**Can Erozer CS506 Midterm Report**

**First Look to the Dataset:**

In a 139753 rowed dataset, there are one null value in Text and Summary columns. And there are 17470 null values in the Score column. (See appendix 1) I could have just dropped the rows that have null values in Text and Summary columns, but I instead decided to fill the null values with empty string. (see fill\_nas() function in starter\_code)I could have trained a N-Gram model for each class and then fill the null values according to their class. But that would have added extra complexity and it wouldn’t be reasonable to do such thing for only two null values.

When it comes to 17470 null values in the Score column, I have nothing to do but drop these rows (I dropped it after X\_Test file is created since not doing so resulted an empty X\_Test file). (see drop\_nas())

There is a great imbalance between the classes of Score column. (See appendix 2) I decided to under-sample my data. (See appendix 3) However, since there is a huge difference between the numbers 65313 and 7309, doing so led me to lose so much information about the nature of the data. (The RMSE score with training my model with under-sample data was 2.99) The other way to deal with this problem was trying to over-sample my data. But again, I didn’t oversample my data because the gap between classes was too much. Therefore, I proceeded with the original dataset, with the assumption that class\_weights parameter in the model would deal with it by putting different weights/ importance the classes during training.

**Finding Features:**

* Helpfulness and Review\_Length features were already given in template.
* I realized that products and users appear more than one time in the dataset. To do this, I counted the number of occurrences of product and users by using default dictionary. And then added the corresponding number of occurrences to the rows according to their UserId or ProductId. (see product\_popularity() and user\_popularity())
* I thought that number of question and exclamation marks could reflect some emotion in the text. So, I counted the number of occurrences question and exclamation marks in the Summary and Text. I was considering to merge Summary and Text columns. But I didn’t merge them because each of them has, though little, correlation to the score. (I will show the correlation matrix in latter sections)
* I thought that the use of capital letters could also reflect some emotion. For example, a text in Summary is “A GOOD MOVİE!!!”. Even though “good movie” is a positive review but not an extremely positive review, this user scored 5.0 to the product. I spotted plenty of examples like this. So, I decided to add the frequency of capital letters in the text. (Not the number of capital words since the length of the text can vary.) (see add\_capital\_freq())
* I found a large list of positive and negative words from nltk’s opinion lexicolon. And I counted the number of positive and negative words of nltk in both Summary and Text columns. I realized that could be deceptive since, for example, “not great” is not a positive review. So, I checked at the one previous word of these positive and negative words. If there is “not” word behind them, I counted them as their opposite emotion. But then I realized that not could appear in the form like “don’t like”. So, I did lemmatization (my own implementation that I will further describe in the next sections) before checking the appearance of these words. (see sentiment\_analysis1())
* In my research, I found that there is another sentiment analysis tool called polarity and subjectivity. So, I used textblob library to calculate the subjectivity and polarity of bot Summary and Text.
* I found out that the values in Time column can be converted to a formatted value. So I extracted the year that the review was written using numpy’s to\_datetime() method. (see add\_time\_year())
* In my research I found out that there are many sentiment analysis tools. But most of them provide basic information like positive-negative-neutral. But I think this is alone not enough to classify a five-classed label. Therefore, I found a tool that gives an information about the text that has more variability. In afinn library, I used a method called afn.score() that returns a score between -5 and 5. In order to not lose any information from the text I first l did lemmatization to the text. (see sentiment\_score())
* In order to vectorize the texts I found out that I can use tf, tfdif, embedding methods. But I couldn’t find any library about embeddings that doesn’t use deep learning techniques. So, I proceeded with tf and tfidf. They basically create a vector based on the unique words in the corpus. To perform this, I first cleaned the text: I got rid of punctuation marks. There were really weird use of punctuation marks that built-in cleaners couldn’t deal with it so I decided to wrote my own cleaner. (see special\_remove()). See the transformation of the text before and after special\_remove(). And to reduce the dimension of the vector and I got rid of stop words that don’t give any information about the nature of the text. (see remove\_stopwords()). And lastly, I stemmed and tokenize the text using nltk’s library. (see stem\_tokenize()).
  + In order to decide whether to use tf or tfidf, I applied both of them to the text. And then I reduced the dimensions to 2-D to plot the vectors using SVD. I randomly sampled 40 rows from each class and then assigned different colors to each class. (Blue: 1.0, 2.0: red, 3.0: orange, 4.0: purple, 5.0: black) (see appendix 5) For Summary the tf vector is at appendix 6. For Summary the tfidf vector is at appendix 7. For Text the tf vector is at appendix 8. For Text the tfidf vector is at appendix 9. We can observe that the grouping in the plot is not very intuitive. But we can make an inference that tfidf vectors are sparser- they are grouped more spread. But tf vectors are denser- they are grouped closer. This inference can alter as we increased the dimensions. So, I decided not to make an inference from these plots and try tf and tfidf differently in my model. (At the end tfidf yielded greater RMSE score.) At the end, I didn’t include tfidf vector for Summary column. Because the resulting vector is so sparse (too many zeros). This is because there are few words in each Summary column.

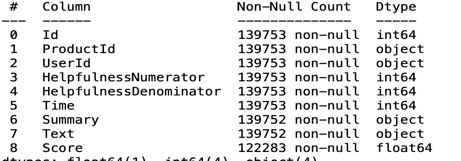
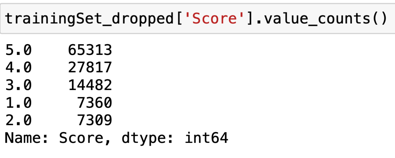
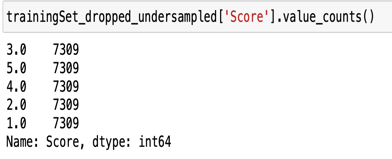
After creating all the features, I investigated the correlation between my features and the Score column. (See appendix 10) I first tried to eliminate the features that have smaller than 0.1 correlation. But removing them resulted to a smaller RMSE score 1.22. So, I checked if the features that I created are linearly dependent or independent: I used 22 features (see columns\_to\_include list in stater\_code). Due to the calculation that I provided in stater\_code all 22 features are linearly independent. So, I thought using them wouldn’t harm my model except the run time.

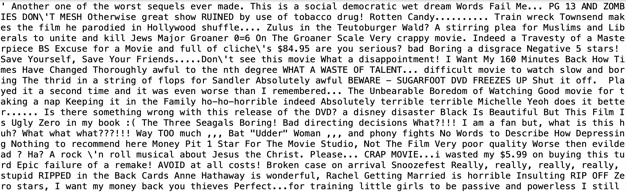
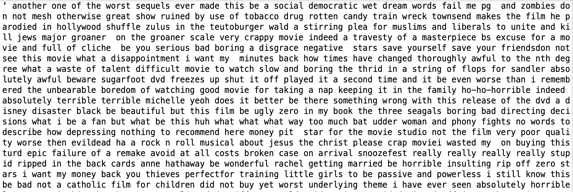
Additionally, I added the tfidf vector to the model to train. I first used SVD to reduce number of dimensions. I tried n\_dimensions=10,10,30,50,100,1000, but neither of them yielded a better RMSE score. So, I decided to use csr\_matrix to merge the features that I described above and the tfidf matrix.

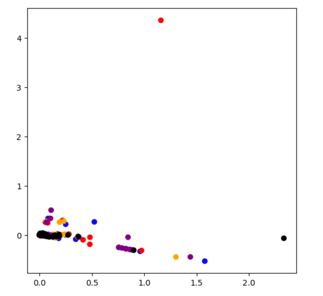
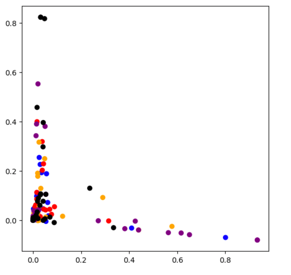
**Model Selection:**

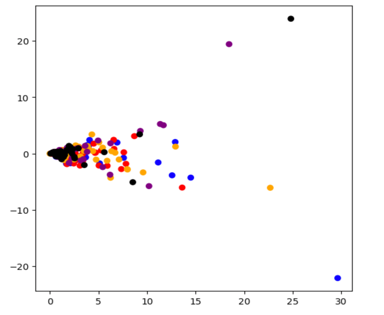
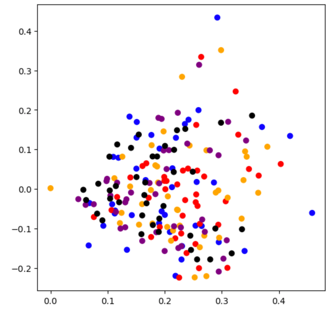
Because the number of features that I put to the model is very large, I didn’t use KNN. I instead decided to use that work well for high-dimensional data like decision tree classifiers (XGBoost), SVC, Random Forest Classifier. I also wanted to use them because they accept the parameter that takes the class weights. By this way, I thought I could deal with imbalance data. But since the dimension is so high the run time for training is extremely long (more than 30 mins) or the kernel is restarted all over again. So, I had to use less complex model. Due to the nature of my dataset, overfitting was inevitable. This because the imbalance data and the large number of dimensions. So, I used Ridge Regression Classifier which is less complex than other models and deals with the problem of overfitting. I tuned the parameter by using different alpha values. I tried alpha values=1,2,3,4,5,6,7,8,9, 10, 100. And the most optimal was 5.0. The final and the best accuracy model and RMSE value I could achieve is 0.61 and 1.108 respectively.

**Appendix:**

1) 2) 3)

4) 

5) 6) 7) 

8) 9) 10) 