Recommender System Report

Ruoting Liang , Tianpei Shen , Yuyi Zhou

Abstract

In this project, the team trained models on 2038130 user-item pairs with true ratings provided in train_rating.txt, from the Yelp dataset, to predict ratings for 108024 user-item pairs provided in test_rating.txt and evaluate the RMSE score on test_rating.txt on Kaggle. Three models, Singular-Value Decomposition (SVD), Factorization Machines (FM), and customized methodology to incorporate review text were experimented in this project. Performance of each model was evaluated using root mean square error. Finally, the team ensembled SVD and FM by averaging the rating score. Individually, both SVD and FM have excellent performance in the prediction, and the lowest RMSE on Kaggle was achieved by ensemble method.

1 Approaches & Techniques

1.1 SVD

1.1.1 Model Description

SVD is a model-based collaborative filtering (CF) algorithm in the recommender system. It is a matrix factorization method that decomposes the rating matrix of size N × M to two smaller user feature matrix of F × N and item feature matrix of F × M where F is usually smaller than 30. It is noted that the conventional SVD algorithm cannot handle missing value and it is not efficient at handling high-dimensional matrix. Alternatively, the python library Surprise[1] is capable in handling this issue and was used in this project.

Assume the predicted rating of user u to item i is $\hat{r_{ui}}$, item i vector and user u vector are q_i and p_u, therefore the predicted rating is set as

$$\hat{r_{ui}} = \mu + b_u + b_i + {q_i}^T p_u$$

The SVD model in the Surprise minimize the following regularized squared error and the minimization is performed by using stochastic gradient descent[1].

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r_{ui}})^2 + \lambda({b_i}^2 + {b_u}^2 + \|\vec{p_u}\|^2 + \|\vec{q_i}\|^2)$$

1.1.2 Hyper-parameters tuning with Surprise

There are 12 hyper-parameters in the SVD model in Surprise: n_factor, n_epoches, lr_bu, lr_bi, lr_pu, lr_qi, reg_bu, reg_bi, reg_pu, reg_qi, int_mean, init_std_dev. The gridSearch and 10 fold Cross Validation in Surprise were used in tuning those parameters. Each model simulation took approximately 30 minutes to complete. The lowest 10-fold-C.V. RMSE achieved before tuning n_factor was 1.2563, with the score on Kaggle was 1.29924.

After reducing n_factor, the RMSE was significantly improved. The best n_factor chosen is 1. Regulation is also a very important parameter. Increasing reg_bu and reg_bi will decrease 10-fold-CV RMSE. With more epochs, low learning rate, and a bit high regulation, the best model was selected with parameters n_factor=1, n_epoches=45, lr_bu=0.004, lr_bi=0.008, lr_pu=0.0015, lr_qi=0.0000025, reg_bu=0.24, reg_bi=0.24, reg_pu=0.055, reg_qi=0.0085, int_mean=0, init_std_dev=0. This model resulted in the RMSE of 1.2525 in 10-fold-CV, and Kaggle score of 1.29273.

1.2 Factorization Machines

The Factorization Machines model is used in this project to deal with date and time feature.

1.2.1 Model Description

Factorization machines (FM) combines the advantages of Support Vector Machines with factorization models [2].It can mimic most of the factorization models including TimeSVD++[3].The FMs model all interactions between variables using factorized parameters. Thus, it is optimal at dealing with high sparsity data like recommender systems[2]. The input of FM is real valued feature vector[2]. The FM model of order d=2 is defined as[3]:

$$\hat{y}(x) := w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{j,f} v_{j',f}$$

where k is the dimensional of the factorization and the model parameters are[3]:

$$w_0, w_1, ...w_p, v_{1,1}, ...v_{p,k}$$

Optimizing model parameters with regulation is defined as[3]:

$$OPTREG(A, \lambda) := \underset{\Theta}{argmin}(\sum_{(x,y)\in S} l(\hat{y}(x \mid \Theta), y) + \sum_{\theta\in\Theta} \lambda_\theta \theta^2)$$

There are three learning methods for FMs: stochastic gradient descent, alternating least-squares, Markov Chain Monte Carlo(MCMC).

1.2.2 Hyper-parameters tuning with libFM

In this approach, the libFM[3] was used to handle the date and time feature. The libFM is an open source library which implements FM and supports all three learning methods. Among all libraries that implement FM, the libFM performs the best in terms of functionalists, convenience, accuracy and run time.

The input format of libFM is as follows "<rating> <feature1-non-0-index:1> <feature2-non-0-index:1> ....<featureN-non-0-index:1>". The user_id, business_id are converted as well as the date and time feature. The date and time are converted into categorical type of data as day of week, month of year, day of month, week of year, day of year. For example, the first line in the train_rating.txt is converted to "5 0:1 693209:1 838514:1 838530:1 83855:1 838609:1 838938:1". There are 838980 features after conversion. Both train_rating.txt and test_rating.txt are converted.

The train_rating.txt was randomly split into 80% data as the train set, with the remaining 20% data saved as the validation set. The libFM uses RMSE error. The result of using libFM on both the train and validation set with default parameters was 1.10526 on the train set and 1.26153 on the validation set respectively. The default parameters are: -method MCMC, -iter 100, -dim (1,1,8). By following the libFM's manual for tuning the parameters, -init_dev was tuned with values (0.1, 0.2, 0.5, 1.0), with the value of 0.1 achieving the best result. In libFM, features can be grouped. The project team created a group.txt and it is used by libFM to regularize each grouped feature. By using group, with default parameters, the RMSE was 1.11666 on the train set and 1.25776 on the validation set. This result performed better than the one without using group on the validation set. With next step tuning dimension, the result showed that low dimension is better than high dimension, with the final dimension selected to be (1,1,2).

In addition, the SGD learning method was also experimented. However, the result was poor and discarded. The final trained model uses -dim (1,1,2), -iter 500, using group. This resulted in a RMSE of 1.11361 on the train set and 1.25701 on the validation set. The RMSE 1.25701 on the validation set is a bit higher than the best SVD model.

After the team got the score of 1.29184 (ensemble with SVD model) on the Kaggle Leader board, the team tried another approach to split the train and validation set for the FM model. The team noticed that the test_rating.txt only contains months from January to July. As a result, 20% data with month from January to July was randomly selected as the validation set and the remaining data was used as the train set. After several iterations and tuning of parameters (similar to the previous approach), the parameters of the best model for this train-validation set is different from the random split train-validation set. The RMSE on the validation set was 1.25033, which is much lower than the best SVD model. Hence, the team uploaded the prediction result from this FM model to Kaggle and received a score of 1.29267, which has a lower error than the Kaggle score of the best SVD model of 1.29273. This model uses parameters -dim (1,1,2), -iter 255, -seed 7, -init_stdev 0.2, using group.

1.3 Text Mining

In this approach, the team attempted to extract features from review text with some text mining techniques and use the extracted features to train a Machine Learning model to predict the rating of a user gives to a business.

1.3.1 Text Data Processing

The team needed to preprocess the review text before extracting useful features. The processing mainly comprises of the following three steps: tokenizing the text, removing stop words, and stemming the text.

The first process requires to convert the text to its atomic elements in lower case by removing any non-word character such as symbols or white spaces. For example, “Hello world!” will become [“hello”, “world”].

A stop word is a commonly used word (such as “a”, “the” and “at”,etc) that are distracting and non-informative. In this project, a Python package *stop_words* was used to filter out all the stop words in a document. As a result, noise information and memory overhead was reduced. It may also potentially improve power of prediction.

Finally, the inflected words to their word stem was reduced by stemming a document. For example, “catlike”, “catty” are based on its root word “cat”, which both are related to “cat”. By applying this filter, memory requirements were further reduced. Additionally, all words that has length less than or equal to 1 were removed since single character usually does not have any significant meaning.

1.3.2 Review Text Features

Defining d_u as documents for user u and d_b as documents for business b, built two feature matrices for user and business separately.

$TFIDF(t) = TF(t) * IDF(t)$[4] was used to measure importance of a word, while TF measures how frequently a term appears in a document and IDF measures how important a term is with in all documents.

$$TF(t) = \frac{\text{Num of times term t appears in a document}}{\text{total num of terms in a document}}$$
$$IDF(t) = \frac{\text{total num of documents}}{\text{num of documents with term t in it}}$$

Subsequently, a $TfidfVectorizer$ in *scikit-learn* for user documents was built to extract the top 500 features based on their $TFID$ score. This produced a matrix $U \in \mathbb{R}^{n\times 500}$, where n is the number of distinct users. Similarly, a feature matrix $B \in \mathbb{R}^{m\times 500}$, where m is the number of distinct businesses, was produced for the distinct businesses. The two matrices from $TfidfVectorizer$ are then combined and fitted into the Latent Dirichlet Allocation model (LDA)[5] with 10 topics. The final feature matrix for users is defined as $U_{lda} \in \mathbb{R}^{n\times 10}$ and feature matrix for business is defined as $B_{lda} \in \mathbb{R}^{m\times 10}$

Hence, for one given training case, training features of $\mathbb{R}^{20}$ were built by searching related rows for the user *user_id* in U_{lda} related row for the *business_id* in B_{lda}. Training targets will be set as the rating value of the training case. In addition, as there are no cold start issues identified in the test data set, a feature matrix for testing data can be built such that it can be used to predict test targets in the future.

1.3.3 Training Model

To train the model, the *train_rating.txt* was divided into the training data set(80%) and the validation data set(20%). The Random Forest Regressor was then used to train the model. The best RMSE achieved in this model on the validation data set was 1.36.

The performance from the text mining component is relatively poor compared to the SVD and FM models. This may be due to the fact that a large number of features in $TfidfVectorize$ were not incorporated due to memory limitation.

1.4 Ensemble

Since a decent SVD model and FM model using time as the feature were produced, the average of predicted results from both models were uploaded to Kaggle. The results on Kaggle were surprising. The 1.29307 on Kaggle was from ensembling the best SVD model and the FM model without using group. The 1.29226 on Kaggle was from ensembling the best SVD model and the FM model with using group. Finally, the 1.29184 was derived using the same method as 1.29226, but with a different seed in the FM model.

After a score of 1.29267 on Kaggle was achieved from the FM model using the validation set following the same pattern as the test set, we ensembled this model with the best SVD model using their average predicted results. The ensembled result uploaded to Kaggle got score of 1.29168, which is the final best RMSE score.

2 Final Submission

We decided to choose two uploads as our final submission. One is *ensemble_k2_255_svd.csv* which produced the lowest score of 1.29168 on Kaggle. The other is *ensemble_3.csv* which produced the second lowest score of 1.29184 on Kaggle. Based on their excellent performances on the train_rating.txt and 30% of the test data on Kaggle, the team expects that either of the uploads will have excellent performance on the remaining of the 70% test data.

Reference

[1] Nicolas Hug. Surprise, a python scikit for recommender systems.

[2] Steffen Rendle. Factorization machines. *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 995–1000, 2010.

[3] Steffen Rendle. Factorization machines with libFM. *ACM Trans. Intell. Syst. Technol.*, 3(3):57:1–57:22, May 2012.

[4] Hong-Bo Shi Li-Ping Jing, Hou-Kuan Huang. Improved feature selection approach tfidf in text mining. *Computer*, November 2002.

[5] Jure Leskovec Julian McAuley. Hidden factors and hidden topics: Understanding rating dimensions with review text. *Computer*, pages 165–172, October 2013.