Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2014

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

Acknowledgements

I would like to thank my project supervisors Dr Wei Dai and Dr Moez Draief for their advice and support throughout the progression of this project.

I would also like to thank my friends for their support throughout this final year at Imperial.

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. Existing active sampling techniques currently differ in their accuracy and performance. A technique is usually considered to perform well when giving it extra samples leads to a lower error 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 and look-ahead methods are also investigated. An algorithm of good performance and complexity is also proposed. 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, and requesting the most informative is very useful.

Contents

List of Figures

List of Abbreviations

BPMF	Bayesian Probabilistic Matrix Factorisation
CF	Collaborative Filtering
CKS	Clustered Knowledge Search - Active sampling algorithm described in section 4.1
GMM	Gaussian Mixture Model - A probabilistic model for representing the presence of subsets within an overall dataset
K-Means	K-Means Clustering - A popular algorithm for cluster analysis, which separates N data points into K clusters around centroids
KMPF	Kernelized Probabilistic Matrix Factorisation
MCMC	Markov chain Monte Carlo - A means of approximating a probability distribution
MKS	Minimum Knowledge Search - Active sampling algorithm described in section 3.5
MMMF	Maximum Margin Matrix Factorization - A collaborative prediction algorithm using low-norm instead of low-rank factorizations
MN-V	Matrix-Normal Variational framework - Variational inference of the probability distribution of data in a matrix assuming Gaussian distribution
PMF	Probabilistic Matrix Factorisation
PPMF	Parametric PMF - PMF and BPMF hybrid, uses variational inference for its learning process
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[1] and instead seeks to build upon these systems, considering them to be a black box. What active

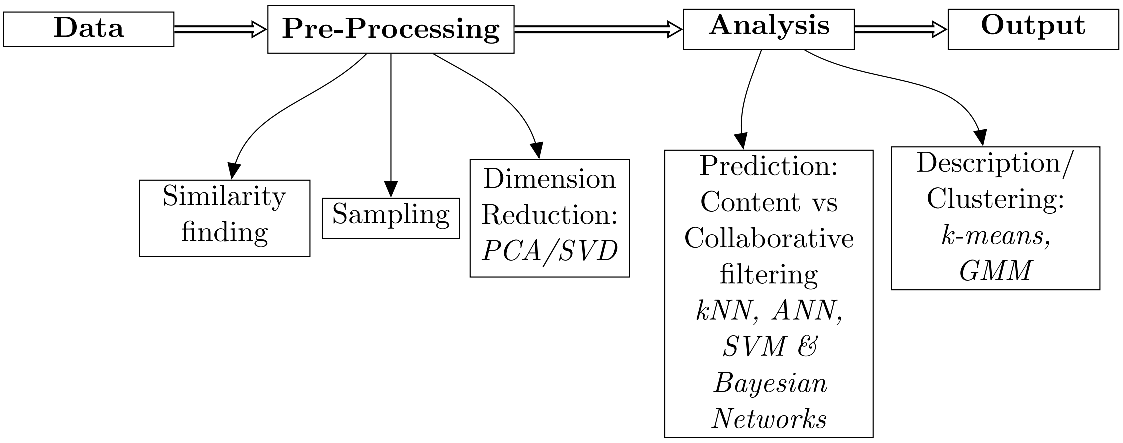
[1] We will however be exploring many non-agnostic active learning algorithms relying on a specific completion technique

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

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*

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

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

Matrix Factorization (PMF), which is based on the work of [11]. 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 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 identified 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 [22]. 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.

Small versions (100 thousand entries) of the datasets will be used for development purposes and performance (with respect to error rate and computational intensity) and larger scale use is used for further testing.

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.[15], 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 Jorge Silva's[14] implementation.

In summary:

- *Software/language used*: Matlab
- *Data used*: Synthetic Data, Images, MovieLens, DrugBank and the Active Learning Challenge datasets
- *Deliverable*: Active sample selection model which should be able to have a better prediction performance than random sampling, ideally with better time complexity.

1.3 Report Structure

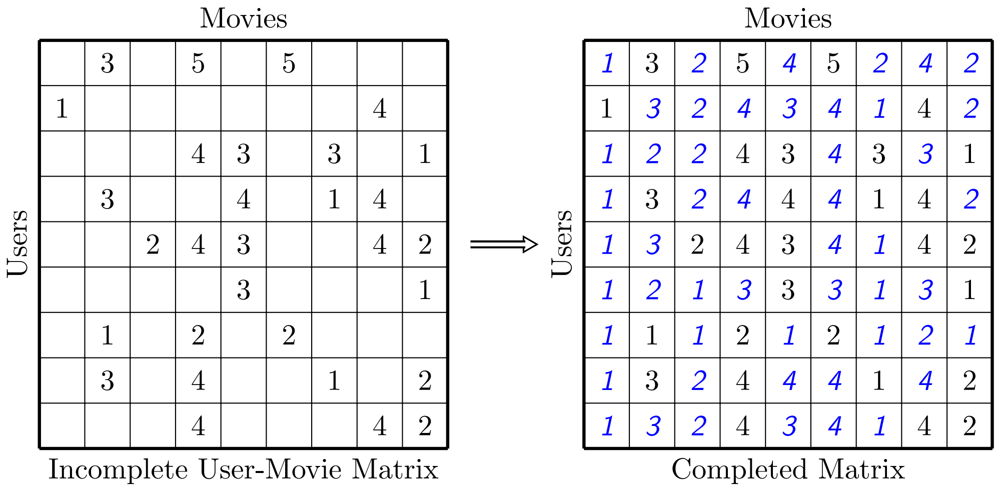
The report will first give an overview of the various recommender systems (matrix factorisation in particular) with a review of their performance. Given an insight into what matrix factorisation can achieve the concept of active sampling will be introduced by the use of a very basic algorithm. From this a more advanced algorithm is constructed and compared to other already existent algorithm. Performance of each algorithm is compared.

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.



	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

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

Figure 2.1: Sample Matrix Completion on Movie Data

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.

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 sub-groups 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 [12], an extension of PMF, will also quickly be covered due to its good performance on very sparse datasets. Finally it decomposes a matrix into two feature matrices which turn out to be very useful in the goal of active sample selection.

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.

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

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

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

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

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.

$$\begin{aligned}
\ln(P(U, V|R, \sigma, \sigma_U^2\mathbf{I}, \sigma_V^2\mathbf{I})) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}\sum_{j=1}^{M} Z_{ij}(R_{ij} - U_i^T V_j)^2 - \frac{1}{2\sigma_U^2}\sum_{i=1}^{N} \|U_i\|_{Fro}^2 \\
-\frac{1}{2\sigma_V^2}\sum_{j=1}^{M} \|V_j\|_{Fro}^2 + C
\end{aligned} \tag{2.5}$$

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).

$$E = \sum_{i=1}^{N}\sum_{j=1}^{M} Z_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{N} \|U_i\|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{M} \|V_j\|_{Fro}^2 \tag{2.6}$$

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 [7]. 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:

$$\hat{R} = U^T V \tag{2.7}$$

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 [12]. It places Gaussian priors on U and V and Gaussian-Wishart priors on the row and column hyperparameters. A graphical representation to compare to PMF is found in figure 2.2.

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

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.

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

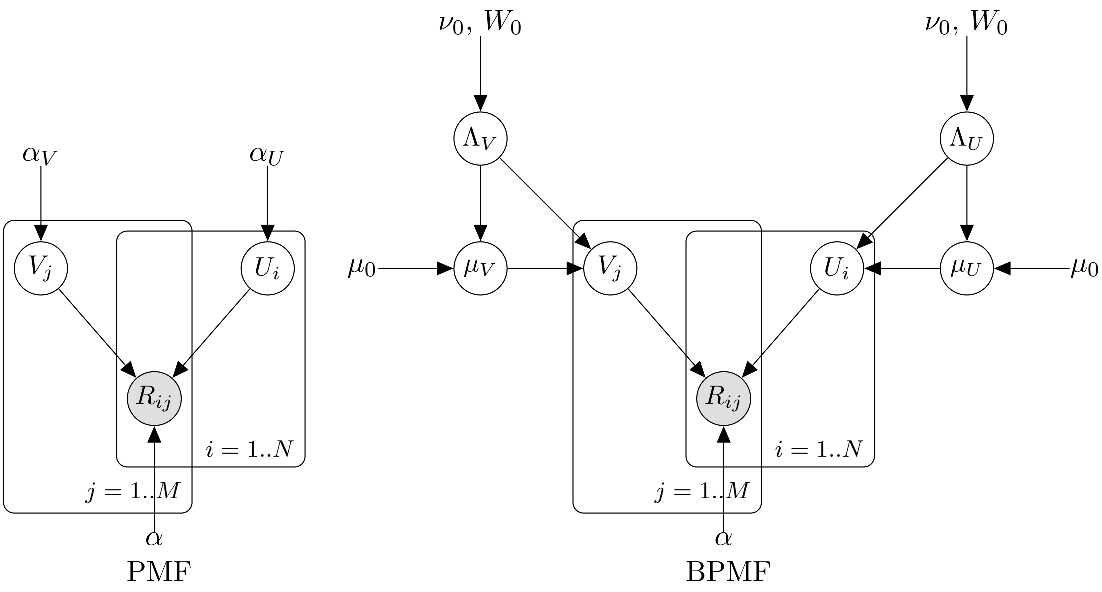


Figure 2.2: Graphical Model for PMF and BPMF

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 $t = 1 \ldots T$

    - Sample hyperparameters Θ_U and Θ_V
    - For $i = 1 \ldots N$ 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 [21]. 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.

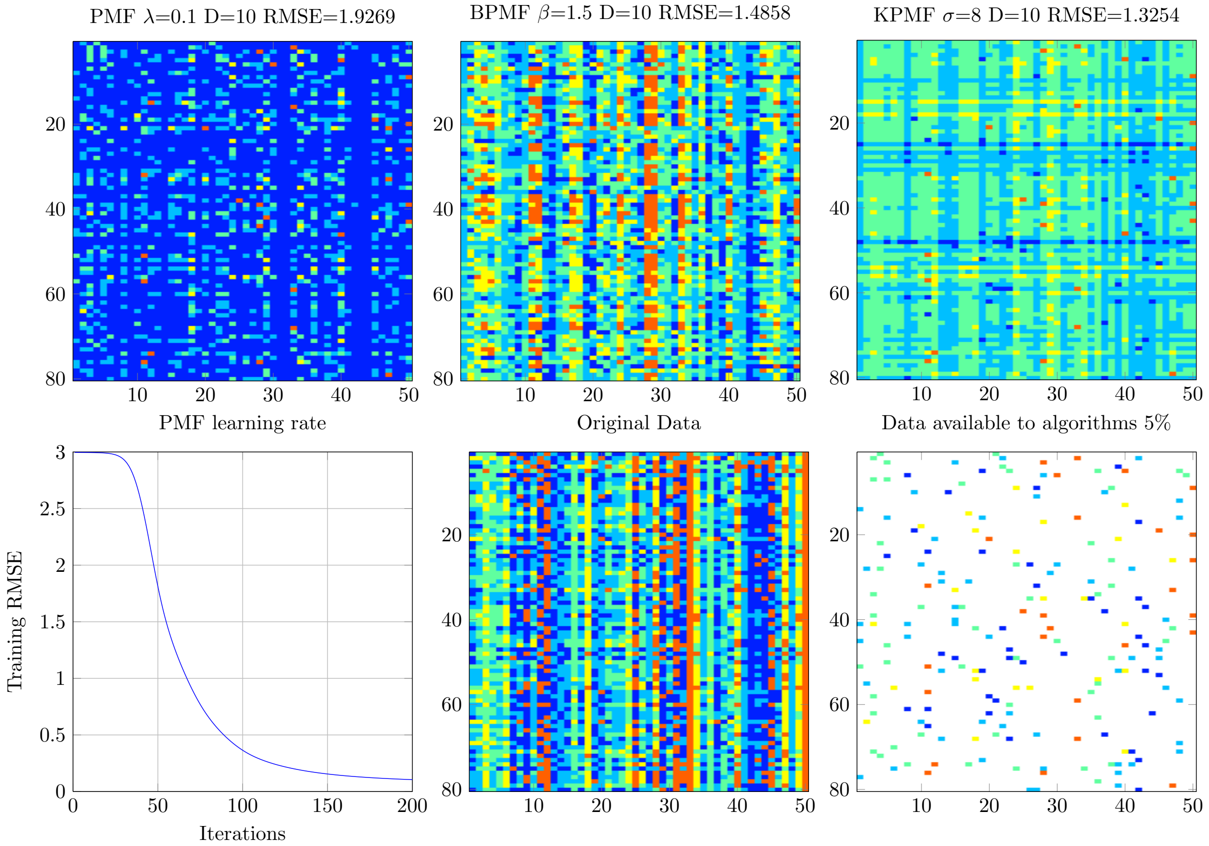
Performance of KPMF is best on very sparse data due to its ability to *smooth* out data from its assumption of inter-row and inter-column correlation. In fact KPMF can work very well on empty rows or columns by filling them in with the most likely values - something PMF and BPMF struggle with. Once more 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

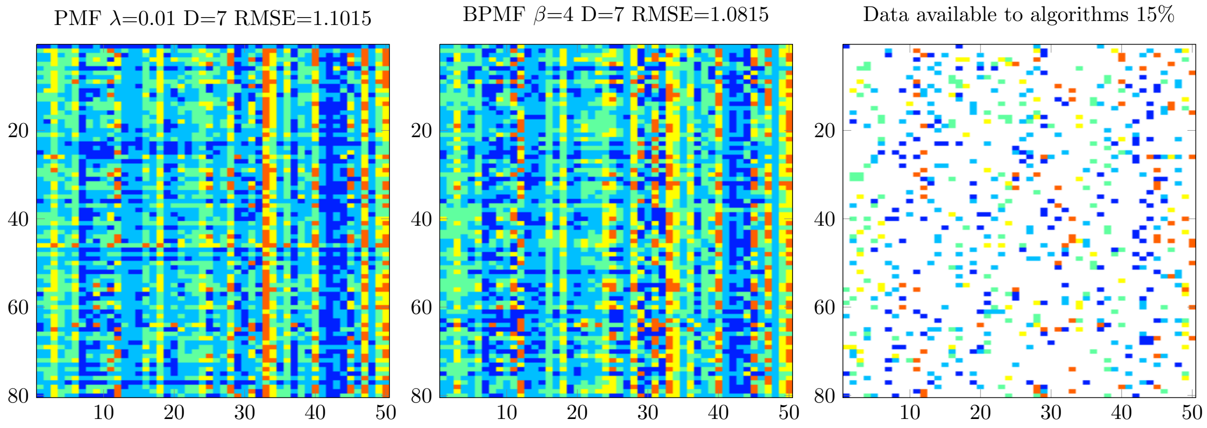
To understand what may be best to use we test each type of algorithm on a dataset. This allows to see its advantages and weaknesses. For the first test we look at figure 2.3.

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



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 2.3: Rating Predictions on synthetic data 5% complete



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

Figure 2.4: Rating Predictions on synthetic data 15% complete

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 2.4. PMF experienced the largest RMSE decrease - something to keep in mind for sample selection. KPMF's RMSE has also improved however its output did not look any better.

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 2.5 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 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 [21]. However, despite providing a diffusion kernel

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

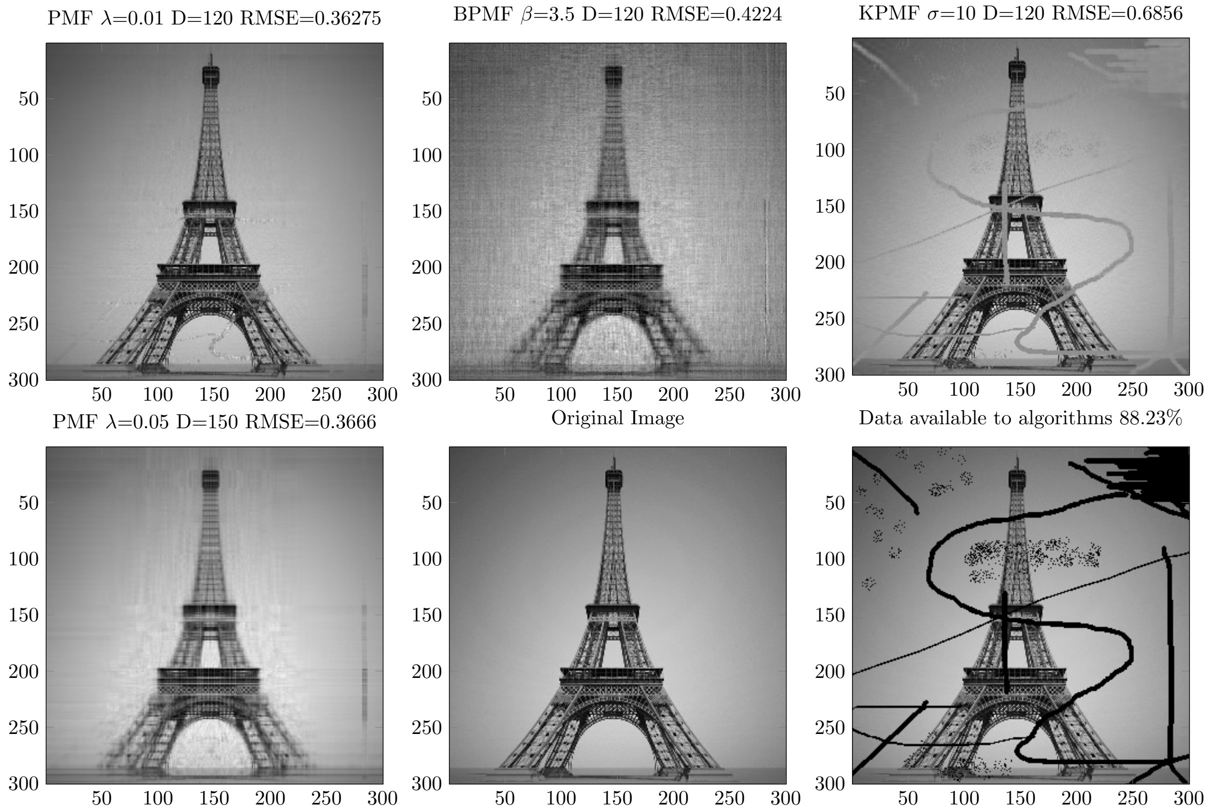

PMF λ=0.01 D=120 RMSE=0.36275
BPMF β=3.5 D=120 RMSE=0.4224
KPMF σ=10 D=120 RMSE=0.6856
PMF λ=0.05 D=150 RMSE=0.3666
Original Image
Data available to algorithms 88.23%
50
100
150
200
250
300

Image corrupted by drawing over and virtual spray paint

Figure 2.5: Image Restoration

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

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 perfectly (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.

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.

[1] This is actually a combination of λ_U and λ_V from equation 2.6

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

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 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 [8] by first training U and V for initial data and then updating these parameters at each incoming sample.

Table 2.1 shows a comparison of the pros and cons of online and offline learning for matrix factorisation systems. The general problem with online PMF is that the error function for PMF 2.6 is biconvex in U and V and that the minimum is found by alternatively updating each variable, which does not ensure global minimum. Thus if the parameters are stuck in a local

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

	Online	**Offline**
Pro	Fast update, less iterations Reuses parameters	Adapts better to new items Less local minima issues
Con	Can get stuck in local minima	Time intensive Reinitialises parameters

Table 2.1: Comparison of online and offline matrix completion algorithms

minimum online updating will not help and randomly reinitialising them will be better. This is why we will default to using offline updating for the purpose of this report.

2.3.5 Test and Validation dataset

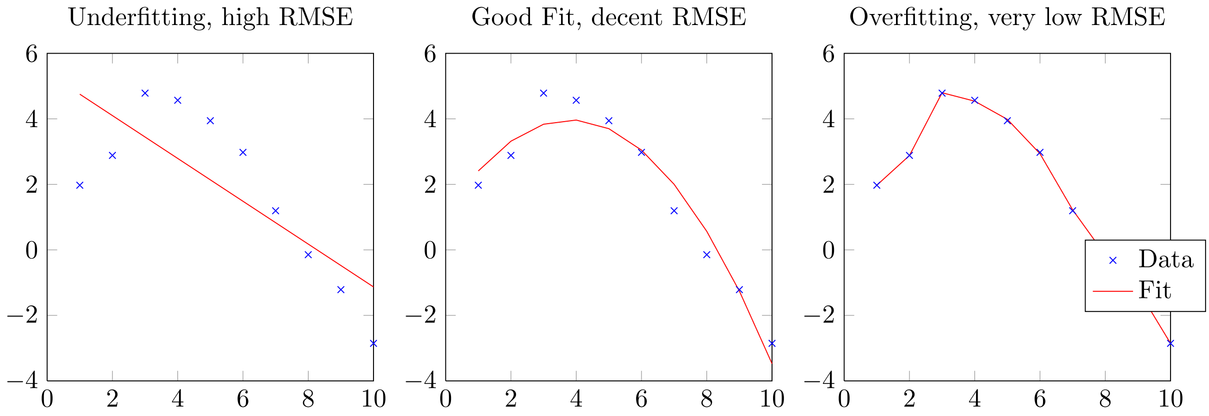


Figure 2.6: Examples of different model fitting

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 2.6, 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 ([17] 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.

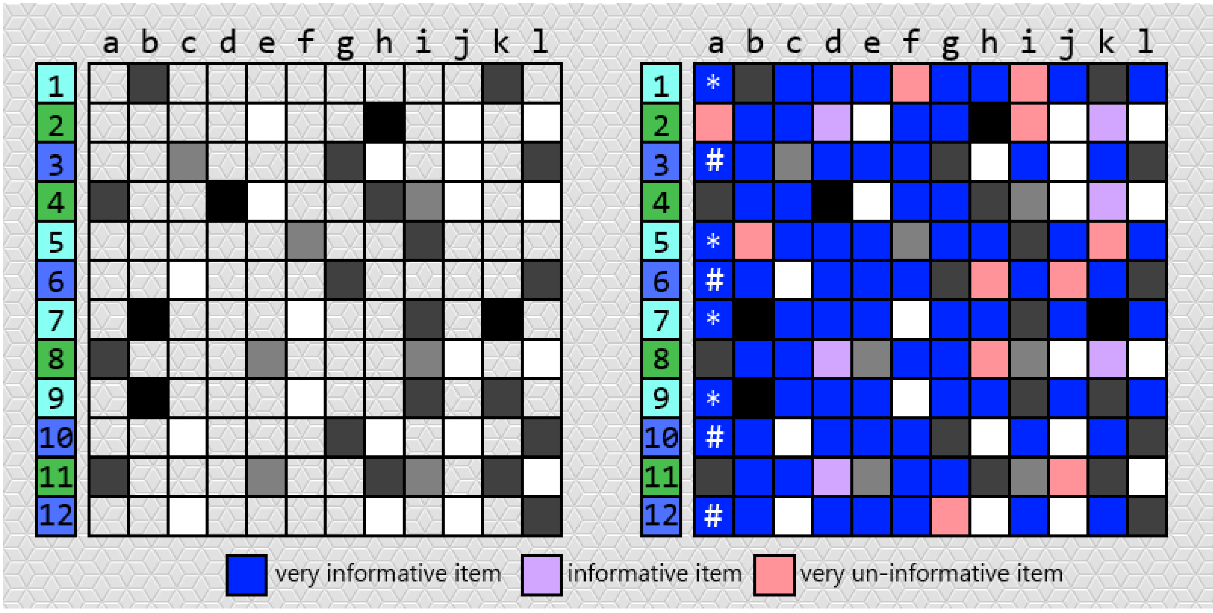
To avoid these issues most simulations will be carried on synthetic data, which is less affected by noise.

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.



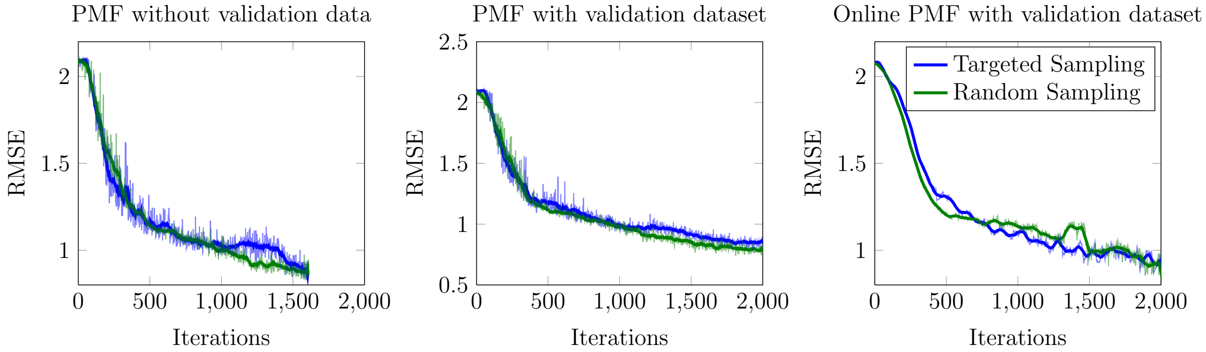
Users are represented in rows, and there has been 3 type of users dened, dark blue, light blue and green. Items are represented in the columns by letters. User item pairs that are rated are represented by a shade of grey and unrated squares are left blank.

Figure 3.1: Diagram of a simplied 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 2.3.

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.

It is important to be consistent with what model is used to predict data as some samples may be more useful for one model than an another. For example one model, like BPMF, will perform well by knowing a bit of everything to infer a posterior distribution of data, however PMF which does not do this will not find this sample as useful.

Effectiveness could also be linked to individual RMSE decrease, that is when a sample is discovered the amount by which it has decreased RMSE is used to rate how useful it has been. This can be a somewhat subjective error as the more about a model is know the harder RMSE decrease is.

3.2.1 Discrete vs Continuous Data

RMSE will be different for discrete and continuous data due to the way it is post-processed. Continuous data, obtained from $\hat{R} = U^T V$, is left untouched and used to calculate RMSE. For discrete data the values of $\hat{R}$ are first bounded between the minimum and maximum, for example 1 and 5 for film ratings. After this $\hat{R}$ is rounded to the nearest number in the discrete set. Error can then be calculate as traditional RMSE or as number of correct(sometimes referred to as positive) samples [10].

3.3 Formal Definition

The mask matrix Z is used to determine whether a value is known (or more accurately, part of the training set, the training set mask is called Z_{tr}). Its indices are the same as R except that values can only be 0 or 1. The known values are represented by 1 and form the set $\mathcal{O}$. In other words $R_{\mathcal{O}}$ is the matrix of the known values. We will also have the pool of queriable samples $\mathcal{P}$, that is the samples not yet known but that may be requested. Note that it is not always the case that $R_{\mathcal{O}} \cap R_{\mathcal{P}} = R$ as there may be unqueriable unknown samples. The samples queried part of the active sampling process will form the set $\mathcal{A}$, with $R_{\mathcal{A}}$ being the requested values. In graphical form the matrix representing $\mathcal{A}$ will usually be coloured. White represents the unrequested samples and the coloured cells represent the order of sampling.

3.4 Goals of Active Sampling

To select certain samples some aspects which are thought to be able to reduce RMSE better than random selection are outlined.

Model Look at samples that may produce the greatest change in the parameters U and V, with he hope that this will mean a large change towards the true distribution.

Sample Uncertainty Try to seek the samples that are most likely to vary based on current distribution of parameters, that is their uncertainty.

Knowledge How much is known about a current row or column and aiming to maximise this overall, with the aim of getting a global insight of the data.

Max-Minimum To best determine the boundaries of the data the largest and smallest estimated values are queried with the aim of minimising over and under estimation error.

3.5 Minimum Knowledge Search

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

3.5.1 Algorithm

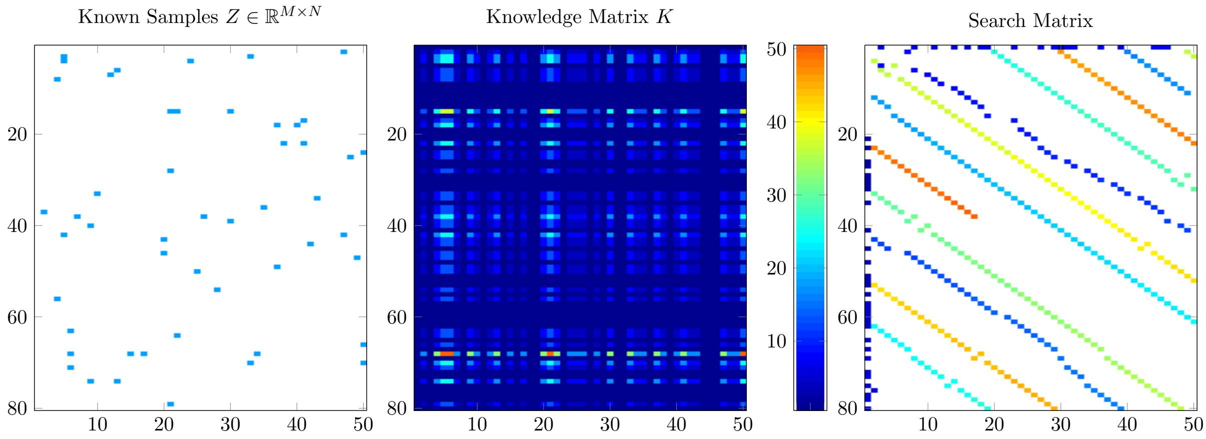
For any matrix factorisation problem we have the mask matrix $Z \in \mathbb{R}^{M \times N}$, with elements being 1 for every known value and 0 for every unknown value (values in the validation set would be also 0).

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, $\mathbf{b} \in \mathbb{R}^M$
    $K \leftarrow \mathbf{a} \cdot \mathbf{b}$                            ▷ $K \in \mathbb{R}^{M \times N}$
    $x, y \leftarrow index\_of\_min(K)$         ▷ Often multiple candidates, select first one
    while $Z(x, y) == 1$ 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.

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



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

randomly select an element to potentially try and get a temporary advantage.

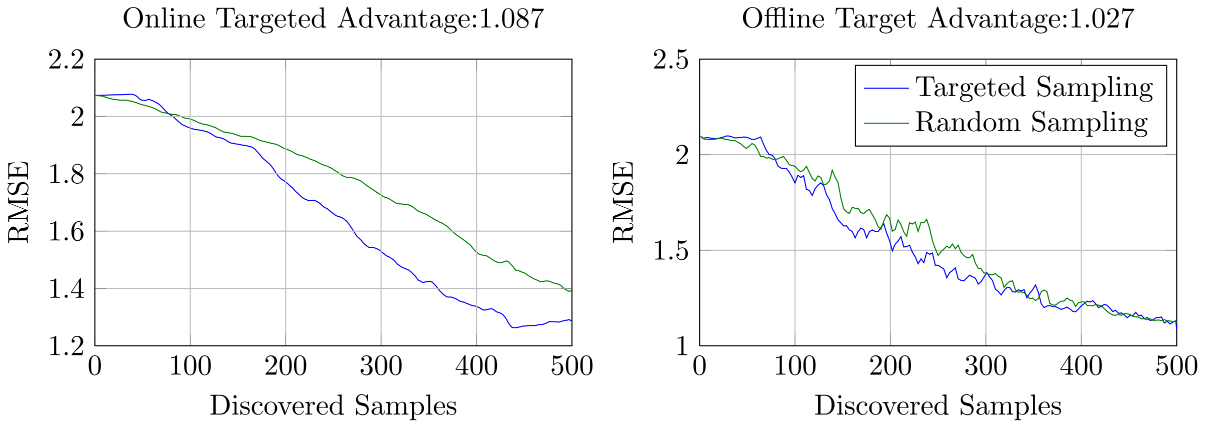
Note: Other variants of this algorithm that were tried included selecting the minimum suitable index of **a** and **b** - i.e. the coordinates that intersect with the least known row and column. However this technique did not perform as well due to not seeking to maximise one row and column first at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

3.5.2 Notes on performance

To better compare sampling algorithms we benchmark them against random sampling by defining the advantage value as

$$\frac{\text{RMSE Area under random selection}}{\text{RMSE Area under targeted selection}}$$

Figure 3.4 shows the change in RMSE over time as a function of discovered samples. From this it can be seen that Minimum Knowledge Search works well on the synthetic dataset compared to random sampling, nearly always outperforming it. This is in part due to the nature of the data, which has most rows and columns containing a similar level of information that their neighbours, thus a search to find out more about the average row and column is preferred and useful. However, should most of the information only be contained in one area of the matrix then the performance would not be as satisfactory. For datasets of very low variance or uniform regions, this search could perform even better. This search method was tested on non-synthetic



Over 10 online trials, minimum knowledge search advantage was 1.043. Final random RMSE:1.390, targeted RMSE:1.289.
Over 10 offline trials, minimum knowledge search advantage was 1.017. Final random RMSE:1.120, targeted RMSE:1.087.
Carried out on the sample synthetic dataset with 1.25% of samples initially discovered. $\lambda = 0.01$, $D = 7$

Figure 3.4: RMSE vs Samples Discovered for Online PMF

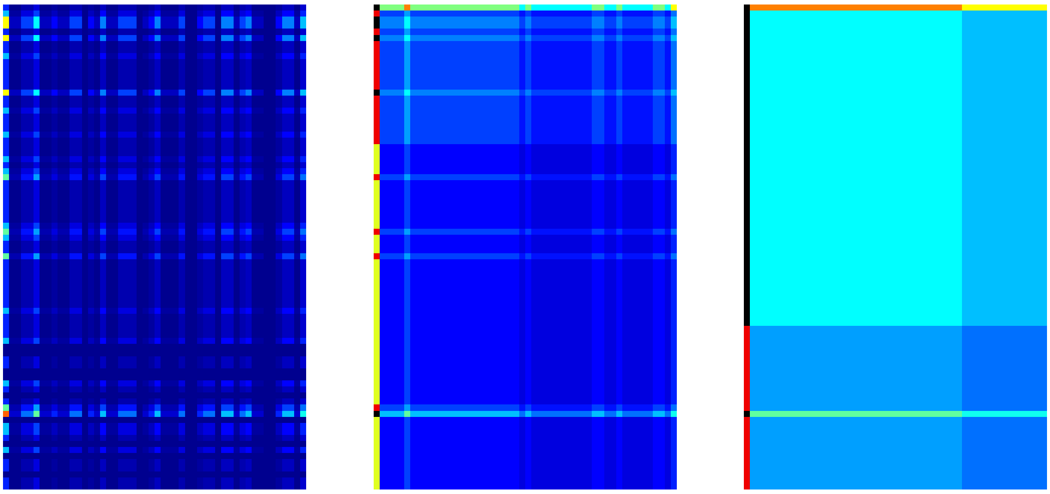
data, trying to recover an Eiffel Tower image, as seen in appendix figure B.3. As this is a somewhat uniform picture (a sky and just a structure) minimum knowledge search performed very well - 12% advantage in this case.

The main weakness in this algorithm is described in figure 3.5. As it targets the row-column combinations with the least knowledge the number of samples available for selection at each step can actually increase due to the matrix becoming more uniform. Thus the potential for sample discrimination decreases over time and the advantage only exists during initial matrix sampling. Initialising a search trial with many more known samples can cause minimum knowledge search to perform 20% worse [1].

3.5.3 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

[1] In terms of advantage value



(a) 10 new samples (b) 50 new samples (c) 200 new samples

Knowledge scaled is same as figure 3.3 - the darker the blue the less is known.

Figure 3.5: Evolution of Knowledge Matrix over time

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.

Additionally the aim of homogenising the amount known in each column and row is not always possible in constrained situations where the sampling space is limited, that is a sample determined for selection is not available(i.e. it may not be possible to ask a user his 5 star rating of a film if he hasn't seen it).

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

Chapter 4

Advanced Sample Selection

Now that we have a good idea of what active sampling involves attention will be focused to more advanced algorithms with the aim of improving performance.

4.1 Clustered Knowledge Sampling

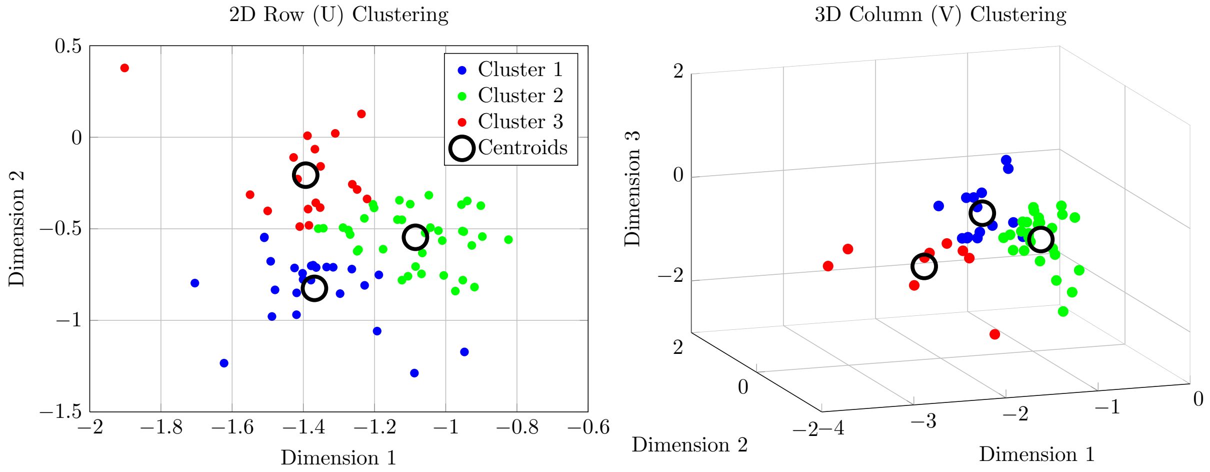
In the proposed minimum knowledge search algorithm (section 3.5), its poor performance was mainly due to ignoring the data and only focusing on the mask matrix, which homogenised the amount of known samples. This had two problems, one in situations with search space constraints and the other for actually ignoring data and not taking "information" into account.

4.1.1 Outline

Minimum knowledge search failed to integrate information about the known data to target more informative samples. Here we decide to take advantage of the U and V matrices formed during matrix factorisation , as in equation 2.1. Each of the columns of U and V are supposed to represent features of the related row or column. We now assume that if we knew a minimal amount of each feature it will be possible to cluster rows and columns into groups. Figure 4.1 shows the features of U and V clustered into 3 different groups[1]. In the case of drug-target interaction database dimension 1 could represent the estimated presence of a chemical, say H_2O and the second dimension of the presence that represents the presence of another chemical.

In collaborative filtering we seek to infer these features to best reconstruct the full matrix. The idea behind clustered sample selection is to target the groups with the least certainty, to improve their accuracy and the model as a whole. In a movie-user scenario this means that we can separate the thriller film liking users from the drama movie liking users, then if we find out

[1] 3 was arbitrarily chosen as the number of clusters



U and V from synthetic dataset with 15% of samples initially discovered. $\lambda = 0.01$, $D = 7$
Dimensions reduced for plotting purposes

Figure 4.1: Sample k-means clustering of U and V by features

that less is know on thriller liking people then the model should be better off learning an extra datapoint on that group. This would allow for row selection and the selection of columns can be done in a similar way. We will define knowledge in the same way as minimum knowledge search, as the fraction of known rows and columns.

4.1.2 Algorithm

We base the work of the clustered knowledge search on the minimum knowledge search algorithm, in that we will target not the specfic row-column combination we know less about but the row-colum cluster we know the less about.

The first step involves determining the ideal number of clusters for a dataset. In itself this is not an easy task and can increase algorithm complexity due to having to test for all different candidate number of clusters. To determine the number of clusters we use the silhouette measure [9] which essentially measures how tight data is when in a group. It is not always effective but does the job.

Once we have determined the number of clusters, we cluster the rows and columns according to the features of U and V using a clustering algorithm. K-means was chosen for being fast yet good at clustering. Alternatives such as GMM were considered but found to not be appropriate due to their great increase in complexity and little increase in clustering quality.

From the clustered rows we create the vectors $\mathbf{a}$ and $\mathbf{b}$ which contain the amount know of each cluster. This allows the create of the knowledge matrix, $K = \mathbf{a}^T \cdot \mathbf{b}$. From this a sample

Algorithm 2 Clustered Knowledge Search algorithm

```
procedure ClusKnowSearch($U$,$V$,$Z$)
    $k_U \leftarrow \text{NumOfClust}(U)$    ▷ Returns best guess of number of clusters
    $k_V \leftarrow \text{NumOfClust}(V)$
    $u_{\text{clusters}} = \text{kmeans}(U, k_U)$    ▷ Assigns cluster IDs
    $v_{\text{clusters}} = \text{kmeans}(V, k_V)$
    $\mathbf{u_{info}} \leftarrow \text{meanrow}(Z)$    ▷ $\in \mathbb{R}^N$
    $\mathbf{v_{info}} \leftarrow \text{meancol}(Z)$    ▷ $\in \mathbb{R}^M$
    $\mathbf{a} \leftarrow \mathbf{0}^N$    ▷ $\in \mathbb{R}^N$
    $\mathbf{b} \leftarrow \mathbf{0}^M$    ▷ $\in \mathbb{R}^M$
    for $i = 1 \ldots k_U$ do
        $\mathbf{u_{knowl}}[i] \leftarrow \sum_{\text{index}=i} \mathbf{u_{info}}$    ▷ Adds up all info values
        $\mathbf{a}[index\ u_{\text{clusters}} = i] = \mathbf{u_{knowl}}[i]$    ▷ Assigns sum of info to clusters
    end for
    for $j = 1 \ldots k_V$ do
        $\mathbf{v_{knowl}}[j] \leftarrow \sum_{\text{index}=j} \mathbf{v_{info}}$
        $\mathbf{b}[index\ v_{\text{clusters}} = j] = \mathbf{v_{knowl}}[j]$
    end for
    $K \leftarrow \mathbf{a}^T \cdot \mathbf{b}$    ▷ $K \in \mathbb{R}^{M \times N}$
    $x, y \leftarrow index\_of\_min(K)$    ▷ Many candidates, one is chosen at random
    while $x, y$ not valid sample request do
        $x, y \leftarrow next\_min\_index(K)$
    end while
    return $x, y$    ▷ Return $x, y$ that has least knowledge
end procedure
```

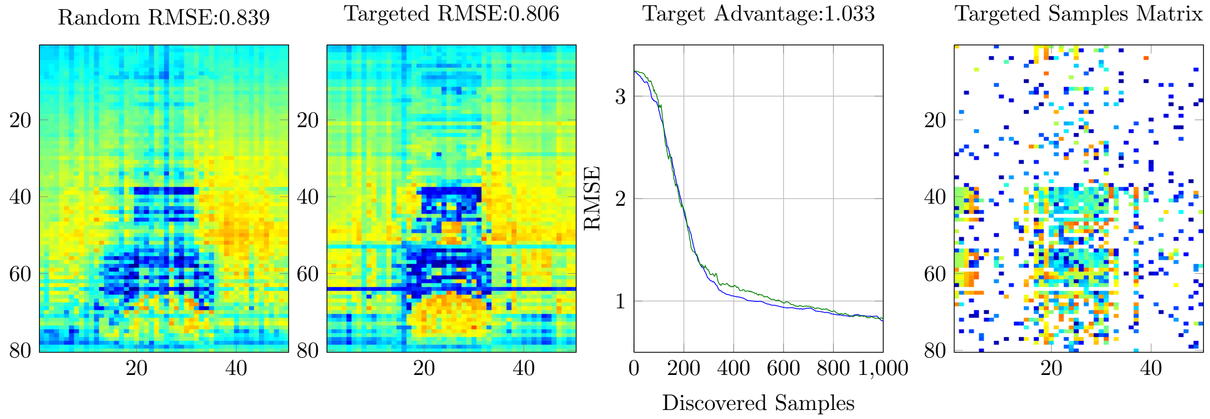
is selected as in section 3.5.

This algorithm is defined formally in algorithm 2.

4.1.3 Performance

The first initial observations on the performance is that during a cold-start, that is when very little is know about the matrix, clustering will not perform well and sample selection will effectively be random. As soon as enough data is gathered a large advantage compared to random sampling is usually observed. This is because, unlike the minimum knowledge search, the search area is always restricted to a reasonably small subset.

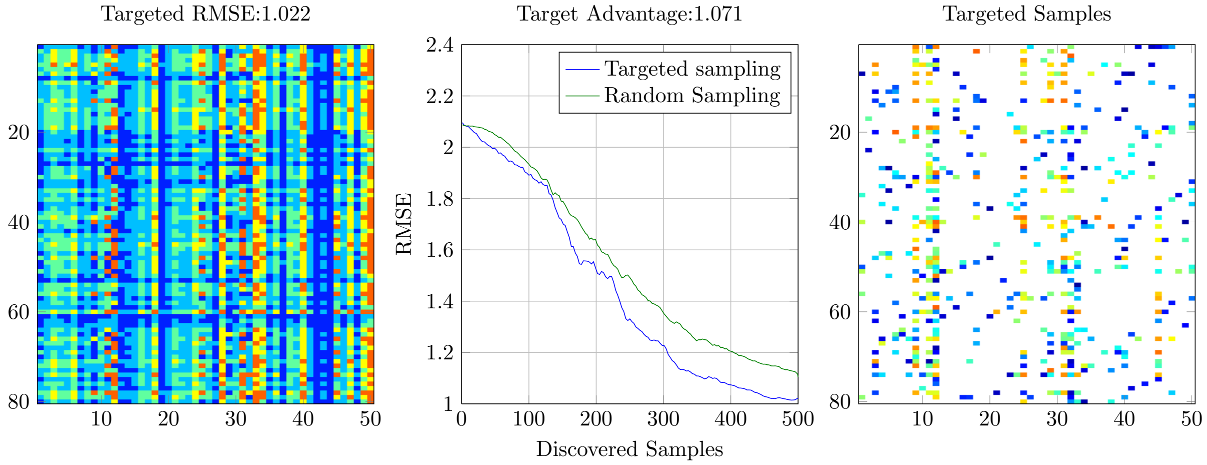
It will also tend to cluster currently unknown groups of data together and targeting them even later on in execution which is useful. However once the number of unknown rows and columns have fallen down it suffers from not requesting individual rows and columns with no knowledge if they are assigned to a cluster of high information - this seemed to be the main drawback from this algorithm. This can be seen from the lines in figure 4.2 which are never requested. This is why it is preferred to randomly sample from the restricted subset (line 19 in algorithm 2) as it can mitigate this problem. In cases where the first available value of the subset is selected, the algorithm was found to sometimes perform worse than random in very late stages, due to removing the ability to sample from certain, useful, areas.



5% of samples initially discovered. PMF $\lambda = 0.01$, $D = 15$
The targeted samples matrix uses the same scale as in figure 3.3. The darker blue the colour is the earlier on it was targeted.
Blue line is targeted sampling and green on is random sampling.
The random subgroup sampling method was employed for algorithm 2

Figure 4.2: Clustering Knowledge Search on Eiffel Tower Image

In figure 4.2 we see from the targeted samples matrix that the search first started a bit randomly (the scatter blue points) and that as data was gathered the areas with higher data variance were targeted, that is the groups belonging to level 1 and 2 of the Eiffel tower. A realtime simulation of allows us to see that the sky area of the picture is discovered early on and then ignored in favour of the more complex areas of the structure.



1.25% of samples initially discovered. Offline PMF $\lambda = 0.01$, $D = 7$
Random sampling RMSE curve is average of 10 trials. On average CKS performs 5% better over 500 samples.
The random subgroup sampling method was employed for algorithm 2

Figure 4.3: Clustering Knowledge Search on Synthetic Data

The eiffel tower image, while real word data, has very uneven groups (about 70% is sky) and has very high dimensions for reconstruction. Thus it is more informative to see how CKS performs on low rank data that resembles that of a movie recommendation system. This is done in figure 4.3 on synthetic data. This time the increase in performance is obvious - the minimum knowledge search had a random advantage of 1.017 compared to 1.071 for MKS over 500 new requested samples. Additionally CKS led to CKS RMSE being consistently under the random RMSE curve over trials carried out on synthetic data. From the targeted samples matrix, constructed in the same way as figure 4.2, we can see the search strategy CKS employs, first randomly targeted samples where little is know and then moving on to specific films and users. The advantages of using the currently known data as well as the mask matrix is clear, in contrast to MKS ignoring the underlying data.

The complexity of CKS is bounded by that of the clustering stage, which for K-Means is $O(n^{Dk+1} \ln n)$. k is the number of clusters and n of data-points.

4.1.4 Improvements and Limitations

Some improvements involve more effectively detecting outliers. For example in figure 4.1 there is a clear outlier in the red row cluster. Detecting these and considering them separately would help avoid some of the problems detected in figure 4.2 and demonstrated on synthetic data in appendix figure B.4. This can be as simple as changing clustering algorithm - though this will increase complexity. Related to this would be the ability to deal with uneven clusters, for example in the case of an image there can be very even areas, such as the sky, yet very little is needed to be known due to the uniform colour.

Also for high dimension data, clustering is not always reliable [5]. Thus if a high feature dimension for U and V is chosen some pre-processing such as PCA to reduce dimension should be considered.

It would also be useful to use side information when clustering rows and columns. For example in a user-item scenario the user age and gender could be useful in better clustering.

Finally the issue of differently sized clusters can be an issue while targeting, as is expanded upon later on in section 4.6.2.

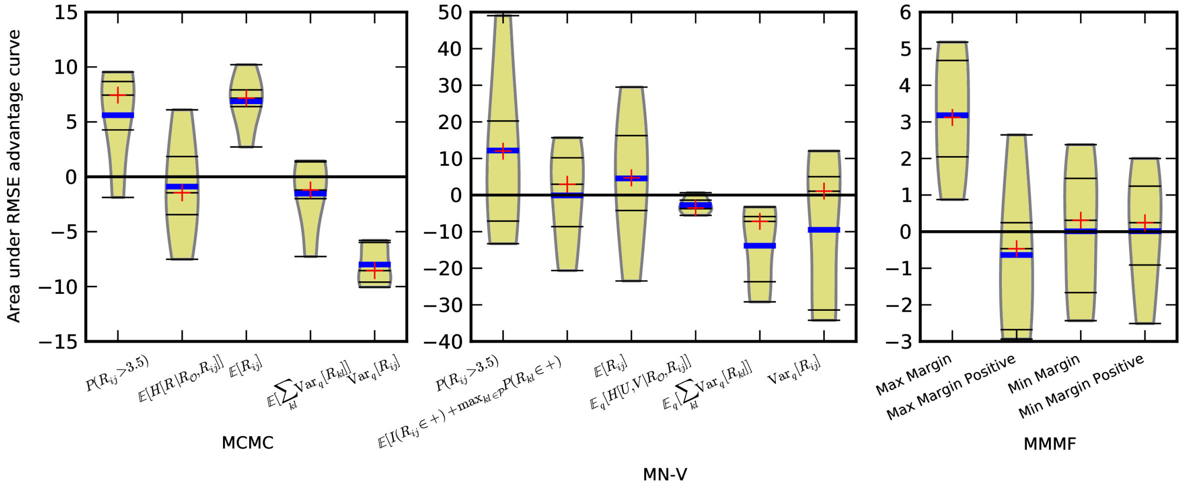
4.2 Cell Specific Search Methods

In the methods outlined above samples were targeted by restricting the search space in some way to allow choosing one particular sample that satisfied the requirements. Sutherland et al. [15] outline active sampling methods that give a criteria for selecting specific cells. An overview of their performance is given in figure 4.4.

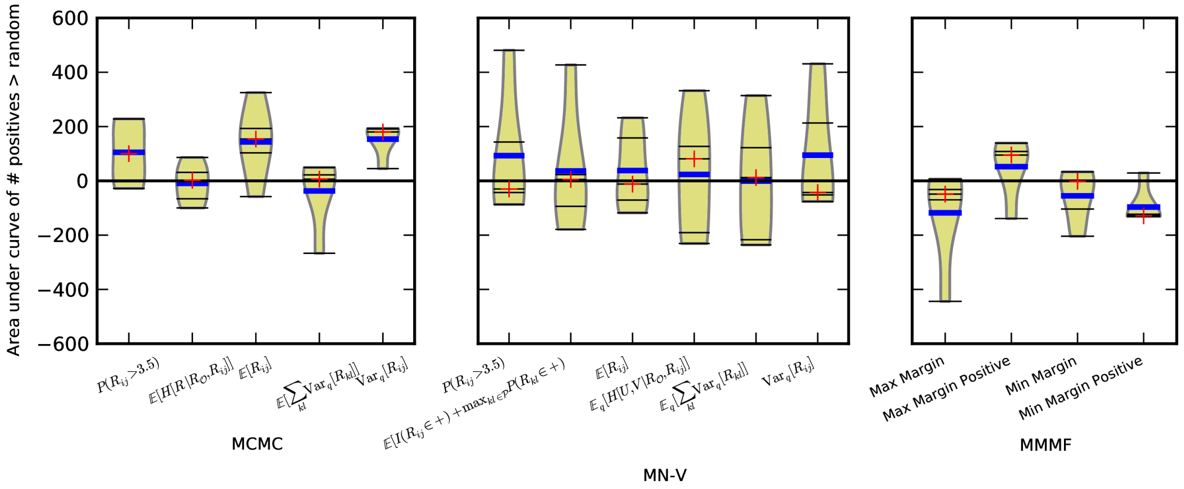
As can be seen, not all proposed methods performed well and even underperformed random selection. For this reason all Maximum Margin Matrix Factorization based methods are ignored, having little significant improvement over random selection. The two methods that exhibited good performance over the discrete and continuous datasets where the $\mathrm{Var}[R_{ij} \mid R_{\mathcal{O}}]$ searches, under Matrix-Normal and MCMC approximations. Thus, these will be the two algorithms implemented to be tested.

4.3 Matrix-Normal Variance Search

Here the idea is to infer the variance of unknown samples to be able to choose on to target. The variance is used as a proxy for uncertainty, where samples with a high variance are assumed to be the most “uncertain”, due to being able to vary more.



This figure from Sutherland et al.'s paper [15] gives the prediction results of various techniques observed. The Area under RMSE advantage curve values represent the results of five runs of a 10 × 10 rank 1 continuous synthetic experiment against random sampling. Thus a negative value represents a method outperforming random selection.



This figure gives the prediction results of various techniques observed for a synthetic discrete dataset. This time the measure is the number of useful values queried in a 10 × 10 rank 4 discrete synthetic experiment against random sampling. A positive value represents a method outperforming random selection.

Figure 4.4: Outline of the performance of algorithms proposed by Sutherland et al.

4.3.1 Derivation

The value we seek for each sample is the variance of each individual sample given the observed set, that is $\mathrm{Var}[R_{ij} \mid R_{\mathcal{O}}]$. We derive it as such:

$$\begin{aligned}
\mathrm{Var}[R_{ij} \mid R_{\mathcal{O}}] &= \mathbb{E}[\mathrm{Var}[R_{ij} \mid U, V] \mid R_{\mathcal{O}}] + \mathrm{Var}[\mathbb{E}[R_{ij} \mid U, V] \mid R_{\mathcal{O}}] \\
&= \mathbb{E}[\sigma^2] + \mathrm{Var}[\mathbf{u}_i^T \mathbf{v}_j \mid R_{\mathcal{O}}] \\
\mathrm{Var}[\mathbf{u}_i^T \mathbf{v}_j \mid R_{\mathcal{O}}] &= \mathrm{Var}\left[\sum_{k=1}^{D} U_{ik} V_{jk} \mid R_{\mathcal{O}}\right] \\
&= \sum_{k=1}^{D} \sum_{l=1}^{D} \mathrm{Cov}[U_{ik} V_{jk}, U_{il} V_{jl} \mid R_{\mathcal{O}}] \\
&= \sum_{k=1}^{D} \sum_{l=1}^{D} \mathbb{E}[U_{ik} V_{jk} U_{il} V_{jl} \mid R_{\mathcal{O}}] - \mathbb{E}[U_{ik} V_{jk} \mid R_{\mathcal{O}}] \mathbb{E}[U_{il} V_{jl} \mid R_{\mathcal{O}}]
\end{aligned}$$

As $\mathbb{E}[\sigma^2]$ is a constant, we can ignore it. Now $\mathbb{E}[U_{ik} V_{jk} \mid R_{\mathcal{O}}]$ and $\mathbb{E}[U_{il} V_{jl} \mid R_{\mathcal{O}}]$ are not yet usable in their current form, we find their value via:

$$\mathbb{E}[X_a X_b] = \mathbb{E}[X_a]\mathbb{E}[X_b] + \mathrm{Cov}[X_a, X_b] = \mu_a \mu_b + \Sigma_{a,b}$$

Where $\Sigma_{a,b}$ is the covariance matrix - $\Sigma_{ab} = \mathrm{Cov}(X_a, X_b) = \mathbb{E}\left[(X_a - \mu_a)(X_b - \mu_b)\right]$

The expression for $\mathbb{E}[U_{ik} V_{jk} U_{il} V_{jl} \mid R_{\mathcal{O}}]$ is slightly more complicated and requires the use of Isserlis' theorem [4]. Note that it assumes the random variables used are normal.

$$\begin{aligned}
\mathbb{E}[X_a X_b X_c X_d] = {} & \mu_a \mu_b \mu_c \mu_d + \mu_c \mu_d \Sigma a, b + \mu_b \mu_d \Sigma_{a,c} + \mu_b \mu_c \Sigma_{a,d} + \mu_a \mu_d \Sigma_{b,c} + \mu_a \mu_c \Sigma_{b,d} \\
& + \mu_a \mu_b \Sigma_{c,d} + \Sigma_{a,b} \Sigma_{c,d} + \Sigma_{a,c} \Sigma_{b,d} + \Sigma_{a,d} \Sigma_{b,c}
\end{aligned}$$

To move from the a, b, c, d coordinate system to i, j, k, l to calculate $\mathrm{Var}[\mathbf{u}_i^T \mathbf{v}_j \mid R_{\mathcal{O}}]$ we create what is called the Matrix-Normal Framework approximation, which is a multivariate matrix assuming Gaussian distribution [3]. For this we build a matrix system as such [16]:

$$\Sigma_{ij} = \mathrm{Cov}(\left[\begin{smallmatrix} U^T \\ V^T \end{smallmatrix}\right], \left[\begin{smallmatrix} U^T \\ V^T \end{smallmatrix}\right]) = \begin{array}{c|c|c|c|c|c|c|c|c|} & \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 & \mathbf{u}_4 & \mathbf{u}_5 & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \\ \hline \mathbf{u}_1 & & & & & & & & \\ \hline \mathbf{u}_2 & & & & & & & & \\ \hline \mathbf{u}_3 & & & & & & & & \\ \hline \mathbf{u}_4 & & & & & & & & \\ \hline \mathbf{u}_5 & & & & & & & & \\ \hline \mathbf{v}_1 & & & & & & & & \\ \hline \mathbf{v}_2 & & & & & & & & \\ \hline \mathbf{v}_3 & & & & & & & & \\ \hline \end{array} \tag{4.1}$$

$$\Omega_{kl} = \mathrm{Cov}([\,U\ V\,], [\,U\ V\,]) = \begin{array}{c|c|c|c|c|} & \mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 \\ \hline \mathbf{f}_1 & & & & \\ \hline \mathbf{f}_2 & & & & \\ \hline \mathbf{f}_3 & & & & \\ \hline \mathbf{f}_4 & & & & \\ \hline \end{array} \tag{4.2}$$

That is we assemble U and V into a single matrix and create the covariance matrices Σ and Ω. Σ is the covariance of the features of each row (U) and column (V), thus $\Sigma \in \mathbb{R}^{(M+N)\times(M+N)}$. Ω is the covariance of the features themselves, thus $\Omega \in \mathbb{R}^{D\times D}$. This is used to get the full covariance matrix over all features of rows and columns by $\Sigma \otimes \Omega$.

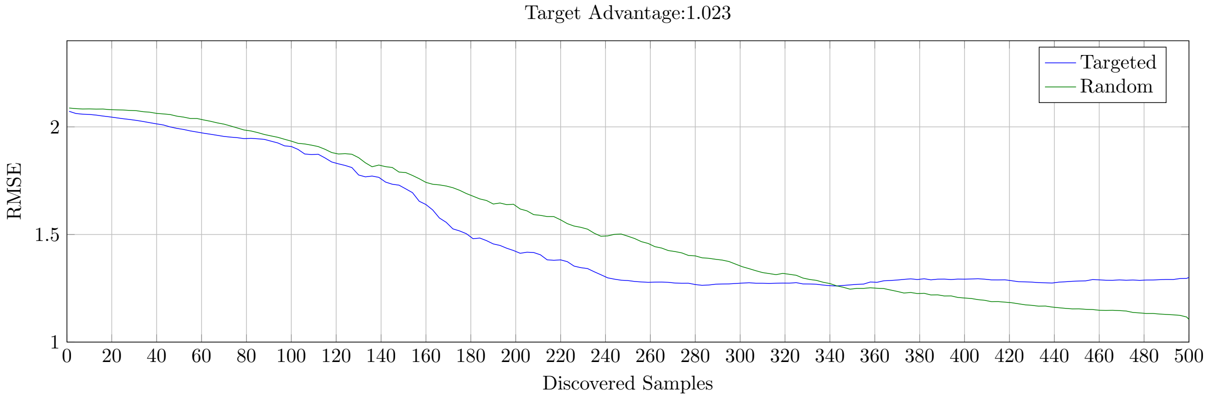
From this we can consider U and V to be one variable, thus $\mathbb{E}[U_{ik}V_{jl}] = \mathbb{E}[X_a X_b]$ where $a = (i, j)$ and $b = (k, l)$. This gives:

$$\begin{aligned} \mathbb{E}[X_a X_b] &= \mu_a \mu_b + \Sigma_{a,b} \\ &= \mathbb{E}[U_{ik}V_{jl}] = U_{ik}V_{jl} + \Sigma_{ij}\Omega_{kl} \end{aligned}$$

The indices i, j refer to the U, V coordinates in Σ in equation 4.1. k, l refers to the coordinates of the features of Ω in equation 4.2. Note that the complexity to calculate Σ and Ω combined is $O(D^3 + (N + M)^3)$. Additionally note that $U_{ik}V_{jl}$ are used as means taken from the PMF best fit - while not ideal is an assumption required for calculation [15].

4.3.2 Performance

This was first tested on the same 80 × 50 synthetic data as in previous experiments and while the performance was good, it tended to "edge out" after more than 25% of the dataset became available. A typical run is show in figure 4.5. This is consistent with the performance reported by Sutherland et al. in figure 4.4.



Performed on Synthetic Data.

Figure 4.5: Matrix Normal Maximum Variance Search

4.3.2.1 Improving Performance

Calculating the variance has a complexity of about $O(MND^2)$ in addition to the calculation of the covariance matrices, thus calculating the criteria at each new sample is not ideal. These tweaks were done to improve performance:

Variance Matrix Calculation Update Rather than calculating the matrix everytime a new sample comes in, we calculate the matrix at fixed intervals of incoming samples, for example after 10 have been requested.

Memory Optimisation There are many combinations of $\mathbb{E}[U_{ik}V_{jl}]$ over the double summation. To optimise, we use the symmetry property of the covariance matrix and locally cache already calculated values.

Unknown Samples We only calculate the variance for unknown samples.

Online Updating When U and V are calculated online it is also possible to gradually update the variance matrix.

Basic memory optimisation alone allowed the execution to be more than halved. In the case of a 80×50 matrix with $D = 7$, from 1.81 seconds down to 0.81 seconds on average.

4.4 MCMC Variance Search

From results obtained in simulations as well as Sutherland et al.'s we saw that the Matrix-Normal Variance Search performed reasonably well, but had instances of under performing random selection. One of the reasons for this was that the data distribution was not entirely reflected in U and V used to calculate $\mathrm{Var}[\mathbf{u}_i^T \mathbf{v}_j \mid R_{\mathcal{O}}]$.

4.4.1 Derivation

Here we keep the same equations for $\mathrm{Var}[\mathbf{u}_i^T \mathbf{v}_j \mid R_{\mathcal{O}}]$ but instead take U and V from an estimate sampled across the expected distribution of data. In BPMF, section 2.2.3, we had U and V expressed as:

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

Unfortunately an analytical solution is difficult to achieve and we instead rely on approximation methods. For quick intuition on how this works consider the following, simplified, function [1]:

$$I = \int g(\theta)p(\theta)d\theta$$

Where $g(\theta)$ is a function of θ and $p(\theta)$ is the distribution of this variable. In cases where this is not possible we resort to Monte-Carlo Markov Chain Integration:

$$\hat{I}_M = \frac{1}{M}\sum_{i=1}^{M} g(\theta^{(i)})$$

Where M is the number of values sampled and i the index. We have that as $M \to \infty$, $\hat{I}_M = I$. By process of iteration and random sampling the real distribution is approached and $\hat{I}_M$ is used instead of I.This is the process we will use to sample from the already known values.

Sutherland et al.[15] use Hamiltonian Monte Carlo sampling methods but here we will use Gibbs Sampling, in a similar way to Salakhutdinov and Mnih [12] in BPMF. Algorithm 3 describes the process to sample from V, which is exactly the same for U. A U and V obtained via PMF are used as inputs.

α_V is the precision hyperparameter for V and it is used to ensure non-singularity of C. μ_V is the average vector. β is a parameter used for the Inverse-Wishart distribution, used as the prior of covariance matrix from data assumed to be from a normal distribution. Details are

Algorithm 3 Gibbs Sampling for BPMF

```
procedure GibbsSamplingForV($U$,$V$,$R$)
    for j=1..M do
        $row_{in}$ =row indices of known samples in column $j$
        $\mathcal{M} = U_{index=row_{in}}$        ▷ Get feature vectors of known rows in column $j$
        $\mathbf{r} = R_{row_{in},j}$        ▷ Vector of known values in column $j$
        $C = (\alpha_V + \beta \cdot \mathcal{M}^T\mathcal{M})^{-1}$        ▷ Covariance of known feature vectors $U$
        $\mu_V = C \cdot (\beta \cdot \mathcal{M}^T\mathbf{r} + \alpha_V\mu_V)$        ▷ Update mean vector
        $\Lambda = \text{Cholesky}(C)$        ▷ Cholesky upper triangular decomposition
        $\mathbf{x} \sim \mathcal{N}(0, 1) \in \mathbf{R}^D$        ▷ Randomly sample $D$ variables from Normal
        $V_j = \Lambda\mathbf{x} + \mu_V$        ▷ Update $V$
    end for
    return $V$
end procedure
```

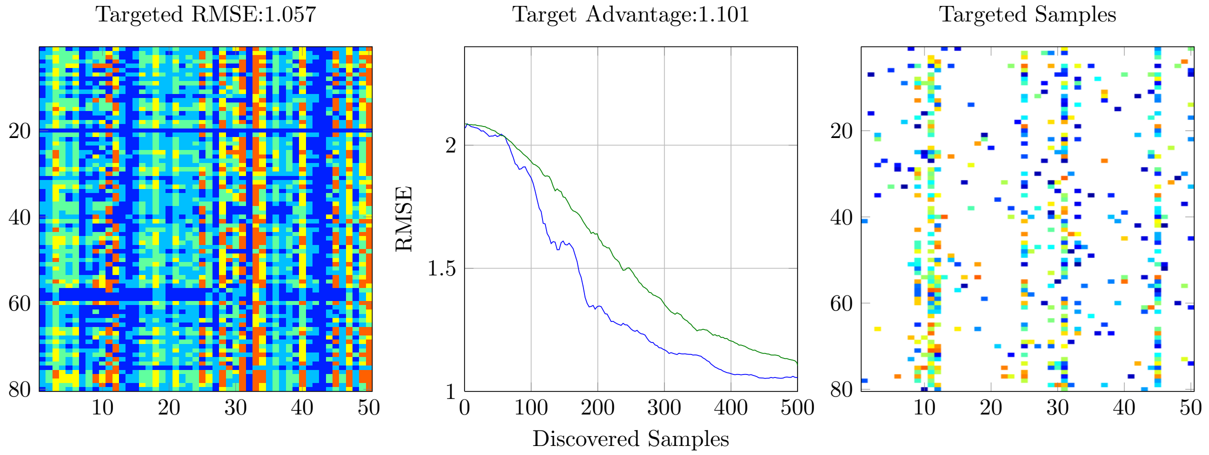
available in the appendix A.2. We take Λ as the Cholesky decomposition of the covariance C. If applied to a vector of uncorrelated samples ($\mathbf{x}$) it produces a vector with covariances of sampled system, which is why we add it to μ_V, creating a feature vector of the samples. Note that hyperparameters are reinitialised each loop before sampling trials.

In simpler terms, we capture the properties of the current distribution of known values in V_j in C and using the Cholesky decomposition update V with samples randomly generated from the captured distribution. Repeating this a few times(for our case, 3) over U and V asymptotically captures the real distribution of the samples.

From this we get a better estimate of $\text{Var}[\mathbf{u}_i^T\mathbf{v}_j \mid R_{\mathcal{O}}]$ and select the highest value for sampling.

4.4.2 Performance

MCMC Maximum Variance search was done on figure 4.6 and 4.7. As we see the performance on the synthetic data was better than the Matrix Normal version, being able to always perform better than random sampling. It should be noted that for image discovery, as in 4.7, MCMC performed less well than the clustered search (compare to figure 4.2) - this was observed over multiple runs. Finally the running speed was found to be quite a lot slower, taking 0.8 seconds on average compared to 0.01 for clustered knowledge search to calculate the criteria for sample selection.



PMF with $\lambda = 0.01$ and an initial 1.25% of samples discovered

Figure 4.6: MCMC Maximum Variance Search

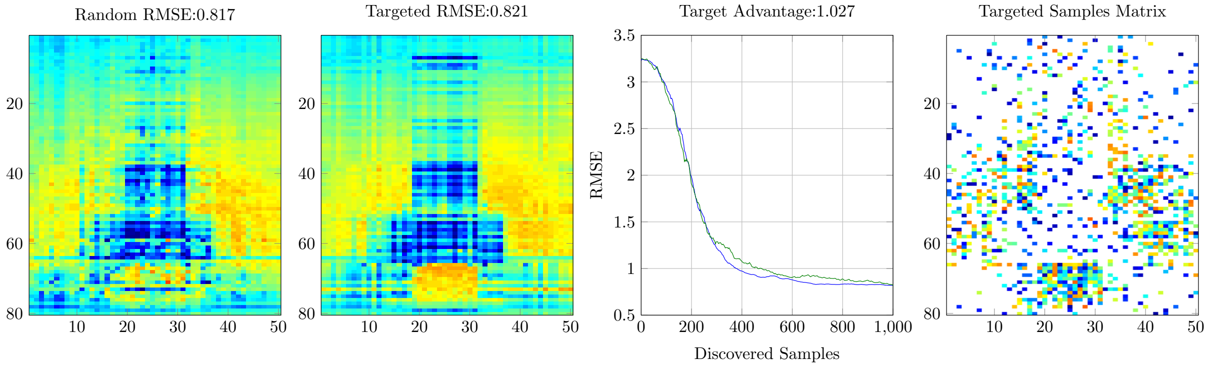


Figure 4.7: MCMC Maximum Variance Search on Eiffel Tower Image

4.5 Lookahead Search

Lookahead search is the idea of inserting multiple values in a dataset and seeing what effect the new value has on the output model [19]. This is a greedy approach as it essentially looks at every single unknown and places all the possible values inside it to see the effect on the model. A value of model quality is taken for all the simulations and from this the sample with the highest quality is requested.

4.5.1 Algorithm

For this greedy approach we will use output values variance as a measure of sample usefulness. A range of values will be inserted into the unknown index i, j and the prediction of unknown values will be saved. Once all the range of values for i, j have been tested the variance of each individual value other each simulated instance is taken, from this the average of the variance of each value is taken and assigned to index i, j of variance matrix Var_R. Once each value of Var_R is completed the sample with the highest mean variance is selected. This would imply that it is the index most likely to impact the model's output. This is represented in algorithm 4.

Algorithm 4 Lookahead Calculation

```
procedure LookaheadVariance($R$,$Z$,$s$)                    ▷ $s$ is number of steps
    $s_{inc} = \frac{maxR - minR}{s}$
    for i=1..N do
        for j=1..M do
            $\mathfrak{V}$ = zeros(M,N,s)                     ▷ $\in \mathcal{R}^{M\times N\times s}$
            for $v$ =minR: $s_{inc}$ :maxR do               ▷ Range with configurable step
                $\mathfrak{V}_v$ = PMF($R + R_{ij} = v$,$Z + Z_{ij} = 1$)   ▷ Train with new value
            end for
            $RVar_{ij}$ = mean(Var$_v(\mathfrak{V})$)        ▷ Average standard deviation across $v$
        end for
    end for
    return $RVar$
end procedure
```

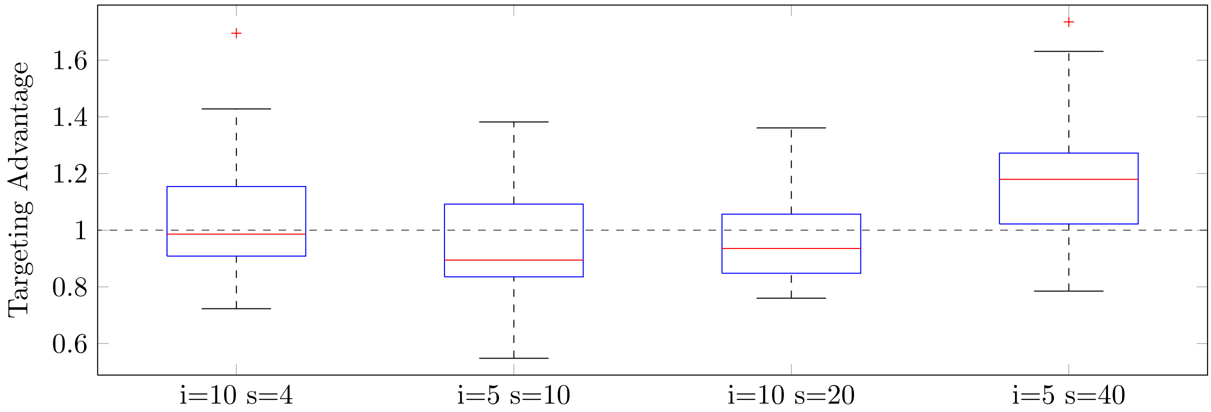
Note that the meaning of variance in the lookahead algorithm and the variance search are different. On one hand the variance of section 4.3 and 4.4 refers to how much the value is expected to change, that is how uncertain we are about it. Here the variance refers to the total possible change over the model - this is essentially a brute force approach to the $\mathbb{E}_q\left[\sum_{kl}\mathrm{Var}_q(R_{kl})\right]$ criteria tried by Sutherland et al. in figure 4.4. However here we look directly at the impact on the model and Sutherland et al. use the variances of the parameters as a proxy.

4.5.2 Performance

Few papers report on the performance of this greedy approach for good reason - the complexity of creating the selection criteria is $O(M^3N^3Dsi)$ where s is the number of values to try on each sample and i is the number of iterations for the PMF to converge.

For this reason the lookahead method was only done on 10 × 10 synthetic data where computation was still lengthy but reasonably fast.

It was observed that selecting the sample that can impact the model the most does not mean a better result as it often selected a sample that led to greater overfitting (as observed by the large variation in output values).



Here us the boxplot of the targeted advantage ratio results from 20 simulations of a randomly generated synthetic matrix each (i.e. a total of 80). Runtime to generate was roughly 2 hours.
i represents the number of samples sampled before lookahead matrix is regenerated - lower is better but more time consuming.
s represents the number of values tested for each unknown sample - higher is preferred as it reflects variance better but is more time consuming.

Figure 4.8: Lookahead Performance on 10 × 10 rank 3 synthetic matrices

Figure 4.8 shows the performance of the lookahead criteria on 10×10 random rank 3 matrices. As it can be seen a good performance is only obtained when many values are tested (s=40) and the matrix variance update (i=5) is low. This is consistent with the expectations that as more samples are available, we reach the real variance of the model. Trying to “cheat” the high complexity of the model by only updating the variance matrix every other new sample or so did not result in good performance.

4.6 Combining CKS with MCMC Variance

In the clustered knowledge search algorithm we only restricted the search space to a range of cells. Other algorithms, such as maximum variance search, targeted specific cells. Search space restriction (CKS) often had good performance time wise but did not fare as well. Single cell selection (such as variance search) often required more computation and fared better when recalculated at each new incoming sample. In the aim to calculate the variance matrix on a less regular basis and retain performance the CKS algorithm was used as a mask to the variance matrix, calculated at a less regular interval.

4.6.1 Algorithm

Combining CKS with MCMC variance search is relatively easy and is outlined in algorithm 5. The CKS matrix is selected and a variance matrix calculated every few samples is used to select the group of minimum valued indices.

Algorithm 5 Combination of CKS and MCMC Variance

```
procedure CKS_MCMC_VAR($U$,$V$,$Z$)                      ▷ $s$ is number of steps
    if $c\%50 == 0$ then                  ▷ recalculate every 50 samples, configurable
        $\mathcal{V} \leftarrow MCMCVar(U, V)$                              ▷ $\in R^{M\times N}$
    end if
    $c \leftarrow c + 1$
    $\mathcal{K} \leftarrow$Clus_Know_Search$(U, V, Z)$                      ▷ $\in R^{M\times N}$
    $\mathcal{K}(\mathcal{K} \neq \min(\mathcal{K})) \leftarrow 0$                 ▷ Set all non minimum values to 0
    $\mathcal{K}(\mathcal{K} = \min(\mathcal{K})) \leftarrow \mathcal{V}_{\mathcal{K}=\min(\mathcal{K})}$     ▷ Min values equal to the variance in $\mathcal{V}$
    $x, y =$MaxIndex$(\mathcal{K})$
    return $x, y$                          ▷ Return subgroup index of max variance
end procedure
```

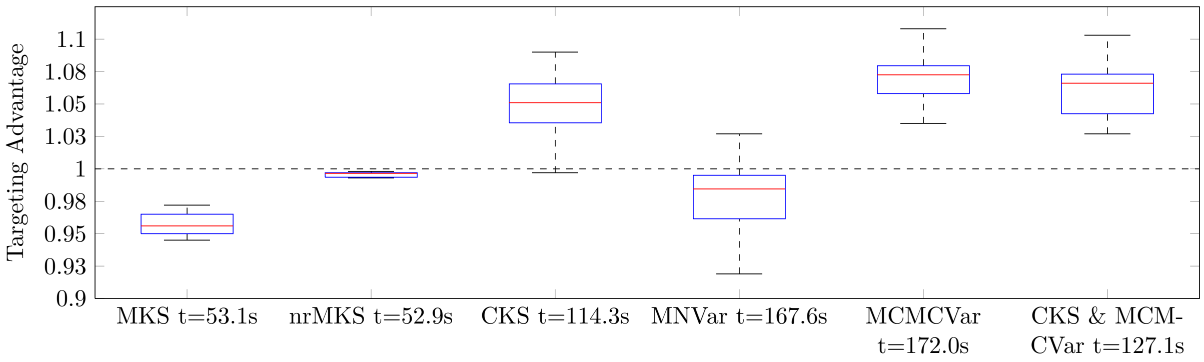
The performance of this hybrid is evaluated in section 4.7.

4.6.2 Improvements

From Figure 4.9 we see that this hybrid approach performs well with reasonably good efficiency. Some ways to improve it could deal with the difference in clusters that CKS uses to target samples. For example there may be a very large group of samples all belonging to the same cluster (say cluster A) for which 4 % is known - enough to correctly infer various properties. We have a much smaller group, cluster B, for which 5 % is known, but as this is a small cluster this is not enough to carry out correct inferences yet this group will not be targeted. Taking the

MCMC Variance into account and having a voting algorithm to decide whether to really target the least known about group would be a step forward.

4.7 Comparing Algorithms



MKS:	Minimum Knowledge Search	**nrMKS:**	non random MKS
CKS:	Clustered Knowledge Search	**MNVar:**	Matrix Normal Variance Search
MCMCVar:	Markov Chain Monte Carlo Variance Search	**CKS & MCMC-var:**	CKS combined with MCMC-Var calculated at 50 sample intervals

Box plot for 20 simulations of active sampling with offline PMF.
t is the time taken for one run of 500 new sample requests.
Values larger than 1 indicate better than random performance.
Lookahead sampling was ignored for computational efficiency reasons.

Figure 4.9: Comparison of multiple runs on 80 × 50 synthetic data

Here the performance of all the encountered algorithms so far is evaluated. For fair comparisons it was compared to the average random sampling performance rather than an individual instance(average of random sampling for 80 × 50 is found in appendix figure B.2). This avoid instances of bad random sampling performance and very good targeted sampling performance creating artificially high values, as in figure 3.4.

The first observation to be made is that the proposed MKS algorithm does not perform well, only ensuring close to expected random performance, the non random variation was found to slightly outperform the random variation due to the initial column and row knowledge maximisation (as seen in the knowledge matrix in figure 3.5).

The CKS search did yield better than random performance nearly consistently, which shows that the targeted approach was successful, though not as well as the more complex MCMCVar one.

Matrix Normal Variance Search (MNVar) did not results in the results found by Sutherland et. al, however this is most likely due to MNVar being sensitive to the parameters used and the parameters found by PMF do not complete empty rows and cells, meaning parameter quality can be very low at the start of the sampling process.

MCMCVar was found to perform best, consistently outdoing random selection, though at the cost of greater complexity. An attempt at reducing complexity while retaining targeting performance is done with the hybrid CKS MCMC algorithm. This resulted in a performance consistently above random sampling and was above to reduce the runtime of 500 sample requests by 27%. While not very useful for small matrices, as active sampling rarely needs to be done in realtime, this is useful for very large databases such as the Netflix one.

Finally the Max-Min value search mentioned in section 3.4 was tested but had an average targeting advantage of 0.8 and was only found to be useful to remove some over and under estimation errors, as expected.

Chapter 5

Evaluation and Observations

5.1 Test on Larger Datasets

The final test was to try active sampling on larger and different datasets. For this the movielens dataset [2], Drugbank [6] and HIVA [18] were used.

5.1.1 Pre-Processing

Most of the time the datasets came in various formats that had to be cleaned up and converted to matlab format. Additionally the movielens dataset had many incomplete entries (unrated movies). This meant creating a subset as non-sparse as possible to split up into useful test, validation and training sets. For the other datasets the data was split into respective training,test and validation sets, with an initial training set of about 2.5 %.

5.1.2 Dealing with large sizes

The main challenge with using large datasets is the time taken, especially due to scaling up with the complexity of datasets. For this reason data was often reduced to manageable datasets and process in batches for training. This is advantageous when algorithms are bounded by $O(MN)$, $O(M^2N^2)$ and higher as halving data results in $O((\frac{M}{2})^2(\frac{N}{2})^2) + O((\frac{M}{2})^2(\frac{N}{2})^2) = \frac{1}{8}O(M^2N^2)$, which results in an improvement of nearly a factor of 10 for splitting it into two equal sets. For more datasets and high complexities this is even more interesting. The main disadvantage is that by reducing the data available to PMF and other algorithms in one batch that the model quality can be reduced. For this reason it was ensured that each batch have roughly 5000 entries or more. This is a variant of the batch splitting that appears in the code published by Ruslan

Salakhutdinov[12] [1].

5.1.3 Results

	CKS	**MCMCVar**	**CKS MCMCVar Comb**
Targeting Advantage	1.036	1.094	1.073
Time (hours)	1.677	2.083	1.833
Time/sample (s)	2	2.5	2.2

Table 5.1: Active Sampling Results on MovieLens Dataset

Figure 5.1 and table 5.1 show the sampling performance on the MovieLens database. This database is very similar in nature to the synthetic one, having discrete values ranging from 1 to 5. However as MovieLens is a "real" database the values are more concentrated around 3, due to users often being and rating a film 3 stars. This explains the better RMSE performance on a set of similar values. The prediction matrices visually show the average rating being closer to 3 (yellow and green representing 3 and 4). Due to time only one run was done for each algorithm and dataset. The performance of the CKS MCMCVar is, as expected, in between CKS and MCMVar in targeting advantage but its runtime is slightly better than the MCMCVar.

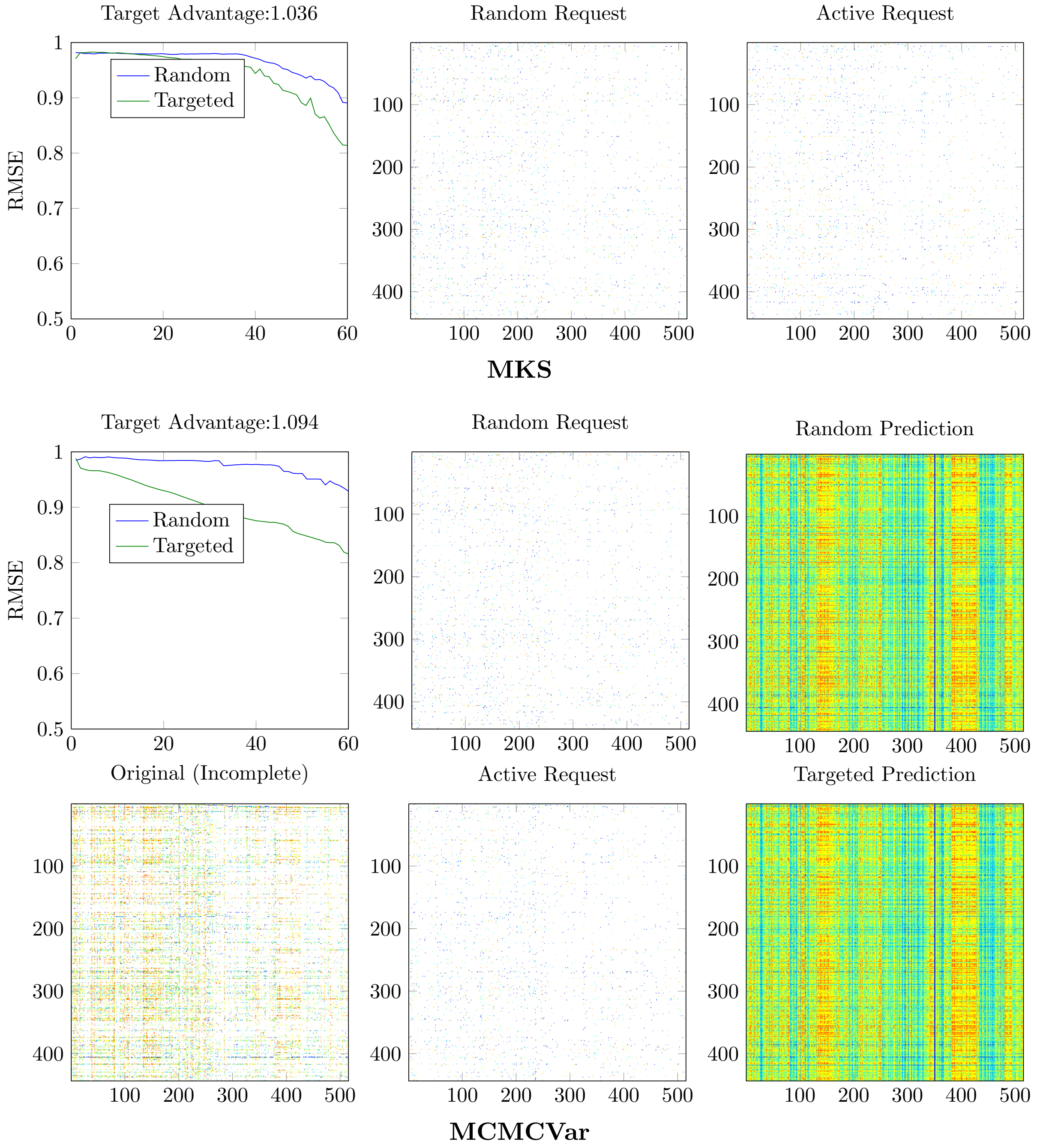
	CKS	**MCMCVar**	**CKS MCMCVar Comb**
Targeting Advantage	1.087	1.046	1.064
Time (hours)	2.64	3.4	3.05
Time/sample (s)	2	2.45	2.2

Note that the number of samples in one iteration is different for the Drugbank Trial than the MovieLens one, explaining different runtimes.

Table 5.2: Active Sampling Results on Drugbank Dataset

Figure 5.2 and table 5.2 show the results of active sampling on the Drugbank dataset. The results are interesting because both active and random sampling algorithms spent some time randomly sampling values (due to having no information at all about the data) and once enough positive (red in the figure) values were found this data was used to successfully target other informative cells, leading to active sampling nearly always out-beating random sampling. On the downside the quality of the output for both random and active sampling was not very

[1] http://www.cs.toronto.edu/~rsalakhu/BPMF.html

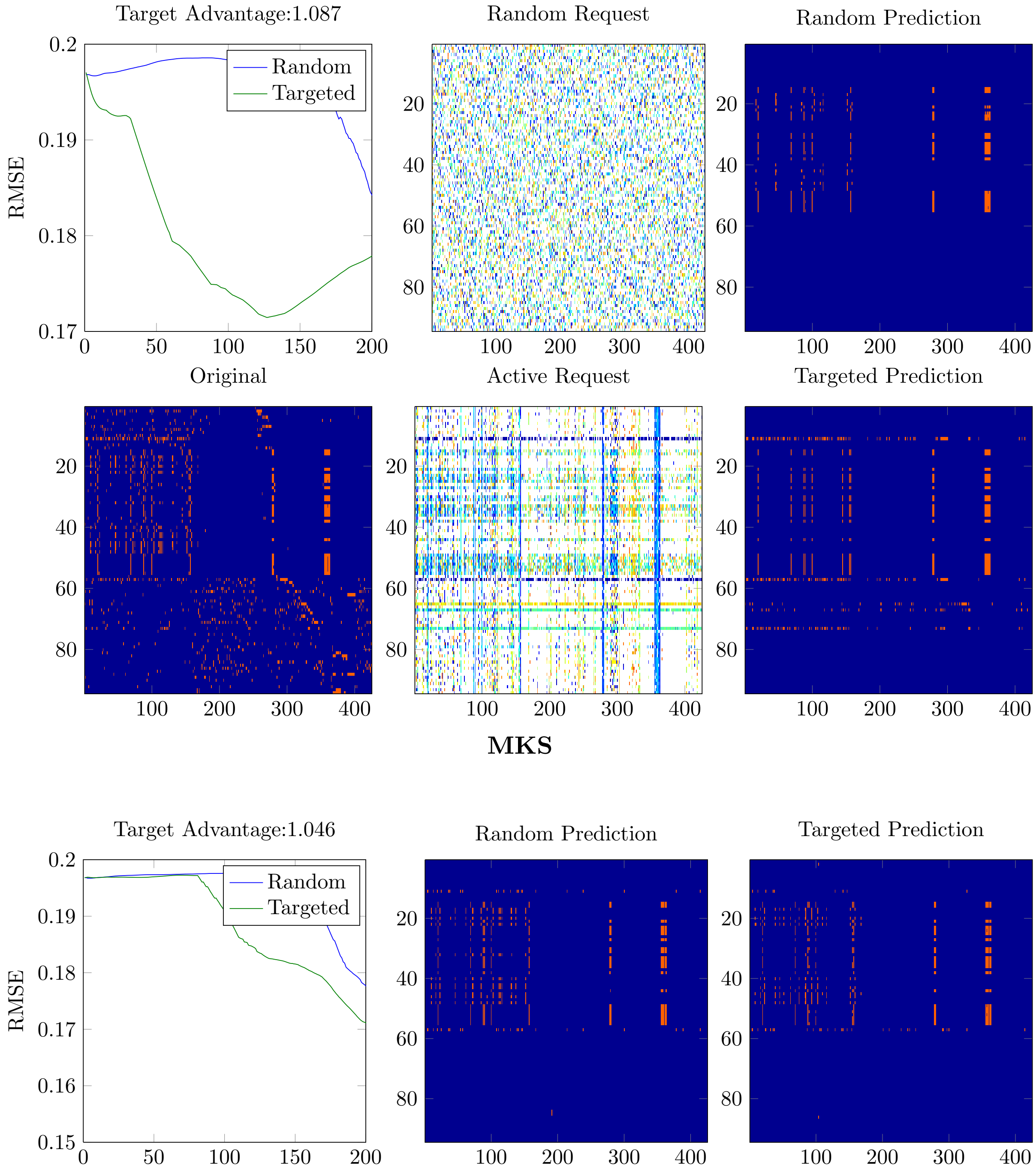


MKS

MCMCVar

Each iteration represents adding a batch of 50 samples based on the selection criteria.
The predictor used was PMF with 20 features, $\lambda = 0.01$ and $\mu = 0.005$.
The Active and Random request matrix may be better seen in colour.

Figure 5.1: MovieLens Dataset Active Sampling Test on 443 × 515 subgroup batch



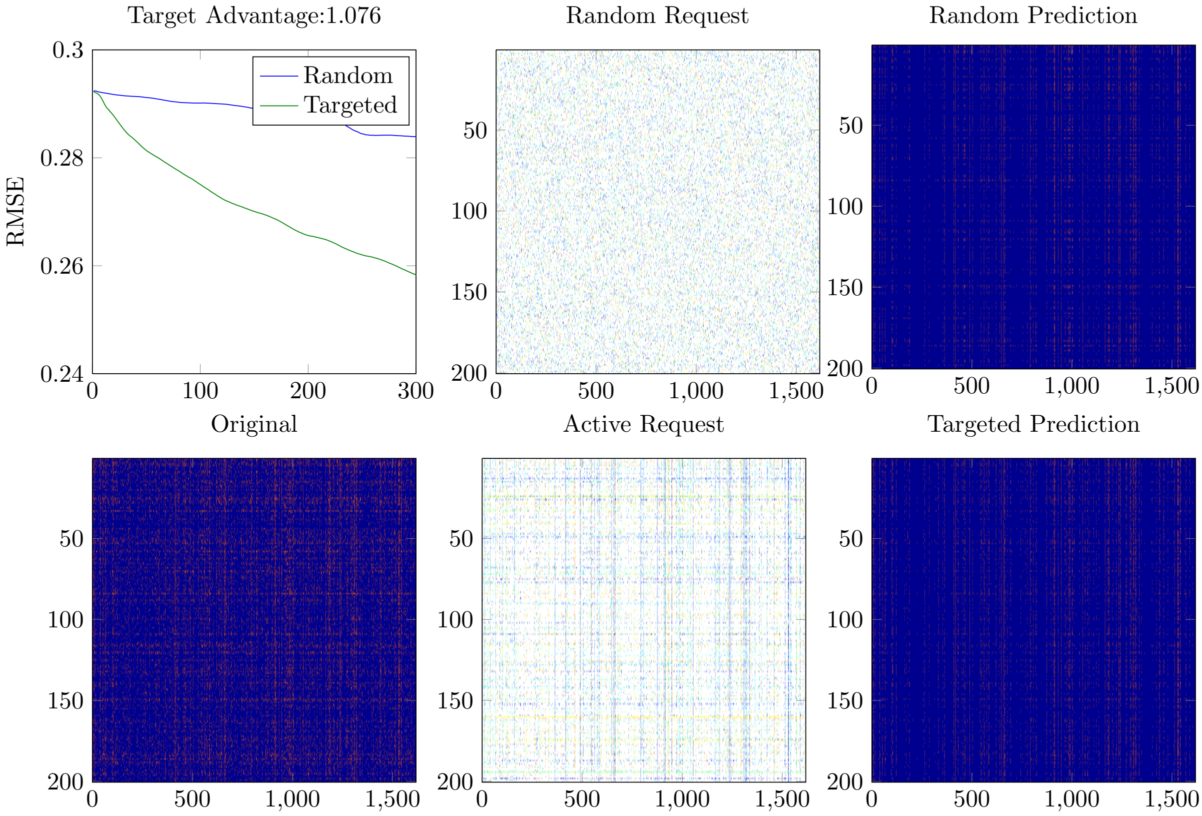
MKS

MCMCVar with PPMF [13]

Drugbank 94 × 425 subgroup.
Each iteration represents adding a batch of 75 samples based on the selection criteria.
The predictor used was PMF with 15 features, $\lambda = 0.01$ and $\mu = 0.005$ for CKS.
The predictor for MCMCVar and MCMCVAr/CKS was PPMF with 15 features.

Figure 5.2: Drugbank Dataset Active Sampling

satisfactory. This is why the PPMF matrix completion variant was tested, though not found as useful as hoped.



HIVA 200 × 1617 subgroup.
Each iteration represents adding a batch of 100 samples based on the selection criteria.
Run too 4 hours and 30 minutes.
The predictor was with 15 features.

Figure 5.3: HIVA Dataset Active Sampling: MCMC Variance Search

Figure 5.3 contains the result of MCMC Variance Search on the HIVA dataset, a binary biological dataset like Drugank. Here MCMCVar performed well instantly recovered the global picture faster than random selection.

5.1.4 General Observations

Often some algorithms would perform near random sampling performance on synthetic datasets such as CKS. However when using "real", larger scale, data the sampling performance was nearly always superior to random - potentially due to there being a higher likelihood of not being any useful data to sample and a guided approach scales better.

5.2 Notes on General Active Sampling Performance

5.2.1 Future Work and Improvements

The field of active sampling often explore various aims with the aim of improving to improve a model performance, such as minimising the variance in latent space to maximising local knowledge. At the beginning of the run the maximum model variance methods (lookahead) were found to perform better due to improving the certainty of unknown latent features but often lost their advantage later on in the sampling process as these features became more certain (this partly explains the large variance of the lookahead search). An individual element variance search was then found to perform better. Like the winner of the Netflix prize, a good active sampling algorithm would be a mixture of different kinds. This is partly what the CKS and MCMCVar hybrid did,by combining an algorithm that seeks to find more about badly classified items and MCMCVar which targets by uncertainty , though the aim was to try and retain MCMCVar performance and improving runtime.

Thus the ideal one so far would start by an algorithm that first targets less known groups to be able to get good features and then perform a minimum/maximum search to avoid over and underestimation errors. Finally it would request all other samples on an uncertainty basis. A quick implementation of CKS followed by min-max search finishing by MCMCVar was tested on toy synthetic matrices but did not yield good performance. The main issues were around focusing on when was best to switch and how much to compromise on a period of worse RMSE to collect more information (CKS sometimes got beaten by random sampling but greatly helped later on in MCMCVar search).

Given more time a method based on the RMSE gradient would have been tested.

On a more general basis active sampling methods taking advantage of different matrix factorisation methods would be interesting. For example some systems derived specifically for movie recommendation take extra data, such as movie year release to take into account the decay of ratings over time. Others such as MMMF act more like Support Vector Machines [15] and inherently have active sampling relevant information such as uncertainty included in the model.

5.2.2 Real-World Application Consideration

Active Sampling often gives a row and a column coordinate to target. However sometimes, for example on Amazon, we may want to only find about the best product to ask a user (i.e we are restricted to a single row). Using the criteria given by active sampling locally (i.e. choosing the local row optimum active search criteria) may result in an improvement of the model overall rather than the specific user (i.e. it may be better at helping determine the average product rating rather than specific ratings the user gets). Thus a focus on algorithms able to find the best item to enquire about to maximise user knowledge may be better short term if we aim to serve better a loyal customer.

5.3 Observations on RMSE

Throughout this project the main measure of performance was RMSE. This was used as the main objective numerical measure but often during simulations visual feedback was also used - as put in evidence by the included figures. It was also often found that a good RMSE is easily achievable simply by guessing the average value. For example the RMSE of KPMF in figure 2.3 was nearly 50% better than the one for PMF and still better than the one for BPMF. Yet for the guessed values, PMF and BPMF look closer to reality and in fact BPMF seems to better capture the distribution of values. Another example is in figure 2.5 where we have two examples of PMF done to restore a picture. The higher regularisation parameter was found to “smooth” out the picture, but with a similar effect to JPEG compression corruption. Despite this it achieved a very similar RMSE to the lower regularisation parameter. In fact other matrix factorisation variants tested during initial search stage, such as KPMF and PPMF[13], did achieve better RMSEs than PMF but were not chosen simply because they were found to only be good for certain situations (PPMF fared better for binary data) and often were just better than PMF at “guessing average”.

In essence using RMSE does not encourage risk taking in guessing as the penalty is high and instead it favours the average. This problem was found throughout the project and meant that some active sampling techniques which sometimes gave a better visual output (such as MKS) had a worse RMSE due to “taking risks” in guessing some values that lie on the boundaries of the distribution of the data. This is a major problem as it is the outliers or edge case values that are of most interest to us as they are the rarest and often the most valuable. For example in drug interaction prediction, guessing that most drugs have a neutral reaction is a very good start, however the real game changer is guessing when a bad or very good interaction will occur, which is not possible when attempting to guess these outliers is penalised.

This is not a newly discovered problem[20] and is not easily solved by changing error calculation. Thus visual feedback is always preferred when available.

Chapter 6

Conclusion

This project has briefly introduced us to the Probabilistic Matrix Factorisation variant of matrix completion methods and their applications to various situations including movie recommendation, drug discovery and image restoration. From this many essential concepts and algorithms relating to machine learning were covered such as regularisation, cost function optimisation and clustering. With this knowledge, active sampling was introduced as a formal method to request new samples from a matrix formatted dataset with the goal of increasing model performance. The motivation for active sampling included helping the scientific decision process as well as knowing what products are best to ask a user to improve recommendations.

The active sampling algorithms presented included some already existent ones, as well as a greedy approach. Two self proposed algorithms are proposed. One called Minimum Knowledge Search is very basic and serves more an educational purpose. The second one called Clustered Knowledge Search takes some of the concepts learnt from other algorithms to create a similarly performing sampling algorithm.

Performance was first tested on smaller datasets and then on larger ones - with satisfactory performance of the proposed Clustered Knowledge Search which performs similarly to more complex algorithms with a smaller overhead, and was noticeably so on very large datasets - though at the cost of targeting accuracy. The higher targeting efficiency of the Markov Chain Monte Carlo Variational Search technique was also appreciated for its good performance.

References

[1] Alan Genz. Hastings-Metropolis for Integration Problems, 2011. URL http://www.math.wsu.edu/faculty/genz/416/lect/l10-4.pdf. 37

[2] GroupLens. MovieLens 100k Dataset. URL http://grouplens.org/datasets/movielens/. 45

[3] A.K. Gupta and D.K. Nagar. *Matrix Variate Distributions*. Monographs and Surveys in Pure and Applied Mathematics. Taylor & Francis, 1999. ISBN 9781584880462. URL http://books.google.co.uk/books?id=PQOYnT7P1loC. 35

[4] L. Isserlis. On a Formula for the Product-Moment Coefficient of any Order of a Normal Frequency Distribution in any Number of Variables. *Biometrika*, 12(1/2):134–139, 1918. ISSN 00063444. URL http://www.jstor.org/stable/2331932. 34

[5] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering High-dimensional Data: A Survey on Subspace Clustering, Pattern-based Clustering, and Correlation Clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, Mrz 2009. ISSN 1556-4681. doi: 10.1145/1497577.1497578. URL http://doi.acm.org/10.1145/1497577.1497578. 32

[6] Vivian Law, Craig Knox, Yannick Djoumbou, Tim Jewison, An Chi Guo, Yifeng Liu, Adam Maciejewski, David Arndt, Michael Wilson, Vanessa Neveu, et al. DrugBank 4.0: shedding new light on drug metabolism. *Nucleic acids research*, 42(D1), 2014. URL http://www.drugbank.ca. 45

[7] Chih-Jen Lin. Projected Gradient Methods for Nonnegative Matrix Factorization. In *Neural Computation*, Seiten 2756–2779, 2007. URL http://dx.doi.org/10.1162/neco.2007.19.10.2756. 8

[8] 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*, Seiten 1–8, June 2012. doi: 10.1109/IJCNN.2012.6252670. URL http://dx.doi.org/10.1109/IJCNN.2012.6252670. 16

[9] MatlabCookbook. K-Means Validating cluster identity, 2013. URL http://www.matlab-cookbook.com/recipes/0100_Statistics/150_kmeans_clustering.html. 28

[10] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul Kantor. *Introduction to Recommender Systems Handbook*. 2011. ISBN 9780387858203. doi: 10.1007/978-0-387-85820-3_1. URL http://dx.doi.org/10.1007/978-0-387-85820-3_1. 22

[11] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic Matrix Factorization, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.127.6198. 3

[12] Ruslan Salakhutdinov and Andriy Mnih. Bayesian Probabilistic Matrix Factorization using Markov chain Monte Carlo. In *Proceedings of the 25th International Conference on Machine Learning*, Seiten 880–887, 2008. URL http://dx.doi.org/10.1145/1390156.1390267. 7, 9, 37, 46

[13] Hanhuai Shan and Arindam Banerjee. Generalized probabilistic matrix factorizations for collaborative filtering. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, Seiten 1025–1030. IEEE, 2010. URL www-users.cs.umn.edu/~shan/icdm10_gpmf.pdf. 48, 51

[14] 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*, Seiten 325–333, 2012. URL http://doi.acm.org/10.1145/2339530.2339584. 4

[15] 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. 3, 32, 33, 35, 37, 50

[16] Dougal J. Sutherland, Barnabas Poczos, and Jeff Schneider. Active Learning and Search on Low-Rank Matrices Slides - Variational PMF, 2013. URL http://www.cs.cmu.edu/~dsutherl/active-pmf/kdd-13-slides.pdf. 35

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

[18] The National Cancer Institute (USA). HIVA dataset for ETHZ Active Learning Challenge. URL http://dtp.nci.nih.gov/docs/aids/aids_data.html. 45

[19] Xuezhi Wang, Roman Garnett, and Jeff Schneider. Bayesian Active Search on Graphs. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, Seiten 731–738, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2174-7. doi: 10.1145/2487575.2487605. URL http://doi.acm.org/10.1145/2487575.2487605. 40

[20] Zhou Wang and Alan C. Bovik. Mean Squared Error: Love it or leave it? *IEEE SIGNAL PROCESSING MAGAZINE*, Seiten 98–117, 2009. URL http://dx.doi.org/10.1109/MSP.2008.930649. 51

[21] 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, 13

[22] ETH Zurich. Active Learning Challenge, 2010. URL http://www.causality.inf.ethz.ch/activelearning.php?page=datasets. 3

Appendix A

A.1 Probabilistic Matrix Factorization

A.1.1 Details of derivation

In equation 2.5 we had a partial expression of log maximum likelihood the full equation is:

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

Thus the remaining is still dependant on the parameters but independant of U and V.

The Frobenius Norm is defined as:

$$\|A\|_{Fro} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\text{trace}(A^* A)} \tag{1}$$

Finally for practicality we symmetrise λ_U and λ_V from equation 2.6 to be a single regularisation parameter λ.

A.1.2 λ as a Lagrange multiplier

In the situation that we want to optimise a function $f(x)$ with a constraint $g(x) = c$ we have:

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

It is easy to see how the λ from equation 2.6 also corresponds to a Lagrange parameter. Indeed we have $\sum_{i=1}^{N}\sum_{j=1}^{M} Z_{ij}(R_{ij}-U_i^T V_j)^2$ as $f(x)$ and $\sum_{i=1}^{N}\|U_i\|_{Fro}^2 + \sum_{j=1}^{M}\|V_j\|_{Fro}^2$ as $g(x)$ with $c = 0$. Thus essentially we are restricting U and V to be close to zero - essential if we are to avoid overfitting. This is why we do not want $\lambda = 0$ as it would remove this constraint. In fact using the Widrow-Hoff learning rule to update λ iteratively would cause it to tend to zero and remove the constraint, explaining why this parameter cannot be optimised like U and V.

A.2 Bayesian Probabilistic Matrix Factorization

The parameters of BPMF are placed upon a Gaussian-Wishart distribution with the parameter W_0:

$$\mathcal{W}(\Lambda|W_0, \nu_0) = \frac{1}{C}|\Lambda|^{(\nu_0-D-1)/2} \exp\left(-\frac{1}{2}\mathrm{Tr}(W_0^{-1}\Lambda)\right)$$

C is a normalising constant.

Details on the Gibbs sampling process are found in section 4.4.

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

Synthetic Data was generated to replicate a movie database, with a few users types and the other users being some variation of their group. For example there would be the base “16 year old boy” profile as well as the “young adult” profile. From this random users with a varying mix (for example 80% 16 year old and 20% young adult) between the base profiles are made.

The code used is the following:

Listing 1: Code to generate synthetic Data

```
%-------------------------------------------
% Correlated matrix creation                 |
%--------------------------------------------|
% This function creates a matrix with base   |
% row profiles (i.e. users) and randomly     |
% generates their preferences. The remaining |
% users are a mixture of the base type users.|
%--------------------------------------------|

clear all
clc

% types of users
user_t = 3;
% Rank of matrix
rank = 3;
% total number of users
users = 9;
```

```
% movies
columns = 16;
% Randomly generate
% round_dec(x,1) rounds number to first decimal
% rand_dual is a two ended Gaussian random generator.
U = round_dec(rand_dual(user_t,rank),1);
V = round_dec(rand_dual(rank,columns),1);
% Loop through base users and create correlations
for i=user_t+1:users
    % randomly choose association
    profile = randsample(1:user_t,randsample(2:user_t-1,1));
    k = length(profile);
    % correlation probabilities
    prob = zeros(k,1);
    for j=1:k
        if (j ≠ 1) && (j ≠ k);
            prob(j) = round_dec((1 - sum(prob))*rand(1),1);
        end
        if j == 1;
            prob(j) = round_dec(rand(1),1);
        end
        if j == k;
            prob(j) = 1 - sum(prob);
        end
    end
    for j=1:k
        U(i,:) =  prob'*U(profile,:);
    end
end
% Plot to check
R = U*V;
R = round(R/max(max(R))*5);
R(R==0) = 5;
hist(R(:))
figure(2)
image(R/max(max(R))*50)
```

A.4 Clustering

Clustering is used in the CKS algorithm, section 4.1.

For this the K-Means algorithm was used due to its simplicity. However other more complex algorithms could have been used - though this was not the focus of research on this report.

A.4.1 K-Means

The K-Means algorithm works given a set $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n]$ that we want to cluster around k clusters[1]. For this we minimise the cost function:

[1] $k < n$

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

With μ_i being the mean vector belonging to the ith cluster.

The optimum is typically found by iterative refinement in a similar way to mean squares estimation.

Appendix B

B.1 Test Data and Figures

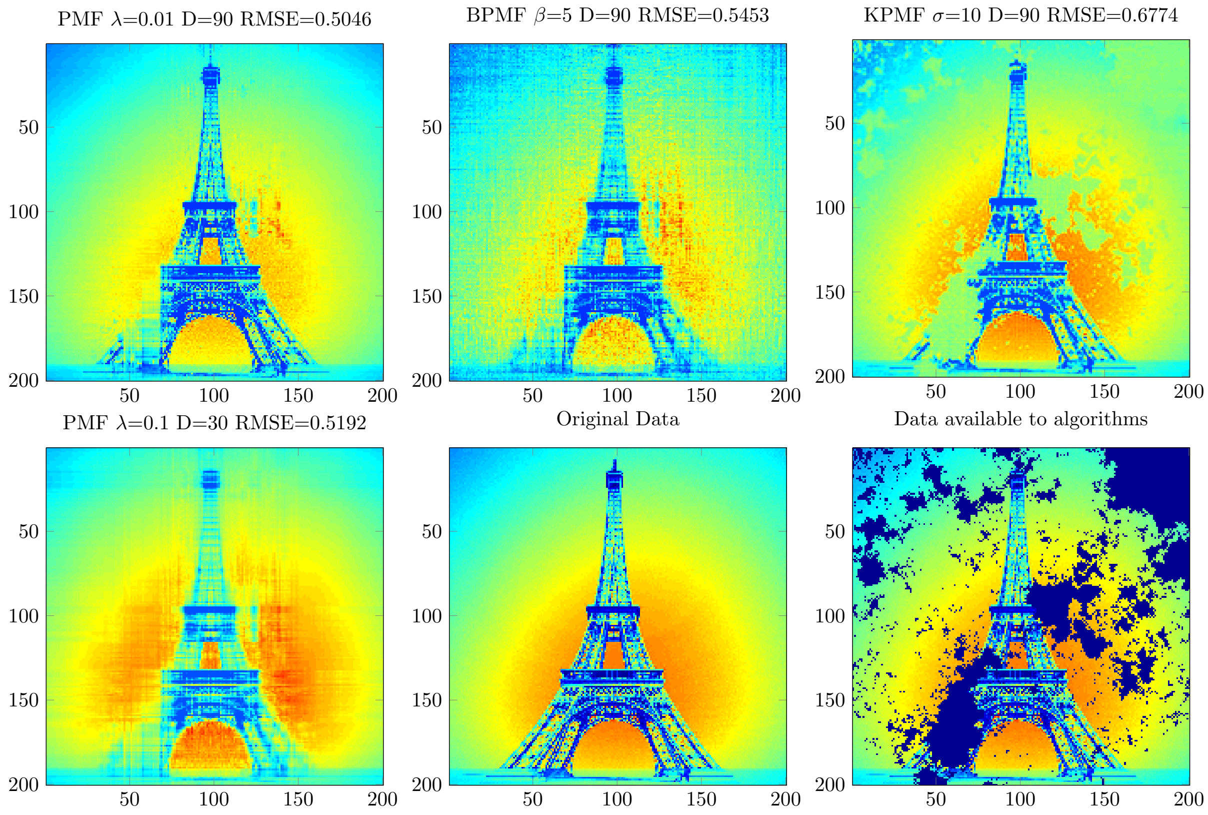
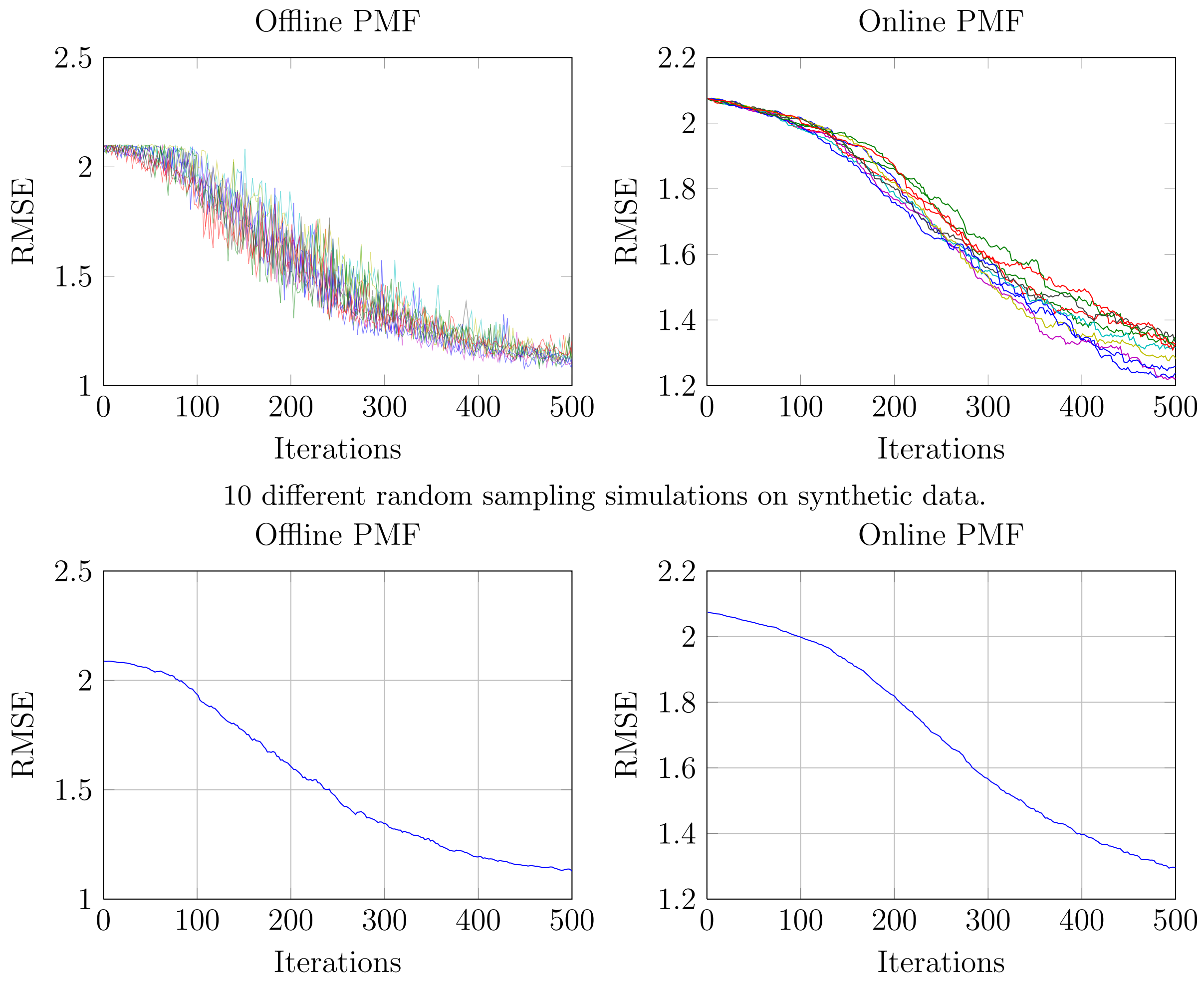


Figure B.1: Image Restoration with Matrix Factorisation with 75% complete data

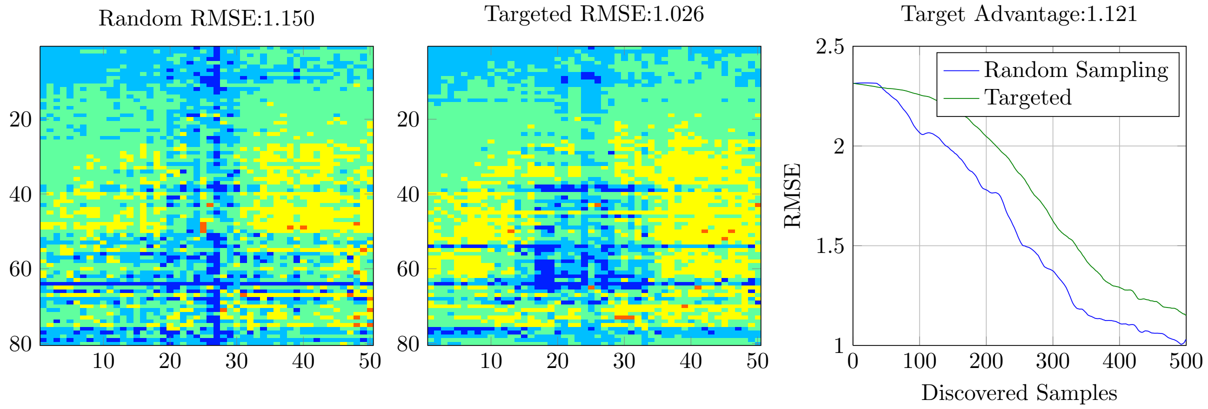


10 different random sampling simulations on synthetic data.

Average of 10 simulations.

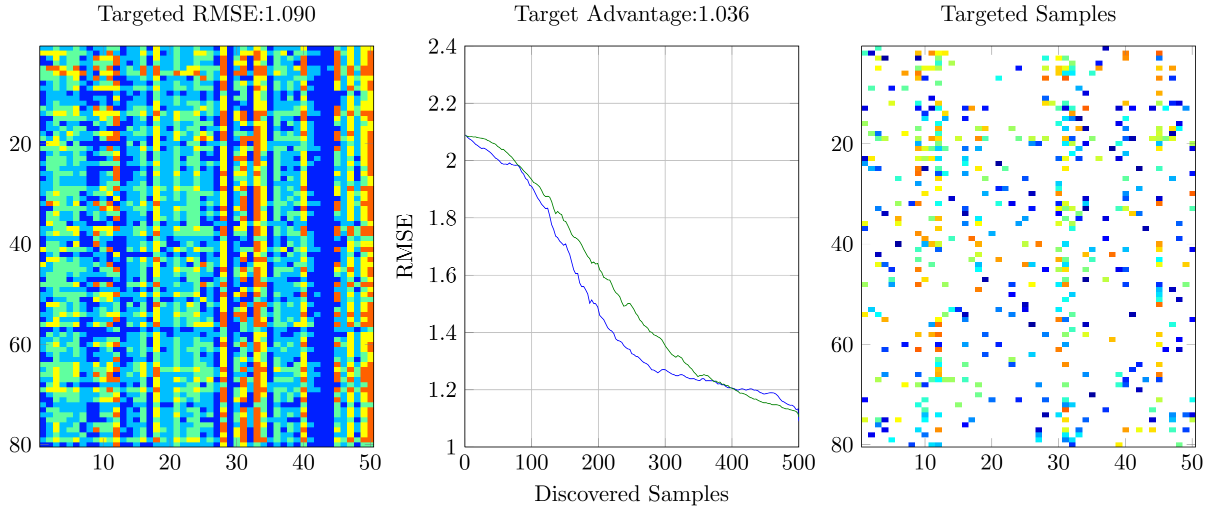
Original data complete at 1.25% (with another 1.25% as validation data)
$\lambda = 0.01$, $D = 30$

Figure B.2: Random Sampling Performance



Original data complete at 1.25% (with another 1.25% as validation data)
Data is downsampled and rounded Eiffel tower image to fit $[1, 5]$.
Online PMF, $\lambda = 0.01$, $D = 30$

Figure B.3: Minimum Knowledge Search on Eiffel Tower



Average of 10 simulations.

Original data complete at 1.25% (with another 1.25% as validation data)
Note how the performance at the beginning is very good but as soon as information about most clusters is gained performance is worse due to an area of little information in a cluster of high information being ignored.
$\lambda = 0.01$, $D = 7$

Figure B.4: CKS Performance on Synthetic Data - Local Cluster Oversight

B.2 Code

This contains samples of the main code used throughout the project. The full code can be found on https://github.com/sg3510/al_proj/

B.2.1 PMF

A version with validation set and overfitting is present on github.

Listing 2: PMF Gradient Descent

```
%--------------------------------------------
% Probabilistic Matrix Factorisation         |
% - Based on work of Ruslan Salakhutdinov and|
%   Andriy Mnih                              |
%--------------------------------------------|
% Seb Grubb - sg3510@ic.ac.uk                |
%--------------------------------------------|
% This function returns a matrix             |
% decomposition for a partially completed    |
% matrix in U and V variables.               |
%====================Inputs==================|
% - R: incomplete matrix to use for training |
% - z_m: mask matrix, 1 for known value, 0   |
%        otherwise                           |
% - iter: number of iterations               |
% - num_feat: number of latent features      |
% - lambda: regularisation parameter         |
% - epsilon: leanring rate                   |
%====================Outputs=================|
% - U,V: feature matrices, aim to reconstruct|
%        matrix by U'*V+mean_r = R_estimate  |
% - e: training error vector                 |
% - mean_r: mean value of R for known values |
%--------------------------------------------|
function [U,V,e, mean_r ] = pmf_func(R, z_m, iter,num_feat,lambda,epsilon)
    % get size
    [x,y] = size(R);
    %remove mean
    mean_r = sum(sum(R))/samples;
    R = R - mean_r;
    % Latent matrix initialisation
    U = 0.01*randn(x, num_feat);
    V = 0.01*randn(y, num_feat);
    sample_no = sum(sum(z_m));
    % start training
    for step = 1:iter
        for i = 1:x
            for j = 1:y
                if z_m(i,j) == 1
                    eij = R(i,j) - U(i,:)*V(j,:)';
                    if eij > 100000
                        eij
                    end
                    for k=1:num_feat
                        U(i,k) = U(i,k) + epsilon * (2*eij * V(j,k) - lambda ...
                            * U(i,k));
                        V(j,k) = V(j,k) + epsilon * (2*eij * U(i,k) - lambda ...
                            * V(j,k));
                    end
                end
            end
        end
        e(step) = sqrt(sum(sum((z_m.*(R-U*V').^2)))/sample_no);
    end
end
```

B.2.2 CKS

Listing 3: Clustered Knowledge Selection criteria generation

```
%--------------------------------------------
% Clustered Knowledge Search                 |
%--------------------------------------------|
% Seb Grubb - sg3510@ic.ac.uk                |
%--------------------------------------------|
% This function returns a knowledge matrix of|
% areas with a rating of their knowledge     |
% on the clusters the the feature matrices.  |
%===================Inputs===================|
% - U,V: feature matrices learnt from R      |
% - z_m: mask matrix, 1 for known value, 0   |
%        otherwise                           |
%===================Outputs==================|
% - knowledge: matrix of same size as R with |
%              values representing the amount|
%              known of the cell based on    |
%              cluster.                      |
%--------------------------------------------|
function [ knowledge ] = CKS_UV_cluster(U,V,z_m)
% Cluster UV by features
% Determine best cluster size
k_u = cluster_det(U,2,round(2*log(length(U))));
k_v = cluster_det(V,2,round(2*log(length(V))));
% Error handling
while 1
    try
        u_clusters = kmeans(U,k_u);
        v_clusters = kmeans(V,k_v);
        break
    catch
        k_u = k_u - 1;
        k_v = k_v - 1;
        fprintf('Reducing cluster size k_u = %d k_v = %d',k_u,k_v)
    end
end
% create knowledge variables
u_info = mean(z_m,2); u_knowledge = zeros(1,k_u);
v_info = mean(z_m,1); v_knowledge = zeros(1,k_u);
for i =1:k_u
    u_knowledge(i) = sum(u_info(u_clusters==i));
end

for i =1:k_v
    v_knowledge(i) = sum(v_info(v_clusters==i));
end

%create knowledge matrix
a = zeros(length(U),1);
b = zeros(length(V),1);

% Assign amount known
for i =1:k_u
    a(u_clusters==i) = u_knowledge(i);
end
for i =1:k_v
    b(v_clusters==i) = v_knowledge(i);
end

knowledge = a*b';

end
```

B.2.3 Matrix-Normal Variance calculation

Listing 4: Minimum Variance Calculation

```
%--------------------------------------------
% Matrix Normal Variance Calculation         |
% Based on work of Sutherland et al.         |
%--------------------------------------------|
% Seb Grubb - sg3510@ic.ac.uk                |
%--------------------------------------------|
% This function returns a matrix of the      |
% estimated variance Var[u_i^Tv_j|R_O].     |
% As E[sigma^2] is ignored this is only      |
% useful for relative variance investigation.|
%====================Inputs==================|
% - U,V: feature matrices learnt from R      |
% - z_m: mask matrix, 1 for known value, 0   |
%        otherwise                           |
%====================Outputs=================|
% - R_var: matrix of same size as R with     |
%              values representing estimated |
%              sample variance based on input|
%              parameters.                   |
%---------------------------------------------|

function R_var = MN_V(U,V,z_m)
U_V = [U;V];
row_cov = cov(U_V');
dim_cov = cov(U_V);

[x,¬] = size(U);
[y,d] = size(V);
R_var = zeros(x,y);
% Speed optimisation variables
E_ijk = zeros(x,y,d);
E_ijk_m = zeros(x,y,d);
% Attempt at optimisation - makes runtime worse
% E_ijkl = zeros(x,y,d,d);
% E_ijkl_m = zeros(x,y,d,d);
for i=1:x
    for j=1:y
        % E[Uki Vkj Uli Vlj] - E[Uki Vkj] E[Uli Vlj]
        % E[XaXb] =  a b  + Sigma(a,b.)
        if z_m(i,j) == 1
            for k=1:d
                for l=1:d
                    % Speed optimisation
                    if E_ijk_m(i,j,k) == 0
                        E_ijk_m(i,j,k) = 1;
                        E_ijk(i,j,k) = expect_norm2(U,V,row_cov,dim_cov,i,j,k);
                    end
                    if E_ijk_m(i,j,l) == 0
                        E_ijk_m(i,j,l) = 1;
                        E_ijk(i,j,l) = expect_norm2(U,V,row_cov,dim_cov,i,j,l);
                    end
    % Found to make performance worse
    %                 if E_ijkl_m(i,j,k,l) == 0
    %                     E_ijkl_m(i,j,k,l) = 1;
    %                     E_ijkl_m(i,j,l,k) = 1;
    %                     E_ijkl(i,j,k,l) = ...
        expect_norm4(U,V,row_cov,dim_cov,i,j,k,l);
    %                     E_ijkl(i,j,l,k) = E_ijkl(i,j,k,l);
    %                 end
                    R_var(i,j) = expect_norm4(U,V,row_cov,dim_cov,i,j,k,l) - ...
                        E_ijk(i,j,k)*E_ijk(i,j,l);

    % Original non-optimised code
    %                 R_var(i,j) = expect_norm4(U,V,row_cov,dim_cov,i,j,k,l) ...
        - ...
        expect_norm2(U,V,row_cov,dim_cov,i,j,k)*expect_norm2(U,V,row_cov,dim_cov,i,j,l);
                end
            end
        end
    end
```

```
end
end
```

Listing 5: 2 Input sample expectation based on parameters

```
function [ out ] = expect_norm2(U,V,row_cov,dim_cov,i,j,k)
    %E[XaXb] =  a b  + Sigma(a,b).
    %E[Uki Vkj]
    % Sigma(a,b) = Sigma(i,j)Sigma(k,l)
    out = U(i,k)*V(j,k) + row_cov(i,j+length(U))*dim_cov(k,k);
end
```

Listing 6: 4 Input sample expectation based on parameters

```
function [ out ] = expect_norm2(U,V,row_cov,dim_cov,i,j,k)
    %E[XaXb] =  a b  + Sigma(a,b).
    %E[Uki Vkj]
    % Sigma(a,b) = Sigma(i,j)Sigma(k,l)
    out = U(i,k)*V(j,k) + row_cov(i,j+length(U))*dim_cov(k,k);
end
```

B.2.4 Main Runtime Code for active sampling simulation

Listing 7: Code base for Active Sampling

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% - Active Sample Selection for Matrix Completion -
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  - Sebastian Grubb - sg3510@ic.ac.uk
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% - Description -
% This allows a dataset and selection criteria to be chosen allowing the
% active sampling algorithm to be tested.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Setup

% Clear Environment
clc
clear all
close all

% Add File Paths
addpath('data')
addpath('mat2tikz')
addpath('matrixfactoriser')
addpath('routines')
```

```

% Load Data
load('hiva_subset.mat')
R_full = R_train + R_test + R_val;
discrete = 1; % 1 - Discrete, 0 - Continuous
% Min and Max if discrete
min_d = 0;max_d = 1;

% Initialise Main Variables
samples = 500; % Ensure not larger than total set
[x,y] = size(R_train);
mf_type = 4; % 1 - PMF, 2 - BPMF, 3 - KPMF,  4 - PPMF

num_feat = 15;

% PMF Settings
lambda_pmf = 0.01; % Regularisation
mu_pmf = 0.005; % Learning rate
iter_pmf = 100; % Iterations

% BPMF Settings
beta_bpmf = 3.5; % Precision
iter_bpmf = 15; % Iterations

% KMPF settings
iter_kmpf = 75; % Iterations
epsilon_kpmf = 0.005; % Learning rate
diffu_kmpf = 0.1; %Diffusion value if using dissuion kernel
%% Setup for Active Sampling Process

% Initialise sampling variables
freq_count = 0;

% Initialise for Random Sampling
z_train_rand = z_train; z_test_rand = z_test;
R_train_rand = R_train; R_test_rand = R_test;
sample_rand = zeros(x,y);
err_rand = zeros(1,samples);

% Initialise for targeted Sampling
z_train_targ = z_train; z_test_targ = z_test;
R_train_targ = R_train; R_test_targ = R_test;
sample_targ = zeros(x,y);
err_targ = zeros(1,samples);
U_targ = randn(x,num_feat); V_targ = randn(y,num_feat);

% Choose Targeting
% 1 - MKS, 2 - CKS, 3 - MNVar, 4 - MCMCVar, 5 - Hybrid
targeting_type = 4;
% How many samples to wait until knowledge recalculation
% For MCMCVar CKS hybrid this is the MCMCVar reculculation period
% 50 is agood setting
targeting_freq = 10;
select_min = 0; % 0 - Select Max, 1 - Select Min
```

```
randomise = 1; % 1 - Randomise variants, 0 - Deterministic
batch_req = 100; % How many samples to request in one sampling
% process, 1 for true active sampling basis

%% Start Active Sampling
for req = 1:samples
    %-------------------------------
    % Random Sampling Stage
    %-------------------------------

    fprintf('Getting Random Sample(s)\n');
    % Get random sample from test set
    for samp_req = 1:batch_req
        [R_train_rand,z_train_rand, R_test_rand, z_test_rand,a,b] = ...
            random_sample(R_train_rand,z_train_rand, R_test_rand, z_test_rand);
        sample_rand(a,b) = req;
    end

    fprintf('Trainging with Random Sample(s)\n');
    % Train
    switch mf_type
        case 1 % PMF
            [U_rand,V_rand,¬ ] = pmf_func_val(R_train_rand, ...
                z_train_rand,R_val,z_val, iter_pmf,num_feat,lambda_pmf,mu_pmf);
            R_pred_rand = U_rand*V_rand';
        case 2 % BPMF
            %Train first for PMF
            [U_rand,V_rand,¬ ] = pmf_func_val(R_train_rand, ...
                z_train_rand,R_val,z_val, 50,num_feat,lambda_pmf,mu_pmf);
            [U_rand,V_rand,¬,mv_rand] = bpmf_func(R_train_rand,z_train_rand, ...
                U_rand,V_rand,iter_bpmf,num_feat,beta_bpmf);
            R_pred_rand = U_rand*V_rand' + mv_rand;
        case 3 % KPMF
            [U_rand,V_rand,¬] = kpmf_func(R_train_rand,z_train_rand, ...
                iter_kmpf,num_feat,epsilon_kpmf,diffu_kmpf);
            R_pred_rand = U_rand*V_rand';
        case 4 % PPMF
            [ U_rand,V_rand,mv_rand ] = run_ppmf(R_train_rand, ...
                z_train_rand,num_feat, R_val,z_val);
            R_pred_rand = U_rand*V_rand' + mv_rand;
        otherwise
            disp('wrong type of Matrix Factorisation selected'); break;
    end

    if discrete
        R_pred_rand = min_max_round(R_pred_rand,min_d,max_d);
    end

    % RMSE calculation
    err_rand(req) = rmse_calc( R_pred_rand, R_test_rand, z_test_rand);


    %-------------------------------
    % Active Sampling Stage
    %-------------------------------

    fprintf('Getting Targeting Criteria\n');
    % Criteria for targeted
    switch targeting_type
        case 1 % MKS
            knowledge =  mean(z_train_targ,2)*mean(z_train_targ,1);
        case 2 % CKS
            knowledge  = CKS_UV_cluster(U_targ,V_targ, z_train_targ);
        case 3 % MNVar
            if ((mod(freq_count,targeting_freq) == 0)|| freq_count == 1)
                knowledge = MN_V(U_targ,V_targ,1-(z_train_targ+z_val));
                knowledge = knowledge - min(knowledge(:));
                knowledge = knowledge/max(knowledge(:));
            end
            freq_count = freq_count + 1;
        case 4 % MCMCVar
            if ((mod(freq_count,targeting_freq) == 0) || freq_count == 1)
                [U_targ,V_targ,¬,¬] = ...
                    GS_U_V(R_train_targ,z_train_targ,U_targ,V_targ);
                knowledge = MN_V(U_targ,V_targ,1-(z_train_targ+z_val));
                knowledge = knowledge - min(knowledge(:));
                knowledge = knowledge/max(knowledge(:));
            end
            freq_count = freq_count + 1;
        case 5 %MCMCVar and CKS hybrid
            if ((mod(freq_count,targeting_freq) == 0) || freq_count == 1)
                [U_b,V_b,e,mean_rating] = ...
                    GS_U_V(R_train_targ,z_train_targ,U_targ,V_targ);
                var_know = MN_V(U_b,V_b,1-(z_train_targ+z_val));
                var_know = var_know - min(var_know(:));
                var_know = var_know/max(var_know(:));
            end
            [a,b] = find(knowledge==max(knowledge(:)));%a = a(j);b = b(j);
            knowledge(:,:) = 0;
            knowledge(a,b) = var_know(a,b);
        otherwise
            disp('Wrong type of Active Sampling selected'); break;
    end

    fprintf('Getting Targeted Sample(s)\n');
    %Request Samples in batch
    for samp_req = 1:batch_req
        [ R_train_targ,z_train_targ, R_test_targ, z_test_targ,a,b] = ...
            select_sample( R_train_targ,z_train_targ, R_test_targ, ...
            z_test_targ, knowledge, select_min,randomise);
        sample_targ(a,b) = req;
    end

    fprintf('Training with Targeted Samples\n');
    % Train
    switch mf_type
        case 1 % PMF
```

```
            [U_targ,V_targ,¬ ] = pmf_func_val(R_train_targ, ...
                z_train_targ,R_val,z_val, iter_pmf,num_feat,lambda_pmf,mu_pmf);
            R_pred_targ = U_targ*V_targ';
        case 2 % BPMF
            %Train first for PMF
            [U_targ,V_targ,¬ ] = pmf_func_val(R_train_targ, ...
                z_train_targ,R_val,z_val, 50,num_feat,lambda_pmf,mu_pmf);
            [U_targ,V_targ,¬,mv_targ] = bpmf_func(R_train_targ,z_train_targ, ...
                U_targ,V_targ,iter_bpmf,num_feat,beta_bpmf);
            R_pred_targ = U_targ*V_targ' + mv_targ;
        case 3 % KPMF
            [U_targ,V_targ,¬] = kpmf_func(R_train_targ,z_train_targ, ...
                iter_kmpf,num_feat,epsilon_kpmf,diffu_kmpf);
            R_pred_targ = U_targ*V_targ';
        case 4 % PPMF
            [ U_targ,V_targ,mv_targ ] = run_ppmf(R_train_targ, ...
                z_train_targ,num_feat, R_val,z_val);
            R_pred_targ = U_targ*V_targ' + mv_targ;
        otherwise
            disp('wrong type of Matrix Factorisation selected'); break;
    end

    if discrete
        R_pred_targ = min_max_round(R_pred_targ,min_d,max_d);
    end

    % RMSE calculation
    err_targ(req) = rmse_calc( R_pred_targ, R_test_targ, z_test_targ);

    %-------------------------------
    % Draw
    %-------------------------------
    subplot(2,3,1);
    plot([smooth(err_rand(1:req),30) smooth(err_targ(1:req),30)])
    legend('Random','Targeted')
    title(sprintf('Target Advantage:%1.3f',sum(err_rand)/sum(err_targ)))
    ylabel('RMSE')
    subplot(2,3,2);

    image(sample_rand/req*50)
    title('Random Request')
    subplot(2,3,3);
    image(R_pred_rand*50)
    title('Random Prediction')


    subplot(2,3,4);
    image(R_full/max(R_full(:))*50)
    title('Original')
    subplot(2,3,5);

    image(sample_targ/req*50)
    title('Active Request')
    subplot(2,3,6);
    image(R_pred_targ*50)
    title('Targeted Prediction')
    drawnow

end
```