Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2014

Project Title:	**Active Sample Selection for Large Scale Matrix Completion**
Student:	**Sebastian Grubb**
CID:	**00639926**
Course:	**EE4**
Project Supervisor:	**Dr Wei Dai & Dr Moez Draief**
Second Marker:	**Dr Cong Ling**

Acknowledgements

Abstract

The aim of this project is to investigate and propose active sampling methods. These are useful in the context of matrix completion where an incomplete dataset, such as user-movie ratings or drug-target interactions, are completed by predicting what the empty entries could be. Active sample selection deals with the issue of which new entries are best to request. For example if we know that people liking the film *Alien* also like *Aliens*, then asking a new user if he likes *Aliens* given that he has told us he likes *Alien* is less informative than asking him if he likes *The Good, the Bad and the Ugly* or another unrelated film. Active sampling techniques differing in their success and complexity have been tried. A technique is usually considered to perform well when giving it extra samples leads to a better RMSE on a test set than when the same number of entries are randomly sampled. Active sampling techniques such as requesting the row and column combination which we know the least about are tried. More advanced techniques such as estimating the variance of each predicted sample or look-ahead methods are also investigated. When dealing with datasets containing millions of data-points or more, active sampling can be very useful as less samples need to be requested for sizeable prediction improvements - this is especially practical when sampling a new datapoint costs a lot effort-wise or financially, such as carrying out a new scientific experiment.

Contents

List of Figures

List of Abbreviations

BPMF	Bayesian Probabilistic Matrix Factorisation
CF	Collaborative Filtering
KMPF	Kernelized Probabilistic Matrix Factorisation
MCMC	Markov chain Monte Carlo - A means of approximating a probability distribution
PMF	Probabilistic Matrix Factorisation
RMSE	Root Mean Square Error - Measure of differences between values predicted by a model and the actual values: $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}_i - x_i)^2}$
SVD	Singular Value Decomposition - A method to decompose a matrix into smaller dimension matrices

Chapter 1

Introduction

1.1 Outline

The aim of this project is to develop an active learning system for recommender systems. Recommender systems are a collection of techniques that are able to find resemblances in certain types of datasets and infer missing information, thus providing *recommendations* as to what the missing items may be. If a column is a song and a row a user we would have a music recommendation system. Applications range from user movie ratings prediction to aiding the discovery of drug-target interactions.

Recommender systems work by having a matrix with each row representing a user (or another type of object) and each column a product or item related to the user. The value of a user-item combination is a numerical value indicating the user's score of the item. Not knowing most values results in a sparse matrix with many empty entries, all representing unscored row-column combinations. A recommender system thus aims to infer the score of missing entries from other existing ones. This means that we can try to predict the empty entries and fill in the empty values, thus leading us to complete the matrix and end up with what we estimate the full dataset to look like.

Research in recommender systems is mainly done in Machine Learning, Data Mining and Statistics, though it is also of interest in other fields, such as marketing. Research can either be focused on novel types of recommender systems, ways to improve the accuracy and precision of current systems or ways in which to improve the performance.

Active learning is generally agnostic to the technique used to complete the matrix and instead seeks to build upon these systems, considering them to be a black box. What active learning seeks to do is to tell the system what currently unknown entries in the matrix would be useful in having. For example in the context of a drug-drug interaction database the process of acquiring a new datapoint is expensive and time consuming due to having to carry out new trials. Thus we would want the system to tell us what drug-drug combination experiment would help us improve our predictions the best.

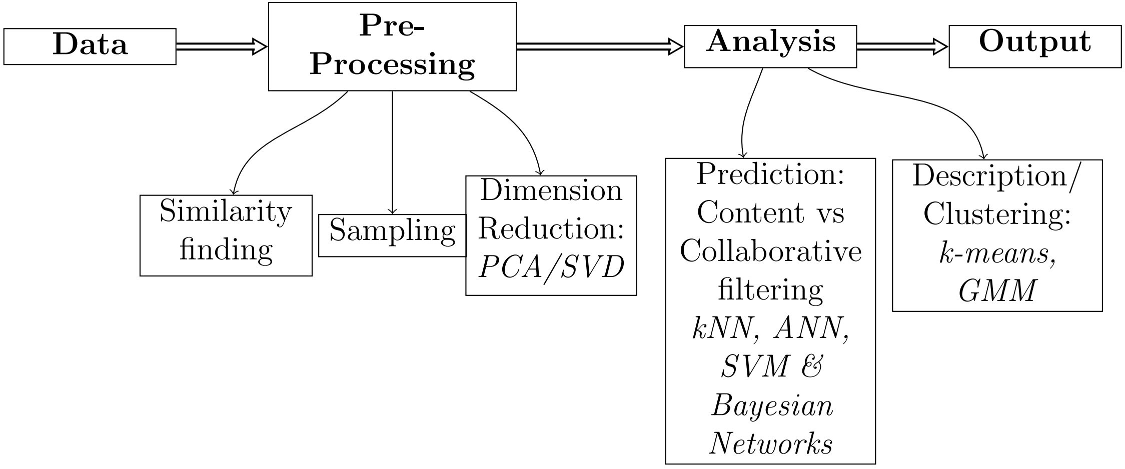
Also in databases such as the movielens one[1], complications arise in relation to memory and speed due to their large size. Movielens has 10 million ratings (and this database is just a subset of movielens' full one), other datasets can be in the order of billions as a Netflix user has rated 200 movies on average and has more than 30 million users making an expected 6 billion ratings. Thus going iteratively through each user to look for other similar users is inefficient if not impossible. To efficiently deal with this problem, the most informative users and items could be selected to form a smaller but equally useful subset. From this, recommendations could still be made but much faster.

1.2 Project Specification

A typical recommender system works by having multiple data processing stages and active sample selection is part of one of those steps (Figure 1.1, explained more in detailed in the Background section, provides a quick overview fo these steps). Thus the first aim is to build a working but flexible recommender system which would integration active sample selection. It is expected to base the recommender system on a collaborative filtering model called Probabilistic Matrix Factorization (PMF), which is based on the work of Salakhutdinov and Mnih [2007]. This is because current active sample selection work is often based on this model and this model performs well on large sparse datasets. A library or premade model will not be used as it is less flexible and does not have a useful

[1] http://grouplens.org/datasets/movielens/

Figure 1.1: A diagram of the steps for a typical recommender system with the various methods of achieving each step.


Data
Pre-Processing
Analysis
Output
Similarity finding
Sampling
Dimension Reduction: PCA/SVD
Prediction: Content vs Collaborative filtering kNN, ANN, SVM & Bayesian Networks
Description/ Clustering: k-means, GMM

learning process that will help give insight into the inner workings of a recommender system.

To test the recommender system sample datasets will be used. Several datasets have been chosen due to their popularity in recommender systems literature:

MovieLens is a movie recommendation website that makes subsets of its information(100 thousand, 1 and 10 million ratings in each respective dataset) publicly available for research purposes. Each available dataset is already set-up for use in recommender system testing, with cross-validation and test data already present.

DrugBank is a bioinformatics and cheminformatics database that combines detailed drug (i.e., chemical, pharmacological and pharmaceutical) data with comprehensive drug target data. A subset can be downloaded for easier processing and testing. As the data is biological and not user derived it is less likely to contain noise derived from indecisive users and fluctuations of opinions over time.

Active Learning Challenge Rather than one dataset, this is a set of datasets used explicitly for an active learning competition, sponsored by two journals Zurich [2010]. The datasets contained are :

- HIVA is a dataset aimed at predicting which compounds are active against the AIDS HIV infection.
- IBN_SINA is a dataset of arabic words in an ancient manuscript to facilitate indexing.
- ORANGE is a noisy marketing dataset. The goal is to predict the likelihood of customers to change network provider, buy complimentary or value-added products or services.

Initially the small version (100 thousand entries) of the MovieLens dataset will be used for development purposes and performance (with respect to error rate and computational intensity) will be measured on the larger (10 million entries) MovieLens dataset and the DrugBank one.

Once a working implementation of a recommender system is achieved (this will be done by checking for an acceptable prediction/error rate on the provided test set) focus will be on implementing the work of Sutherland et al. [2013], where his work is made available on his website.

Once this first step is achieved, different methods will be looked at to improve the accuracy or performance of this implementation - whichever one is determined to be the largest bottleneck. The idea would be to implement either custom ideas or ones present in other papers, such as Silva and Carin [2012]'s implementation.

In summary:

- *Software/language used*: Matlab
- *Data used*: Synthetic Data, MovieLens, DrugBank and the Active Learning Challenge datasets
- *Deliverable*: Active sample selection model which should be able to have a better prediction performance with less requested samples

1.3 Report Structure

Chapter 2

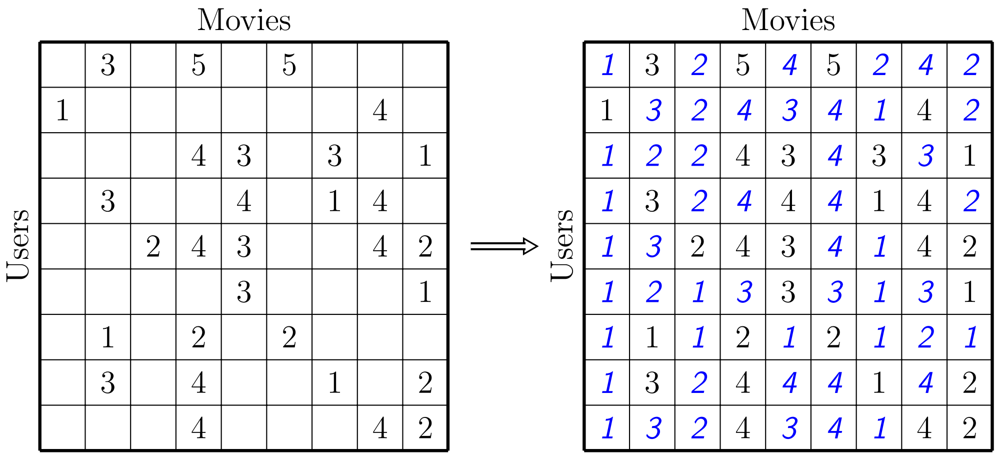
Recommender Systems

2.1 Basics

For any decent test of active sample selection a matrix completion algorithm must be used. The basic idea such a system is graphically described in figure 2.1. We have an incomplete matrix that tells us a certain amount of data about what movies each user likes or dislikes. From this we use a matrix completion algorithm to infer what the empty entries are. This thus allows a system to recommend potential movies a user may like or useful drug-target interactions that have not been tested out.

Many commercial systems for recommender systems currently exist. For example Amazon, YouTube, Google, Netflix, IMDb and Last.fm are but a small sample of websites using recommender systems to suggest films, ads, music or products to users. In fact Netflix is known for the "Netflix Prize", an open competition that awarded $1 million to a group of researchers that could create a recommender system achieving a 10% or more improvement on their previous one.

More formally, a recommender system database comes in matrix form, $R \in \mathbb{R}^{M \times N}$ with each entry containing a numerical quantifier of the relationship between the row item and column item. As not every entry is complete we can associate this to a mask matrix $Z \in \mathbb{R}^{M \times N}$, where $Z_{ij} \in \{0, 1\} \forall i, j$. 0 represents an unknown entry and 1 a currently known entry.


Movies
Users
Movies
Users

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

Incomplete User-Movie Matrix

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

Completed Matrix

Completed entries have their number represented in blue. Each movie is rated out of 5.

Figure 2.1: Sample Matrix Completion on Movie Data

2.2 Types of Recommender Systems

2.2.1 Overview

Recommender systems typically work by having a pre-processing and analysis/matching stage with Figure 1.1 showing the various stages. Pre-processing will typically be clustering the data into groups, reducing the dimensionality (for example by a process similar to SVD) or creating a subset of data that is more manageable. Active sample selection fits in here. Then the actual processing stage is where the empty samples are filled in. There can also be a post-processing stage where certain generated samples are selected either due to their usefulness or high score (i.e. a high predicted product score may be selected as part of a monetisation strategy).

There are several types of Recommender Systems:

Collaborative Filtering In this type of system the user is placed into subgroups that have similar taste. From this, it is expected that if user A has similar taste than user B on some products he is more likely to have the same one about different products.

Content-based filtering In this system features are learned about content (such as item color, film type, music genre etc ...) and a user's profile of their tastes is built, allowing products to be recommended by their features rather than group similarity.

Demographic Based In this system extra data that groups columns or users into categories (based on age or sex for example) is used to recommend users items in their respective category.

Each of the above methods can be done in several different ways and have varying performance. It would thus make sense to select the Netflix prize winner algorithm. However the winning proposal, by a team named "BellKor's Pragmatic Chaos", consisted of an ensemble of various recommender systems. This increases complexity and is not very practical to work with. Thus an algorithm that performs well but simple (and easily built upon) is preferred. This explains the choice of the Probabilistic Matrix Factorisation algorithm. Its RMSE on the Netflix dataset is 0.8861, about 7% better than Netflix system[1]. Bayesian Probabilistic Matrix Factorisation Salakhutdinov and Mnih [2008], an extension of PMF, will also quickly be covered due to its good performance on very sparse datasets.

2.2.2 Probabilistic Matrix Factorisation

PMF works by assuming that each input sample comes with Gaussian noise. If we decompose R into two matrices $\mathrm{U}^{\mathrm{T}}\mathrm{V}$ we can describe each item, with index ij as $x_{ij} = \mathbf{u}_i^T \mathbf{v}_j + \epsilon_{ij}$ where $\epsilon_{ij} \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Equations 2.1 and 2.2 provide a quick intuition into the decomposition. The reason we decompose the matrix in two matrices is motivated by the want to extract features from each column and row item. This can be done by performing what is called a latent aspect model, which essentially creates a column and row matrices composed of the features and weightings of these items.

$$R \backsimeq U^T \cdot V \tag{2.1}$$

[1] The goal of the Netflix prize was to achieve a performance increase of 10%

$$U^T \cdot V = \begin{bmatrix} u_{11} & u_{12} & \cdots \\ u_{21} & u_{22} & \cdots \\ u_{31} & u_{32} & \cdots \\ u_{41} & u_{42} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \cdot \begin{bmatrix} v_{11} & v_{21} & v_{31} & \cdots \\ v_{12} & v_{22} & v_{32} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{2.2}$$

We can choose an arbitrary number of features D to form the matrices $U \in \mathbb{R}^{D\times N}$ and $V \in \mathbb{R}^{D\times M}$. Essentially U and V contain the latent features of each row and column items, with U_i and V_j containing the row and column latent features. Assuming the Gaussian noise we can define the conditional distribution of R as:

$$p(R|U,V,\sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|U_i^T V_j,\sigma^2)\right]^{Z_{ij}} \tag{2.3}$$

Remember that Z is the mask matrix. i and j are the matrix entry coordinates. In addition to this we place Gaussian priors on row and column feature vectors:

$$p(U|\sigma_U^2) = \prod_{i=1}^{N}\mathcal{N}(U_i|0,\sigma_U^2\mathbf{I}), \quad p(V|\sigma_V^2) = \textstyle\prod_{j=1}^{M}\mathcal{N}(V_j|0,\sigma_V^2\mathbf{I}) \tag{2.4}$$

Now we assume row and column independence giving:

$$P(U,V|\sigma_U^2\mathbf{I},\sigma_V^2\mathbf{I}) = p(U|\sigma_U^2)p(V|\sigma_V^2)$$

We want to find the likelihood of U and V given the supplied parameters.

$$\begin{aligned} P(U,V|R,\sigma,\sigma_U^2\mathbf{I},\sigma_V^2\mathbf{I}) &= \frac{P(U,V,R|\sigma,\sigma_U^2\mathbf{I},\sigma_V^2\mathbf{I})}{P(R|\sigma^2)} \\ &= \frac{P(R|U,V,\sigma^2)P(U,V|\sigma_U^2\mathbf{I},\sigma_V^2\mathbf{I})}{P(R|\sigma^2)} \\ &= \frac{P(R|U,V,\sigma^2)p(U|\sigma_U^2)p(V|\sigma_V^2)}{P(R|\sigma^2)} \end{aligned}$$

We take the log likelihood of $P(U, V|R, \sigma, \sigma_U^2\mathbf{I}, \sigma_V^2\mathbf{I})$ to maximise it.

$$\ln(P(U, V|R, \sigma, \sigma_U^2\mathbf{I}, \sigma_V^2\mathbf{I})) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{M} Z_{ij}(R_{ij} - U_i^T V_j)^2 - \frac{1}{2\sigma_U^2}\sum_{i=1}^{N} \|U_i\|_{Fro}^2 - \frac{1}{2\sigma_V^2}\sum_{j=1}^{M} \|V_j\|_{Fro}^2 + C$$

Where C is a number not depending on U or V. From this we can get an error function to minimise (by inverting the signs).

$$E = \sum_{i=1}^{N}\sum_{j=1}^{M} Z_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N} \|U_i\|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \|V_j\|_{Fro}^2 \qquad (2.5)$$

With $\lambda_U = \frac{\sigma^2}{\sigma_U^2}$ and $\lambda_V = \frac{\sigma^2}{\sigma_V^2}$ being regularisation parameters. We can remove the mask matrix from the equation and create a cell specific error function:

$$e_{ij}^2 = (R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N} \|U_i\|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \|V_j\|_{Fro}^2$$

From e_{ij}^2 we can find U V satisfying $\arg\min_{U,V} E$ through simple gradient descent Lin [2007]. We use the simple but effective Widrow-Hoff learning rule:

$$\begin{aligned} U_{ik}^{t+1} &= U_{ik}^t - \mu \frac{\partial}{\partial U_{ik}}(e_{ij}^2) \\ &= U_{ik}^t + \mu(2e_{ij}V_{jk} - \lambda_U U_{ik}^t) \\ V_{jk}^{t+1} &= V_{jk}^t - \mu \frac{\partial}{\partial V_{jk}}(e_{ij}^2) \\ &= V_{jk}^t + \mu(2e_{ij}U_{ik} - \lambda_V V_{jk}^t) \end{aligned}$$

k is the feature index, as defined by D, μ is the learning rate and t is the iteration index. The steps above are repeated until convergence (as defined by a custom criteria) or a fixed number of iterations. This allows us to learn the features of each column and row. Once U and V are learned we can estimate the full matrix by:

$$\hat{R} = U^T V \tag{2.6}$$

For very large matrices where only a specific entry is needed a single entry can be predicted by $\hat{R}_{ij} = U_i^T V_j$.

2.2.3 Bayesian Probabilistic Matrix Factorisation

Bayesian Probabilistic Matrix Factorisation has been proposed as an extension of PMF Salakhutdinov and Mnih [2008]. It places Gaussian priors on U and V and Gaussian-Wishart priors on the row and column hyperparameters.

$$p(U|\mu_U, \Lambda_U) = \prod_{i=1}^{N} \mathcal{N}(U_i|\mu_U, \Lambda_U^{-1})$$
$$p(V|\mu_V, \Lambda_V) = \prod_{j=1}^{M} \mathcal{N}(V_j|\mu_V, \Lambda_V^{-1})$$

Λ^{-1} is the precision matrix and μ is the mean of each feature vector. $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$ are defined as the row and column hyper parameters.

$$p(\Theta_U|\Theta_0) = p(\mu_U|\Lambda_U)p(\Lambda_U) = \mathcal{N}(\mu_U|\mu_0, (\beta_0\Lambda_U)^{-1})\mathcal{W}(\Lambda_U|W_0, \nu_0)$$
$$p(\Theta_V|\Theta_0) = p(\mu_V|\Lambda_V)p(\Lambda_V) = \mathcal{N}(\mu_V|\mu_0, (\beta_0\Lambda_V)^{-1})\mathcal{W}(\Lambda_V|W_0, \nu_0)$$

We have $\mathcal{W}$ as the Wishart distribution with ν_0 degrees of freedom and $W_0 \in \mathbb{R}^{D \times D}$. $\Theta_0 = \{\mu_0, \nu_0, \Lambda_0\}$ is defined with $\mu_0 = 0$, $\nu_0 = D$ and $W_0 = \mathbf{I}_{D \times D}$. After rearranging we end up with:

$$P(U, V|R, \Theta_U, \Theta_V) = \int\int P(R|U, V)P(U, V|R, \Theta_U, \Theta_V)P(\Theta_U, \Theta_V|\Theta_0)d\Theta_U d\Theta_V$$

The above equation cannot be resolved analytically and approximate methods must be used. Thus we resort to a MCMC method, Gibbs sampling. This allows us to generate multiple approximations of U and V feature matrices and then get a better approximation out of it. The outline of Gibbs sampling for BPMF is as follows:

1. Initialise U^1, V^1
2. For $t = 1 \dots T$
   - Sample hyperparameters Θ_U and Θ_V
   - For $i = 1 \dots N$ sample row features in parallel:

     $$U_i^{t+1} \sim p(U_i^t | R, V^t, \Theta_U^t)$$

   - For $j = 1 \dots M$ sample column features in parallel:

     $$V_j^{t+1} \sim p(V_j^t | R, U^t, \Theta_V^t)$$

More details on BPMF and can be found in the appendix section A.2.

For the purpose of this project it is mainly useful to remember that BPMF performs well on very sparse matrices - in part due to the MCMC sampling process which provides a good approximation of the true probability distribution of data points.

2.2.4 Kernelized Probabilistic Matrix Factorization

Another variant of PMF that was tested was KMPF Zhou et al. [2012]. The main ideas to take from it are that it functions like PMF but with the crucial difference that a latent Gaussian process prior is placed over all rows and columns on top of a latent vector on each row and column. This means that it captures side information through kernel matrices that contain the covariance information. The intended use of these kernels is to use separate graph data, such as the social connection between users. However in the case this data does not exist a certain correlation between columns and rows can be assumed, which still results in good performances.

Performance of KPMF is best on very sparse data due to its ability to *smooth* out data from its assumption of inter-row and inter-column correlation. In fact KPMF can work very well on empty rows or columns by filling them in with the most likely values - something PMF and BPMF struggle with. Once more data on the dataset is available KPMF's performance can even be worse than the traditional model due to forcing these correlations. In the context of active sample selection, where more cells become available over time, this could cause a problem in determining the usefulness of each new discovered cell.

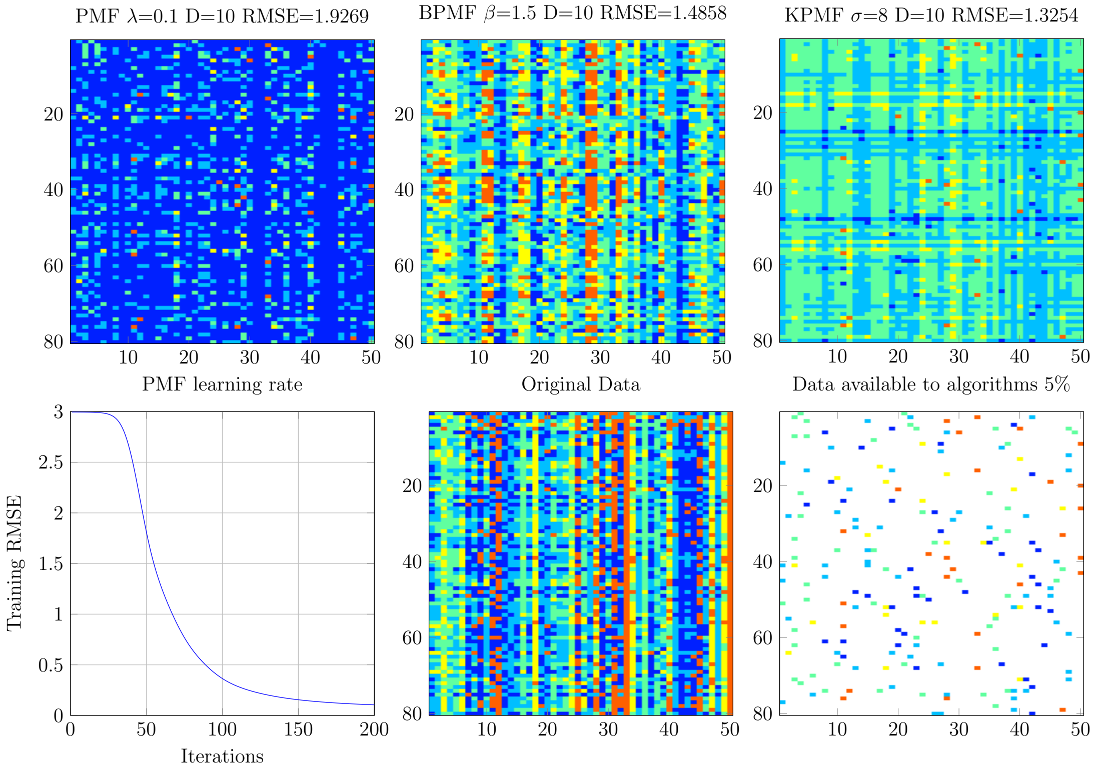
2.3 Evaluation and comments

2.3.1 Performance test

To understand what may be best to use we test each type of algorithm on a dataset. This allows to see its advantages and weaknesses. For the first test we look at figure 2.2.

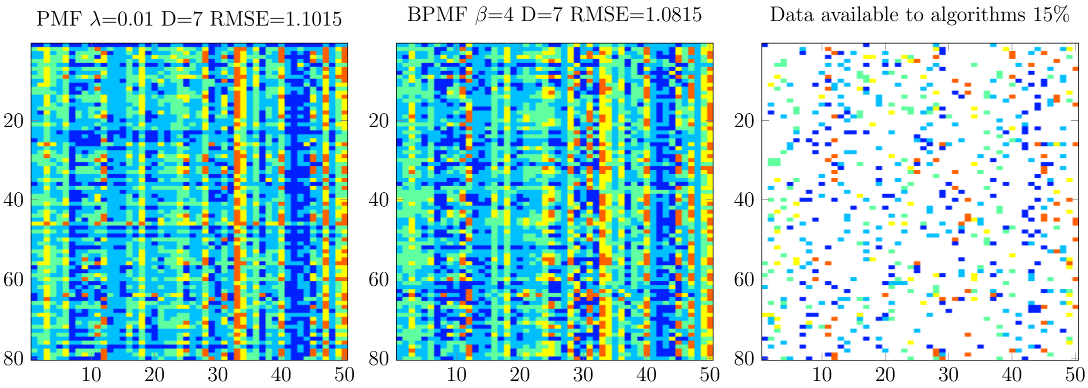
For this figure data was randomly generate by generating 2 matrices $U \in \mathbb{R}^{80\times 5}$ and $V \in \mathbb{R}^{50\times 5}$, that is 5 latent features. A random mask allowing the variants of PMF to only access 5% of the data was made and used for matrix completion. The Root Mean Square Error(RMSE) of the predicted data was used as a measure of success. According to RMSE the best performing algorithm is KPMF, which was on average 1.3 off the real value. However all it has effectively done is predicted the mean with some amount of variation for all unknown values. We can see the effect of Gibbs sampling from BPMF as it captures the data distribution better than PMF or KPMF. Despite this it has a higher RMSE. This is one weakness of RMSE as some data sets may penalise more for guessing a value off the mean than one nearer to the mean. Finally PMF is seen to perform the worse, simply failing to predict many entries having very little information.

Making more data available, such as 15 % of the dataset lead to an improvement for both BPMF and PMF as seen in figure 2.3. PMF experienced the largest RMSE decrease - something to keep in mind for sample selection. KPMF's RMSE



Each colour is made to represent a rating out of 5. Dark blue represents a 1 and red a rating of 5. The mean is represented in green.
A row could be used to represent a user and a column a movie.

Figure 2.2: Rating Predictions on synthetic data 5% complete



For KPMF: RMSE=1.1867 for $\sigma = 10$

Figure 2.3: Rating Predictions on synthetic data 15% complete

has also improved however its output did not look any better.

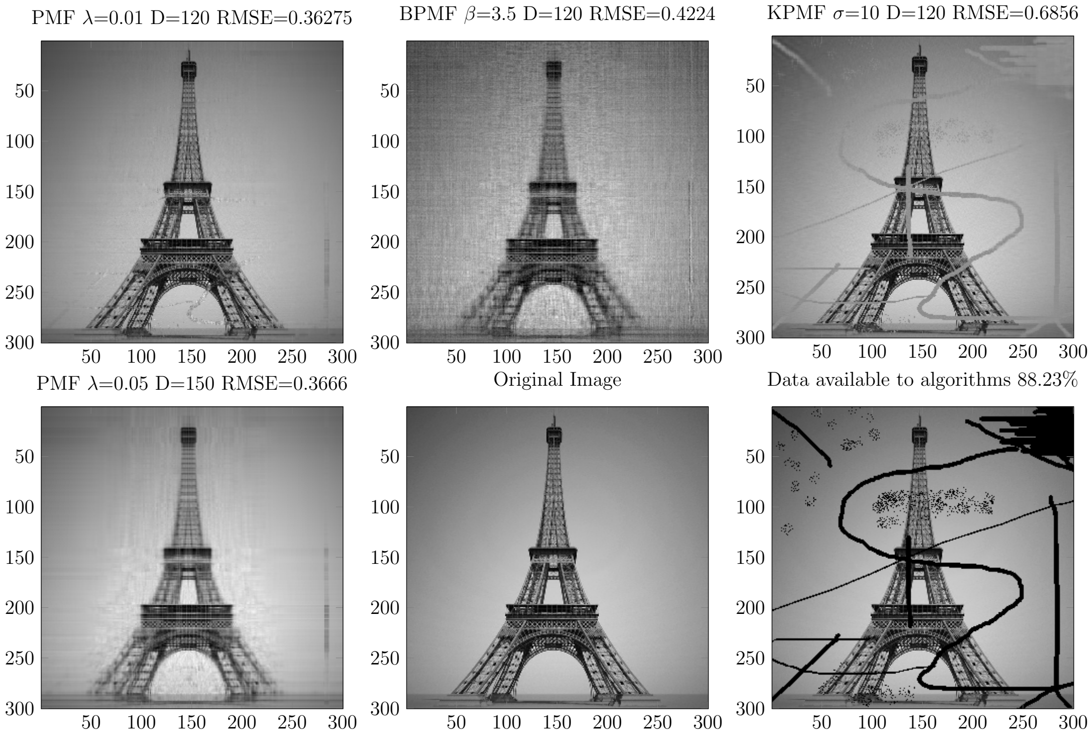

PMF λ=0.01 D=120 RMSE=0.36275
BPMF β=3.5 D=120 RMSE=0.4224
KPMF σ=10 D=120 RMSE=0.6856
PMF λ=0.05 D=150 RMSE=0.3666
Original Image
Data available to algorithms 88.23%
50
100
150
200
250
300

Image corrupted by drawing over and virtual spray paint

Figure 2.4: Image Restoration

In an effort to better understand each algorithm an image restoration trial was done. This is possible due to the rows and columns (pixel wise) in images to exhibit features about them. For example a column may have features indicating 45% sky hue, 25% Eiffel tower hue and 30% ground hue with a row another having its own distribution. Multiplying both together would give the best estimate of the pixel at the specific row-column combination. In figure 2.4 a picture with drawing over was supplied to each PMF variant. This was equivalent with supplying 88% of the data. This time the scales are reversed, with PMF coming out as the clear winner. Not only is the image very similar to the original but little corruption in the overall restoration happened. BPMF failed at this task and distorted the picture overall, blurring it up[1]. KPMF was able to restore the

[1] In a practical application of this algorithm it would be possible to only replace the unknown

know pixels correctly but did not do a good job of predicting the missing ones, choosing to predict the mean color instead. KMPF is supposed to provide superior image restoration if supplied the correct kernel Zhou et al. [2012]. However, despite providing a diffusion kernel indicating that each pixel is correlated to its neighbouring ones, KPMF did not live up to the task.

For the purpose of this project mainly PMF and BPMF will be used as they showed the most promising performances and can be easily customised and built upon. This is especially useful as active sampling builds on top of recommender systems.

Note that the used datasets, synthetic and movielens, have their values ranging from 1 to 5. Thus a RMSE of 2 would mean that predictions are off by 2 stars on average. A good RMSE would thus be under 1, meaning movie predictions are only off by 1 star or less. When possible, other datasets used will be normalised between 1 and 5 to have the RMSE somewhat comparable.

2.3.2 Choosing parameters and cleaning up data

All the above PMF variants all relied on one or many parameters. For example PMF uses[1] λ. These parameters nearly always affect the system performance and choosing them correctly is essential. While it is possible optimise them as such:

$$\underset{U,V,\lambda}{\arg\min}\, E(U,V,\lambda)$$

This is not a good idea. Indeed λ is a regularisation parameter needed to allow the function to correctly generalise[2] and if optimised in a similar way it would tend to 0 defeating its purpose, see A.1.2. It would essentially perfecty (over)fit the data but not perform predictions correctly.

Other parameters, such as D (the number of features), could be optimised in a similar way but this increases complexity necessarily. Instead these parameters are usually tweaked by hand and this is what has been done here as this is not the main focus of research.

pixels.

[1] This is actually a combination of λ_U and λ_V from equation 2.5

[2] In an optimisation context it can be seen as a Lagrange multiplier that enforces a constraint

2.3.2.1 Learning rate μ

μ is an essential parameter to most iterative learning algorithms. It is appropriately called the learning rate as it defines the rate at which parameters are learnt. If we imagine a convex curve and we are located at a random location on it μ essentially defines how far we can go on each iteration to get closer to the minimum. Should it be too small a very high number of iterations will be needed to reach satisfactory performance. A too large learning rate may fail to converge due to the steps "jumping around" a minimum but never reaching it - alternatively it could simply overflow with the error increasing at each iteration.

2.3.3 Pre-Processing datasets

Before processing data, it is often advisable to pre-process it. This means cleaning up the data in ways that ensure good performance.

2.3.3.1 Normalisation

The first step that is often taken is to normalise data between two values - by convention 0 and 1 [1]. This ensures that some parameters used, such as regularisation or precision, need not be tweaked a lot when switching between datasets. Additionally when performing gradient descent (as in section 2.2.2) over multiple dimensions it is preferable the error for each datapoint is in a similar order of magnitude.

2.3.3.2 Discrete Data

Discrete data is data that only takes a fixed number of values, which is applicable to movie-user datasets where a 5 star rating is given. To process these data is first be normalised between 0 and 1, 0 representing 1 and 1 representing 5. From this, parameters are learnt and data can be predicted. Placing the values back between 1 and 5 will result in non-integer values. Expectedly values are rounded up or down accordingly. Since 3.1 and 3.45 will both be rounded down to 3 we can

[1] For other types of learning algorithms normalisation may be between $[-1, 1]$ and zero meaned.

use the decimal as a measure of uncertainty, that is 3.1 has a greater probability of actually being 3 rather than 3.45. While this is not a fail safe system the deviation from the discrete values can be useful for measuring uncertainty in the model.

2.3.4 Online vs Offline learning

Usually data will be fed to an algorithm and parameters returned to be used for prediction. However there are instances when new data points will be made available and a better performance will be achieved by retraining the parameters with the new data. When a model is retrained from scratch it is said to be done offline. Doing it online means training it once initially and then only carrying out a few steps to update the model when a new sample is received. Collaborative Filtering Algorithms, such as PMF, can be made online Ling et al. [2012] by first training U and V for initial data and then updating these parameters at each incoming sample.

2.3.5 Test and Validation dataset

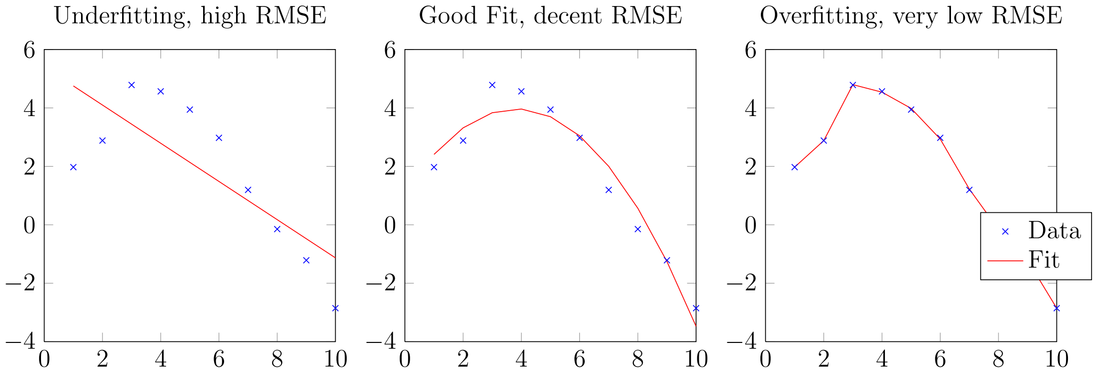


Figure 2.5: Examples of different model fitting

From section 2.2.2 we see that learning from a dataset involves minimising an error function. The problem with this is that we are subject to a “tunnel vision” where only the given data is considered while minimising the error function. A model that only fits the test data, as in figure 2.5, can overfit it and not generalise well. This is one of the reason the regularisation parameters λ_U and λ_V exist, as they avoid U and V becoming too large and only being specific to the test data. Another way to avoid this is to consider splitting up the dataset into a test set and a validation set. The test set will be used for training. However at each parameter step update, we can calculate the RMSE from the validation data - i.e. data not used for training. As soon as a step is found to lead to an increase of validation RMSE we can infer that the model is potentially starting to overfit and stopping training of parameters is preferable.

2.3.6 Data quality

Another issue with any machine learning system is that of data quality. Any inaccurate or badly generated data can prove to be a major bottleneck in the performance of machine learning systems. Many datasets, especially when they come from human made sources (such as film ratings or opinion polls), have a lot of noise. Noise is essentially the ϵ_{ij} component which we referred to earlier and creating a model assuming a certain amount of noise is one of the first steps that can be taken to deal with this issue. Note that only incidental noise rather than intentional noise can be dealt with (intentional noise would be defined as data that is maliciously entered or be subject to a strong bias for external reasons). Incidental noise could be reduced to a certain extent by grouping items or users into categories and smoothing out the similar ratings. This and similar techniques can introduce bias in the system and their use is only suggested if it translates to real-world performance increase. Other methods to reduce noise include re-querying a particular sample (Toledo et al. [2013] i.e. asking a user to re-rate an item, or performing the same experiment again), however this will not be done as the aim of this project is to reduce requeries as well as the fact that we cannot just ask for a data-point to be re-evaluated in our datasets.

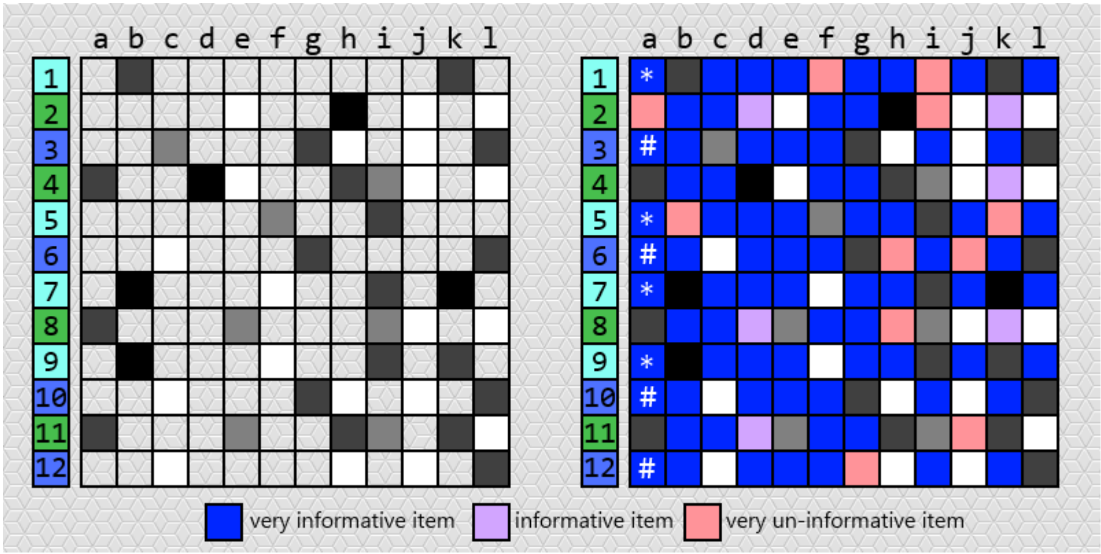
To avoid these issues most simulations will be carried on synthetic data, which is less affected by noise.

Chapter 3

Active Sample Selection

3.1 Background

Now that we have a good understanding of what matrix factorisation can do we focus on active learning. Typical collaborative filtering system applications are done online. This means that once a base model is learnt it evolves over time by adding new samples. For example Amazon would gradually add samples of products a user has rated, improving its model. In others contexts a research laboratory will be conducting experiments on drug-biological targets and gradually adding the results of each experiment to a database. All of these situations involve a new row-column combination being sampled. Choosing the new sample can be a matter of human judgement but cannot guarantee model improvement. Active sample selection is the process of intelligently selecting a new sample that best increases the models performance.



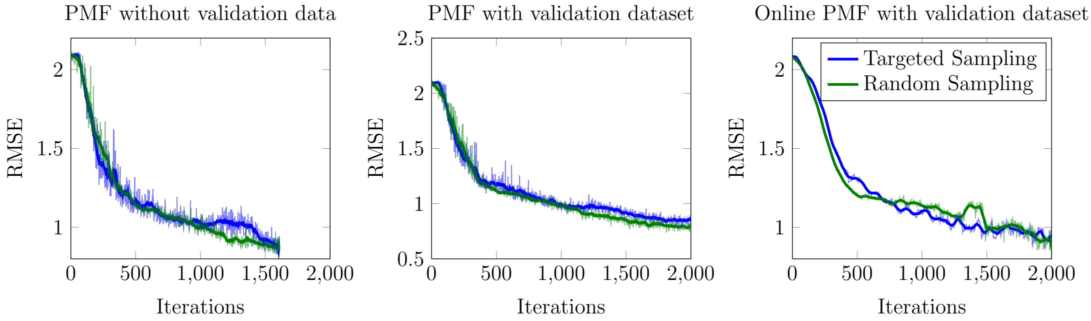
Users are represented in rows, and there has been 3 type of users defined, dark blue, light blue and green. Items are represented in the columns by letters. User item pairs that are rated are represented by a shade of grey and unrated squares are left blank.

Figure 3.1: Diagram of a simplified user-item matrix used for recommender systems

Figure 3.1 gives us a very simplified graphical explanation of what an active learning system can achieve. To the left we have a sparse user-item matrix, each coloured square represents a rating, with the intensity representing a rating or interaction (for example black could represent 4, dark grey 3, light grey 2 and white 1). To simplify, we can assume that the system has recognised that there are 3 types of user groups, light blue, dark blue and green. Each of these user groups have the same interests. Using a film database analogy, a green user could be an action movie fan whereas light blue users could be comedy aficionados. Thus if it needed to predict what the rating `a2` would be, extra sampling would not really be useful as we know `a4`, `a8` and `a11`, which are all part of the green group. Knowing `a2`, along with `b5` and other pink ratings (from the right matrix) is of little use. However knowing just one of `a1-5-7-9` and `a3-6-10-12` is very useful as it gives us a rough idea as to what light and dark blue users think of item `a`. Collecting samples for users we already have a reasonably good profile of only helps us improve the certainty of certain ratings rather than be able to say something new about the dataset. Again it may be the case that the item rating isn't the same for one user group but this is an idealistic scenario.

3.2 Measuring Effectiveness

Determining whether or not a sampling method is effective or not is mainly a matter of seeing how well it impacts the performance. For example if the average sample added to a model leads to a 0.01 RMSE decrease and we can select the ones leading to 0.02 RMSE decrease on average then it can be said that this is effective. The benchmark case is defined to be random sampling, that is choosing a new sample at random. As random sampling may select an informative sample just as well as a less useful one, many random sampling trials must be done and averaged together to see the expected random performance. The benchmark RMSE for random sampling can be seen in the appendix figure B.2. This is because one random sampling instance may outperform a poor active sampling method. Figure 3.2 shows a case of random sampling performing as well or outperforming an active sampling method (in this case this is a basic one developed for this project called minimum knowledge search).



Each iteration involves the discovery of one new sample. Dataset is the synthetic one from figure 2.2.

Figure 3.2: Active Sampling trial for the “minimum knowledge” selection

Another measure of the active sampling algorithm is to compare the area under the curve of the RMSE over the number of discovered samples. This measure gives an idea of what total advantage the targeted sampling may have.

3.3 Minimum Knowledge Search

Before looking at advanced sampling techniques a basic one was made to illustrate the concepts and basic increase in performance possible.

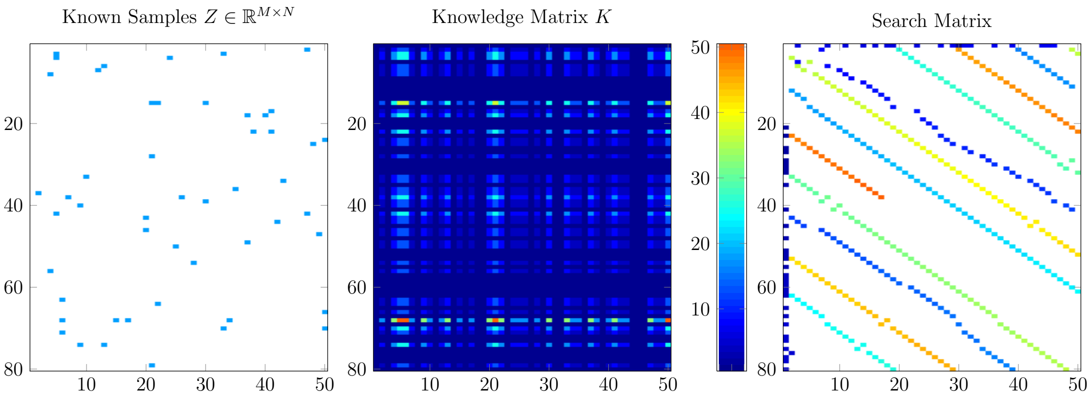
3.3.1 Algorithm

For any matrix factorisation problem we have the mask matrix $Z \in \mathbb{R}^{M \times N}$, with elements being 1 for every known value and 0 for every unknown value (values in the validation set would be also 0).

Algorithm 1 Minimum Knowledge Search algorithm

```
procedure MinKnowSearch($Z$)                 ▷ The mask matrix as input
    $\mathbf{a} \leftarrow$ meanrow($Z$)      ▷ Mean of rows, $\mathbf{a} \in \mathbb{R}^N$
    $\mathbf{b} \leftarrow$ meancol($Z$)      ▷ Mean of columns, $\mathbf{b} \in \mathbb{R}^M$
    $K \leftarrow \mathbf{a} \cdot \mathbf{b}$   ▷ $K \in \mathbb{R}^{M \times N}$
    $x, y \leftarrow index\_of\_min(K)$
    while $Z(a, b) == 1$ do                   ▷ Also check for validation mask
        $x, y \leftarrow next\_min\_index(K)$
    end while
    return $x, y$                             ▷ Return $x, y$ that has least knowledge
end procedure
```

Knowing a bit about each column and row is a good first step to discover more about a matrix. This is the motivation of creating a heatmap of what we know about each cell. To do this we take the mean of the row and columns of Z. Say $\mathbf{a} = meanrow(Z)$ and $\mathbf{b} = meancolumn(Z)$. This defines the amount known for each row and column - i.e. if $\mathbf{a}_i = 0$ then there is no known sample of row i, if $\mathbf{b}_j = 1$ then we know all samples of column j. From this we can get a knowledge matrix $K = \mathbf{a} \cdot \mathbf{b}$, which acts as a heatmap of what is known of R. From this we can find the cells with the minimum values and target them. There will often be multiple cells of lower value but not all may be available. For example it may be impossible to sample them or it may be a cell in the validation set. For this reason we select the lowest compatible cell. The full algorithm is described in algorithm 1.



The darker the colour in the knowledge matrix, the least is known about that cell due to the column-row combination.

Figure 3.3: Diagram of initial parameters and search path

Figure 3.3 illustrates the way minimum knowledge search works. From the known samples we see that the knowledge matrix has many "low knowledge" areas (in dark blue) to select from. A sequence of 200 sample selections is shown on the rightmost image. The first samples are blue in color, tending to red as the final samples are targeted (as shown by the colour legend to the left of it). As we see it selects the least known elements in the first row then column first. The pattern is due to selecting the very first possible least known element - a variant would be to randomly select an element to potentially try and get a temporary advantage.

Note: Other variants of this algorithm that were tried included selecting the minimum suitable index of **a** and **b** - i.e. the coordinates that intersect with the least known row and column. However this technique did not perform as well due to not seeking to maximise one row and column first at the start. The random minimum knowledge matrix selection variant also suffers from this.

3.3.2 Notes on performance

... and some more ...

3.3.3 Comments

The minimum knowledge search algorithm does not take the known matrix elements, nor the predicted ones, into consideration. These also contain information in themselves and could greatly help with the selection of better samples. Being dataset agnostic is an weakness of minimum knowledge search as dataset properties can greatly impact performance. For example a dataset where a row and column that has most of the less useful elements known but very useful unknown elements will not be targeted until very late on, potentially giving the random sampling an edge. Minimum knowledge search only really works on datasets where useful elements are not found in clusters.

Algorithms presented later on will take advantage of the dataset to try and get better performance.

Chapter 4

Advanced Sample Selection

4.1 Advanced Sample Selection

And now I begin my third chapter here ...

4.1.1 first subsection in the First Section

... and some more

4.1.2 second subsection in the First Section

... and some more ...

4.1.2.1 first subsub section in the second subsection

... and some more in the first subsub section otherwise it all looks the same doesn't it? well we can add some text to it ...

4.1.3 third subsection in the First Section

... and some more ...

4.1.3.1 first subsub section in the third subsection

... and some more in the first subsub section otherwise it all looks the same doesn't it? well we can add some text to it and some more and some more and some more and some more and some more and some more and some more ...

4.1.3.2 second subsub section in the third subsection

... and some more in the first subsub section otherwise it all looks the same doesn't it? well we can add some text to it ...

4.2 Second Section of the Third Chapter

and here I write more ...

Chapter 5

My Conclusions ...

Here I put my conclusions ...

Appendix A

A.1 Probabilistic Matrix Factorization

A.1.1 Full derivation

A.1.2 λ as a Lagrange multiplier

$$\begin{aligned} \text{maximize} \quad & f(x, y) \\ \text{s.t.} \quad & g(x, y) = c \\ \Lambda(x, y, \lambda) = & f(x, y) + \lambda \cdot (g(x, y) - c) \end{aligned}$$

A.2 Bayesian Probabilistic Matrix Factorization

The exact details and sample code can be found on http://www.cs.toronto.edu/~rsalakhu/BPMF.html and the published paper.

A.3 Synthetic Data Generation

Appendix B

B.1 Test Data

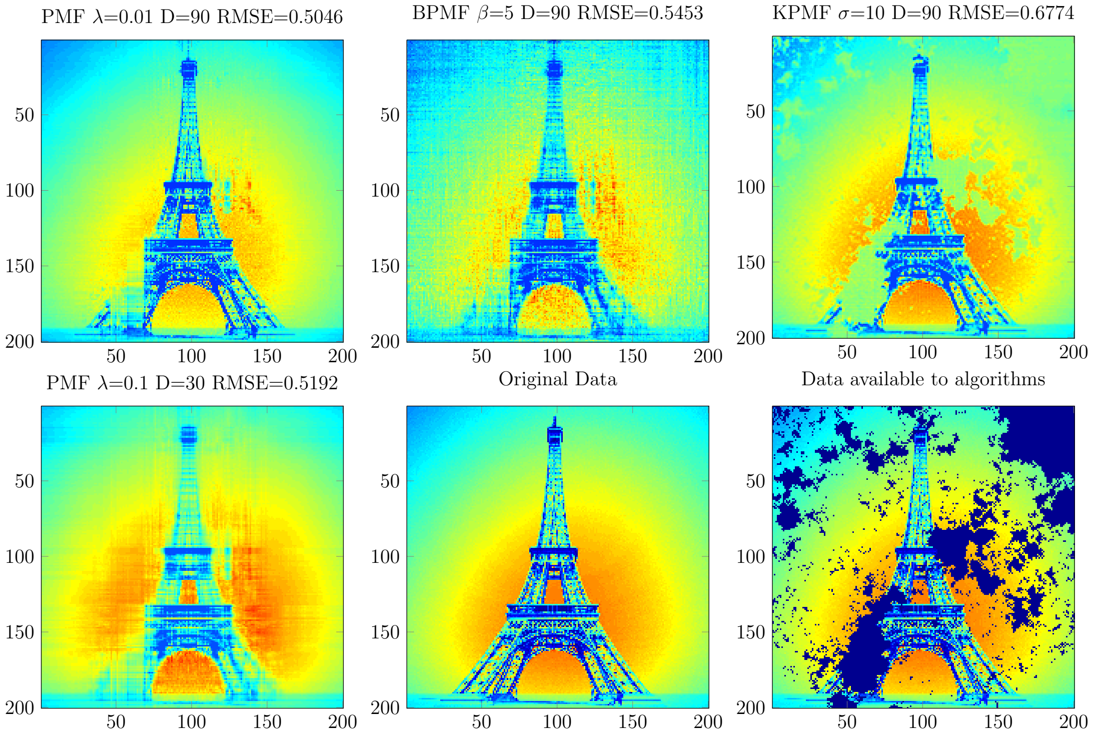
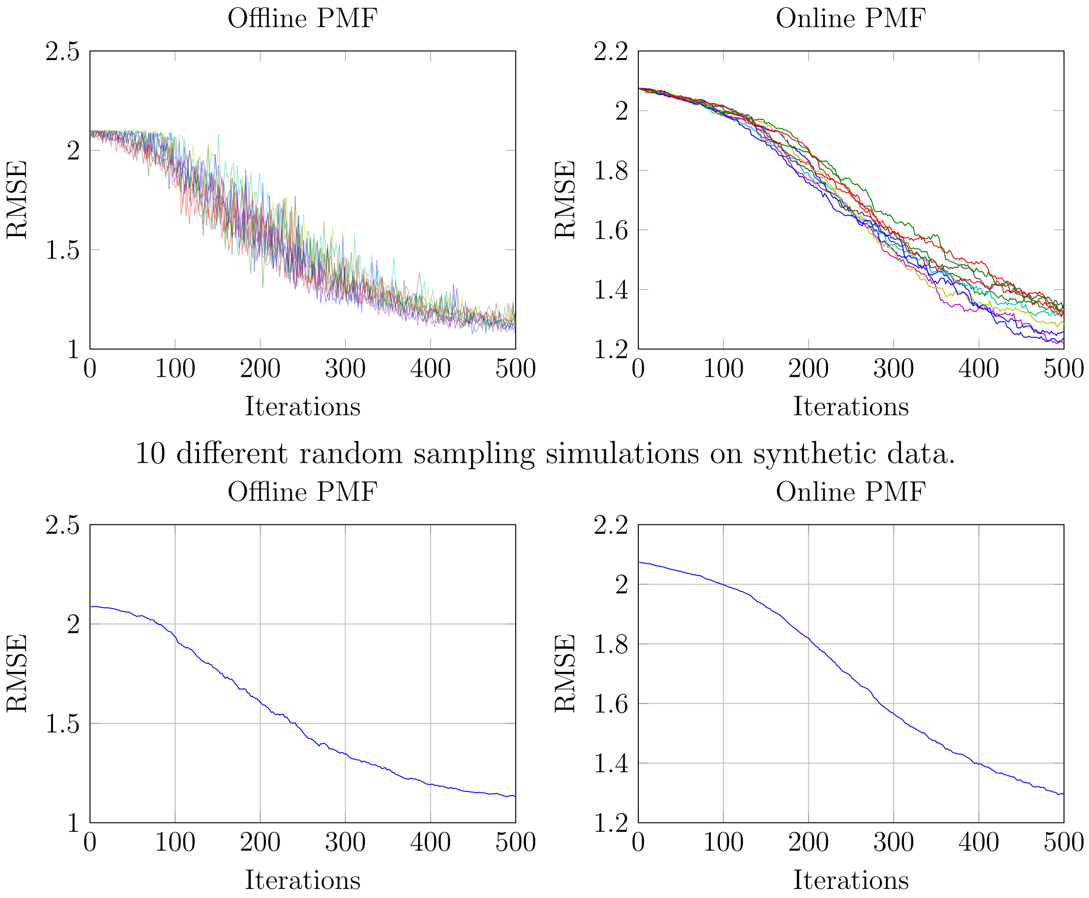


Figure B.1: Image Restoration with Matrix Factorisation with 75% complete data



10 different random sampling simulations on synthetic data.

Average of 10 simulations.

Original data complete at 1.25% (with another 1.25% as validation data)
$\lambda = 0.01$

Figure B.2: Random Sampling Performance

References

Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. In *Neural Computation*, pages 2756–2779, 2007. URL http://dx.doi.org/10.1162/neco.2007.19.10.2756. 9

Guang Ling, Haiqin Yang, I. King, and M.R. Lyu. Online learning for collaborative filtering. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, June 2012. doi: 10.1109/IJCNN.2012.6252670. URL http://dx.doi.org/10.1109/IJCNN.2012.6252670. 17

Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.6198. 2

Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*, pages 880–887, 2008. URL http://dx.doi.org/10.1145/1390156.1390267. 7, 10

Jorge Silva and Lawrence Carin. Active learning for online bayesian matrix factorization. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 325–333, 2012. URL http://doi.acm.org/10.1145/2339530.2339584. 4

Dougal J. Sutherland, Barnabas Poczos, and Jeff Schneider. Active learning and search on low-rank matrices. In *ACM SIGKDD*, 2013. URL http://www.autonlab.org/autonweb/21710/version/2/part/5/data/kdd-13.pdf. 4

Raciel Yera Toledo, Luis Martinez Lopez, and Yailé Caballero Mota. Managing natural noise in collaborative recommender systems. In *IFSA/NAFIPS*, pages 872–877, 2013. URL http://dx.doi.org/10.1109/IFSA-NAFIPS.2013.6608515. 18

Tinghui Zhou, Hanhuai Shan, Arindam Banerjee, and Guillermo Sapiro. Kernelized probabilistic matrix factorization, 2012. URL http://dx.doi.org/10.1137/1.9781611972825.35. 11, 15

ETH Zurich. Active learning challenge, 2010. URL http://www.causality.inf.ethz.ch/activelearning.php?page=datasets. 3