Design of a Recommendation System using Ensembling

Lucía Montero Sanchis, Nuno Miguel Mota Gonçalves, Matteo Yann Feo,
Department of Computer Science, EPFL Lausanne, Switzerland

***Abstract*—This report presents the results of using different Collaborative Filtering methods for implementing a Recommendation System. We use both neighborhood methods and latent factor models for predicting new ratings, and these methods are analyzed and compared to each other. We then build an ensemble of those methods using Ridge Regression and consider different ways to improve the ensemble in order to obtain better predictions.**

I. Introduction

The aim of this project is to implement a Recommendation System that predicts a numerical value r_{ui} corresponding to the predicted rating of a user u for an item i, based on the known past ratings for other users and items. The data provided consist on a first dataset of known user-item ratings (*training*) and on a set of user-item pairs which rating values we need to predict (*test*).

Since the available data consists only on the past ratings without any information about the users or films, we focus on Collaborative Filtering (CF) techniques. Ensemble methods are one of the most successful approaches for this problem.

We use an ensemble approach to combine the predictions obtained with well-known CF techniques, including neighborhood methods and latent factor models [1]. Ensemble methods achieve better results when combining predictions that are very different from each other but achieve low RMSE scores individually, which is the reason why we use a variety of CF techniques. After adjusting the parameters of these predictors to minimize their individual RMSE, we use their predictions as features for Ridge Regression.

One of the challenges we face is dealing with users and items that have a lower number of ratings, since some methods do not yield good results for these and are better at making predictions for items and users with a higher number of ratings.

II. Models and methods

We start by obtaining ratings predictions with each of the CF methods considered, that will be used as features when applying Ridge Regression for the ensembling.

It is important to note that for the ensemble we will be training the Ridge Regression model on the predictions for the *training* dataset, and then use the resulting weights to predict ratings for the *test* dataset. In order to obtain better results we split the *training* into 5 disjoint subsets with approximately the same number of entries. Then we obtain the predictions for each subset by training the models in the remaining 4 subsets. This assures that in the ensembling step more weight is given to the methods that obtain better results, which is important since when using ensembles it is not so essential that individual predictors have a very low RMSE [2].

We can improve the ensembling results by increasing the number of features for Ridge Regression. We do this by defining *meta-features* and afterwards building a polynomial basis of the features - which now include both the CF methods predictions and the meta-features.

Lastly, we explain how we solved the problem of having a lower number of ratings for some users and items.

A. Collaborative Filtering (CF) methods

In a first step we use several CF methods to obtain different predictions that we will later combine in the ensemble. We adjusted the parameters for each method using 5-fold cross-validation and grid-search.

Afterwards we obtain the *predictions* that we will use as features in the ensembling step. As explained previously, to obtain predictions for the *training* dataset we divide it into 5 subsets, and for each one we generate the predictions by training on the remaining 4. When generating the predictions for the *test* dataset we train the methods on the entire *training* dataset.

1) Baseline: Much of the variation in the ratings is related to either users or items [1]. One possible method for predicting values of ratings is shown in equation 1, where μ represents the global average rating, b_i is the bias involved in rating i and b_u is the bias of user u.

$$b_{ui} = \mu + b_i + b_u = \mu + \mu_i + \mu_u \tag{1}$$

In our approach b_i and b_u are the average ratings per item and per user respectively, after subtracting μ. This *baseline* prediction ignores the interaction between users and items [1], but we still decided to consider it as a prediction method by itself.

2) Slope One: Slope One algorithms propose a simple predictor function $f(x) = x+b$ to avoid overfitting. Equation 2 shows the rating prediction $\hat{r}_{ui}$ for item i by user u.

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} dev(i,j) \tag{2}$$

Where $R_i(u)$ is the set of relevant items and $\text{dev}(i, j)$ is the average difference between the ratings of items i and j [3]. We used the implementation of this algorithm provided in the Python Surprise library [4].

3) K Nearest Neighbors (KNN): Neighboring methods base their predictions on the relationships between items or between users. The first item-oriented approach predicts a rating $\hat{r}_{ui}$ based on the known ratings of user u for items *similar* to i. Alternatively, the user-oriented approach predicts a rating $\hat{r}_{ui}$ based on the known ratings for i by users with similar preferences to those of u [1], [5].

We used both approaches with the implementations provided in the Python Surprise library [4] that take into account a baseline rating.

- Item-based KNN parameters (`KNN(i)`):
  - k = 60
  - min_k = 20
  - Similarity measure: Pearson baseline
- User-based KNN parameters (`KNN(u)`):
  - k = 300
  - min_k = 20
  - Similarity measure: Pearson baseline

4) Matrix Factorization (MF) methods: As mentioned in the previous section, the baseline method (equation 1) does not take into account the interaction between users and items. This is why it is often used together with MF to predict ratings as shown in equation 3.

$$\hat{r}_{ui} = b_{ui} + q_i^T p_u \tag{3}$$

In this equation q_i and p_u are the factors that describe item i and user u respectively, and they are obtained by *factorizing* the matrix that contains the known ratings.

There are several approaches for MF in recommendation systems. One of them is Singular Value Decomposition (SVD), however its use is not recommended for sparse matrices since it is prone to overfitting. A similar approach includes a regularization term to avoid overfitting, as shown in equation 4, and can be optimized using Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) [1].

$$\min_{q*.p*} \sum_{(u,i)\in k} (r_{ui} - q_i^\top p_u)^2 + \lambda(||q_i||^2 + ||p_u||^2) \tag{4}$$

The implementations we used for the project are the following:

- *Truncated Singular Value Decomposition* (`SVD1`)
  We used the approach from equations 3 and 1. After subtracting the bias component, to obtain the item-user interaction we used the Truncated SVD implementation from the Scipy Python library [6]. This function factorizes the entire sparse matrix (taking the zeros into account as well), so although the result can be computed analytically the number of features k needs to be small. Otherwise for larger k values the *unknown* entries of the matrix will be predicted to be 0.
  The number of features we used is $k = 13$.
- Singular Value Decomposition (`SVD2`)
  We used the Python Surprise library [4] implementation of SVD for sparse matrices including bias, that minimizes equation 4 using SGD methods.
  The parameters used are:
  - biased: True
  - k: 130
  - reg_all: 0.08
  - n_epochs: 50
- Alternating Least Squares with Weighted Lambda Regularization (ALS-WR) (`ALS1`)
  We used the ALS-WR implementation in Python from [7], which again minimizes equation 4. The parameters used are:
  - k: 20
  - n_iter: 50
  - reg: 0.085
- Alternating Least Squares implemented from scratch (`ALS2`)
  We used the ALS algorithm that we implemented during the course, together with the baseline prediction from equation 1. Therefore, the prediction is done as shown in equation 3.
  However, since this algorithm takes a long time to converge to a solution we modified it slightly in order to allow the feature vectors to start from some given vectors (instead of starting from random values). These initial vectors (`u_feats` and `i_feats`) are the ones we obtained after the first iteration when training on the first of the fold *training* dataset.
  The parameters we used are:
  - k: 20
  - lambda_u: 0.1
  - lambda_i: 0.1
  - tol: 1e-4
  - max_iter: 100
  - init_u_features: u_feats
  - init_i_features: i_feats
- *Non-negative Matrix Factorization* (`NMF`)
  Non-negative Matrix Factorization is a MF technique where the user and item factors are kept non-negative. We have used the implementation provided in the Python Surprise library [4], where they apply stochastic descent with a step-size that ensures non-negativity as formally presented in [8], [9].

B. Ridge Regression ensembling

After obtaining the predictions for the *training* and *test* datasets we use Ridge Regression to create our ensemble.

However, by using only these predictions as features the ensemble underfitted considerably. Therefore we calculate a set of additional features based on the ones suggested in [10] as explained below.

1) Additional features: The additional features are computed for each user u or item i based on all the ratings in the *training* dataset.

- Number of ratings per user
- Number of ratings per item
- Logarithm of the number of ratings per user
- Logarithm of the number of ratings per item
- Standard deviation of ratings per user
- Standard deviation of ratings per item

2) Polynomial basis: After obtaining the set of features including the predictions and the additional features, we generate the degree-2 polynomial and interaction features and use them for training and predicting the ratings for the *test* dataset with Ridge Regression.

C. Splitting by user support

User support for a rating (u, i) is described in [2] as the number of ratings of user u. We use the *binned* approach described in [11] and split the data into 2 disjoint sets based on user support (*low support* and *high support*), so that both sets have an approximately equal number of ratings. As shown in the plot for accumulated number of ratings in Figure 1 the 218 users with more ratings account for half of the total number of votes. As a result, it is shown in the second plot in Figure 1 that users with more 1802 ratings were considered to have a high support.

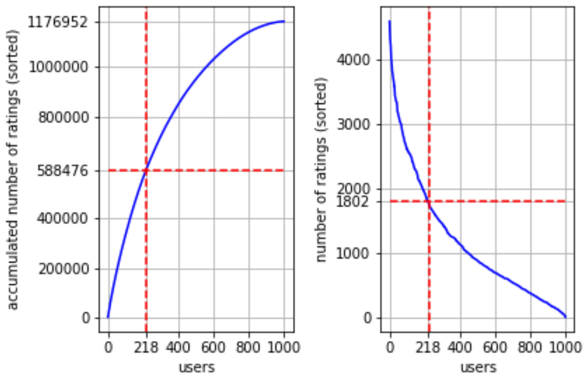

Figure 1. Determining number of ratings for split on support

Then we use the same features as previously and apply Ridge Regression to each of the two sets separately, obtaining different weights for the ratings depending on whether they have a *low* or *high* user support.

III. Results

In this section we present the results obtained by applying the prediction models described previously. Although the ratings are meant to be integer values, we did not round our predictions since this increased our RMSE value.

A. Collaborative Filtering methods

Figure 2 shows the test errors obtained for each of the collaborative filtering methods we described previously, that were obtained using the same 5 folds that we used for generating the predictions of the *training* dataset. The baseline prediction has the highest RMSE, whereas the lowest RMSE is achieved by the SVD method from the Surprise library followed by the matrix factorization by ALS method that we implemented.

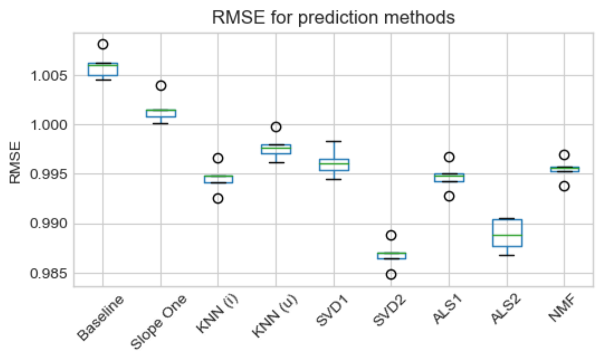

Figure 2. Test RMSE for CF prediction methods

It is interesting for both this and the upcoming sections to compare how similar the predictions are to each other, especially since this is desirable in order to obtain a better ensemble. Figure 3 shows the Pearson correlation between each pair of prediction methods, and Figure 4 shows the maximum difference between predictions.

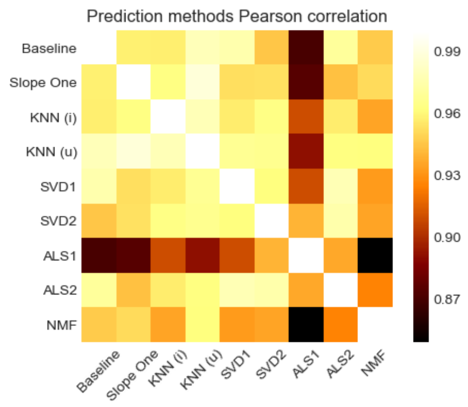

Figure 3. Correlation between predictions

Although the correlation is high, it can be seen in Figure 4 that there are certain ratings for which there are considerable differences across different methods. It is for this reason that we have decided to include all of the predictions obtained with these prediction methods in the ensemble.

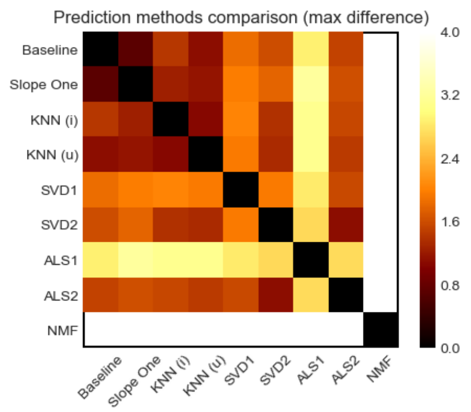

Figure 4. Maximum difference between predictions

B. Ridge Regression ensembling

In order to test the RMSE achieved by the ensemble we decided to split the *training* dataset in 95% for train and 5% for test. We then obtained the predictions for the CF models for the train dataset dividing it into 5 disjoint subsets - as we had done previously. We obtained an RMSE of 0.982649 for test with this ensemble.

C. Ensembling splitting by user support

Afterwards we split the ratings into two subsets according to their user support, and apply Ridge Regression to each of them separately.

The justification for this is shown in Figures 5 and 6, which show the RMSE that the CF methods obtain for the ratings depending on their level of support. These figures were obtained by using all the *training* dataset, so they can be compared to Figure 2 as well. It is expected that the

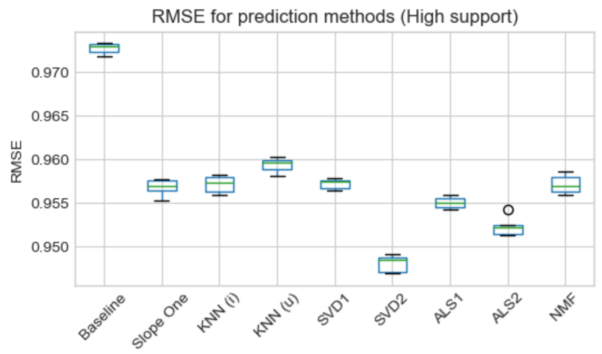

Figure 5. Test RMSE for prediction methods (high support)

error is higher when there are fewer ratings available, but it is particularly interesting to compare how well the models do compared to each other depending on the support.

We then tested the ensemble considering the same 95% and 5% train and test subsets as in the previous ensemble, in this case achieving an RMSE score of 0.975737.

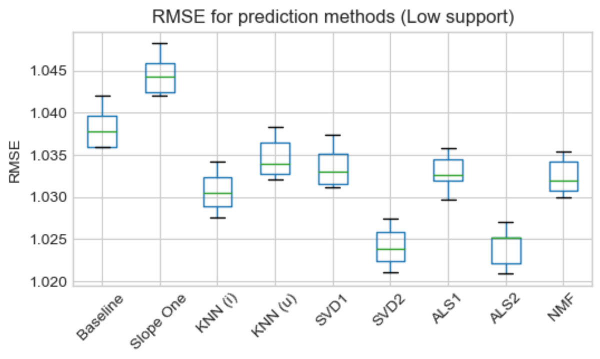

Figure 6. Test RMSE for prediction methods (low support)

IV. Discussion

Our first approach for improving the predictions was using an ensemble of methods. The best simple predictor was the MF by ALS method that we implemented, that achieved an average RMSE score of 0.9875 for the test data. Ridge Regression applied to the set of features that we created lowered the RMSE to 0.9826. Afterwards we lowered the RMSE to 0.9757 by ensembling high and low user support ratings separately.

Some CF models achieved very different RMSE scores depending on if the ratings had a high or a low user support. For instance, we can look at Baseline and Slope One since when predicting low support ratings the baseline method yields better results than Slope One, as opposed to what we saw in Figure 2. It is likely that Slope One reduced its overall RMSE by improving its predictions for high support ratings, resulting in the baseline method outperforming it for low support data.

However we did see that the errors achieved by CF methods for low user support were higher than for high support. Although applying Ridge Regression to each group of ratings separately improved the score, it seems that training some models so that they specifically minimize prediction errors for low support ratings could have improved our results. This is a common approach in the literature [11] and we can consider it as future work.

V. Summary

We focused on implementing a recommendation system based on ensemble techniques, using Ridge Regression for ensembling. The advantage of this is that we can use individual predictions already obtained with CF methods, and combine them in a relatively simple way.

We were able to improve the RMSE score of our prediction ensemble by doing feature engineering and using a binned approach.

References

[1] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, 2009.

[2] A. Töscher, M. Jahrer, and R. M. Bell, “The bigchaos solution to the netflix grand prize,” *Netflix prize documentation*, pp. 1–52, 2009.

[3] D. Lemire and A. Maclachlan, “Slope one predictors for online rating-based collaborative filtering,” in *Proceedings of the 2005 SIAM International Conference on Data Mining*. SIAM, 2005, pp. 471–475.

[4] N. Hug, “Surprise, a Python library for recommender systems,” http://surpriselib.com, 2017.

[5] Y. Koren, “Factor in the neighbors: Scalable and accurate collaborative filtering,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 1, p. 1, 2010.

[6] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–, accessed 21 Dec 2017. [Online]. Available: http://www.scipy.org/

[7] C.-K. Yau, “Recomend: Simple recommendation system implementation with Python,” https://github.com/chyikwei/recommend, 2017.

[8] S. Zhang, W. Wang, J. Ford, and F. Makedon, “Learning from incomplete ratings using non-negative matrix factorization,” in *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM, 2006, pp. 549–553.

[9] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, “An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1273–1284, 2014.

[10] J. Sill, G. Takács, L. Mackey, and D. Lin, “Feature-weighted linear stacking,” *arXiv preprint arXiv:0911.0460*, 2009.

[11] M. Jahrer, A. Töscher, and R. Legenstein, “Combining predictions for accurate recommender systems,” in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2010, pp. 693–702.