scenario**, for the SML dataset. The best results are shown in bold font.

Imp. Method	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
ItemAvg	0.908 ± 0.002	0.887 ± 0.002	0.885 ± 0.002	0.883 ± 0.002	0.882 ± 0.002
UserAvg	1.087 ± 0.002	0.928 ± 0.002	0.903 ± 0.002	0.878 ± 0.002	0.877 ± 0.002
UserItemAvg	$\mathbf{0.901 \pm 0.002}$	$\mathbf{0.855 \pm 0.002}$	$\mathbf{0.850 \pm 0.002}$	0.843 ± 0.002	0.839 ± 0.002
UBCF	1.080 ± 0.002	0.886 ± 0.002	0.865 ± 0.002	0.841 ± 0.002	0.825 ± 0.002
IBCF	1.082 ± 0.002	0.896 ± 0.002	0.868 ± 0.002	0.844 ± 0.002	0.817 ± 0.002
UBIBCF	1.071 ± 0.002	0.891 ± 0.002	0.862 ± 0.002	$\mathbf{0.837 \pm 0.002}$	$\mathbf{0.816 \pm 0.002}$
SVMReg	0.962 ± 0.002	0.912 ± 0.002	0.873 ± 0.002	0.841 ± 0.042	0.836 ± 0.002

Table B.12: Comparing the MAE observed in different imputation methods under the **long tail scenario**, for the SML dataset. The best results are shown in bold font.

Imp. Method	**Best MAE**					
	MAE2	**MAE**4	**MAE**6	**MAE**8	**MAE**10	**MAE**15
ItemAvg	1.090 ± 0.003	0.891 ± 0.003	0.879 ± 0.003	0.867 ± 0.003	0.861 ± 0.003	0.853 ± 0.002
UserAvg	0.881 ± 0.003	0.878 ± 0.003	0.869 ± 0.003	0.866 ± 0.003	0.865 ± 0.002	0.865 ± 0.002
UserItemAvg	0.884 ± 0.003	0.882 ± 0.003	0.871 ± 0.003	0.863 ± 0.003	0.861 ± 0.003	0.858 ± 0.002
UBCF	0.881 ± 0.003	0.874 ± 0.003	0.847 ± 0.003	0.838 ± 0.003	0.819 ± 0.002	$\mathbf{0.814 \pm 0.002}$
IBCF	0.886 ± 0.003	0.875 ± 0.003	0.861 ± 0.003	0.860 ± 0.003	0.856 ± 0.003	0.842 ± 0.002
UBIBCF	0.882 ± 0.003	0.869 ± 0.003	0.844 ± 0.003	0.836 ± 0.003	0.824 ± 0.002	0.820 ± 0.002
SVMReg	$\mathbf{0.879 \pm 0.002}$	$\mathbf{0.865 \pm 0.002}$	$\mathbf{0.842 \pm 0.002}$	$\mathbf{0.833 \pm 0.002}$	$\mathbf{0.817 \pm 0.002}$	0.815 ± 0.002

B.6 Implementation

We used the Weka collection of machine learning algorithms (Hall et al., 2009) for building the classification and regression algorithms. For the SVM classifier, we normalised the data in the scale of $0-1$ and used LibSVM (Chang and Lin, 2011) for binary classification. We used the Colt library (`acs.lbl.gov/software/colt/`) for computing the SVD. Java (Jre 6) language was used for implementation. A copy of the code can be obtained on request (eng.musi@gmail.com).

Appendix C

Using K-Means Clustering Algorithms to Solve the Gray-Sheep Users Problem

C.1 Learning the Optimal System Parameters

C.1.1 Optimal Number of Clusters

For finding the optimal number of clusters, we changed the cluster size from 10 to 200 with a difference of 10 (keeping the remaining parameters fixed), and measured the corresponding MAE, over the validation set. Figure C.1(a) shows that the MAE decreases with an increase in the number of clusters. We observe that the MAE keeps on decreasing, however after 140 clusters for the SML dataset and 100 clusters for the FT1 dataset, the decrease is very insignificant. For this reason we choose the cluster sizes to be 140 and 100 for the SML and FT1 datasets respectively for the subsequent experiments.

C.1.2 Optimal Number of Neighbours for the CCF

We changed the number of neighbours for the Cluster-based CF (CCF) from 10 to 100 and measured the corresponding MAE. Figure C.1(b) shows how the MAE changes as a function of neighbourhood size. We note that in the case of SML dataset, the MAE keeps on decreasing with an increase in the number of neighbours, reaches at its minimum for neighbourhood size of 50, and then starts increasing again. For the FT1 dataset, the neighbourhood size of 40 gives the lowest MAE. We choose the optimal neighbourhood sizes to be 50 and 40 for the SML and FT1 datasets respectively.

Table C.1: Comparing the performance of different variants of the CF-based algorithms over different types of users. "All" represents all users, "GS" represents the gray-sheep users, and "Remaining" represents the users not identified as the gray-sheep. We observe that the CF-based algorithms fail to produce good recommendations for the gray-sheep users.

Metric	Approach	Users		
		All	**GS**	**Remaining**
MAE	UBCF	1.486 ± 0.008	1.601 ± 0.032	1.457 ± 0.007
	IBCF	1.449 ± 0.007	1.567 ± 0.052	1.432 ± 0.007
	CCF	1.492 ± 0.021	1.611 ± 0.042	1.470 ± 0.022
ROC-Sensitivity	UBCF	0.441 ± 0.008	0.351 ± 0.022	0.462 ± 0.007
	IBCF	0.534 ± 0.007	0.421 ± 0.019	0.564 ± 0.005
	CCF	0.492 ± 0.008	0.412 ± 0.021	0.513 ± 0.008
Coverage	UBCF	93.851 ± 0.071	76.732 ± 1.23	94.871 ± 0.006
	IBCF	95.262 ± 0.088	89.11 ± 0.901	95.674 ± 0.082
	CCF	95.539 ± 0.080	91.243 ± 0.901	95.770 ± 0.071

C.1.3 Optimal Number of Iterations

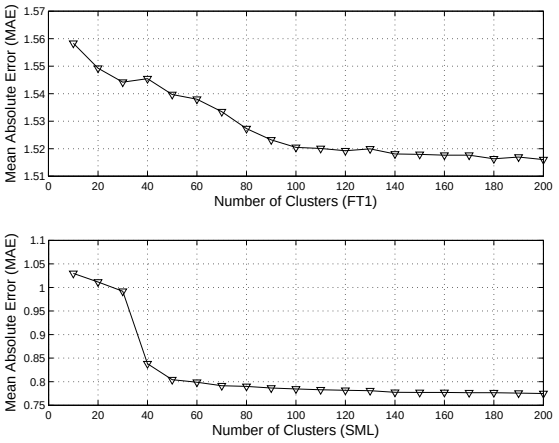
Figure C.1(c) shows how the MAE changes with an increase in the number of iterations. For the SML dataset the MAE keeps on decreasing with an increase in the number of iterations until it converges. After 5 iterations, the difference in MAE observed between two iterations becomes very small. To keep a good balance between computation and performance requirement, we choose the optimal number of iterations to be 5. Similarly, we tuned the optimal number of iterations for the FT1 dataset, which are found to be 2.

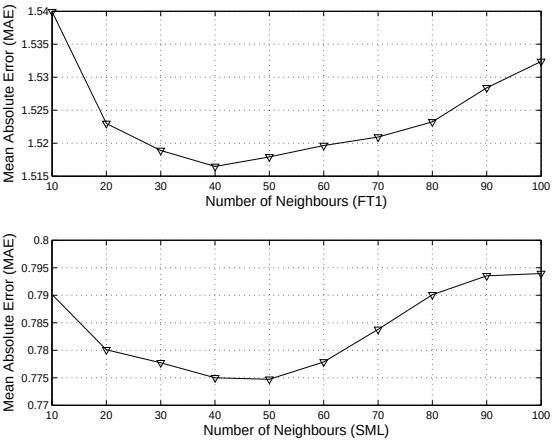
C.1.4 Optimal value of pow_{thr}

The optimal value of pow_{thr} is found to be 0.63 for the SML and 0.26 for the FT1 dataset.

C.2 Performance Evaluation of Different CF-Based Algorithms for the Gray-Sheep Users

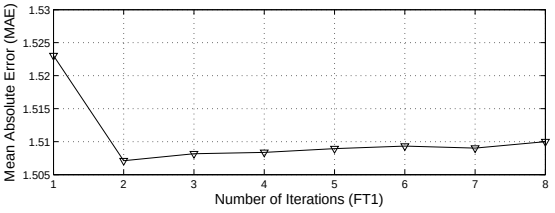
Table C.1 shows the performance of different algorithms for the gray-sheep users. In the table, UBCF represents the user-based CF, IBCF represents the item-based CF, and CCF represents the clusters-based CF (refer to Section 6.2). We observe that the CF-based algorithms fail to produce good recommendations for the gray-sheep users. The reason is the same as discussed in Chapter 6.

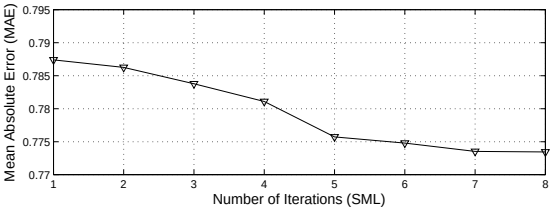




(a) Finding the optimal numbers of clusters

(b) Finding the optimal neighbourhood size





(c) Finding the optimal numbers of iterations

Figure C.1: (From left to right, top to bottom) Determining the optimal number of clusters, neighbourhood size in the Cluster-based CF algorithm (CCF), and number of iterations (itr) in K-means clustering algorithm through the validation set.

Appendix D

KMR Algorithms

D.1 Comparing the KMR with Others in Terms of ROC-Sensitivity and F1 Measure

Tables D.1 and D.2 compare the performance of the KMR algorithms with others in terms of ROC-sensitivity and F1 measure respectively. We observe that the KMR algorithms outperform (or gives comparable results to) other algorithms.

D.2 New Item Cold-Start Scenario

We tested the new item cold-start scenario in exactly the same way we did the new user cold-start scenario. That is, we selected 100 random items, and kept the number of users in the training set who have rated these item to 2, 5, 10, 15, and 20. Table D.3 shows again that the standard predictor fails under this scenario, whereas including the mean, mode and median predictor gives very good performance. We note that for new items the feature-based and demographic-based recommenders work well for the cold-start scenario as these measures are not strongly influenced by a lack of rating information for an item.

D.3 Long Tail Scenario

The long tail scenario (Park and Tuzhilin, 2008) is an important scenario for practical recommender systems. In a large e-commerce system like Amazon, there are huge number of items that are rated by very few users and hence the recommendations generated for these items would be poor, which could weaken the customers trust in the system.

Table D.1: A comparison of the KMR algorithm with others in terms of ROC-sensitivity. The best results are shown in bold font.

Algorithm	**Best *ROC-Sensitivity***				
	SML	**ML**	**ML**10	**FT**5	**NF**
$UBCF_{DV}$	0.714 ± 0.004	0.742 ± 0.001	0.721 ± 0.002	0.526 ± 0.008	0.651 ± 0.002
IBCF	0.654 ± 0.003	0.730 ± 0.002	0.731 ± 0.002	0.541 ± 0.006	0.642 ± 0.002
Hybrid CF	0.708 ± 0.003	0.755 ± 0.002	0.724 ± 0.001	0.535 ± 0.006	0.658 ± 0.002
SVD	0.607 ± 0.002	0.634 ± 0.002	0.652 ± 0.002	0.469 ± 0.012	0.624 ± 0.003
KMR_{ib}	0.708 ± 0.002	0.732 ± 0.001	0.732 ± 0.002	0.570 ± 0.008	0.654 ± 0.003
KMR_{ub}	0.716 ± 0.002	0.752 ± 0.001	0.718 ± 0.002	0.590 ± 0.006	0.661 ± 0.003
KMR^{var}_{hybrid}	$\mathbf{0.729 \pm 0.002}$	$\mathbf{0.758 \pm 0.001}$	$\mathbf{0.738 \pm 0.002}$	$\mathbf{0.592 \pm 0.006}$	$\mathbf{0.662 \pm 0.002}$

Table D.2: A comparison of the proposed algorithm with others in terms of F1. The best results are shown in bold font.

Algorithm	**Best *F*1**				
	SML	**ML**	**ML**10	**FT**5	**NF**
$UBCF_{DV}$	0.536 ± 0.003	0.549 ± 0.002	0.542 ± 0.041	0.516 ± 0.004	0.427 ± 0.002
IBCF	0.526 ± 0.002	0.542 ± 0.002	0.550 ± 0.040	0.521 ± 0.004	0.434 ± 0.003
Hybrid CF	0.528 ± 0.021	0.562 ± 0.002	0.549 ± 0.040	0.518 ± 0.004	0.436 ± 0.002
SVD	0.441 ± 0.002	0.407 ± 0.003	0.477 ± 0.012	0.445 ± 0.003	0.404 ± 0.003
KMR_{ib}	0.533 ± 0.003	0.596 ± 0.003	0.549 ± 0.018	0.523 ± 0.005	0.439 ± 0.003
KMR_{ub}	0.531 ± 0.003	0.587 ± 0.003	0.545 ± 0.013	0.540 ± 0.004	0.440 ± 0.003
KMR^{var}_{hybrid}	$\mathbf{0.539 \pm 0.003}$	$\mathbf{0.606 \pm 0.003}$	$\mathbf{0.551 \pm 0.004}$	$\mathbf{0.545 \pm 0.004}$	$\mathbf{0.442 \pm 0.003}$

Table D.3: Comparing the MAE observed in different approaches under **new item cold-start scenario**, for the SML dataset. The superfix M^4 represents the corresponding version of the KMR algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. Average represents the average rating given by all users in the dataset. The best results are shown in bold font.

Approach	**Best MAE**				
	MAE2	**MAE**5	**MAE**10	**MAE**15	**MAE**20
KMR_{ib}	1.782 ± 0.003	1.692 ± 0.004	1.421 ± 0.004	1.321 ± 0.004	1.221 ± 0.004
KMR_{ub}	3.253 ± 0.005	3.055 ± 0.004	2.811 ± 0.004	2.610 ± 0.004	2.453 ± 0.004
KMR_{F}	0.881 ± 0.005	$\mathbf{0.821 \pm 0.002}$	$\mathbf{0.792 \pm 0.002}$	$\mathbf{0.783 \pm 0.003}$	$\mathbf{0.776 \pm 0.002}$
KMR_{D}	0.924 ± 0.005	0.873 ± 0.002	0.813 ± 0.002	0.809 ± 0.004	0.809 ± 0.002
$KMR_{ib}^{M^4}$	0.953 ± 0.003	0.948 ± 0.004	0.928 ± 0.003	0.918 ± 0.003	0.887 ± 0.003
$KMR_{ub}^{M^4}$	$\mathbf{0.840 \pm 0.002}$	0.848 ± 0.004	0.847 ± 0.003	0.837 ± 0.003	0.832 ± 0.002
$KMR_{F}^{M^4}$	0.905 ± 0.003	0.904 ± 0.003	0.838 ± 0.003	0.790 ± 0.003	0.782 ± 0.003
$KMR_{D}^{M^4}$	0.916 ± 0.003	0.916 ± 0.003	0.863 ± 0.003	0.815 ± 0.003	0.796 ± 0.003
$KMR_{F+ib}^{M^4}$	0.849 ± 0.002	0.837 ± 0.002	0.807 ± 0.002	0.795 ± 0.002	0.786 ± 0.002
Average	0.918 ± 0.003	0.915 ± 0.003	0.840 ± 0.003	0.831 ± 0.002	0.824 ± 0.002

Table D.4: Comparing MAE observed in different approaches under the **long tail scenario**, for the SML dataset. The superfix M^4 represents the corresponding version of the KMR algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. Average represents the average rating given by all users in the dataset. The best results are shown in bold font.

Approach	**Best MAE**					
	MAE2	**MAE**4	**MAE**6	**MAE**8	**MAE**10	**MAE**15
KMR_{ib}	3.666 ± 0.005	3.652 ± 0.005	3.487 ± 0.005	3.432 ± 0.005	3.414 ± 0.004	3.371 ± 0.005
KMR_{ub}	3.481 ± 0.004	3.415 ± 0.004	3.336 ± 0.005	3.265 ± 0.004	3.239 ± 0.004	3.208 ± 0.004
KMR_{F}	3.022 ± 0.004	3.017 ± 0.004	2.964 ± 0.004	2.894 ± 0.005	2.822 ± 0.004	2.761 ± 0.004
KMR_{D}	2.963 ± 0.004	2.946 ± 0.004	2.872 ± 0.004	2.820 ± 0.005	2.683 ± 0.004	2.608 ± 0.004
$KMR_{ib}^{M^4}$	0.976 ± 0.005	0.966 ± 0.003	0.865 ± 0.003	0.840 ± 0.003	0.820 ± 0.004	0.817 ± 0.003
$KMR_{ub}^{M^4}$	$\mathbf{0.884 \pm 0.005}$	0.875 ± 0.005	0.843 ± 0.003	0.834 ± 0.003	0.828 ± 0.003	0.820 ± 0.003
$KMR_{F}^{M^4}$	0.988 ± 0.003	0.970 ± 0.003	0.869 ± 0.003	0.845 ± 0.003	0.818 ± 0.003	0.810 ± 0.003
$KMR_{D}^{M^4}$	0.966 ± 0.004	0.964 ± 0.004	0.867 ± 0.004	0.841 ± 0.004	0.819 ± 0.004	0.815 ± 0.004
$KMR_{ib+F}^{M^4}$	0.885 ± 0.003	$\mathbf{0.860 \pm 0.003}$	$\mathbf{0.835 \pm 0.003}$	$\mathbf{0.829 \pm 0.003}$	$\mathbf{0.809 \pm 0.003}$	$\mathbf{0.802 \pm 0.002}$
Average	0.956 ± 0.004	0.951 ± 0.004	0.927 ± 0.004	0.881 ± 0.003	0.872 ± 0.003	0.863 ± 0.003

Table D.5: Comparing the performance of different KMR approaches under **imbalanced and sparse datasets**. The superfix M^4 represents the corresponding version of the KMR algorithm, where we take into account the max, mean, mode, and median of the output probability distribution. Average represents the average rating given by all users in the dataset. The best results are shown in bold font.

Approach	**MAE**			
	$\mathbf{x} \in \{\mathbf{50\%}, \mathbf{100\%}\}$		$\mathbf{x} \in \{\mathbf{75\%}, \mathbf{100\%}\}$	
	FT5	**SML**	**FT5**	**SML**
KMR_{ib}	1.790 ± 0.002	1.040 ± 0.002	1.930 ± 0.002	1.171 ± 0.002
KMR_{ub}	2.237 ± 0.002	1.091 ± 0.002	2.250 ± 0.002	1.182 ± 0.002
KMR_{D}	1.801 ± 0.002	1.052 ± 0.001	1.943 ± 0.002	1.174 ± 0.002
KMR_{F}	1.773 ± 0.002	1.030 ± 0.001	1.912 ± 0.002	1.162 ± 0.002
$KMR_{ib}^{M^4}$	1.752 ± 0.002	0.941 ± 0.001	1.771 ± 0.002	0.981 ± 0.001
$KMR_{ub}^{M^4}$	1.775 ± 0.001	0.945 ± 0.001	1.791 ± 0.002	0.983 ± 0.001
$KMR_{D}^{M^4}$	1.762 ± 0.002	0.931 ± 0.001	1.781 ± 0.003	0.955 ± 0.001
$KMR_{F}^{M^4}$	1.758 ± 0.002	0.938 ± 0.001	1.775 ± 0.002	0.951 ± 0.001
$KMR_{ib+F}^{M^4}$	$\mathbf{1.739 \pm 0.001}$	$\mathbf{0.921 \pm 0.001}$	$\mathbf{1.749 \pm 0.001}$	$\mathbf{0.931 \pm 0.001}$
Average	1.788 ± 0.001	1.021 ± 0.001	1.943 ± 0.001	1.152 ± 0.001

To test the performance of the proposed algorithms under long tail scenario, we created the artificial long tail scenario by randomly selecting the 80% of items in the tail. The number of ratings given in the tail part were varied between 2 to 15—this ensure that the item is new and have very few ratings. Table D.4 again shows the failure of the standard predictor in the long tail scenario and the improvement obtained by using the mean, mode and median predictor.

D.4 Very Sparse and Imbalanced Dataset

To check the performance of the proposed approaches under (very) sparse and imbalanced dataset, we created subsets of the datasets by withholding x% ratings from the rating profiles of users/items, where $x \in [x_{min}, x_{max}]$. We show results for two scenarios: (1) $x_{min} = 50\%$, $x_{max} = 100\%$, (2) $x_{min} = 75\%$, $x_{max} = 100\%$. Changing the value of x_{min} creates different sparse subsets of the dataset, whereas, keeping the value of x_{max} to 100% ensures that the imbalanced dataset is created for each scenario.

For the SML and FT5 dataset, the results are shown in Table D.5. Again this follows the same pattern as the long tail and cold-start scenarios.

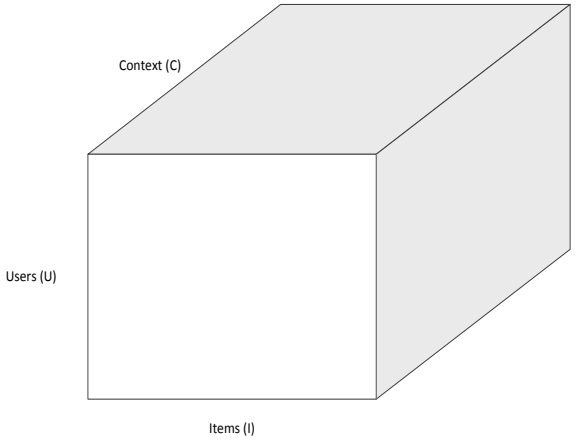


Figure D.1: The conventional user-item rating matrix extended by the context information.

D.5 Adding Contextual Information

In the case of two-dimensional users $\times$ items space, the ratings given by $\mathcal{U}$ users on $\mathcal{I}$ items can be expressed by:

$$f : U \times \mathcal{I} \rightarrow \mathbb{R},$$

For the contextual data, each item rated by a user is associated with the context ($C = \{c_1, c_2, \cdots, c_c\}$. We might express this relationship by:

$$\begin{aligned} f &: \mathcal{U} \times \mathcal{I} \times C \rightarrow \mathbb{R} \\ f &: (\mathcal{U}^{c_1} \times \mathcal{I}^{c_1} \times c_1) + (\mathcal{U}^{c_2} \times \mathcal{I}^{c_2} \times c_2) + \cdots (\mathcal{U}^{c_c} \times \mathcal{I}^{c_c} \times c_c) \rightarrow \mathbb{R}, \\ f &: \underbrace{(\mathcal{U}^{c_1} \times \mathcal{I}^{c_1} \times c_1)}_{\gamma_{iu}^{c_1}} + \underbrace{(\mathcal{U}^{c_2} \times \mathcal{I}^{c_2} \times c_2)}_{\gamma_{iu}^{c_2}} + \cdots \underbrace{(\mathcal{U}^{c_c} \times \mathcal{I}^{c_c} \times c_c)}_{\gamma_{iu}^{c_c}} \rightarrow \mathbb{R} \end{aligned}$$

where, $\gamma_{iu}^{c} = K_{\hat{r}}(\hat{r}_{iu}(c), \hat{r}_{i'u}(c)) K_{\mathbf{q}}(\mathbf{q}_i(c), \mathbf{q}_{i'}(c))$. We compute separate feature and residual kernels (refer to Section 7.3) for the data associated with each context.

We considered trusted friends of an active users as their social context and shed light on how does the recommendation accuracy improves if we use information from friend of a friend (FOAF), or friend of FOAF, and so on. Specifically, we want to answer the question: *"how many friends must I ask to get a good recommendation"*? In the following, KMR^{full} represents the KMR algorithm that takes the whole network of the users into account, KMR^{foaf}, which produce recommendation by taking friends,

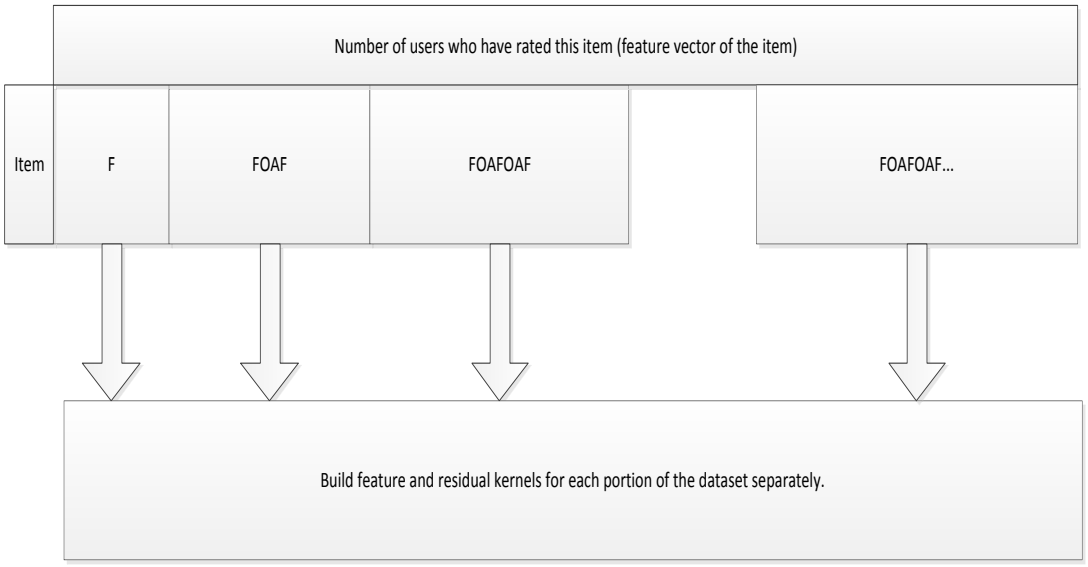


Figure D.2: We divide the users who have rated an item into different blocks based on their relation with the active user. Specifically, we divide users into friends, friend of a friend (FOAF), and so on. We compute separate kernel functions for each block of data. The final recommendation is defined as an aggregate function of these kernels.

friend of a friend (FOAF), etc. information into account, KMR^{rand}, which produces recommendations by taking random users into account.

We obtained relationships between users in the FilmTrust dataset using FOAF. We constructed a simple undirected graph containing 1 092 symmetrical relationships. We removed all users with no friends and fewer than 5 ratings, leaving 513 users. We also removed movies with only 1 rating, leaving 881 movies. This filtered dataset has a sparsity level of approximately 97%. It leaves total rating to 13 560.

Figure D.3 shows that the performance of KMR^{foaf} is comparable to the KMR^{full} when a sufficient number of users are consulted. Furthermore, the performance of the KMR^{foaf} is better than the KMR^{rand}. This is probably because friends are more trusted and have similar tastes. It also shows that in movies domains, users develop social connections based on similar preferences. It must be noted that both of these variants, i.e. KMR^{foaf} and KMR^{rand} consider the same number of users to produce recommendations. The difference is that, KMR^{foaf} only takes users who are friends (or friends of the friends etc.) of the active users, where as KMR^{rand} takes any random user into account.

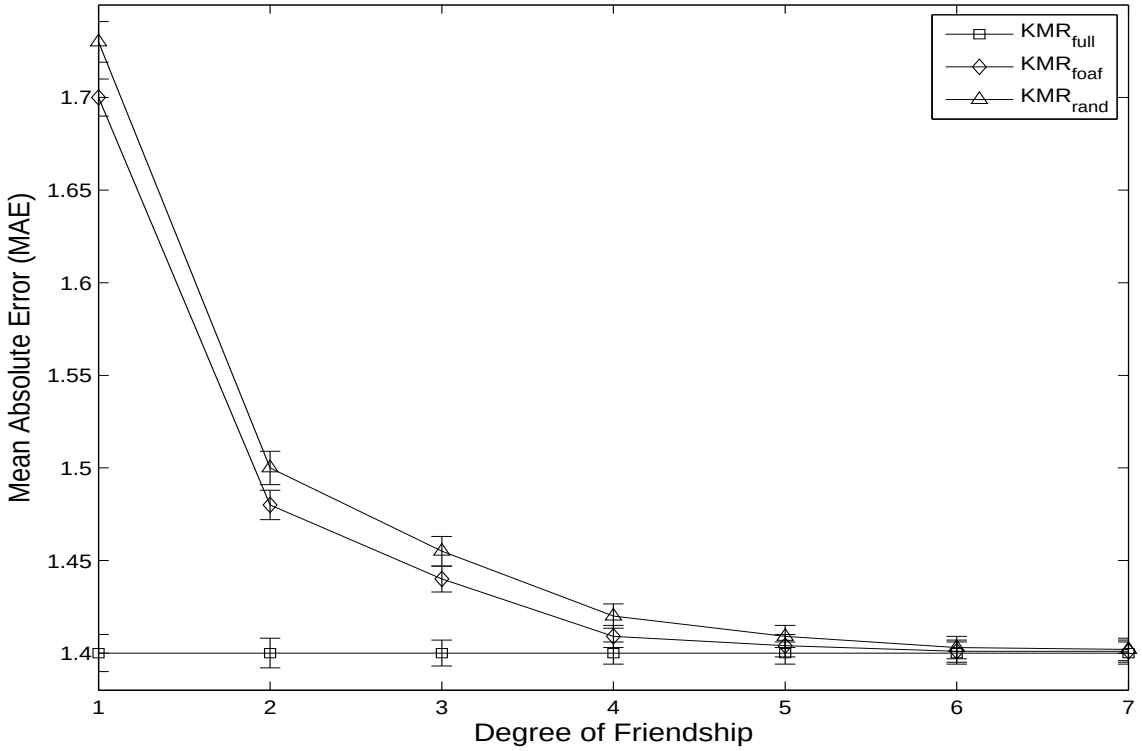


Figure D.3: Comparing the performance of the KMR^{full} with KMR^{foaf}. The degree 1 means friends of an active user, 2 means friends and FOAF (friend of a friend) of an active user, and so on. KMR^{rand} employs the same number of randomly selected users as many friends we have in that degree of separation.

Appendix E

Incremental and On-line KMR Algorithms

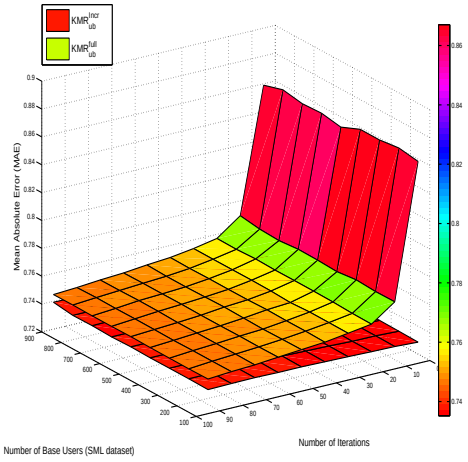
E.1 Comparing the Performance of the KMR_{ub}^{incr} Algorithm With the KMR_{ub}^{full}

Figure E.1 compares the performance of the proposed algorithm, KMR_{ub}^{incr}, with the baseline one, KMR_{ub}^{full}, when new users enter the system. We observe that the proposed algorithm outperforms the baseline algorithm at every combination of base model size and number of iterations. The percentage decrease in the MAE in case of the proposed algorithm compared to the baseline one, keeping the number of base users and iterations fixed to 500 and 5 respectively, is found to be 7% for the SML and 1.5% for the FT5 dataset.

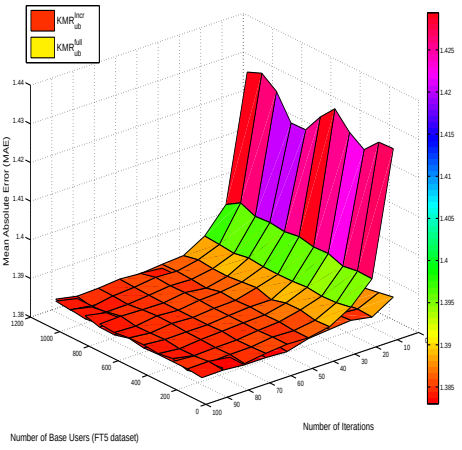
Figure E.2 compares the performance of the proposed algorithm, KMR_{ub}^{incr}, with the baseline one, KMR_{ub}^{full}, when new movies enter the system. The percentage decrease in the MAE in case of the proposed algorithm compared to the baseline one, keeping the number of base movies and iterations fixed to 200 and 5 respectively, is found to be 8.3% for the SML and 1.4% for the FT5 dataset.

E.2 Results of the $KMR^{percept}$ Algorithm, When New Users/Movies are Introduced in the System

We performed experiments for the scenarios discussed in Section 8.3. Figure E.3 shows that the performance of the proposed algorithm, $KMR^{percept}$, is comparable to the baseline one, KMR^{full}. Note that, the KMR^{full} updates the model using 400 iterations on the arrival of new data, which is expensive.

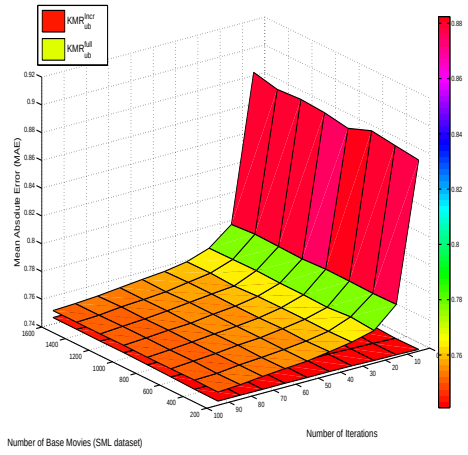


(a) KMR_{ub}^{incr} vs. KMR_{ub}^{full} (SML dataset)

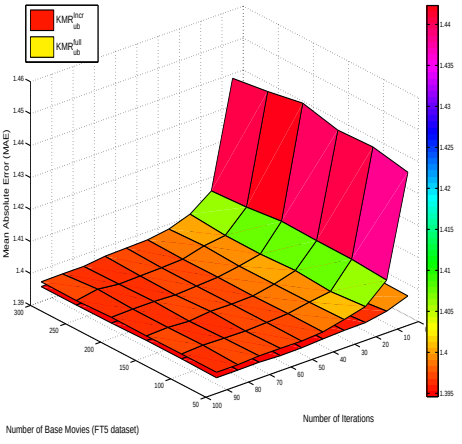


(b) KMR_{ub}^{incr} vs. KMR_{ub}^{full} (FT5 dataset)

Figure E.1: Comparing the performance of the proposed algorithm, KMR_{ub}^{incr}, with the baseline one, KMR_{ub}^{full}, when new users are added in the system. "Number of Base Users" represents the number of users used to build the base model (i.e. $\mathcal{U}^{base}$). The model was updated by adding the remaining users (i.e. $\mathcal{U}^{new}$). "Number of Iterations" represents the number of iterations used to train the updated model.



(a) KMR_{ub}^{incr} vs. KMR_{ub}^{full}(SML dataset)



(b) KMR_{ub}^{incr} vs. KMR_{ub}^{full}(FT5 dataset)

Figure E.2: Comparing the performance of the proposed algorithm, KMR_{ub}^{incr}, with the baseline one, KMR_{ub}^{full}, when new movies are added in the system. "Number of Base Movies" represents the number of movies used to build the base model (i.e. $\mathcal{I}^{base}$). The model was updated by adding the remaining movies (i.e. $\mathcal{I}^{new}$). "Number of Iterations" represents the number of iterations used to train the updated model.

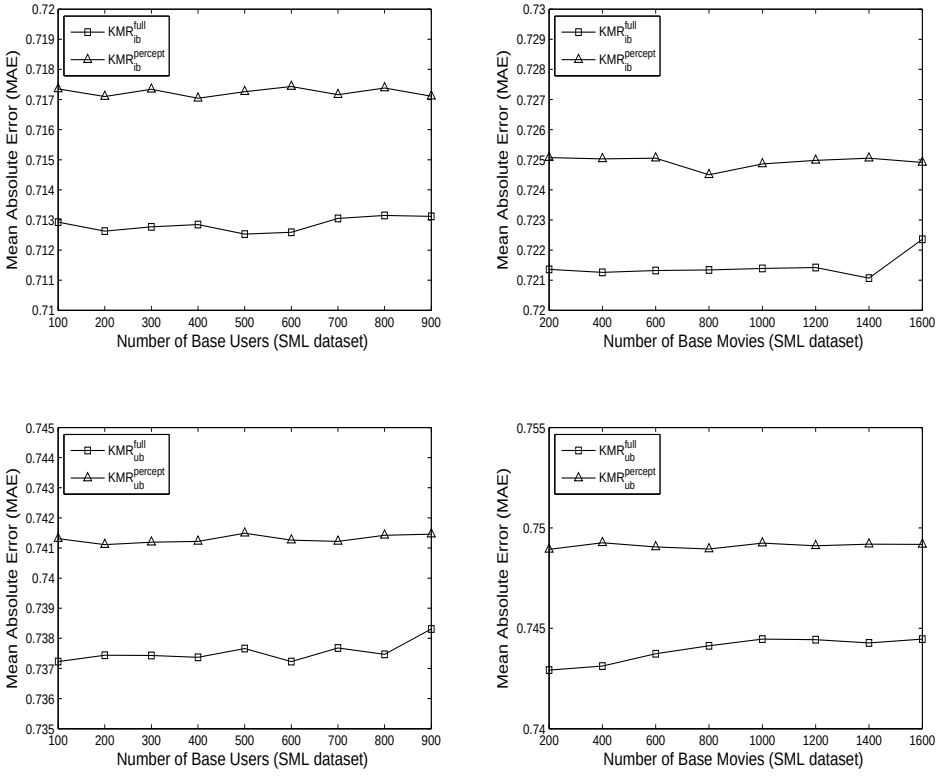


Figure E.3: Comparing the performance of the proposed algorithm $KMR^{percept}$ with the baseline one, KMR^{full}, when new users and movies are added in the system. The "Number of Base Movies" represents the number of movies used to build the base model (i.e. $\mathcal{I}^{base}$). The model was updated by adding the remaining movies (i.e. $\mathcal{I}^{new}$). Similarly, the 'Number of Base Users" represents the number of users used to build the base model (i.e. $\mathcal{U}^{base}$). The model was updated by adding the remaining users (i.e. $\mathcal{U}^{new}$). The KMR^{full} is trained using 400 iterations.

References

Kjersti Aas and Line Eikvil. Text categorisation: A survey., 1999.

Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23:103–145, January 2005. ISSN 1046-8188.

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17:734–749, June 2005. ISSN 1041-4347.

Hyung Jun Ahn. A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem. *Inf. Sci.*, 178:37–51, January 2008. ISSN 0020-0255.

Satnam Alag. *Collective Intelligence in Action*. Manning Publications, October, 2008.

Xavier Amatriain, Neal Lathia, Josep M. Pujol, Haewoon Kwak, and Nuria Oliver. The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 532–539, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-483-6.

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 978-0-898716-24-5.

Katja Astikainen, Liisa Holm, Esa Pitkanen, Sandor Szedmak, and Juho Rousu. Towards structured output prediction of enzyme function. *BMC Proceedings*, 2(Suppl 4):S2+, 2008. ISSN 1753-6561.

Yossi Azar, Amos Fiat, Anna Karlin, Frank McSherry, and Jared Saia. Spectral analysis of data. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 619–626, New York, NY, USA, 2001. ACM. ISBN 1-58113-349-9.

Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40:66–72, March 1997. ISSN 0001-0782.

Linas Baltrunas. Exploiting contextual information in recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 295–298, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7.

Ana Belén Barragáns-Martínez, Enrique Costa-Montenegro, Juan C. Burguillo, Marta Rey-López, Fernando A. Mikic-Fonte, and Ana Peleteiro. A hybrid content-based and item-based collaborative filtering approach to recommend tv programs enhanced with singular value decomposition. *Inf. Sci.*, 180:4290–4311, November 2010. ISSN 0020-0255.

J. Basilico and T. Hofmann. Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first international conference on Machine learning*, pages 65–72, New York, NY, USA, 2004. ACM Press.

Farran Bassam. *One-pass algorithms for large and shifting data sets*. PhD thesis, UNIVERSITY OF SOUTHAMPTON, UK, 2010.

Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: using social and content-based information in recommendation. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 714–720, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7.

R.M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix prize, in: AT&T Labs–Research: Technical report November, 2007.

Robert M. Bell and Yehuda Koren. Lessons from the netflix prize challenge. *SIGKDD Explor. Newsl.*, 9:75–79, December 2007a. ISSN 1931-0145.

Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 43–52, Washington, DC, USA, 2007b. IEEE Computer Society. ISBN 0-7695-3018-4.

James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD Cup and Workshop*, volume 2007. Citeseer, 2007.

Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37:573–595, December 1995. ISSN 0036-1445.

Dimitri P. Bertsekas. *Nonlinear programming*. Athena Scientific Belmont, MA, 1999.

Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, pages 46–54, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-556-8.

Daniel Billsus and Michael J. Pazzani. User modeling for adaptive news access. *User Modeling and User-Adapted Interaction*, 10:147–180, February 2000. ISSN 0924-1868.

John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-555-X.

Derek G. Bridge and Jerome Kelleher. Experiments in sparsity reduction: Using clustering in collaborative recommenders. pages 144–149, 2002.

Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2:121–167, June 1998. ISSN 1384-5810.

Luca Buriano, Marco Marchetti, Francesca Carmagnola, Federica Cena, Cristina Gena, and Ilaria Torre. The role of ontologies in context-aware recommender systems. In *Proceedings of the 7th International Conference on Mobile Data Management*, MDM '06, pages 80–, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2526-1.

Robin Burke. Integrating knowledge-based and collaborative-filtering recommender systems. In *In AAAI Workshop on AI in Electronic Commerce*, pages 69–72. AAAI, 1999.

Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002.

Robin Burke. The adaptive web. chapter Hybrid web recommender systems, pages 377–408. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72078-2.

John Canny. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 238–245, New York, NY, USA, 2002. ACM. ISBN 1-58113-561-0.

Iván Cantador, Alejandro Bellogín, and Pablo Castells. A multilayer ontology-based hybrid recommendation model. *AI Commun.*, 21:203–210, April 2008. ISSN 0921-7126.

Iván Cantador, Pablo Castells, and Alejandro Bellogín. Modelling Ontology-based Multilayered Communities of Interest for Hybrid Recommendations. In *Workshop on Adaptation and Personalisation in Social Systems: Groups, Teams, Communities, at the 11th International Conference on User Modeling*, June 2007.

Claire Cardie. Empirical methods in information extraction. *AI magazine*, 18:65–79, 1997.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2:27:1–27:27, May 2011. ISSN 2157-6904.

William Cheetham and Joseph Price. Measures of Solution Accuracy in Case-Based Reasoning Systems. *Advances in Case-Based Reasoning*, pages 106–118, 2004.

Mark Claypool, Anuja Gokhale, Tim Mir, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *In Proceedings of ACM SIGIR Workshop on Recommender Systems*, Berkeley, California, 1999. ACM.

Patrick Clerkin, Pádraig Cunningham, and Conor Hayes. Concept discovery in collaborative recommender systems. pages 34–39, Dublin, Ireland, September 17–19 2003.

Mark Connor and John Herlocker. Clustering items for collaborative filtering. 2001.

Paul Cotter and Barry Smyth. Ptv: Intelligent personalised tv guides. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 957–964. AAAI Press, 2000. ISBN 0-262-51112-6.

Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods*. Cambridge Univ Pr, 2000.

Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 271–280, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-654-7.

Dennis DeCoste. Collaborative prediction using ensembles of maximum margin matrix factorizations. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 249–256, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2.

C.B. Do and S. Batzoglou. What is the expectation maximization algorithm? *Nature biotechnology*, 26(8):897–899, 2008.

Jonathan Gemmell, Thomas Schimoler, Bamshad Mobasher, and Robin D. Burke. Resource recommendation in collaborative tagging applications. In Francesco Buccafurri and Giovanni Semeraro, editors, *EC-Web*, volume 61 of *Lecture Notes in Business Information Processing*, pages 1–12. Springer, 2010. ISBN 978-3-642-15207-8.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. An Improved Switching Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering. In *Lecture Notes in Engineering and Computer Science: Proceedings of The International Multi Conference of Engineers and Computer Scientists 2010*, pages 493–502. IMECS 2010, 17–19 March, 2010, Hong Kong, 2010a.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. Building Switching Hybrid Recommender System Using Machine Learning Classifiers and Collaborative Filtering. *IAENG International Journal of Computer Science*, 37(3):272–287, 2010b.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. Novel heuristics for coalition structure generation in multi-agent systems. In *The 2010 International Conference of Computational Intelligence and Intelligent Systems*. ICCIIS'10, 30 June–2 July 2010, London, U.K., 2010c.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. Novel significance weighting schemes for collaborative filtering: Generating improved recommendations in sparse environments. In *DMIN*, pages 334–342. CSREA Press, 2010d. ISBN 1-60132-138-4.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. A scalable, accurate hybrid recommender system. In *Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining*, WKDD '10, pages 94–98, Washington, DC, USA, 2010e. IEEE Computer Society. ISBN 978-0-7695-3923-2.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. The advantage of careful imputation sources in sparse data-environment of recommender systems: Generating improved svd-based recommendations. In *IADIS European Conference on Data Mining*, July 2011a.

Mustansar A. Ghazanfar and Adam Prügel-Bennett. Fulfilling the needs of gray-sheep users in recommender systems, a clustering solution. In *2011 International Conference on Information Systems and Computational Intelligence*, January 2011b.

Mustansar A. Ghazanfar, Adam Prügel-Bennett, and Sándor Szedmák. Kernel mapping recommender system algorithms. *Information Sciences*, 2012.

Mustansar A. Ghazanfar, Sándor Szedmák, and Adam Prügel-Bennett. Incremental kernel mapping algorithms for scalable recommender systems. In *ICTAI*, pages 1077–1084, 2011.

David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, December 1992. ISSN 0001-0782.

Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4:133–151, July 2001. ISSN 1386-4564.

Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence. ISBN 0-262-51106-1.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009. ISSN 1931-0145.

Choochart Haruechaiyasak, Chatchawal Tipnoe, Sarawoot Kongyoung, Chaianun Damrongrat, and Niran Angkawattanawit. A dynamic framework for maintaining customer profiles in e-commerce recommender systems. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, EEE '05, pages 768–771, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2274-2.

Conor Hayes and Padraig Cunningham. Context boosting collaborative recommendations. *Knowledge-Based Systems*, 17(2-4):131–138, May 2004. ISSN 09507051.

Jon Herlocker, Joseph A. Konstan, and John Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inf. Retr.*, 5:287–310, October 2002. ISSN 1386-4564.

Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22:5–53, January 2004. ISSN 1046-8188.

C.W. Hsu, C.C. Chang, C.J. Lin, et al. A practical guide to support vector classification, 2003.

Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett.*, 31: 651–666, June 2010. ISSN 0167-8655.

Thorsten Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, pages 137–142, London, UK, 1998. Springer-Verlag. ISBN 3-540-64417-2.

Thorsten Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 217–226, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

Jerome Kelleher and Derek Bridge. Rectree centroid: An accurate, scalable collaborative recommender. In *Procs. of the Fourteenth Irish Conference on Artificial Intelligence and Cognitive Science*, pages 89–94. Citeseer, 2003.

Dohyun Kim and Bong-Jin Yum. Collaborative filtering based on iterative principal component analysis. *Expert Syst. Appl.*, 28:823–830, May 2005. ISSN 0957-4174.

Heung-Nam Kim, Abdulmotaleb El-Saddik, and Geun-Sik Jo. Collaborative error-reflected models for cold-start recommender systems. *Decision Support Systems*, 51 (3):519 – 531, 2011. ISSN 0167-9236.

Arnd Kohrs and Bernard Merialdo. Clustering for collaborative filtering applications. *Computational Intelligence for Modelling, Control & Automation. IOS Press*, 1999.

Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Commun. ACM*, 40:77–87, March 1997. ISSN 0001-0782.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009. ISSN 0018-9162.

M. Kurucz, A.A. Benczúr, and K. Csalogány. Methods for large scale SVD with missing values. In *Proceedings of KDD Cup and Workshop*. Citeseer, 2007.

Ken Lang. NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995.

Neil D. Lawrence and Raquel Urtasun. Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 601–608, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1.

George Lekakos and Petros Caravelas. A hybrid approach for movie recommendation. *Multimedia Tools Appl.*, 36:55–70, January 2008. ISSN 1380-7501.

Qing Li and Byeong Man Kim. An approach for combining content-based and collaborative filters. In *Proceedings of the sixth international workshop on Information retrieval with Asian languages - Volume 11*, AsianIR '03, pages 17–24, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

Antonis Loizou. *How to recommend music to book lovers: Enabling the provision of recommendations from multiple domains.* PhD thesis, 2009.

Hao Ma, Irwin King, and Michael R. Lyu. 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, pages 39–46, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7.

Lester Mackey, David Weiss, and Michael I. Jordan. Mixed membership matrix factorization. In *Proceedings of the 27th International Conference on Machine Learning*, June 2010.

Veronica Maidel, Peretz Shoval, Bracha Shapira, and Meirav Taieb-Maimon. Evaluation of an ontology-content based filtering method for a personalized newspaper. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 91–98, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7.

Benjamin Marlin. Collaborative Filtering: A Machine Learning Perspective. Master's thesis, University of Toronto, 2004.

A. Martinez, J. Arias, A. Vilas, J. Garcia Duque, and M. Lopez Nores. What's on tv tonight? an efficient and effective personalized recommender system of tv programs. *Consumer Electronics, IEEE Transactions on*, 55(1):286–294, 2009.

Sean M. McNee. *Meeting user information needs in recommender systems*. PhD thesis, UNIVERSITY OF MINNESOTA, USA, 2006.

Sean M. Mcnee, Shyong K. Lam, Catherine Guetzlaff, Joseph A. Konstan, and John Riedl. Confidence Displays and Training in Recommender Systems. In Matthias Rauterberg, Marino Menozzi, Janet Wesson, Matthias Rauterberg, Marino Menozzi, and Janet Wesson, editors, *INTERACT*. IOS Press, 2003. ISBN 1-58603-363-8.

Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth national conference on Artificial intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0.

Stuart E. Middleton. *Capturing knowledge of user preferences with recommender systems*. PhD thesis, UNIVERSITY OF SOUTHAMPTON, UK, September 2002.

Stuart E. Middleton, Harith Alani, and David C. De Roure. Exploiting Synergy Between Ontologies and Recommender Systems. In *The Eleventh International World Wide Web Conference (WWW2002)*, 2002.

Stuart E. Middleton, David D. Roure, and Nigel R. Shadbolt. Ontology-based recommender systems. pages 779–796. Springer, 2009.

Stuart E. Middleton, Nigel R. Shadbolt, and David C. De Roure. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22:54–88, January 2004. ISSN 1046-8188.

Bamshad Mobasher. Recommender systems. *KI*, 21(3):41–43, 2007.

Bamshad Mobasher, Xin Jin, and Yanzan Zhou. Semantically Enhanced Collaborative Filtering on the Web. 3209:57–76, September 2003.

Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, DL '00, pages 195–204, New York, NY, USA, 2000. ACM. ISBN 1-58113-231-X.

H. Jr. Murray. Methods for satisfying the needs of the scientist and the engineer for scientific and technical communication. A Press Release, 1966.

U. Nahm and R. Mooney. Text mining with information extraction, 2002.

Miki Nakagawa and Bamshad Mobasher. A hybrid web personalization model based on site connectivity. In *Proceedings of WebKDD*, pages 59–70, 2003.

Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer verlag, 1999.

Douglas Oard and Jinmook Kim. Implicit feedback for recommender systems. In *Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83, 1998.

Seung-Taek Park, David Pennock, Omid Madani, Nathan Good, and Dennis DeCoste. Naïve filterbots for robust cold-start recommendations. KDD '06, pages 699–705, New York, NY, USA, 2006. ACM. ISBN 1-59593-339-5.

Yoon-Joo Park and Alexander Tuzhilin. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 11–18, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7.

Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup and Workshop*. Citeseer, 2007.

Michael J. Pazzani. A framework for collaborative, content-based and demographic filtering. *Artif. Intell. Rev.*, 13:393–408, December 1999. ISSN 0269-2821.

Michael J. Pazzani and Daniel Billsus. The adaptive web. chapter Content-based recommendation systems, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72078-2.

David M. Pennock, Eric Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory and model-based approach. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 473–480, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-709-9.

Martin Piotte and Martin Chabbert. The pragmatic theory solution to the netflix grand prize, in: Netflix prize documentation, 2009.

Sutheera Puntheeranurak and Hidekazu Tsuji. A multi-clustering hybrid recommender system. In *Proceedings of the 7th IEEE International Conference on Computer and Information Technology*, pages 223–228, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2983-6.

Amirthalingam Ramanan. *Designing a resource-allocating codebook for patch-based visual object recognition.* PhD thesis, UNIVERSITY OF SOUTHAMPTON, UK, 2010.

Al Mamunur Rashid, Shyong K. Lam, George Karypis, and John Riedl. Clustknn: a highly scalable hybrid model-& memory-based cf algorithm. In *Proc. of WebKDD 2006: KDD Workshop on Web Mining and Web Usage Analysis, in conjunction with the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), August 20-23 2006, Philadelphia, PA*. Citeseer, 2006.

Steffen Rendle and Schmidt-Thie Lars. Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 251–258, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7.

Jasson D. M. Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 713–719, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM. ISBN 0-89791-689-1.

Colm Ryan, Derek Greene, Gerard Cagney, and Padraig Cunningham. Missing value imputation for epistatic MAPs. *BMC Bioinformatics*, 11(1):197+, 2010. ISSN 1471-2105.

R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. *Advances in Neural Information Processing Systems*, 20:1257–1264, 2008.

B. Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM conference on Electronic commerce*, EC '00, pages 158–167, New York, NY, USA, 2000a. ACM. ISBN 1-58113-272-7.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system–a case study. In *IN ACM WEBKDD WORKSHOP*. Citeseer, 2000b.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference in Computers and Information Technology*, pages 27–28. Citeseer, 2002a.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, 2002b.

Badrul M. Sarwar, Joseph A. Konstan, Al Borchers, Jon Herlocker, Brad Miller, and John Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, CSCW '98, pages 345–354, New York, NY, USA, 1998. ACM. ISBN 1-58113-009-0.

Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 253–260, New York, NY, USA, 2002. ACM. ISBN 1-58113-561-0.

Vincent. Schickel-Zuber and Boi Faltings. Using an Ontological A-priori Score to Infer User's Preferences. In *Workshop on Recommender Systems-ECAI06*, pages 102–106, 2006.

Deerwester Scott C., Dumais Susan T., Landauer Thomas K., Furnas George W., and Harshman Richard A. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34:1–47, March 2002. ISSN 0360-0300.

Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating word of mouth. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '95, pages 210–217, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-84705-1.

Andriy Shepitsen, Jonathan Gemmell, Bamshad Mobasher, and Robin Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 259–266, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-093-7.

P. Shoval, V. Maidel, and B. Shapira. An ontology- content-based filtering method. *International Journal on Information Theories and Applications*, 15:303–318, 2008.

Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *ICML*, volume 20, page 720, 2003.

Nathan Srebro, Jasson D. M Rennie, and T. Jaakkola. Maximum-margin matrix factorization. *Advances in neural information processing systems*, 17:1329–1336, 2005.

Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In Marko Grobelnik, Dunja Mladenic, and Natasa Milic-Frayling, editors, *KDD-2000 Workshop on Text Mining, August 20*, pages 109–111, Boston, MA, 2000.

David H. Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 111–120, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009. ISSN 1687-7470.

Xiaoyuan Su, Taghi M. Khoshgoftaar, and Russell Greiner. A mixture imputation-boosted collaborative filter. In *Proceedings of the 21th International Florida Artificial Intelligence Research Society Conference (FLAIRS'08)*, pages 312–317, 2008a.

Xiaoyuan Su, Taghi M. Khoshgoftaar, Xingquan Zhu, and Russell Greiner. Imputation-boosted collaborative filtering using machine learning classifiers. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC '08, pages 949–950, New York, NY, USA, 2008b. ACM. ISBN 978-1-59593-753-7.

Sandor Szedmak, Ni Yizhao, and Gunn Steve R. Maximum margin learning with incomplete data: Learning networks instead of tables. *Journal of Machine Learning Research - Proceedings Track*, 11:96–102, 2010.

Martin Szomszor, Ciro Cattuto, Harith Alani, Kieron O'äôhara, Andrea Baldassarri, Vittorio Loreto, and Vito D. P. Servedio. Folksonomies, the semantic web, and movie recommendation. In *Bridging the Gep between Semantic Web and Web 2.0 (SemNet 2007)*, pages 71–84, 2007.

Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Investigation of various matrix factorization methods for large recommender systems. In *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, NETFLIX '08, pages 6:1–6:8, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-265-8.

Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10: 623–656, June 2009. ISSN 1532-4435.

Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter. Phoaks: a system for sharing recommendations. *Commun. ACM*, 40:59–62, March 1997. ISSN 0001-0782.

Thomas Tran and Robin Cohen. Hybrid recommender systems for electronic commerce. In *Knowledge-Based Electronic Markets, Papers from the AAAI Workshop, Technical Report WS-00*, volume 4, 2000.

Tran Tran. Combining collaborative filtering and knowledge-based approaches for better recommendation systems. *Journal of Business and Technology*, 2(2):17–24, 2007.

Gulden Uchyigit and Keith Clark. Agents that learn to give personalized tv program recommendations, in aaai technical report fs-02-04, 2002.

Gulden Uchyigit and Keith Clark. Hierarchical agglomerative clustering for agent-based dynamic collaborative filtering. In *IDEAL*, pages 827–832, 2004.

Robin van Meteren and Maarten van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*. Citeseer, 2000.

M. Van Setten. Supporting people in finding information: hybrid recommender systems and goal-based structuring. 2005.

Manolis Vozalis and Konstantinos G. Margaritis. Applying SVD on generalized item-based filtering. *International Journal of Computer Science and Applications*, 3(3): 27–51, 2006a.

Manolis Vozalis and Konstantinos G. Margaritis. On the enhancement of collaborative filtering by demographic data. *Web Intelli. and Agent Sys.*, 4:117–138, April 2006b. ISSN 1570-1263.

Manolis Vozalis and Konstantinos G. Margaritis. Using svd and demographic data for the enhancement of generalized collaborative filtering. *Information Sciences*, 177: 3017–3037, August 2007. ISSN 0020-0255.

Manolis G. Vozalis and Konstantinos G. Margaritis. Applying svd on item-based filtering. In *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, ISDA '05, pages 464–469, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2286-06.

Sung-Shun Weng and Hui-Ling Chang. Using ontology network analysis for research document recommendation. *Expert Syst. Appl.*, 34:1857–1869, April 2008. ISSN 0957-4174.

I. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999. ISBN 1558605525.

Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. Kea: practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries*, DL '99, pages 254–255, New York, NY, USA, 1999. ACM. ISBN 1-58113-145-3.

Mingru Wu. Collaborative filtering via ensembles of matrix factorizations. In *Proceedings of KDD Cup and Workshop*. Citeseer, 2007.

M. Wurst. Evaluating knowledge management in heterogeneous domains by agent-based simulation. *Agent Mediated Knowledge Management*, page 82, 2005.

Gui-Rong Xue, Chenxi Lin, Qiang Yang, WenSi Xi, Hua-Jun Zeng, Yong Yu, and Zheng Chen. 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, pages 114–121, New York, NY, USA, 2005. ACM. ISBN 1-59593-034-5.

Valentina Zanardi. *Addressing the cold start problem in tag-based recommender systems*. PhD thesis, UCL (University College London), 2011.

Valentina Zanardi and Licia Capra. A scalable tag-based recommender system for new users of the social web. In *Proceedings of the 22nd international conference on Database and expert systems applications - Volume Part I*, DEXA'11, pages 542–557, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-23087-5.

Jiyong Zhang and Pearl Pu. A recursive prediction algorithm for collaborative filtering recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 57–64, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-730–8.

Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *6th SIAM Conference on Data Mining (SDM)*, pages 548–552. Citeseer, 2006.

Sheng Zhang, Weihong Wang, James Ford, Fillia Makedon, and Justin Pearlman. Using singular value decomposition approximation for collaborative filtering. In *Proceedings of the Seventh IEEE International Conference on E-Commerce Technology*, pages 257–264, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2277-7.

Tong Zhang and Vijay S. Iyengar. Recommender systems using linear classifiers. *J. Mach. Learn. Res.*, 2:313–334, March 2002. ISSN 1532-4435.