Exploration-Exploitation trade-offs via Probabilistic Matrix Factorization

Master Thesis
Nedyalko Prisadnikov
August 20, 2014

Advisors: Prof. Dr. Andreas Krause, Hastagiri Prakash Vanchinathan
Department of Computer Science, ETH Zürich
Abstract

Movie recommendation has become a hot topic in the recent years. Companies which sell or rent movies have larger and larger catalogs. Therefore, recommending the best movies to the users has become more important with time. Making good recommendations accounts to higher user engagement and higher profits for the companies.

Usually movie recommender systems are aiming at predicting the ratings for each user-movie pair. The most popular solutions for the problem, defined that way, are based on probabilistic matrix factorization. We approached the topic from a different angle. Instead of predicting all ratings, we modeled the recommendation as a sequential decision problem in which we repeatedly suggest movies to a given user and receive feedback about the quality of our suggestions. That setup is similar to the problem of multiarmed bandits.

The main contribution of the current thesis is combining probabilistic matrix factorization with bandit-like algorithms for solving the movie recommendation problem as sequential decision problem. We introduced greedy (purely exploitation) algorithm which had big improvement over a naive non-learning solution. Then we introduced upper confidence bound (UCB) solution which further improved the performance.
# Contents

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>iv</td>
</tr>
</tbody>
</table>

## 1 Introduction

1.1 Recommender systems .......................... 1
  1.1.1 Collaborative filtering .................. 2
  1.1.2 Matrix Factorization ..................... 2
  1.1.3 Multiarmed bandits ....................... 3
1.2 Thesis contributions .......................... 3
1.3 Thesis outline ................................ 4

## 2 Matrix Factorization

2.1 Singular Value Decomposition .................. 5
  2.1.1 Definition .................................. 5
  2.1.2 Closest approximation ..................... 6
  2.1.3 Application in movie recommendations .... 6
  2.1.4 Drawbacks .................................. 7
2.2 Probabilistic approaches ..................... 8
  2.2.1 Probabilistic Matrix Factorization ....... 8
  2.2.2 Bayesian approach ......................... 9

## 3 Multiarmed bandits

3.1 Problem statement ............................ 11
3.2 Simple strategies ............................ 12
  3.2.1 $\epsilon$-greedy ........................ 12
  3.2.2 UCB1 ...................................... 12
3.3 Contextual Bandits ............................ 12

## 4 Mathematical foundations

.................................................... 15
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.5</td>
<td>Greedy solutions (20 features)</td>
<td>34</td>
</tr>
<tr>
<td>6.6</td>
<td>UCB solutions with constant rate $\beta_t$ (10 features)</td>
<td>35</td>
</tr>
<tr>
<td>6.7</td>
<td>UCB solutions with constant rate $\beta_t$ (20 features)</td>
<td>35</td>
</tr>
<tr>
<td>6.8</td>
<td>UCB solutions with logarithmic rate $\beta_t = \beta_0 \log t$ (10 features)</td>
<td>36</td>
</tr>
<tr>
<td>6.9</td>
<td>UCB solutions with logarithmic rate $\beta_t = \beta_0 \log t$ (20 features)</td>
<td>36</td>
</tr>
<tr>
<td>6.10</td>
<td>Thompson sampling solutions (10 features)</td>
<td>37</td>
</tr>
<tr>
<td>6.11</td>
<td>Thompson sampling solutions (20 features)</td>
<td>37</td>
</tr>
<tr>
<td>6.12</td>
<td>Solutions with 5 cluster Gaussian mixture prior for 20 rounds (10 features)</td>
<td>38</td>
</tr>
<tr>
<td>6.13</td>
<td>Single greedy solutions with varying $\alpha$ parameter (10 features)</td>
<td>39</td>
</tr>
</tbody>
</table>
List of Algorithms

5.1 Common structure ........................................... 24
5.2 Single Update of User Distribution ...................... 25
5.3 GMM Update of User Distribution ....................... 25
5.4 Thompson sampling ........................................ 27
Chapter 1

Introduction

This chapter is an introduction to the main topic of the thesis. First, we describe the topic of movie recommendations. Then, we briefly present the prior work in the area. We continue with showing the main challenges for the thesis and the main contributions. Finally, we give outline of how the thesis is structured in the remaining chapters.

1.1 Recommender systems

Recommender systems are information systems aiming at predicting the preference a given user will have towards some item. The main theme of the current thesis is movie recommendations. In this sense a recommender system would be a system that tries to predict the rating a user will give to a particular movie. Although, the presentation and the examples will be primarily about movies, all algorithms and ideas are valid for various other items, like books, songs, news articles, etc.

The established way of evaluating movie recommender systems is by calculating the average difference between predicted and actual ratings. However in an actual system we would be more eager to predict accurately the ratings for the movies a particular user likes [8]. For example, if Alice dislikes Inception we would only be interested to know that fact. The exact rating she would give for that movie is not that important. However, if Alice really likes The Green Mile it would be important for us to know the rating as best as possible, because this would make it easier to decide if we should recommend the movie or not. This is the main aim of the current thesis.
1. Introduction

1.1.1 Collaborative filtering

Collaborative filtering [11] is an approach for recommender systems which aims at inferring relationships between users and similarities between items based on ratings given by some users to some items. The main idea behind the approach is that similar users tend to give similar ratings to similar items.

An important characteristic of collaborative filtering is that it infers those relationships purely based on user ratings. There is no need for any external features. This will be the scenario in which we will be working throughout the thesis. There will be no external information about the movies. There are approaches like content filtering which require external features. In the case of movies such features could be genres, actors, directors, length, etc. Such features could be included in the algorithms presented in the current work, but it is beyond the scope of the thesis.

1.1.2 Matrix Factorization

One of the accepted methods for collaborative filtering is based on low-dimensional latent factor models. The assumption behind these methods is that both users and movies (items) lie in a common low-dimensional vector space. Let us say, for example, that the dimensionality of the space is $D$. Then, a user will be represented as a $D$-dimensional vector $u \in \mathcal{R}^D$, a movie will also be represented as a $D$-dimensional vector $v \in \mathcal{R}^D$, and the predicted rating of user $u$ for movie $v$ will be modeled as the inner product of the two vectors, i.e. $r = u^\top v$. Intuitively, this is similar to how we assign genres to movies. For example, each dimensionality of the movie vector could show how much the movie is of particular genre. For the users each dimensionality will show how much each user like that particular genre.

It turns out that the problem of finding the best $D$-dimensional representation of the users and movies is related to the problem of finding the closest rank-$D$ approximation of the preference matrix $R$. The most popular methods for approaching this problem are based on matrix factorization. The preference matrix $R \in \mathcal{R}^{N \times M}$ is approximated with the factorization $U^\top V$, where $U \in \mathcal{R}^{D \times N}$ and $V \in \mathcal{R}^{D \times M}$. Here $N$ is the number of users and $M$ is the number of movies. The columns $U_i$ of the matrix $U$ are the $D$-dimensional user representations, while the columns $V_j$ of the matrix $V$ are the $D$-dimensional movie representations.

A good overview of how the problem is approached as matrix factorization is given by [14]. The matrix factorization needed for the problem is closely related to the Singular Value Decomposition (SVD) of matrices. Introduction
1.2 Thesis contributions

to latent factor models is given by [12]. [20] and [19] give a probabilistically based solutions to the problem.

1.1.3 Multiarmed bandits

The problem of multiarmed bandits [7] (also known as k-armed bandits) is an old and well studied problem. This is the problem a gambler faces in a casino when given the option to play on multiple slot machines. Each slot machine has fixed unknown probability of winning. The gambler is allowed to play on the machines unbounded amount of times and their aim is to maximize the expected total amount of wins. This classic problem puts the player in the dilemma of trading off between exploitation and exploration. At every round the player is able to either play the most successful machine so far (exploitation) or to play relatively unknown machine (exploration) to learn more about it.

In the case when items come with external features, ideas from the multiarmed bandits problem can be used for recommending items. In [15] it is shown how those ideas could be applied to the problem of recommending news articles. Those approaches are good at trading off between exploitation and exploration. [21] shows how Upper Confidence Bound algorithms can be used when the score function is modeled as Gaussian Process [18]. The intuition behind upper confidence bound algorithms is the so called optimism in face of uncertainty. In simple words this means that when we are uncertain about the result of an action, we are optimistically eager to execute it and learn more about its performance.

The main difference between multiarmed bandits and movie recommendations is that in the case of slot machines we are allowed to play each machine multiple number of times. This means that after some reasonable number of rounds we will know enough for the underlying model as to figure out which is the machine with highest winning probability. After that we can just continue playing only on that machine. However, in a realistic movie recommendation scenario we are not interested in recommending a single movie multiple times to the same user. That’s why the number of rounds we can “play” are limited and if the user is not satisfied in the beginning they would probably leave the recommender system.

1.2 Thesis contributions

The main contribution of our work is that we combine matrix factorization with the upper confidence bound algorithms from the bandit setting. We apply algorithms similar to the ones studied in [15] to the scenario of movie
recommendations when we have no external information about the users and the movies, besides a training set of user, movie and rating tuples.

Another contribution of the thesis is an offline evaluation of movie recommenders based on synthetically generated dataset. In our work we are not interested in the Root Mean Square Error (RMSE). The reason for this is that our aim is not predicting the ratings for user movie pairs missing from the training set. Our aim is to sequentially recommend movies to new users while trying to minimize to accumulated regret in the way recommender systems are evaluated in [21] and [15]. Unfortunately, evaluating a movie recommender that way is unfeasible for any real world dataset. We need the ratings for all movies given by a particular user in the testing dataset. Only in such case we are able to calculate the regret, i.e. the difference between the ratings of the recommended movies and the ratings of the best possible movies for a particular user. To know the best ratings for a particular user, we need to know all ratings. That is why we have devised a way to generate full synthetic dataset from a partial real dataset. In our experiments we used the movielens [2] dataset to produce synthetic dataset on which to evaluate our recommenders.

1.3 Thesis outline

Here is how the thesis will be structured in the following chapters. In Chapters 2 and 3 we will give the mathematical background needed for the rest of the thesis. Chapter 2 gives small introduction for matrix factorization methods and Chapter 3 introduces multiarmed bandits and how they could be used for recommendation. After that in Chapter 4 we introduce the scenario in which we will be working and give the mathematical foundations on which our algorithms are based. Then, in Chapter 5 we present the algorithms, the main contribution of the current thesis. In Chapter 6 we proceed with explanation of how we generated a synthetic dataset and show the results from our experiments. Finally in Chapter 7 we sum up the contributions of the current work and discuss possible future improvements.
Chapter 2

Matrix Factorization

Some of the most successful and widely used methods for movie recommendations are based on matrix factorization. In this chapter we will give an introduction on using matrix factorization towards collaborative filtering. First we will define the singular value decomposition (SVD) and show how it is useful for collaborative filtering. Then we will address the drawbacks of pure SVD approaches and introduce probabilistically based factorization methods.

2.1 Singular Value Decomposition

2.1.1 Definition

Singular value decomposition (SVD) is a widespread method for matrix factorization. Every rectangular real, or complex, matrix has singular value decomposition. For example, let $A \in \mathcal{R}^{M \times N}$ be a real matrix with $M$ rows and $N$ columns. Then, the singular value decomposition of $A$ is

$$A = UDV^T$$

where $U \in \mathcal{R}^{M \times M}$ is an $M$ by $M$ orthogonal matrix ($U^T U = UU^T = I_{(M)}$), $D \in \mathcal{R}^{M \times N}$ is a diagonal matrix and $V^T \in \mathcal{R}^{N \times N}$ is an $N$ by $N$ orthogonal matrix ($V^T V = VV^T = I_{(N)}$). The matrix $D$ is diagonal and $d_{ii} \neq 0$ for $1 \leq i \leq r$, where $r$ is the rank of the matrix $A$. The values $d_{ii}$ (or $d_i$ for shorter) are called the singular values of the matrix $A$. It is accepted that the factorization is written in such a way so that

$$d_1 \geq d_2 \geq \cdots \geq d_r \geq d_{r+1} = 0$$
2. Matrix Factorization

The first $r$ columns of $U$ are called the left singular vectors of $A$ and they form orthogonal basis of the space spanned by the columns of $A$. Respectively, the first $r$ rows of $V^\top$ are called the right singular vectors of $A$ and form orthogonal basis of the space spanned by the rows of $A$.

The algorithms for computing the singular value decomposition are well studied, but the topic of the concrete implementations does not intersect with the scope of the current work. It boils down to finding the eigenvalues of $A^\top A$. We advice the interested reader to check [13] for more elaborate introduction on SVD.

2.1.2 Closest approximation

Before looking at how SVD is related to movie recommendations let us note an interesting property of the decomposition. Let the singular value decomposition of the matrix $A \in \mathbb{R}^{M \times N}$ be $UDV^\top$ and $k < r$, where $r$ is the rank of $A$.

Now, let $A_k$ be the matrix $A$ after removing all but the largest $k$ singular values. In other words

$$A_k = \sum_{i=1}^{k} d_i u_i v_i^\top$$

It turns out that $A_k$ is the closest rank-$k$ approximation of $A$ with respect to Euclidean matrix distance, i.e.

$$A_k = \arg\min_{\text{rank}(B)=k} \|A - B\|_2$$

Note that a matrix norm is derived from a vector norm in the following way

$$\|M\|_p = \max_{x \neq 0} \frac{\|Mx\|_p}{\|x\|_p}$$

2.1.3 Application in movie recommendations

Now, let’s see how the singular value decomposition can be used for collaborative filtering and movie recommendations, concretely. As we already mentioned collaborative filtering are methods which try to infer relationships between users and items based on ratings from some users to some items.

First, let’s introduce the preference matrix $R$. $R$ is matrix with $N$ rows and $M$ columns. Each row represents a user and each column represent an item (movie). $R_{ij}$ is the rating of user $i$ for movie $j$. Obviously, in any real world
2.1. Singular Value Decomposition

scenario this matrix is sparse, in the sense that, there are many missing entries. Usually a user rates only a small subset of all the movies. Thus, $R$ is not even a matrix in the mathematical sense of the word. Let’s define $A$ as a completion of $R$. For all $i$ and $j$ for which there is an entry $R_{ij}$, we have that $A_{ij} = R_{ij}$. And, for all $i$ and $j$ for which the entry $R_{ij}$ is missing $A$ contains an imputed value. There are various ways for imputing values for missing entries. For example, one way would be to use the average rating of the movie or the average rating for the user.

If we take the singular value decomposition $A = UDV^T$ we could interpret the $r$ singular values as different concepts for the items. Intuitively, those concepts are similar to genres. However, there is no mapping between genres and those automatically inferred concepts. The concepts are called hidden latent factors. The singular values $d_i$ represent the expressiveness of each concept. The matrices $U$ and $V$ represent the relationships between user and concepts, and movies and concepts, respectively. For example, $U_{ij}$ shows how much user $i$ likes concept (hidden latent factor) $j$. Similarly, $V_{ij}$ shows how much movie $i$ is related to concept $j$. Again, only speaking intuitively, let’s say that the first concept is the concept of comedy. Then, the first column of $U$ shows how much each user likes comedy, and the first column of $V$ show how much each movie is related to comedy. The singular value $d_1$ shows how much comedy is important about the final rating.

2.1.4 Drawbacks

There are two main drawbacks for using pure SVD approaches for collaborative filtering. The first is that finding the eigenvalues of $A^\top A$ is not scalable for large datasets. There are, however, gradient descent algorithms which are scalable and provide a good approximation. The second drawback is that SVD is matrix factorization and it works only for full matrices, but not partial ones. However, as we already mentioned, in real world situations we have only a small fraction of all the entries of the preference matrix $A$, and our goal is to predict the remaining entries. A way to tackle this disadvantage is to impute the missing entries before the decomposition. For example, we could calculate the average rating and use this value for all the missing entries. That way we would have complete matrix on which we could perform singular value decomposition. However, since the original matrix is sparse, most of the entries will be imputed and this is not very desirable. In the next section we will see how to overcome this problem through a probabilistic view of the problem.
2.2 Probabilistic approaches

Probabilistic approaches towards latent factor models are studied in [16], [20] and [19].

2.2.1 Probabilistic Matrix Factorization

We will first introduce probabilistic matrix factorization as proposed in [20]. Let’s say we have $N$ users and $M$ movies, and that $R$ is the preference matrix, i.e. $R_{ij}$ is the rating given by user $i$ to movie $j$. Since, $R$ is a partial matrix, we also introduce an indicator matrix $I$, where

$$I_{ij} = \begin{cases} 1 & \text{if user } i \text{ has rated movie } j \\ 0 & \text{otherwise} \end{cases}$$

The aim of the model is to find user and movie latent feature matrices. In other words we want to find matrices $U \in \mathbb{R}^{D \times N}$ and $V \in \mathbb{R}^{D \times M}$, where the columns $U_i$ of $U$ are the $D$-dimensional user features and the columns $V_j$ of $V$ are the features of the movies. The model is a linear model with independent Gaussian noise. The rating from user $i$ for movie $j$ is modeled as inner product between $U_i$ and $V_j$, plus independent Gaussian noise. The likelihood of the preference matrix given the user and movie feature vectors is

$$p(R | U, V, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} \mathcal{N}(R_{ij} | U_i^T V_j, \sigma^2) I_{ij}$$

In the above distribution, $\mathcal{N}(x | \mu, \sigma^2)$ denotes the density function of a Gaussian distribution with mean $\mu$ and variance $\sigma^2$. $\sigma$ is the noise deviation, i.e. the rating $R_{ij}$ is modeled as $R_{ij} = U_i^T V_j + \epsilon$, where $p(\epsilon) = \mathcal{N}(\epsilon | 0, \sigma^2)$. We assume a spherical 0-mean Gaussian priors for the user and movie features,

$$p(U | \sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i | 0, \sigma_U^2 I)$$

$$p(V | \sigma_V^2) = \prod_{j=1}^{M} \mathcal{N}(V_j | 0, \sigma_V^2 I)$$
Now, we can write the logarithm of the posterior distribution

\[
\ln p(U, V | R, \sigma^2, \sigma^2_U, \sigma^2_V) = -\frac{1}{2\sigma^2} \sum_{i=1}^N \sum_{j=1}^M I_{ij}(R_{ij} - U_i^\top V_j)^2 - \frac{1}{2\sigma^2_U} \sum_{i=1}^N U_i^\top U_i - \frac{1}{2\sigma^2_V} \sum_{j=1}^M V_j^\top V_j - \frac{1}{2} \left( \sum_{i=1}^N \sum_{j=1}^M I_{ij} \ln \sigma^2 + ND \ln \sigma^2_U + MD \ln \sigma^2_V \right) + C
\]

where C is a constant independent of the parameters. Maximizing this likelihood is equivalent to minimizing the following function

\[
E = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^M I_{ij}(R_{ij} - U_i^\top V_j)^2 + \frac{\lambda_U}{2} \sum_{i=1}^N \|U_i\|^2 + \frac{\lambda_V}{2} \sum_{j=1}^M \|V_j\|^2
\]

A simple gradient descent on U and V can be used to find a local minimum of that function.

The main drawback of the presented method is that we have to manually choose the parameters \(\lambda_U\) and \(\lambda_V\). These are regularization parameters which trade model complexity for model’s fit to the data. If we allow the model to be arbitrarily complex, we will be able to find an almost perfect match for the training data. This, however, means that we would have learned the noise as part of the model. This phenomena is called overfitting. Usually models prone to overfitting are known to be able to match the training data very well, but perform badly on new, unseen, data.

### 2.2.2 Bayesian approach

Now we will introduce the approach proposed by [19]. It will be initial step for our solutions and also takes part in the generation of our synthetic dataset. The approach from Section 2.2.1 only cared about finding the matrices U and V which maximized the likelihood. The current approach aims at estimating the distributions of U and V given the observed ratings. It is a fully bayesian extension of the probabilistic matrix factorization model. The distribution of R given U and V is the same. Again, the assumed prior distributions for the user and movie latent vectors are Gaussians. However, this time their parameters are not fixed. We call the parameters of these distributions hyperparameters. \(\Theta_U = \{\mu_U, \Lambda_U\}\) and \(\Theta_V = \{\mu_V, \Lambda_V\}\) are the hyperparameters for the users and movies, respectively. The prior distribu-
2. **Matrix Factorization**

Factorizations are

\[
p(U | \mu_U, \Lambda_U) = \prod_{i=1}^{N} \mathcal{N}(U_i | \mu_U, \Lambda_U^{-1})
\]

\[
p(V | \mu_V, \Lambda_V) = \prod_{j=1}^{M} \mathcal{N}(V_j | \mu_V, \Lambda_V^{-1})
\]

Further, there are priors for the hyperparameters as well

\[
p(\Theta_U | \Theta_0) = p(\Lambda_U) p(\mu_U | \Lambda_U)
\]

\[
= \mathcal{W}(\Lambda_U | \mathbf{W}_0, \nu_0) \mathcal{N}(\mu_U | \mu_0, (\beta_0 \Lambda_U)^{-1})
\]

\[
p(\Theta_V | \Theta_0) = p(\Lambda_V) p(\mu_V | \Lambda_V)
\]

\[
= \mathcal{W}(\Lambda_V | \mathbf{W}_0, \nu_0) \mathcal{N}(\mu_V | \mu_0, (\beta_0 \Lambda_V)^{-1})
\]

Here, \( \mathcal{W} \) is the Wishart distribution [23] with \( \nu_0 \) degrees of freedom and scale matrix \( \mathbf{W}_0 \).

To predict the rating of an unseen user-movie pair \( R_{ij}^* \) we need to solve the following integral

\[
p(R_{ij}^* | R, \theta_0) = \int \int p(R_{ij}^* | U_i, V_j) p(U, V | R, \Theta_U, \Theta_V)
\]

\[
p(\Theta_U, \Theta_V | \Theta_0) d\{U, V\} d\{\Theta_U, \Theta_V\}
\]

Exactly solving this integral is infeasible, and that is why [19] propose a *Markov Chain Monte Carlo* [5] approximation method. For the exact algorithm we advise the reader to look at [19]. It is using *Gibbs sampling*. We later use this method as an initial step in our algorithms. The exact *matlab* implementation that we used can be found in [1].
Chapter 3

Multiarmed bandits

In this chapter we will introduce the problem of multiarmed bandits, also known as \( k \)-armed bandits. Then, we will show a few approaches to the problem and the concept of exploration-exploitation trade-off. After that, we will introduce the contextual bandits and their application for personalized recommendations.

3.1 Problem statement

Suppose we are a gambler faced with the problem of playing on a slot machine with \( k \) arms. Pulling the \( i \)-th arm we have probability of winning \( \mu_i \). These \( k \) probabilities are unknown to us and our aim is to find a strategy of pulling arms as to maximize the number of wins after many (possibly infinite) number of rounds.

Let’s formalize the problem and the way we evaluate our actions. As we already mentioned the \( i \)-th arm has winning probability of \( \mu_i \). This means that for the best possible decision the expected win will be \( \mu^* = \max_i \mu_i \). Let \( i_1, \ldots, i_T \) be the decisions that we have made, i.e. in round \( t \) we have pulled the \( i_t \)-th arm. We call the regret at round \( t \) the value \( r_t = \mu^* - \mu_{i_t} \). The total regret is

\[
R_T = \sum_{t=1}^T r_t
\]

Typically, our aim is to find strategy that guarantees sub linear total regret. In other words, as \( T \) tends to infinity we want \( R_T / T \to 0 \).
3. Multiarmed bandits

3.2 Simple strategies

In the simplest possible scenario we have perfect information for all probabilities and thus we can always play the optimal arm. At the other extreme, we may be interested only in learning all arms as much as we can without caring about the regret. In such case we would pull all arms equally many times. If we have pulled each arm $m$ times, then for the $i$-th arm we will have the following estimate for the winning probability of that arm

$$\hat{\mu}_i = \frac{1}{m} \sum_{l=1}^{m} y_{i,l}$$

The problem with picking the best strategy is that we have to trade off between exploitation (pulling the best arm so far) and exploration (making suboptimal choices in hope to learn more about the arms’ pay-offs).

3.2.1 $\epsilon$-greedy

A simple way of explicitly choosing between exploration and exploitation, is to fix time dependent constant $\epsilon_t$ for each round $t$ and then explore with probability $1 - \epsilon_t$. In other words, during round $t$ with probability $\epsilon_t$ we choose the best arm so far, and with probability $1 - \epsilon_t$ we choose an arm uniformly at random. [6] shows that for suitable choice of $\epsilon_t$ it holds that $R_T = O(k \log T)$.

3.2.2 UCB1

[6] also studies a more elegant way to implicitly choose between exploration and exploitation. The strategy is based on the idea of optimism in face of uncertainty. If an arm is pulled just a few times and we are uncertain of its pay-off, we might optimistically pull it. However, if after some number of pulls it consistently performs poor, then we are not willing to pull it anymore. This strategy can be implemented without explicitly differentiating between explore and exploit steps. All steps are exploitation steps, but instead of using the mean pay-off of each arm, for choosing the best arm we use an upper confidence bound of the posterior distribution for the pay-off.

3.3 Contextual Bandits

Now, we will see how we can use the problem of multiarmed bandits for news article recommendation as proposed by [15]. First, let’s see the problem scenario. At round $t$, a user comes to the system with feature vector
3.3. Contextual Bandits

This feature vector could come from any external information we might have about the user, such as demographic. Then, we are given a set of $m$ articles which we can recommend to the user. The $i$-th article has feature vector $x_i$. Again, how we get this vector is not central to the current discussion. Given the set of the available articles, the recommender system picks an item $i_t$ to recommend to the user. After that, we receive the reward for our action, which we model as a function depending on the recommended article and the user features plus independent Gaussian distributed noise, i.e.

$$y_t = f(i_t, x_{i_t}, z_t) + \epsilon_t$$

To model this problem as a bandit problem, we will assume some regularity for the reward function $f$. For example, let’s say that the reward function for article $i$ depends linearly on the user features

$$y_t = w_i^T z_t + \epsilon_t$$

Such model treats all articles (items) independently. Each article has associated independent vector which inner product with the context (user feature vector) produces prediction for the reward of the action of recommending that article to the user. That’s why this setup is called contextual bandits. The reward of pulling the $i$-th arm depends on the current context (user).

The aim while learning the model is to find the $w_i$ coefficients for each article. Let’s say that article $i$ has been recommended $m$ times. Then, we want to find the coefficients which minimize the following square loss

$$\hat{w}_i = \arg\min_w \sum_{t=1}^{m} (y_t - w^T z_t)^2$$

Finding the vector which minimizes the loss is a matter of simple linear regression. After that, when a new user (context) comes with feature vector $z_{t+1}$, the value $\hat{w}^T z_{t+1}$ is the prediction of the reward for showing the article.

In order to apply solutions with ideas similar to UCB1, we need to be able to calculate a confidence interval around the prediction $\hat{w}^T z$. The resulting algorithm is called LinUCB. We suggest the interested reader to refer to [15] for detailed description of the algorithm. The exact details are not related to the approach of the thesis. At the current point we are interested only to show how contextual bandits can be used for recommendation.

A drawback of the LinUCB algorithm is that it treats each article (arm) independently. When we are presented with new article we are faced with the so called cold start as we know nothing about it. If we have external features for the articles as well, [15] provides a slightly different solution called HybridUCB which does not treat each article independently.
In our scenario we have neither user nor movie features. In the next chapter we will show how in combination with matrix factorization methods we can use bandit-like methods for movie recommendations.
Mathematical foundations

In Chapter 2 and Chapter 3 we saw how to use matrix factorization and contextual bandits for recommendations. In this chapter we present the mathematical foundations on which the main contributions of the thesis are based. The topics studied here are the basis of the algorithms introduced in Chapter 5.

4.1 Setup

Let’s start with describing the setup in which we will be working. There is a fixed set of movies which are available for recommendation. There is a number of training users for which we know ratings for some movies. After a learning phase we have sequential decision problem for each new coming user. Let’s imagine a user comes to the system. We don’t know anything about that user and we don’t have any external features. To start we assume a prior distribution from which users are sampled. This distribution could be “guessed” based on the training users. We start sequentially recommending movies to them. At round $t$ we recommend movie $j_t$ and the reward we receive is the rating the user gives to the movie. Then, we refine our model, i.e. the distribution from which we assume users are drawn. With the refined model we are ready to recommend a new movie.

The aim is to maximize the total sum of rewards (ratings). We will be calculating the regret after $T$ rounds, i.e. after recommending $T$ movies. If at round $t$ we recommended movie $j_t$ with rating $y_t$ and $y^*_t$ is the rating for $j^*_t$, the $t$-th best movie for the given user, then the regret is

$$R_T = \sum_{t=1}^{T}(y^*_t - y_t)$$
4. **Mathematical foundations**

Here comes the first big problem in evaluating such system. In order to calculate the regret we need to know which are the best $T$ movies for the test user together with their ratings. However, this is infeasible for any real world dataset. We have overcome this problem by using synthetic dataset. We will see how in Chapter 6.

4.2 **Rating prediction model**

In order to use bandit like approach towards movie recommendations we need to have a prediction model for the ratings the users would give to the movies. We assume a latent factor model as in the matrix factorization methods. Users and movies are assumed to lie in a low-dimensional vector space. Each user and movie is represented with a latent factor vector. Those vectors’ inner product plus independent noise gives the rating the user will give to the movie. If a user has feature vector $u$ and a movie has feature vector $v$, then the rating $y$ will be given with

$$y = u^\top v + \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian distributed noise.

4.3 **Latent factor vectors**

In order to use the model from Section 4.2 we need to have some features. In our experiments we start with a partial preference matrix $R$ just like in matrix factorization. $R \in \mathcal{R}^{N \times M}$, meaning that we have $N$ training users and $M$ movies. These $M$ movies form a fixed set that will not change over time. These will be the movies available for recommendation later.

The first step of our approach is to take the preference matrix $R$ and implement matrix factorization as in [19]. We use the Gibbs sampling [10] implementation presented in [1, 19]. After the factorization we have $M$ movie feature vectors and $N$ user feature vectors. The $i$-th user is represented with $D$-dimensional real vector $u_i$ and the $j$-th movie is represented with $v_j \in \mathcal{R}^D$. The $u_i$ vectors are only training vectors which we will not use directly. The $v_j$ vectors however will be used directly during the recommendation stage. From now on, we will assume that the features of the $j$-th movie are $v_j$.

When a new user comes to the system, initially we will know absolutely nothing about them. That’s why, we are going to use these $u_i$ vectors to build a prior assumption for the distribution from which users are drawn. For example, we could fit a Gaussian distribution $\mathcal{N}(\mu_U, \Sigma_U)$ to the training data $u_i$. Then, when a new user comes we will initially assume it is sample from the distribution $\mathcal{N}(\mu_U, \Sigma_U)$. 

16
4.4 Gaussian prior

We will examine two options for the prior user distribution. The first option is to assume that users are sampled from a multivariate Gaussian distribution.

4.4.1 Recommendation

Let’s say, that a new user with feature vector $u$ comes. The vector $u$ is unknown to us. In order to make recommendations we need to make predictions for the ratings that the user will give to the movies. Let $v_j$ be the feature vector of the $j$-th movie, $v_j \in \mathbb{R}^D$. As we said we don’t know $u$, but we have prior distribution from which we assumed it is sampled, $u \sim \mathcal{N}(\mu_U, \Sigma_U)$. In this case the inner product of the two vectors is univariate Gaussian distribution $u^\top v_j \sim \mathcal{N}(\mu^\top_U v_j, v_j^\top \Sigma_U v_j)$.

Recall, that in our rating prediction model, the rating $y_j$ is a sum of the inner product and independent Gaussian noise, i.e.

$$ y_j = u^\top v_j + \epsilon_j $$

$$ y_j \sim \mathcal{N}(\mu^\top_U v_j, v_j^\top \Sigma_U v_j + \sigma^2) $$

where $\epsilon_j \sim \mathcal{N}(0, \sigma^2)$.

This means that for each movie $j$ we have a predictive univariate Gaussian distribution for its rating $y_j \sim \mathcal{N}(\hat{\mu}_j, \hat{\sigma}_j^2)$.

Now, if we want to perform a completely exploitation step, we have to pick the movie with the highest expected rating, i.e.

$$ j^\star = \arg \max_j \hat{\mu}_j $$

If we want to use algorithm similar to UCB1, we can hide the distinction between exploitation and exploration by using an upper confidence bound of the rating distribution. We can pick the movie in the following way

$$ j^\star = \arg \max_j \hat{\mu}_j + \sqrt{\beta_t \hat{\sigma}_j} $$

where $\beta_t$ is time dependent constant. Recall, that we are in the scenario of sequential recommendation and $t$ shows the number of the current round. $\beta_t$ will be parameter for the UCB algorithms which we have to manually tune. It controls the balance between exploitation and exploration in each round. If $\beta_t$ is 0 we have completely exploitation step. The greater the value of $\beta_t$, the higher the probability that we will explore relatively unknown option.
4. Mathematical foundations

4.4.2 Update rule

Now, when we have recommended a movie with feature vector \( v \) to the user with prior distribution \( u \sim \mathcal{N}(\mu_U, \Sigma_U) \), we have received a rating \( y \) and we need to modify the user distribution, i.e. estimate the posterior distribution, \( P(u|y) \):

\[
P(u|y) \propto P(u, y) = P(u)P(y|u) \\
= \mathcal{N}(u|\mu_U, \Lambda_U^{-1})\mathcal{N}(y|u^\top v, \alpha^{-1}) \\
= \mathcal{N}(u|\mu_U^*, [\Lambda_U^*]^{-1})
\]

where

\[
\Lambda_U^* = \Lambda_U + \alpha vv^\top \\
\mu_U^* = [\Lambda_U^*]^{-1}(\Lambda_U \mu_U + \alpha yv)
\]

Note that \( \Lambda_U \) is the precision matrix of the prior Gaussian distribution and is the inverse of the covariance matrix, \( \Lambda_U^{-1} = \Sigma_U \). We are using the precision matrix instead of the covariance, because that way it is easier to express the posterior distribution. In the same way, \( \alpha \) is the precision of the Gaussian noise. It is the reciprocal of the variance, i.e. \( \alpha^{-1} = \sigma^2 \).

The posterior user distribution is again multivariate Gaussian, which we can use as prior distribution for the next recommendation. This update rule is very similar to the update rule in [19]. The prior distribution for the next round would be

\[
u \sim \mathcal{N}(\mu_U^*, [\Lambda_U^*]^{-1})
\]

4.5 Gaussian mixture prior

The second option for prior user distribution is Gaussian Mixture Model (GMM) [17]. A Gaussian mixture is a weighted sum of \( k \) Gaussian distributions. In Section 4.4 we assumed that users are drawn from Gaussian distribution, i.e.

\[
P(u) = \mathcal{N}(u|\mu_U, \Sigma_U)
\]

Now we will assume that a user is drawn from one of \( k \) Gaussian distributions, i.e.

\[
P(u) = \sum_{i=1}^{k} w_i \mathcal{N}(u|\mu_i, \Sigma_i)
\]

The weights \( w_i \) show how likely it is for a user to be drawn from a particular cluster (Gaussian distribution with mean \( \mu_i \) and covariance matrix \( \Sigma_i \)).
4.5. Gaussian mixture prior

Intuitively, this is a more natural assumption, because there are different types of users. For example, some users like comedies, while others prefer action or romantic movies. That’s why we decided to try with a prior distribution which models that intuition.

4.5.1 Recommendation

A new user comes to the system. We assume that its feature vector is drawn from a Gaussian mixture, i.e.

\[ P(u) = \sum_{i=1}^{k} w_i N(u|\mu_i, \Sigma_i) \]

where

\[ \sum_{i=1}^{k} w_i = 1 \]

and all \( w_i \)s are nonnegative. The \( w_i \) values are the cluster weights and they show how likely is each cluster. For example, \( w_i \) shows the probability that a new user will be drawn for the \( i \)-th cluster (Gaussian distribution). Now, let’s introduce a hidden random variable \( z \). \( z \) has multinomial distribution, such that \( P(z = i) = w_i \). This allows us to rewrite \( P(u) \) thusly

\[ P(u) = \sum_{z} P(z)P(u|z) = \sum_{z} P(z)N(u|\mu_z, \Sigma_z) \]

Now, let’s see the distribution of the inner product \( u^\top v \), where \( v \) is the feature vector of the movie which rating we would like to predict.

\[ P(u^\top v) = \sum_{z} P(z)P(u^\top v|z) = \sum_{z} w_z N(u^\top v|\mu_z^\top z, v^\top \Sigma_z v) \]

Notice that the inner product is a mixture of univariate Gaussian distributions.

Mixture of univariate Gaussians

Now, we will see how to work with mixture of univariate Gaussians for predicting the ratings. Let \( X \) be an univariate mixture with the following distribution

\[ P(x) = \sum_{i=1}^{k} w_i N(x|\mu_i, \sigma_i^2) \]

Now, let \( z \) be the latent random variable, such that \( P(z = j) = w_j \). Let’s rewrite the distribution

\[ P(x) = \sum_{z} P(z)P(x|z) \]
4. Mathematical foundations

We are ready to calculate the mean $\mu$,

$$\mu = \mathbb{E}_X \{ x \} = \sum_z w_z \mu_z$$

This is true because of the linearity property of the expectation. The variance of the distribution is the following,

$$\text{Var}_X \{ x \} = \mathbb{E}_X \{ (x - \mu)^2 \} = \sum_{i=1}^k w_i (\sigma_i^2 + \mu_i^2 - \mu^2)$$

Now, when we have to pick a movie to recommend, we can use the mean $\mu$ for greedy, completely exploitation step. Alternatively we can use expression of the form $\mu + \sqrt{\beta \sigma}$, like in Section 4.4.1. Note that here $\sigma$ is the square root of the variance that we just calculated. However, for fixed value of $\beta$ this expression evaluates different quantiles for different mixtures, i.e. different confidence bounds. This is not the case with Gaussian distributions where for fixed value of $\beta$ the expression $\mu + \sqrt{\beta \sigma}$ evaluates the same quantile.

In our initial implementation we used the expression $\mu + \sqrt{\beta \sigma}$, but later we also used a version which calculated fixed quantiles of the mixture distribution.

So, let’s see how to calculate the $q$-th quantile of the mixture distribution. The $q$-th quantile is the value $x$ for which $P(X \leq x) = q = \text{cdf}(x)$. $\text{cdf}$ is the cumulative distribution function. For distribution $X$ with probability density function $p(x)$ the cumulative distribution function is calculated in the following way

$$\text{cdf}(x) = \int_{-\infty}^{x} p(x) \, dx$$

In the case of mixture the $\text{cdf}$ is calculated the following way

$$\text{cdf}(x) = \int_{-\infty}^{x} \sum_{i=1}^k w_i \mathcal{N}(x | \mu_i, \sigma_i^2) \, dx$$

$$= \sum_{i=1}^k w_i \int_{-\infty}^{x} \mathcal{N}(x | \mu_i, \sigma_i^2) \, dx$$

$$= \sum_{i=1}^k w_i \text{cdf}_i(x)$$

where $\text{cdf}_i$ is the $\text{cdf}$ of the $i$-th cluster, $\mathcal{N}(\mu_i, \sigma_i^2)$.

To find the $q$-th quantile of the mixture we have to solve the equation $\text{cdf}(x) = q$ for $x$. Since, the $\text{cdf}$ function is monotone, we used simple Newton algorithm for finding the root.
4.5. Gaussian mixture prior

4.5.2 Update rule

Now, let’s see how we update the user distribution after we have been fed the true rating of the movie we recommended. First, let’s see how to recalculate the cluster weights, \( P(z|y) \), where \( z \) is the latent random variable such that \( P(z = i) = w_i \):

\[
P(z|y) \propto P(z, y) = P(z)P(y|z) = w_zP(y|z)
\]

Recall from our rating model that \( y = u^\top v + \epsilon \), where \( \epsilon \) is independent Gaussian noise, \( \epsilon \sim \mathcal{N}(0, \sigma^2) \). The following steps show how to calculate the distribution \( y|z \).

\[
\begin{align*}
    u|z & \sim \mathcal{N}(u|\mu_z, \Sigma_z) \\
    u^\top v & \sim \mathcal{N}(u^\top v|\mu_z^\top v, v^\top \Sigma_z v) \\
    y|z & \sim \mathcal{N}(y|\mu_z^\top v, v^\top \Sigma_z v + \sigma^2)
\end{align*}
\]

This means that

\[
P(z|y) \propto w_z\mathcal{N}(y|\mu_z^\top v, v^\top \Sigma_z v + \sigma^2)
\]

Note that the number of clusters is finite (\( k \), more precisely). So we have to calculate the expression \( w_z\mathcal{N}(y|\mu_z^\top v, v^\top \Sigma_z v + \sigma^2) \) for each cluster \( z \) and then normalize to get the new cluster weights. Basically, a cluster weight is modified by multiplying the old weight with the likelihood of the received rating if the user was drawn from that cluster.

Now, that we have calculated the cluster weights, we can go forward and recalculate the clusters themselves. Recall that the clusters are just multivariate Gaussian distributions:

\[
P(u|z, y) \propto P(u|z)P(y|u)
\]

\[
P(u|z, y) \propto \mathcal{N}(u|\mu_z, \Sigma_z)\mathcal{N}(y|u^\top v, \sigma^2)
\]

This is true, because \( y \) is independent of \( z \) given \( u \). This is exactly the same update rule as the one for single Gaussian prior from Section 4.4.2. In other words, we modify each Gaussian cluster the same way as if it was a single Gaussian distribution.
Chapter 5

Algorithms

Having all the required technical background, we are ready to present the recommendation algorithms that we evaluated in the current thesis. All presented algorithms have a common structure shown in Algorithm 5.1. The two things that differentiate the different solutions are the implementations of \texttt{CALCSCORE} and \texttt{UPDATEUSERDIST}. The first method returns the score for each movie given the movie latent vector and the user distribution. And the second method updates the user distribution given the received rating of the recommended movie. The \texttt{CALCSCORE} methods are implemented based on the mathematical derivations from Section 4.4.1 and Section 4.5.1. And the \texttt{UPDATEUSERDIST} methods follow the derivations from Section 4.4.2 and Section 4.5.2.

Important characteristic of the common algorithm infrastructure shown in Algorithm 5.1 is that unlike in the multiarmed bandit problem, here we do not recommend a movie more than once. In the \texttt{k} armed bandit problem we were able to pull each arm multiple times. Here we only play each movie at most once.

As we see from the pseudo code, at each round \(t\) we recommend the movie with highest score given the current assumption (knowledge) of the user distribution, \texttt{CALCSCORE}(\texttt{u, v, t}). After that we receive the real rating for that movie, \(y^*\), and update the user distribution accordingly.

5.1 Naive algorithm

This is a very simple algorithm presented only for initial benchmark. There is no update of the user distribution between the consecutive trials. This means that the method \texttt{UPDATEUSERDIST} is empty. The score of the movie
5. **Algorithms**

**Algorithm 5.1 Common structure**

Load user prior distribution \( u \)
Load movie feature vectors \( v_j \)
Load noise deviation \( \sigma \)
Load number of trials \( T \)

\( \text{usedMovies} \leftarrow \{ \} \)

**for** \( t = 1 \ldots T \) **do**

**for** all movies \( j \) **do**

\( \text{if } j \notin \text{usedMovies} \text{ then} \)

\( \text{score}_j = \text{CALCSCORE}(u, v_j, t) \)

**else**

\( \text{score}_j = -\infty \)

**end if**

**end for**

\( j^* = \arg \max_j \text{score}_j \)

\( \text{usedMovies} = \text{usedMovies} \cup \{ j^* \} \)

Recommend movie \( j^* \) and fetch rating \( y^* \)

\( u \leftarrow \text{UPDATEUSERDIST}(u, v_j, y^*) \)

**end for**

\( j \) with feature vector \( v_j \) is the inner product between the mean of the user distribution and the movie vector, \( \mu^\top_U v_j \).

We saw how to calculate the mean of the user distributions in Section 4.4.1 and Section 4.5.1 for Gaussian and Gaussian mixture priors, respectively.

In simpler words, the naive algorithm takes the mean of the training users’ feature vectors and returns the best \( T \) movies for that fictional user. In multiarmed bandits terminology, at each round the naive algorithm makes exploitation step. However, it does not learn between the rounds.

5.2 **Greedy algorithm**

In our second solution we do not change the \text{CALCSCORE} method and again it returns the inner product \( \mu^\top_U v_j \):

However, now the \text{UPDATEUSERDIST} method is not empty anymore. The method is different for the two assumed prior distributions, Gaussian and Gaussian mixture. Both implementations precisely follow the derivations from Section 4.4.2 and Section 4.5.2, respectively. The implementations of \text{UPDATEUSERDIST} are unmodified in all algorithms that follow. Both the UCB and the Thompson sampling solutions update the user distribution
the same way. They only differentiate in the way they pick the next movie to recommend.

**Algorithm 5.2** Single Update of User Distribution

```plaintext
function SINGLE_UPDATE_USER_DIST(u, v, y) -> u is Gaussian Distribution
    \( \Lambda_U = u, \Sigma^{-1} \)
    \( \alpha = \frac{1}{\sigma^2} \)  # noise precision
    \( \Lambda_U^* \leftarrow \Lambda_U + \alpha vv^T \)
    \( \mu_U^* \leftarrow [\Lambda_U^*]^{-1}(\Lambda_U * u, \mu + \alpha yv) \)
    return new GaussianDistribution(\( \mu_U^*, [\Lambda_U^*]^{-1} \))
end function
```

Algorithm 5.2 shows the implementation of the UPDATE_USER_DIST function for the case of single Gaussian prior and Algorithm 5.3 show the implementation of the function for the case of Gaussian mixture prior. Note that, the parameters of the two functions differ although they are visually the same. In the case of single Gaussian prior, the parameter \( u \) represents a Gaussian distribution. However, in the case of Gaussian mixture the \( u \) parameter is a mixture of Gaussian distributions. The implementation of SINGLE_UPDATE_USER_DIST(\( u, v, y \)) follows the derivations from Section 4.4.2, while the implementation of GMM_UPDATE_USER_DIST(\( u, v, y \)) follows the derivations from Section 4.5.2.

**Algorithm 5.3** GMM Update of User Distribution

```plaintext
function GMM_UPDATE_USER_DIST(u, v, y) -> u is GMM distribution
    for all clusters \( i \in u \) do
        \( \hat{w}_i \leftarrow u, \hat{w}_i * N(y | \mu^*_i, v^T \Sigma^*_i + \sigma^2) \)
    end for
    \( S \leftarrow \sum_{i=1}^{k} \hat{w}_i \)
    \( \hat{u} \leftarrow \text{new GMM}(k) \)  # \( k \) is the number of clusters
    for all clusters \( i \in u \) do
        \( \hat{u}, \hat{w}_i \leftarrow \frac{\hat{w}_i}{S} \)
    end for
    for all clusters \( i \in u \) do
        \( \hat{u}, \text{dist}_i \leftarrow SINGLE_UPDATE_USER_DIST(u, \text{dist}_i, v, y) \)
    end for
    return \( \hat{u} \)
end function
```
5. Algorithms

5.3 UCB algorithm

As we already said, the UpdateUserDist method is the same as the one for the greedy solution. Thus, it only remains to explain the CalcScore one. When the prior is a single Gaussian, we first calculate the mean, $\mu$, and the deviation, $\sigma$, of the rating distribution as we showed in Section 4.4.1. Then we return $\mu + \sqrt{\beta_t}\sigma$, where $\beta_t$ is parameter which depends on the trial number $t$. In the next chapter, we will see a few ways to choose that parameter.

When the prior is mixture of Gaussian distributions we have two options. The first option is again to calculate the expression $\mu + \sqrt{\beta_t}\sigma$. In Section 4.5.1 we saw how to calculate the mean and the deviation of the predictive rating distribution. The second option is to directly compute the $q_t$-th quantile of the predictive rating distribution. We, also, saw how to do this in Section 4.5.1.

5.4 Thompson sampling algorithm

One drawback of the UCB solution is that we have to manually fine tune the $\beta_t$ (or $q_t$) parameters. Thompson sampling [22] is a simple and elegant way to bypass that fine tuning. The main idea is that given the prior user distribution we sample a random user from it and calculate the inner product with each movie vector. Then we pick the movie with the highest inner product. After that, update the user distribution the same way as in the greedy and UCB solutions. The implementation is shown in Algorithm 5.4.
Algorithm 5.4 Thompson sampling

Load user prior distribution $\mathbf{u}$
Load movie feature vectors $\mathbf{v}_j$
Load noise deviation $\sigma$
Load number of trials $T$

$usedMovies \leftarrow \{\}$

for $t = 1 \ldots T$ do

$\hat{\mathbf{u}} \leftarrow \text{RandomSample}(\mathbf{u})$

for all movies $j$ do

if $j \notin usedMovies$ then

$score_j = \hat{\mathbf{u}}^\top \mathbf{v}_j$

else

$score_j = -\infty$

end if

end for

$j^* = \arg\max_j score_j$

$usedMovies = usedMovies \cup \{j^*\}$

Recommend movie $j^*$ and fetch rating $y^*$

$\mathbf{u} \leftarrow \text{UpdateUserDist}(\mathbf{u}, \mathbf{v}_j, y^*)$

end for
In Chapter 5 we presented the algorithms that we studied in the thesis. Now we show the benchmarking experiments that we performed. First, in Section 6.1 we introduce the setup and how we generated the dataset. As we already mentioned due to the nature of evaluation we were forced to use a synthetic dataset. Then in Section 6.2 we show the evaluation results.

6.1 Testing setup

All algorithms presented in the previous chapter are evaluated by calculating the regret. Let’s say a particular solution has ran for $T$ rounds for a given user. At round $t$ it recommended movie $j_t$ which received rating $y_t$ by the test user. Now, let’s denote with $y^*_t$ the rating that the user would give to their $t$-th favourite movie. For example, $y^*_1$ is the rating for their best favourite movie and $y^*_2$ is the rating for the second favourite one. The regret after $T$ rounds is defined as

$$ R_T = \sum_{t=1}^{T} (y^*_t - y_t) $$

For the purpose of nicer comparison of the regret for different number of rounds, in our plots we will be showing the average regret per round, i.e.

$$ \frac{R_T}{T} = \frac{\sum_{t=1}^{T} (y^*_t - y_t)}{T} $$

The explained scenario has a few details that make it almost impossible to evaluate offline on real world dataset. The algorithms are not restricted to recommend only fixed subset of the movies and are free to recommend any
6. Evaluation and results

We need to know the rating for every movie, because potentially any one can be recommended. Moreover, for the regret calculation we need to know the best \( T \) movies for the test user together with their ratings. This is infeasible for any real world dataset as usually users only watch and rate only a small portion of all the available movies.

That’s why, we had to come up with a way to synthetically test our solutions. In Section 6.1.1 we will see how to generate synthetic total dataset out of partial real dataset.

6.1.1 Synthetic dataset based on MovieLens

Our starting point for the generation of a synthetic dataset is choosing an authentic real world dataset. For the purpose of the thesis we have chosen the MovieLens dataset [2] provided by grouplens. The dataset contains ten million ratings applied to 10,000 movies by 70,000 users. Basically, MovieLens provides us a partial preference matrix \( R \) with \( N \approx 10,000 \) rows and \( M \approx 70,000 \) columns. After initial filtering we ended up with training dataset containing 69,878 users and 8,940 movies.

We take the matrix \( R \) and apply Bayesian Probabilistic Matrix Factorization as in [19, 1]. This gives us \( N \) \( D \)-dimensional user latent vectors \( u_i \) and \( M \) \( D \)-dimensional film latent vectors \( v_j \). \( D \), the number of features is a parameter to our data generation process. We can pick any value, but our assumption is that users and movies lie in low dimensional space. That’s why we have been working with values for \( D \) between 10 and 20. From now on, in our dataset we assume that the \( M \) vectors \( v_j \) are ground truth. This will be the allowed movies for recommendation and at each round we have to recommend one of those movies. The vectors \( v_j \) are the features of the movies which we take for granted in our system.

Now, we can divide the user vectors \( u_i \) into two groups training and testing vectors. The training vectors we use for estimating the prior user distribution, based on our assumption. For example, if the assumed prior distribution is Gaussian, we fit Gaussian distribution to the training vectors. And if the assumed distribution is Gaussian mixture, we use an Expectation Maximization algorithm [9, 17] for fitting GMM (GMM is abbreviation for Gaussian Mixture Model). The rest of the vectors are test users, i.e. we consider their feature vectors as ground truth. We calculate the inner product between each test user and each movie, and add independent Gaussian noise. This results into a full preference matrix which we use for evaluating the solutions. Given that full matrix, we are easily able to calculate the regret, i.e. the rating of the best \( T \) movies for each user and the rating for every user-movie pair.
6.2 Results

In the remaining part of the chapter we will present the evaluation results of the algorithms introduced in Chapter 5. In all cases we used the synthetic dataset from Section 6.1.1. In all plot the $y$ axis will show the average regret per round $R_T/T$ and the $x$ axis will show the number of rounds. Unless explicitly mentioned all plots are generated with the aggregated data from a 1000 test users.

6.2.1 Results for solutions with Gaussian prior

In Figure 6.1 and Figure 6.2 are displayed the average regret plots for the solutions with Gaussian prior. This is the case when we assume that a user with vector $u$ is sampled from single Gaussian distribution $\mathcal{N}(\mu_U, \Sigma_U)$. Figure 6.1 depicts the case when we have 10 features, i.e. $D = 10$, and Figure 6.2 depicts the case of 20 features.

Most figures in the remaining of this chapter will consist of two side-by-side plots. The left plot contains results for up to 30 rounds and the right plot shows results for up to 100 rounds. The left plots usually have error bars depicting the standard error in our measurements. Recall, that the plots are generated from the running for 1,000 test users. The standard error gives us bigger confidence that one solution is better than another one, and is defined as $s/\sqrt{n}$, where $s$ is the standard deviation and $n$ is the number of experiments. The idea of having two side-by-side plots is purely for nicer exposition.

The first thing we notice is that the naive solution is competitive only in the first few rounds, and after that it is significantly outperformed by the greedy and UCB solutions. Intuitively, this is exactly what we would expect. Usually in the first few rounds the naive algorithm recommends the very top movies which are liked by huge part of the user population, hence the small regret for the first 5 movies. After that the other algorithms catch up due to the fact that they are learning and refining their model.

Another thing we could notice on the plot is that UCB solutions slightly outperform greedy solutions after the first few rounds. Again, intuitively, this is expected as the UCB solutions take more risk in the beginning, but the result is that they learn more about the user. Notice that there are two UCB solutions evaluated. For example, in Figure 6.1 there are $ucb-beta08$ and $ucb-beta05-log$. Recall from Section 4.4.1 that the score for the $j$-th movie was $\hat{y}_j = \hat{\mu}_j + \sqrt{\beta_t \hat{\sigma}_j}$, where $\hat{\mu}_j$ and $\hat{\sigma}_j$ are the mean and the standard deviation of the predicted rating distribution, respectively, and $t$ is the number of the round. For the $ucb-beta08$ solution we have fixed $\beta_t = 0.8$ and for the

31
6. Evaluation and results

Figure 6.1: Solutions with single Gaussian prior (10 features)

Figure 6.2: Solutions with single Gaussian prior (20 features)

The most noticeable fact from the plot is that the solution based on Thompson sampling is quite under par compared to greedy and UCB. This is most probably due to the fact that in the very beginning when we sample from quite unknown distribution we could recommend a few random movies. This damages the regret a lot even though later the recommendations are good.

In Figure 6.3 we see a plot for 1000 rounds. It shows that with the accumulation of rounds Thompson sampling starts performing as good as greedy and UCB. The plot also shows that after 200 rounds the greedy and UCB
solutions are almost indistinguishable. However, in movie recommender systems we are usually interested in the initial rounds, not in the long term regret. Very few users of a real recommender system will rate more than a hundred movies, while the satisfaction in the first few rounds is quite important. A user who doesn’t like the initial recommendations is very likely to leave the system.

6.2.2 Results for solutions with Gaussian mixture prior

In this section we present the benchmark evaluation for the algorithms when the prior user distribution is not only a single Gaussian distribution, but a mixture of $k$ Gaussian distributions. In other words the assumed prior distribution is a weighted cluster of $k$ Gaussian distributions. We have benchmarked the solutions with number of clusters varying between 3 and 15. The results were very similar and for the plots here we have used prior distributions with 5 clusters.

In Figure 6.4 and Figure 6.5 we compare the regrets of the greedy solutions for 10- and 20-dimensional vectors, respectively. We can see that at times the solutions with GMM prior has slight advantage over the solution with single Gaussian prior. In all cases the difference is well within the measured standard error. It seems that introducing the mixture prior has negligible effect on the performance of the algorithm.
In Figure 6.6 and Figure 6.7 are shown the comparisons of the UCB solutions with constant rate $\beta_t$ for 10- and 20-dimensional vectors, respectively. Figure 6.8 and Figure 6.9 contain the corresponding plots for the case when the rate of $\beta_t$ is logarithmic.

Recall from Section 4.5.1 that there were two different ways to score each movie before recommendation, when the prior user distribution is a Gaussian mixture. The first way is by calculating an expression of the form $\hat{y}_i = \hat{\mu}_i + \sqrt{\beta_t} \hat{\sigma}_i$, where $\hat{\mu}_i$ and $\hat{\sigma}_i$ are the mean and the standard deviation of the predictive rating distribution, respectively. This is our default implementation. For example in Figure 6.6 the solution named 5 cluster gmm ucb-beta07 is such solution with fixed $\beta_t = 0.7$.

The solution 5 cluster gmm ucb-beta02-quant from the same figure is a
solution which calculates the $q$-th quantile of the mixture distribution. The value of $\beta_t$ is used to calculate the value of $q$. When we have a value for $\beta_t$ we calculate to which quantile $q$ will the expression $\mu + \sqrt{\beta_t} \sigma$ map for a Gaussian distribution. For example, let’s say we have the Gaussian distribution $\mathcal{N}(0, \sigma^2)$ and a value of $\beta$. Then the value of the Cumulative Distribution Function for the Gaussian distribution at $\sqrt{\beta} \sigma$ will evaluate to $q$ ($q = \text{cdf} (\sqrt{\beta} \sigma)$). Then, we calculate the $q$-th quantile of the mixture distribution as explained in Section 4.5.1.

Figure 6.6 and Figure 6.7 compare the solutions when the rate $\beta_t$ is fixed and independent of the round number and Figure 6.8 and Figure 6.9 compare the solutions when the rate is logarithmic, $\beta_t = \beta_0 \log t$.

Figure 6.10 and Figure 6.11 show the benchmark results for solutions using
6. Evaluation and results

Thompson sampling. Again, the solutions using mixture prior has slight unnoticeable advantage.

Figure 6.12 does not introduce new solution benchmarks. It just plots the comparison between greedy and UCB solutions when the prior user distribution is mixture on one plot.

6.2.3 Importance of $\alpha$

According to our rating prediction model, the rating a user with vector $\mathbf{u}$ would give to a movie with vector $\mathbf{v}$ is a sum of the inner product between $\mathbf{u}$ and $\mathbf{v}$ and independent Gaussian noise. Since our dataset is synthetic we have added the noise manually, and we know what is its precision (inversed
6.2. Results

variance), \( \alpha \). Moreover, \( \alpha \) is parameter to our algorithms. As we saw in Section 4.4.2 and Section 4.5.2 the way we update the user distribution after a round \( t \) depends on the value of \( \alpha \). The precision \( \alpha \) in the generated dataset that we used for benchmarking is 2, i.e. the standard deviation \( \sigma \approx 0.7 \). We used the same value of \( \alpha = 2 \) as parameter for our algorithms.

Figure 6.13 shows what happens when the greedy algorithm is fed with different value of \( \alpha \). The plot shows that being a little bit optimistic (\( \alpha = 4 \)) is not bad, but being too optimistic or conservative could harm the performance of the algorithm. \( \alpha = 2 \) corresponds to standard deviation of approximately 0.7. This is the true noise deviation. If the assumed precision is 4, i.e. standard deviation of 0.5 than the performance of the algorithm is very similar. However, when \( \alpha \) becomes 1 or 10, we notice worse performance. \( \alpha = 1 \) correspond to noise deviation of 1, while \( \alpha = 10 \) corresponds to noise
6. Evaluation and results

![Graph showing solutions with 5 cluster Gaussian mixture prior for 20 rounds (10 features)](image)

**Figure 6.12:** Solutions with 5 cluster Gaussian mixture prior for 20 rounds (10 features)

deviation of approximately 0.32.
6.2. Results

Figure 6.13: Single greedy solutions with varying $\alpha$ parameter (10 features)
Chapter 7

Conclusion

To sum up, we dealt with the problem of combining matrix factorization methods with multiarmed bandit approaches for solving the movie recommendation problem. We implemented and evaluated a number of algorithms. By taking an alternative approach for evaluating movie recommender systems we were forced to come up with novel synthetic offline testing. We weren’t able to confirm our result using real world dataset. It remains interesting opportunity to test our solutions online on real traffic with real users.

In our experiments it turned out that using mixture of Gaussian distributions as prior user distribution does not improve the performance of the algorithms. However this might not be the case in non-synthetic environment. Our dataset generation process was biased towards the fact that users are drawn from a Gaussian distribution. Recall that the first step of the synthetic data generation was running a probabilistic matrix factorization on a training matrix. One of the assumptions of that method was that both the users and the movies have Gaussian distributions. That may be the reason why the solutions with single Gaussian prior had advantage in our testing environment. At least, even in this scenario the GMM solutions were not inferior to the Gaussian solutions. Also, intuitively, it is more natural to assume that users are sampled from a mixture of distributions but not a single one. For example, there are users that like comedy movies and other types of users that like action or romantic movies. It seems logical to expect that each user is a weighted combination of different user types. That’s why according to us it is interesting to see how GMM and non-GMM solutions would compare in a more realistic scenario.

One negative take away from our experiments seems to be that solutions based on Thompson sampling are not on par with greedy and UCB solutions. This is due to the randomness of Thompson sampling in the begin-
7. Conclusion

ning and the short term regret in which we are interested. It might be interesting to see how Thompson sampling would perform when long term regret is more important. Such scenario could be music online radios, like Spotify and Pandora [4, 3]. In this case we have much more rounds and unlike movies, song recommendations can be repeated. In such case Thompson sampling could be preferable as it explores more than greedy solutions and requires less tuning compared to UCB solutions.

A possible improvement to our algorithms would be if we remove the constraint that the set of movies is fixed. We treated movies as a fixed set with fixed feature vectors. This restriction, however, can be loosen. We can introduce a new movie the same we introduce a new user. First we assume some prior distribution and then after every recommendation refine the distribution. The question then would be to which users should we recommend a new movie.

In our work we introduced a way to evaluate the regret of a movie recommender. A measurement we think is more interesting than the Root Mean Squared Error (RMSE). However, this could be further improved. In the way that accumulated regret is defined it is a non-weighted average of the regret in each round. In a real recommender system it is natural to assume that the regret in the very first rounds is more important. In real world environments it is a common situation that if the user is not satisfied with the initial results they tend to leave the system. That’s why it is very interesting to see how greedy and UCB solution would compare in real environment. UCB had slight advantage in our experiments, but at the cost of “risky” recommendations in the beginning.

In overall, we managed to combine factorization methods with multiarmed bandit approaches for movie recommendations. We tested our implementations in synthetic environment and showed that the presented algorithms make sense given some initial assumptions.
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):
EXPLORATION-EXPLOITATION TRADE-OFFS VIA PROBABILISTIC MATRIX FACTORIZATION

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):
PRISADNIKOV

First name(s):
VEDYALKO

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date
ZÜRICH, 19.08.2014

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.