United We Stand: Combinations for Collaborative Filtering

Mariyana Koleva
ETH Zurich
Switzerland
Email: kolevam@student.ethz.ch

Karim Labib
ETH Zurich
Switzerland
Email: labibk@student.ethz.ch

João Lourenço Ribeiro
ETH Zurich
Switzerland
Email: ljoao@student.ethz.ch

***Abstract*—A critical part of scientific discovery is the communication of research findings to peers or the general public. Mastery of the process of scientific communication improves the visibility and impact of research. While this guide is a necessary tool for learning how to write in a manner suitable for publication at a scientific venue, it is by no means sufficient, on its own, to make its reader an accomplished writer. We also describe the rules for submission in the computational intelligence laboratory. This guide should be a starting point for further development of writing skills.**

I. Introduction

Recommender systems have gained increasing popularity in the last 10-15 years, mainly due to their immediate market applications. The huge selection of products online offer retailers the opportunity to attract a wide-range of clients. However, it also means that users can no longer be expected to browse through everything available until they find the film, song or book they like the most. Good recommendations are key to both making the correct products available to the appropriate audience, but also to creating a unique, personalised experience which will attract new clients. These ideas are fundamental for many leading online businesses, notably Netflix [1], Amazon [2].

In this paper, we concentrate on the collaborative-filtering approach to recommender systems, which relies on previous user experiences (ratings), rather than pre-defined user profile [some ref about the two types]. We use a dataset with ratings from 10000 users on 1000 films.

Our main contributions are two-fold: we first combine latent factor models (Singular Value Decomposition (**SVD**), Stochastic Gradient Descent for matrix factorization (**SGD**), Restricted Boltzmann Machines (**RBM**) and neighbourhood models (user- and item-based k-Nearest Neighbors (**kNN**) to produce a robust recommender system which optimizes the Root Mean Sqaured Error (RMSE) of predicted ratings on X[how many films] from Blaa to Bllaa. Second, we show that the addition of a more complex learning rate heuristic on SGD and RBM improves the score significantly, to X.

In Section II we describe the idea and the implementation of our systems. We present the results from the local evaluation of the models and from the validation set on Kaggle in Section III. Finally, in Section IV we interpret the improvements we achieved and discuss potential weaknesses and future work.

II. Models and Methods

All work is done on the dataset provided for the project. The data is a collection of *(user, film, rating)* tuples. For training, we have 1176953 ratings available. The online evaluation on Kaggle is performed on another 1176953 ratings. We were able to see the prediction accuracy on 50% of that online dataset and our goal was to improve that as it is believed to be representative of the full dataset.

A. Data Model

The provided dataset can be most naturally seen as an $N \times M$ matrix, with N being the number of users ($N = 10000$), and M the number of films ($M = 1000$). If every user had rated every film, the matrix would have 10^7 entries. However, in this realistic scenario, this is not the case and our matrix is quite sparse. From here stems the difficulty of the problem - we attempt to predict more than a million ratings based on a very incomplete dataset.

There are two main views which yield different methods for tackling this problem.

Latent factor models: These models attempt to explain film ratings by users as being influenced by a set of unknown (also called *latent*) factors.

One approach characterizes users and films by low-dimensional feature vectors in a latent feature space. The rating of a given film by a given user is modelled as the dot product between the corresponding user and film feature vectors. The different methods aim to infer the underlying film and user feature vectors from the available ratings. This is done most successfully through matrix factorization techniques and approximations. As examples of this approach, we discuss SVD and SGD.

It is also possible to predict missing ratings through neural networks which are composed by visible and latent units. In particular, we discuss RBM. At a high level, for each user-film pair whose rating is missing, the neural network learns a probability distribution over the possible ratings (1 to 5) from the known ratings. The prediction is then simply the expected value of this distribution.

Neighbourhood-based models: Another approach to the collaborative filtering problem is to view the ratings as features of either the users (user-based) or the films (item-based). In the former case, a user A is represented as a sparse feature vector of their ratings and we would base our prediction on other users who have rated those films similarly. In the latter scenario, we would use the similarity between the new film and the films already rated by A to estimate the new rating. Each film is a feature vector of all users' ratings. We review an implementation of both in the next subsection.

B. Algorithms

We investigated three algorithms which concentrate on solving the latent-factor model problem and two which work on the neighbourhood model:

SVD: Given the latent factor model described above, the rating matrix is assumed to be produced by the multiplication of a users' matrix and the films' matrix which are all on the same feature space, namely that consisting of the latent factors. The most natural way to infer the users and films representation is using a matrix factorization approach so SVD comes directly to mind [3]. The main issue is that SVD works on full matrices and as mentioned above, this is not the case. Hence, we need to impute the missing values before we can proceed. We chose the following imputation strategy for a missing rating r_{ij}

$$r_{ij} = \lambda \text{AvgUser}[i] + (1-\lambda)\text{AvgFilm}[j] \tag{1}$$

where $\lambda \in [0,1]$, AvgUser$[i]$ and AvgFilm$[j]$ are the average ratings of user i and movie j respectively. Imputation in general introduces noise and leads to sup-optimal results [needs some refs here]. We hope to minimize the noise and over-fitting by looking for a low-rank (k) approximation of the rating matrix.

We experimented with different values for λ and numbers of latent features. We concluded that the lowest RMSE on a probe set was accomplished using 13 features and $\lambda = 0.25$

Despite the validation strategies, we expect the results achieved with SVD to be suboptimal due to the large number of missing values, and the inaccuracy introduced by the imputation of those values. We use this algorithm as a baseline and we try to improve it with better strategies for matrix factorization or alternative approaches of finding the latent factor representation.

SGD: The main idea of the algorithm is to avoid the need of filling in the missing values in the rating matrix.

Instead of using the standard SVD which requires a full matrix, we use only the available ratings in a Stochastic Gradient Descent method. The regularized (to avoid over-fitting) objective function to minimize is as follows:

$$\begin{aligned}\min_{p,q,b_u,b_m} \sum_{(i,j)\in T} &(r_{ij} - (\mu + b_u(i) + b_m(j) + p_i^T \cdot q_j))^2 \\ &+ \lambda(\|p_i\|^2 + \|q_j\|^2 + b_u(i)^2 + b_m(j)^2)\end{aligned} \tag{2}$$

T is the set of observed ratings (pairs (i,j) of user i rating movie j), r_{ij} is the known rating of user i to movie j, b_u and b_m are the bias vectors of users and movies respectively. It has been pointed out in [3] that the bias vectors lead to higher accuracy. The intuition being that the variation in the observed ratings can not be only explained by the latent vectors of users and movies but also need to consider effects and biases specific for users and movies (i.e some users tend to rate higher than others). In order to predict an unknown rating of user i to movie j

$$r_{pred} = \mu + b_u(i) + b_m(j) + p_i^T \cdot q_j \tag{3}$$

The parameters are modified during the algorithm by looping through all ratings in the training set for several epochs. For each data point (i,j), the magnitudes of the corresponding parameters are updated by a magnitude proportional to a learning rate γ in the opposite direction to the gradient of the objective function as follows:

$$\begin{aligned} p_i &\leftarrow p_i + \gamma(e_{ij}q_j - \lambda p_i) \\ q_j &\leftarrow q_j + \gamma(e_{ij}p_i - \lambda q_j) \\ b_u(i) &\leftarrow \gamma(e_{ij} - \lambda b_u(i)) \\ b_m(j) &\leftarrow \gamma(e_{ij} - \lambda b_m(j)) \end{aligned} \tag{4}$$

The approximation should be much more representative of the data as it only uses known ratings. The algorithm was very sensitive to hyper-parameters so those were optimized on both a local validation set and the online Kaggle score.

Restricted Boltzmann Machines (RBM's): RBM's are neural networks consisting of a layer of visible units and a layer of hidden units. Each unit has a stochastic binary random variable associated with it. There may be undirected connections between hidden and visible units, but no hidden-hidden or visible-visible connections are allowed. Thus, an RBM takes the form of a bipartite graph, where each layer corresponds to a different bipartition. These networks are a restricted version of Boltzmann machines (hence the name), introduced in [4].

Recently, RBM's have found applications in collaborative filtering, as illustrated in [5]. A very nice overview of RBM's for collaborative filtering can be found in [6]. Our approach is based on these two works, although there are some differences.

We first build an RBM with 5000 visible units v_i^k, for $k \in \{1,\ldots,5\}$ and $i \in \{1,\ldots,1000\}$, and L hidden units $h_1,\ldots,h_L$, where L is a hyper-parameter of the algorithm. Furthermore, for each hidden unit h_j we add connections to every visible unit v_i^k. Each connection between h_j and v_i^k

has an associated weight W_{ij}^k. Each hidden unit h_j has an associated bias b_j and each visible unit v_i^k has an associated bias b_i^k. Finally, each unit can be in state 0 or 1.

The probability that a given unit is turned on (i.e. its associated state is 1) is a logistic function of the states of the units it is connected to. In our case we have

$$\begin{aligned} p(v_i^k = 1|h) &= \frac{\exp(b_i^k + \sum_{j=1}^{L} h_j W_{ij}^k)}{\sum_{l=1}^{5} \exp(b_i^l + \sum_{j=1}^{L} h_j W_{ij}^l)} \\ p(h_j = 1|V) &= \frac{1}{1 + \exp(-b_j - \sum_{i=1}^{1000} \sum_{k=1}^{5} v_i^k W_{ij}^k)}. \end{aligned} \tag{5}$$

To train our RBM, we divide the users into small batches. For each user u in a batch, we set $v_i^k = 1$ if and only if user u gave rating k to film i. We then compute $p_j = p(h_j = 1|V)$ when V is set according to each of the users in the batch and record v_i^k and p_j for each triple (i, j, k).

After going through each batch, we update the weights and biases. The learning rules are

$$\begin{aligned} W_{ij}^k &\leftarrow W_{ij}^k + \gamma(< v_i^k h_j >^+ - < v_i^k h_j >^-) \\ b_i^k &\leftarrow b_i^k + \gamma(< v_i^k >^+ - < v_i^k >^-) \\ b_j &\leftarrow b_j + \gamma(< h_j >^+ - < h_j >^-), \end{aligned} \tag{6}$$

where γ is the learning rate, $< v_i^k h_j >^+$ is the average (over all users in the batch) of $v_i^k \cdot p_j$, $< v_i^k >^+$ is the average of v_i^k and $< h_j >^+$ is the average of p_j. Furthermore, $< v_i^k >^-$ and analogous quantities correspond to averaged probabilities of finding the corresponding units turned on simultaneously when the neural network is running at equilibrium. These quantities are hard to compute exactly, but they can be approximated via Gibbs sampling (see [7]). In our algorithm we use a single step of Gibbs sampling. Increasing the number of steps had no effect on the prediction score of our RBM.

Note that we train only a single RBM. In [5], the method presented consists of building a different RBM for each user, which all share the same weights and biases. In particular, they keep only connections from hidden units to visible units corresponding to observed ratings of a particular user.

The prediction algorithm is the same in our method and in [5] and [6]. Namely, for a fixed user u, we set $v_i^k = 1$ if and only if user u ranked film i with rating k. We then compute $\hat{p}_j = p(h_j = 1|V)$ for $j = 1, \ldots, L$. If we want to predict the rating of film i, we compute

$$p(v_i^k = 1|\hat{p}) = \frac{\exp(b_i^k + \sum_{j=1}^{L} \hat{p}_j W_{ij}^k)}{\sum_{l=1}^{5} \exp(b_i^l + \sum_{j=1}^{L} \hat{p}_j W_{ij}^l)}, \tag{7}$$

for $k = 1, \ldots, 5$. This yields a probability distribution over the possible ratings for movie i. Our prediction is the expected value of this distribution, $\sum_{k=1}^{5} k \cdot p(v_i^k = 1|\hat{p})$.

We make use of the built-in BernoulliRBM algorithm in the *sklearn* Python library to create our RBM and train it. Furthermore, we modified the source code to add an heuristic for the learning rate of the RBM, which improved its performance.

User-based k-Nearest Neighbors: User based neighborhood methods are considered one of the most intuitive methods for collaborative filtering. It consists of two steps. First, the algorithm computes the similarity between different users through some similarity measure. Second, to predict the rating of user u for a particular movie m, the ratings of the most similar k users who have rated movie m are aggregated according to their similarity measure with user u to estimate the required rating. The similarity measure, which according to [6] was found to be the most accurate was the Pearson correlation measure. We denote the Pearson correlation between two users u_1 and u_2 by $w(u_1, u_2)$ and it is calculated as follows:

$$\frac{\sum_{j \in C}(R(u_1, j) - \overline{R}_C(u_1)) \cdot (R(u_2, j) - \overline{R}_C(u_2))}{\sqrt{\sum_{j \in C}(R(u_1, j) - \overline{R}_C(u_1))^2} \sqrt{\sum_{j \in C}(R(u_2, j) - \overline{R}_C(u_2))^2}} \tag{8}$$

C being the set of co-rated items between users u_1 and u_2. $R(u, j)$ is the rating of user u to movie j. $\overline{R}_C(u)$ is the average rating of user u on the set of movies C. The higher the correlation, the more similar the two users u_1 and u_2. The pearson correlation can be shown [6] to be equivalent to the cosine based similarity of the centered rating matrix (averages of users are subtracted from the corresponding ratings). This allows for easier and faster computation of correlation between every pair of users using sklearn. The best k is chosen by calculating the error of the algorithm on a validation set for different values of k. Finally, to predict the unknown rating of a user u for a particular movie i, we compute an averaged weighted sum of the ratings of the k nearest neighbors to user u as follows:

$$R(u, i) = \overline{R}(u) + \frac{\sum_{u_k \in N(u)} w(u, u_k)(R(u_k, i) - \overline{R}(u_k))}{\sum_{u_k \in N(u)} w(u, u_k)} \tag{9}$$

where $N(u)$ is the set of the most similar k users who rated item i.

Item-based k-Nearest Neighbors: The item based kNN is considered to be the dual approach of the user based approach discussed above. Rather than finding similar users, the algorithm finds similar movies based on the ratings of all users. As above, a similarity measure such as the Pearson correlation is used to assess the similarity between each pair of movies. The set of users who rated movies i and j are used to measure as two vectors and the correlation is measured between them in a similar fashion as shown in equation (8). in order to predict the rating of user u for movie i, This is computed as follows:

$$R(u, i) = \overline{R}(i) + \frac{\sum_{j \in k\_R(u)} w(i, j)(R(u, j) - \overline{R}(j))}{\sum_{j \in k\_R(u)} w(i, j)} \tag{10}$$

where $\overline{R}(i)$ is the average rating of movie i. $k\_R(u)$ is a set of movies for which we know the rating of user u. $w(i,j)$ is the similarity measure of movie i and movie j. We can choose the size of the set k_R as we wish. In out experiments we found that the best approach is to use all rated films by a user - usually between 70 and 250.

C. *The bold driver heuristic*

A significant improvement to the running time and efficiency of our algorithm came from incorporating the bold driver heuristic for the learning rate in both SGD and RBM. This technique works as follows: Suppose we start with learning rate λ. After each epoch we compute the current error on the validation set and compare it with the error of the previous epoch. If the error increased from one epoch to the other, we decrease λ to $d\lambda$, where $d \in [0,1]$ is a parameter we must set. On the other hand, if the error decreased, we increase λ to $(1+c)\lambda$, where $c \geq 0$ is a parameter we must also set. Intuitively, the bold driver heuristic adapts the learning rate to the surface we are trying to maneuver.

It is interesting to note that usual implementations of the bold driver heuristic also erase an update when it increases the error. In our implementation we opted to keep all such updates. Through experimentation we came to the conclusion that erasing "bad" updates did not lead to any increase of performance for our methods. Should we test this further? In the Results section we report the convergence speed of the different methods.

III. Results

In this section we report the most interesting results achieved during our experiments. Our goal was to understand how the different hyper-parameters affect the prediction accuracy and what the best combination of algorithms is. We describe the validation protocol and results and in Section IV we derive conclusions.

A. *Parameter optimization and validation*

The general protocol for training and testing followed the steps below:

1) Choose a random subset of the train samples for a local validation set if not already chosen. This was in most cases 5% of the data, so 58847 samples and was only done by the first algorithm, so that all algorithms are trained on the same data.
2) Train each individual algorithm on the leftover train dataset. In some cases, such as in the item-base kNN, the validation set was used to automatically choose the optimal k in the predefined range. In the iterative algorithms like SGD and RBM, we used the validation set as an early indication on whether the error is really falling with every epoch. We would often optimise certain parameters using this validation data, such as number of features to user or algorithm iterations **(see results for details)**.

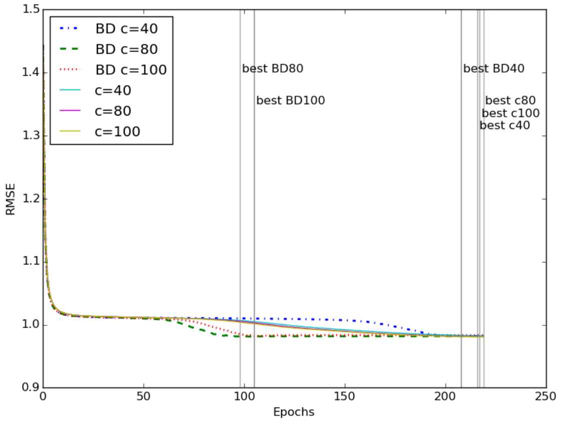

Figure 1. RMSE decreases over time (epochs) for the SGD algorithm with varying number of the C parameter and with and without the use of the Bold Driver learning rate heuristic. Optimal error 0.98066 achieved after 217 epochs with the C100, and 0.9811 achieved after 98 epochs with the BD C80 setup. Results are on a randomly generated dataset of 58847 user-film pairs. The vertical lines mark the epochs when the minimum error was achieved.

3) Each algorithm generates predictions for the local validation set and the test set which would be uploaded on Kaggle.
4) The predictions from all algorithms are combined using linear regression. The regressor is trained on the predictions generated for local validation set for which we have ground truth. The coefficients which fit the results best are then used to generate a final prediction from all individual predictions for the online test dataset.

B. *SGD Bold Driver and C parameter validation*

The accuracy of the SGD algorithm for matrix factorisation is strongly linked to the number of iterations of the algorithms. The C parameter which controls the penalty for wrong answers also affects both convergence speed and accuracy. Figure 1 and Figure 3 show how the different setups affect the number of epochs to achieve optimal values and the optimal values. **In our final experiments, we used the SGD with Bold Driver and C=80 as this provided the best trade-off of accuracy (0.9811 on validation) against running time (less than 100 epochs).Did we?**

C. *RBM Bold Driver*

Some comments here.

D. *kNN validation maybe?*

E. *Comparison between baseline and optimal algos*

Most important chart here

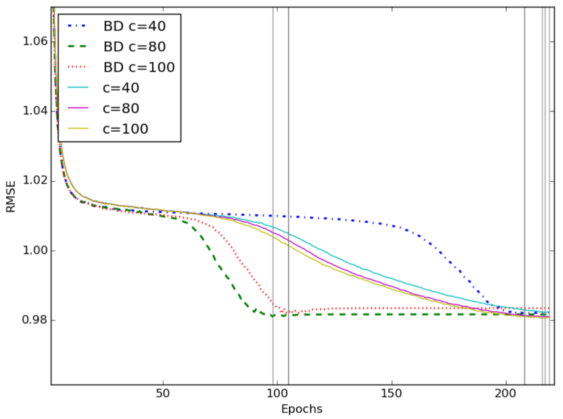

Figure 2. Zoomed-in version of Figure 1. Dashed graphs show results for algorithm with bold driver heuristic. Those converge much faster than the pure algorithm at the cost of 0.0004 compared to the optimal algorithm without the heuristic.

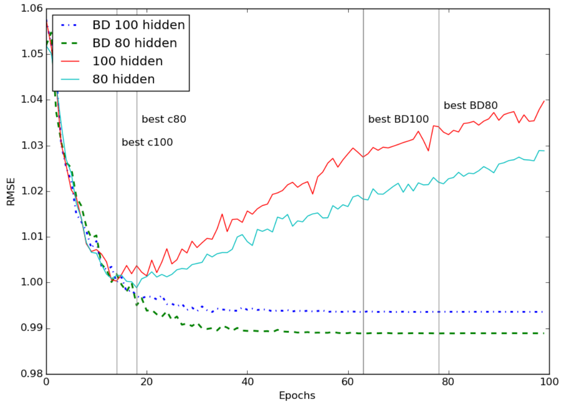

Figure 3. Effect of Bold Driver heuristic on RMSE error of RBM with 80 and 100 hidden units. The optimal error achieved without Bold Driver is 0.99883 with 100 units after 18 epochs and 0.98885 with Bold Driver after 78 epochs.

IV. Discussion

The aim of our experiments was to investigate if we could achieve better overall accuracy by combining different algorithms which were known to perform well on the Netflix dataset and collaborative filtering problems in general. The intuition was that latent factor models and neighbourhood based models capture different features of the underlying signal of the data. This was confirmed by bla bla bla **when we report the scores of the pure SGD and SVD against the combination we should discuss it here**.

Our second contribution was the investigation of the Bold Driver heuristic. It halved the number of epochs required for the SGD algorithm to achieve the optimal values (**and RBM?** at a small cost. This was 0.0005 on the validation set, but less than **Some number here?** on the online Kaggle dataset. The convergence and the accuracy of the iterative algorithm depends the C value as well, but as we can see in Figure 3, all algorithms with the Bold Driver heuristic achieve faster convergence than those without from the beginning regardless of the value of C. The Bold Driver SGD with C=40 has a later second drop (after 180^{th} epoch), compared to the same with $C = 80$ or 100, however it is still faster than each of the pure algorithms at the cost of < 0.001.

A. *Submission System*

The deadline for submitting your project report is Friday, 22 June 2012. You need to submit:

- PDF of paper.
- Archive (`.tar.gz` or `.zip`) of software. Please do not forget to include author information in the source archive.

Important: Please check the submission instructions on the webpage as it is the most updated instructions.

V. Summary

The aim of a scientific paper is to convey the idea or discovery of the researcher to the minds of the readers. The associated software package provides the relevant details, which are often only briefly explained in the paper, such that the research can be reproduced. To write good papers, identify your key idea, make your contributions explicit, and use examples and illustrations to describe the problems and solutions.

Acknowledgements

The author thanks Christian Sigg for his careful reading and helpful suggestions.

References

[1] X. Amatriain and J. Basilico, “Netflix recommendations: beyond the 5 stars (part 1),” *Netflix Tech Blog*, vol. 6, 2012.

[2] G. Linden, B. Smith, and J. York, “Amazon. com recommendations: Item-to-item collaborative filtering,” *IEEE Internet computing*, vol. 7, no. 1, pp. 76–80, 2003.

[3] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” 2009.

[4] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[5] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted boltzmann machines for collaborative filtering,” in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 791–798.

[6] G. Louppe, “Collaborative filtering: Scalable approaches using restricted boltzmann machines,” Master’s thesis, Université de Liège, Liège, Belgique, 2010.

[7] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.