Collaborative Filtering on a Budget

Alexandros Karatzoglou
Telefonica Research
Barcelona, Spain
alexk@tid.es

Alex Smola
Yahoo! Research
Santa Clara, CA, USA
alex@smola.org

Markus Weimer
Yahoo! Labs
Santa Clara, CA, USA
weimer@acm.org

Abstract

Matrix factorization is a successful technique for building collaborative filtering systems. While it works well on a large range of problems, it is also known for requiring significant amounts of storage for each user or item to be added to the database. This is a problem whenever the collaborative filtering task is larger than the medium-sized Netflix Prize data.

In this paper, we propose a new model for representing and compressing matrix factors via *hashing*. This allows for essentially unbounded storage (at a graceful storage / performance trade-off) for users and items to be represented in a pre-defined memory footprint. It allows us to scale recommender systems to very large numbers of users or conversely, obtain very good performance even for tiny models (e.g. 400kB of data suffice for a representation of the EachMovie problem).

We provide both experimental results and approximation bounds for our compressed representation and we show how this approach can be extended to multipartite problems.

1 Introduction

Recommender systems are crucial for the success of many online shops such as Amazon, iTunes and Netflix. Research on the topic has been stimulated by the release of realistic data sets and contests. In the academic literature, Collaborative Filtering is widely accepted as the state-of-the-art data mining method for building recommender systems. Collaborative Filtering methods exploit collective taste patterns found in user transaction or rating data that web shops can often easily collect while preserving privacy.

Matrix factorization models have attracted a large body of work, such as (Takacs et al., 2009; Rennie and Srebro, 2005; Weimer et al., 2008a; Hu et al., 2008; Salakhutdinov and Mnih, 2008). Moreover, it is one of the core techniques used in the Netflix Prize Challenge, e.g. (Bell et al., 2007; Töscher and Jahrer, 2008). Factor models are based on the notion that the predicted rating F_{ij} of item j by user i can be written as an inner product of *item factors* $M_j \in \mathbb{R}^d$ and *user factors* $U_i \in \mathbb{R}^d$ via $F_{ij} = \langle U_i, M_j \rangle$. Finding U and M can be achieved by minimizing a range of different (typically convex) distance functions $l(\langle U_i, M_j \rangle, F_{ij})$ between the predicted entries F_{ij} and the given data. The model is then used to predict the missing ratings, each corresponding to a particular (user,item) pair.

One of the key problems of factor modes is that they have *linear* memory requirements in the number of users and items: For each user i, one vector $U_i \in \mathbb{R}^d$ needs to be stored; similarly, for each item j, one vector $M_j \in \mathbb{R}^d$ needs to be stored. Most freely available datasets like the Netflix Prize data set, which with roughly 10^8 known ratings constitutes one of the biggest data sets available to academic research, can fit comfortably in the main memory of a laptop.

The scaling in terms of users and items obscures an important issue: while there may be some users and items for which large amounts of data is available, many items / users will only come with small amounts of rating / feedback. Hence it is highly inefficient to allocate an equal amount of storage for all items. This can be addressed by sparse models, such as via ℓ_1 regularization, or topic models (Porteous et al., 2008) at the expense of a significantly more complex optimization setting and rather nontrivial memory allocation procedures – we now need to store a list of sparse vectors, requiring to store the index structure itself, the footprint can only be controlled indirectly via regular-

Appearing in Proceedings of the 13^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2010, Chia Laguna Resort, Sardinia, Italy. Volume 9 of JMLR: W&CP 9.

ization parameters, and the problem persists as users and items are added to the database. Worst case, we need disk access for every user, reducing throughput to 200 users per second, assuming 5ms disk seek time.

As a consequence, many of the aforementioned models break for large scale matrix factorization problems: some companies have 10^8 customers. In computational advertising when treating each page as an entity of its own we are faced with a similar number of 'users' (Agarwal et al., 2007). This leads to an explosion in storage requirements in a naive approach.

In this paper, we introduce hashing as an alternative. Just like in the case of linear models (Weinberger et al., 2009) we use a compressed representation for the factors (in this case the two parts of the matrix). We show that it is the overall weight (the Frobenius norm) of the factors that influences approximation quality of the factorization problem, thus allowing us to store large numbers of less relevant factors and less relevant users / items easily. This also puts an end to the problem of not having enough storage space to store a sufficiently high-dimensional model.

Furthermore, we experiment with a range of different loss functions beyond the Least Mean Squares loss (in particular the ϵ-insensitive loss, a smoothed variant thereof and Hubert's robust loss) and we show that stochastic gradient descent is well applicable to large scale collaborative filtering problems.

- We present a hashing model for matrix factorization with limited memory footprint.
- We use Hubert's robust loss and an ϵ-insensitive loss function for matrix factorization, thus allowing for large margins.
- We integrate these two extensions into standard online learning for matrix factorization.

Paper structure Section 2 discusses related work, section 3 describes the general matrix factorization problem, it introduces loss functions, and it discusses online optimization methods. Section 4 shows how matrices can be stored efficiently using hashing and it provides approximation guarantees for compressed memory representations. Experimental results are given in section 5 and we conclude with a discussion.

2 Related Work

Factor models and more specifically matrix factorization methods have been successfully introduced to Collaborative Filtering and form the core of many successful recommender system algorithms. The basic idea is to estimate vectors $U_i \in \mathbb{R}^d$ for each user i and $M_j \in \mathbb{R}^d$ for every item j of the data set so that their inner product minimizes an explicit (Srebro et al., 2005) or implicit loss function between the predictions and the train data ((Hoffman, 2004) introduced a probabilistic approach to factor models). Such factor models are statistically well motivated since they arise directly from the Aldous-Hoover theorem of partial exchangeability of rows and columns of matrix-valued distributions (Kallenberg, 2005).

In *matrix factorization* the observations are viewed as a sparse matrix Y where Y_{ij} indicates the rating user i gave to item j. Matrix factorization approaches then fit this matrix Y with a dense approximation F. This approximation is modeled as a matrix product between a matrix $U \in R^{n\times d}$ of user factors and a matrix $M \in R^{m\times d}$ of item factors such that $F = UM^T$.

Directly minimizing the error of F with respect to Y is prone to overfitting and capacity control is required. For instance, we may limit the rank of the approximation by restricting d. This leads to a Singular Value Decomposition of F, which is known as Latent Semantic Indexing in Information Retrieval. Note that this approach ignores the sparsity of the input data and instead models Y as a dense matrix with missing entries being assumed to be 0, thereby introducing a bias against unobserved ratings.

An alternative is proposed in (Srebro and Jaakkola, 2003) by penalizing the estimate only on observed values. While finding the factors directly now becomes a nonconvex problem, it is possible to use semidefinite programming to solve the arising optimization problem for hundreds, at most, thousands of terms, thereby dramatically limiting the applicability of their method. An alternative is to introduce a matrix norm which can be decomposed into the sum of Frobenius norms (Rennie and Srebro, 2005; Srebro et al., 2005; Srebro and Shraibman, 2005). It can be shown that the latter is a proper matrix norm on F. Together with a multiclass version of the hinge loss function that induces a margin, (Srebro et al., 2005) introduced Maximum Margin Matrix Factorization (MMMF) for collaborative filtering. We follow their approach in this paper. Similar ideas were also suggested by (Takacs et al., 2009; Salakhutdinov and Mnih, 2008; Takács et al., 2007; Weimer et al., 2008b; Bell et al., 2007) mainly in the context of the Netflix Prize.

3 Regularized Matrix Factorization

In the following, we denote by $Y \in \mathcal{Y}^{n\times m}$ the (sparse) matrix of observations defined on an observation domain $\mathcal{Y}$, and we let U and M be d-dimensional factors such that $F := UM^\top$ should approximate Y. Whenever needed, we denote by $S \in \{0;1\}^{n\times m}$ a binary matrix with nonzero entries S_{ij} indicating whenever

Y_{ij} is observed. Finding U and M requires three components: a loss measuring the distance between F and Y, a regularizer penalizing the complexity in U and M, and an optimization algorithm for computing U and M.

3.1 Loss

In analogy to (Srebro et al., 2005) we define the loss:

$$L(F,Y) := \frac{1}{\|S\|_1} \sum_{i,j} S_{ij} l(F_{ij}, Y_{ij}) \tag{1}$$

where $l : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ is a pointwise loss function penalizing the distance between estimate and observation. A number of possible choices are listed below:

Squared error: Here one chooses

$$l(f,y) = \frac{1}{2}(f-y)^2 \text{ and}$$
$$\partial_f l(f,y) = f - y$$

ϵ-insensitive loss: It is chosen to ignore deviations of up to ϵ via

$$l(f,y) = \max(0, |y-f| - \epsilon) \text{ and}$$
$$\partial_f l(f,y) = \begin{cases} \operatorname{sgn}[f-y] & \text{if } |f-y| > \epsilon \\ 0 & \text{otherwise} \end{cases}$$

Smoothed ϵ-insensitive loss: In order to deal with the nondifferentiable points at $|y-f| = \epsilon$ one may choose (Dekel et al., 2005) the loss function

$$l(f,y) = \log(1 + e^{f-y-\epsilon}) + \log(1 + e^{y-f-\epsilon})$$
$$\partial_f l(f,y) = \tfrac{1}{1+\exp(\epsilon-y+f)} - \tfrac{1}{1+\exp(\epsilon-f+y)}.$$

Huber's robust loss: This loss function (Huber, 1981) ensures robustness for large deviations in the estimate while keeping the squared-error loss for small deviations. It is given by

$$l(f,y) = \begin{cases} \frac{1}{2\sigma}(f-y)^2 & \text{if } |f-y| \le \sigma \\ |f-y| - \frac{1}{2}\sigma & \text{otherwise} \end{cases}$$
$$\partial_f l(f,y) = \begin{cases} \frac{1}{\sigma}[f-y] & \text{if } |f-y| \le \sigma \\ \operatorname{sgn}[f-y] & \text{otherwise} \end{cases}$$

3.2 Regularization

Given the factors U, M which constitute our model we have a choice of ways to ensure that the model complexity does not grow without bound. A simple option is (Srebro and Shraibman, 2005) to use the penalty

$$\Omega[U,M] := \frac{1}{2}\left[\|U\|^2_{\text{Frob}} + \|M\|^2_{\text{Frob}}\right]. \tag{2}$$

Indeed, the latter is a good approximation of the penalty we will be using. The main difference being that we will scale the degree of regularization with the amount of data similar to (Bell et al., 2007):

$$\Omega[U,M] := \frac{1}{2}\left[\sum_i n_i \|U_i\|^2 + \sum_j m_j \|M_j\|^2\right] \tag{3}$$

Here U_i and M_j denote the respective parameter vectors associated with user i and item j. Moreover, n_i and m_j are scaling factors which depend on the number of reviews by user i and for item j respectively.

3.3 Optimization

Overall, we strive to minimize a regularized risk functional, that is, a weighted combination of $L(UM^\top, Y)$ and $\Omega[U,M]$, such as

$$R[U,M] := L(UM^\top, Y) + \lambda\Omega[U,M] \tag{4}$$

As dataset sizes grow, it becomes increasingly infeasible to solve matrix factorization problems by batch optimization. Instead, we resort to a simple online algorithm which performs stochastic gradient descent in the factors U_i and M_j for a given rating F_{ij} simultaneously. This leads to Algorithm 1. This algorithm

Algorithm 1 Matrix Factorization

Input Y, d
Initialize $U \in \mathbb{R}^{n\times d}$ and $M \in \mathbb{R}^{m\times d}$ with small random values.
Set $t = t_0$
while (i,j) in observations Y **do**
  $\eta \longleftarrow \frac{1}{\sqrt{t}}$ and $t \longleftarrow t+1$
  $F_{ij} := \langle U_i, M_j \rangle$
  $U_i \longleftarrow (1-\eta\lambda)U_i - \eta M_j \partial_{F_{ij}} l(F_{ij}, Y_{ij})$
  $M_j \longleftarrow (1-\eta\lambda)M_j - \eta U_i \partial_{F_{ij}} l(F_{ij}, Y_{ij})$
end while
Output U, M

is easy to implement since it accesses only one row of U and M at a time. Indeed, it is easy to parallelize it by performing several updates independently, provided that the pairs (i,j) are all non-overlapping.

4 Hashing

Storing U and M quickly becomes infeasible for increasing sizes of m, n and d. For instance, for $d = 100$ and a main memory size of 16 GB, the limit is reached for 20 Million users / items combined. This is very likely even for some of the most benign industrial applications. Hence, we would like to find an approximate compressed representation of U and M in the form of some parameter vectors u and m.

4.1 Compression

In the following, h, h' denote two independent hash functions with image range $\{1, \dots N\}$ where N denotes the amount of floating point numbers to be allocated in memory, as represented by the array $u \in \mathbb{R}^N$ and $m \in \mathbb{R}^N$. Moreover, denote by σ, σ' two independent Rademacher functions with image range $\{\pm 1\}$ with expected value 0 for any argument of σ and σ'.

We now construct an explicit compression algorithm which stores U and M in w as follows:

$$u_i := \sum_{(j,k):h(j,k)=i} U_{jk}\sigma(j,k) \text{ and} \tag{5}$$

$$m_i := \sum_{(j,k):h'(j,k)=i} M_{jk}\sigma'(j,k) \tag{6}$$

In other words, we add random entries in U and M to u and m respectively. This scheme works whenever only a small number of matrix values are significant. We reconstruct U and M via

$$\tilde{U}_{ij} := u_{h(i,j)}\sigma(i,j) \text{ and } \tilde{M}_{ij} := m_{h'(i,j)}\sigma'(i,j). \tag{7}$$

This allows us to reconstruct F_{ij} via

$$\tilde{F}_{ik} = \sum_j u_{h(i,j)} m_{h'(k,j)} \sigma(i,j)\sigma'(k,j). \tag{8}$$

Please note that the use of hashing not only allows us to compress the model, but also facilitates using *arbitrary types for i and j,* as long as a hash function is supplied. This allows e.g. to refer to a user through its email address and to a book through its ISBN,

While the hasing approximation may seem overly crude, we prove that it generates high quality reconstruction of the original inner product below. This analysis is then followed by an adjusted variant of the stochastic gradient descent algorithm for matrix factorization.

4.2 Guarantees

Mean: We begin by proving that the compression is in expectation accurate. The reconstruction of U can be computed as:

$$\tilde{U}_{ij} = \sum_{(a,b):h(a,b)=h(i,j)} \sigma(a,b)\sigma(i,j)U_{ab}$$

Since all $\sigma(a,b)$ are drawn independently, it follows that only the term $\sigma^2(i,j)$ survives which proves the claim. Clearly the same holds for $\mathbf{E}[\tilde{M}_{ij}] = M_{ij}$. To see that the same property holds for $\tilde{F}_{ij}$ we use the fact that $\tilde{U}$ and $\tilde{M}$ are *independent* random variables. Hence we have $\mathbf{E}[\tilde{F}_{ij}] = F_{ij}$.

Variance: To compute the latter we need to compute the expected value of $\tilde{F}^2_{ik}$. This is somewhat more tedious since we need to deal with all interactions of terms. We have

$$\tilde{F}^2_{ik} = \sum_{j,j'} \sum_{\substack{h(a,b)=h(i,j),h'(c,d)=h'(k,j)\\ h(e,f)=h(i,j'),h'(g,h)=h'(k,j')}} \sigma(a,b)\sigma(i,j)\sigma'(c,d)\sigma'(k,j)\times$$
$$\sigma(e,f)\sigma(i,j')\sigma'(g,h)\sigma'(k,j')U_{ab}M_{cd}U_{ef}M_{gh}$$

We know that $\mathbf{E}_\sigma[\sigma(a,b)\sigma(c,d)] = \delta_{(a,b),(c,d)}$. Hence, when taking expectations with respect to σ, σ' and h, h' we can decompose the sum into the following contributions:

For $(j = j')$ the expectation is nonzero only if $(a,b) = (c,d)$ and $(e,f) = (g,h)$. In this case the sum over the hash functions is nonzero only whenever $h(a,b) = h(i,j)$ and $h(e,f) = h(g,h)$ which yields the following contribution to be summed over all j:

$$\left[\frac{1}{N}\sum_{a,b} U^2_{ab} + \left[1 - \tfrac{1}{N}\right] U^2_{ij}\right]\left[\frac{1}{N}\sum_{a,b} M^2_{ab} + \left[1 - \tfrac{1}{N}\right] M^2_{kj}\right]$$

For $(j \neq j')$ we may pair up $(a,b) = (i,j), (c,d) = (k,j), (e,f) = (i,j'), (g,h) = (k,j')$ to obtain the contribution

$$\sum_{j \neq j'} U_{ij}M_{kj}U_{ij'}M_{kj'} = F^2_{ik} - \sum_j U^2_{ij}M^2_{kj}.$$

For $(j \neq j')$ when pairing up $(a,b) = (i,j'), (c,d) = (k,j), (e,f) = (i,j), (g,h) = (k,j')$ we obtain

$$\sum_{j \neq j'} \frac{1}{N} U_{ij'}M_{kj}U_{ij}M_{kj'} = \frac{1}{N}\left[F^2_{ik} - \sum_j U^2_{ij}M^2_{kj}\right]$$

The same holds when combining $(a,b) = (i,j), (c,d) = (k,j'), (e,f) = (i,j'), (g,h) = (k,j)$. Note that the factor in both cases arose from the fact that the terms are only nonzero for a hash function collision, that is, for $h(i,j) = h(i,j')$ and $h'(k,j) = h'(k,j')$ respectively.

Finally, when combining $(a,b) = (i,j'), (c,d) = (k,j'), (e,f) = (i,j), (g,h) = (k,j)$ we again obtain the same contribution, albeit now with a multiplier of $\frac{1}{N^2}$ since this is only nonzero if both h and h' have collisions. In summary, the expectation of $\tilde{F}^2_{ij}$ is given

by

$$\begin{aligned}\mathbf{E}[\tilde{F}_{ij}^2] = &\left[1+\tfrac{1}{N}\right]^2\left[F_{ik}^2 - \sum_j U_{ij}^2 M_{kj}^2\right] + \tfrac{d}{N^2}\left\|U\right\|^2\left\|M\right\|^2 \\ &+ \tfrac{N-1}{N^2}\left[\left\|U\right\|^2\left\|M_k\right\|^2 + \left\|U_i\right\|^2\left\|M\right\|^2\right] \\ &+ \left[1-\tfrac{1}{N}\right]^2\sum_j U_{ij}^2 M_{kj}^2 \\ <&F_{ij}^2 + \tfrac{1}{N}\left[\left\|U\right\|^2\left\|M_k\right\|^2 + \left\|U_i\right\|^2\left\|M\right\|^2 + 2F_{ik}^2\right] \\ &+ \tfrac{d}{N^2}\left\|U\right\|^2\left\|M\right\|^2 - 3N\sum_j U_{ij}^2 M_{kj}^2\end{aligned}$$

Here inequality follows by dropping all negative terms in N^{-2} and by using convexity to bound $N^{-2}F_{ij}^2$. The matrix norms are all Frobenius norms. Ignoring the last term the variance of $\tilde{F}_{ij}$ is bounded by

$$\begin{aligned}\mathrm{Var}[\tilde{F}_{ik}] <&\tfrac{1}{N}\left[\left\|U\right\|^2\left\|M_k\right\|^2 + \left\|U_i\right\|^2\left\|M\right\|^2 + 2F_{ik}^2\right] \\ &+ \tfrac{d}{N^2}\left\|U\right\|^2\left\|M\right\|^2 \qquad (9)\end{aligned}$$

Default Inner Product: It is worthwhile checking guarantees regarding the default matrix value F_{ik} for given storage vectors u, m. Assume that $\left\|u\right\|_\infty, \left\|m\right\|_\infty \leq C$. In this case we have by default that matrix entries F_{ik} satisfy

$$\Pr_{h,h',\sigma,\sigma'}\left\{\left|F_{ik}\right| \geq \delta\right\} \leq 2\exp\left(-\frac{d\delta^2}{2C^4}\right). \qquad (10)$$

This guarantee follows directly from Hoeffding's inequality and the fact that individual terms in the sum $\sum_j \sigma(i,j)\sigma'(k,j)u_{h(i,j)}m_{h'(k,j)}$ are all bounded by C^2 and that moreover the summands are independent random variables with respect to σ, σ' (we did not evaluate $\sigma(i,j)$ and $\sigma(k,j)$ previously).

Obviously this bound does *not* hold for previously seen arguments of σ, σ'. This is actually desirable — we *want* to have generalization for users and items where we observed data before.

Finally note that it is possible to improve (10) simply by replicating dimensions r times and by scaling down each coordinate in U and M respectively by $r^{-\frac{1}{2}}$. This significantly improves the Hoeffding bound in (10), albeit at a slight increase in variance. While the first three terms in (9) remain unchanged (replication and scaling cancel in the matrix norms), the last term changes from $\frac{d}{N^2}$ to $\frac{dr}{N^2}$. Moreover, computation increases by a factor of r.

4.3 Joint Storage

The variance bound still assumes that we have separate storage for each of the matrices. This is inefficient, since it requires us to state a trade-off between space for U and M separately. Indeed, we can use the same joint storage for U and M which yields the compression relation for a storage vector w:

$$w_i := \sum_{h(a,b)=i} U_{ab}\sigma(a,b) + \sum_{h'(a,b)=i} M_{ab}\sigma'(a,b) \qquad (11)$$

Moreover, the reconstruction is identical to separate storage allocation via

$$\tilde{U}_{ij} = w_{h(i,j)}\sigma(i,j) \text{ and } \tilde{M}_{ij} = w_{h'(i,j)}\sigma'(i,j). \qquad (12)$$

As previously we have that $\mathbf{E}[\tilde{U}_{ij}] = U_{ij}$ and $\mathbf{E}[\tilde{M}_{ij}] = M_{ij}$. The expected value of $\tilde{F}_{ij}$ has a small correction term, though: when taking expectations with respect to σ, σ, only those terms containing the same arguments in both σ and σ' remain. Hence we have that

$$\mathbf{E}_{\sigma,\sigma'}\left[\tilde{F}_{ik}\right] = \sum_j U_{ij}M_{kj} + \delta(h'(k,j), h(i,j))\sum_j U_{ij}M_{kj}$$

Taking expectations over h and h' yields that

$$\mathbf{E}\left[\tilde{F}_{ik}\right] = \left[1+\tfrac{1}{N}\right]F_{ik}.$$

Bounding the variance is considerably more tedious, since we now have to take care of collisions in both construction and reconstruction. Nonetheless, the variance for $\tilde{F}_{ik} = \sum_j \tilde{U}_{ij}\tilde{M}_{kj}$ is still $O(N^{-1})$.

4.4 Algorithm

As a consequence of the compressed storage, we need to modify Algorithm 1 to operate on a joint storage location. This is fairly straightforward — we simply substitute $\tilde{U}, \tilde{M}$ for U and M.

Algorithm 2 Compressed Matrix Factorization

Input Y,
Initialize $w \in \mathbb{R}^N$ with small random values.
Set $t = t_0$
while (i,k) in observations Y **do**
  $\eta \longleftarrow \frac{1}{\sqrt{t}}$ and $t \longleftarrow t+1$
  $F_{ik} := \sum_j \sigma(i,j)\sigma'(k,j)w_{h(i,j)}w_{h'(k,j)}$
  $\gamma := \eta\partial_{F_{ik}} l(F_{ik}, Y_{ik})$
  $\mu := (1-\eta\lambda)$
  for $j = 1$ **to** d **do**
    $w_{h(i,j)} \leftarrow \mu w_{h(i,j)} - \gamma\sigma(i,j)\sigma'(k,j)w_{h'(k,j)}$
    $w_{h'(k,j)} \leftarrow \mu w_{h'(k,j)} - \gamma\sigma(k,j)'\sigma(i,j)w_{h(i,j)}$
  end for
end while
Output w

Algorithm 2 has a number of advantages over a two-stage procedure which first computes U and M and subsequently compresses the values:

Data set	Users	Movies	Ratings	Scale
EachMovie	61265	1623	2811717	1-6
MovieLens	6040	3900	1000209	1-5

Table 1: Data set statistics

1. Since we are optimizing directly the compressed representation the approximation error is considerably smaller.
2. We never need to allocate full memory for U and M which can be infeasible for very large problems.
3. The number of users and items may vary over time. In our construction, this is a non-issue — simply start optimizing for additional i and k to accommodate for this fact.
4. We may even change the dimensionality of the model on the fly: change the summation range from d to d'.

5 Experiments

In this section, we describe the empirical evaluation conducted to assess the performance of the two main contributions of this paper: (1)The ϵ-insensitive and the Huber loss function and (2)The memory bound matrix factorization model.

5.1 Data

We report experiments conducted on the EachMovie and MovieLens datasets summarized in Table 1. As we are investigating the *relative* performance of the losses and the compression scheme, we restrict ourselves to basic preprocessing of the data by removing the global, user and movie means. Further increases in absolute performance are conceivable based upon the techniques presented in e.g. (Bell and Koren, 2007) and (Potter, 2008).

For the EachMovie dataset, we randomly extracted 10 ratings of each user to form the *test set* for this user. The remaining known ratings were used as the training set. We repeated the procedure ten times with different test sets to obtain 10 folds of the data. For the Movielens dataset, we used the 5 train-test splits generated by the script provided by the GroupLens research lab[1] with the dataset.

5.2 Evaluation of the loss functions

We compared the performance of the smooth ϵ-insensitive loss and the Huber loss functions to the widely adopted squared error loss. We performed extensive parameter tuning on one of the folds of each data set and report averaged results on the remaining folds. We tuned the learning rate η, the regularization parameter λ and in the case of the ϵ-insensitive and Huber loss ϵ and σ respectively.

[1] `http://www.grouplens.org/`

Squared	ϵ-insensitive	Huber
	MovieLens	
$\mathbf{0.857} \pm 0.006$	0.859 ± 0.004	$\mathbf{0.857} \pm 0.004$
	EachMovie	
1.177 ± 0.003	$\mathbf{1.161} \pm 0.007$	1.180 ± 0.005

Table 2: RMSE values for the tuned models for different train/test splits for the MovieLens and EachMovie datasets.

The mean values of the test set RMSE for this set of experiments along with the standard deviation are given in Table 2. The ϵ-insensitive loss significantly outperforms the Squared and Huber loss for the EachMovie data (p-value=0.0091). For the MovieLens data, the losses perform in the same range without significant differences. The good performance of the ϵ-insensitive loss can be attributed to its resilience against the noise prevalent in recommender systems data. The ϵ-insensitive loss is by design more suited to this type of data. Note that on EachMovie, higher RMSE scores are to be expected than on MovieLens due to the wider rating scale.

5.3 Evaluation of the model compression

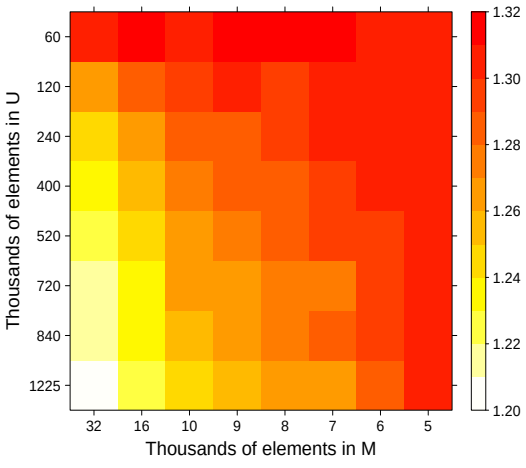
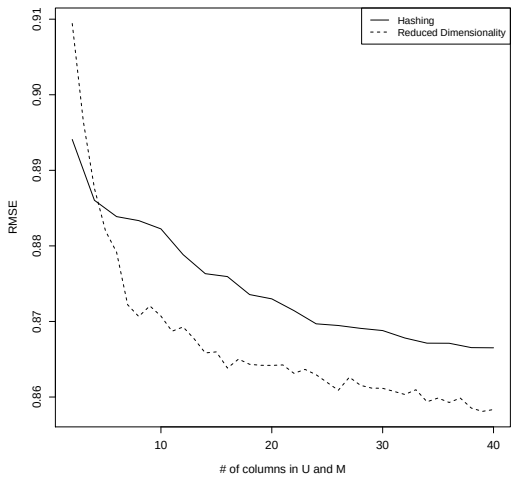


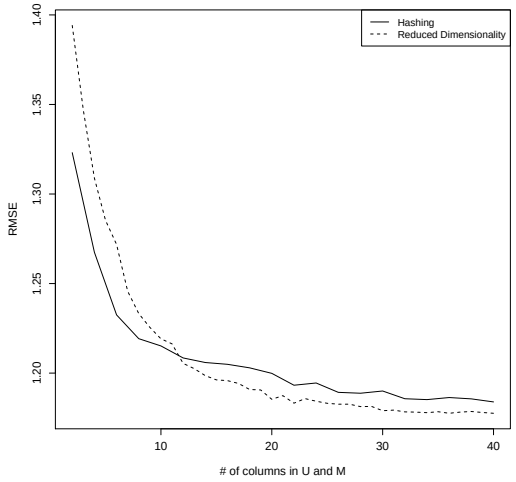
Figure 1: Heatmap of RMSE values on the test set for the EachMovie data for different hashed model sizes.

The main goal of this evaluation is to examine the impact that different constraints on the available memory have on the performance of the prediction as compared to the full, unconstrained model. To this end, we var-

ied the size of the available memory in terms of the number of elements used for the user and item matrix representations, yielding a compression ratio between $1:1$ and $1:20$. We fixed the parameters ($\lambda = 0.04$ and $\eta = 0.005$) and restricted us to the squared error to isolate the impact of the model compression.



(a)



(b)

Figure 2:
Line plot of the RMSE's for different numbers of columns of a simple matrix factorization model and an equally sized hashing-model for the MovieLens (a) and the EachMovie (b) data.

Figure 1 shows a heatmap representation of the results for different available memory for U and M. For the heatmap representation we use 10 passes over the data using the algorithm described in 2 and a dimensionality of 20 for U and M. Please note that the dimensionality of U and M has no influence on the memory requirements when using the hashed model.

Surprisingly only very little ($\sim 3\%$) performance is lost even when the model is compressed to almost half its original size for e.g. 720k elements for U and 16k elements for M resulting in a total memory requirement of just above 2MB. We furthermore observe a smooth and gradual increase in the RMSE values for smaller models. The standard deviation of the RMSE scores over the 10 runs is less than 0.01.

The variance over the ten runs on different data in all experiments is surprisingly low, especially given the fact that we are optimizing a non-convex function. The low variance may mean that we always reach the same local minimum or that this minimum is indeed a global one.

Different Model Compression Schemes: We now compare minimizing memory space usage by using a hashed model versus decreasing the dimensionality d of the U and M matrices by varying the number of columns used. We perform a set of experiments taking an unhashed matrix model and varying the number of columns used in U and M from 40 to 1. We compare performance to a hashed model with a fixed dimensionality of 40, but varying amounts of compression to match the memory requirements of the lower dimensional models.

Results are shown in Figure 2: we observe that the hashed model outperforms the simple model for dimensionalities below 10. As can be seen, hashing performs especially well under strong memory constraints. This is particularly important in real-world applications where the number of users can be in the order of hundreds of millions. In these cases, even storing 10 user and item factors quickly becomes infeasible, while hashing provides a natural trade-off between the memory requirements and the model performance. Additionally, the memory requirements of the hashed model are constant and do not scale with the number of users or items as is the case with the lower dimensional, unhashed models.

Influence of σ, σ': We conducted experiments to investigate the importance of obtaining an unbiased estimator by means of the Rademacher hash functions σ, σ'. Setting them to the same constant adds a bias to the estimation towards positive predictions. Experiments on data are in sync with this observation: For unnormalized data, a constant value of σ, σ' *seems* to perform better than a proper hash function. However, this effect is an artifact as such a model is outperformed by a model using the proper Rademacher functions on normalized data as well as, in preliminary experiments, by a model learning a per-user and per-item bias on the unnormalized data.

6 Conclusion

In this paper, we introduced a new factor model for collaborative filtering that differs in two main aspects from the state-of-the-art: (1) We introduced two new loss functions: the ϵ-insensitive loss function and the Huber Loss to factor models in order to achieve a large margin model. (2) We introduce the notion of hashing to bound the memory requirements of the model.

The benefits of the memory bound model are twofold: first, it allows the model to be scaled to bigger datasets on large servers, and to still big datasets on smaller machines; second, the constant memory and hashing requirements facilitates many optimizations in the storage and use of the model such as using higher dimensional models, arbitrary user and item identifiers and seemless addition and removal of users and items.

The ϵ-insensitive loss and the Huber loss functions not only provide for a theoretically desirable large margin model, they also perform better on at least one of the tested datasets compared to the often used least squares loss function. The hashed memory bound model exhibits a smooth gradual trade-off between the model size and performance which can be adjusted given the computational resources.

References

D. Agarwal, A. Z. Broder, D. Chakrabarti, D. Diklic, V. Josifovski, and M. Sayyadian. Estimating rates of rare events at multiple resolutions. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, 2007.

R. Bell and Y. Koren. Improved neighborhood based collaborative filtering. In *The Netflix-KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*. ACM Press, 2007.

R. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the neflix prize. Technical report, AT&T Labs, 2007.

O. Dekel, S. Shalev-Shwartz, and Y. Singer. Smooth ϵ-insensitive regression by loss symmetrization. *Journal of Machine Learning Research*, 6:711–741, 2005.

T. Hoffman. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the Eighth IEEE International Conference on Data Mining, 2008. ICDM*. IEEE, 2008.

P. J. Huber. *Robust Statistics*. John Wiley and Sons, New York, 1981.

O. Kallenberg. *Probabilistic symmetries and invariance principles*. Springer, 2005.

I. Porteous, E. Bart, and M. Welling. Multi-HDP: A non parametric bayesian model for tensor factorization. In D. Fox and C. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1487–1490. AAAI Press, 2008.

G. Potter. Putting the collaborator back into collaborative filtering. In *The 2nd-Netflix-KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*. ACM Press, 2008.

J. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd International Conference on Machine Learning ICML*, 2005.

R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20 NIPS*, Cambridge, MA, 2008. MIT Press.

N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning ICML*, pages 720 – 727. AAAI Press, 2003.

N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In P. Auer and R. Meir, editors, *Proc. Annual Conf. Computational Learning Theory*, number 3559 in Lecture Notes in Artificial Intelligence, pages 545–560. Springer-Verlag, June 2005.

N. Srebro, J. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17 NIPS*, Cambridge, MA, 2005. MIT Press.

G. Takács, I. Pilászy, B. Németh, and D. Tikk. Major components of the gravity recommendation system. *SIGKDD Explorations Newsletter*, 9(2):80–83, 2007. ISSN 1931-0145.

G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

A. Töscher and M. Jahrer. The bigchaos solution to the netflix prize 2008. Technical report, commendo research & consulting, 2008.

M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofirank - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20 NIPS*, Cambridge, MA, 2008a. MIT Press.

M. Weimer, A. Karatzoglou, and A. Smola. Improving maximum margin matrix margin factorization. *Machine Learning*, 72(3), September 2008b.

K. Weinberger, A. Dasgupta, J. Attenberg, J. Langford, and A. Smola. Feature hashing for large scale multitask learning. In L. Bottou and M. Littman, editors, *International Conference on Machine Learning*, 2009.