Machine Learning Project 2
Recommender System with Blending

Ozkalay Ahmed Furkan
Radosław Dryzner
Samuel Rey

***Abstract*—Recommender systems are extensively used in many applications to recommend products or services. In this work, we present a recommender system that we worked on for the class' project based on different machine learning techniques. Using a large number of well known algorithms used for recommender systems, we look at what kind of results we obtain by using them. To obtain a highly accurate set of predictions, we use the interesting technique of blending our predictions in another machine learning algorithm, effectively combining different predictions to obtain a final, more accurate one.**

I. Introduction

For the second project of Machine Learning's class, we are tasked with the creation of an accurate recommender system with what we learned. Many different algorithms and methods exists for this task. In our project, we try many different models and techniques. We start by some baseline models implemented manually, we then look at the Surprise library for more models including another baseline, slope one and kNN inspired algorithms. Finally, we use matrix factorization algorithms implemented in Spotlight, a framework based on Pytorch.

We use the techniques that we learned in class to separate the data, train it and effectively compare them, obtaining already very interesting results. However, we also decided to use another technique, based on previous work, which uses blending of different results, that is, running a machine learning algorithm on collected predictions, to obtain a single, more accurate prediction.

As a result, we have a good comparison of many different and well known models and a technique to combine these models to achieve a higher overall accuracy in predicting ratings.

II. Models and Methods

In this section, we start by explaining the different Machine Learning models and algorithms we use to make the predictions. Then we will walk over the methodology used to work on our dataset and obtain our predictions.

A. Baseline Models

Let's start by looking at some baseline models we implemented.

1) Mean Predictors: The first kind of baseline models we have implemented are based on the means of ratings. Three models in total, global mean, mean per user and mean per item. These models simply predict new ratings by assigning the global mean rating, the mean rating of the user or the mean rating of the item respectively as the new prediction. In other words, the prediction for the global mean model is given as:

$$\hat{x} = \overline{x} = \frac{1}{|\Omega|} \sum_{(u,i)\in\Omega} x_{ui} \tag{1}$$

for the user mean:

$$\hat{x_u} = \overline{x_u} = \frac{1}{|\Omega_{u:}|} \sum_{(i,u)\in\Omega_{u:}} x_{ui} \tag{2}$$

for the movie:

$$\hat{x_i} = \overline{x_i} = \frac{1}{|\Omega_{:i}|} \sum_{(i,u)\in\Omega_{:i}} x_{ui} \tag{3}$$

where Ω is the set of non-zero elements in the user-movie ratings training matrix, $\Omega_{i:}$ are the users who rated movie i and $\Omega_{:u}$ are movies rated by user u. For user mean (respectively movie mean) predictor, if a prediction is asked for a user (respectively item) that isn't in the training matrix, the global mean of the matrix is predicted instead.

2) Mood Predictor: The Mood predictor uses the baseline mean predictors to calculate a 'mood' for each user. This mood m_u for user u is defined as:

$$m_u = \overline{x_u} - \sum_{u\in U} \overline{x_m} \tag{4}$$

where $\overline{x_m}$ is the mean ratings of user m and U is the set of all users.
Then we predict a rating for user u and movie i as following:

$$\hat{x} = \overline{x_i} + m_u \tag{5}$$

where $\overline{x_i}$ is the mean ratings of movie i and m_u is the mood of user u.
The goal of this baseline predictor is to capture the 'mood' of a user. If a particular user scores movies generally higher

than all other users, it will be taken into account by the prediction.

3) Loss function: To evaluate all our baseline models, we use the following loss function:

$$L = \frac{1}{|\Omega|} \sum_{(i,u)\in\Omega} (\hat{x_{ui}} - x_{ui})^2 \tag{6}$$

where $\hat{x_{ui}}$ is the predicted rating for user u and item i and x_{ui} is the actual rating of user u and item i.

This function is the well known RMSE function.

B. Surprise Models

The next models we have tried are the ones implemented in the Surprise library[1]. This library specializes in recommender systems and implements many different algorithms. In this section we explain the ones that we tried and considered in our final model comparison.

1) Baseline Algorithm: An algorithm called Baseline is used to get the predictions, it is implemented as the `BaselineOnly` class. The algorithm tries to find bias terms for each user and item and adds user and item biases to global mean when predicting rating for given user and item pair. In other terms, the prediction $\hat{x_{ui}}$ for user u and item i is given by:

$$\hat{x_{ui}} = \mu + b_u + b_i \tag{7}$$

Where μ is the global average of the rating matrix, b_u is the observed deviation of user u from the global average and b_u is the observed deviation of the movie i from the global average. The deviation is just the difference of the average rating for the particular user (respectively movie) from the means of all users (respectively items).

If we are predicting for a user (respectively item) that is not present in the training set, the b_u (respectively b_i) term is set to 0.

2) Slope One: Slope One is a simple collaborative filtering algorithm implemented as part of Surprise. The prediction $\hat{x}_{ui}$ for user u and item i is given as follows:

$$\hat{x}_{ui} = \overline{x_u} + \frac{1}{|R_i(u)|} \sum_{j\in R_i(u)} dev(i,j) \tag{8}$$

where x_u is the average of ratings made by user u and $R_i(u)$ is the set of relevant items. The relevant items here are all items j rated by the user u that also have at least one other common user with i. $dev(i,j)$ is defined as

$$dev(i,j) = \frac{1}{|U_{ij}|} \sum_{u\in|U_{ij}} r_{ui} - r_{uj} \tag{9}$$

where r_{ui} is the rating for the user u and item i in the training rating matrix.

3) Co-Clustering: The Co-Clustering algorithm is a collaborative filtering algorithm based on co-clustering implemented in the Surprise library.

Users and items are clustered into clusters C_u and C_i,and some co-clusters C_{ui}. The prediction $\hat{x}_{ui}$ is computed as following:

$$\hat{x}_{ui} = \overline{C}_{ui} + (\mu_u - \overline{C}_u) + (\mu_i - \overline{C}_i) \tag{10}$$

where $\overline{C}_{ui}$ is average rating of co-cluster C_{ui}, $\overline{C}_u$ is average rating of u's cluster, $\overline{C}_i$ is average rating of i's cluster, μ_u is the average of user u's ratings and μ_i is the average of item i's rating.

If the user we want to predict for is unknown, the prediction is the average over the ratings of item, while if the item we want to predict for is unknown, the prediction is the average over the ratings of the user and if both the user and the item are unknown then the prediction is the global average over all ratings.

4) kNN Inspired Algorithms: We tried 4 different implementation of kNN inspired algorithms which are available in Surprise library. These algorithms try to find similar user/items for making prediction, after finding similar user/items prediction is done by weighted averaging the ratings given by similar users/ratings given for similar items where weights are similarities. KNNBasic algorithms predicts based on that weighted rating. KNNWithMeans algorithm adds a term on top of user/item mean where the additional term is calculated using weighted average of differences between rating and mean, where weight are again similarity scores. KNNWithZScore does similar to KNNWithMeans, but difference between ratings and means are scaled by inverse standard deviations. KNNBaseline is also similar to KNNWithMeans, but instead of bias it uses baseline parameters for each user and item, where baseline parameters are calculated iteratively using Stochastic Gradient Descent or Alternating Least Squares. We implemented these algorithms using Surprise library. In order to find best performing hyperparameters we did grid search using 3-fold cross-validation. Mean RMSE on validation folds of consolidation is reported on the table for each algorithm where best hyperparameters combination is used for each.[1]

C. Matrix Factorization Models using Spotlight

Spotlight is a Python framework built on top of Pytorch which we have seen in class. It contains multiple models and algorithms. The model we used in our project is the `ExplicitFactorizationModel` which builds on top of Pytorch's framework to implement the classic matrix factorization algorithm.

Although this algorithm doesn't use any actual neural nets, it uses Pytorch's structure to implement it. As we have seen in class, in this algorithm we are looking to find a factorization of the training rating matrix into two matrices that represent features of users and items (movies) respectively. Let's call the user features matrix $W \in \mathbb{R}^{U \times K}$ and the item features matrix $Z \in \mathbb{R}^{I \times K}$. K is a parameter, corresponding to the number of features (latent factors) for each user and item.

The model predicts a rating $\hat{x_{ui}}$ for user u and item i as follows:

$$\hat{x_{ui}} = \left(\left(WZ^T \right)_{ui} + w_u + z_i \right) \tag{11}$$

where w is a vector denoting bias term for each user and z is a vector denoting bias term for each item. The two matrices W and Z as well as the bias vectors are not known and it's these quantities that needs to be learned.

Given U users, I items and the corresponding rating matrix $R \in \mathbb{R}^{U \times I}$, we aim to find the feature matrices W and Z, and the two bias vectors w and z such that the quantity

$$L = \frac{1}{2} \sum_{\substack{u=1,\dots,U \\ i=1,\dots,I}} \left(R_{ui} - \hat{x_{ui}} \right)^2 + \frac{\lambda}{2} \|W\|_2^2 + \frac{\lambda}{2} \|Z\|_2^2 \tag{12}$$

is minimized. λ is regularization parameter for weight matrices and x_{ui} is the prediction as defined in 11

In order to the train the model with loss function given in 12, Mini-Batch Gradient Descent algorithm is used. The gradient of the loss function 12 is computed only for randomly chosen batch of examples in summation and the update is done according to gradient descent.

We also use the loss function 12 to evaluate our model.

D. Training the Models

1) Data splitting: The training dataset that we have is split into two datasets, the training set and the testing set. The testing set is obtained by randomly removing ratings from the sparse rating matrix and putting them in a new sparse rating matrix. Thus we obtain the two sparse matrices, train and set.

2) Hyperparameter tuning: Our spotlight models requires several hyperparameters such as the number of latent features in our feature matrices or the regularization factors. In order to find the best set of hyperparameters, we use another validation set to compare them. We split the train set further into a smaller train set and the remaining ratings are put in a validation set. We train several models on the reduces training set and compare the validation errors on the validation test.

The set of parameters having the lowest validation error is chosen as the best.

E. Training and Model Selection

Once we have our hyperparameters, we train all our models on the training set. Then we calculate the test errors of the trained models on the test set. We can directly compare the models now on the test set and pick the one that has the lowest test error as our best model.

F. Blending

Despite our different models, the errors that we obtained were not as good as we would like. After reading about the famous "Netflix Prize" competition in 2009 and the method for achieving best result described by the winner team, we decided to try a similar method[2]. This method is called 'blending' and involves mixing the predictions generated from multiple different models into a new single prediction. This new prediction gives better results that the individual model predictions.

First, we split training set into two, one called the model training set, and the other one called the blending set. Secondly, we train several different models on the model training set using the models which we explained in II. Then we put predictions of each model in a matrix. Each row of the matrix corresponds to an example in our dataset (a value that needs to be predicted) and each column corresponds to one of our trained models. The targets we want to achieve are true ratings corresponding to the predictions. In order to find in which weights different models should be blended we used linear and ridge regression. We try different regularization parameters for ridge regression, and evaluate linear regression and ridge regression with different regularization parameters on blending set. Model having the smallest test RMSE on blending set is chosen to be used in blending of models. In order to decide how many models and which models should be used in blending we mainly used two simple heuristics. First, models achieving good test RMSE score on separate validation set should be useful in blending, as these models already give seemingly good predictions. In order to decide which models should be added into blending, we used blending set as validation set for models that are trained on model training set. Second, there should be variety of models that are not very similar in their predictions, giving flexibility in predictions of blending after weighted averaging predictions of models used in blending.

III. Results

Now that we have defined all the models that we explored, let us talk about the results that we have gotten when using them.

We start by splitting our data into 90% train and 10% test data.

Baselines models where used as is, since they don't need any hyperparameter tuning. The models are trained on our training set (90% of our data). The results for these models are found in III.

For models of the surprise lib, we have used 3-fold cross validation to fine-tune the hyperparameters with a grid search to test different parameters. The average RMSEs on the validation folds for the best model are shown in III.

For spotlight, we used a single validation set to fine tune the parameters and also used grid-search to test different combinations of them. Then the best model was chosen and trained on our training set and the RMSE calculated on the testing set with this trained model is shown in III.

Concerning the blending model, we use the following parameters to get the best spotlight models: The embedding dimensions of the spotlight models varied between [8, 16, 32, 64], the number of iterations varied between [3, 4, 6], the regularization parameter coefficient λ varied between [0, 1e-8, 1e-6, 1e-4], and the learning rate varied between [1e-4, 1e-3]. We testing with different number of spotlight models (2). At the end, we selected the 192 best one to be used in our blending model. The loss functions were the linear regression, and the ridge regression model. The linear regression that gave us 0.6552 of error when run on the train set, and the ridge regression function that gave us 0.85303 of error.

Model	RMSE Error
Global Mean Predictor	1.119
User Mean Predictor	1.029
Item Mean Predictor	1.085
Mean Mood Predictor	0.995
Baseline Algorithm	1.114
KNNBasic	1.053
KNNWithMeans	0.990
KNNWithZScore	0.989
KNNBaseline	0.988
Slope One	0.997
Co-clustering	1.008
Best Spotlight Model	0.998

Figure 1. RMSE Error for different models

The run.py python file do the following steps: First, we load the sample submission into run.py. Then, we run the spotlight models on the 90 percent of the with different parameters to get as much as variety possible. When the models are finally ready, we train the blending models using the left 10 percent of the training data. Once all these tasks are fulfilled, we load the submission file and predict the ratings using the blending model.

IV. Discussion

The results of the blending model proved to be effective. By using different models of recommender systems, the blending model seemed to be a solid predicator. The final score done on Kaggle was 0.97613, which is rather impressive given the error on each individual Spotlight models. In order to improve the results, we could have trained more spotlight models. The difference on the blending model with 60 or 192 spotlight models is noticeable, so we think that training on more models could help. Also, we could have trained the blending model with more diversity of model. For instance, adding neural network models could have grasp different subtility than the one of the spotlight model. It is interesting to see how much The BellKor Solution to the Netflix grand prize is still relevant.

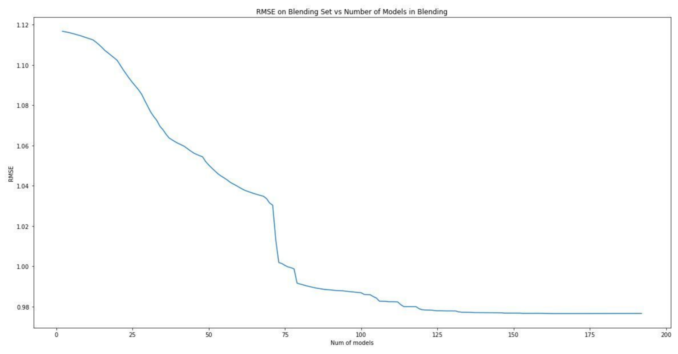

Figure 2. Blending model RMSE for different number of spotlight models

V. Summary

As a result of this work, we have implemented and tried different models and algorithms widely used in recommender systems. Our analysis has helped us get an idea about the performance of the various models on our datasets. We have learned how the algorithms are used to train and predict new ratings.

We have found out that the regular biased matrix factorization model, as implemented by the Spotlight library, performs the best in our case, and decided to use it for our submissions.

However, not satisfied with the raw performance of a single model, even after finding good hyperparameters, we decided to improve our results by adding blending. By training many different models and using them to get predictions, we have obtained a large number of prediction for a simple example we actually have to predict. Blending these predictions in a ridge regression model proved very effective at improving the performance of our recommender system. The implementation of blending we wrote can further be used to blend the predictions of any kind of model to boost the accuracy of a recommender systems.

References

[1] N. Hug, “Surprise, a Simple Recommender System Library for Python,” 2017, [Online; accessed 21-December-2017]. [Online]. Available: http://surpriselib.com/

[2] Y. Koren, “The BellKor Solution to the Netflix Grand Prize,” 2009.