TECHNICAL UNIVERSITY OF DENMARK

M.Sc. in Mathematical Modelling and Computation

Master's Thesis

LEARNING TO INDEX

Marco Fraccaro

Supervisor:
Prof. Ole Winther
Co-Supervisor:
Dr. Ulrich Paquet

Kongens Lyngby 2014

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk IMM-M.Sc.-2014

To my family

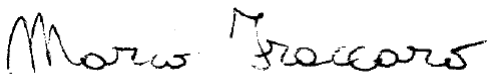
Abstract

Recommender systems are used in online platforms to make personalized suggestions of new items to users. For systems based on matrix factorization techniques the recommendation step scales linearly with the number of objects in the catalog. This leads to a serious bottleneck in large-scale applications that have a strict time budget and in which there may be millions of items.
In this work it is developed a probabilistic model for the recommender system that thanks to some constraints allows to give high-quality suggestions in sublinear time, exploiting fast tree structures originally built for nearest neighbor searches.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science (DTU Compute) at the Technical University of Denmark in fulfillment of the requirements for acquiring an M.Sc. in Mathematical Modelling and Computation.
The project supervisors are prof. Ole Winther from the Technical University of Denmark and dr. Ulrich Paquet from Microsoft Research Cambridge.
The work of this thesis will be continued in a PhD at the Technical University of Denmark.

Lyngby, 09-August-2014



Marco Fraccaro

Acknowledgements

This degree wouldn't have been possible without the support of a lot of people that I now wish to acknowledge!
I owe a big thank you to my supervisor Ole Winther, for our fruitful discussions and for always challenging me with advanced projects. Thanks to my co-supervisor Ulrich Paquet as well, for his valuable suggestions that helped me to understand the problems in large-scale recommender systems and for hosting me in Cambridge. It's going to be a (tough but) fun PhD! :-)
It is impossible to mention all the wonderful people I have met both in Italy and in Denmark, hence I thank you all for the awesome time spent together! Among the "Italians" I want to acknowledge in particular my dear friends Davide, Matteo (Paglia) and Alessio for all our adventures and for supporting me when I moved to Denmark. The time spent at the University of Padova would have not been the same without Davide and Giulio: thanks to our mutual support we are now great Telecommunication Engineers! ;-)
In Copenhagen I have lived with some very cool people in "Little Italy" (Alessandro, Dario, Alessia, Maicol, Claudia): thanks for being my Danish family and making me feel always at home (not to mention the all the fancy dinners and parties)! A special thanks also to Stefano, Matteo, Varun and all the people in Container G! I hope one day to visit you all wherever you may be all over the world!
Last but not least I want to show my gratitude to my family, that have supported me both personally and economically through all my studies, allowing me to become a better person and now an Engineer!!!

Contents

Chapter 1

Introduction

Over the last few years *recommender systems* have become fundamental in helping users to find the right items in the enormous catalogs the Web offers. When entering an e-commerce website such as Amazon for example, one of the first things a customer is shown is a list of recommended products that match the customer's tastes. In a similar way, the most suitable songs, videos and movies are suggested to users in online platforms such as Spotify, Youtube and Netflix. Good recommendation are crucial to increase customers' satisfaction and loyalty, and assume therefore a major role for the success and revenue of these platforms.
The development of reliable recommender systems has become possible after the advent of machine leaning techniques: making the system learn from available training data, one can in fact perform tasks that were considered too difficult to solve in the past using just explicit programs (constructed for a specific operation). With recommender systems in particular it is possible to teach a computer how to understand what a user likes and use this information to suggest new items. In this thesis we will go one step further: not only we want the system to learn how to understand the user's tastes, but we also want it to learn how to organize the data in such a way that the best recommendations for each of the users can be found efficiently[1]. One of the main bottlenecks in large scale systems is in fact the enormous number of items in the catalogs (e.g. one could choose among millions of songs): this makes it is impossible to give accurate

[1] This is the reason behind the title of the thesis, *learning to index*.

suggestions in real time. To deal with this issue modern systems only consider small subsets of the items formed with some heuristics but, to the best of our knowledge, no one before has tried do deal with the whole catalog at once.

Some of the most common recommender systems suggest an item to an user according to what several other people with similar tastes have liked in the past. In particular, knowing for each user the list of rated items in the catalog, it is possible to construct a rating matrix $\mathbf{R}$, whose element (i, j) contains the rating that user i has given to item j. If we have N users and M items, $\mathbf{R}$ is then a $N \times M$ matrix, and it is typically extremely sparse (it can have for example much less than 1% of non-empty elements). Matrix factorization techniques in particular (Koren, 2009), that are presented in chapter 5, approximate the rating matrix with the product of two much smaller matrices plus a residual term, i.e. $\mathbf{R} = \mathbf{U}^T\mathbf{V} + \mathbf{E}$, where $\mathbf{U}$ is an $D \times N$ user matrix, the item matrix $\mathbf{V}$ is $D \times M$ and the residual term $\mathbf{E}$ is $N \times M$. All the N user and M items are hence associated to a D-dimensional vector in which each of the dimensions represents a different hidden factor. In movies recommendations for example these factors may be interpreted as common aspects such as the type of movie (e.g. comedy vs horror) while some of them can be really difficult to explain. The factorization is performed using a rather complex probabilistic graphical model, that allows to take into account uncertainties in the suggestions. The machine learning tools necessary for the training phase of the model are introduced in Chapter 3.
Once the model is trained, it recommends an item to an user if they both have similar hidden factors. This means that to make a suggestion to an user one has to compute the similarity between the user vector and all the M item vectors: the recommendation step to offer the best possible choice scales therefore linearly with the number of items. This is can be a huge problem for all the applications in which the number of items in the catalog is bigger than some hundreds of thousands of elements. In this case it is in fact impossible to scan the whole list of items in real time, therefore, as said above, just some small subsets of elements formed with some heuristics can be analyzed, leading to non-optimal suggestions.
In chapter 6 we will however show that putting some particular constraints to the norm of the item vectors and exploiting the so called metric data structures (Samet, 2005), the recommendation step can be performed with a drastic reduction in the retrieval time (in logarithmic time with respect to the number of items in the ideal case). We will therefore need to formulate a new model for the recommender system that takes these constraints into account. This will be done in chapter 7. The resulting model will be more complex to construct and handle: some advanced Markov Chain Monte Carlo techniques that exploit differential geometry (presented in chapter 4) will be needed for an efficient training phase. The results obtained using this model show in particular that the constraints imposed only slightly influence the final quality of the recom-

mendations.
Combining the proposed model with the data structures introduced in chapter 8 we will be finally able to obtain a 4x speedup with high quality suggestions for the recommendation step. The example proposed has relatively a small size, meaning that the actual speedup achievable in larger scale systems will be much higher.
In the next chapter we will give a general introduction on recommender systems, that is necessary to understand the main issues involved in real world applications.

Chapter 2

Recommender Systems

2.1 Introduction

The main goal of recommender systems is that of supporting users in their (online) decision making, helping them to discover interesting items they might not have found by themselves (Rajaraman and Ullman, 2012; Jannach et al., 2011). These systems are nowadays used in a wide range of different applications: they can suggest for example movies, books, music, games or news to read. As we will see shortly, the recommendations are personalized, in the sense that they are based on an user's past preferences.
Recommender systems have become so popular as both service providers and users benefit from them (Ricci et al., 2011):

- From a *service providers* point of view, they allow to increase the number of items sold, as the suggestions are likely to meet the user's needs. Good recommendations lead furthermore to a high users' satisfaction and fidelity, as customers know they can trust the system. Finally, service providers can also better understand what the user wants, and use this knowledge for other goals such as for advertisements and inventory prediction.
- For *users*, on the other hand, recommender systems allow to always dis-

cover new and very different items that were suggested based on his/her previous past history. Due to the huge size of modern online catalogs it is in fact difficult to explore them in a principled way and to find the right items. The more the user interacts with the system, the more data is available to improve the quality of the recommendations.

The machine learning techniques usually employed for recommender systems follow often a probabilistic approach to be able to deal with uncertainties in the recommendations: if the system is in fact not sure if a suggestion is really good for the user, then it should not be given.

2.2 The long-tail phenomenon

Due to a limited space in the shelves, physical shops such as bookstores cannot display all the items in their catalog, and are therefore forced to show just the most popular ones. Online systems, on the other hand, can make anything that exists available to the customer. Anderson (2006) theorizes this from an economical point of view, showing that this focus on a large amount of items is part of the business model and a fundamental source of profit of online companies such as Amazon. The term *long tail* was introduced to describe this strategy of selling a large number of unique items in relatively small quantities together with the few very popular items. A pictorial representation of the long tail phenomenon is shown in Figure 2.1, where we can see that there exists a power law relation between popularity and the items ordered according to their popularity: in every catalog there is a small amount of very popular items (the red area in the figure) and a large number of less popular ones (green area). While the main source of income of physical retailers comes from the red area, the success of online companies is given by selling items in the whole spectrum.

This phenomenon leads to a very important difference between physical and online shops when considering recommendations. Recommendations in the physical world are rather simple: as it is not possible to tailor the store to each individual customer the best-selling strategy is that of showing (i.e. suggesting) just the most popular items to the left of the vertical line in Figure 2.1. To earn as much as possible, on-line companies on the other hand are forced by the long tail phenomenon to create recommender systems that provide personalized suggestions to their users: it is in fact not possible to show the whole catalog at once, and furthermore we cannot expect users to have heard of each of the items they might like.
It is interesting to see in Figure 2.2 how the online streaming provider Netflix recommends movies and tv series. The first thing one sees when accessing the

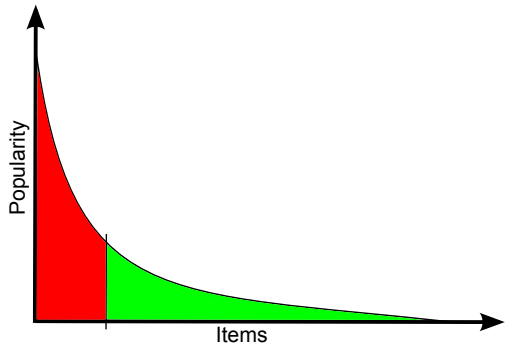


Figure 2.1: The long tail. The red area represents few very popular items (e.g. movies), while the majority of them are less popular and lie in the green area. Recommender systems are built to suggest items on the whole spectrum.

home page is a list of suggestions divided in two parts. On the top we find "popular on Netflix", i.e. very popular items belonging to the red area in Figure 2.1 such as Gossip Girl and Breaking Bad. On the bottom we find instead the "top picks" for the user, hence elements mostly from the green area: we see that in this case the items are not well known, but are personalized considering the taste of the user (in this case romantic and comedy movies).

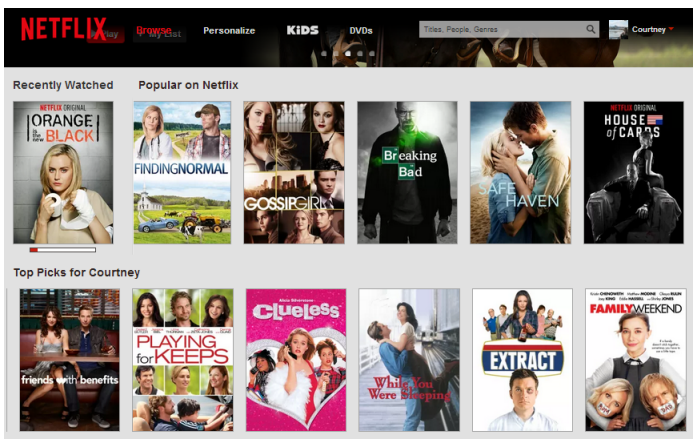


Figure 2.2: The recommender system used at Netflix. On the top row we find suggestions for very popular movies belonging to the red area of Figure 2.1, while on the bottom one we find movies from the green one as well.

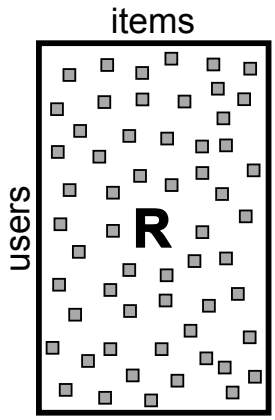


Figure 2.3: Example of utility matrix. Each user is represented by a row, whereas each item is represented by a column; the grey squares represent ratings.

2.3 Building the utility matrix

The only way to recommend items in a personalized manner, is that of gathering some information on what the user likes. This is the purpose of the *utility matrix*, that holds in its element (i, j) information about the degree to which user i likes item j (Rajaraman and Ullman, 2012). This is usually a very sparse matrix, as each user usually deals with just a small subset of the items, see Figure 2.3 for an example. The goal of a recommender system is therefore that of predicting the blanks in the utility matrix based on the values of the known entries, so that it is possible to suggest to the user the items that he/she would probably like.
There are two main ways to acquire data for the utility matrix:

- **Explicit feedback.** We can ask users to rate items. In this case the utility matrix is also called *user rating matrix*. The main issue with this approach is however that some users are rather lazy and not willing to rate their items, to the detriment of the quality of their suggestions.

- **Implicit feedback.** We can infer whether a user likes an item analyzing his/her behavior. This can be done for example in movie recommendation assuming that users only see movies that they like. This approach has the advantage that no direct feedback is required from the user, but on the other hand the assumption made can be not completely true.
  This rating system has only one value (e.g. a 1 in the utility matrix) that

indicates that the user likes the item.

As both methods have pros and cons, hybrid solutions that combine them are also well established. They are particularly important to face the *cold start* problem, that may occur in the absence of enough data for newly created recommender systems or newly introduced users/items.

2.4 Types of recommender systems

Recommender systems can be broadly classified in two groups:

1. In **content-based systems** for each item it is constructed a profile that summarizes its most important characteristics (e.g. for movies: the director, the actors, the year and the genre). Using the corresponding row of the utility matrix we also build a profile for each user based on a weighted vector of item features. The best-matching items are then recommended: if a user has seen lots of modern action movies, then so will be the recommended ones. Due to these association properties it is however not possible to suggest items that are very different from the ones a user has seen but that he/she still would like, i.e. discovery is penalized.
2. **Collaborative filtering systems** on the other hand recommend items that similar users like[1]. Therefore, we do not focus any longer directly on user and item profiles (that can be quite complex to construct) but rather on the best way to define a similarity among users. This approach mainly suffers from the cold start problem mentioned above, from its scalability (as in systems with a huge number of users and items the comparisons are very expensive) and finally form the sparsity of the data. Collaborative filtering systems however allow users to discover very different items, as if two users have similar tastes then any item one user has liked will likely provide a good suggestion for the other.
   This class of algorithms is widely used for example in social networks such as Facebook and LinkedIn to recommend new friends/connections (Ricci et al., 2011). A collaborative filtering algorithm known as matrix factorization will be introduced in detail in chapter 5.

Also in this case it is possible to combine the two methods to overcome the issues of the two approaches (hybrid systems).

[1] The process of identifying similar users and recommending what similar users like is called collaborative filtering.

In practice recommender systems can be very complex if one wishes to model as much as possible the behavior of real users. It is important to consider for example that some users tend to give systematically higher/lower ratings compared to the average or that some items are much more popular than others and tend therefore to receive higher ratings (Paquet and Koenigstein, 2013). Also, as the taste of an user and the popularity of an item change over time, a system that takes temporal dynamics into account can lead to great improvements (Koren, 2009). Finally, very useful can also be the usage of implicit information such as the browsing history of the customer or the time spent observing each item (Pilászy et al., 2010).

2.5 The Netflix data set

In October 2006 the American online DVD-rental service Netflix released as part of the "Netflix prize" challenge a data set containing 100480507 ratings given to 17770 movies by 480189 users (Bennett and Lanning, 2007)[2]. The purpose of the challenge was that of improving its own recommender system, used to suggest new movies to its users according to their previous ratings history. A prize of 1 million US dollars was promised to the first team able to improve Netflix's system *Cinematch* by more than 10% in terms of Root Mean Square Error (RMSE) on the test set.
The training set is constructed with data collected between October 1998 and December 2005. Each training sample comprises two integer IDs for users and movies respectively, a rating from 1 to 5 (integer) stars and the date of the rating. The qualifying set contains 2817131 triplets with the ID of the user, the ID of the movie and the date of the rating: the actual rating is not given to the contestants but it is known only to the jury, as it will be used to find the best performing team. The triplets in the qualifying set were selected among the most recent ratings of a subset of the users in the training set. To determine their progress maximum once a day the teams could upload their predicted ratings for all the elements in the qualifying set. The RMSE of an unknown half of the qualifying set (called quiz set) was then posted to the public leader board; the results on the other half of the qualifying set (the test set) were not reported in the leader board but were instead used to determine the final winner. The prize was won after three years, in September 2009, by the BellKor's Pragmatic Chaos team, that achieved a 10.06% improvement over Netflix's recommender system (Koren, 2009).
We will now provide a more detailed analysis of the data set, as it will be useful in the evaluation of the recommender system developed in this thesis.

[2]See also www.netflixprize.com.

Ratings

An histogram of the ratings in the training set is shown in figure 2.4. We see that the number of low ratings (1 and 2 stars) is quite small, as expected from the fact that a user tends to only see movies that he thinks he could like. The mean rating is 3.6043 stars.

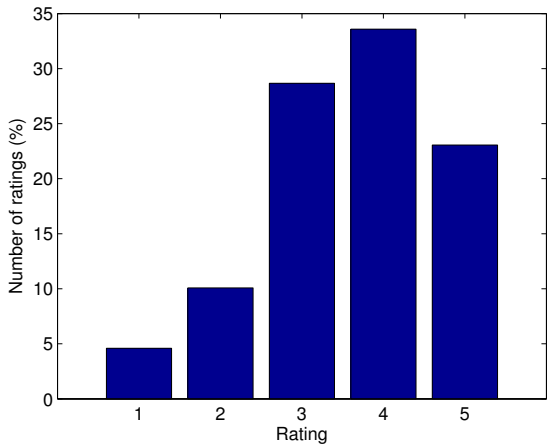


Figure 2.4: Histogram of the ratings for the Netflix data set.

Groups of Users

In any recommender system there will be different types of users: some of them will only rate few items whereas others may rate a big chunk of them. It is however important that all the different kind of users benefit from the recommendations, i.e. the system should provide high quality suggestions to all of them. Following the same approach as in (Salakhutdinov and Mnih, 2008b) we therefore divide all the users with a similar number of ratings in 9 groups, and compute a separate RMSE for each of them. The number of users belonging to each group is shown in Figure 2.5.
The average user has rated around 209 movies, but we see that there is a huge variability in the data: some of the users have just one rating and one of them has seen 17653 movies.

Groups of Movies

In a similar fashion, we can also divide the movies in groups according to the number of ratings they have, see Figure 2.6. The intervals were chosen both to have a big focus on rare movies and to have a bell-shaped histogram similar to the one in Figure 2.5.
The average movie was seen around 5650 times, with some of them having just 3 ratings and the most popular ones with more than 200000 views.

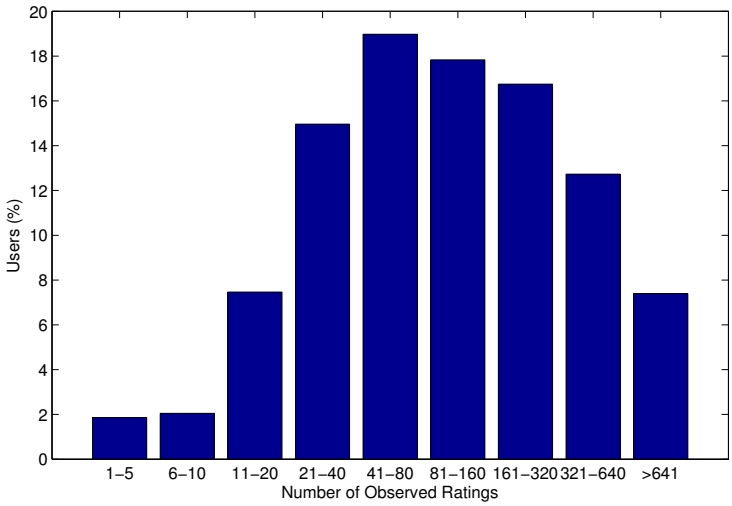


Figure 2.5: Grouping of the users in the Netflix data set. We see for example that around 2% of the users have given between 6 and 10 ratings.

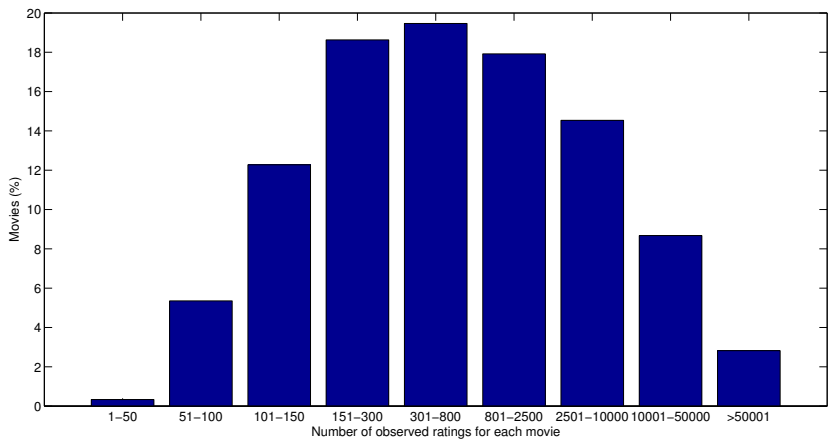


Figure 2.6: Grouping of the movies in the Netflix data set.

In the next two chapters we will introduce some necessary machine learning methods that will be needed in the development of the recommender system.

Chapter 3

MCMC Methods for Bayesian Inference

3.1 The Bayesian perspective

Let us consider a set $\mathcal{D}$ of observed data, and the parameters $\mathbf{q}$ of a model that we want to fit to it. We can make assumptions about the parameters before observing the data through a *prior probability* $p(\mathbf{q})$ independent form $\mathcal{D}$. The uncertainty in the parameters after we have seen the data can be then represented by the *posterior probability*, calculated using *Bayes' theorem* as

$$p(\mathbf{q}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{q})p(\mathbf{q})}{p(\mathcal{D})} \ . \tag{3.1}$$

The quantity $p(\mathcal{D}|\mathbf{q})$ is called *likelihood function*, and represents the probability of observing the data $\mathcal{D}$ given the set of parameters $\mathbf{q}$. Note that with respect to the parameters of the model the likelihood is not a probability but simply a function, and its integral over $\mathbf{q}$ will not necessarily sum to one. Finally the quantity $p(\mathcal{D})$ is called *normalizing constant*, as it makes $p(\mathbf{q}|\mathcal{D})$ integrate to one. For many of the most common models the value of $p(\mathcal{D})$ is very difficult to obtain or even approximate, hence it will be important to develop methods that do not depend on it but that are able to deal with the posterior distribution up to a proportionality factor.

The frequentist approach to perform inference of the model assumes that $\mathbf{q}$ is a fixed vector, that is determined using some estimator. A widely used one is the *maximum likelihood (ML)* estimator, in which a point estimate of $\mathbf{q}$ is obtained, as the name suggests, using optimization techniques to find the value of $\mathbf{q}$ that maximizes the likelihood function (equivalently, minimizing the negative log of the likelihood function). This approach is often the simplest to implement, but some problems arise as no assumption on the distribution of the parameters is made: $p(\mathcal{D}|\mathbf{q})$ could be maximized by some parameters whose a priori probability is very low. For example, the parameters found by the optimization procedure, could not even have a physical meaning. A closely related but improved frequentist approach maximizes the probability of the parameters given the data instead, i.e. the posterior distribution $p(\mathbf{q}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{q})p(\mathbf{q})$. This is called *maximum a posteriori (MAP)* estimate, and it takes into account the prior distribution of the parameters through $p(\mathbf{q})$.
Both the ML and MAP estimators provide a point estimate of the parameters, that can be used for example to predict new outcomes. A Bayesian approach on the other hand considers in all the calculations $\mathbf{q}$ as a random variable with probability distribution $p(\mathbf{q})$, i.e. no point estimate is obtained but the sum and product rules of probability are used to consider all the possible values of $\mathbf{q}$ weighted by their own prior probability.
Note that the prior distribution $p(\mathbf{q})$ could be itself a parametrized distribution, and depend on some parameters $\mathbf{\Theta}$. It is of course possible to place a prior distribution $p(\mathbf{\Theta})$ on $\mathbf{\Theta}$ as well: we call this distribution *hyperprior*.

Example
In a simple linear regression model the output is modelled with a linear combination of the input parameters plus a residual term:

$$y = \mathbf{q}^T\mathbf{x} = q_1x_1 + \dots q_Dx_D + \varepsilon \ .$$

Having N training examples $(\mathbf{x}_i, y_i)$, $i = 1, \dots N$, and with the assumption of gaussian noise it can be easily shown (Bishop, 2006; Murphy, 2012) that the maximum likelihood estimate of the parameters is given by

$$\mathbf{q}_{ML} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\boldsymbol{y} \ ,$$

where $\mathbf{X}$ is a $N \times D$ matrix containing the training inputs in each row and $\boldsymbol{y}$ is the $N \times 1$ vector of training outputs.
Especially with data sets of limited sizes the ML estimate tends to overfit, and a regularization term λ is often necessary. This can be equivalently seen in a MAP framework as placing a gaussian prior with mean zero and covariance matrix $\lambda\mathbf{I}$ (where $\mathbf{I}$ is the identity matrix) to the parameters:

$$\mathbf{q}_{MAP} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\boldsymbol{y} \ .$$

Given an estimate $\hat{\mathbf{q}}$ obtained for example as just shown with ML or MAP and a new test input $\mathbf{x}_{new}$, the predicted output can be evaluated as

$$y_{new} = \hat{\mathbf{q}}^T \mathbf{x}_{new} \ .$$

As we are only interested in predicting a new outcome given the input vector $\mathbf{x}_{new}$, we do not necessary need to have a point estimate of $\mathbf{q}$. This is the main idea behind the Bayesian point of view, in which the posterior distribution is evaluated using (3.1) and it is obtained a predictive distribution $p(y_{new}|\mathcal{D}, \mathbf{x}_{new})$ instead of just a single predictive value y_{new}:

$$p(y_{new}|\mathcal{D}, \mathbf{x}_{new}) = \int p(y_{new}|\mathbf{q}, \mathbf{x}_{new}) p(\mathbf{q}|\mathcal{D}) \mathrm{d}\mathbf{q} \ .$$

It is important to point out in particular that there is no dependence on the parameters in the predictive distributions: they are in fact integrated out. We can also exploit the new information on the probability distribution of the output to deal in a more principled way with the uncertainties in the estimate, for example calculating the confidence intervals.
For the considered linear regression model in particular, it is possible to show that both the posterior and the predictive distribution have a normal distribution (Bishop, 2006).

For many models of practical interest it will be unfeasible to calculate the posterior distribution or to evaluate expectations with respect to this distribution (e.g. the integral in the predictive distribution for the linear regression model). This could be because of the complexity of the posterior distribution itself that leads to non-analytic solutions of the integral and/or because the high dimensionality of the problem in hand may prohibit numerical integration. To deal with these situations there are two main families of approximation schemes:

- Stochastic techniques (*sampling methods*) draw samples from the posterior distribution and approximate any expected value using a sample average. These methods have the nice property of producing exact results given infinite computational power, but they are computationally expensive (hence they are mostly used for small-scale problems). Furthermore, it is often difficult to tune them and to assess their convergence. Among this family of algorithms we find for example *Markov Chain Monte Carlo (MCMC)* methods (Gilks, 1999).

- Deterministic techniques on the other hand, approximate analytically the posterior distribution combining analytically convenient assumptions on

its form or factorization (e.g. the posterior distribution is approximated as a product of parametrized Gaussians) with optimization techniques. They can never generate exact results but they scale really well to large applications. Among these techniques we can find *variational inference* (Bishop, 2006) and *expectation propagation* (Minka, 2001).

We will now give a theoretical introduction of sampling methods focusing mainly on MCMC, as they will be used later on to sample from the posterior distribution of the model parameters given the training data in our recommender system application.

3.2 Sampling methods

As shown above, in most of the applications the posterior distribution is necessary to compute expectations, for example in order to make predictions. We now therefore consider the problem of evaluating the expectation of some function $f(\mathbf{q})$ with respect to any probability distribution $p(\mathbf{q})$. In our case, $p(\mathbf{q})$ represents a posterior distribution but we drop the dependence on the training set $\mathcal{D}$ for notational convenience.
For continuous variables the expectation is given by

$$\mathbb{E}[f] = \int f(\mathbf{q})p(\mathbf{q})\mathrm{d}\mathbf{q} \ , \tag{3.2}$$

whereas the integral is replaced by a summation for discrete variables. If we have a set of independent samples $\mathbf{q}^{(r)}$ drawn from the distribution $p(\mathbf{q})$, where $r = 1, \dots R$, the expectation in (3.2) can be approximated with an unbiased estimatior obtained with a finite sum

$$\hat{f} = \frac{1}{R}\sum_{r=1}^{R} f(\mathbf{q}^{(r)}) \ .$$

Despite being conceptually straightforward, this families of methods usually require a lot of experience and fine tuning to work properly. Any computing environment can easily draw independent random samples from a simple posterior distributions (e.g. the function `randn` in Matlab for Gaussians), but as soon as the model used is slightly more complex no standard sampling algorithms are known (Murphy, 2012). Several methods have been developed to allow the evaluation of the expectation from any distribution, among them *rejection sampling*, *importance sampling*, *particle filters* and the already mentioned *MCMC*, that will be the subject of the rest of the chapter. It is finally worth noting that

the main problem that arises when using these techniques is that the samples that can be obtained from $p(\mathbf{q})$ are usually quite far from being independent. If the samples are too correlated they might not provide an accurate empirical estimate of the distribution $p(\mathbf{q})$, and the approximated value of the expectation can be very different from the real one.

3.3 Markov chains and MCMC methods

A *stochastic process* is a collection of random variables that represents the evolution of a system over time. Stochastic processes represent the probabilistic counterpart of deterministic systems, as instead of having just one possible solution to the evolution of a system (such as for deterministic systems governed by ordinal differential equations) there is some indeterminacy that is described by probability distributions.

Markov chains are random processes characterized by a memoryless property, in that the state of the Markov chain at time $n+1$ depends only on the state at time n (we are therefore considering more precisely *first order* Markov chains). This can be written in terms of a conditional distribution as

$$p(\mathbf{q}^{(n+1)}|\mathbf{q}^{(1)},\ldots,\mathbf{q}^{(n)}) = p(\mathbf{q}^{(n+1)}|\mathbf{q}^{(n)}) \ .$$

$\mathbf{q}^{(n+1)}$ is therefore conditionally independent from $\mathbf{q}^{(1)},\ldots,\mathbf{q}^{(n-1)}$. To fully specify a Markov chain, one has to assign the probability density $p(\mathbf{q}^{(0)})$ of the initial state and the *transition probabilities* $T_n(\mathbf{q}^{(n)},\mathbf{q}^{(n+1)}) \equiv p(\mathbf{q}^{(n+1)}|\mathbf{q}^{(n)})$. We consider *homogeneous* Markov chains, that have the property that the transition distributions do not depend on the time instant n, i.e. $T_n = T$.
For a particular variable we can evaluate its marginal probability given the marginal probability of the previous variable and the transition probabilities:

$$p(\mathbf{q}^{(n+1)}) = \int_{\mathbf{q}^{(n)}} p(\mathbf{q}^{(n+1)}|\mathbf{q}^{(n)})p(\mathbf{q}^{(n)})\mathrm{d}\mathbf{q}^{(n)} \ .$$

A *stationary or invariant distribution* $p(\mathbf{q})$ for a Markov chain has the property that

$$p(\mathbf{q}) = \int_{\mathbf{q}'} T(\mathbf{q}',\mathbf{q})p(\mathbf{q}')\mathrm{d}\mathbf{q}' \ ,$$

in other words if the marginal distribution at time n is given by $p(\mathbf{q})$, then the marginal distribution for all the following time instants will be $p(\mathbf{q})$ (note that a Markov chain can have more than one stationary distribution).
A sufficient condition for a distribution to be stationary is that it satisfies the property of *detailed balance*:

$$p(\mathbf{q})T(\mathbf{q},\mathbf{q}') = p(\mathbf{q}')T(\mathbf{q}',\mathbf{q}) \ .$$

To prove this we just need to notice that

$$\int_{\mathbf{q}'} p(\mathbf{q}')T(\mathbf{q}',\mathbf{q})\mathrm{d}\mathbf{q}' = \int_{\mathbf{q}'} p(\mathbf{q})T(\mathbf{q},\mathbf{q}')\mathrm{d}\mathbf{q}' = p(\mathbf{q})\int_{\mathbf{q}'} p(\mathbf{q}'|\mathbf{q})\mathrm{d}\mathbf{q}' = p(\mathbf{q}) \ .$$

A Markov chain that satisfies detailed balance is defined as *reversible*. The detailed balance equation says that the probability flux from state $\mathbf{q}$ to $\mathbf{q}'$ equals the flux from $\mathbf{q}'$ to $\mathbf{q}$, or equivalently that the flux out of state $\mathbf{q}$ should be equal to the flux into $\mathbf{q}$.
Starting from the initial state $\mathbf{q}^{(0)}$, the stochastic process can be simulated by moving to a new state according to the transition probabilities. A stationary distribution of the Markov chain can be seen as the long term distribution of the chain obtained simulating the process.

MCMC methods draw posterior samples by constructing a Markov Chain on the parameter space whose stationary distribution is the distribution of interest $p(\mathbf{q})$. After some burn-in samples, we will eventually converge to the stationary distribution of the chain and will visit the states with probability given by $p(\mathbf{q})$, i.e. the fraction of time spent in each state $\mathbf{q}$ is proportional to $p(\mathbf{q})$. However, we also have to require that the convergence is obtained for any choice of the starting point: we want therefore a so called *ergodic* Markov chain. It should be clear that an ergodic Markov chain can only have a single stationary distribution, that can be called in this case *equilibrium distribution*. Under some weak restrictions on the stationary distribution and the transition probabilities an homogeneous Markov chain will be ergodic, see for example (Murphy, 2012) for a detailed proof. Apart from the high computational power required, one of the main issues with MCMC methods is that there are no standard ways to assess the convergence of the chain to the stationary distribution.

3.4 The Metropolis-Hastings Algorithm

The first MCMC method that we introduce is the *Metropolis-Hastings (MH)* algorithm (Metropolis et al., 1953; Hastings, 1970). Being in state $\mathbf{q}$, it proposes to move to a new state $\mathbf{q}'$ with probability $g(\mathbf{q}'|\mathbf{q})$, where g is called the *proposal distribution* (that needs to be easy to sample from). The proposed state is then accepted with probability

$$\alpha(\mathbf{q}',\mathbf{q}) = \min\left\{1, \frac{p(\mathbf{q}')g(\mathbf{q}|\mathbf{q}')}{p(\mathbf{q})g(\mathbf{q}'|\mathbf{q})}\right\} = \min\left\{1, \frac{p(\mathbf{q}')/g(\mathbf{q}'|\mathbf{q})}{p(\mathbf{q})/g(\mathbf{q}|\mathbf{q}')}\right\} \ , \tag{3.3}$$

otherwise the new state is set to the current one, i.e. the sample $\mathbf{q}$ is repeated. As we will see shortly, this acceptance rate is necessary for detailed balance to

be satisfied.
A common choice for the proposal distribution is a Gaussian distribution centered on the current state, i.e. $g(\mathbf{q}'|\mathbf{q})) = \mathcal{N}(\mathbf{q}'; \mathbf{q}, \mathbf{\Sigma})$. In this case the tuning of the covariance matrix $\mathbf{\Sigma}$ is critical for an efficient sampling procedure, especially when dealing with highly correlated distributions: a too small variance would give high acceptance probabilities but not allow to explore thoroughly the state space. On the other hand, if the variance is too high many of the proposed steps will be towards regions of low probability density $p(\mathbf{q})$ and the acceptance rate would be really small. The sequence of states with Gaussian proposals represents a random walk, and when necessary to avoid confusion the algorithm is dubbed *random walk Metropolis-Hastings*. Taking L small steps in random directions is likely to move the state just about $\sqrt{L}$ steps away from the starting point (Neal, 2010).
In the Gaussian case, and more in general for all symmetric proposal distributions, i.e. $g(\mathbf{q}'|\mathbf{q}) = g(\mathbf{q}|\mathbf{q}')$, the acceptance probability is simply given by:

$$\alpha(\mathbf{q}', \mathbf{q}) = \min\left\{1, \frac{p(\mathbf{q}')}{p(\mathbf{q})}\right\} . \tag{3.4}$$

We therefore notice that moves that lead to regions with higher density are always accepted, whereas if $\mathbf{q}'$ is less probable than $\mathbf{q}$ we move with a probability given by $\frac{p(\mathbf{q}')}{p(\mathbf{q})}$. The extra term in (3.3) can be therefore seen as a compensating the fact that the proposal distribution itself might favor certain states.
An appealing property of the MH algorithm is that the target distribution needs to be known only up to a normalizing constant. If we consider the unnormalized target distribution $\tilde{p}(\mathbf{q})$ such that $p(\mathbf{q}) = \frac{1}{Z}\tilde{p}(\mathbf{q})$, then the possibly unknown normalizing constant Z cancels in the acceptance ratio:

$$\alpha(\mathbf{q}', \mathbf{q}) = \min\left\{1, \frac{\frac{1}{Z}\tilde{p}(\mathbf{q}')g(\mathbf{q}|\mathbf{q}')}{\frac{1}{Z}\tilde{p}(\mathbf{q})g(\mathbf{q}'|\mathbf{q})}\right\} = \min\left\{1, \frac{\tilde{p}(\mathbf{q}')g(\mathbf{q}|\mathbf{q}')}{\tilde{p}(\mathbf{q})g(\mathbf{q}'|\mathbf{q})}\right\} .$$

We can therefore sample from a distribution with MH even if we do not know its normalizing constant. To show that the target distribution $p(\mathbf{q})$ is a stationary distribution for the Markov chain we can for example prove that detailed balance is satisfied:

$$\begin{aligned} p(\mathbf{q})g(\mathbf{q}'|\mathbf{q})\alpha(\mathbf{q}', \mathbf{q}) &= \min\{p(\mathbf{q})g(\mathbf{q}'|\mathbf{q}), p(\mathbf{q}')g(\mathbf{q}|\mathbf{q}')\} \\ &= \min\{p(\mathbf{q}')g(\mathbf{q}|\mathbf{q}'), p(\mathbf{q})g(\mathbf{q}'|\mathbf{q})\} \\ &= p(\mathbf{q}')g(\mathbf{q}, \mathbf{q}')\alpha(\mathbf{q}, \mathbf{q}') . \end{aligned}$$

3.5 Gibbs sampling

Gibbs sampling is a special case of the MH algorithm, where the proposal distributions are the exact conditional distributions of the target that we want to sample from and thus the acceptance probability is always 1. At each step only one of the variables is updated[1] by drawing a value from the distribution $p(q_i|\mathbf{q}_{\setminus i})$ of the variable q_i conditioned on the values of all the remaining variables $\mathbf{q}_{\setminus i}$. This procedure is then repeated by cycling over all the variables. Having a 3 dimensional (this can be easily generalized to D dimensions) vector $\mathbf{q}^s$ at iteration s for example, and reading the symbol $\sim$ as "is sampled from" we repeat until convergence the following:

1. $q_1^{s+1} \sim p(q_1|q_2^s, q_3^s)$
2. $q_2^{s+1} \sim p(q_2|q_1^{s+1}, q_3^s)$
3. $q_3^{s+1} \sim p(q_3|q_1^{s+1}, q_2^{s+1})$

We need of course to be able to sample from the conditional distributions, either directly or using other sampling methods such as rejection sampling or slice sampling (Neal, 2000).
Gibbs sampling can be seen as a special case of the MH algorithm by considering proposal distributions having the form $g_k(\mathbf{q}'|\mathbf{q}) = p(q_k|\mathbf{q}_{\setminus k})$. Of course we will have $\mathbf{q}'_{\setminus k} = \mathbf{q}_{\setminus k}$ as only the kth variable is updated. The acceptance probability is calculated as

$$\alpha(\mathbf{q}', \mathbf{q}) = \min\left\{1, \frac{p(\mathbf{q}')g_k(\mathbf{q}|\mathbf{q}')}{p(\mathbf{q})g_k(\mathbf{q}'|\mathbf{q})}\right\} = \min\left\{1, \frac{p(q'_k|\mathbf{q}'_{\setminus k})p(\mathbf{q}'_{\setminus k})p(q_k|\mathbf{q}'_{\setminus k})}{p(q_k|\mathbf{q}_{\setminus k})p(\mathbf{q}_{\setminus k})p(q'_k|\mathbf{q}_{\setminus k})}\right\} = 1 \ . \tag{3.5}$$

Even if all the steps are always accepted, this does not mean that the Gibbs sampler will perform always well: as for the MH algorithm it suffers from random walk behavior and it mixes poorly when sampling from highly correlated distributions.

3.6 Hamiltonian Monte Carlo

The key to the success of any method based on a Metropolis-Hastings accept/reject scheme is the usage of proposal distributions that allow long distance

[1] Gibbs sampling can be seen as the MCMC analog of the widely used optimization technique known as coordinate descent.

moves that also have a high probability of being accepted. In this section we will see how concepts developed in physics can be used to improve the sampling procedure. We will first introduce these concepts from a purely physical point of view and then show how these can be used when dealing with probabilities.

3.6.1 Hamiltonian mechanics

The classical way to study the motion of a particle with mass m under the action of a system of forces is through Newtonian mechanics: after computing the total force $\mathbf{F}_{tot}$ exerted on the particle, its position $\mathbf{q}$ and momentum $\mathbf{p}$ over time[2] are computed starting from Newton's second law of motion, i.e. $\mathbf{F}_{tot} = m\frac{\mathrm{d}^2\mathbf{q}}{\mathrm{d}t^2}$.

Hamiltonian mechanics on the other hand reformulates Newtonian mechanics in a more general framework: the evolution of the system is uniquely described given a function $H(\mathbf{q}, \mathbf{p})$ called *Hamiltonian* (Neal, 2010). For closed systems, the Hamiltonian is simply the sum of the potential energy $U(\mathbf{q})$ and the kinetic energy $K(\mathbf{p})$, i.e. $H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$, and it represents the total energy of the system. The motion of the particle is then obtained solving Hamilton's equations:

$$\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} = \frac{\partial H}{\partial \mathbf{p}} \tag{3.6}$$

$$\frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} = -\frac{\partial H}{\partial \mathbf{q}} \ . \tag{3.7}$$

The partial derivatives of the Hamiltonian determine therefore how $\mathbf{q}$ and $\mathbf{p}$ change over time. The joint space of position and momentum variables is called *phase space*.
Hamiltonian dynamics has three fundamental properties that will justify its usage in an MCMC setting to sample from the posterior distribution of interest:

1. **Reversibility**. The mapping T_s from the state $(\mathbf{q}(t), \mathbf{p}(t))$ at time t to the state $(\mathbf{q}(t+\varepsilon), \mathbf{p}(t+\varepsilon))$ at time $t+\varepsilon$ is injective, and has an inverse, T_{-s}, obtained negating the time derivatives in (3.6) and (3.7). In other words, if we applied the dynamics to return to the previous point we would follow the same path.

2. **Volume preservation**. According to Liouville's Theorem, Hamiltonian dynamics preserves volume in the phase space $(\mathbf{q}, \mathbf{p})$, or equivalently the determinant of the Jacobian matrix of the transformation is one. If we

[2] In the simplest case $\mathbf{p} = m\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t}$.

apply the mapping to a region R in the phase space with volume V, the image R' of R will also have volume V. This means that the mapping can distort the shape of R but not its volume (if one direction is stretched, then one will be necessarily squashed).

3. **Conservation of the Hamiltonian**. The trajectories in the phase space describe contours of constant total energy (Hamiltonian).

As in the majority of the problems an analytic solution to Hamilton's equations does not exist, it is necessary to discretize time and resort to numerical approximations. The simplest method to approximate the solution to a system of ordinary differential equations (ODE) with a given initial value is *Euler's method*. For the i-th component of position and momentum ($i = 1, \dots D$) it gives

$$\begin{aligned} p_i(t+\varepsilon) &= p_i(t) + \varepsilon \frac{\mathrm{d}p_i}{\mathrm{d}t} = p_i(t) - \varepsilon \frac{\partial U(\mathbf{q}(t))}{\partial q_i} \\ q_i(t+\varepsilon) &= q_i(t) + \varepsilon \frac{\mathrm{d}q_i}{\mathrm{d}t} = q_i(t) + \varepsilon \frac{\partial K(\mathbf{p}(t))}{\partial p_i} \ , \end{aligned}$$

where it was made use of (3.6) and (3.7) and the fact that in the Hamiltonian only the potential energy U depends on $\mathbf{q}$ and only the kinetic energy K depends on $\mathbf{p}$. Iterating this procedure we can compute the position and momentum variables at time $t + 2\varepsilon$, $t + 3\varepsilon$ and so on. This method however returns a poor approximation, as the discretized trajectory tends to diverge to infinity.
Much better results can be obtained with the *leapfrog method*:

$$\begin{aligned} p_i(t+\varepsilon/2) &= p_i(t) - (\varepsilon/2) \frac{\partial U(\mathbf{q}(t))}{\partial q_i} \\ q_i(t+\varepsilon) &= q_i(t) + \varepsilon \frac{\partial K(\mathbf{p}(t+\varepsilon/2))}{\partial p_i} \\ p_i(t+\varepsilon) &= p_i(t+\epsilon/2) - (\varepsilon/2) \frac{\partial U(\mathbf{q}(t+\varepsilon))}{\partial q_i} \ . \end{aligned}$$

After half a step for the momentum variables, we do a full step for the position variables using the new momentum variables, and finally we do the missing half step for the momentum variables using the updated position variables. It is not difficult to prove that despite the approximation due to the discretization the leapfrog method is such that the motion satisfies two of the properties introduced above for Hamiltonian mechanics: it preserves volume exactly and it is reversible by simply negating $\mathbf{p}$, applying the same number of steps again and finally negating $\mathbf{p}$ again. The value of the Hamiltonian is no longer conserved, but it is a good approximation of the true one (Neal (2010) shows that the error is of order $\mathcal{O}(\varepsilon^2)$).

3.6.2 Sampling using Hamiltonian dynamics

In the following we will show how Hamiltonian dynamics can be used to form efficient proposal distributions for a Metropolis-Hastings sampler, that allow large moves in parameter space while keeping a high acceptance rate. Thanks to this approach we will be able to overcome two of the main shortcomings of the original Metropolis-Hastings algorithm, namely the random walk behavior and the poor mixing obtained in case of highly correlated variables. This method can be applied to any distribution over continuous variables for which we can easily compute the gradient of the (unnormalized) log-posterior with respect to the state variables. We see that as for optimization techniques, for sampling as well it is convenient to use this first order information if available.

The main idea behind this algorithm, known as *Hamiltonian Monte Carlo (HMC)* or *Hybrid Monte Carlo* (Neal, 2010; Girolami and Calderhead, 2011), is to define an Hamiltonian function in terms of the target probability distribution, and move a sample from this distribution as if it was a particle in space following the corresponding Hamiltonian dynamics.
To sample a random variable $\mathbf{q} \in \mathbb{R}^{\mathrm{D}}$ from the probability distribution $p(\mathbf{q})$, we first have to define an independent auxiliary variable $\mathbf{p} \in \mathbb{R}^{\mathrm{D}}$, that is given a Gaussian density $p(\mathbf{p}) = \mathcal{N}(\mathbf{p}; \mathbf{0}, \mathbf{M})$. Due to independence, the joint density can be written as $p(\mathbf{q}, \mathbf{p}) = p(\mathbf{q})p(\mathbf{p})$. For notational convenience, we also define the log-probability distribution we want to sample from as $\mathcal{L}(\mathbf{q}) = \log p(\mathbf{q})$. The negative joint log-probability is then

$$H(\mathbf{q}, \mathbf{p}) = -\mathcal{L}(\mathbf{q}) + \frac{1}{2} \log \left((2\pi)^{\mathrm{D}} \det(\mathbf{M}) \right) + \frac{1}{2} \mathbf{p}^T \mathbf{M}^{-1} \mathbf{p}$$

and, ignoring the constant term, it can be interpreted as an Hamiltonian $H(\mathbf{q}, \mathbf{p})$ with potential and kinetic energy respectively

$$\begin{aligned} U(\mathbf{q}) &= -\mathcal{L}(\mathbf{q}) \\ K(\mathbf{p}) &= \frac{1}{2} \mathbf{p}^T \mathbf{M}^{-1} \mathbf{p} \ . \end{aligned}$$

The Hamiltonian represents the total energy of a closed system, in which $\mathbf{q}$ is the position of the particle in space. Thanks to the quadratic kinetic term, the auxiliary variable $\mathbf{p}$ can be seen as a momentum variable, and the covariance matrix $\mathbf{M}$ as a mass matrix.
HMC samples from the joint distribution $p(\mathbf{q}, \mathbf{p})$ using a Gibbs sampling scheme with auxiliary variables $\mathbf{p}$: we are only interested in the samples $\mathbf{q}$, that will be then marginally distributed according to $p(\mathbf{q})$. The Gibbs sampler works as follows: first, given the position and momentum $(\mathbf{q}^n, \mathbf{p}^n)$ at time n, the momentum is updated drawing a sample from the correct conditional distribution

$$\mathbf{p}^{n+1} | \mathbf{q}^n \sim p(\mathbf{p}^{n+1} | \mathbf{q}^n) = p(\mathbf{p}^{n+1}) = \mathcal{N}(\mathbf{p}^{n+1} | \mathbf{0}, \mathbf{M}) \ ,$$

where it was used the independence of $\mathbf{q}$ and $\mathbf{p}$. The variable $\mathbf{q}^{n+1}$ is then sampled from the conditional distribution $p(\mathbf{q}^{n+1}|\mathbf{p}^{n+1})$ with the Metropolis-Hastings algorithm: a deterministic proposal distribution is defined by simulating the behavior of the physical system evolving under Hamiltonian dynamics and with initial position $(\mathbf{q}^n, \mathbf{p}^{n+1})$. Given the Hamiltonian, we can in fact solve Hamilton's equations

$$\begin{aligned}\frac{\mathrm{d}\mathbf{q}}{\mathrm{d}t} &= \frac{\partial H}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p} \\ \frac{\mathrm{d}\mathbf{p}}{\mathrm{d}t} &= -\frac{\partial H}{\partial \mathbf{q}} = \nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q}) \ .\end{aligned}$$

and follow a trajectory to obtain a new pair $(\mathbf{q}^*,\mathbf{p}^*)$, that is accepted with probability

$$\alpha = \min\left(1, \frac{e^{-H(\mathbf{q}^*,\mathbf{p}^*)}}{e^{-H(\mathbf{q}^n,\mathbf{p}^{n+1})}}\right) = \min\left(1, e^{-H(\mathbf{q}^*,\mathbf{p}^*)+H(\mathbf{q}^n,\mathbf{p}^{n+1})}\right) \ .$$

It is worth pointing out that in this case the proposal distribution is not a random walk but is deterministic: it is in fact the solution of a system of nonlinear differential equations. From a probabilistic point of view, this can be justified using for the proposal distribution a Dirac distribution. This deterministic component will propose moves along the isocontours of the energy (Hamiltonian) in the phase space, whereas the random draws of the momentum from the exact conditional distribution will change the energy levels.

Validity of the proposed method

To use the proposals derived from Hamiltonian dynamics in the MCMC scheme we need to make sure that the moving particle is always a sample from the desired target distribution, i.e. that the Gibbs sampler produces an ergodic, time reversible Markov Chain satisfying detailed balance and with stationary marginal density $p(\mathbf{q})$. Fundamental to prove this are the first two properties of Hamiltonian mechanics introduced in section 3.6.1; the third property, despite not being necessary for the correctness of the method, is fundamental for an efficient sampling scheme:

1. **Reversibility**. As Hamiltonian dynamics is reversible, so will be the defined Markov Chain (detailed balance is respected), and this is a sufficient condition for the target distribution to be stationary.

2. **Volume preservation**. Thanks to this property we do not need to take into account any change in volume in the acceptance probability α of the

MH updates. If the volume was changing, on the other hand, we would have to compute the determinant of the Jacobian matrix of the mapping and include it in the acceptance probability (this computation can be rather complex).

3. **Conservation of the Hamiltonian**. If the trajectories in the phase space exactly conserve the Hamiltonian (hence we are moving along isocontours of equal joint density), then the acceptance probability α is always one.

As already said, for practical applications of interest Hamilton's equations do not have an analytic solution[3], and only discretized approximations obtained with numerical methods are available. However, not all the existing numerical integration schemes can be applied to the problem in hand, as we have seen that reversibility and volume preservation are necessary for a correct algorithm. Luckily, as seen in section 3.6.1, these two properties are satisfied by the *leapfrog integrator*: in this case the proposed sample ($\mathbf{q}^*$,$\mathbf{p}^*$) for the MH algorithm is obtained running the leapfrog method starting from $(\mathbf{q}^n, \mathbf{p}^{n+1})$ for L steps of width ε. Due to integration errors, using the leapfrog integrator method the Hamiltonian is not exactly conserved, but it is still close to the true value, giving therefore a high acceptance probability. This error can be also controlled tuning properly (manually or automatically) the step size ε and the number of integration steps L. As the deterministic proposal distribution will point around the same direction for several leapfrog steps, HMC can lead to potentially very large moves while keeping a much higher probability of acceptance with respect to a random walk Metropolis-Hastings.

Tuning of HMC

There are three main parameters that need to be tuned to use HMC, namely the step size ε, the number of leapfrog steps L done at each iteration and the mass matrix $\mathbf{M}$ of the momentum auxiliary variable.
L and ε can be tuned either manually with some pilot runs on a validation set or in an automated way (Hoffman and Gelman, 2014). Accurate tuning is essential:

- If the step size ε is too low than it is not possible to explore completely the whole space unless a really high number of steps are used (with obvious inefficiency). On the other hand, a too big ε can lead to an unstable algorithm that suffers from a low acceptance rate.

[3] For Gaussian target distributions an analytic solution can be found.

- If the number of leapfrog steps L is too small there will be slow mixing due to random-walk behavior, whereas if it is too high it will return to points already seen (double-back behavior).

It is worth noting that an acceptance rate of 1 is not the optimal choice, as this would mean for example that ε could be increased to move even further in the phase space. Neal (2010) shows that the optimal acceptance rate should be around 0.65. One can also consider a number of burn-in samples to avoid highly correlated proposed samples, not too high though for an efficient implementation.
Picking the right $\mathbf{M}$ is essential for a performing algorithm, as its diagonal terms have to reflect the scale of the sampled momentum variables and the off-diagonal terms their correlation: a simple default choice using a (possibly scaled) identity matrix will give poor results for highly correlated variables. Apart from using some heuristics that rely on the knowledge of the marginal variance of the target density obtained with several pilot runs (Neal, 2010), a principled manner to choose $\mathbf{M}$ is still unknown. This is one of the main reasons why despite all the improvements it can give with respect to Metropolis-Hastings, Hamiltonian Monte Carlo is still not widely used (Girolami and Calderhead, 2011).

Chapter 4

Improving Sampling Using Differential Geometry

4.1 Differential geometry

The main focus of this section will be that of introducing some necessary background from *differential geometry*. The deep mathematical formalism required in many of the definitions will be partly neglected, to give space to a more intuitive explanation of the concepts introduced. A more detailed presentation on differential geometry can be found for example in (Amari and Nagaoka, 2000).

4.1.1 Manifolds

A manifold can be thought intuitively as a multidimensional generalization of a surface. More formally, an n-dimensional *manifold* $\mathcal{S}$ is a set that locally acts like $\mathbb{R}^n$ (locally homeomorphic to the Euclidean space): for each point $\mathbf{q}$ of the manifold there exists a one-to-one (bijective) mapping $\varphi : \mathcal{S} \to \mathbb{R}^n$, called a *coordinate system*, from an open set around $\mathbf{q}$ to an open set of $\mathbb{R}^n$. This means that we can specify any point $\mathbf{q} \in \mathcal{S}$ as an n-dimensional vector called the *coordinate* of the point, that is given by $\varphi(\mathbf{q}) = [\varphi_1(\mathbf{q}), .., \varphi_n(\mathbf{q})]$. We

assume φ to be a smooth function, hence the manifold S is differentiable. In the following we will mostly focus on manifolds that are *embedded* in some higher dimensional Euclidean space $\mathbb{R}^{\mathrm{D}}$. We can therefore define a parametrization of an open neighborhood that includes $\mathbf{q}$ on $\mathcal{S}$ as

$$\begin{aligned} \sigma : \quad & \Phi(\subseteq \mathbb{R}^n) \to \Omega(\subseteq \mathbb{R}^{\mathrm{D}}) \\ & \boldsymbol{\theta} \qquad\quad \mapsto \mathbf{q} \end{aligned}$$

Example
The unit sphere $\mathcal{S}_3$ is a 2-dimensional manifold[1] embedded in $\mathbb{R}^3$, formed by the set of points $\mathbf{q} \in \mathbb{R}^3$ that are the solution of

$$q_1^2 + q_2^2 + q_3^2 = 1 \ .$$

It can be parametrized in spherical coordinates as

$$\begin{aligned} q_1 &= \sigma_1(\theta_1, \theta_2) = \cos\theta_1 \sin\theta_2 \\ q_2 &= \sigma_2(\theta_1, \theta_2) = \sin\theta_1 \sin\theta_2 \\ q_3 &= \sigma_3(\theta_1, \theta_2) = \cos\theta_2 \end{aligned}$$

where $\theta_1 \in [-\pi, \pi)$ and $\theta_2 \in [0, \pi]$. Note that a sphere is a manifold as any of its points can be locally (in a neighborhood small enough) seen as a plane in $\mathbb{R}^2$. This example will be extended in the rest of the section and represents a simple and intuitive way to grasp the new concepts introduced.

4.1.2 Tangent spaces

Informally, a *tangent* at a point $\mathbf{q} \in \mathcal{S}$ is a vector that lies "flat" on the manifold. It can be more precisely defined starting from the notion of *curve* in the manifold passing in $\mathbf{q}$, that is a smooth map

$$\begin{aligned} \gamma : (-\epsilon, \epsilon) \in \mathbb{R} &\to \mathcal{S} \\ t &\mapsto \gamma(t) \end{aligned}$$

with $\gamma(0) = \mathbf{q} \in \mathcal{S}$.
In the case of manifolds embedded in $\mathbb{R}^{\mathrm{D}}$, the *tangent vector* to a curve in a manifold can be defined as (Marriott and Salmon, 2000):

$$\dot{\gamma}(0) = \frac{\mathrm{d}}{\mathrm{d}t}\gamma(t)|_{t=0} = \lim_{h \to 0} \frac{\gamma(h) - \gamma(0)}{h} \ .$$

[1] In the rest of the thesis we will deal with $(\mathrm{D}-1)$-dimensional unit spheres $\mathcal{S}_{\mathrm{D}}$. For simplicity of calculations, in this example we will however restrict to the case $\mathrm{D} = 3$.

The tangent vector will be the best linear approximation in $\mathbf{q}$ to the curve defined on the manifold. Due to the embedding, we have $\dot{\gamma}(0) \in \mathbb{R}^{\mathrm{D}}$.
Knowing what a tangent vector is, we can now define the tangent space at $\mathbf{q}$, denoted as $T\mathcal{S}_{\mathbf{q}}$, as the set of all tangent vectors to curves through $\mathbf{q}$. To compute a basis for $T\mathcal{S}_{\mathbf{q}}$ we use the above introduced parametrization σ, that allows us to write all the curves γ on $\mathcal{S}$ as a composite function

$$\sigma \circ \gamma_{\boldsymbol{\theta}} : (-\epsilon, \epsilon) \to \Phi \to \Omega \ ,$$

where

$$\begin{aligned} \gamma_{\boldsymbol{\theta}} : (-\tilde{\epsilon}, \tilde{\epsilon}) \in \mathbb{R} &\to \Phi \\ t &\mapsto \gamma_{\boldsymbol{\theta}}(t) \end{aligned}$$

is a curve defined on the parameter space Φ. Curves on $\mathcal{S}$ can therefore be formed first defining a curve on the parameter space Φ and then mapping it using σ to the open subset Ω of the manifold $\mathcal{S}$.

Deriving $\sigma \circ \gamma_{\boldsymbol{\theta}}$ with respect to time, we see that any tangent vector will therefore have the form

$$\sum_{i=1}^{n} \frac{\partial \sigma}{\partial \theta_i} \frac{\mathrm{d}\theta_i}{\mathrm{d}t}$$

and a basis for $T\mathcal{S}_{\mathbf{q}}$ will be

$$\left\{ \frac{\partial \sigma}{\partial \theta_i}, i = 1, \ldots n \right\} \ . \tag{4.1}$$

$T\mathcal{S}_{\mathbf{q}}$ is hence a n dimensional subspace of $\mathbb{R}^{\mathrm{D}}$, and it is possible to prove that it is independent of the parametrization used (Amari and Nagaoka, 2000).

Example (continued)
The basis of the tangent space for the unit sphere discussed above is given by

$$\begin{aligned} \frac{\partial \sigma}{\partial \theta_1} &= (-\sin\theta_1 \sin\theta_2, \cos\theta_1 \sin\theta_2, 0) \in \mathbb{R}^3 \\ \frac{\partial \sigma}{\partial \theta_2} &= (\cos\theta_1 \cos\theta_2, \sin\theta_1 \cos\theta_2, -\sin\theta_2) \in \mathbb{R}^3 \end{aligned}$$

4.1.3 Metric tensors and Riemmanian manifolds

The addition of a metric tensor to a manifold $\mathcal{S}$ allows us to define, in the tangent space $T\mathcal{S}_{\mathbf{q}}$ of a point $\mathbf{q} \in \mathcal{S}$, lengths of tangent vectors and angles between them,

in a way that is invariant to a reparametrization of the coordinate system. It can be seen as a generalization of the dot product of vectors in the Euclidean space. As we will see, this will turn out to be the building block in the definition of other geometric structures such as the length of a path in the manifold and the geodesics.
We assume that an inner product (i.e. a metric) has been defined on the tangent space $T\mathcal{S}_{\mathbf{q}}$ at each point $\mathbf{q} \in \mathcal{S}$. The inner product is a function

$$\langle\ ,\ \rangle_{\mathbf{q}} : T\mathcal{S}_{\mathbf{q}} \times T\mathcal{S}_{\mathbf{q}} \to \mathbb{R}$$

that is linear, symmetric and positive-definite.
A *metric tensor* g on $\mathcal{S}$ assigns to each point $\mathbf{q} \in \mathcal{S}$ an inner product: $g : \mathbf{q} \mapsto \langle\ ,\ \rangle_{\mathbf{q}}$. It is also required that the mapping varies smoothly across the manifold with $\mathbf{q}$. Finally, we note that it is possible to consider an infinite number of metric tensors for $\mathcal{S}$, in other words the structure of g is not naturally derived from $\mathcal{S}$. If a manifold $\mathcal{S}$ is equipped with a metric tensor g, we define the pair $(\mathcal{S}, g)$ as a *Riemannian manifold*.
If we now introduce a basis for the tangent space such as the one in (4.1) (that was derived after introducing a parametrization $\boldsymbol{\theta}$ of the manifold) we can calculate the components of the inner product for that particular basis:

$$g_{ij} = \left\langle \frac{\partial \sigma}{\partial \theta_i}, \frac{\partial \sigma}{\partial \theta_j} \right\rangle_{\mathbf{q}} \qquad i, j = 1, \ldots n\ .$$

As using coordinate vectors any vector $\mathbf{X}, \mathbf{Y} \in T\mathcal{S}_{\mathbf{q}}$ of the tangent space can be expressed in terms of the basis, i.e.

$$\mathbf{X} = \sum_{i=1}^{n} X_i \frac{\partial \sigma}{\partial \theta_i}, \qquad \text{and} \qquad \mathbf{Y} = \sum_{j=1}^{n} Y_j \frac{\partial \sigma}{\partial \theta_j}\ ,$$

then the value of the inner product can be simply expressed as

$$\langle \mathbf{X}, \mathbf{Y} \rangle_{\mathbf{q}} = \sum_{i=1}^{n} \sum_{j=1}^{n} g_{ij} X_i Y_j\ .$$

We can now therefore define the length of a vector $\mathbf{X}$ tangent to the manifold in $\mathbf{q}$ as $||\mathbf{X}|| = \sqrt{\langle \mathbf{X}, \mathbf{X} \rangle_{\mathbf{q}}}$.
The calculation above can be also expressed in matrix form, defining G as the $n \times n$ matrix whose (i, j)-th element is given by g_{ij}:

$$\langle \mathbf{X}, \mathbf{Y} \rangle_{\mathbf{q}} = [X_1, \ldots X_n] G [Y_1, \ldots Y_n]^T\ .$$

Note that, due to the properties of the inner product, G will be a positive definite symmetric matrix. Conversely, we can also proceed the other way round: given

a parametrization $\boldsymbol{\theta}$ and a $n \times n$ symmetric positive definite matrix G, then the metric tensor on $\mathcal{S}$ that has g_{ij} as components is uniquely determined. The metric tensors $\tilde{G}$ relative to a different coordinate systems can be found starting from G using some simple differential relations involving the transformation, see (Amari and Nagaoka, 2000) for more details.

Example (continued)
The metric tensor for the sphere $\mathcal{S}_3$ is easily found starting from the basis of the tangent space already computed. Assuming the standard Euclidean dot product in $\mathbb{R}^3$ as the inner product for the tangent space we have:

$$
\begin{aligned}
g_{11} &= \left\langle \frac{\partial \sigma}{\partial \theta_1}, \frac{\partial \sigma}{\partial \theta_1} \right\rangle_{\mathbf{q}} = \sin^2 \theta_2 \\
g_{12} &= \left\langle \frac{\partial \sigma}{\partial \theta_1}, \frac{\partial \sigma}{\partial \theta_2} \right\rangle_{\mathbf{q}} = 0 \\
g_{21} &= \left\langle \frac{\partial \sigma}{\partial \theta_2}, \frac{\partial \sigma}{\partial \theta_1} \right\rangle_{\mathbf{q}} = g_{12} = 0 \\
g_{22} &= \left\langle \frac{\partial \sigma}{\partial \theta_2}, \frac{\partial \sigma}{\partial \theta_2} \right\rangle_{\mathbf{q}} = 1 \ .
\end{aligned}
$$

The metric tensor for this Riemannian manifold and with the above defined parametrization is therefore

$$
G = \begin{bmatrix} \sin^2 \theta_2 & 0 \\ 0 & 1 \end{bmatrix} \ .
$$

4.1.4 Geodesics

Having defined the metric tensor, the length of a curve $\gamma : [a, b] \to \mathcal{S}$ can be calculated as

$$
||\gamma|| = \int_a^b \left|\left| \frac{\mathrm{d}\gamma}{\mathrm{d}t} \right|\right| \mathrm{d}t \ .
$$

When dealing with manifolds the apparently simple task of finding the shortest path between two points is rather complex. The main issue is that when considering points that are far from each other, one has to take into account that their tangent spaces will be defined by two different sets of basis, that will also lead to different formulations of local geometries. The minimization of the length of the curves connecting the two points requires therefore to define a way to compare local geometries at two different points, giving rise to the notions of

(affine) connections, *parallel transport* and *covariant derivatives*.[2]
In the 3-D world we live in, the shortest path between two points, called *geodesic*, is a straight line, where by *straightness* of the line we mean that the vectors tangent to it are always parallel to some fixed direction. When dealing with Riemannian manifolds a geodesic is defined to be a curve whose tangent vectors remain parallel if they are transported along it (a rigorous definition requires the concept of *affine connection*). If the connection used is the *Levi-Civita connection* induced by the metric tensor and using the *Christoffel symbols*, then the geodesic will also be the (locally) shortest path between the two points. To find a geodesic, it is necessary to solve the system of differential equations

$$\frac{\mathrm{d}^2\theta_i}{\mathrm{d}t^2} + \sum_{k,l} \Gamma^i_{kl} \frac{\mathrm{d}\theta_k}{\mathrm{d}t} \frac{\mathrm{d}\theta_l}{\mathrm{d}t} = 0$$

where the Christoffel symbols Γ^i_{kl} depend on the metric tensor and the parametrization:

$$\Gamma^i_{kl} = \frac{1}{2} \sum_m g_{im} \left(\frac{\partial g_{mk}}{\partial \theta_l} + \frac{\partial g_{ml}}{\partial \theta_k} - \frac{\partial g_{kl}}{\partial \theta_m} \right) .$$

The solution of this system will return a curve $\gamma_{\boldsymbol{\theta}}$ on the parameter space, that if mapped with the function σ will return the geodesic on $\mathcal{S}$.
From a physical point of view geodesics can be interpreted as the motion of a free particle in a manifold: as there are no external forces the path followed by the particle is only determined by the bending of the surface, and the only energy acting on it is the kinetic one.

Example (continued)
Solving the system above, it is possible to prove (Byrne and Girolami, 2013) that the geodesics for the sphere are the *great circles*, i.e. the intersection of the sphere and any plane passing through the center of the sphere. Given an initial position $\mathbf{s}(0) \in \mathbb{R}^{\mathrm{D}}$ and an initial velocity $\dot{\mathbf{s}}(0)$ in the tangent space (orthogonal to $\mathbf{s}(0)$), a great circle is given by

$$\mathbf{s}(t) = \mathbf{s}(0)\cos(\alpha t) + \frac{\dot{\mathbf{s}}(0)}{\alpha}\sin(\alpha t) ,$$

where $\alpha = ||\dot{\mathbf{s}}(0)||$.

[2] A detailed presentation of these topics is out of the scope of this thesis, the interested reader is referred to (Amari and Nagaoka, 2000).

4.2 Information Geometry

In the following we will see how concepts from differential geometry can be extended to probability distributions, defining the field of *information geometry*. These concepts will be then used in sections 4.3 and 4.4 to improve the Hamiltonian Monte Carlo sampler introduced in section 3.6.
Let us assume that we are given a parametrized density $p(\boldsymbol{y}|\mathbf{q})$, with $\mathbf{q}$ an $n \times 1$ parameter vector. For notational simplicity we define the log-likelihood $\mathcal{L}(\mathbf{q}) = \log p(\boldsymbol{y}|\mathbf{q})$. The *score function* is given by $\nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q})$ and intuitively it indicates how sensitively $\mathcal{L}(\mathbf{q})$ depends on its parameters. The *(expected) Fisher Information Matrix* $G(\mathbf{q})$ is defined as the covariance of the score:

$$G(\mathbf{q}) = \mathrm{cov}(\nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q})) \ .$$

Rao (1945) discusses how it is possible to define a notion of distance between two distributions. He defines the distance between the probability density evaluated at $\mathbf{q}$ and at $\mathbf{q} + \mathrm{d}\mathbf{q}$ as

$$\chi^2(\mathrm{d}\mathbf{q}) = \int \frac{[p(\boldsymbol{y};\mathbf{q}+\mathrm{d}\mathbf{q}) - p(\boldsymbol{y};\mathbf{q})]^2}{p(\boldsymbol{y};\mathbf{q})}\mathrm{d}\boldsymbol{y} \ .$$

In particular, he shows that a first order approximation to $\chi^2(\mathrm{d}\mathbf{q})$ is given by

$$\chi^2(\mathrm{d}\mathbf{q}) \approx \mathrm{d}\mathbf{q}^T G(\mathbf{q})\mathrm{d}\mathbf{q} \ .$$

Jeffreys (1948) approximates the KL divergence (Bishop, 2006) in a similar manner:

$$D(\mathbf{q}||\mathrm{d}\mathbf{q}) = \int p(\boldsymbol{y};\mathbf{q}+\mathrm{d}\mathbf{q}) \log \frac{p(\boldsymbol{y};\mathbf{q}+\mathrm{d}\mathbf{q})}{p(\boldsymbol{y};\mathbf{q})}\mathrm{d}\boldsymbol{y} \approx \mathrm{d}\mathbf{q}^T G(\mathbf{q})\mathrm{d}\mathbf{q} \ .$$

As by construction $G(\mathbf{q})$ is a $n \times n$ positive definite symmetric matrix, it can be seen as the metric tensor of a Riemannian manifold on the n-dimensional parameter space of the distributions. Each point $\mathbf{q}$ on the manifold represents a distribution, forming therefore a so called *statistical manifold*. From the quadratic form obtained with a first order expansion of the quantities above we see that the density functions do not reside in an Euclidean space, but rather in a Riemannian manifold with metric tensor $G(\mathbf{q})$ and other geometric characteristics such as invariances, connections and geodesics.

Example
The 1-dimensional normal distribution $\mathcal{N}(\mu, \sigma^2)$ has 2 parameters, namely the mean μ and the variance σ^2: it lies therefore in a 2-dimensional Riemannian manifold given by the metric tensor (the

Fisher Information Matrix)

$$G(\mathbf{q}) = \begin{bmatrix} \sigma^{-2} & 0 \\ 0 & 2\sigma^{-2} \end{bmatrix} ,$$

where $\mathbf{q} = [\mu, \sigma]$. A first order approximation of the distance is given by

$$\mathrm{d}\mathbf{q}^T G(\mathbf{q}) \mathrm{d}\mathbf{q} = \frac{(\mathrm{d}\mu^2 + 2\mathrm{d}\sigma^2)}{\sigma^2} ,$$

and hence the space is hyperbolic. This means for example that the distribution $\mathcal{N}(1, 2)$ is closer to $\mathcal{N}(1, 3)$ than $\mathcal{N}(1, 1)$.

4.3 Riemann manifold Hamiltonian Monte Carlo

The geometric structure given by the metric tensor defined by the Fisher information matrix can be employed in MCMC methods to counteract some of the problems typically encountered. As pointed out in section 3.6, one of the main issues when using Hamiltonian Monte Carlo is the difficult tuning of the mass matrix $\mathbf{M}$, that is extremely important for a good convergence of the sampler. Girolami and Calderhead (2011) show that if the (position dependent) expected Fisher information matrix $G(\mathbf{q})$ is used instead of the fixed mass matrix $\mathbf{M}$ many of the shortcomings of HMC can be addressed. The introduced algorithm is the *Riemann manifold Hamiltonian Monte Carlo*.
As it is shown in (Amari and Nagaoka, 2000), the score $\nabla_{\mathbf{q}}\mathcal{L}(\mathbf{q})$ has zero mean and the (i, j)-th element of $G(\mathbf{q})$ can be written under mild regularity conditions as

$$G_{ij}(\mathbf{q}) = E_{\boldsymbol{y}|\mathbf{q}} \left[\left(\frac{\partial}{\partial q_i} \mathcal{L}(\mathbf{q}) \right) \left(\frac{\partial}{\partial q_j} \mathcal{L}(\mathbf{q}) \right) \right] = -E_{\boldsymbol{y}|\mathbf{q}} \left[\frac{\partial^2}{\partial q_i \partial q_j} \mathcal{L}(\mathbf{q}) \right] .$$

We can therefore interpret what Girolami and Calderhead (2011) do as exploiting second order information in their sampler. This is similar to what happens in optimization techniques, where if possible the Hessian matrix is used to achieve a better convergence (Nocedal and Wright, 2006).

The algorithm is an extension of Hamiltonian Monte Carlo in the case of paths constrained in a manifold. The main difference is that the covariance structure of the auxiliary Gaussian momentum variable is equal to the metric tensor $G(\mathbf{q})$ in the current position $\mathbf{q}$. The joint log density, or Hamiltonian $H(\mathbf{q}, \mathbf{p})$, is defined as before:

$$H(\mathbf{q}, \mathbf{p}) = -\mathcal{L}(\mathbf{q}) + \frac{1}{2} \log \left((2\pi)^D \det(G(\mathbf{q})) \right) + \frac{1}{2} \mathbf{p}^T G(\mathbf{q})^{-1} \mathbf{p} . \tag{4.2}$$

Note that in this case the term that derives from the normalizing constant of the Gaussian distribution depends on $\mathbf{q}$ and is therefore no longer constant over the whole parameter space. This means that we cannot neglect it, but as it is position-dependent it has to be included in the potential energy term. It is easy to prove that if we manage to obtain $(\mathbf{q}, \mathbf{p})$ samples from the joint density, then the set of sampled $\mathbf{q}$ will form an empirical approximation of the desired density. If we marginalize over the auxiliary variables $\mathbf{p}$ we get in fact

$$\int e^{-H(\mathbf{q},\mathbf{p})} \mathrm{d}\mathbf{p} = \frac{e^{\mathcal{L}(\mathbf{q})}}{\sqrt{(2\pi)^D \det(G(\mathbf{q}))}} \int e^{-\frac{1}{2}\mathbf{p}^T G(\mathbf{q})^{-1}\mathbf{p}} \mathrm{d}\mathbf{p} = e^{\mathcal{L}(\mathbf{q})} = p(\mathbf{q}) \ .$$

The same Gibbs sampler introduced for HMC is used in this case as well, hence first the momentum is sampled from the conditional distribution $\mathbf{p}|\mathbf{q}$ and then a new proposal for a Metropolis-Hastings sampler is found following a trajectory obtained solving Hamilton's equations. It turns out that in this case the trajectories are (local) geodesics of the manifold (Girolami and Calderhead, 2011).
As the kinetic term depends on the position as well, in this case the Hamiltonian is no longer separable: the leapfrog integrator therefore cannot be used in the numerical evaluation of the paths, as detailed balance would no longer be respected. To overcome this problem, Girolami and Calderhead (2011) use an extension of the leapfrog integrator which is semiexplicit (i.e. the update equations are defined implicitly and need to be solved with some fixed point iterations) but that satisfies reversibility and volume preservation, giving therefore a correct sampler.

Riemann manifold Hamiltonian Monte Carlo manages to exploit the geometric structure induced on the manifold by the metric tensor $G(\mathbf{q})$. We can now get an insight on why Hamiltonian Monte Carlo may have problems: it in fact implicitly assumes that the metric structure of the parameter space is position independent and given by the fixed metric tensor $\mathbf{M}$. If $\mathbf{M}$ is given by the identity matrix, for example, the whole parameter space is considered as Euclidean.

4.4 Geodesic Monte Carlo

Riemann manifold Hamiltonian Monte Carlo manages to exploit the geometric structure induced on the manifold by any positive definite matrix $G(\mathbf{q})$. This means that we can define other metrics than the one obtained with the expected Fisher information matrix.
Geodesic Monte Carlo (GMC) applies Riemann manifold Hamiltonian Monte Carlo techniques to distributions that are themselves defined over a manifold

embedded in the Euclidean space, using the metric tensor that is implicitly imposed by the constraints (Byrne and Girolami, 2013).

Example
All the points belonging to a von Mises-Fisher distribution are constrained to be on an hypersphere (a detailed introduction to this distribution will be given in section 6.3). As we have seen in the previous sections, when the embedding space is $\mathbb{R}^3$ the metric tensor using spherical coordinates is given by

$$G = \begin{bmatrix} \sin^2\theta_2 & 0 \\ 0 & 1 \end{bmatrix} .$$

When using distributions constrained on an embedded manifold, MCMC techniques are the preferred way to perform inference, as the normalizing constants are usually intractable or very expensive to evaluate and therefore standard methods such as rejection sampling cannot be easily implemented. Byrne and Girolami (2013) show that the same concept of Riemann manifold Hamiltonian Monte Carlo can be applied, and consider therefore the Hamiltonian defined in equation (4.2):

$$H(\mathbf{q},\mathbf{p}) = -\log(p(\mathbf{q})) + \frac{1}{2}\log\left((2\pi)^D \det(G(\mathbf{q}))\right) + \frac{1}{2}\mathbf{p}^T G(\mathbf{q})^{-1}\mathbf{p} .$$

If the target distribution is defined with respect to the Hausdorff measure[3] of the manifold (this will be the case in our recommender system application), then the first two terms of the above Hamiltonian can be combined together:

$$H(\mathbf{q},\mathbf{p}) = -\log(p_{\mathcal{H}}(\mathbf{q})) + \frac{1}{2}\mathbf{p}^T G(\mathbf{q})^{-1}\mathbf{p} ,$$

where the $\mathcal{H}$ in $p_{\mathcal{H}}(\mathbf{q})$ indicates that the Hausdorff measure is being considered. As seen in section 4.3, proposals for a Metropolis-Hastings sampler with a high acceptance rate can be obtained solving Hamilton's equations with a generalized leapfrog integrator. In the case case of distributions defined on a manifold for which the form of the geodesics is known, it is however more convenient to "split the Hamiltonian" before solving the equations (Neal, 2010):

$$H(\mathbf{q},\mathbf{p}) = H^{[1]}(\mathbf{q},\mathbf{p}) + H^{[2]}(\mathbf{q},\mathbf{p}) \tag{4.3}$$

[3]The reference measure in probability theory (e.g. to compute normalizing constants) is obtained with the Lebesgue integral (a generalization of the standard Riemann integral to a broader class of functions). The Lebesgue measure can be further generalized using the Hausdorff measure, particularly useful to measure submanifolds embedded in $\mathbb{R}^D$. A relatively simple relationship between these two measures exists, see (Diaconis et al., 2011) for more details.

where

$$
\begin{aligned}
H^{[1]}(\mathbf{q},\mathbf{p}) &= -\log(p_{\mathcal{H}}(\mathbf{q})) \\
H^{[2]}(\mathbf{q},\mathbf{p}) &= \frac{1}{2}\mathbf{p}^T G(\mathbf{q})^{-1}\mathbf{p} \ .
\end{aligned}
$$

Considering each term in (4.3) as a distinct Hamiltonian we alternately simulate their dynamics for some time step ϵ to obtain new proposals. It is possible to prove that this scheme allows sampling from the desired distribution (Neal, 2010; Byrne and Girolami, 2013).
More specifically, the proposed algorithm is as follows:

1. Sample an initial momentum $\mathbf{p}_0$ from $\mathcal{N}(\mathbf{p}_0; \mathbf{0}, G(\mathbf{q}_0))$.
2. Run L iterations of the following integrator to get a proposal $(\mathbf{q}_T, \mathbf{p}_T)$:
   (a) Solve $H^{[1]}$ for a period of $\epsilon/2$.
   Due to the absence of a kinetic term Hamilton's equations are given by
   $$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q} = 0 \qquad\qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{p} = \nabla_{\mathbf{q}} \log p_{\mathcal{H}}(\mathbf{q}) \ .$$
   The position variable therefore does not change and we get just a linear update of the momentum $\mathbf{p}$:
   $$\mathbf{p} \leftarrow \mathbf{p} + \frac{\epsilon}{2}\nabla_{\mathbf{q}} \log p_{\mathcal{H}}(\mathbf{q}) \ .$$
   (b) Update following for a time interval ϵ the path due to $H^{[2]}$.
   Due to the absence of a potential term the trajectory followed will be a geodesic (see section 4.1.4).
   (c) Update again according to $H^{[1]}$ for a period of $\epsilon/2$.
3. Use $\mathbf{q}_T$ as new value with probability
   $$\min\left[1, e^{-H(\mathbf{q}_T,\mathbf{p}_T)+H(\mathbf{q}_0,\mathbf{p}_0)}\right] \ ,$$
   otherwise return $\mathbf{q}_0$.

If Hamilton's equations can be solved analytically for both $H^{[1]}$ and $H^{[2]}$, the implementation of the proposal mechanism will be then more efficient: there is no longer the need of using the fixed-point iterations of the generalized leapfrog algorithm and to invert the matrix $G(\mathbf{q})$. Also, the above algorithm can be written as well by completely avoiding the computation of the metric tensor, whose calculation can be rather cumbersome. In some of the manifolds of practical interest in fact, it exist an explicit form of the normal space to the tangent space (in a sphere for example, the normal to the tangent space at $\mathbf{q}$ is $\mathbf{q}$ itself). In this case to constrain the momentum to be in the tangent space, it is more convenient to subtract its projection on the normal space, rather that calculating a basis for the tangent space.

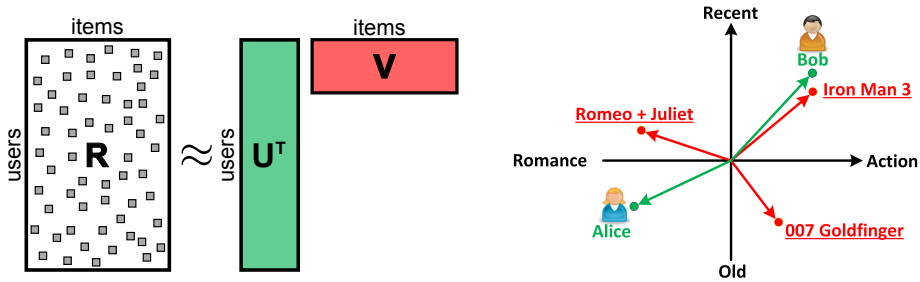
A detailed example of application of the Geodesic Monte Carlo will be given in chapter 7, where it will be used in the training phase of a recommender system.

Chapter 5

Matrix Factorization Techniques for Recommender Systems

5.1 Factorizing the user ratings matrix

Given the past users' history from the training set, it is possible to construct the *user ratings* matrix $\mathbf{R}$, whose element in row i and column j represents the rating that user i has given to item j. If we denote with N the number of users and with M the number of items in the catalog, then the user ratings matrix has dimension $N \times M$. This matrix is usually very sparse, as each user is likely to have rated just a small subset of all items. In the Netlfix data set for example (see section 2.5) just around 100 millions out of 9 billions elements are present (slightly more than the 1%), whereas the rest are missing.
The user ratings matrix can be factorized using two smaller dense matrices $\mathbf{U}$ and $\mathbf{V}$ learned from the available elements of $\mathbf{R}$, i.e. $\mathbf{R} = \mathbf{U}^T\mathbf{V} + \mathbf{E}$, where $\mathbf{E}$ is a residual term (see Figure 5.1a for an illustration). $\mathbf{U}$ and $\mathbf{V}$ are respectively $\mathrm{D} \times N$ and $\mathrm{D} \times M$ matrices, with typical values for the number of factors D of, say, between 5 and 100. Each column of $\mathbf{U}$ is a D-dimensional vector $\mathbf{u}_i$ that represents a single user, analogously the columns $\mathbf{v}_j$ of $\mathbf{V}$ represent the items. The main idea behind this model is therefore that the preferences of a



(a) Matrix factorization. (b) Example of latent features.

Figure 5.1: The user ratings matrix is decomposed as the product of $\mathbf{U}^T$ and V, that provide a latent factor representation for users and items respectively.

user are are determined by a small number of unobserved/latent factors. Having the user and item vectors, with a simple scalar product it is possible to predict new ratings: the rating that user i would give to item j can be predicted as $\hat{r}_{ij} = \mathbf{u}_i^T \mathbf{v}_j$. To recommend the best new item $\hat{j}$ to an user it is therefore sufficient to compute a scalar product for each item and take the maximum rating obtained:

$$\hat{j} = \arg\max_j (\mathbf{u}_i^T \cdot \mathbf{v}_j) \ .$$

Each element of $\mathbf{u}_i$ and $\mathbf{v}_j$ can be seen as a latent feature learned from the available data. In movie recommendations for example, an element could tend to be higher both for action movies and for users that like action movies, giving therefore a positive contribution in the prediction of the rating. Figure 5.1b provides a simplified illustration with two latent features. We see for example that the vector representing Bob is oriented towards "recent" and "action", meaning that he would probably like Iron Man 3, an action movie from 2013. Note that in reality it is often not trivial or even impossible to find the meaning of the latent features obtained with matrix factorization techniques, as a lot of analysis and domain knowledge is required. Nevertheless, this latent feature representation allows high quality recommendations, as proved by its usage in the winning solution of the Netflix prize (Koren et al., 2009).
As already said, $\mathbf{U}$ and $\mathbf{V}$ have to be learned from the available data, and several matrix factorization models with different assumptions and degrees of complexity allow it. We will now introduce in detail and extend two of these models, forming the basis for the recommender system developed in this thesis. As we will use the Netflix data set introduced in section 2.5 as a running example, the "items" in our recommender system will be also denoted as "movies".

5.2 Probabilistic Matrix Factorization

Given the available entries $\mathcal{R}$ in the matrix $\mathbf{R}$, we want to find an estimate for the user and item matrices $\mathbf{U}$ and $\mathbf{V}$, in such a way that $\hat{r}_{ij} = \mathbf{u}_i^T \cdot \mathbf{v}_j$ provides high quality predictions. A standard way to factorize matrices is through Singular Value Decomposition (Berry et al., 1995): in this case however it is not possible to use it, as conventional Singular Value Decomposition is undefined when the the knowledge about the matrix is incomplete, and addressing just the few known entries would lead to overfitting. A common choice to find the factorizing matrices is therefore to use the known elements of the matrix in the minimization of the prediction error (Pilászy et al., 2010)

$$E(\mathbf{U}, \mathbf{V}) = \sum_{(i,j)\in\mathcal{R}} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \lambda_u \sum_{i=1}^{N} ||\mathbf{u}_i||^2 + \lambda_v \sum_{j=1}^{M} ||\mathbf{v}_j||^2 \ , \tag{5.1}$$

where the regularization terms over the squared norm of $\mathbf{u}_i$ and $\mathbf{v}_j$ are introduced to avoid the overfitting caused by the huge number of parameters of the model[1]. Salakhutdinov and Mnih (2008a) provide a probabilistic interpretation of this model, the *Probabilistic Matrix Factorization (PMF)*. They show that minimizing the error in (5.1) is equivalent to finding a MAP solution for the graphical model shown in Figure 5.2. The likelihood of the observed ratings is given by

$$p(\mathbf{R}|\mathbf{U}, \mathbf{V}, \alpha) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[\mathcal{N}(r_{ij}; \mathbf{u}_i^T \mathbf{v}_j, \alpha^{-1})\right]^{I_{ij}} \ , \tag{5.2}$$

where $I_{ij} = 1$ if user i has rated movie j, 0 otherwise. We assume therefore a Gaussian observation noise (with mean $\mathbf{u}_i^T \mathbf{v}_j$ and precision parameter α) and that the the ratings are conditionally independent given the user and movie vectors. We further place spherical Gaussian priors for both user and item vectors, assuming independence among users and among movies:

$$p(\mathbf{U}|\alpha_u) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{u}_i|\mathbf{0}, \alpha_u^{-1}\mathbf{I}), \qquad p(\mathbf{V}|\alpha_v) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{v}_i|\mathbf{0}, \alpha_v^{-1}\mathbf{I}) \ .$$

The equivalence with (5.1) can be proven by maximizing the log-posterior over user and movie vectors:

$$\log p(\mathbf{U}, \mathbf{V}|\mathbf{R}, \alpha, \alpha_u, \alpha_v) \propto \log p(\mathbf{R}|\mathbf{U}, \mathbf{V}, \alpha) + \log p(\mathbf{U}|\alpha_u) + \log p(\mathbf{V}|\alpha_v)$$

and setting $\lambda_u = \frac{\alpha_u}{\alpha}$ and $\lambda_v = \frac{\alpha_v}{\alpha}$.

[1] We have $(N + M) \cdot \mathrm{D}$ parameters: in the Netflix case for example, considering $\mathrm{D} = 20$ latent features we have $(480189 + 17770) \cdot 20 \approx 10$ millions parameters.

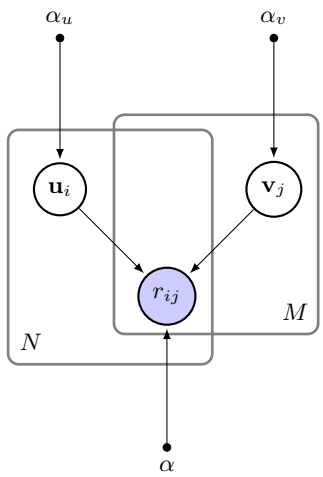


Figure 5.2: Graphical Model for the Probabilistic Matrix Factorization.

A careful tuning of the regularization parameters λ_u and λ_v is required (e.g. using cross-validation) as the model tends to overfit quite easily increasing the number of iterations of the training phase or the number of latent features. Zhou et al. (2008) however show that a robust choice that never (empirically) overfits the data is obtained making the regularization term proportional to the number of ratings that a given user or movie has. More specifically, let us denote with I_i the set of movies rated by user i and with N_i its cardinality, i.e. the number of rating given by user i. Similarly, we use I_j and M_j to indicate the set of users that have rated movie j and its cardinality . The error function to minimize is then given by

$$E(\mathbf{U}, \mathbf{V}) = \sum_{(i,j)\in\mathcal{R}} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 + \sum_{i=1}^{N} \lambda_u N_i ||\mathbf{u}_i||^2 + \sum_{j=1}^{M} \lambda_v M_j ||\mathbf{v}_j||^2 \ . \tag{5.3}$$

Note that in general we may want $\lambda_u \neq \lambda_v$; in practice however the choice $\lambda_u = \lambda_v$ works well and has the advantage that just one parameter has to be found with cross-validation.
To minimize the prediction error (5.3), we first compute the partial derivative of the error with respect to a single element k of the user vector $\mathbf{u}_i$:

$$\frac{\partial E}{\partial u_{i_k}} = 2 \sum_{j\in I_i} -(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j) v_{j_k} + 2\lambda_u N_i u_{i_k} \ . \tag{5.4}$$

We can now either calculate the full gradient and use it in a stochastic gradient descent approach (Salakhutdinov and Mnih, 2008a; Koren, 2009), or use Alternating Least Squares (Bell and Koren, 2007; Zhou et al., 2008). We prefer the

second technique as it gives rise to a very computationally efficient *embarrassingly parallel* problem, as each $\mathbf{u}_i$ can be updated independently of the other user vectors and each $\mathbf{v}_i$ can be updated independently of the other movie vectors.
Alternating Least Squares (ALS) alternates between two steps: in the *U-step* the matrix $\mathbf{V}$ is fixed and $\mathbf{U}$ is recomputed, whereas in the *V-step* we fix $\mathbf{U}$ and recompute $\mathbf{V}$. Fixing one of the two matrices the optimization problem (5.3) becomes quadratic and can be solved optimally. We will now only focus on the U-step, as the V-step is analogous.
We first define $\mathbf{V}_i = \mathbf{V}(:, I_i)$ as the submatrix of $\mathbf{V}$ containing the columns relative to the movies that user i has seen, and $\mathbf{r}_i = \mathbf{R}(i, I_i)$ the corresponding training ratings. Setting (5.4) to zero we obtain

$$\frac{\partial E}{\partial u_{i_k}} = 0 \qquad \Rightarrow \qquad \sum_{j \in I_i} v_{j_k} (\mathbf{u}_i^T \mathbf{v}_j) + \lambda_u N_i u_{i_k} = \sum_{j \in I_i} v_{j_k} r_{ij} \ ,$$

and after some calculations and combining the results for the different elements k, we see that $\mathbf{u}_i$ can be recomputed solving a regularized linear least squares problem:

$$\left(\mathbf{V}_i \mathbf{V}_i^T + \lambda_u N_i \mathbf{I}\right) \mathbf{u}_i = \mathbf{V}_i \mathbf{r}_i^T \ .$$

The new value for $\mathbf{u}_i$ can be therefore robustly found using standard solvers for least squares problems. Note in particular that $\mathbf{u}_i$ can be updated independently from the others user vectors, and that the old value of $\mathbf{u}_i$ is completely ignored, i.e. we are actually performing a recomputation rather than an update. The matrix factorization procedure with ALS can be summarized as follows:

1. Initialize $\mathbf{V}$ using some small random values.
2. For $it = 1, \ldots itMax$
   (a) Recompute the *user vectors* $\mathbf{u}_i$ in parallel $(i = 1 \ldots N)$
   (b) Recompute the *movie vectors* $\mathbf{v}_j$ in parallel $(j = 1 \ldots M)$

Figure 5.3 shows for the Netflix data set with D = 50 latent features how the quiz (validation) error decreases during the training phase. It was used $\lambda_u = \lambda_v = 0.05$, and this value was found using cross-validation on the quiz set. The final quiz RMSE is 0.91033 whereas the test RMSE is slightly higher, 0.91122. Note in particular that due to the big number of parameters the convergence is very fast and that thanks to the used regularizer the model does not overfit.

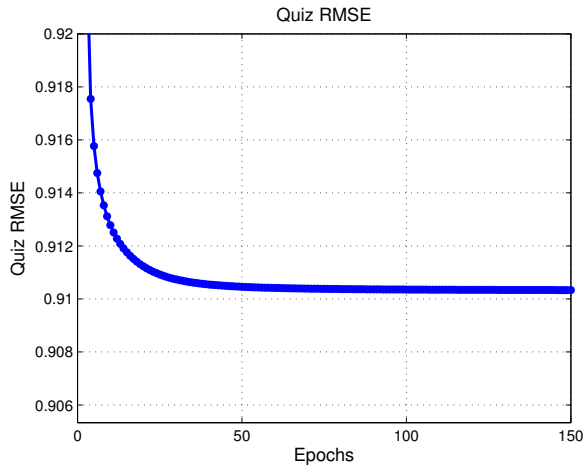


Figure 5.3: RMSE on the quiz set obtained with ALS. After the first iterations in which the RMSE rapidly decreases, it stabilizes without overfitting.

5.2.1 Modelling biases

To further improve the model it is important to consider that some users tend to give systematically higher/lower ratings compared to the average or that some items are much more popular than others and tend therefore to receive higher ratings (Koren, 2009; Paquet and Koenigstein, 2013). To take this into account we decided to extend the presented model introducing for each user a bias parameter c_i, for each movie a bias parameter d_j and finally removing the overall mean rating μ ($\mu = 3.6043$ for the Netflix data set). The predicted rating is then given by $r_{ij} = \mathbf{u}_i^T \mathbf{v}_j + c_i + d_j + \mu$, with the term $\mathbf{u}_i^T \mathbf{v}_j$ that does no longer need to model the biases but can only focus on the true user-item interactions. The error function (5.3) depends now also on the user and movie biases, and can be rewritten as

$$\tilde{E}(\mathbf{U}, \mathbf{V}, \mathbf{c}, \mathbf{d}) = \sum_{(i,j)\in\mathcal{R}} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j - c_i - d_j - \mu)^2 + \sum_{i=1}^{N} \lambda_u N_i ||\mathbf{u}_i||^2 + \\ + \sum_{j=1}^{M} \lambda_v M_j ||\mathbf{v}_j||^2 + \lambda_b \left(\sum_{i=1}^{N} N_i c_i^2 + \sum_{j=1}^{M} M_j d_j^2 \right) ,$$

where the $N \times 1$ vector $\mathbf{c}$ and the $M \times 1$ vector $\mathbf{d}$ contain the introduced biases and are regularized as before using N_i and M_j but with a common parameter λ_b.

The optimization of $\tilde{E}$ is very similar as before: in the U-step we have for example

$$\frac{\partial \tilde{E}}{\partial u_{i_k}} = 0 \qquad \Rightarrow \qquad \sum_{j \in I_i} -(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j - c_i - d_j - \mu) v_{j_k} + \lambda_u N_i u_{i_k} = 0$$

and therefore

$$\left(\mathbf{V}_i \mathbf{V}_i^T + \lambda_u N_i \mathrm{I}\right) \mathbf{u}_i = \mathbf{V}_i [\mathbf{R}_i^T - c_i \mathbf{e} - \mathbf{d}(I_i) - \mu \mathbf{e}] \ ,$$

where $\mathbf{e} = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}^T$.
In a similar way, to optimize the biases we set to zero the partial derivative of the error with respect to the elements of $\mathbf{c}$ and $\mathbf{d}$. Again, as the computations are very similar for user and item biases, we only show the results for $\mathbf{c}$:

$$\frac{\partial \tilde{E}}{\partial c_i} = 0 \qquad \Rightarrow \qquad \sum_{j \in I_i} -(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j - c_i - d_j - \mu) + \lambda_b N_i c_i = 0$$

that leads to

$$c_i = \frac{\sum_{j \in I_i} (r_{ij} - \mathbf{u}_i^T \mathbf{v}_j - d_j - \mu)}{N_i (1 + \lambda_b)} \ .$$

The algorithm is exactly as before with the exception that now after the U-step we also optimize $\mathbf{c}$ and after the V-step we optimize $\mathbf{d}$. A detailed analysis of the results will be presented after having introduced a Bayesian extension of this model, namely the Bayesian Probabilistic Matrix Factorization.

5.3 Bayesian Probabilistic Matrix Factorization

We will now introduce the *Bayesian Probabilistic Matrix Factorization (BPMF)* (Salakhutdinov and Mnih, 2008b), a fully Bayesian treatment of the model described in the previous section. We will therefore no longer look for a point estimate of $\mathbf{U}$ and $\mathbf{V}$ but we will consider them as random variables in all the calculations, making recommendations by averaging over all settings of parameters that are compatible both with the prior and the training data.

The graphical model of the BPMF can be seen in Figure 5.4. We use again the likelihood function (5.2), already introduced for the PMF. Also in this case we place Gaussian priors over the feature vectors, but to increase the predictive power now we consider the means and the covariance matrices as random variables:

$$p(\mathbf{U} | \boldsymbol{\mu}_u, \boldsymbol{\Lambda}_u) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{u}_i; \boldsymbol{\mu}_u, \boldsymbol{\Lambda}_u^{-1}) \ ,$$

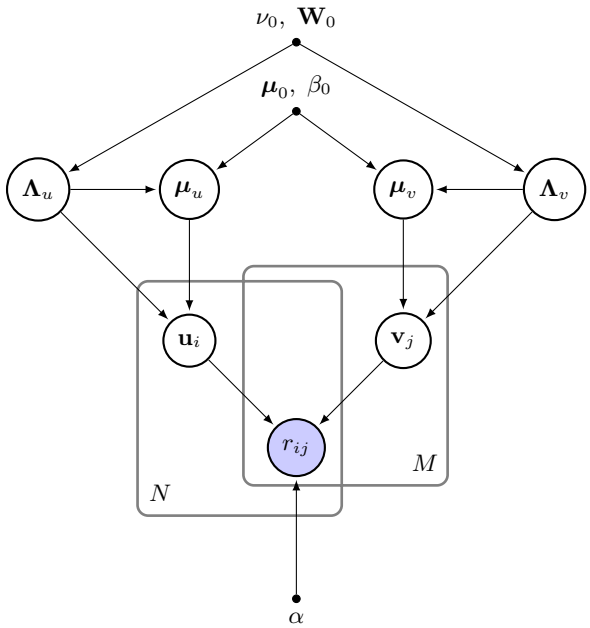


Figure 5.4: Graphical Model for the Bayesian Probabilistic Matrix Factorization.

$$p(\mathbf{V}|\boldsymbol{\mu}_v, \boldsymbol{\Lambda}_v) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{v}_i; \boldsymbol{\mu}_v, \boldsymbol{\Lambda}_v^{-1}) \ .$$

We use Gaussian-Wishart hyperpriors (Bishop, 2006) for the user and movie hyperparameters: the advantage of obtained with these hyperpriors is that they are conjugate priors (in the case in which both the mean and the covariance matrix are unknown) and this leads to analytic convenience. If we denote $\Theta_u = \{\boldsymbol{\mu}_u, \boldsymbol{\Lambda}_u\}$ and $\Theta_v = \{\boldsymbol{\mu}_v, \boldsymbol{\Lambda}_v\}$, we then have

$$\begin{aligned} p(\boldsymbol{\Theta}_u|\Theta_0) &= p(\boldsymbol{\mu}_u, \boldsymbol{\Lambda}_u|\boldsymbol{\mu}_0, \beta_0, \nu_0, \mathbf{W}_0) = p(\boldsymbol{\mu}_u|\boldsymbol{\Lambda}_u, \boldsymbol{\mu}_0, \beta_0)p(\boldsymbol{\Lambda}_u|\nu_0, \mathbf{W}_0) \\ &= \mathcal{N}(\boldsymbol{\mu}_u; \boldsymbol{\mu}_0, (\beta_0\boldsymbol{\Lambda}_u)^{-1})\mathcal{W}(\boldsymbol{\Lambda}_u; \nu_0, \mathbf{W}_0) \end{aligned}$$

$$\begin{aligned} p(\boldsymbol{\Theta}_v|\Theta_0) &= p(\boldsymbol{\mu}_v, \boldsymbol{\Lambda}_v|\boldsymbol{\mu}_0, \beta_0, \nu_0, \mathbf{W}_0) = p(\boldsymbol{\mu}_v|\boldsymbol{\Lambda}_v, \boldsymbol{\mu}_0, \beta_0)p(\boldsymbol{\Lambda}_v|\nu_0, \mathbf{W}_0) \\ &= \mathcal{N}(\boldsymbol{\mu}_v; \boldsymbol{\mu}_0, (\beta_0\boldsymbol{\Lambda}_v)^{-1})\mathcal{W}(\boldsymbol{\Lambda}_v; \nu_0, \mathbf{W}_0) \end{aligned}$$

where $\beta_0 \in \mathbb{R}^+$, $\mathcal{W}$ is a Wishart distribution[2] with ν_0 degrees of freedom, $\mathbf{W}_0$ is the D $\times$ D scale matrix and $\Theta_0 = \{\boldsymbol{\mu}_0, \beta_0, \nu_0, \mathbf{W}_0\}$.
In the experiment it was used in particular $\boldsymbol{\mu}_0 = \mathbf{0}$, $\beta_0 = 2$, $\nu_0 = \mathrm{D}$, $\mathbf{W}_0 = \mathbf{I}$ and $\alpha = 2$.
Using the Bayesian model in Figure 5.4 we are interested in the predictive distribution

$$p(r_{ij}|\mathbf{R}, \boldsymbol{\Theta}_0) = \int\int p(r_{ij}|\mathbf{u}_i, \mathbf{v}_j) p(\mathbf{U}, \mathbf{V}|\mathbf{R}, \boldsymbol{\Theta}_u, \boldsymbol{\Theta}_v) \\ p(\boldsymbol{\Theta}_u, \boldsymbol{\Theta}_v|\boldsymbol{\Theta}_0) \mathrm{d}\{\mathbf{U}, \mathbf{V}\} \mathrm{d}\{\Theta_u, \boldsymbol{\Theta}_v\} \ . \tag{5.5}$$

We therefore integrate out the user and item matrices and the model hyperparameters $\boldsymbol{\Theta}_u$ and $\boldsymbol{\Theta}_v$. Due to the complexity of the posterior this integral is however not analytically tractable. We therefore resort to sampling methods, drawing K samples $\{\mathbf{U}^{(k)}, \mathbf{V}^{(k)}\}$ from the posterior distribution over $\{\mathbf{U}, \mathbf{V}, \boldsymbol{\Theta}_u, \boldsymbol{\Theta}_v\}$ and approximating (5.5) as

$$p(r_{ij}|\mathbf{R}, \boldsymbol{\Theta}_0) \approx \frac{1}{K} \sum_{k=1}^{K} p(r_{ij}|\mathbf{u}_i^{(k)}, \mathbf{v}_j^{(k)}) \ .$$

Similarly, to make predictions for the rating r_{ij} we can use the empirical approximation of the expected rating.

5.3.1 Inference using MCMC methods

The posterior samples are drawn using a Gibbs sampler (see section 3.5), as the necessary conditional distribution will turn out to be rather simple to sample from (due to the use of conjugate priors).
Salakhutdinov and Mnih (2008b) show that the posterior distribution of the movie vectors $\mathbf{v}_j$ given the rest of the variables has a Gaussian distribution

$$p(\mathbf{v}_j|\mathbf{R}, \mathbf{U}, \boldsymbol{\Theta}_v, \alpha) = \mathcal{N}(\mathbf{v}_j; \boldsymbol{\mu}_i^\star, [\boldsymbol{\Lambda}_i^\star]^{-1}) \ , \tag{5.6}$$

where

$$\boldsymbol{\Lambda}_i^\star = \boldsymbol{\Lambda}_v + \alpha \mathbf{U}_j \mathbf{U}_j^T \\ \boldsymbol{\mu}_i^\star = (\boldsymbol{\Lambda}_i^\star)[\boldsymbol{\Lambda}_v \boldsymbol{\mu}_v + \alpha \mathbf{U}_j \mathbf{r}_j] \ .$$

[2] The Wishart distribution with ν_0 degrees of freedom, a D $\times$ D scale matrix $\mathbf{W}_0$ and normalizing constant Z is given by:

$$\mathcal{W}(\boldsymbol{\Lambda}; \nu_0, \mathbf{W}_0) = \frac{1}{Z} (\det \boldsymbol{\Lambda})^{(\nu_0 - \mathrm{D} - 1)/2} \exp\left(-\frac{1}{2} \mathrm{Tr}(\mathbf{W}_0^{-1} \Lambda)\right) \ .$$

$\mathbf{U}_j$ is defined as the submatrix of $\mathbf{U}$ containing the columns relative to the users that have rated movie j, and $\mathbf{r}_j$ contains the corresponding ratings. We note in particular that the updates of the movie vectors are independent from each other, and can be therefore performed really efficiently in parallel.
As in (5.5) we are marginalizing over the hyperparameters of the gaussian prior, we need to sample them as well. The conditional distribution of the movie hyperparameters given the current value of $\mathbf{V}$ (it is independent from the rest of the variables) is a Gaussian-Wishart distribution (Salakhutdinov and Mnih, 2008b)

$$p(\boldsymbol{\mu}_v, \boldsymbol{\Lambda}_v|\mathbf{V}, \boldsymbol{\Theta}_0) = \mathcal{N}(\boldsymbol{\mu}_v; \boldsymbol{\mu}_0^\star, (\beta_0^\star \boldsymbol{\Lambda}_v)^{-1})\mathcal{W}(\boldsymbol{\Lambda}_v; \nu_0^\star, \mathbf{W}_0^\star) \tag{5.7}$$

with

$$\boldsymbol{\mu}_0^\star = \frac{\beta_0 \boldsymbol{\mu}_0 + M\bar{\mathbf{v}}}{\beta_0 + M}, \qquad \beta_0^\star = \beta_0 + M \qquad \nu_0^\star = \nu_0 + M$$

$$[\mathbf{W}_0^\star]^{-1} = \mathbf{W}_0^{-1} + M\bar{\mathbf{S}} + \frac{\beta_0 M}{\beta_0 + M}(\boldsymbol{\mu}_0 - \bar{\mathbf{v}})(\boldsymbol{\mu}_0 - \bar{\mathbf{v}})^T$$

$$\bar{\mathbf{v}} = \frac{1}{M}\sum_{j=1}^{M} \mathbf{v}_j \qquad \bar{\mathbf{S}} = \frac{1}{M}\sum_{j=1}^{M}(\mathbf{v}_j - \bar{\mathbf{v}})(\mathbf{v}_j - \bar{\mathbf{v}})^T$$

The update of the users vectors and hyperparameters is analogous. The necessary posterior samples for the whole model can be therefore sampled with the following scheme:

1. Initialize $\mathbf{U}^1$ and $\mathbf{V}^1$ using for example some small random values or the MAP solution.
2. For $t = 1, \dots T$
   (a) Sample the user and movie hyperparameters form $p(\boldsymbol{\mu}_u, \boldsymbol{\Lambda}_u|\mathbf{U}, \boldsymbol{\Theta}_0)$ and $p(\boldsymbol{\mu}_v, \boldsymbol{\Lambda}_v|\mathbf{V}, \boldsymbol{\Theta}_0)$ respectively.
   (b) Sample the *user vectors* $\mathbf{u}_i$ $(i = 1 \dots N)$ from $p(\mathbf{u}_i|\mathbf{R}, \mathbf{V}, \boldsymbol{\Theta}_u, \alpha)$ in parallel.
   (c) Sample the *movie vectors* $\mathbf{v}_j$ $(j = 1 \dots M)$ from $p(\mathbf{v}_j|\mathbf{R}, \mathbf{U}, \boldsymbol{\Theta}_v, \alpha)$ in parallel.

As typical of MCMC methods, the calculations involved to find the posterior distributions are quite long and tedious. We will now show as an example the approach we used to derive the update of (5.6).

Rather that working directly with $p(\mathbf{v}_j|\mathbf{R}, \mathbf{U}, \mathbf{\Theta}_v, \alpha) = p(\mathbf{v}_j|\mathbf{r}_j, \mathbf{U}_j, \mathbf{\Theta}_v, \alpha)$ it is more convenient to start with the joint distribution of movie vectors and ratings:

$$\begin{aligned} p(\mathbf{v}_j, \mathbf{r}_j|\mathbf{U}_j, \mathbf{\Theta}_v, \alpha) &= p(\mathbf{r}_j|\mathbf{U}, \mathbf{v}_j, \alpha)p(\mathbf{v}_j|\mathbf{\Theta}_v) \\ &= \prod_{i=1}^{N} \left[\mathcal{N}(r_{ij}; \mathbf{u}_i^T\mathbf{v}_j, \alpha^{-1})\right]^{I_{ij}} \mathcal{N}(\mathbf{v}_j; \boldsymbol{\mu}_v, \mathbf{\Lambda}_v^{-1}) \ . \end{aligned}$$

Taking the logarithm of $p(\mathbf{v}_j, \mathbf{r}_j|\mathbf{U}_j, \mathbf{\Theta}_v, \alpha)$ we can more easily combine the Gaussian distributions. In fact, after some calculations and using matrices to avoid the explicit presence of summations we get to:

$$\log p(\mathbf{v}_j, \mathbf{r}_j|\mathbf{U}_j, \mathbf{\Theta}_v, \alpha) \propto -\frac{1}{2}(\mathbf{r}_j - \mathbf{U}_j^T\mathbf{v}_j)^T\alpha\mathbf{I}(\mathbf{r}_j - \mathbf{U}_j^T\mathbf{v}_j) - \frac{1}{2}(\mathbf{v}_j - \boldsymbol{\mu}_v)^T\Lambda_v(\mathbf{v}_j - \boldsymbol{\mu}_v)] \ .$$

Note that the constant terms derived from the normalizers of the Gaussians can be neglected in this phase. We can now exploit the distributive property of matrix multiplication to obtain after some rearrangements of the terms

$$\log p(\mathbf{v}_j, \mathbf{r}_j|\mathbf{U}_j, \mathbf{\Theta}_v, \alpha) \propto -\frac{1}{2}\begin{bmatrix}\mathbf{v}_j \\ \mathbf{r}_j\end{bmatrix}^T \begin{bmatrix}\mathbf{\Lambda}_v + \alpha\mathbf{U}_j\mathbf{U}_j^T & -\alpha\mathbf{U}_j \\ -\alpha\mathbf{U}_j & \alpha\mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{v}_j \\ \mathbf{r}_j\end{bmatrix} + \begin{bmatrix}\mathbf{v}_j \\ \mathbf{r}_j\end{bmatrix}^T \begin{bmatrix}\mathbf{\Lambda}_v\boldsymbol{\mu}_v \\ 0\end{bmatrix} \ .$$

We have therefore shown that the logarithm of the joint distribution $p(\mathbf{v}_j, \mathbf{r}_j|\mathbf{U}_j, \mathbf{\Theta}_v, \alpha)$ is a quadratic form, meaning that the distribution will be a Gaussian: the exponent in a generic Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$ can be in fact written as

$$-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) = -\frac{1}{2}\mathbf{x}^T\mathbf{\Sigma}^{-1}\mathbf{x} + \mathbf{x}^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu} + \text{const} \ ,$$

giving a quadratic form with respect to $\mathbf{x}$. To find the mean and covariance/precision matrix of $p(\mathbf{v}_j, \mathbf{r}_j|\mathbf{U}_j, \mathbf{\Theta}_v, \alpha)$ we hence just need to equate

$$\mathbf{\Lambda} = \mathbf{\Sigma}^{-1} = \begin{bmatrix}\mathbf{\Lambda}_v + \alpha\mathbf{U}_j\mathbf{U}_j^T & -\alpha\mathbf{U}_j \\ -\alpha\mathbf{U}_j & \alpha\mathbf{I}\end{bmatrix}, \qquad \mathbf{\Sigma}^{-1}\boldsymbol{\mu} = \begin{bmatrix}\mathbf{\Lambda}_v\boldsymbol{\mu}_v \\ 0\end{bmatrix} \ .$$

Calculating the analytic form of $\mathbf{\Sigma}$ using the formulas for the inverse of a partitioned matrix (Bishop, 2006) and using it to compute $\boldsymbol{\mu} = \mathbf{\Sigma}\begin{bmatrix}\mathbf{\Lambda}_v\boldsymbol{\mu}_v \\ 0\end{bmatrix}$ we finally get

$$\begin{bmatrix}\mathbf{v}_j \\ \mathbf{r}_j\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\mathbf{v}_j \\ \mathbf{r}_j\end{bmatrix}; \begin{bmatrix}\boldsymbol{\mu}_v \\ \mathbf{U}_j^T\boldsymbol{\mu}_v\end{bmatrix}, \begin{bmatrix}\mathbf{\Lambda}_v^{-1} & \mathbf{\Lambda}_v^{-1}\mathbf{U}_j \\ \mathbf{\Lambda}_v^{-1}\mathbf{U}_j & \frac{1}{\alpha}\mathbf{I} + \mathbf{U}_j^T\mathbf{\Lambda}_v^{-1}\mathbf{U}_j\end{bmatrix}\right) \ .$$

Note in particular that as the explicit form of the normalizing constant of a Gaussian distribution is known given the the covariance matrix, we can neglect the constant terms that we find in the calculations (as we did above) and still get correct results.

Having obtained a Gaussian joint distribution, we can then easily derive (5.6) using the well known formulas for the conditional distributions of a partitioned Gaussian (Murphy, 2012).

5.3.2 Modelling biases

As done in section 5.2.1, also for the Bayesian Probabilistic Matrix Factorization we decided to extend the model by introducing biases (see Figure 5.5).
The likelihood function is in this case

$$p(\mathbf{R}|\mathbf{U},\mathbf{V},\alpha) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(r_{ij};\mathbf{u}_i^T\mathbf{v}_j + c_i + d_j + \mu, \alpha^{-1})\right]^{I_{ij}} ,$$

hence the form of (5.6) will change. Defining $\mathbf{c}_j$ as the vector containing the biases relative to the users that have rated movie j, with calculations similar as before we obtain

$$\mathbf{v}_j \sim p(\mathbf{v}_j|\mathbf{R},\mathbf{U},\mathbf{\Theta}_v,\mathbf{c},\mathbf{d},\alpha) = \mathcal{N}(\mathbf{v}_j;\boldsymbol{\mu}^*,(\mathbf{\Lambda}^*)^{-1})$$

with

$$\begin{aligned}\boldsymbol{\mu}^* &= (\mathbf{\Lambda}^*)^{-1}[\mathbf{\Lambda}_v\boldsymbol{\mu}_v + \alpha\mathbf{U}_j(\mathbf{r}_j - \mathbf{c}_j - d_j\mathbf{e} - \mu\mathbf{e})]\\ \mathbf{\Lambda}^* &= \mathbf{\Lambda}_v + \alpha\mathbf{U}_j\mathbf{U}_j^T\end{aligned}$$

We also need in this case to sample user and movie biases from their posterior conditioned on all the other variables. We therefore modelled them as Gaussian distributions, i.e.

$$c_i \sim \mathcal{N}(c_i;\mu_c,\sigma_c^2), \qquad d_j \sim \mathcal{N}(d_j;\mu_d,\sigma_d^2) .$$

This choice is partly justified by inspecting the histogram of the biases obtained with the MAP solution presented in section 5.2.1 (see Figure 5.6), and partly due to analytic convenience.
The joint distribution of movie biases and ratings is given by

$$p(d_j,\mathbf{r}_j|\mathbf{U}_j,\mathbf{V},\mathbf{c},\mathbf{\Theta}_d,\alpha) = \prod_{i=1}^{N}[\mathcal{N}(r_{ij};\mathbf{u}_i^T\mathbf{v}_j + c_i + d_j + \mu,\alpha^{-1})]^{\mathcal{I}_{ij}}\mathcal{N}(d_j;\mu_d,\sigma_d^2)$$

and after long calculations similar to the ones shown before we finally obtain

$$d_j \sim p(d_j|\mathbf{R},\mathbf{U},\mathbf{V},\mathbf{c},\mathbf{\Theta}_d,\alpha) = \mathcal{N}(d_j;\mu_{d|rest},\sigma_{d|rest}^2) ,$$

where

$$\begin{aligned}\mu_{d|rest} &= \mu_d + \frac{\sigma_d^2\alpha}{1+\sigma_d^2 M_j\alpha}[\mathbf{e}^T\mathbf{r}_j - M_j\mu_d - \mathbf{e}^T\mathbf{U}_j^T\mathbf{v}_j - \mathbf{e}^T\mathbf{c}_j - M_j\mu]\\ \sigma_{d|rest}^2 &= \frac{\sigma_d^2}{1+\sigma_d^2 M_j\alpha} .\end{aligned}$$

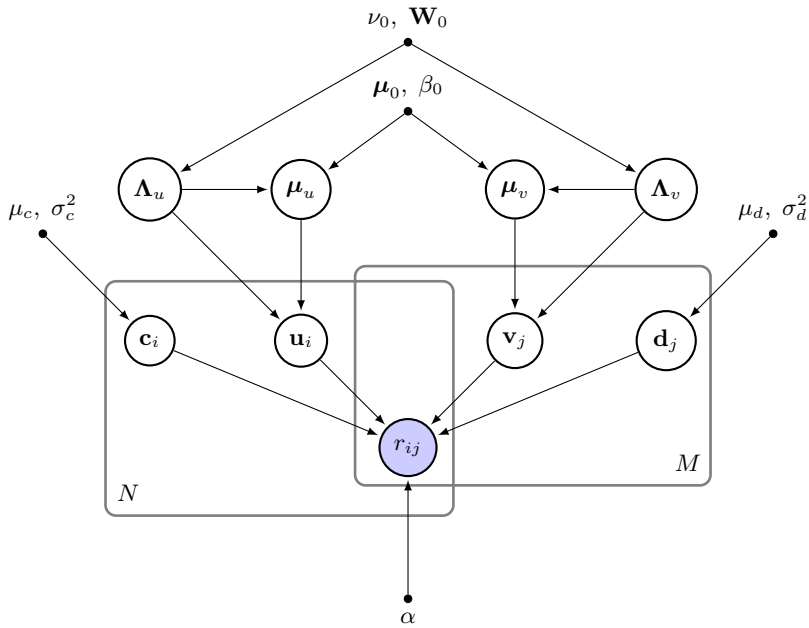


Figure 5.5: Graphical Model for the Bayesian Probabilistic Matrix Factorization with biases taken into account.

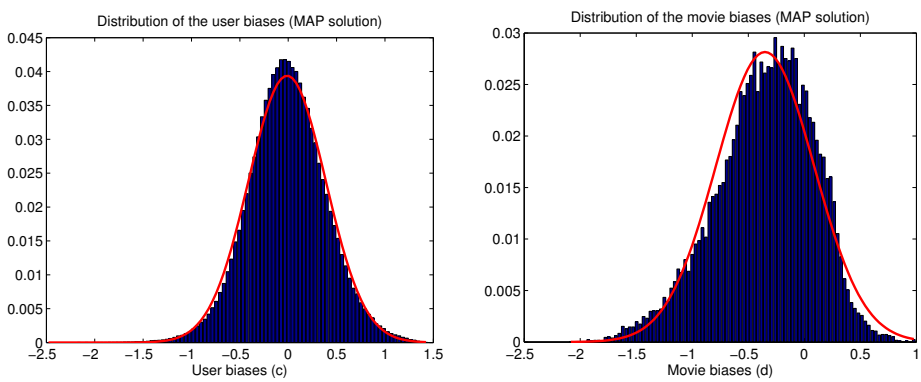


Figure 5.6: Histogram of the biases for the MAP solution. A Gaussian distribution is fitted to it (ML estimates of mean and variance).

5.4 Comparison of the results

We evaluate the performance of the recommender systems in terms of *Root Mean Squared Error* (RMSE), that is defined over the N_{test} test objects as:

$$RMSE = \sqrt{\frac{1}{N_{test}} \sum_{n=1}^{N_{test}} (r_n - \hat{r}_n)^2} \ ,$$

where r_n is the true test rating and $\hat{r}_n$ the predicted one. The RMSEs obtained with PMF and BPMF with different numbers of latent features and both with and without biases are shown in Table 5.1.

RMSE for the test set				
D	**PMF**	**PMF bias**	**BPMF**	**BPMF bias**
8	0.9333	0.9248	0.9198	0.9173
20	0.9275	0.9162	0.9070	0.9065
50	0.9182	0.9113	0.8992	0.8994
100	0.9143	0.9098	0.8955	0.8957

Table 5.1: Test RMSE for different values of D

Wee notice that:

- BPMF performs much better than PMF, as in the first case we are considering all the possible choices of the user and item matrices weighted by their prior probability, and in the second one just a single value.
- The addition of biases in the model improves a lot the recommendations for the PMF model. In the BPMF however the error slightly changes, and it mainly decreases in lower dimensions.
- As expected, increasing D improves the results: we have in fact more modeling power (the computational time however also increases).

In Figure 5.7 we show the RMSEs calculated separately for groups of users with a similar number of ratings, as shown in section 2.5. It is also plotted the RMSE that the naive algorithm of assigning to each movie its average rate would give (movie average algorithm). We notice in particular that the less training data a user has, the harder is to predict their ratings. The BPMF preforms much better than the PMF for groups with a small number of ratings, whereas the performance for the other groups is not too different.

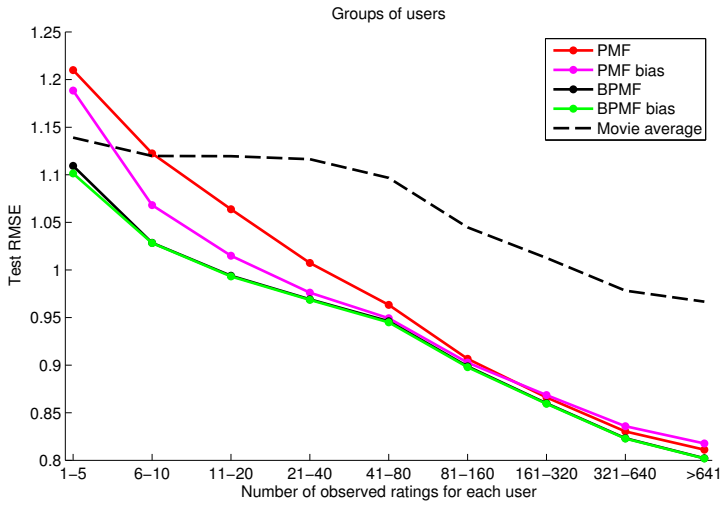


Figure 5.7: Test RMSE for the different groups of similar users.

We can gain more insight on the role of the biases looking at Figure 5.8. We trained different models of increasing complexity, starting from the ones that only used biases (i.e. D = 0) and progressively adding new terms. As we can see, the more complex models allow to improve the recommendations in terms of RMSE. Also, the user-item interaction term $\mathbf{u}_i^T \mathbf{v}_j$ is fundamental to obtain high-quality personalized recommendation.

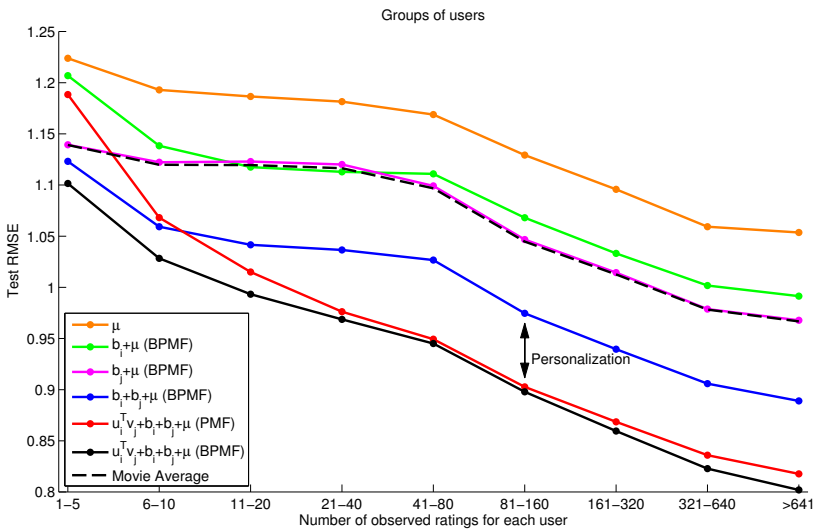


Figure 5.8: Importance of the biases and personalization given by the user-item interaction term.

Chapter 6

Constraining Models to Improve Recommendations

6.1 Introduction

Performing recommendations quickly is crucial in any recommender system: users do not like to wait and tend to abandon after a bit of latency. Considering also the other operations needed in the whole system pipeline, in practice one has to do recommendations in as few milliseconds as possible.
This time constraint limits the quality of the suggestions achievable in large-scale applications, as it is impossible to analyze efficiently the whole catalog. Recommender systems require in fact linear retrieval time to suggest with no approximations a new item to an user. Having M items, to predict for example the best movie for user i with a recommender system based on matrix factorization techniques, we have to find the maximum of M inner products

$$\hat{j} = \arg\max_{j}(\mathbf{u}_i^T \cdot \mathbf{v}_j) \ , \tag{6.1}$$

leading therefore to $\mathcal{O}(M)$ processing time. This linear retrieval time is one of the main bottlenecks in large-scale applications: to overcome this issue some rather strong approximations that limit the number of items inspected are necessary, leading therefore to worse suggestions. In music recommendations for

example the catalog may contain several millions of items, way too many to be analyzed efficiently; one is therefore forced to focus the analysis for example just on the genres the user listens to the most. *Discovery* is however key to a good user experience and one important reason why people use recommender systems (see the discussion about the long tail phenomenon in section 2.2). Hence, we would like to be able to use the whole catalog and suggest songs that are very different from the ones the user has previously listened to, but that we are quite certain he/she will like.
Note finally that it is not possible to pre-compute the recommendations, as it is a costly operation and in large scale applications there is a continuous flow of new data, meaning that the systems need to be retrained several times every day to keep improving the quality of the recommendations.

6.2 Recommendations as a distance minimization problem

Constraining all the item vectors to have fixed norm we can transform the necessary maximization over inner products in a minimization over Euclidean distances:

$$\begin{aligned}
\hat{j} &= \arg\max_j (\mathbf{u}_i^T \cdot \mathbf{v}_j) \\
&= \arg\max_j \left(\frac{1}{2} \left(||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2 - ||\mathbf{u}_i - \mathbf{v}_j||^2 \right) \right) \\
&\overset{(*)}{=} \arg\max_j \left(-||\mathbf{u}_i - \mathbf{v}_j||^2 \right) \\
&= \arg\min_j ||\mathbf{u}_i - \mathbf{v}_j|| \ .
\end{aligned}$$

The step denoted with $(*)$ is justified by the imposed constraint of fixed $||\mathbf{v}_i||$ and the fact that when doing recommendations to a particular user $||\mathbf{u}_i||$ is fixed as well. The best item can therefore be suggested looking among the movie vectors for the *nearest neighbor* to the user vector $\mathbf{u}_i$.
Nearest neighbor search is a fundamental problem in many fields (such as machine learning and computer vision) and several data structures have been developed to perform it efficiently. A detailed analysis of these topics will be presented in Chapter 8, for now it is sufficient to know that using them the $\mathcal{O}(M)$ recommendation step can be performed much more efficiently from the computational point of view, $\mathcal{O}(\log M)$ in the ideal case.
From a probabilistic point of view, the constraints can be imposed extending the recommender systems presented in Chapter 5 with a fixed-norm prior over the item vectors. The main issue one has to face when dealing with unconventional

priors for recommender systems is the tractability of the inference procedure, that cannot be as efficient as when using for example conjugate exponential models. It is therefore critical the development of an algorithm efficient from a computational point of view and that allows to keep the quality of the predictions almost unchanged, as it would be worthless to speedup the retrieval process if the recommendations were much worse. The Von Mises-Fisher distribution, presented next, is a possible choice for the prior, that leads in particular to a very efficient implementation of the training phase of the recommender system.

6.3 The Von Mises-Fisher Distribution

The Von Mises-Fisher (vMF) distribution is the most important probability density in directional statistics (Dhillon and Sra, 2003). Given a mean direction $\boldsymbol{\mu}$ and a concentration parameter κ, the Von Mises-Fisher distribution $vMF(\mathbf{x}; \boldsymbol{\mu}, \kappa)$ is defined as follows:

$$p(\mathbf{x}; \boldsymbol{\mu}, \kappa) = C_{\mathrm{D}}(\kappa)\, e^{\kappa \boldsymbol{\mu}^T \mathbf{x}}\, \mathcal{I}_{\mathcal{S}_D}(\mathbf{x}) \qquad \kappa \geq 0, \quad ||\boldsymbol{\mu}|| = 1\ . \tag{6.2}$$

In (6.2), $\mathcal{S}_{\mathrm{D}}$ represents the D dimensional unit hypersphere (i.e. $\mathcal{S}_{\mathrm{D}} = \{\mathbf{x} \in \mathbb{R}^{\mathrm{D}} | \sqrt{\mathbf{x}^T\mathbf{x}} = 1\}$), $\mathcal{I}_A$ is the indicator function of the set A, and the normalizing constant $C_{\mathrm{D}}(\kappa)$ is given by

$$C_{\mathrm{D}}(\kappa) = \frac{\kappa^{\mathrm{D}/2-1}}{(2\pi)^{\mathrm{D}/2} I_{\mathrm{D}/2-1}(\kappa)},$$

where I_p is the modified Bessel function of the first kind and order p. Higher values of κ increase the concentration of the distribution around the mean direction $\boldsymbol{\mu}$, while $\kappa = 0$ gives an uniform distribution over the the D dimensional unit hypersphere[1].
Note in particular that the normalizing constant is defined with respect to the Hausdorff measure, hence, to use the same notation of section 4.4, it would be more correct to indicate the distribution as $p_{\mathcal{H}}(\mathbf{x}; \boldsymbol{\mu}, \kappa)$. To keep the notation uncluttered in the following we will however remove the $\mathcal{H}$ from the distribution. We can also generalize the distribution to be defined over the hypershere with radius' length q, i.e. $\mathcal{S}_{\mathrm{D}}^{(q)} = \{\mathbf{x} \in \mathbb{R}^{\mathrm{D}} | \sqrt{\mathbf{x}^T\mathbf{x}} = q\}$. The vMF distribution becomes

$$p(\mathbf{x}; \boldsymbol{\mu}, \kappa) = \overline{C}_{\mathrm{D}}(\kappa)\, e^{\kappa \boldsymbol{\mu}^T \mathbf{x}}\, \mathcal{I}_{\mathcal{S}_D^{(q)}}(\mathbf{x}) \qquad \kappa \geq 0, \quad ||\boldsymbol{\mu}|| = q\ , \tag{6.3}$$

[1] The concentration κ has a function similar to the one that the precision parameter (the inverse of the variance) has for the Gaussian distribution.

and it will be denoted as $vMF_q(\mathbf{x}; \boldsymbol{\mu}, \kappa)$. In (6.3) the normalization constant $\overline{C}_{\mathrm{D}}(\kappa)$ is different from the one in (6.2), but its exact value has no interest for us in the following.

6.3.1 Simulating a Von Mises-Fisher Distribution

A widely used algorithm to draw samples from a Von Mises-Fisher Distribution is presented in (Wood, 1994). It is shown that the unit vector $\mathbf{z} = ([1-w^2]^{1/2}\mathbf{s}^T, w)^T$, where $\mathbf{s}$ is uniformly sampled from $\mathcal{S}_{\mathrm{D}-1}$ an the scalar variable $w \in (-1, 1)$ from the density proportional to $(1-w^2)^{(D-3)/2}e^{\kappa w}$ (using rejection sampling), is $vMF(\mathbf{x}; \mathbf{0}, \kappa)$ distributed (Hoff, 2009). Multiplying then the obtained samples by any orthogonal matrix whose last column is $\boldsymbol{\mu}$, one gets samples from a $vMF(\mathbf{x}; \boldsymbol{\mu}, \kappa)$ distribution, and finally if we multiply them also by the scalar q they become $vMF_q(\mathbf{x}; \boldsymbol{\mu}, \kappa)$ distributed.
Figure 6.1 shows an example of this sampling procedure in the unit hypersphere in $\mathbb{R}^3$ for two different values of the concentration parameter κ. It is used the Matlab implementation of Wood's algorithm of Banerjee et al. (2005). As expected, the higher is κ the more concentrated around the mean direction is the distribution.

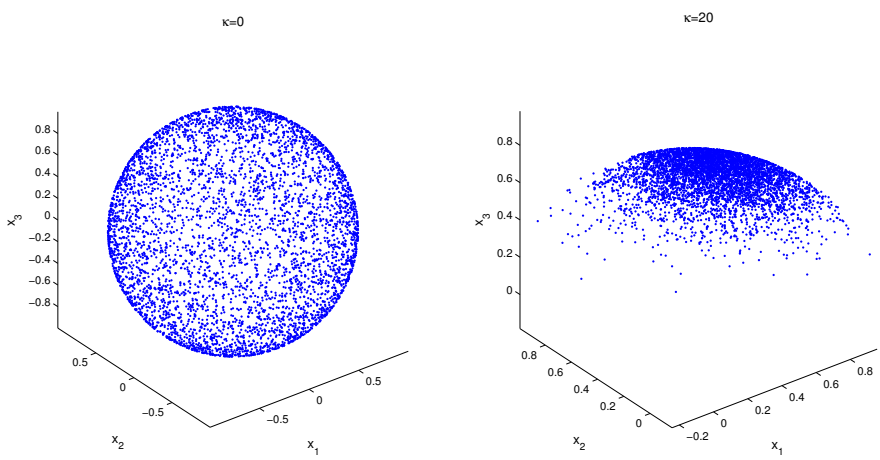


Figure 6.1: Samples drawn from a von Mises-Fisher distribution with $\kappa = 0$ and $\kappa = 20$.

6.4 Modelling biases

As shown in Chapter 5, introducing biases for both users and items in the model can improve the quality of the recommendations. We would like therefore to be able to include them in this constrained framework as well.
Due to time constraints and the complexity of the problem there will be just some first empirical results regarding this problem in the following. For completeness we will now however give hints on what could be the starting point to tackle this in future research. We envision two main options:

1. Similarly to the derivation in section 6.2 we can rewrite the scalar product as follows (when suggesting an item to user i in this case apart from the norm of its vector also its bias is fixed and can therefore be neglected):

$$\begin{aligned}
\hat{j} &= \arg\max_j (\mathbf{u}_i^T \cdot \mathbf{v}_j + c_i + d_j + \mu) \\
&= \arg\max_j \left(\frac{1}{2} \left(||\mathbf{u}_i||^2 + ||\mathbf{v}_j||^2 - ||\mathbf{u}_i - \mathbf{v}_j||^2 \right) + c_i + d_j + \mu \right) \\
&= \arg\max_j \left(\frac{1}{2} \left(||\mathbf{v}_j||^2 - ||\mathbf{u}_i - \mathbf{v}_j||^2 \right) + d_j \right) \\
&= \arg\max_j \left(||\mathbf{v}_j||^2 - ||\mathbf{u}_i - \mathbf{v}_j||^2 + 2d_j \right) \\
&= \arg\min_j \left(||\mathbf{u}_i - \mathbf{v}_j||^2 - (||\mathbf{v}_j||^2 + 2d_j) \right)
\end{aligned}$$

   Constraining the items vector and biases we can again write the problem as a nearest neighbor search: if we now fix the quantity $||\mathbf{v}_j||^2 + 2d_j$ we get in fact

$$\hat{j} = \arg\min_j ||\mathbf{u}_i - \mathbf{v}_j|| \ .$$

   as before. The constraint could be imposed for example using a von Mises-Fisher prior for the joint vector

$$\mathbf{w}_j = \begin{bmatrix} \mathbf{v}_j \\ \sqrt{2d_j} \end{bmatrix} \ .$$

   The main issue with this approach might be that it is no longer possible to specify a prior for the bias only as done in Chapter 5, i.e. we have no direct control on the marginal distribution of d_j and it could be quite far from the reality.

2. In practice a recommender system shows to the user not only a single item, but a list of the best ones. If we were looking for the top 10 items for example, we could find the top 30 ones using the same approach as in section 6.2 (constraining just $||\mathbf{v}_j||$), use the bias term to rearrange them and then taking the resulting top 10 items. In this case however we will have to make sure that these procedure does not lead to poor suggestions due to high biases not considered in the right manner.

Chapter 7

Constrained Bayesian Probabilistic Matrix Factorization

7.1 The model

As seen in section 6.2, efficient recommendations can be achieved constraining the movie vectors to have fixed norm. This can be done using a Von Mises-Fisher distribution as a prior: if use a $vMF_q(\mathbf{v}_j; \boldsymbol{\mu}_v, \kappa)$ of the form

$$p(\mathbf{v}_j|\Theta_v) = \overline{C}_{\mathrm{D}}(\kappa)\, e^{\kappa \boldsymbol{\mu}_v^T \mathbf{v}_j}\, \mathcal{I}_{\mathcal{S}_D^{(q)}}(\mathbf{v}_j), \qquad \Theta_{\mathbf{v}} = \{\boldsymbol{\mu}_v, \kappa\}\ , \tag{7.1}$$

then all the movie vectors will have norm q by construction.
We can therefore follow the same approach as in the Bayesian Probabilistic Matrix Factorization (see section 5.3), in which we use this vMF prior over the movie vectors instead of the Gaussian one. The graphical model for the Constrained Bayesian Probabilistic Matrix Factorization is shown in Figure 7.1. Note in particular that no hyperpriors are used for the von Mises-Fisher distribution: we will in fact impose for now a uniform distribution over the hypersphere with radius q, i.e. all the directions have the same prior probability and $\kappa = 0$. Due to the conditional independence properties of this graphical model, with

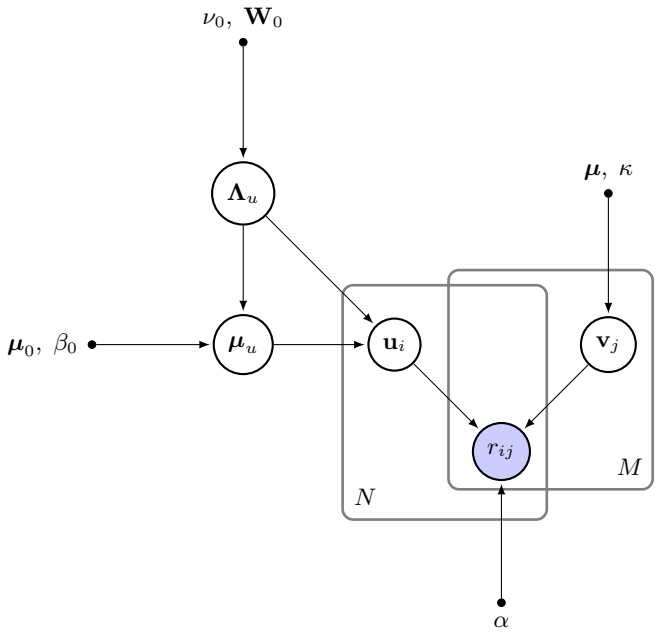


Figure 7.1: Graphical Model for the Constrained Bayesian Probabilistic Matrix Factorization.

respect to the BPMF's Gibbs sampler we only need to change one step, namely the sampling of the movie feature vectors according to the new posterior distribution. We analyze this in detail in the following.

7.2 Posterior distribution over the movie vectors

The joint distribution of movie vectors and observed ratings is given by

$$\begin{aligned} p(\mathbf{v}_j, \mathbf{R}|\mathbf{U}, \Theta_v, \alpha) &= p(\mathbf{R}|\mathbf{U}, \mathbf{v}_j, \alpha) p(\mathbf{v}_j|\Theta_v) \\ &= \prod_{i=1}^{N} \left[\mathcal{N}(r_{ij}; \mathbf{u}_i^T \mathbf{v}_j, \alpha^{-1})\right]^{\mathcal{I}_{ij}} \overline{C}_{\mathrm{D}}(\kappa)\, e^{\kappa \boldsymbol{\mu}_v^T \mathbf{v}_j}\, \mathcal{I}_{\mathcal{S}_D^{(q)}}(\mathbf{v}_j)\ , \end{aligned}$$

where N denotes the number of users and $\mathcal{I}_{ij}$ is the indicator function that is equal to 1 if user i has rated movie j, 0 otherwise. Taking the logarithm in the

above formula we obtain:

$$\log p(\mathbf{v}_j, \mathbf{R}|\mathbf{U}, \Theta_v, \alpha) = \sum_{i=1}^{N} \mathcal{I}_{ij} \left[-\frac{1}{2} \log \frac{2\pi}{\alpha} - \frac{\alpha}{2}(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 \right] + \\ + \log C_{\mathrm{D}}(\kappa) + \kappa \boldsymbol{\mu}_v^T \mathbf{v}_j + \log \left(\mathcal{I}_{\mathcal{S}_D^{(q)}}(\mathbf{v}_j) \right) \ .$$

With some calculations it is possible to write the summation in terms of matrix multiplications: let $\mathcal{I}_j = [\mathcal{I}_{1j}, \mathcal{I}_{2j}, \ldots \mathcal{I}_{Nj}]$ and M_j be the number of ratings for movie j, i.e. $M_j = \sum_{i=1}^{N} \mathcal{I}_{ij}$. If we define again $\mathbf{U}_j \in \mathbb{R}^{D \times M_j}$ as the restriction of the matrix $\mathbf{U}$ to the columns whose corresponding user has rated movie j (in Matlab notation we have $\mathbf{U}_j = \mathbf{U}(:, \mathcal{I}_j)$), and $\mathbf{r}_j \triangleq \mathbf{R}(\mathcal{I}_j, j)$ as the $1 \times M_j$ vector with the corresponding rates, we get

$$\sum_{i=1}^{N} \mathcal{I}_{ij} \left[-\frac{1}{2} \log \frac{2\pi}{\alpha} - \frac{\alpha}{2}(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2 \right] = -\frac{M_j}{2} \log \frac{2\pi}{\alpha} - \sum_{i=1}^{N} \mathcal{I}_{ij} \frac{\alpha}{2}(r_{ij} - \mathbf{u}_i^T \mathbf{v}_j)^2$$

$$= -\frac{M_j}{2} \log \frac{2\pi}{\alpha} - \frac{1}{2}(\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j)^T \alpha \mathbf{I} (\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j) \ ,$$

and finally

$$\log p(\mathbf{v}_j|\mathbf{R}, \mathbf{U}, \Theta_v, \alpha) \propto -\frac{1}{2}(\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j)^T \alpha \mathbf{I} (\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j) + \kappa \boldsymbol{\mu}_v^T \mathbf{v}_j + \log \left(\mathcal{I}_{\mathcal{S}_D^{(q)}}(\mathbf{v}_j) \right) \ . \tag{7.2}$$

Drawing samples from this distribution is not as simple as for the BPMF, especially due to fact that we are now constrained to be in an hypersphere. In the following we will show how this can be done using two different MCMC methods, and in particular the already introduced Metropolis-Hastings (MH) algorithm and the Geodesic Monte Carlo (GMC).

7.2.1 Sampling using the Metropolis-Hastings algorithm

We recall that we have to sample from

$$p(\mathbf{v}_j|\mathbf{R}, \mathbf{U}, \Theta_v, \alpha) \propto p(\mathbf{R}|\mathbf{U}, \mathbf{v}_j, \alpha) p(\mathbf{v}_j|\Theta_v) \ ,$$

where the prior over $\mathbf{v}_j$ has a $vMF_q(\mathbf{x}; \boldsymbol{\mu}, \kappa)$ distribution. As already said, for our experiments we will use in particular a constant prior distribution (i.e. $\kappa = 0$) over the hypersphere $\mathcal{S}_D^{(q)}$; a possible extension of the proposed algorithm could make use of hyperparameters as well.
As we know how to draw samples from any Von Mises-Fisher distribution (see

Section 6.3.1), a natural way to sample from the posterior of interest is using the Metropolis-Hastings algorithm (see section 3.4) with a $vMF_q(\tilde{\boldsymbol{\mu}}_j, \tilde{\kappa})$ proposal distribution where at iteration t we use as mean direction the current sample, i.e. $\tilde{\boldsymbol{\mu}}_j^{(t)} = \mathbf{v}_j^{(t-1)}$. Due to the presence of the inner product in (7.1) we see that the proposal distribution is symmetric, leading to an acceptance probability of the form (3.3). In particular, as the prior over the movie vectors is assumed constant, the acceptance probability for the MH updates is reduced to (see equation (7.2))

$$\alpha = \min\left(1,\, e^{\left[(\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j^{(t)})^T \alpha \mathbf{I} (\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j^{(t)}) - (\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j^{(t-1)})^T \alpha \mathbf{I} (\mathbf{r}_j - \mathbf{U}_j^T \mathbf{v}_j^{(t-1)})\right]}\right) .$$

Starting from any initial state we would like to construct a Markov Chain that allows us to explore the regions of high probability in the posterior distribution. Figure 7.2 shows an example of application of the MH algorithm being constrained in a sphere in $\mathbb{R}^3$: we start from a random position (red cross in the figure) and slowly move on the unit hypershpere $\mathcal{S}_3$ towards the region of high posterior probability.

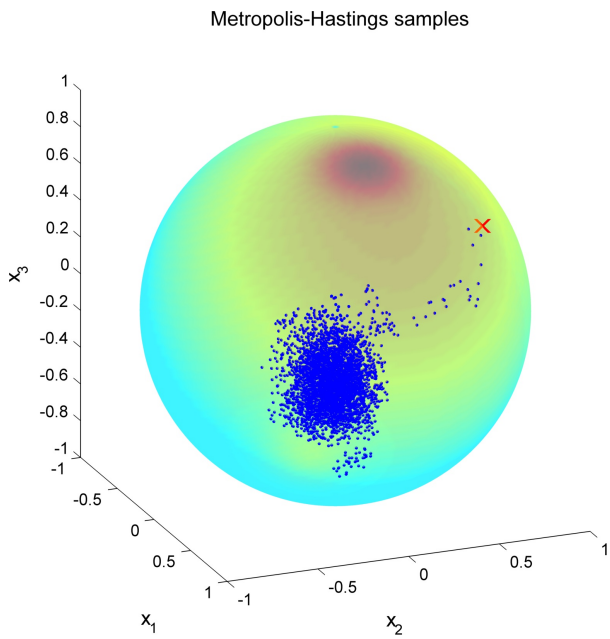


Figure 7.2: Metropolis Hastings algorithm on an hypersphere with $D = 3$, $\tilde{\kappa} = 500$. Many small steps are needed before getting to the high density region.

When using the Metropolis-Hastings algorithm there are some aspects one should

be really careful about. On the one hand at each iteration we would like to move a lot in the hypersphere, both to be able to escape quickly from a bad initial starting point (as in Figure 7.2) and to explore completely the space. This requires a small value for the concentration parameter $\tilde{\kappa}$. On the other hand, if the proposal distribution is too broad ($\tilde{\kappa}$ too small) we will likely have small acceptance probabilities and hence reject all the proposals. The best we can do to overcome these issues is to have a rather high κ but introduce several burn-in samples. In other words, we want to make several small steps that will hopefully lead to the high density region in the beginning and to uncorrelated samples once we are there. This will however increase the computational time.
Of course, the closest is the starting point to the region of interest the less burn-in one has to introduce, as the chain starts early to return samples from the desired stationary distribution. In our case one option is to start with the vector with norm q that has the same direction as the MAP solution for the unconstrained problem (that is obtained using Alternating Least Squares):

$$\tilde{\boldsymbol{\mu}}_j^{(0)} = q\bar{\mathbf{v}}_j^{MAP} = q\frac{\mathbf{v}_j^{MAP}}{||\mathbf{v}_j^{MAP}||} \ .$$

In particular it was taken $q = median(||\mathbf{v}_j^{MAP}||)$; more insights on why this is a starting guess for q good enough will be provided in the next sections.

7.2.2 Sampling using Geodesic Monte Carlo

As the von Mises-Fisher distribution is defined on a hypersphere, i.e. a differentiable manifold, we can also use Geodesic Monte Carlo, previously introduced in section 4.4. We know in fact both

- the gradient with respect to the random vector $\mathbf{v}_j$ of the log-posterior distribution (Petersen and Pedersen, 2012):

$$\nabla_{\mathbf{v}_j} \log p(\mathbf{v}_j|\mathbf{R}, \mathbf{U}, \Theta_v, \alpha) = \alpha \mathbf{U}_j(\mathbf{r}_j - \mathbf{U}_j^T\mathbf{v}_j) \ , \tag{7.3}$$

- the geodesics of the hypersphere, that are given by the *great circles* (see the example in section 4.1.4).

From the computational point of view these operation are very efficient[1], hence as we will see it will be much faster to train the model with GMC than with

[1] Noting that $\alpha\mathbf{U}_j(\mathbf{r}_j - \mathbf{U}_j^T\mathbf{v}_j) = \alpha\mathbf{U}_j\mathbf{r}_j - \mathbf{U}_j\mathbf{U}_j^T\mathbf{v}_j$ we see that we can precompute the very expensive matrix multiplication $\mathbf{U}_j\mathbf{U}_j^T$ just once for each sample, and not every "leapfrog" step.

MH. Also, in this case thanks to the proposal distributions obtained solving Hamilton's equations and considering the metric tensor of the sphere we can overcome several of the shortcomings of MH.
In Figure 7.3 we see the difference between the two algorithms sampling the movie vectors of a model with D = 3. Both Markov Chains start from the same random point and converge to the high density posterior region (we claim that the chain has converged as the results obtained with both methods are similar; of course however this cannot be proved formally). GMC however gets there just after one sample, and starts therefore to return posterior samples much before (less burn-in is needed). For the case D = 20 shown in Figure 7.4 we cannot of have a nice visualization as in Figure 7.3. We can however plot just some random components of the samples obtained with the Markov Chains: again we see that GMC gets to the (assumed) high density region much faster than MH (see e.g. components 12 or 19) and that the samples obtained with GMC are much less correlated.

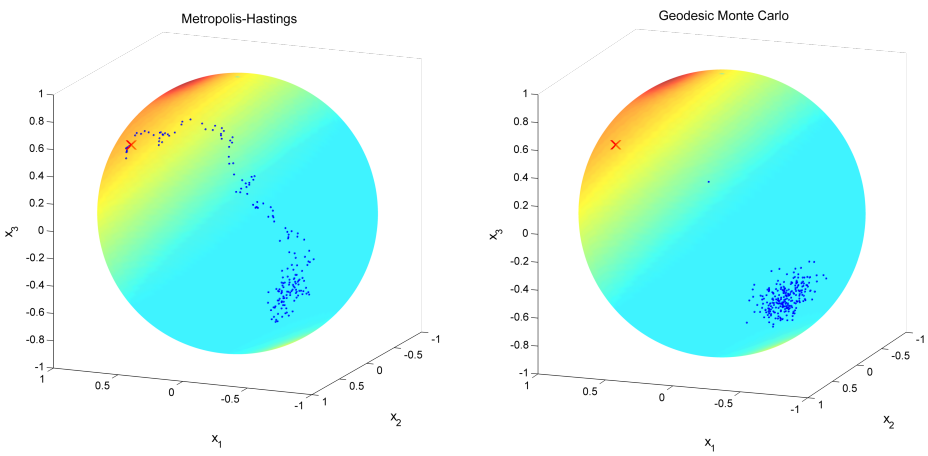


Figure 7.3: Convergence from a random starting point of Metropolis Hastings and Geodesic Monte Carlo for D = 3. We notice in particular that GMC reaches the high density region much faster than MH.

7.3 Gibbs sampler

Due to the increased complexity that one has to face when using a Bayesian approach it is common practice to initialize the variables that are being updated in the Gibbs sampler to their MAP solution (Salakhutdinov and Mnih, 2008b). In our case however a MAP approach for the constrained problem is still not available, but we have found out that using a modification of the MAP solution

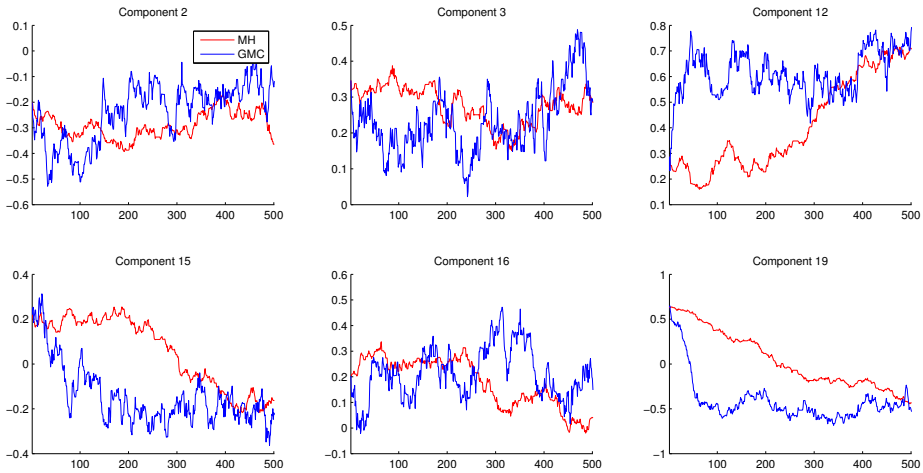


Figure 7.4: Convergence from a random starting point of MH and GMC for D = 20. In few iterations GMC reaches the high density region.

for the unconstrained problem gives good results. The Gibbs sampler used is then as follows:

Gibbs sampling for Constrained BPMF

1. Initialize the model parameters using the MAP solutions for for $\mathbf{U}$ and $\mathbf{V}$
   - the starting point for the user matrix is the MAP solution, i.e. $\mathbf{U}^{(0)} = \mathbf{U}^{MAP}$.
   - $q\bar{\mathbf{v}}_j^{MAP}$ is used as starting mean direction in the vMF proposal distribution of the MH algorithm.
2. For $t = 1, \ldots T$
   (a) Sample the *user hyperparameters* (Gaussian-Wishart distribution)
   (b) Sample the *movie vectors* in parallel (MH or GMC)
   (c) Sample the *user vectors* in parallel

As one can see from the algorithm, first we sample from the movie vectors and then from the user vectors, whereas it is usually done the other way round (Salakhutdinov and Mnih, 2008b; Zhou et al., 2008; Pilászy et al., 2010). The reason why in our case it is better to swap the order of these steps is that if we had to sample **U** first we would have to manually normalize the MAP movie vectors to have fixed norm, and this could potentially lead to a really bad sample from **U**. On the other hand, if we sample **V** first the normalized MAP movie vectors are just the starting point for the MH sampler, and after the burn-in samples we will hopefully end up in a much better solution.
Note also that as we are not starting from the MAP solution for the constrained problem, it may be convenient to introduce some burn-in in the Gibbs Sampler as well (for the results presented next 3 samples were used).

7.4 Results

Figure 7.5 shows the distribution of the norms of the movie vectors $\mathbf{v}_j$ for the unconstrained MAP solution of the Netflix data set in the case D = 50. As we can see, the fixed norm constraint we are imposing is quite strong (very far from the MAP solution). The main purpose of this section is therefore to understand

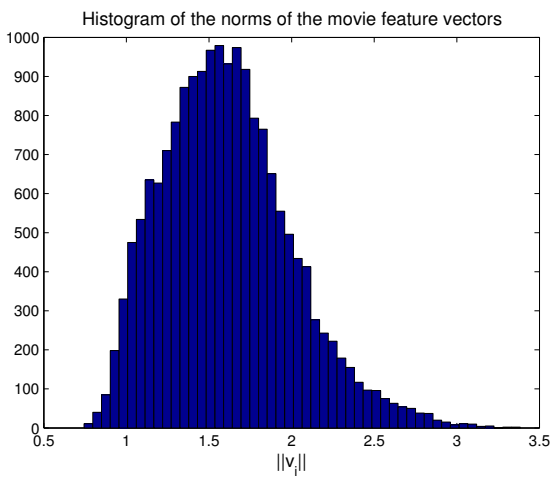


Figure 7.5: Histogram of the norms of the movie vectors for the MAP solution abtained with ALS. The median value is 1.579.

how this affects the recommendations. We want in fact not only to speedup the retrieval process but also to keep providing high quality recommendations

to the users. We will therefore now compare in terms of RMSE the proposed constrained algorithm with GMC (BPMF-vMF) to the unconstrained MAP solution obtained with Alternating Least Squares (PMF) and to the Bayesian solution obtained using the Bayesian Probabilistic Matrix Factorization (BPMF - this comparison is the most important as our approach is very similar). In the Bayesian approaches (BPMF and BPMF-vMF) 151 samples where drawn (MAP solution and 150 new other samples). The hyperparameters for the Geodesic Monte Carlo where optimized using cross-validation on the quiz set.
Due to the big number of parameters in the model with respect to the training data the convergence of the algorithm is quite fast, as one can see from Figure 7.6. We also notice that thanks to the Bayesian approach the model does not overfit the data.

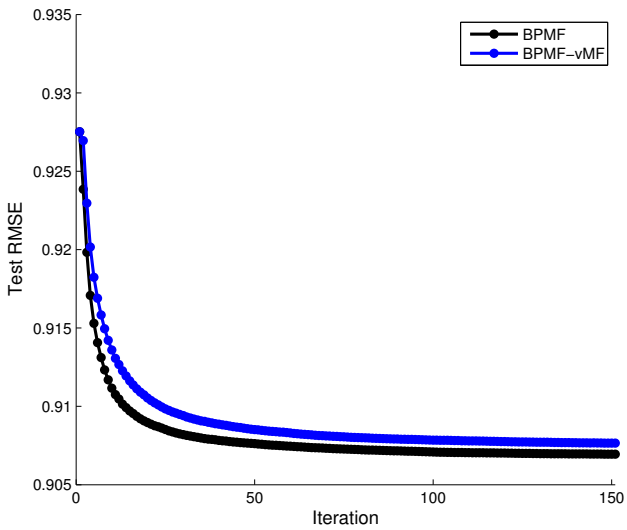


Figure 7.6: Convergence of BPMF and BPMF-vMF ($D = 20$).

The final Test RMSE for different numbers of latent features can be found in Table 5.1 (for now in none of the models the biases are considered).

Test set			
D	**PMF**	**BPMF**	**BPMF-vMF**
8	0.9333	0.9198	0.9210
20	0.9275	0.9070	0.9077
50	0.9182	0.8992	0.8994
100	0.9143	0.8955	0.8966

Table 7.1: Test RMSE for different values of D.

We see that the results obtained with BPMF-vMF are extremely close to the ones obtained with BPMF, especially despite the strong constraints for the 17770 movie vectors. It is however important to check which kind of errors we are doing: a risk that has to be avoided when constraining all the movie vectors to have the same norm is that we loose predictive power only for a specific type of users or movies (e.g. only users with few ratings or movies with few viewers). We test this in the following sections, first for groups of users with similar number of ratings and then groups of movies watched a similar number of times.

7.4.1 RMSE for different groups of users

We use the approach of section 5.4, where the users are grouped according to their number of ratings in the training data and it is computed a separate RMSE for each of these 9 groups.
For $D = 50$, the Test RMSE for the different groups of users and with the different algorithms is shown in Figure 7.7. We see that the blue line relative to the BPMF-vMF is almost indistinguishable from the black one relative to BPMF (it is always slightly above), meaning that there is not a particular group of users significantly penalized when constraining the movie vectors. As if we changed the value of D we would get a very similar plot, only the case $D = 50$ is shown.

7.4.2 RMSE for different groups of movies

As we are imposing constraints on the movie vectors, the analysis in this case will be more interesting and detailed than in the previous one (in which similar users were grouped).
We see from Figure 7.8 that, in the unconstrained MAP solution, for rare movies the mean norm of the corresponding vectors tends to be higher than for popular ones. This suggests that when imposing a fixed norm it is likely that some groups of movies will be more penalized than other ones.
The RMSE for all the groups in the case $K = 50$ are shown in Figure 7.9: the new constraints give some problems mostly for movies with a small number of viewers. The results are still better than the unconstrained MAP solution obtained with Alternating Least Squares, but now the difference between BPMF and BPMF-vMF is evident from the plot for the first 5 groups.
At first sight, these results may seem inconsistent with the RMSEs shown in

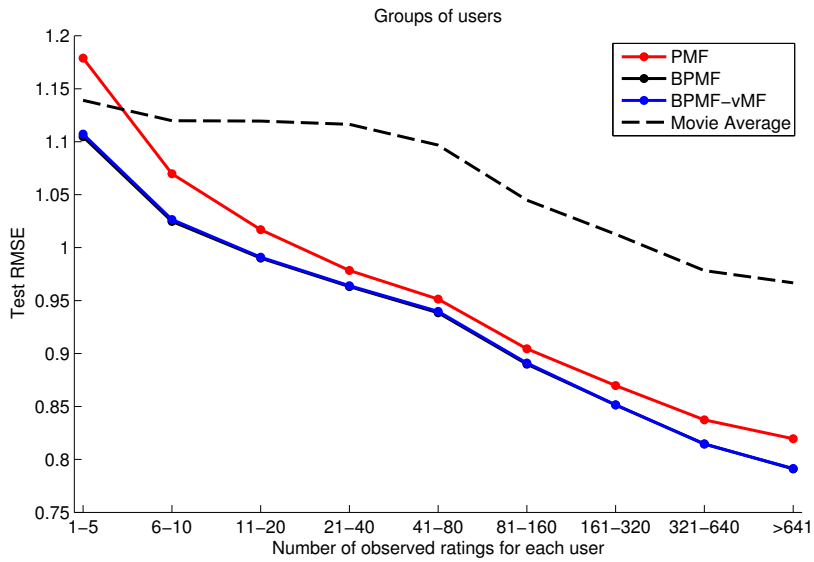


Figure 7.7: Test RMSE for the different groups of users ($D = 50$). The blue line (BPMF-vMF) and the black one (BPMF) are almost overlapping.

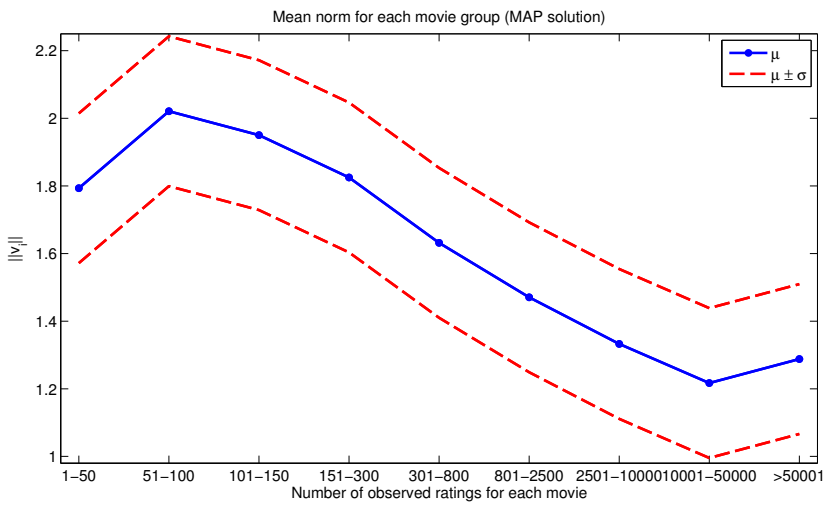


Figure 7.8: Mean norm and confidence interval for the groups of movie vectors of the unconstrained MAP solution.

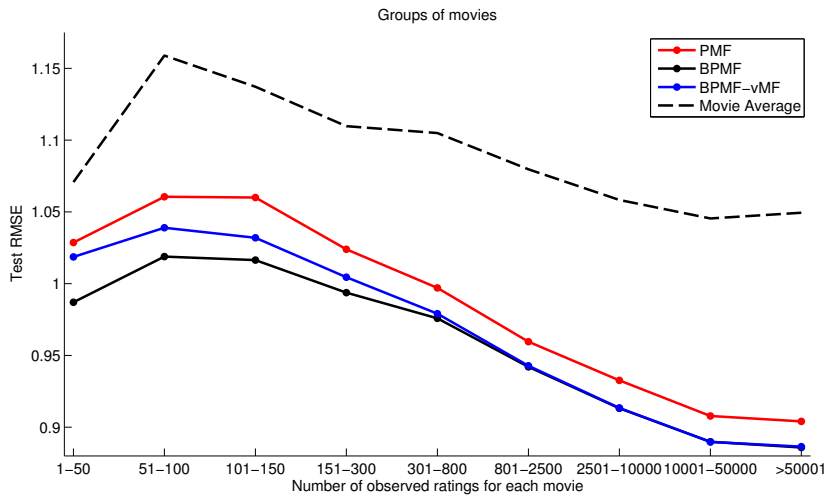


Figure 7.9: Test RMSE for the different groups of movies ($D = 50$). In this case we can clearly see the difference between the blue line (BPMF-vMF) and the black one (BPMF).

Table 7.1, where for $K = 50$ we see that the test RMSE obtained with BPMF is extremely close to the one obtained with BPMF-vMF. However, if we consider the distribution of the observed ratings in the training data set among the 9 groups of Figure 7.10 (the test set was constructed with similar properties), we see that only a very small fraction belongs to the groups in which the constrained algorithm has issues. Therefore, these will just slightly influence the total RMSE.

It is however important to point out that in real large-scale recommender systems such as the one used at Microsoft for Xbox Live, movies with less than a couple of hundreds of views are not even suggested, as they have not enough views and the training data is not sufficient to be confident about the predictions (Ulrich Paquet, personal communication, April 2014). This further shows that the learned constrained vectors can represent the data very well (the RMSE obtained is quite close to the one that BPMF gives).

Finally, considering the mean movie vectors' norm for each group shown in Figure 7.8, it can be interesting to see how changing the value q of the fixed norm affects the performances in the groups consisting of rarer movies. From Figure 7.11 we see that if q is either too small or too big the RMSEs tend to be worse. For popular movies the results do not differ that much, unless q is either too small or too big. For rarer movies instead, the variability is much bigger: as we can see from the quite different results that were obtained running two different times the simulation with $q = median(||v_j||) = 1.5796$ and the exact

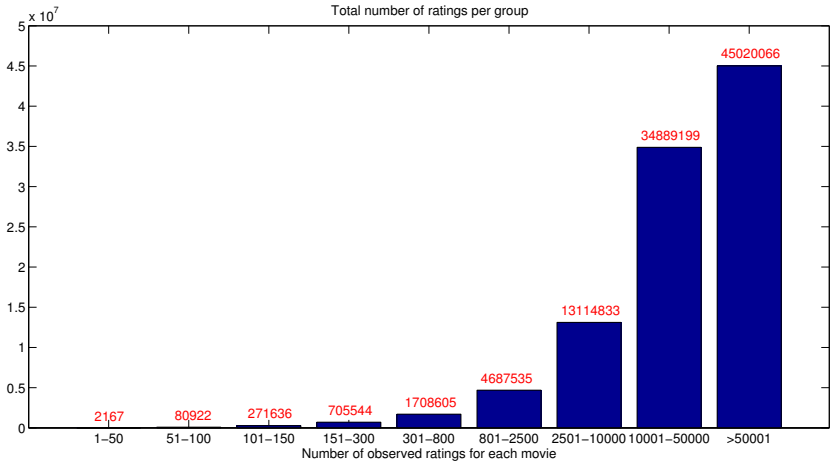


Figure 7.10: Number of ratings in each group of movies. The majority of the movies have a high number of ratings.

same parameters, this difference is mainly due to the intrinsic randomness of the sampling procedure and the small number of ratings available. We can therefore claim that any value of q close enough to the mean values in Figure 7.8 will give similar performances.

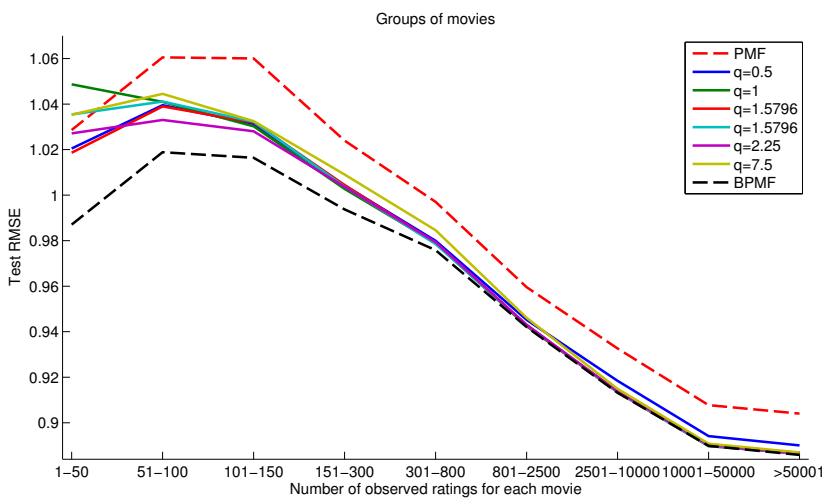


Figure 7.11: Group RMSEs for different values q of the fixed norm.

7.4.3 Choice of the sampling method

In section 7.2 were presented two methods to sample from the posterior distribution over the movie vectors: Metropolis-Hastings (MH) and Geodesic Monte Carlo (GMC). We now want to compare their performances in terms both of RMSE and speed.
The quiz set was used to tune the main parameters of the two algorithms in the case D = 20. To see how robust the methods are with respect to these parameters, their values were kept fixed for the other dimensions as well.

- For MH, the concentration parameter was taken $\kappa = 20000$, and 200 burn-in samples were needed at each iteration to get a good mixing of the chain.
- For GMC the step size we used is $\epsilon = 0.002$ and the number of "leapfrog" steps was $L = 10$. Due to the better proposal distributions derived from the Hamiltonian approach just 10 burn-in samples are necessary.

In terms of RMSE wee see that the two methods give similar results (Table 7.2), and this is mainly due to the careful tuning procedure. The Geodesic Monte Carlo seems to give some advantages in higher dimensions. We also note that, despite the parameters were tuned for the case D = 20 they work well for the other dimensions as well.

Test set			
D	**BPMF**	**BPMF-vMF, GMC**	**BPMF-vMF, MH**
8	0.9198	0.9210	0.9207
20	0.9070	0.9077	0.9077
50	0.8992	0.8994	0.8999
100	0.8955	0.8966	0.8979

Table 7.2: Test RMSE for different values of D using MH and GMC.

More interestingly, if we consider the sampling of the movie vectors, a one order of magnitude speedup is obtained using GMC: it is in fact around 10 times faster than MH (see section 7.4.4 for more details). This is due to the fact that the proposal distributions used in the Hamiltonian approach allow bigger steps accepted with high probability and therefore a much faster exploration of the space.
To provide more insights on why the starting guess proposed is good enough we can also check which RMSE the modified MAP solution would give, computing the error obtained using the matrices $\mathbf{U}$ and the normalized version of $\mathbf{V}$. For $K = 50$ for example, this Test RMSE is 0.9246, that is of course

much worse than the unconstrained MAP solution (RMSE of 0.9182) but that is still acceptable. This also explains why it was chosen a value for the norm of $q = median(||\mathbf{v}_j^{MAP}||)$: analyzing the distribution of the norms in the unconstrained MAP solution in Figure 7.5, it is clear that to have the smallest error in the first iteration we want to be as close as possible to the biggest number of samples, hence the choice of the median.

7.4.4 Running times

As shown in section 7.3, the Gibbs sampler for the introduced model is highly parallelizable. All the simulations presented before were done in the DTU high performance computing cluster using Matlab's function `parfor` to run the updates in parallel with 8 different local workers.
Table 7.3 shows the number of seconds necessary to perform a single parallelized Gibbs sampling iteration for different values of D and for BPMF, BPMF-vMF with GMC updates and finally BPMF-vMF with MH updates. As we are mostly interested in analyzing the loss in performance when imposing the fixed norm constraints, the number of seconds only for the updates of all the movie vectors (V step) is also listed in brackets. We note that while the training time per sample using GMC is comparable to the one for the classical BPMF introduced in section 5.3, MH is extremely slow: the update of the $\mathbf{V}$ matrix takes in fact around 10 times more than with GMC.

Seconds per sample			
D	**BPMF**	**BPMF-vMF (GMC)**	**BPMF-vMF (MH)**
8	22 (6)	33 (17)	236 (220)
20	31 (11)	45 (25)	296 (276)
50	62 (26)	91 (55)	552 (516)
100	137 (60)	205 (128)	1034 (957)

Table 7.3: Running times for a Gibbs sampling iteration, in brackets it is indicated the time to update all the movie vectors.

A visual comparison of the training time for the case D = 20 is shown in Figure 7.12.

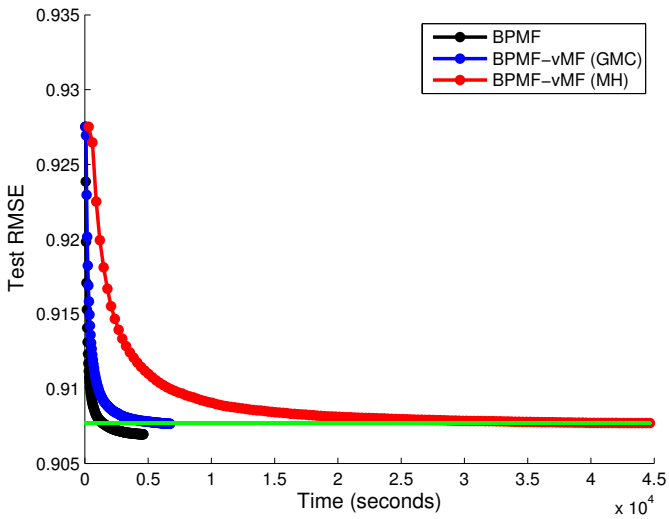


Figure 7.12: Comparison of the overall time taken to get 150 samples with BPMF, BPMF-vMF with MH, BPMF-vMF with GMC.

7.5 Modelling Biases

As explained in section 6.4, the inclusion of biases in the constrained model carries some complications if the final aim is to speedup the recommendation step. We will now show just the results obtained with the second approach introduced in section 6.4, leaving the other more complex option as a topic for future research.
Using this approach the biases are considered in the model in the exact same way as in section 5.3.2. The graphical model in this case is shown in Figure 7.13. From Table 7.13 we see that for the constrained model the biases allow a better modelling of the data especially for lower dimensions, while if the number of latent features is high enough there is no need to consider them.

RMSE for the test set				
D	**BPMF**	**BPMF bias**	**BPMF-vMF**	**BPMF-vMF bias**
8	0.9198	0.9173	0.9210	0.9205
20	0.9070	0.9065	0.9077	0.9070
50	0.8992	0.8994	0.8994	0.8995
100	0.8955	0.8957	0.8966	0.8967

Table 7.4: Comparison of the test RMSE for different values of D with and without biases in the model.

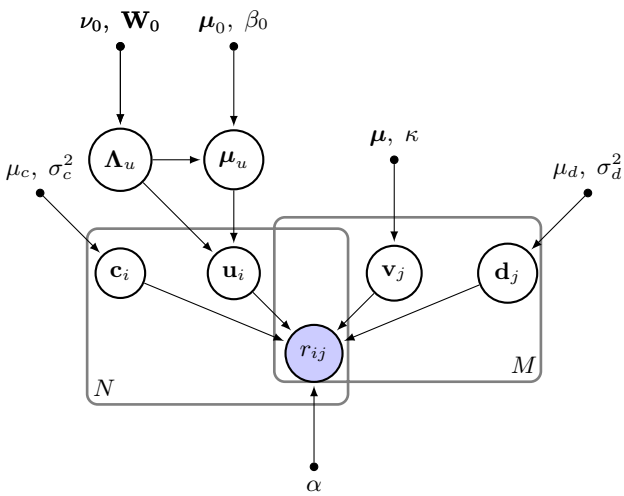


Figure 7.13: Graphical Model for the Constrained Bayesian Probabilistic Matrix Factorization with biases taken into account.

Chapter 8

Data Structures for Efficient Retrieval

8.1 Approximate Nearest Neighbors

Nearest neighbors search has become fundamental in a wide range of applications. In instance-based learning for example (Witten et al., 2011), a new test object is classified according to the class of the nearest neighbor in the training set. Also, for image matching or object recognition, this operation is crucial when using high dimensional features, e.g. SIFT features (Lowe, 2004), that have to be matched to their closest instance in the training data.
The simplest way to find the nearest neighbor of an object is to compute its distance to each training object, an operation whose complexity is linear in the number of training examples. When dealing with very complex problems however, having huge training data sets is often the only way to get accurate performance: the linear scaling of the nearest neighbor search is therefore a serious bottleneck. To overcome this issue, several data structures have been developed to speedup the retrieval phase (Samet, 2005). The main idea behind these structures is that of organizing offline the training data in a smart way, in such a way that with few operations we can rapidly discard whole regions containing points that are far from our test instance. The disadvantage of this approach is that the construction of these data structure is often a computationally expensive operation and they may need a lot of hard-disk space; this

is however not an issue in most of the applications, as often the training data set can be pre-processed and the data structures built only once and stored for future usage.

It should be now clear how we can speedup the prediction step in our recommender system application. We have seen in fact in section 6.2 that setting a constraint on the norm of the movie vectors we can transform the needed maximization over scalar products in a nearest neighbor search. We can therefore construct for the movie vectors of a posterior sample one of the available data structures for efficient retrieval, and recommend a new movie to user i looking efficiently for the nearest neighbor to $\mathbf{u}_i$. The information of several posterior sample has then to be combined to improve the quality of the recommendations.

The feature vectors in our recommender system application have to be high dimensional (e.g. D higher than 10) to model the data accurately. The performance of the above mentioned data structure however rapidly decreases when using high-dimensional data (Muja and Lowe, 2009): in this case there is often no known nearest-neighbor search algorithm that is exact and has acceptable performance. We can however deal with this problem thanks to a property of recommender systems: a recommendation can be considered valid as long as its predicted rating is close to the optimal one. In other words, it does not make much difference if we suggest an item with predicted rating 4.89/5 stars instead of the optimal one with 4.91/5. On the contrary, this can be considered as a positive aspect, as in this way we are able to enhance the engagement of the user by recommending different but good items every time. From a "distance minimization" point of view, this approach is equivalent to relaxing the requirement of finding the exact nearest neighbor, as long as the returned item is very close to it.

Thanks to this reasonable assumption we are now able to exploit a great deal of literature on *Approximate Nearest Neighbor (ANN) search*, that is still an important research topic in the computer vision community (Muja and Lowe, 2014). In object recognition for example, an object is represented using high-dimensional feature vectors and a match to the corresponding object in the training set is found by looking for the closest feature vectors in the training set. Due to the usual symmetries in real world objects and the usage of scale and rotation invariant features (e.g. SIFT features) the used algorithms perform equally well if a close approximation to the nearest neighbor is found. Muja and Lowe (2009) show that for the big and high-dimensional data sets typical in the computer vision community if one is willing to accept an approximate nearest neighbor algorithm with 95% of correct matches it is possible to be even two or more orders of magnitude faster than linear search. The speedup however depends on the particular application and the data structure used, therefore a detailed analysis is required before using this approach in our recommender system.

The most widely used approximate nearest neighbor search methods can be

classified in two main categories (Muja and Lowe, 2014):

1. **Partitioning trees** provide a hierarchical decomposition of the search space, that allows to rapidly focus the search on the region that is likely to contain the nearest neighbor.

2. **Hashing techniques** are based on the constructions of hash functions that have the property that elements that are close in space will likely have close hashes as well, see for example (Andoni and Indyk, 2008).

In this thesis we will focus on two data structures belonging to the first category, especially due to the better results they have shown to give in many different applications (Muja and Lowe, 2014).

8.2 k-d trees

The k-d tree, originally introduced in its basic version in (Friedman et al., 1977), is a tree built by partitioning the k-dimensional data recursively along the dimension of maximum variance.
At the root of the tree the data is split with an hyperplane orthogonal to the dimension of maximum variance, that divides the space into two halves using as a threshold the median value of the data along that dimension. The root node will then have to store the dimension i of the split and its median value m. Each of the two halves of the data is then recursively split in the same way, with the leaves of the tree that form a complete partition of the data space, dividing it in hyper-rectangular bins (depending on the implementation each bin may contain one or more points of the data set). This procedure will produce a well balanced binary tree of height $\log_2 M$, with M being the number of points in the data set.
Given a query point $\mathbf{q}^\star$, the constructed tree is descended using $\log_2 M$ comparisons to arrive to the bin/hyper-rectangle the point belongs to. With high probability the nearest neighbor will be in the bin where the query falls or in one of the adjacent ones. To look for the exact nearest neighbor of $\mathbf{q}^\star$ there are then two main techniques:

1. **Depth-first order**: the search continues examining the surrounding bins according to the tree structure, starting therefore from the sibling of the current node. It is possible to save lots of comparisons pruning whole branches of the tree if the region of space they represent is further from

the query point than the distance to the current hypothesis for the nearest neighbor.

2. **Best-bin-first**: this method allows faster retrieval than the one obtained with the depth-first approach by taking into account the position of the query point rather than the structure of the tree, that depends only on the stored points (Beis and Lowe, 1997; Arya and Mount, 1993). While we descend the tree looking for the bin $\mathbf{q}^\star$ belongs to, when a decision is made at an internal node to branch in one direction an entry is added to a priority queue that stores information about the option not taken, namely the position on the tree and the distance of the query point from the node along the corresponding dimension. After the descent of the tree is completed the search will start again from the closest node in the priority queue, that will be continuously updated with the information of the new nodes visited. As before, it is possible to speedup the retrieval by not visiting whole branches of the trees that represent points that are more distant than the current guess for the nearest neighbor.

Example
Let us consider the 2-dimensional nearest neighbor search problem in Figure 8.1. Using only the information on the data set (red dots) we first partition the space using a k-d tree: as the dimension of highest variance considering the whole data set is the horizontal one, we split the space vertically using as a threshold the median value of all the points along the horizontal dimension (black line). We then recursively split the 2 halves found along their dimension of highest variance to obtain first the green hyerplanes, and then the orange ones.
Given a query point we then descend the tree to get to the corresponding bin (a rectangle in this case) and compute the distance to the data set point stored in it, that will be the first guess for the nearest neighbor. All the bins that are not intersected by the "remaining search hypershpere" will not contain any point closer to the query point than the current guess, and can therefore be neglected in the search, with clear computational savings. The search has now to be continued analyzing adjacent bins: with the depth-first approach we would first analyze the sibling of the current node, i.e. the top-left rectangle, whereas using the best-bin-first method we first analyze the bin formed by the horizontal green line, taking therefore into account that the query point is very close to its border.

The computational analysis of the k-d tree search is tricky and general results are not possible as the efficiency of the algorithm highly depends on the consid-

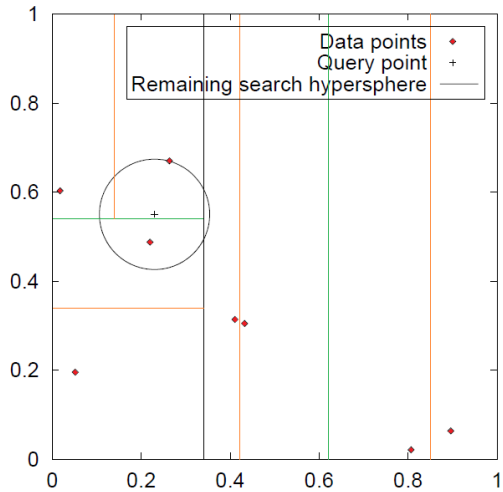


Figure 8.1: A 2-dimensional kd-tree. See the example for a detailed description of the figure.

ered data set (e.g. if the data is highly correlated or not). A lower bound for this operation is however $\mathcal{O}(\log_2 M)$, as we need at least to descend the whole tree, that has height $\log_2 M$. While this can be achieved in low-dimensional problems, this rarely happens in high dimensions, as due to the curse of dimensionality for an exact search we need to visit many more branches (there are many more bins adjacent to the central one). For the most difficult data sets it may even be possible to reach $\mathcal{O}(M)$ complexity, therefore with a small or no improvement with respect to a linear scan of the data set.
If the problem in hand allows the acceptance of approximate nearest neighbors, we can however speedup the retrieval process by imposing a limit on the number N_L of leaf nodes we are willing to examine. With this constraint, Muja and Lowe (2009) show that it is possible to obtain even three order of magnitude speedups while keeping an extremely high accuracy, in the sense that the method will return the nearest neighbor for a large fraction of the queries and a very close neighbor otherwise (especially when using a best-bin-first approach).
It is finally worth noting that with straightforward modifications of the introduced algorithm we can also find the K nearest neighbor to the query point: we just need to maintain the K current best guesses instead of just one and prune a branch if we are sure that none of its points will be closer than one of the current nearest neighbors.

8.2.1 k-d forests

Silpa-Anan and Hartley (2008) proposed a method to further improve the performance of the k-d tree when looking for nearest neighbors in high dimensions. Instead of using just a single tree, they use multiple *randomized k-d trees*, also known as *k-d forests*, that are searched in parallel. Each of the tree is constructed by splitting the data set at each iteration along one random dimension among the top N_D ones with highest variance (e.g. N_D=5). With the best-bin-first approach a single priority queue is maintained across all the randomized trees: setting an upper bound on the number of allowed comparisons as explained above, on average each randomized tree will be then visited $\frac{N_L}{N_D}$ times. We can improve the performances using multiple trees with different structures as we increase the probability of having the query point and its nearest neighbor in the same cell, or at least in a very close one. Also, in high dimensions it turns out that many of them have a similar variance, hence there is little or even no loss choosing a random dimension among the top ones instead of the optimal one.

8.3 Priority search k-means trees

Instead of decomposing the space using hyper-planes along one of the coordinates as done by k-d trees, the *priority search k-means tree* (Muja and Lowe, 2009) decomposes it using hierarchically the clustering algorithm known as k-means (Bishop, 2006). It therefore has the advantage of exploiting the natural structure of the data in the construction phase, as to cluster the points it is computed the distance across all dimensions, and not only across one of them as the k-d tree does.
Each level of the tree is constructed by dividing the data points in k groups using the k-means algorithm. Each group is then recursively partitioned in the same way until the number of points is smaller than k. An example of tree with $k = 2$ is shown in Figure 8.2.
To search for the nearest neighbor of a query point, the tree is descended following the branch with the closer cluster center to the query point. Once a leaf node is reached, the search then continues as for the k-d trees in a best-bin-first manner. Muja and Lowe (2009) show that the priority search k-means tree can outperform k-d trees for some data sets, while for others k-d trees are still better.

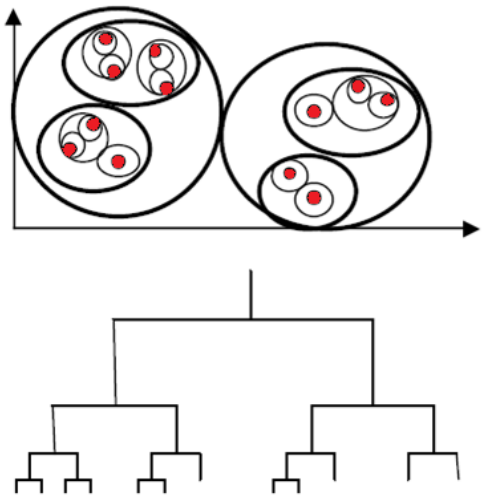


Figure 8.2: A 2-dimensional priority search k-means trees. On the top figure we see how the points are clustered, while the bottom one represents the constructed data structure.

8.4 Approximate Nearest Neighbors for recommender systems

We will now see if the introduced data structures for approximate nearest neighbors (ANN) search can be exploited to speedup the recommendation step and still provide high-quality results.
We assume that the model is properly trained with the learning procedure introduced in Chapter 7, and we focus for now on a single posterior sample. If we could do an exact nearest nearest neighbor search with one of the data structures introduced above, than we would be sure about the quality of the recommendations, as they would be the same obtained by maximizing the scalar product (6.1). Due to the high dimensionality of the problem, however, we can only get faster recommendations by introducing some approximations in the nearest neighbor search. We need to be very careful with the results, as we want the final suggestion to the user to be good enough.
To be useful in real world applications, the introduced approximate procedure should return suggestions as similar as possible to the ones obtained by maximizing in an exact way the scalar product (6.1). In Figure 8.3 we show an example of recommendations returned to a random user by the approximate method for the Netflix data set: in particular, it is shown whether the top 60 movies returned by maximizing the scalar product were found or not by the

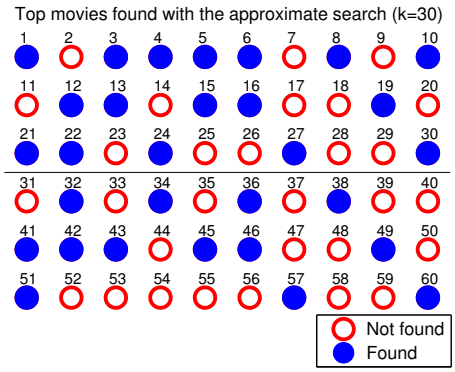


Figure 8.3: Example of approximate recommendation that could be returned by the proposed algorithm.

approximate search. Let us define some quantities that will help us to analyze the quality of the recommendations returned by the proposed method. We assume that with our recommender system we want to show to the user a list with the top $k = 30$ movies to the user. As we can see from Figure 8.3, due to the approximations in the nearest neighbor search we will not be able to find exactly the top 30. This is fine, as long as the 30 movies suggested are good enough: in this example we see that all the 30 movies suggested (the blue dots) are among the top 60, hence very close to the optimal solution (especially if we consider that the catalog of the data set contains 17770 movies).
We then define for a single user the following quantities:

- The **accuracy** is calculated as

$$\frac{\text{number of correct matches in the first k items}}{k}.$$

  In other words, we want to see the ratio of the top k items that was found by the approximate search. In the example in Figure 8.3, the accuracy is $\frac{17}{30} = 0.5667$.

- We are also interested in the position of the **worst suggestion**, as we do not want the k items returned by the approximate search to be bad recommendations. In the example the 30-th item suggested is in position 60.

- We define the recommendations to user i as **accepted** if, given a threshold N_{th} indicating high-quality recommendations, the *worst suggestion* is below it. If for instance in the Neflix data set we decided that any of

the first top $N_{th} = 100$ recommendations is good enough, than for the example in Figure 8.3 the recommendations would be accepted. If on the other hand we wanted $N_{th} = 50$, then the recommendations would not be good enough.

- The **speedup** of the recommender system is defined as the ratio of the time taken to make a recommendation by maximizing the scalar product and the time taken with the approximate nearest neighbor approach.

We would then like our model to have high speedup, mean accuracy across all users close to one and the position of the worst suggestion among all users as close as possible to k. We finally want a high **acceptance rate** for the system, defined as the ratio of users whose recommendations are accepted. Of course all these quantities are strongly correlated among them: it should be clear for example that if the worst suggestion among all users is very low, then the acceptance rate of the system will be really high.

8.5 Results for a single posterior sample

We will now analyze the speedup and performance that one can obtain for the Netflix data set using the presented data structures for efficient retrieval for a single posterior sample (these results would be similar for the MAP solution of the constrained problem).
In our Matlab implementation we used the data structures from the library FLANN - Fast Library for Approximate Nearest Neighbors (Muja and Lowe, 2009) - that implements efficiently the algorithms described previously in this chapter (using `mex` functions in Matlab to run C++ code). Given the data set, in our case the movie vectors, with this library there are two main options to construct the data structure:

1. The data structures and the parameters can be specified manually:
   - For the *k-d forests* the main parameters to set are the number of trees to use and the allowed number N_L of comparisons (that gives the approximation). Of course, the bigger these numbers are, the better will be the nearest neighbors found but the smaller will be the speedup obtained. For our simulations we used 4 trees and tried different numbers of comparisons:
     $N_L \in \{1, 20, 200, 600, 1000, 1500, 2000, 3000, 5000\}$
   - For the *priority search k-means trees* the main parameters to set are the number K of clusters to use, the number of iterations for the

k-means algorithm and again the maximum number of comparisons N_L. 64 clusters with the default value of 5 iterations and the same N_L as for the k-d forests were used in the simulations.

2. It is also possible to decide for an automatic selection of the optimal algorithm (*autotuning*). In this case one has to specify a target mean accuracy[1] and a cross-validation technique is used to determine the optimal data structure and the corresponding parameters. Of course, if one requires a too big accuracy then the nearest neighbor search will be rather slow. See (Muja and Lowe, 2009) for more details. We tried in the simulations 10 different values for the target accuracy, equally spaced between 0.1 and 1.

Figure 8.4 shows the results obtained in our simulations with D = 20 using the methods and parameters specified above, and comparing in various ways the quantities defined in section 8.4. We are interested in giving to the user $k = 10$ suggestions, and we consider the recommendations as accepted if the worst one among them is in the top $N_{th} = 100$. We see that for this data set the priority search k-means trees outperform k-d forests, in the sense that they allow much better nearest neighbors searches while still having a higher speedup. Also, we notice that the autotuned data structures, despite being slower to create due to the required cross-validation, are the best performing ones. In this case, for very low target accuracies the method chooses k-d forests, while for the others priority search k-means trees are always the best option.
The data structure that probably gives the best trade-off between speedup and quality of the recommendations is marked in Figure 8.4 with an asterisk. The automatic tuning procedure has chosen a priority search k-means tree with $K = 32$ clusters, 10 iterations for the k-means algorithm and $N_L = 168$ maximum comparisons. From Figures 8.4a and 8.4b we see that it is possible to achieve a 18.79x speedup with respect to the scalar product computation, while keeping a mean accuracy of 0.549 and an acceptance rate of 0.997. Figure 8.4c shows finally that the worst of the 10 recommendation considering all the 480189 users of the Netflix data set is in position 314 (recall that the Netflix data set has a catalog with 17770 movies). The presented results are very interesting: the 0.997 acceptance rate means that just for 1413 users out of 480189 we have given recommendations not in their top 100 suggestions. Also, as the mean accuracy is 0.549 we know that on average 5.49 suggested movies are in the top 10. The remaining suggestions are very close to the top 10, and we have also proven that the worst mistake the approximate search has done considering all 480189 is that for one of them the 10th movie suggested was in position 314 out of 17770. Figure 8.4d finally shows that if we increase the number of latent

[1] The accuracy in this case is defined as shown in section 8.4 but with $k = 1$ fixed, i.e. only the closest nearest neighbor is considered.

features the achievable speedup decreases. This is not trivial, as (Muja and Lowe, 2014) show that for some data sets the speedup is not affected by the dimensonality of the vectors in hand.
Note in particular that the presented speedup of 18.79x is relative to the computations of scalar products in equation (6.1). If we consider just the distance minimization problem, then the speedup relative to the exact computation of the distances (i.e. the exact nearest neighbor search) would be bigger, in this case 20.19x.

8.6 Combining the information of several posterior samples

As we have just seen, constructing a priority search k-means tree we can perform a much faster but still high-quality recommendation step using a single posterior sample. To fully exploit the power of the Bayesian approach, that is in this case needed particularly to deal with rarer contents, we need however to combine the information given by several posterior samples. This has furthermore to be done in a very efficient way to be able to use the introduced model in a real large-scale system.
The simplest method one could think about to average between samples is to construct a different data structure for each of the samples, and combine the top recommendations from each of them using a suitable voting scheme. Even if this method could give good recommendations it would be rather inefficient: let us assume for example that we have to recommend just a single top item to a given user. In this case, if we considered for each of the samples just the best suggestion, then all of them could be different, making the voting scheme useless. Therefore, to provide just one good recommendation we would need to look for the $T > 1$ best recommendations in each of the samples (e.g. $T = 20$), a rather computationally expensive operation. A better approach that avoids the construction of different data structures for each samples is therefore needed.
From a Bayesian perspective, the best suggestion can be found maximizing the expected rating: given S posterior samples $\mathbf{U}^{(s)}$ and $\mathbf{V}^{(s)}$ for $\mathbf{U}$ and $\mathbf{V}$ respectively we can find the best suggestion $\hat{j}$ as[2]

$$\hat{j} = \arg\max_j \frac{1}{S} \sum_{s=1}^{S} (\mathbf{u}_i^{(s)^T} \cdot \mathbf{v}_j^{(s)}) \ , \tag{8.1}$$

We can of course see this equation as a Bayesian extension of (6.1).
Similarly to what we did in section 6.2, we can rewrite the scalar product in

[2]This equation can be easily extended when considering biases in the model.

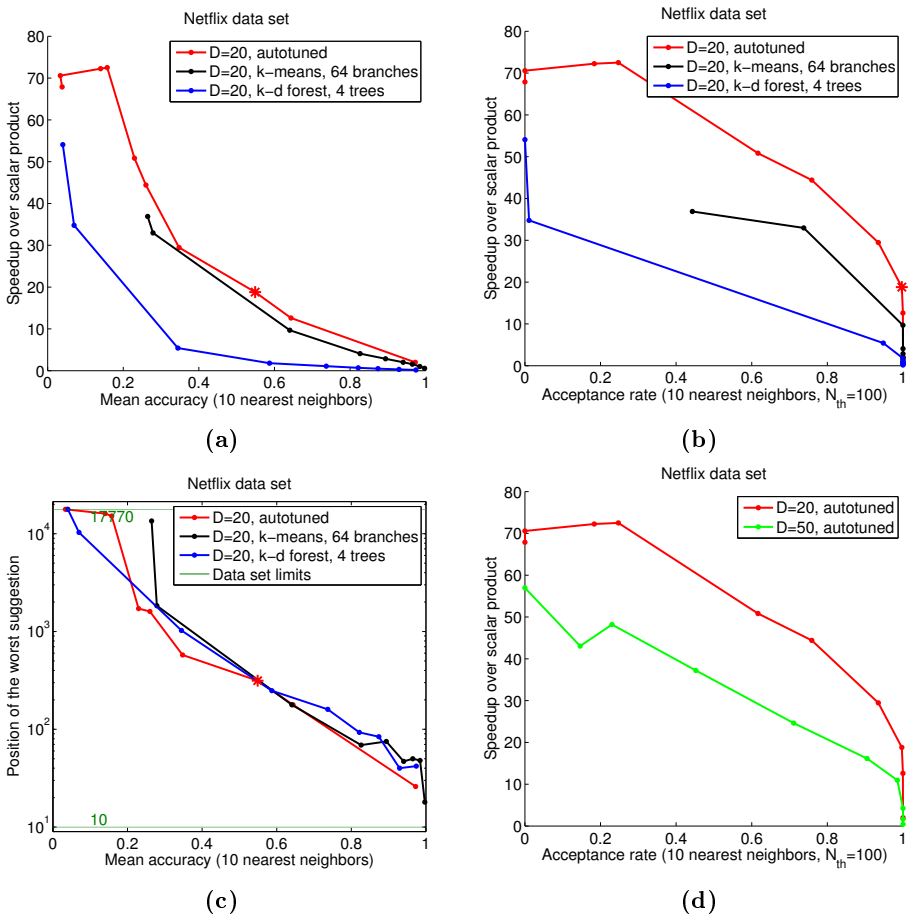


Figure 8.4: Performance analysis for the approximate nearest neighbor step of the recommender system (see the text for a detailed description of the figure).

term of norms:

$$\mathbf{u}_i^{(s)^T} \cdot \mathbf{v}_j^{(s)} = \frac{1}{2}(||\mathbf{u}_i^{(s)}||^2 + || \cdot \mathbf{v}_j^{(s)}||^2 - ||\mathbf{u}_i^{(s)} - \mathbf{v}_j^{(s)}||^2) \ .$$

Then, fixing again the norm of the movie vectors we have

$$\begin{aligned}
\hat{j} &= \arg\max_j \frac{1}{S} \sum_{s=1}^{S} (\mathbf{u}_i^{(s)^T} \cdot \mathbf{v}_j^{(s)}) \\
&= \arg\max_j \frac{1}{S} \sum_{s=1}^{S} \frac{1}{2} \left(||\mathbf{u}_i^{(s)}||^2 + || \cdot \mathbf{v}_j^{(s)}||^2 - ||\mathbf{u}_i^{(s)} - \mathbf{v}_j^{(s)}||^2 \right) \\
&= \arg\max_j \frac{1}{S} \sum_{s=1}^{S} ||\mathbf{u}_i^{(s)}||^2 + \frac{1}{S} \sum_{s=1}^{S} || \cdot \mathbf{v}_j^{(s)}||^2 - \frac{1}{S} \sum_{s=1}^{S} ||\mathbf{u}_i^{(s)} - \mathbf{v}_j^{(s)}||^2 \\
&= \arg\min_j \sum_{s=1}^{S} ||\mathbf{u}_i^{(s)} - \mathbf{v}_j^{(s)}||^2 \ .
\end{aligned}$$

In this case we have therefore no longer a nearest neighbor search as we have to minimize a sum of squared distances. However, if we define

$$\tilde{\mathbf{u}}_i = \begin{bmatrix} \mathbf{u}_i^{(1)} \\ \vdots \\ \mathbf{u}_i^{(S)} \end{bmatrix} \qquad \tilde{\mathbf{v}}_j = \begin{bmatrix} \mathbf{v}_j^{(1)} \\ \vdots \\ \mathbf{v}_j^{(S)} \end{bmatrix}$$

we can rewrite the previous equation as

$$\arg\min_j \sum_{s=1}^{S} ||\mathbf{u}_i^{(s)} - \mathbf{v}_j^{(s)}||^2 = \arg\min_j \left|\left| \begin{bmatrix} \mathbf{u}_i^{(1)} - \mathbf{v}_j^{(1)} \\ \vdots \\ \mathbf{u}_i^{(S)} - \mathbf{v}_j^{(S)} \end{bmatrix} \right|\right|^2 = \arg\min_j ||\tilde{\mathbf{u}}_i - \tilde{\mathbf{v}}_j|| \ ,$$

hence

$$\hat{j} = \arg\max_j \frac{1}{S} \sum_{s=1}^{S} (\mathbf{u}_i^{(s)^T} \cdot \mathbf{v}_j^{(s)}) = \arg\min_j ||\tilde{\mathbf{u}}_i - \tilde{\mathbf{v}}_j|| \ .$$

We have therefore derived another distance minimization problem, with the difference that in this case the vectors to be considered are derived by stacking the posterior samples. To solve it we can use all the previously introduced data structures, in this case however we will work in much higher dimension.

8.6.1 Results

Figure 8.5 shows the results for the nearest neighbor search when considering more than one posterior sample. The data structure that gives the best com-

promise between quality of the recommendations and speedup with 10 posterior samples is shown with an asterisk. If we combine the information of the 10 posterior samples, then we can obtain a 8.16x speedup with respect to the scalar product computation[3] while achieving a mean accuracy of 0.47 and an acceptance rate with $N_{th} = 100$ of 0.976 (Figures 8.5a and 8.5a). In other words, apart from 11736 users out of 480189, the 10 recommendations found with the approximate nearest neighbors search are in the top 100 and around 5 of them are on average in the top 10. From Figure 8.5c we see that the worst error the system does is to recommend for one user a movie in position 879 (out of 17770 movies). This is however just one of a small number of outliers, as from the histogram of the position of the worst recommendations for each user in Figure 8.5d we see that it is almost always below 200. In Figure 8.5 we also show the results combining 20 posterior samples: in this case the speedup that one can achieve is lower, 4.15x, with a mean accuracy of 0.582, an acceptance rate of 0.999 and the worst suggestion in position 355.
It is finally worth to point out that the Netflix data set is a relatively small one (despite being one of the biggest explicit feedback data set publicly available, widely used in the recommender systems community), therefore as we have just seen the speedups obtainable with it are still limited. Also, when the number of items in the catalog is below, say, 50000, then it is still possible in real large scale recommender system such as the one used by Microsoft for Xbox Live to perform a linear search through all the items (Ulrich Paquet, personal communication, June 2014). The model introduced in this thesis could be however fundamental to improve the performances in other domains, such as phone apps and music recommendations, where the number of items in the catalog could easily reach hundreds of thousand and tens of millions respectively. The most important results that we learned analyzing movies recommendation is that despite the constrained model there is almost no loss in the quality of the recommendations. Therefore, it could in principle be used in real world applications where a non-conventional approach is needed due to the big size of the catalog.

[3]The speedup with respect to the exact distance computation is 60x, bigger than in the case of a single posterior sample due to the higher dimensionality of the problem.

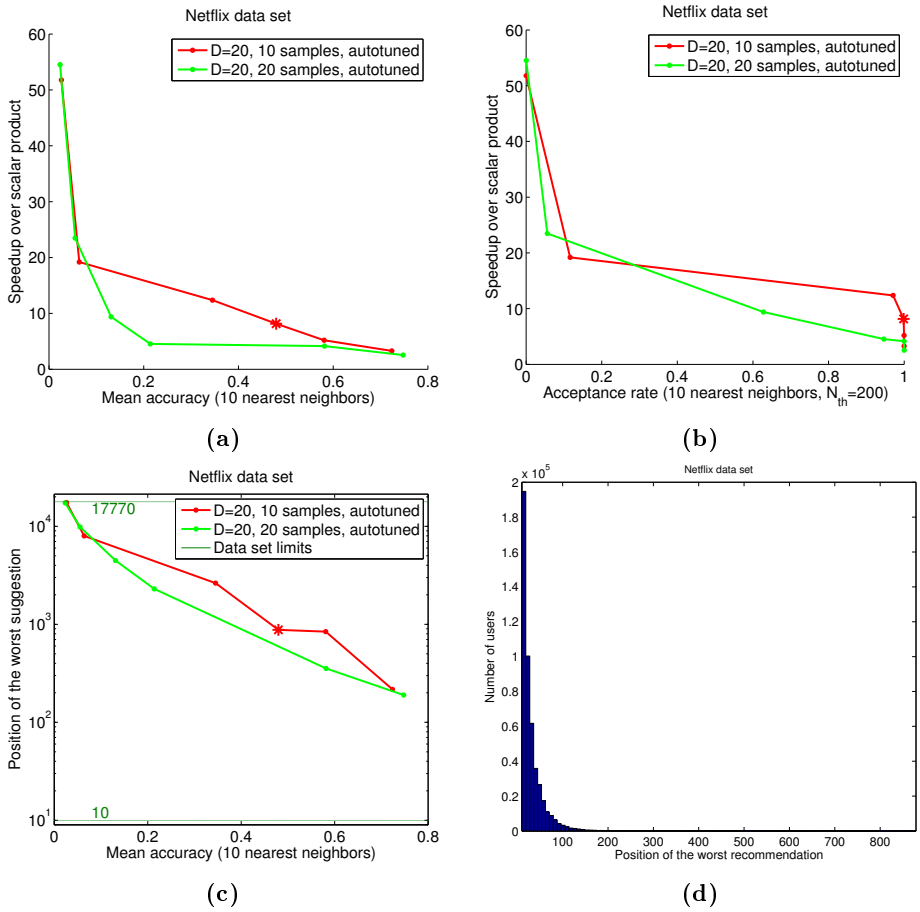


Figure 8.5: Performance analysis for the approximate nearest neighbor step of the recommender system combining the information several posterior samples (see the text for a detailed description of the figure).

Chapter 9

Conclusions and future work

In this thesis we proposed a novel approach to a problem of great importance in real world recommender systems, namely a way to give optimal recommendations in large-scale systems containing a huge number of items in their catalogs. For this purpose it was developed and analyzed a recommender system based on matrix factorization techniques that, thanks to the fixed norm constraints on the item vectors, allows faster but still high quality recommendations. As shown, the methods used can be naturally incorporated in a Bayesian framework, that is essential to deal with rarer contents and account for uncertainties in the predictions.
The model was tested with the Netflix data set, and the results in terms of RMSE showed that the quality of the recommendations is only slightly affected by the imposed constraints. More importantly, we showed that there is no category of users or movies (obtained grouping them according to the number of ratings they have) that is particularly penalized by this approach. Due to the relatively small size of the data set the speedup obtained are promising but still limited (4 times faster combining 20 posterior samples). Tests with some larger data sets used in real world applications will likely give much bigger improvements, possibly justifying the development of recommender systems based on the proposed method. In this case the efficiency of the training phase is equally important, as the systems need to be re-trained even every day to take into account the incoming new data. We showed however that the Geodesic Monte Carlo algorithm allows a very efficient training phase for the proposed model.

It is finally worth considering how some of the best performing ideas were obtained combining the efforts of different research fields: the training phase deeply exploits research topics from physics (Hamiltonian mechanics), differential and information geometry (Riemmanian manifolds), and machine learning (MCMC methods). Also, the data structures for efficient retrieval used in this work were developed to solve nearest neighbor problems in the field of computer vision. There is still a lot of research going on in these fields to improve the algorithms used, meaning that the recommender system developed could potentially benefit from a great deal of literature still to appear.

Considering the future work, it will be essential to try the power of the proposed model with a big real world data set, both to test the performances from a RMSE/speedup point of view and to see if the training phase is fast enough. It will be then necessary to assess how the results change considering a varying number of posterior samples, or using a variational approach to deal with the Bayesian perspective. The model could be extended taking into account for example hyper-priors for the von Mises-Fisher distributions on the item vectors (as it is done for the user vectors with the Gaussian-Wishart hyperprior), biases, or using only implicit feedback for the construction of the utility matrix. Finally, the results could be improved by merging the training procedure with the construction of the data structure, creating therefore a system that is even closer to "learning to index".

Bibliography

S. Amari and H. Nagaoka. *Methods of Information Geometry*, volume 191 of *Translations of Mathematical monographs*. Oxford University Press, 2000.

Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006. ISBN 1401302378.

Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327494. URL `http://doi.acm.org/10.1145/1327452.1327494`.

Sunil Arya and David M. Mount. Algorithms for fast vector quantization. In *Proc. of DCC '93: Data Compression Conference*, pages 381–390. IEEE Press, 1993.

Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, and Suvrit Sra. Clustering on the unit hypersphere using von mises-fisher distributions. *J. Mach. Learn. Res.*, 6:1345–1382, December 2005. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1046920.1088718`.

Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 1000–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7822-4. URL `http://dl.acm.org/citation.cfm?id=794189.794431`.

Robert M. Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 43–52,

Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3018-4. doi: 10.1109/ICDM.2007.90. URL http://dx.doi.org/10.1109/ICDM.2007.90.

James Bennett and Stan Lanning. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, December 1995. ISSN 0036-1445. doi: 10.1137/1037127. URL http://dx.doi.org/10.1137/1037127.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Simon Byrne and Mark Girolami. Geodesic monte carlo on embedded manifolds, June 2013. URL http://arxiv.org/abs/1301.6064.

Inderjit S. Dhillon and Suvrit Sra. Modeling data using directional distributions. Technical report, University of Texas, 2003.

P. Diaconis, S. Holmes, and M. Shahashahani. Sampling from a manifold. *Festschrift for Joe Eaton*, 2011.

Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977. ISSN 0098-3500. doi: 10.1145/355744.355745. URL http://doi.acm.org/10.1145/355744.355745.

W. R. Gilks. *Markov Chain Monte Carlo In Practice*. Chapman and Hall/CRC, 1999. ISBN 0412055511.

Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *J. of the Royal Statistical Society, Series B (Methodological*, 2011.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, April 1970. ISSN 1464-3510. doi: 10.1093/biomet/57.1.97. URL http://dx.doi.org/10.1093/biomet/57.1.97.

Peter D. Hoff. Simulation of the Matrix Bingham-von Mises-Fisher Distribution, with Applications to Multivariate and Relational Data. *Journal of Computational and Graphical Statistics*, 18(2):438–456, 2009.

Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15:1593–1623, 2014. URL http://jmlr.org/papers/v15/hoffman14a.html.

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems An Introduction*. Cambridge University Press, 2011.

H. Jeffreys. *Theory of Probability*. Oxford, Oxford, England, 1948.

Yehuda Koren. The bellkor solution to the netflix grand prize, 2009. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.162.2118`.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL `http://dx.doi.org/10.1109/MC.2009.263`.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

P. Marriott and M. Salmon. An introduction to differential geometry. 2000.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. doi: 10.1063/1.1699114. URL `http://link.aip.org/link/?JCP/21/1087/1`.

Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1.

Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)*, pages 331–340. INSTICC Press, 2009.

Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

Radford Neal. Slice sampling. *Annals of Statistics*, 31:705–767, 2000.

Radford M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162, 2010.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

Ulrich Paquet and Noam Koenigstein. One-class collaborative filtering with random graphs. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 999–1008, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-2035-1. URL `http://dl.acm.org/citation.cfm?id=2488388.2488475`.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012. URL `http://www2.imm.dtu.dk/pubdb/p.php?3274`. Version 20121115.

István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 71–78, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-906-0. doi: 10.1145/1864708.1864726. URL `http://doi.acm.org/10.1145/1864708.1864726`.

Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, Cambridge, 2012. ISBN 9781139157926 1139157922 9781107015357 1107015359.

C. R. Rao. Information and the accuracy attainable in the estimation of statistical parameters. *Bull. Calcutta Math. Soc.*, 1945.

Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer, 2011. ISBN 978-0-387-85819-7. URL `http://dblp.uni-trier.de/db/reference/rsh/rsh2011.html#RicciRS11`.

Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, volume 20, 2008a.

Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the International Conference on Machine Learning*, volume 25, 2008b.

Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 0123694469.

Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*. IEEE Computer Society, 2008. URL `http://dblp.uni-trier.de/db/conf/cvpr/cvpr2008.html#Silpa-AnanH08`.

Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3

edition, 2011. ISBN 978-0-12-374856-0. URL http://www.sciencedirect.com/science/book/9780123748560.

Andrew T.A Wood. Simulation of the von mises fisher distribution. *Communications in Statistics - Simulation and Computation*, 23(1):157–164, 1994. doi: 10.1080/03610919408813161. URL http://www.tandfonline.com/doi/abs/10.1080/03610919408813161.

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th International Conf. Algorithmic Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.