Pattern Classification and Machine Learning - Project 2

Group 'ALS is the best'
Paweł Guzewicz *(271279)*
Akhilesh Gotmare *(272620)*
Ana Stanojevic *(268578)*
Department of Computer Science, EPFL, Switzerland

***Abstract*—Recommender systems are a subclass of information filtering system, that seek to predict the "rating" or "preference" that a user would give to an item. Given data set of ratings we implement collaborative filtering approach by doing matrix factorization.This project attempts to build a recommender system using the collaborative filtering and matrix factorization methods - Alternating Least Squares (ALS) and Stochastic Gradient Descent(SGD). In the report we also provide ideas which could be elaborated in order to obtain more accurate predictions.**

I. Introduction

Recommendation systems changed the way inanimate websites communicate with their users. They have become extremely common in recent years, and are utilized in a variety of areas: movies, music, research articles, and products in general, where the system predict items (ratings for items) that the user may have an interest in. Collaborative filtering approaches build a model from a user's past behavior (items previously purchased or selected and/or numerical ratings given to those items) as well as similar decisions made by other users. This model is then used to predict items (or ratings for items) that the user may have an interest in.

For this project task, we are required to predict recommendations of movies to users. We are given ratings of 10000 users for 1000 different movies. All ratings are integer values between 1 and 5 stars. No additional information is available on the movies or users. Goal of this project is to create collaborative filtering algorithm in order to build our own recommender system, and test its performance. Performance of the system is evaluated according to the prediction error, measured by root-mean-squared error (RMSE).[1]

II. Methodology

A particular type of collaborative filtering algorithm uses matrix factorization. [2] This technique has a very interesting property that it does what is called as feature learning. This means that this is an algorithm that can start to learn for itself what features to use. So we hope to 'explain' each rating x_{dn} by a numerical representation of the corresponding movie and user - in fact by the inner product of a movie feature vector with the user feature vector. In other words, we minimize error which is given as

$$min_{W,Z} L(W,Z) = \frac{1}{2} \sum_{(d,n)\in\Omega} [x_{dn} - (WZ^T)_{dn}]^2 + \lambda_w ||W_{Frob}||^2 + \lambda_z ||Z_{Frob}||^2, \quad (1)$$

where W and Z represent user and item features, respectively and λ parameters are used for regularization. We use two different optimization algorithms to build a model that has minimum error - Alternating Least Squares (ALS) and Stochastic Gradient Descent (SGD).

A. Data pre-processing

All ratings are stored in the form of a matrix (2d numpy array) such that rows represent different movies and columns represent different users. At the i, j th element - intersection of row (movie) *i* and column (user) *j* we have rating which user *j* gave to movie *i*, if the rating exists in the given data or 0 otherwise.

Differences of quality of movies and rating scales of users may influence model that is built, hence we need to normalize the data to eliminate this factor. This can be done in a few different ways.[3] In ALS method, we subtract from each nonzero element the average rating of that user. We modify this resulting matrix by subtracting the average rating (in the modified matrix) of item j. In SGD method, from all nonzero elements we subtract average of the mean rating of user i and mean rating of item j (taken from original matrix). That is, subtracting one half the sum of the user mean and the item mean. These normalization techniques are very similar, nevertheless we use approach which gives better results combined with certain method.

Since we choose to normalize the ratings data matrix, then when we make predictions, we need to undo the normalization. That is, if whatever prediction method we use results in estimate e for an element $R_{i,j}$ of the normalized matrix, then the value we predict for $R_{i,j}$ in the true utility matrix is e plus whatever amount was subtracted from row i and from column j during the normalization process. Hence we store remainders of these subtractions. When the model

is built we make inverse action of adding remainders to each element, in order to make correct final prediction.

Both of optimization tools we used are iterative and request *matrix initialization* of W and Z matrices with some values. This step can be very important, because poorly chosen beginning values of W and Z may lead techniques to end iterations in local minimum. Most common initialization is with small random values. This is one of the logical choices and it gives decent results, but we don't use it in the final prediction. One technique we use with intention to improve performances is initialization using Singular value decomposition (SVD). Therefore we initialize W and Z as the matrices from SVD decomposition of ratings matrix given to us in the form of the dataset but we need to set the unknown ratings to 0. Another technique we use is initialization of first column of W and Z matrices with mean values for certain movie and user, respectively, while other columns we still initialize with small random values (similar to exercises). Previous two techniques we use for the ALS algorithm. By exploring data we also notice that it can be useful to initialize matrices with values such that the inner product their multiplication predict value equal to global minimum of training data set i.e. every element of $W.Z^T$ equals the global minimum by setting every element of W and Z to $\sqrt{\mu/K}$ where μ is the global mean and K is the number of latent features.

B. Baseline cases

Using the global mean as the prediction for all entries returns an rmse of 1.19. Whereas if we predict all the ratings of a user by his user mean, we get an rmse of 1.029. An alternative baseline - where we can predict all ratings of an item by the item's mean rating gives an rmse of 1.08.

C. Alternating Least Squares (ALS)

We implemented this algorithm using coordinate descent (minimize (Equation (1) w.r.t. Z for fixed W and then minimize W, given Z). In this approach we used two different set of parameters to obtain two different models. Prediction is then done by averaging predicted values of these two models. Seed for random number generator is in both cases set to 888. Considering first model number of iterations after which we stop is 100. We initialize matrices using approach with mean values (mentioned third in appropriate subsection). All parameters are optimized using grid search and cross validation techniques. Moreover, by using cross-validation method we avoid over-fitting. One of the most important parameter to set is number of latent features. We optimize this parameter, to value $K = 50$. Moreover, regularization parameters λ_w and λ_z influence accuracy a lot. Similarly, we optimize these hyper-parameters and obtain values λ_w=60 and λ_z=10. Considering second model number of iterations after which we stop is 1. We initialize matrices using SVD approach (mentioned second in appropriate subsection). Latent features parameter is optimized to value $K = 100$ and regularization parameters are optimized to λ_w=60 and λ_z=10.

D. Stochastic gradient descent(SGD)

Stochastic gradient descent is the other algorithm we use to obtain matrix factorization from train data. In order to improve our predictions we come up with a model that is able to capture the biases that a movie or user might possess in their ratings and also the interaction between the movie and the user.[4] The prediction such a model would generate for a rating R_{dn} would be equal to $\mu + b_d + b_n + (WZ^T)_{dn}$ where μ is the global mean, b_d is the bias for item d and b_n is the bias for user n and the d, n - th term of WZ^T captures their interaction. In other words, we come up with slightly different loss error compared to Equation (1), which we want to minimize:

$$\begin{aligned} min_{b_d, b_n, W, Z} L(W, Z) = \\ \frac{1}{2} \sum_{(d,n) \in \Omega} [x_{dn} - \mu - b_d - b_n - (WZ^T)_{dn}]^2 \\ + \lambda_w ||W_{Frob}||^2 + \lambda_z ||Z_{Frob}||^2 + \lambda_d ||b_d||^2 + \lambda_n ||b_n||^2 \end{aligned} \tag{2}$$

μ is a fixed quantity. We treat b_d , b_n, W and Z as the parameters that are to be varied in the SGD algorithm for optimizing the loss function in 2

Because of existence of variables b_d and b_n, we calculate two new gradients over the regular SGD for matrix factorization. In a single pass over an element of the training set, we evaluate the derivative $\frac{\partial}{\partial b_{(d)}} f_{dn}(W, Z, b_i, b_x)$ and update b_d in the direction opposite to this derivative. Similarly we update the item bias b_n in the negative direction of the derivative $\frac{\partial}{\partial b_{(n)}} f_{dn}(W, Z, b_i, b_x)$. [5]

Update steps for rows and columns of W and Z are similar to what has been discussed in the course. There can be some optimization in the execution time of the code by evaluating only the entries of the prediction ratings that get modified in a pass over a single training instance instead of recomputing the entire prediction matrix. We calculate gradient for every nonzero element in ratings matrix and iteratively apply standard formula for gradient descent algorithm, with step γ. Seed for random number generator here is set to 77.

Finding the hyper-parameters is again the tricky part in this case too. To worsen things, run time of a single epoch turns out to be around 650 seconds which makes trying out different choices of hyper-parameters very expensive. In order to deal with this issue, we work on a smaller submatrix of the original ratings matrix. Running SGD on this submatrix takes reasonable although not small amount of time. Trying out different values of the regularization parameter λ for this submatrix revelas that it does not have significant impact on the performance and we stick with

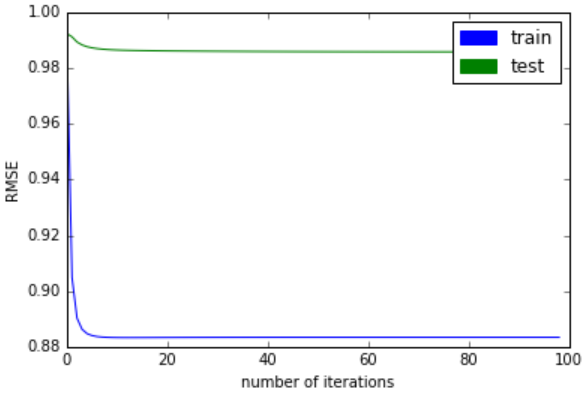

Figure 1. RMSE using ALS technique (first model).

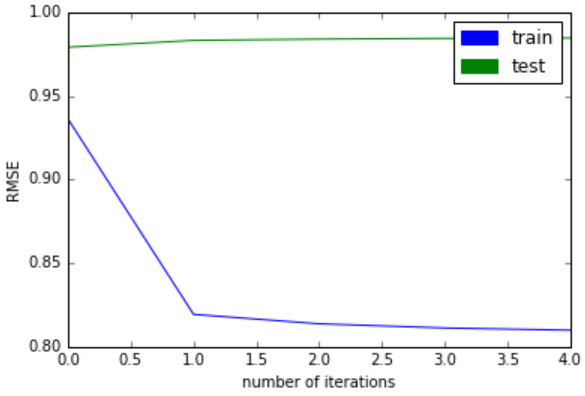

Figure 2. RMSE using ALS technique (second model).

setting all the regularizers equal to zero as this gives an optimal performance compared to different values of λ. Similar investigations are done for other hyper-parameters: γ the step-size and K the number of latent features and optimal values of 0.006 for $gamma$ and 50 for K are found. We run gradient descent algorithm for 15 iterations for the actual data and generate predictions. Fig shows the training and test errors over the number of iterations for this SGD experiment. However this result is not good enough to compete on Kaggle neither it beats the ALS rmse score.

E. Hybrid approach

Hybrid approach implies combining two or more obtained models in order to make the best prediction. Main idea behind this is that one model can be used to overcome problems of the other model, and vice versa.

We combine two models we obtained, one using ALS and other using SGD. Hope is that the hybrid model can provide more accurate recommendations than ”pure” approaches. Each prediction is made by averaging prediction obtained using ALS model and prediction obtained using SGD model. However since SGD seems to still lack behind in the rmse scores, such a hybridization worsens things for ALS. Hence we try merging predictions of different ALS models and that seems to be a better scorer.

F. Making final prediction

Predictions which we obtain from our models are *float* numbers. It is important to notice that there cannot exist ratings which are lower than 1 or higher than 5. Therefore, values on the output of our model which are lower than 1 we declare to be 1, and values higher than 5 we declare to be 5. Ideally our predictions should be integers eg. if the model predicts 4.5, in a practical application we would predict the rating to be 4 or 5, however such discretizing seems to affect the rsme scores in a negative way.

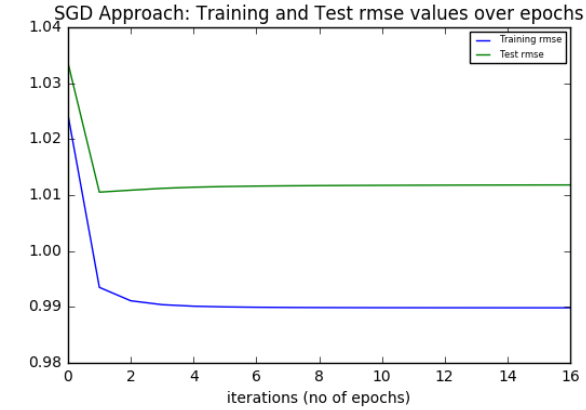

Figure 3. RMSE using SGD algorithm .

III. Results

Using the ALS algorithm we obtain two different models. We measure score on our train set, test set as well as on ”unseen” data (submitting predictions for data that we don’t have predictions for - competition on kaggle platform). Using parameters we stated in the section above for the first model we obtain results - RMSE on training set: 0.88407, RMSE on test set: 0.97660, RMSE on ”unseen” data: 0.99162. Similarly, for the second model we obtain results - RMSE on training set: 0.82788, RMSE on test set: 0.96468, RMSE on ”unseen” data: 0.98890. On Figure 1 and Figure 3 we represent RMSE for these two models. As we said before final prediction is made by averaging predictions which these models provide, where we obtain best RMSE error on ”unseen” data, 0.98864.

This result is our best prediction (and the one on kaggle leader board).

We observe that ALS gives better rmse results compared to SGD. In addition to this, we observe that trimming final prediction (setting values lower than 1 to prediction 1 and greater than 5 to prediction 5) results into better scores as expected (this is included in our best prediction). However,

discretizing *float* values between 1 and 5 make predictions worse, so we don't use it.

IV. Discussion

In this project we tried few different approaches to make the predictions using collaborative filtering techniques - one from the ALS family and the other from SGD. As it turns out, we fail to have a good rmse score with the SGD approach, the reason perhaps lies in the hyper-parameter optimization. We assumed that the hyper-parameters that work well for the submatrix would work well for the original data too, however this is not necessarily the case as is evident from the results.

Our task looks similar to a very famous open competition *Netflix prize*, however it is noteworthy that the results are quite different - there teams ended up with rmse scores close to 0.86 and here the competition seems to converge to 0.97, partly due to the reason that in the Netflix case, the dataset was bigger and also included temporal information about the ratings.

One of the biggest challenge we face in building our models was scalability. Algorithms which are not so difficult to implement on small data set create a lot of troubles when it comes to bigger data sets as was in this task. Those implementation have to be computable on personal computers (PC) and therefore implementation require usage of sophisticated and optimized tools. Even thought we make effort to shorten execution time for model creation, it still linger model creation and therefore makes changes and improvements, such as e.g. parameter optimization, inflexible.

Hybrid approach could most likely be improved. E.g. instead of using average on two predictions we can use weights to favor prediction of one model. It is evident that making good predictions is all about chosing the right model and finding the right set of hyper-parameters that make these models work. Our project incorporates some models that aren't very basic but the task of finding hyper-parameters has to be executed well to have good rmse scores.

References

[1] M. E. Khan, "Machine learning course - cs-433," *EPFL*, vol. 1, no. 1, pp. 1–60, 2015.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, pp. 30—-37, 2009.

[3] J. Leskovec, A. Rajaraman, and J. Ullman, "Mining of massive datasets," *Stanford Univ.*, pp. 307–343, 2014.

[4] C. R. Aberger, "Recommender: An analysis of collaborative filtering techniques," *Stanford Univ.*, 2014.

[5] (2009) Explicit matrix factorization: Als, sgd, and all that jazz. [Online]. Available: http://blog.ethanrosenthal.com/2016/01/09/explicit-matrix-factorization-sgd-als/