Amazon Movie Reviews Analysis Report

Jiahui Zhu {buzjhcs@bu.edu}

1. Preliminary Analysis

The competition goal was to predict the star rating associated with user reviews from Amazon Movie Reviews using the available features. First come to my mind for this would be to clean up the reviews of the training data, then create a model using an available classifier such as KNN, Logistic Regression and Random Forest, and predict the new star ratings using this model. After taking a look at the graphs made by Kaggle on the training data star ratings (as shown below), I noticed an overwhelming amount of 5 star reviews. Therefore, I assumed about the same would occur for the final predicted ratings as well.

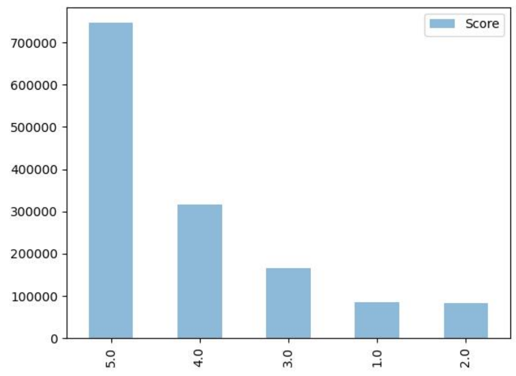
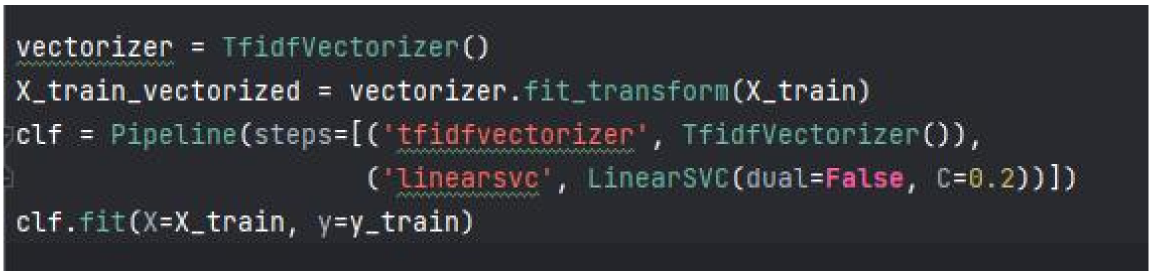


Figure.1 Score

This would help us know the different distributions for each variable. From the analysis result, we can tell that different variables have different distributions on the score, especially time, helpfulnessNumerator and helpfulnessDenominator would be more helpful for the further analysis. Since their counts for different scores are distinct.

2. Feature Extraction

I took a way that look only at the text reviews of the training data in order to create a fitted model for the test data. Another column of the training data that I considered was the summary column. However, I went off the assumption that the reviews in column 'text' would be sufficient for a decently accurate model since they are so longer and seem to be enough descriptive. Additionally, inconsistencies in the positive between the summary and the review could cause an inaccurate model. Thus, I gave up the Summary column. These reviews would be pre-processed and vectorized before being fitted. The actual features extracted from these reviews are done through the scikit-learn function TfidfVectorizer().



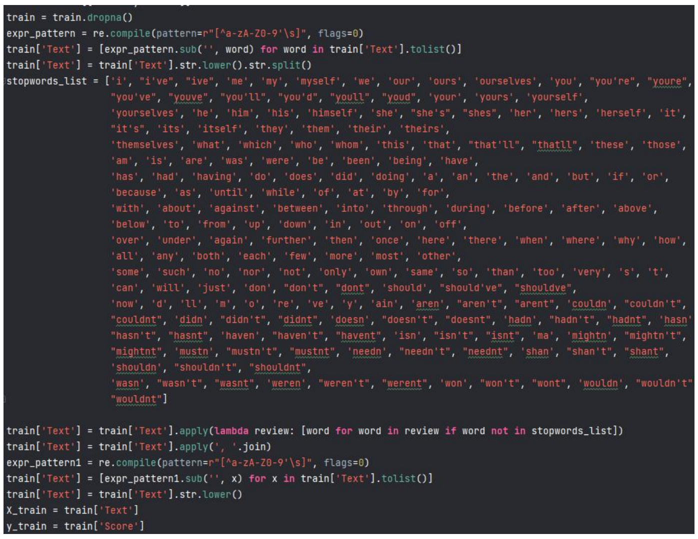
```
vectorizer = TfidfVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
clf = Pipeline(steps=[('tfidfvectorizer', TfidfVectorizer()),
                      ('linearsvc', LinearSVC(dual=False, C=0.2))])
clf.fit(X=X_train, y=y_train)
```

Figure.2 Using TfidfVectorizer()

3. Used Techniques and Experiments

3.1 Pre-processing

The flow of my code begins with the training data. It reads it from the .csv file, extracts the two features I am going to use, and drops any samples with null values. Next is the text pre-processing. This is the first hefty part of the code. The beginning of the pre-processing removes any special characters from the text. The text is then split into lists of single word strings and made lowercase. It is split into lists in order to parse through all of the text word by word and remove any stopwords, defined by a list I made going off of lists from online. I decided to go about it this way instead of using a built-in one like in Natural Language Toolkit because I wanted to easily add my own. I felt the need to add my own after noticing that the reviews contained stopwords but without proper punctuation, so a premade list would fail to remove them. The code then implodes the list of words back into strings and they are ready to be fitted.



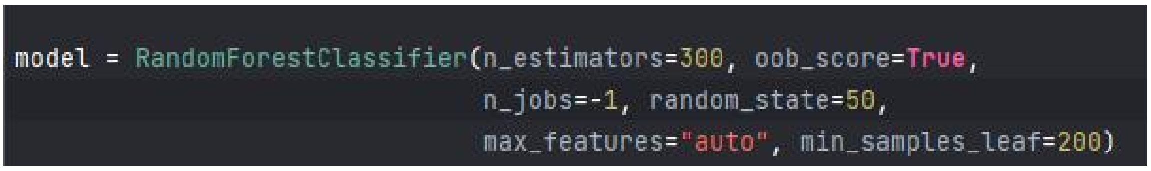
```
train = train.dropna()
expr_pattern = re.compile(pattern=r"[^a-zA-Z0-9'\s]", flags=0)
train['Text'] = [expr_pattern.sub('', word) for word in train['Text'].tolist()]
train['Text'] = train['Text'].str.lower().str.split()
stopwords_list = ['i', "i've", "ive", 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "youre",
                  "you've", "youve", "you'll", "you'd", "youll", "youd", 'your', 'yours', 'yourself',
                  'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", "shes", 'her', 'hers', 'herself', 'it',
                  "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
                  'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", "thatll", 'these', 'those',
                  'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
                  'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
                  'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for',
                  'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
                  'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
                  'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
                  'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other',
                  'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't',
                  'can', 'will', 'just', 'don', "don't", "dont", 'should', "should've", "shouldve",
                  'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", "arent", 'couldn', "couldn't",
                  "couldnt", 'didn', "didn't", "didnt", 'doesn', "doesn't", "doesnt", 'hadn', "hadn't", "hadnt", 'hasn'
                  "hasn't", "hasnt", 'haven', "haven't", "havent", 'isn', "isn't", "isnt", 'ma', 'mightn', "mightn't",
                  "mightnt", 'mustn', "mustn't", "mustnt", 'needn', "needn't", "neednt", 'shan', "shan't", "shant",
                  'shouldn', "shouldn't", "shouldnt",
                  'wasn', "wasn't", "wasnt", 'weren', "weren't", "werent", 'won', "won't", "wont", 'wouldn', "wouldn't"
                  "wouldnt"]

train['Text'] = train['Text'].apply(lambda review: [word for word in review if word not in stopwords_list])
train['Text'] = train['Text'].apply(', '.join)
expr_pattern1 = re.compile(pattern=r"[^a-zA-Z0-9'\s]", flags=0)
train['Text'] = [expr_pattern1.sub('', x) for x in train['Text'].tolist()]
train['Text'] = train['Text'].str.lower()
X_train = train['Text']
y_train = train['Score']
```

Figure.3 Text pre-processing

3.2 Model Selection

The primary model technique is KNN implemented in sklearn. I also try other models and find Random Forest perform best. I also try some different parameters of Random Forest, and finally obtain a set of optimal parameters(as shown below)



```
model = RandomForestClassifier(n_estimators=300, oob_score=True,
                               n_jobs=-1, random_state=50,
                               max_features="auto", min_samples_leaf=200)
```

Figure.4 optimal parameters of Best Model

Some other models I also have tried:

1.Logistic Regression: Relatively lower than KNN

2.Decision Tree: Relatively lower than Logistic Regression

3.KNN: base implementation, a lower score

4. Model Validation

According to the instructions, I use Kaggle pent to measure my performance.

1. KNN: about 0.2225
2. Logistic Regression: about 0.4451
3. Decision Tree: about 0.5351
4. Random Forest: about 0.55522

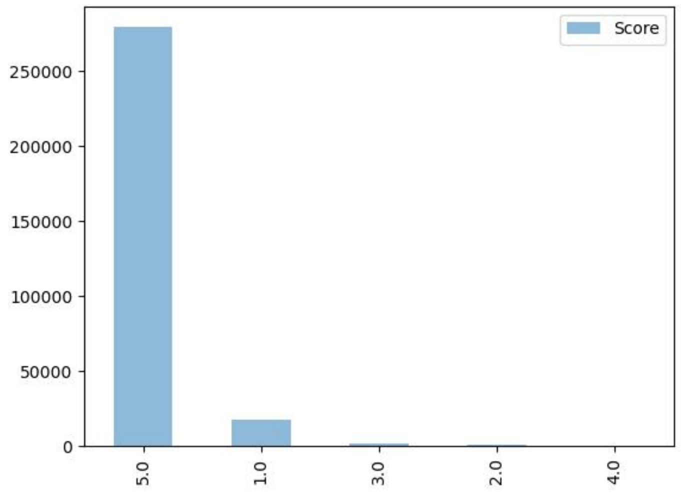


Figure.5 Best Score with Random Forest

5. Creativity and Effort

1. My initial thoughts is to focus on the text column in the data set. But how to extract features from this would be a huge problem since we are not allowed to use deep learning methods.
2. Data is quite large, so it would take a so long time to train the model and preprocess the text. If I could have more time, I would find and create some useful tricks to complete instead of the slow-speed I used in this midterm.
3. I want to try the ensemble of different models. But due to the sparse data we have, some models are really not suitable for this problem. Fortunately, Random Forest solve this and give a pretty good results.