An Alternative Approach to Bootstrap Aggregation-Based Collaborative Filtering

Group: Backpropagaters
Josua Cantieni, Sarah Kamp, Franz Knobel, Silvia La
Department of Computer Science, ETH Zurich, Switzerland
{josuac, skamp, knobelf, sla2}@student.ethz.ch

***Abstract*—In this paper we present a novel approach to the collaborative filtering problem. We have explored various popular baseline models and developed a new approach based on the Embedding-Dot-Product (EDP) algorithm. We consider different optimization techniques using Stochastic Gradient Descent (SGD), Adam, and the One-Cycle Policy and combine the results with bootstrap aggregation. The results show that we can reduce the RMSE significantly, not only compared to our baseline algorithms, but also compared to the used models each on its own. We discuss and analyze our results and consider future improvements to our approach.**

I. Introduction

Many applications use recommender systems to analyze the user's pattern of interest in items to provide personalized recommendations. Especially commercial applications such as media streaming services and online stores rely on recommender systems to suggest content with high personal value to the user, enhancing user experience and thus increasing the company's revenue.

Collaborative filtering represents a type of algorithmic selection. It exploits collective data from many users to generalize across users and possibly across items. This technique can be used to build recommender systems that give suggestions to a user on the basis of the likes and dislikes of similar users.

We present BACF, an alternative approach to bootstrap aggregation-based collaborative filtering. It extends the traditional Embedding-Dot-Product (EDP) algorithm[1] using two different optimizers, namely Stochastic Gradient Descent (SGD) and Adam, each using a different strategy to adapt the learning rate (LR). The resulting rating predictions of these two algorithms are combined by an approach made popular in 1994 by Breiman[2], known as *bagging* or *bootstrap aggregation*, where the two prediction matrices are averaged to receive a combined result matrix. BACF does not only apply this method on different models, but also explores the possibility to similarly combine results from the same model using different parameters.

We compared BACF to three different baseline algorithms: K-Nearest Neighbors (KNN), Singular Value Decomposition (SVD), and Alternating Least Squares (ALS).

II. Data

We obtained our data set from the ETH Zurich Computational Intelligence Lab 2020 Collaborative Filtering Kaggle project page[1]. It contains 1'176'952 integer ratings in the interval [1, 5]. Each entry represents a user rating an item and is identified by user and movie ID. There are a total of 10'000 users and 1'000 items. The given data set only contains 11.8% of all ratings. We denote these ratings given by the data set as *known ratings*.

Looking at the data, we have observed that the majority of users have rated less than 20% of all items and that the majority of items were rated by less than 17% of users. Furthermore, we have seen that the given ratings are not uniformly distributed. 88.9% of known ratings are in the interval [3, 5] with 5 being the rating with the highest occurrence.

III. Models and Methods

The collaborative filtering problem can also be interpreted as follows: Given an incomplete matrix $A \in \mathbb{R}^{mxn}$, where m is the number of users and n the number of items, predict values for all missing ratings, based on the given entries.

A. Evaluation Metric

The Root-Mean-Squared Error (RMSE) was used as evaluation metric. Assume that $r_{u,i}$ denotes the actual rating a user u provided for an item i, and that $\hat{r}_{u,i}$ denotes the rating predicted by an algorithm, for the same entry. Let Q be the set of matrix entries to be predicted, consisting of tuples *(user, item)*. The RMSE is then computed as

$$RMSE = \sqrt{\frac{\sum_{u,i \in Q}(r_{u,i} - \hat{r}_{u,i})^2}{|Q|}} \tag{1}$$

B. Evaluation Methods

1) Submitting to Kaggle: The Kaggle submission page asks for a list of 1'176'952 ratings (~13% of all unknown ratings), none of which occur in the initial data set, and calculates a public score (RMSE), based on approximately 50% of the submitted ratings. Which half of the submitted data is evaluated for the public leaderboard is undisclosed.

[1] https://inclass.kaggle.com/c/cil-collab-filtering-2020

2) Cross Validation: In addition to submitting ratings to Kaggle, cross validation was used to estimate how accurately our predictive models will perform in practice. For $0 < x < 100$, cross validation splits the given data uniformly at random into a training set, containing $(100 - x)\%$ of the known ratings and a validation set, containing the remaining $x\%$. The training set is used for training our models, outputting a prediction matrix. The validation set is used to assess the prediction matrix using an evaluation metric. Splitting the data into disjoint training and validation sets ensures that the model does not train on the validation set. This helps to detect and prevent overfitting on the whole set. We have chosen $x = 10$, and to further improve accuracy, cross validation was repeated several times using the same prediction model, parameters and input data, resulting in one RMSE per repetition. The final RMSE approximation is then calculated as the mean error over all runs.

C. Baseline Algorithms

The following section presents three baseline algorithms for collaborative filtering.

1) KNN: KNN[3] can be used to exploit the similarity in item preferences of users. The key technique of this algorithm is to find the k-most similar users (neighbors) for a given user u based on items that u has already rated[4]. These neighbors probably have a similar taste to u.

Our baseline calculates the similarity $s \in [-1, 1]$ between two users u and $\hat{u}$ based on the Pearson correlation (2), where $s = 1$ denotes a high and $s = -1$ a low similarity. $r_{u,i}$ represent the rating of item i of user u, similarly $r_{\hat{u},i}$ represents the rating of user $\hat{u}$. $\bar{r}_u$ and $\bar{r}_{\hat{u}}$ denote the mean of all ratings of the corresponding user. The k-nearest neighbors are the k users with the highest similarity.

$$s = \frac{\sum_{i=1}^{n}(r_{u,i} - \bar{r}_u)(r_{\hat{u},i} - \bar{r}_{\hat{u}})}{\sqrt{\sum_{i=1}^{n}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i=1}^{n}(r_{\hat{u},i} - \bar{r}_{\hat{u}})^2}} \tag{2}$$

For every item rated by the k-nearest neighbors we weight the corresponding rating by its similarity s with user u. The weighted ratings are then added together and normalized. This gives us the rating predictions of items for user u.

KNN cannot predict ratings for items not rated by the neighbors, thus the resulting prediction matrix is very sparse, so we replaced the missing ratings with the mean of all ratings of the input data.

2) SVD: Collaborative filtering using SVD takes advantage of feature reduction. It takes a matrix $A \in \mathbb{R}^{m \times n}$ and decomposes it into three matrices $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$, so that $A = U\Sigma V$. Using dimension reduction on Σ, one can approximate A by keeping $k \leq$ *rank(A)* latent factors. In our SVD baseline algorithm, we replaced missing ratings in A with the adjusted item mean. That is, for a missing rating $r \in [1, 5]$ of user u and item i, initialize r to be the mean of all known ratings for i. If user u tends to give rather low or high ratings, i.e., the mean of all known ratings of u is less than 2 or greater than 4, decrease or increase r by 1. Further, we have found that a $k \in [8, 17]$ lead to the best results.

3) ALS: ALS is an optimization technique for factor matrices U and V. In every iteration alternately either U or V is fixed and the other is being adjusted. The matrices are updated using the following equations, where λ is a regularization factor:

$$U = (VV^T + \lambda I)^{-1} V A^T \tag{3}$$

$$V = (UU^T + \lambda I)^{-1} U A \tag{4}$$

In each iteration the error can either decrease or remain unchanged and thus, it guarantees convergence to a local minimum and a minimal RMSE.

D. EDP [1]

Instead of considering the full rating matrix, we generate random vectors $u, v \in \mathbb{R}^k$ and biases $bu, bv \in \mathbb{R}$ for each user and item respectively, the so-called embeddings. The rating of user i for item j can then be calculated as:

$$\hat{r}_{i,j} = u_i \cdot v_j + bu_i + bv_j \tag{5}$$

By performing this procedure on all users and items, we obtain two matrices $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{k \times n}$, and two vectors $bU \in \mathbb{R}^m$ and $bV \in \mathbb{R}^n$.

The loss can be calculated using any loss function, for example the RMSE, by comparing the prediction $\hat{r}_{i,j}$ to the actual given rating $r_{i,j}$. The goal is to minimize the calculated loss by adapting the embedding matrices using some optimization technique.

E. EDP with SGD

In the following subsection, we present various techniques to improve the accuracy of the EDP and call the summary of those "EDP with SGD".

1) SGD: The SGD optimization technique minimizes a provided loss function. It does so by progressing in the opposite direction of the loss function gradient. *Stochastic* gradient descent chooses a random sample and thereafter the weights are updated, as opposed to Gradient Descent which considers all samples before updating the weights. To prevent overfitting, regularization factors are added to the loss function. The quality and speed of convergence is controlled by the LR.

2) Combining EDP and SGD: EDP with SGD works as follows: Two matrices U and V are randomly initialized. Thereafter, a given rating, $r_{i,j}$, is chosen u.a.r from the input matrix A and the gradient is calculated:

$$\nabla_{i,j} = r_{i,j} - \hat{r}_{i,j} \tag{6}$$

Where $\hat{r}_{i,j}$ is the embedding dot product as explained in Section III-D. Afterwards, the corresponding columns of U and V are updated, where u_i is the i^{th} column in U, v_j is

the j^{th} column in V, μ is the LR and λ is the regularization parameter:

$$u_i = u_i + \mu \cdot (\nabla_{i,j} \cdot v_j - \lambda \cdot u_i) \quad (7)$$

$$v_j = v_j + \mu \cdot (\nabla_{i,j} \cdot u_i - \lambda \cdot v_j) \quad (8)$$

Let $mean$ describe the global mean of all given ratings. The bias vectors are updated using a regularization factor λ_2:

$$bu_u = bu_u + \mu \cdot (\nabla_{i,j} - \lambda_2 \cdot (bu_u + bv_i - mean)) \quad (9)$$

$$bv_i = bv_i + \mu \cdot (\nabla_{i,j} - \lambda_2 \cdot (bv_i + bu_u - mean)) \quad (10)$$

3) SGD Adjustments: After observing that SGD does not converge significantly after 100 million iterations, this value was set as the default number of iterations after which the algorithm terminates, where one iteration is defined in the subsection above.

The LR was initially set to 0.1, a very popular choice for this purpose. By applying cross validation and grid search, the best values for λ and λ_2 were found to be 0.08 and 0.04 respectively.

4) Decreasing Learning Rate[5]: We have found that a good strategy is to have a rather large LR in the beginning and to slowly decrease it every few iterations. By means of cross validation we have found that the best method is to divide the current LR by two after a constant amount of iterations. Since the number of iterations was fixed to 100 millions, the LR was halved every 10 million iterations, starting with a LR of 0.1.

5) Averaging over k: Increasing the number of factors k will result in a better personalization of the prediction, up until to the point where overfitting starts to happen and the quality of the recommendations starts to decrease. Based on the results obtained in the SVD approach described in Section III-C2, a good range for k was found to be in the interval [8, 17]. The different values for k were then combined, by performing the above procedure for each $k \in [8, 17]$, and computing the average prediction over all resulting matrices. Averaging over k is a form of bootstrap aggregation, however, we will refer to it as "averaging over k", in order to differentiate between this procedure and the bootstrap aggregation described in Section III-G.

F. EDP with Adam

In the following subsection, we present various techniques to improve the accuracy of the EDP and call the implementation of those "EDP with Adam".

1) Adam: The Adam (adaptive moment estimation) optimizer [6] can be interpreted as a combination of RMSprop[7] and SGD, while also including the momentum. It uses squared gradients to scale the LR, like RMSprop, and takes advantage of the momentum by using the moving average of the gradient, as opposed to the gradient itself, like SGD with momentum does. Since it is an adaptive LR method, it computes individual LRs for different parameters. Adam uses estimations of first and second moments of gradient to adapt the LR for each weight.

2) The One-Cycle Policy: Smith[8] describes an approach to reach a better optimum faster by adjusting batch size and LR. In his paper, he suggests to apply the one-cycle policy to LRs. This cycle consists of two steps. During the first step, the LR is linearly increased starting from the lowest rate, which is $1/25^{th}$ of a previously defined maximum LR, until the maximum is reached (after approximately half of all epochs). During the second step, the LR is linearly decreased from the highest LR to the lowest. At this point a few training epochs still remain, during which the LR is further decreased to $1/100^{th}$ of the highest LR to reach a better local optimum.

The motivation behind adapting the LR in this fashion is the ability to use the LR as a regularization and overfitting prevention method while keeping the LR high. This helps to avoid steep areas of loss and to reach a better local optimum.

3) Mean Squared Error (MSE): Instead of the standard error (RMSE) we use the MSE. MSE is a loss function, corresponding to the expected value of the squared error, $MSE = (RMSE)^2$. The advantage over the RMSE is that the MSE considers both the variance of the estimator and its bias. [9]

4) Activation function and range adjustments: The prediction should always be in the range of $[1.0, 5.0]$. Therefore not every rating has an equal error range. We apply an adjusted sigmoid function as an activation function at the end, so that the output always stays in the range of $[0.5, 5.5]$. So every rating can have an error up to ± 0.5. It is important to note, that the final prediction is again cropped to the range of $[1.0, 5.0]$.

5) Hyperparameter tuning: By varying the update interval of the embeddings between all $(user, item)$ samples (i.e., the batch size), we can control the stability of the learning process and counter overfitting. The same applies to the number of times the whole set of $(user, item)$ pairs is used for updating the embeddings (the epochs). The randomly initialized variables for each embedding vector contains latent information in the form of a floating point number. However, this does not automatically mean that more variables are desirable, as with too much information, single information points become less important, and with too few variables, information might be missing. Thus it is of importance to find a vector length with the right amount of information.

G. Bootstrap Aggregation

The last step in our model was to combine the two procedures described in Sections III-E and III-F. The foundation was presented as bagging (or bootstrap aggregation) by Breiman[2]. We call the resulting algorithm Bootstrap Aggregation-Based Collaborative Filtering (BACF). The final prediction matrix is obtained by calculating the average

over the two rating matrices output by the EDP with SGD and EDP with Adam models.

IV. Results

The table below summarizes the best public Kaggle scores we achieved in our implementations.

Algorithm	Score
KNN	1.12769
SVD	1.00636
ALS	1.04141
EDP with SGD, LR = 0.1, k = 17	1.03843
EDP with SGD, adaptive LR	0.97967
EDP with SGD, adapt. LR, k $\in$ [8,17]	0.97728
EDP with Adam, high batchsize	0.98459
EDP with Adam, low batchsize	0.98056
EDP with Adam, averaging all batchsizes	0.97781
BACF	0.97438

We have found that EDP combined with simple SGD, where the LR and k is fixed, already outperforms ALS and KNN. SVD performed better, however the potential for further improvement appeared to be more limited. Using EDP with SGD combined with an adaptive LR, as described in Section III-E4, improves the score significantly, outperforming SVD. Using both an adaptive LR and averaging over k, as explained in Section III-E5, improved the score further. The score for EDP in combination with Adam is comparable to the score obtained from optimizing the EDP with SGD approach. Thus, it also outperforms all baseline algorithms. Combining the scores of results obtained from setting different batch sizes in EDP with Adam resulted in an even better score and was even less prone to overfitting. We have found that combining the results from the two approaches, EDP with SGD with an adaptive LR and multiple k values and EDP with Adam using a low batch size, using bootstrap aggregation (BACF), as described in Section III-G, lead to the highest score improvement, resulting in a score of 0.97438.

V. Discussion

A. Batch size

The EDP with Adam model had two sweet spots on two noticeably different batch sizes, a low one of 80 and a high one of 50'000 samples. Deviating from those numbers lead to a decrease on the training loss but a stagnation of the validation loss, strongly indicating overfitting on the training set. The higher the batch size got, the wider the gap between the losses became, contradicting our intuition that a higher batch size leads to a higher generalization over the data set, as the adjustable parameters only adapt after "seeing" a bigger picture. The same holds for a very low batch size (< 60).

It is important to note that after finding a good training-validation loss relationship, the model was trained again with those parameters on the whole set.

B. Adaptive LR and Averaging Over k

A large LR ensures that the algorithm does not end up in a local minimum, which might be a lot larger than the desired global minimum. A small LR however, prevents a lack of accuracy in the calculation of the predictions. Decreasing the LR with time results in a better prediction accuracy and a better local minimum.

Choosing the number of factors for the matrix factorization is a trade-off between a good personalization and overfitting of the predictions. Considering the error of all asked entries, all values of k in the interval [8,17] perform well, however our assumption is that for every k there are some outliers, resulting in a bad prediction. Those outliers are not the same for all k, such that averaging reduces those deflections, moving them closer to the proper prediction value.

C. BACF

Combining the two approaches has not only the effect of reducing overfitting, as presented by Breiman[2], but presumably also offers the same advantage as averaging over k, namely to reduce the effect of outliers on the overall score, combining their individual strengths. However, simply combining the best individual results does not end in the best overall score. The best score was found to be the combination of EDP with SGD using an adaptive LR and averaging over k, and EDP with Adam using a low batchsize. When considered individually, EDP with Adam resulted in a better score when averaged over all batch sizes.

VI. Summary and Future Work

In this paper we presented a novel approach to the collaborative filtering problem, based on bootstrap aggregation of differently optimized EDP algorithms, namely using SGD and Adam. The SGD optimized EDP algorithm was further improved, using a decreasing LR and averaging over a various number of factors in the matrix factorization. The Adam optimized EDP algorithm was further improved by applying the one-cycle policy, extending the range through an activation function, hyperparamter tuning, and averaging over the results of different hyperparameters. Future possible improvements include different ways of decreasing the LRs for the SGD optimization[5] and averaging not only over k, but also over other parameters, like the regularization factors and the various LRs. When combining the two approaches into BACF, the two optimizers may be weighted differently.

References

[1] S. Gupta, "Collaborative filtering and embeddings — part 1," 2020. [Online]. Available: https://towardsdatascience.com/collaborative-filtering-and-embeddings-part-1-63b00b9739ce

[2] L. Breiman, “Bagging Predictors,” *Machine Learning*, vol. 24, pp. 123–140, 1996.

[3] D.-K. Chae, S.-C. Lee, S.-Y. Lee, and S.-W. Kim, “On identifying k-nearest neighbors in neighborhood models for efficient and effective collaborative filtering,” *Neurocomputing*, vol. 278, pp. 134–143, 2018.

[4] C. Zeng, C.-X. Xing, L.-Z. Zhou, and X.-H. Zheng, “Similarity measure and instance selection for collaborative filtering,” *International Journal of Electronic Commerce*, vol. 8, no. 4, pp. 115–129, 2004.

[5] S. Lau, “Learning rate schedules and adaptive learning rate methods for deep learning,” 2020. [Online]. Available: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-\for-deep-learning-2c8f433990d1

[6] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[7] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[8] L. Smith, “A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay,” 03 2018.

[9] E. L. Lehmann and G. Casella, *Theory of point estimation*. Springer Science & Business Media, 2006.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

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

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

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

Title of work (in block letters):

An Alternative Approach to Bootstrap Aggregation-Based Collaborative Filtering

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Kamp	Sarah
La	Silvia
Knobel	Franz
Cantieni	Josua

With my signature I confirm that

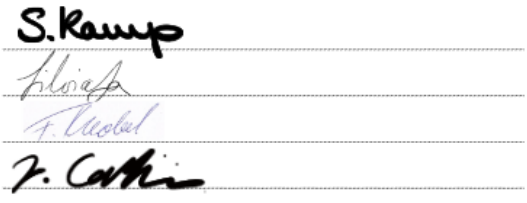
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

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

Place, date

Zurich, 31.07.2020

Signature(s)



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