Midterm Report

Kaggle ID: Nee2641 **Name:** Lihao Zheng **UID:** U18942673

Work flow

- **Finding key features:**

  Text part:

  The most important feature is the text part of data, ‘Summary’ and ‘Text’, which is the most related to the score of the review. It contains the most information of sentiment from users to movies, so I do latent semantic analysis to it by TFIDF (Term Frequency-Inverse Document Frequency) for feature extraction.

  I combine ‘Summary’ and ‘Text’ as a new feature ‘Review’ because they have overlapped information. Then I split the strings to lists of words, and clean data, such as deleting special characters and stopword (the stopword list downloaded from NTLK library). I also use WordNetLemmatizer (ref: NTLK) to return the words unchanged.

  To extract key words and do processing of TFIDF, I use TfidfVectorizer (reference: sklearn) to transform word list to sparse matric of TFIDF for the following train.

  Numerical part:

  I get new features ‘Helpfulness’ (Helpfulness Numerator divided by Helpfulness Denominator) , and ‘ReviewLength’ (Length of ‘Text’) for following train.

- **Model Selection:**

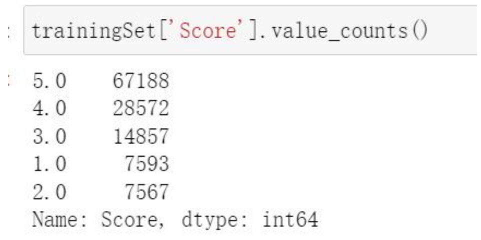
  I divide two parts of the training : Language Model and Error Model.

  Language Model:

  For the first model, I think the score information is almost completely based on this part, so my final prediction is also completely based on the prediction result of this part.

  I finally combine LightGBM(Light Gradient-Boosting Machine, LGBM) (ref: LightGBM), a regressor based on Decision Tree and AdaBoostClassifier (ref: sklearn) with Naive Bayes.

I choose Boost algorithm because it is effective to a unbalanced dataset whose most data belong to some special class.



```
trainingSet['Score'].value_counts()

5.0    67188
4.0    28572
3.0    14857
1.0     7593
2.0     7567
Name: Score, dtype: int64
```

Both of Naive Bayes and Decision Tree train fast for a large dataset (with approximate parameters), and it is easy for fine-tuning. I abandon SVM model just because it takes too much time for training (even split trainset into many small parts).

I choose LightGBM because regressor predicts the continuous data, who performs great in MSE (Mean-Squared-Error). It also has better speed performance than other Gradient Boosting regressor.

I combine them and set the weight LightGBM : Naive Bayes = 2:1, just for the lowest MSE, to get the prediction.

Error Model:

From previous model, I get the base prediction of the score, but it also has some error. I assume the other features are weak-correlation, and the error is from them, so I use them just for compute the error.

I choose the LightBGM to fit. X_train is the difference between of the prediction score of the previous model and true score and y_train is true score. LightGBM predicts continuous data and is effective for reducing MSE. It also trains fast.

The prediction of language model + the prediction of error model is my final prediction.

- **Parameters tuning**

For the parameter controlling the dimension (max_feature) of TFIDF, I set some possible values and test them in the language model, to keep as much as possible latent semantic information, and final choose 9000 as max number of key words.

I do a for loop to find the best parameters for alpha and learning rate of Boost algorithm with Naive Bayes because it runs fast.

LightGBM has great performance, and I just set the max number of leaves.

- **Model Validation**

  I split the train set into 2 part, 20% data for validation. For these data, I use them to reduce MSE and accuracy, to simulate the model performance to the final test set. Normally, better performance in the validation means better performance in the final test set.

  Result:

Model	**LightGBM 1**	**NB(Boost)**	**Combined (LGBM1 + NB)**	**LightGBM2 (final result)**
MSE(validation)	**0.70**	**0.89**	**0.647**	**0.61**

- **Some problems and solution of improving**

  For example, at first, I am confused if I should choose all data of train set to train my model. It may be useful, yet some complex algorithm has too much time-cost and at that moment, some simple classifiers can not get a great performance of evaluation metrics. I finally find to use continuous prediction to reduce MSE, and choose Decision Tree to do regression to make prediction whose train process is also fast if the parameter is approximate, so I avoid complex model.

  I also find the Naive Bayes classifier (it is simple so I can use boost algorithm), has a great performance in accuracy, and I should choose one of NB and LightBGM as Language Model, but finally I combine them because I think the NB could provide information from probability distribution, and Decision Tree could get other detail information, which is not overlapped, so I combine them and get great performance (as the form above, combined model).