Amazon Movie Review Prediction

Kaiyan Xu

Preliminary Analysis:

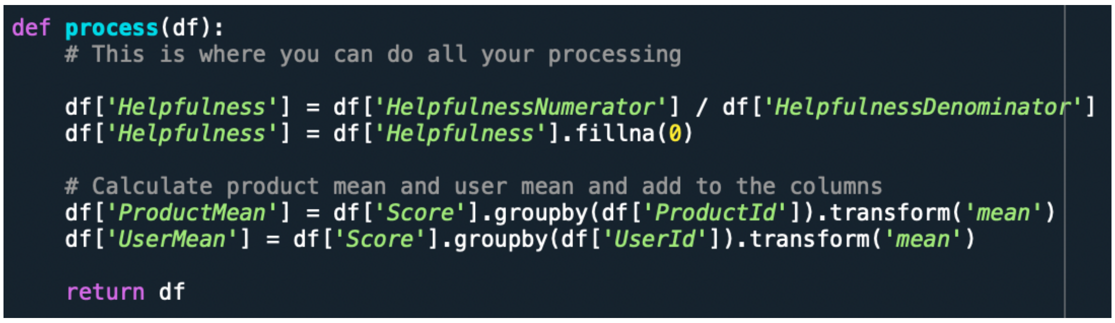
To have a general idea of score distribution, I count the amounts of each score, and present them in the bar chart (in the Jupyter Notebook). From the chart, it is obvious that the distribution is heavily positive skewness: 5 and 4 dominate and 5 has much more counts. Also, after submitting predictions with constant score 4 and 5, I found that a constant prediction of 4 has RMSE 1.20, which is smaller than RMSE of 5 (1.58). This suggests that in the submission file there may be more 4 than 5.

I also plot graphs showing the 5 movies that have highest amount of review and the 5 users that review the most movies. This indicates that many users rate more than one time. Therefore, users' average rating and movie's average rating may be good indicators for making prediction.

Also, after randomly viewing some summary and text, I feel the messages convey similar perspectives and emotion of each user. As a result, it might be better to combine them together for text message analysis.

Feature Extraction:

Based on the idea of preliminary analysis, I generate two new columns "ProductMean" and "Usermean" in the "Process" function in feature_extraction.py, as shown in the figure. This way, average movie rating and average user rating appear in both test file and submission file.



```python
def process(df):
    # This is where you can do all your processing

    df['Helpfulness'] = df['HelpfulnessNumerator'] / df['HelpfulnessDenominator']
    df['Helpfulness'] = df['Helpfulness'].fillna(0)

    # Calculate product mean and user mean and add to the columns
    df['ProductMean'] = df['Score'].groupby(df['ProductId']).transform('mean')
    df['UserMean'] = df['Score'].groupby(df['UserId']).transform('mean')

    return df
```

In the test file, I firstly drop the rows which are NaN in text and summary columns, because the key in my analysis relies on text information. I also cut half amount of the review information that has score 5, so the distribution is less imbalanced, and thus it is more likely to represent the real score distribution in the submission file. Then, I combine text and summary columns to a new column called "Comment" with a comma inside each pair. Comment, average movie rating and average user rating become my variables to process and later fit the model.

Flow, Decisions, and Techniques:

In the beginning, I chose to conduct sentimental analysis, I iterate over the comment and assign a score between -1 (negative) to 1 (positive) for each column (online source, 3.6). I tried KNN (lowest RMSE = 1.12), Naïve Bayes and Logistic regression. However, the accuracy is outcompeted by using TfidfVectorizor later.

To process the text messages, I choose to use TfidfVectorizor to turn the comments into vectors and store as a scipy sparse matrix. The method combines the measurement of how often a word occurs and how important it is. TfidfVectorizor is not case sensitive and does not count punctuation by default. However, I think “!” and “?” are characteristic punctuation to convey the emotion and thus good indications of rating scores, so I do not let TfidfVectorizor to filter them by specifically setting its parameter. To consider another two features: average movie rating and average user rating, the two features should be converted to a csr_matrix as well. I turn them into csr_matrix and merge them with the text csr_matrix.

Next step is to choose classifier. I did not try KNN because the matrix has huge dimension after vectorize the text messages. Therefore, most points are nearly the same distance from the test point by calculating Euclidean distance. I did not try Naïve Bayes because there are 0 entries in the rows, which overrides all the other factors and yields a zero probability, and it is hard to replace zero with the most appropriate small number. I want to use Random Forest to train the model because it consists of a large number of individual decision trees and can improve accuracy and avoid overfitting compared to decision tree model. However, I ran over two hours without returning result. I expected it would be even harder and time consuming if I would like to improve the model in the future, so I did not choose the random forest model.

The two classifiers I tried are support vector machine and logistic regression. Before that, I tried to normalize the scipy sparse matrix consist of two average measurements, so that every column adds up to 0 and it does influence heavily to the general model with a unit change (later I discarded normalizing the csr_matrix because I found it prominently increase my RMSE). I tried both linear and nonlinear SVM model. I competed the RMSE and accuracy/recall of the two classifiers and found that logistic regression is better in the default setting.

Model Validation:

I should change both the hyperparameters of SVM and logistic regression because performances of classifiers are different with different parameter, so it is not necessary that logistic regression is always better than SVM. However, because fitting with SVM is time consuming, I only conducted validation of logistic regression.

To get the best logistic regression model, I modified the percentage of cut of score 5 from 0.5 to 0.6 and 0.65. This way, amount of score 4 in training set is more than amount of score 5, closer to the predicted situation in the submission file. Then I tried several C, the parameter that specifies regularization, and found that C = 4 gives lowest MSE in my sample. I also found that increasing max iteration lowers MSE and improve precision and recall in the confusion matrix. Change to CountVectorizor and normalize the scipy sparse matrix does not help. Here are part of the trials that I still have records:

a. Change percentage of cutting score 5:

MSE(cut 60%, C = 4, max_iter = 1000) = 0.774, precision = 0.56, recall = 0.53

MSE(cut 65%, C = 4, max_iter = 1000) = 0.776, precision = 0.56, recall = 0.53

b. Change C (Inverse of regularization strength) with the same train_test_split:
MSE(C=5, type=L2, max_iter=100) = 0.927
MSE(C=4, type=L2, max_iter=100) = 0.892
MSE(C=0.01, type=L2, max_iter=100) = 0.935, precision = 0.55, recall = 0.47
MSE(C = 10, type = L2, max_iter = 100) = 1.06, precision = 0.52, recall = 0.45

c. Change maximum iteration:
MSE(C=4, max_iter=300) = 0.772, precision = 0.57, recall = 0.53
MSE(C= 4, max_iter = 500) = 0.770, precision = 0.57, recall = 0.53

d. Change to CountVectorizor:
MSE(C=4, max_iter=1000) = 0.940, precision = 0.52, recall = 0.50

e. Normalize numerical column, so that they add up to one:
MSE(C=4, max_iter=300) = 0.803, higher than 0.772

Creativity and Challenges:

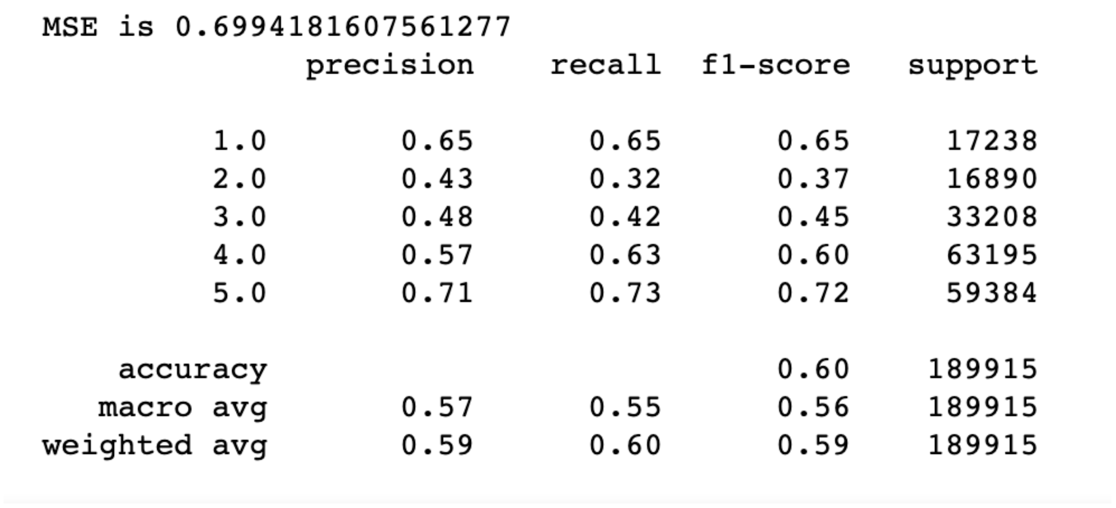
The challenge in the preprocessing is that I did not use stemming, like "Snowball" stemmer to filter the related words. Also, I did not find an effective way to normalize the scipy sparse matrix, which makes the model susceptible to a unit change of small variables.

Secondly, use of TfidfVectorizor remains discussion. Movie reviews should have a lot of emotional words that happen frequently, like "love", "fantastic", "awful", which are helpful for predicting. However, they may be treated equally with those meaningless words like "the", "he". Changing to CountVectorizor does not help because it is heavily influenced by those meaningless words. If I have more time, I should filter these meaningless words by using list of stop words that can be found online (here) and build my own word bag according to the frequency of left words to evaluate the emotion of the comments.

Thirdly, I did not try enough classifiers, mostly limited by the time, and it is partly hindered by my workflow. Logistic regression assumes linear relationship between the dependent variable and the independent variables. It will not perform well with independent variables that are not correlated to the target variable and are very similar or correlated to each other.

In this project, I have a chance to overview the advantages and disadvantages of different classifiers. It is also my first time to process the text messages using TfidfVectorizors or Sentimental Analysis. I need to review the text carefully, and make sure that my model can be more representative, like by adding "?" and "!" when vectorizing the languages. The most challenging part in the process is to figure out the scipy sparse matrix. What it is and how to combine other features inside to fit X. Now, I know how it works and its benefit and function in machine learning.

This is the MSE and the confusion matrix of my final model



```
MSE is 0.6994181607561277
              precision    recall  f1-score   support

         1.0       0.65      0.65      0.65     17238
         2.0       0.43      0.32      0.37     16890
         3.0       0.48      0.42      0.45     33208
         4.0       0.57      0.63      0.60     63195
         5.0       0.71      0.73      0.72     59384

    accuracy                           0.60    189915
   macro avg       0.57      0.55      0.56    189915
weighted avg       0.59      0.60      0.59    189915
```

Online source:
Ultimate guide to deal with Text Data (using Python) – for Data Scientists and Engineers. https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/#h2_6

Stopwords. https://www.ranks.nl/stopwords