# Machine Learning project: Recommendation systems

Bernardo Cardoso Cordeiro, Gauthier Damien, Hugo Richard

January 31, 2017

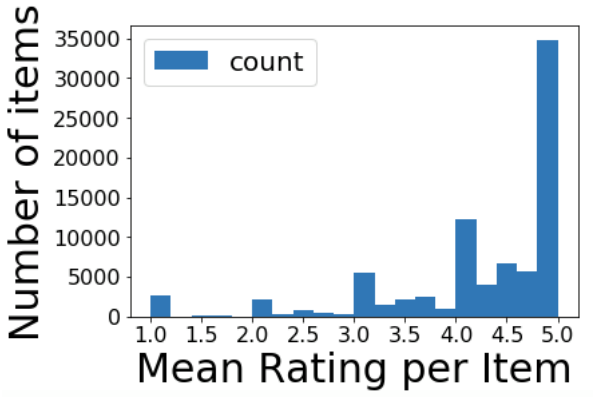


Figure 1: Mean rating per item

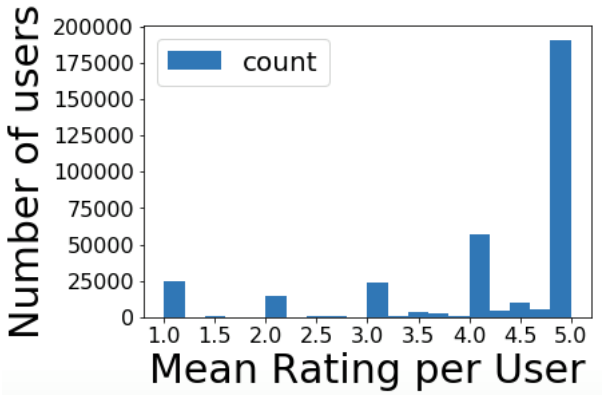


Figure 2: Mean rating per users

# 1 Introduction

The dataset we used, to be found at http://jmcauley.ucsd.edu/data/amazon/, contains 500,176 reviews of music instruments on Amazon website, including 339,231 users and 83,046 different items. The following figures 1 and 2 show the mean rating per users and items.

In order to have a matrix not too sparse we decided to restrict the dataset in the following fashion:

- Keep only users that have rated at least 5 items
- Keep only items that were rated by at least 20 users

# 2 Content-based Approach

## 2.1 State of the art

A classical way to address the recommendation task of recommending a set of items to a user is to focus on items properties and recommend an item similar

to others items a given user bought in the past. This is what is commonly called content-based methods. One first look at the properties of the items, be a set of actors for a movie, the year the movie was made, the genre etc., and compute the similarity between these items. These features may be available, but in practice are often hidden.

In the Amazon dataset, we do not have much information about the item type, its price, or even the color. However, a feature describing the item in one sentence is available, and could be useful to characterize the items on our set. In this respect, we calculated a similarity between items using the Term Frequency - Inverse Document Frequency score of their description as a design, and then calculated a cosine-similarity on these scores, reflecting the similarity between items.

Now that we have similarities, we go back to the recommendation task: for a given item i and a user u we want to predict the rating $R_u,_i$ of this specific user for this specific item. This prediction will be done computing the sum of the ratings given by the user on the items similar to i. Each ratings is weighted by the corresponding similarity $s_i,_j$ between items i and j. We can denote the prediction $R_u,_i$ as:

$$R_{u,i} = \frac{\sum_{allsimilaritems,N}(s_{i,N}) * (R_{u,N})}{\sum_{allsimilaritems,N}(|s_{i,N}|)}$$

Basically, this approach tries to capture how the active user rates the similar items. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range.

### 2.2 Implementation

Due scaling issues for this method, the implementation was realized using a sample of the original Amazon dataset we selected. Others approaches described below are more suited to high scale problems.

After training the model on 75% of the data and testing it on the remaining 25%, we measured the prediction score using the Mean Square Error as a metric.

We obtained a MSE of 2.46 with this method, which is better than a random model but still not tremendous. Looking at the similarity matrix, we notice that most similarity measures between items are very low so we guess that this method may not be well adapted to this dataset (either the items are very different or more features about the item characteristics may be useful.)

## 3 Collaborative Filtering using Matrix Factorisation

### 3.1 State of the art

Latent Factor Model was created by Simon Funk, a participant of Netflix Prize in 2006. Since then, it has been widely used in many prediction tasks. Briefly speaking, this model tries to find latent factors for each user and item and it performs by projecting user features and item features into low-dimensional

spaces using Singular Value Decomposition (SVD) to factorize the matrix. A simple form of this model can be written as:

$$\hat{r_{u,i}} = p_u^T.q_i$$

where $q_i$ is a vector associated with each item i, $p_u$ is also a vector associated with each user u. For a given item i, the elements of $q_i$ measure the extent to which the item possesses those factors; for a given user u, the elements of $p_u$ measure the extent of interest the user has in items that are high on the corresponding factors. Therefore, their dot product $p_u^T.q_i$ denotes the overall interest of the user in characteristics of the item. Besides, we should also add the regularization terms to the optimization problem as:

$$min \sum_{(u,i \in Train)} [r_{u,i} - \hat{r_{u,i}}]^2 + \lambda(||p_u||^2 + ||q_i||^2)$$

For this project, we applied this method using two optimization methods : Stochastic Gradient Descent (SGD) and Alternative Least Squares (ALS). Typically, using SGD is faster than ALS when the matrix is sparse but ALS has the advantage to be easily parallelized so it scales well to large datasets.

### 3.1.1 Matrix Factorization using Stochastic Gradient Descent

To update the matrices P and Q, we first used stochastic gradient descent where looping over every observation in the training set we update Q and P on the way:

$$q_{i+1} = q_i + \gamma(e_{u,i}.p_u - \lambda q_i)$$
$$p_{u+1} = p_u + \gamma(e_{u,i}.q_i - \lambda p_u)$$

where $\gamma$ is the learning rate and $\lambda$ is the regularization term. The error $e_{u,i}$ is the difference between the actual rating and the predicted value.

$$e_{u,i} = r_{u,i} - p_u^T.q_i$$

### 3.1.2 Matrix Factorization using Alternating Least Squares

When looking at the matrix factorization problem, it is clearly non-convex. Hence a difficult optimization problem. However, the ALS algorithm turns the problem into a quadratic problem. It works when fixing the values of $p_u$ and optimizing on $q_i$, then fixing $q_i$ and optimizing on $p_u$. Differentiating the loss function w.r.t each parameter, equating to zero and solving for the respective paramter yields the following update equations (derived in [ZWSP08]):

$$p_i = A_i^{-1}V_i with A_i = Q_{I_i}Q_{I_i}^T + \lambda n_{p_i} E and V_i = Q_{I_i}R_{i,I_i}^T$$
$$q_j = A_j^{-1}V_j with A_j = P_{I_j}P_{I_j}^T + \lambda n_{q_j} E and V_j = P_{I_j}R_{I_j,j}^T$$

where k a number of latent features, E is the $kxk$-identity matrix, $Q_{I_i}$ and $P_{I_j}$ are the sub-matrices of Q and P with the appropriate columns selected. R is the ratings matrix. Finally $n_{p_i}$ denotes the number of items user i has rated and $n_{q_j}$ denotes the number of users that rated item j.

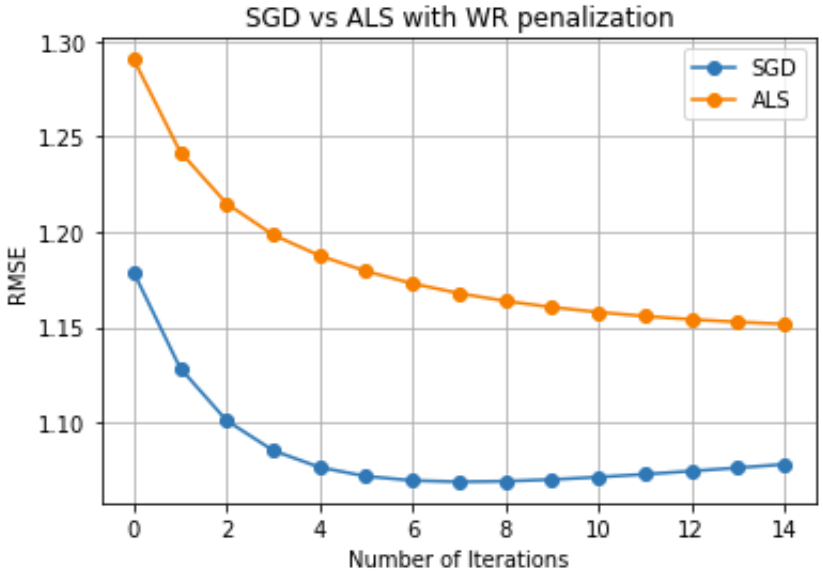


Figure 3: SGD vs ALS with WR penalization

### 3.2 Implementation

Training our algorithms on Amazon data, it appears clearly that the implementation using SGD is way faster than ALS to perform 15 iterations and converges faster.

The plots 3, 4 and 5 illustrate the performance of the two algorithms on a sample matrix of size 1762x2966:

We finally obtain a MSE of 1.07 for SGD ans 1.17 for ALS after 15 iterations, which is a significant improvement over the first implemented method.

## 4 Application of low rank factorization of determinant point processes to predict the rating of items

In [GPK16], the authors describe an innovative method for performing recommendation using analogies with statistical physics. Their work enables to predict the next item likely to be bought by a customer given a set of already bought items. We will first describe their approach, then explain how we adapted their work to our specific problem and discuss the results obtained.

### 4.1 State of the art

#### 4.1.1 Recommendation, statistical physics and determinant distribution processes

The key problem in recommendation is to learn the user's interest. In order to achieve this we would like to recommend products that the user might like with

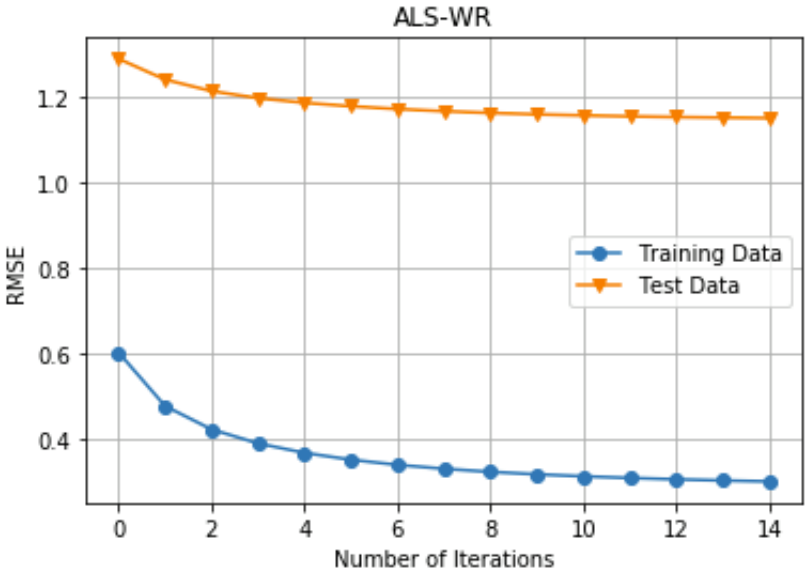


Figure 4: ALS-WR

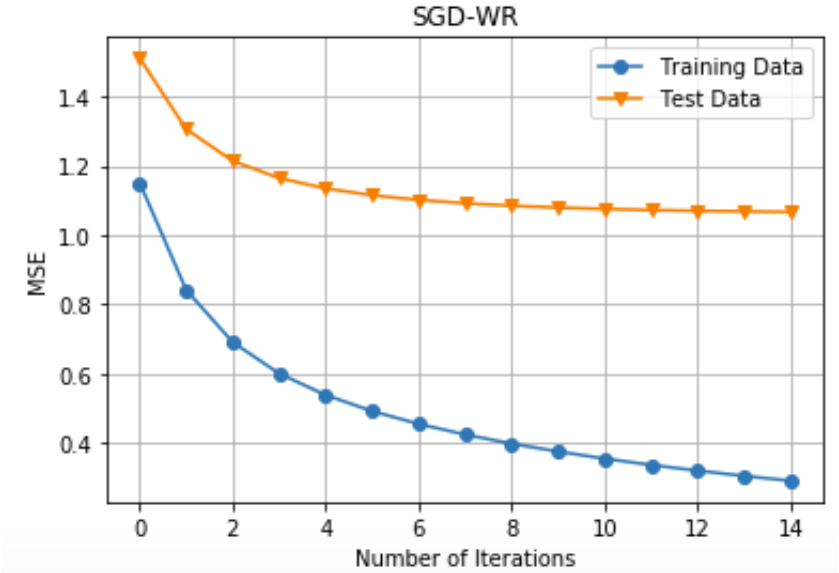


Figure 5: SGD-WR

high probability but that are also diverse from what has been bought already. For example, if a user has just bought a computer, it is very unlikely that he will buy a similar one. He will rather most likely buy a bag for his computer.

For $U$ users, we observe the set of items bought by each user: $A_u = \{items - bought - by - user - u\}$. Each items are taken in a catalog of size $M$. We would like to know what is the probability that a given user $u*$ buys a particular item $i*$ in the catalog given the set of item that he has already bought $A*$. What we want to predict is therefore: $p(u* - buys - i* | u* - has - bought - A*)$.

In physics, determinant distribution processes are used to model the distribution of fermions in thermal equilibrium. An important property of fermions is that they cannot share the exact same quantum state (described by 4 quantum numbers) i.e. they are diverse. We will therefore use determinant distribution processes to model the distribution of users' baskets so that we will enforce diversity in our prediction.

The probability of seeing a basket of items $A$ is given by:

$$p(A) = \frac{det(L_A)}{det(L+I)}$$

In the definite positive matrix $L$ of size MxM, $L_{ii}$ represents the importance of item $i$ whereas $L_{ij}$ measures the similarity between item $i$ and item $j$. $L_A$ is given by the restriction of $L$ to the items in basket $A$.

### 4.1.2 Learning of the determinant distribution processes

We can learn $L$ using the log-likelihood given by:

$$p(A_1, .., A_U, L) = \sum_{u=1}^{U} log(det(L_{A_u})) - Ulog(L+I)$$

This function can be particularly difficult to optimize. A low rank factorization can be used for $L$ so that $L = V.V^T$ where $V$ is of size MxK where $K << M$ in order to speed-up computations. The restriction on $K$ is that no users can have more than $K$ items in its basket. We naturally have $L_A = V_A.V_A^T$.

The log-likelihood can be optimized using stochastic gradient ascent. A regularization is added to avoid overfitting. The function to optimize $p(A_1, .., A_U, L) = f(V)$ is given by:

$$f(V) = \sum_{u=1}^{U} log(det(L_{A_u})) - Ulog(L+I) - \frac{\alpha}{2} \sum_{i=1}^{M} \lambda_i ||V[i,:]||^2$$

where $\lambda_i = 1/C_i$ with $C_i$ the number of occurrence of item $i$ in the purchases of all users.

The gradient of $p(A_1, .., A_U, L) = f(V)$ writes as:

$$\frac{df}{dik} = \sum_{A \in \mathbb{A}_i} L_A^{-1}[i,:].V_A[:,k] + L_A^{-1}[:,i].V_A[:,k] - U(B[i,:].V[:,k] + B[:,i].V[:,k])$$

where $B = I_M - V(I_K + V^T.V)^{-1}V^T$ and $\mathbb{A}_i = \{\text{baskets A that contains i}\}$.

We see here the benefit from using a low rank factorization: B fast to compute because we only have a KxK matrix to inverse instead of a MxM matrix in the full rank case. Computing the inverse of $L_A$ is also fast because the number of items in a given basket $A$ is often small.

### 4.1.3 Prediction given by the distribution

Once $L$ is learned we want to predict the probability that item $i$ will be in the basket of a user $u$ in the future given its current basket $A$. Formally we want to compute $p(A \cup i|A)$. It can be shown that:

$$p(A \cup i|A) \propto L^A[i,i]$$

We have $L^A = V^A.(V^A)^T$ where $V^A = V_{\overline{A}}.Z^A$ with $Z^A = I - V_A^T.(V_A.V_A^T)^{-1}.V_A$.

$V_{\overline{A}}$ is the matrix $V$ restricted to items not in basket $A$.

## 4.2 Implementation and results

### 4.2.1 Adaptation of the problem

We need to modify the behavior of the algorithm presented above such that it can predict the rating given by a given user $u*$ to a given item $i*$. In other words we want to compute the probability that a user $u*$ gives the rate $r_{u*i*}$ to item $i*$.

The work above gives us a measure of similarity between an item and the current basket. The very concept of the basket does not exist here but we can recreate something similar. Grouping each items by grades, we can try to find the next item similarly rated.

We therefore define for each user $u$ and each rating $g$, the basket $A_{u,g}$ which is the set of items graded $g$ by user $u$. We will then assign to $r_{ui}$ the grade that is the most likely.

$$r_{ui} = argmax_g p(A_{u,g} \cup i|A_{u,g})$$

### 4.2.2 Implementation and results

The function to optimize is non-convex and difficult to optimize. The nesterov accelerated gradient descent gives a local minimum but fails to increase sharply the log likelihood.

On top of restriction already imposed to our dataset we add that no users should have more than 100 items in its basket (condition imposed by the value of $K$, note that this is not a big restriction on this dataset).

We obtain a mean square error of 5 which is worse than average. This is due to a poor optimization of the log-likelihood and to the fact that we categorize our data using ratings therefore not using any distance between ratings. This method also does not take into account the subjectivity of users with regards to rating or the item's popularity. It may explain why so much high ratings are observed.

# 5 Factorization meets the neighborhood- a multifaceted collaborative filtering model

For this algorithm, the paper by Korben [Kor08] was used. This model takes into account several factors that, together compose the entire model. Basically, we can divide the model in three parts:

- Baseline estimates
- Neighborhood estimates
- Latent factors

## 5.1 Baseline estimates

The baseline consists of finding, for each user and for each item, their respective inherent deviations in relation to the average rating. For example, a user $u$ might always score movies slightly better than the average, while an item $i$ might always receive scores that are a bit smaller than the average. This can be represented with the following equation:

$$b_{ui} = \mu + b_u + b_i$$

A practical example, as given in [1], is the following: suppose we want the baseline estimate for Titanic, as rated by user Joe. The average rating over all movies, $\mu$, is 3.7. Furthermore, Titanic is a good movie, and is thus rated 0.5 stars above the average. However, Joe is a very critical user, and hence tends to rate movies 0.3 stars lower than the average. With these values, the baseline estimate for Titanic's rating, as given by Joe, would be 3.9 (3.7 + 0.5 – 0.3). To calculate the values of $b_u$ and $b_i$ , the following minimization problem needs to be solved:

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

This can be done by iterating through all known ratings, and updating the values of $b_u$ and $b_i$ in relation to each rating, using a stochastic gradient descent approach:

- For $r_{ui} \in \mathcal{K}$, $e_{ui} = r_{ui} - b_{ui}$:
  - $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_1 \cdot b_u)$
  - $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_1 \cdot b_i)$

Where $\gamma$ is the learning rate. With these values calculated, we can calculate $b_{ui}$ for all the user-item pairs available for us, which will be used later on in the final model.

## 5.2 Neighborhood estimates

This part of the model takes care of finding similarities between items. With these values in hand, it checks, for each user, how they scored items that are similar to the one being considered. The logic behind this is that similar items tend to get similar scores.

According to the paper, other models that had been tried before tended to use interpolation weights relating an item $i$ to the items in a user-specific neighborhood $S^k(i;u)$ . The authors, however, wanted to facilitate global optimization, and thus opted for using global weights $w_{ij}$ between pairs of items, that are learned through optimization.

Moreover, implicit feedback is also considered in the model. This can be represented, for example, by the fact a user clicked on a product or not, their mouse movements, and so on. Thus, for the products that received implicit feedback from a user, offsets $c_{ij}$ are used to also fine-tune the predictions. Therefore, the model at this point is:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj}) \cdot w_{ij} + \sum_{j \in N(u)} c_{ij}$$

Where $R(u)$ is the set of items to which the user $u$ gave ratings, and $N(u)$ is the set of items to which user $u$ gave implicit feedback of some kind. For the dataset we had, the implicit feedback is just the presence or not of a rating. Which means the implicit feedback matrix is the same as the ratings matrix, with all ratings replaced by ones. Due to the fact interpolation is no longer used, it is possible to decouple the values of $b_{ui}$ and $b_{uj}$, such that $b_{ui}$ can be written as $\mu + b_u + b_i$, with $b_u$ and $b_i$ being parameters we need to optimize, and $b_{uj}$ are the baselines calculated in the previous section. Moreover, since unlikely item-item relations don't influence the results too much, instead of using $R(u)$ and $N(u)$, the authors choose to calculate the top-k items most similar to the one being analyzed, and only predict using the items that are both in the sets $R(u)$ or $N(u)$, and that are also present in the top-k most similar items. Thus, we have:

$$\hat{r}_{ui} = \mu + b_u + b_i + |R^k(i;u)|^{-\frac{1}{2}} \sum_{j \in R^k(i;u)} (r_{uj} - b_{uj}) \cdot w_{ij} + |N^k(i;u)|^{-\frac{1}{2}} \sum_{j \in N^k(i;u)} c_{ij}$$

Where $N^k(i;u) = N(u) \cap S^k(i)$ and $R^k(i;u) = R(u) \cap S^k(i)$. A final detail is that, for each pair of items $i, j$, the cosine similarity is calculated as per the following formula:

$$cos\_sim(i,j) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

This is done by using a scikit-learn's CSR sparse matrix, and doing matrix multiplications, which makes these calculations relatively fast to do. With the results in hand, it is easy to get the top-k most similar items $S^k(i)$.

## 5.3 Latent factors

Lastly, users and items are modeled as vectors of latent factors, which is the basic algorithm for matrix factorization. Each user is represented by a vector $p_u$, and each of the items by vectors $q_i$. This tries to identify hidden attributes of the items and users and integrate them into the predictions.

For the version used here however, the users are modeled as a sum of two parts: a vector $p_u$ for the latent factors, and a vector of the form $|N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$. This second component takes the implicit feedback a user provides, and uses it as an offset, which gives additional information about a given person. The prediction, as given by the latent factors is therefore:

$$\hat{r}_{ui} = b_{ui} + q_i^T \Big( p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \Big)$$

## 5.4 Putting it together

Putting it all together, we get the following equation for the predictions:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \Big( p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \Big) + |R^k(i;u)|^{-\frac{1}{2}} \sum_{j \in R^k(i;u)} (r_{uj} - b_{uj}) \cdot w_{ij} + |N^k(i;u)|^{-\frac{1}{2}} \sum_{j \in N^k(i;u)} c_{ij}$$

Where $b_{uj}$ is given by the baseline section. The other parameters are optimized by using iterations of the kind:

- For $r_{ui} \in \mathcal{K}$, $e_{ui} = r_{ui} - \hat{r}_{ui}$:
  - $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
  - $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
  - $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j) - \lambda_7 \cdot q_i)$
  - $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
  - $\forall j \in N(u)$: $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |N(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
  - $forall j \in R^k(i;u)$: $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot (|R^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij})$
  - $forall j \in N^k(i;u)$: $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot (|N^k(i;u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij})$

The values of the hyperparameters $\gamma_1, \gamma_2, \gamma_3, \lambda_6, \lambda_7, \lambda_8$ were slightly changed in relation to what the authors used in the paper. Most notably, $\gamma_1, \gamma_2$ were set to 0.001 instead of 0.007 in the paper due to the fact that these values were causing the algorithm to diverge towards infinity. By giving these learning rates a smaller value, the algorithm started converging again. Also, one iteration goes through all the entries in the dataset. Not surprisingly, increasing the number of iterations also improves the final results.

## 5.5 Results

### 5.5.1 Toy Dataset

For the toy dataset, data was generated randomly. Given:

- $n_{users}$ – desired number of users
- $n_{items}$ – desired number of items
- $p$ – percentage of ratings available

It is possible to generate $p * n_users * n_items$ ratings randomly, which are used to test the model. For the tests on this toy dataset, 50 users, 100 products, and p = 0.2 was used. An obvious downside to this approach of generating data randomly is that all the relations between items and users are also random, which might hurt the overall performance of the algorithm. This can be easily seen when doing Leave-One-Out cross validation. By checking the performance of the model using this scheme, a MSE of 2.29 was found (which is extremely high) compared to a MSE of 0.27 for the training error. Both of them ran the calculations for 1000 iterations.

The graphs 6 show the obtained MSE for a given number of iterations of the algorithm. The 'average' line represents the MSE given by simply taking the mean over the training set, and using it to predict the test set.

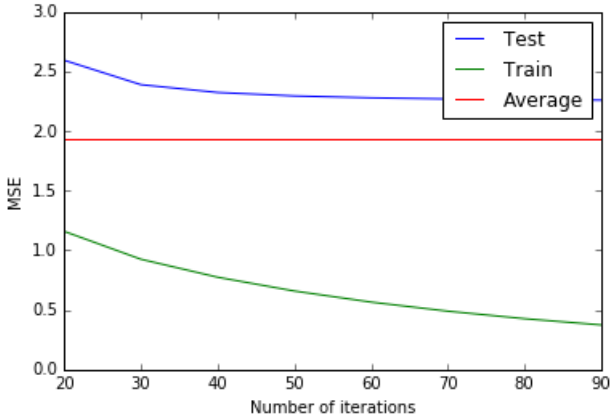


Figure 6: Obtained MSE for the Toy dataset

#### 5.5.2 Amazon Musical Instruments Dataset

When using the Musical Instruments dataset from Amazon, due to the size of the data, it was impossible to evaluate the model through a Leave-One-Out scheme. Hence, the dataset was simply separated into a train and test sets, with 75% of the data being used for training. Here, the number of iterations plays an important role in the performance of the algorithm. If only a few iterations are performed then a high MSE is obtained. However, by increasing the number of iterations, the performance of the model becomes better and better. This comes at a price, though, as the time taken to finish them also goes up considerably. Nevertheless, it is necessary so that a reasonable enough model can be built.

The graphs 7 show the obtained MSE for a given number of iterations of the algorithm. The 'average' line represents the MSE given by simply taking the mean over the training set, and using it to predict the test set. A point worth mentioning is that, these results might still be improved through optimization of the hyperparameters (the learning rates, number of iterations of the baseline calculations, number of latent factors, etc), so these are not the best possible values for the MSE.

## References

[GPK16] Mike Gartrell, Ulrich Paquet, and Noam Koenigstein. Low-rank factorization of determinantal point processes for recommendation. *arXiv preprint arXiv:1602.05436*, 2016.

[Kor08] Yehuda Koren. Factorization meets the neighborhood: a multi-faceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[ZWSP08] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *In-*

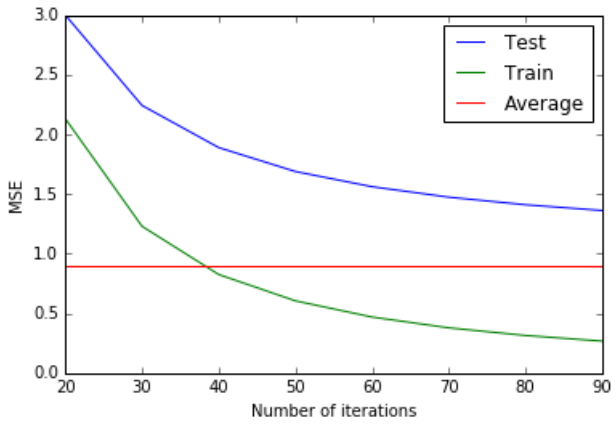


Figure 7: Obtained MSE for the Amazon Musical Instrument dataset

*ternational Conference on Algorithmic Applications in Management*, pages 337–348. Springer, 2008.