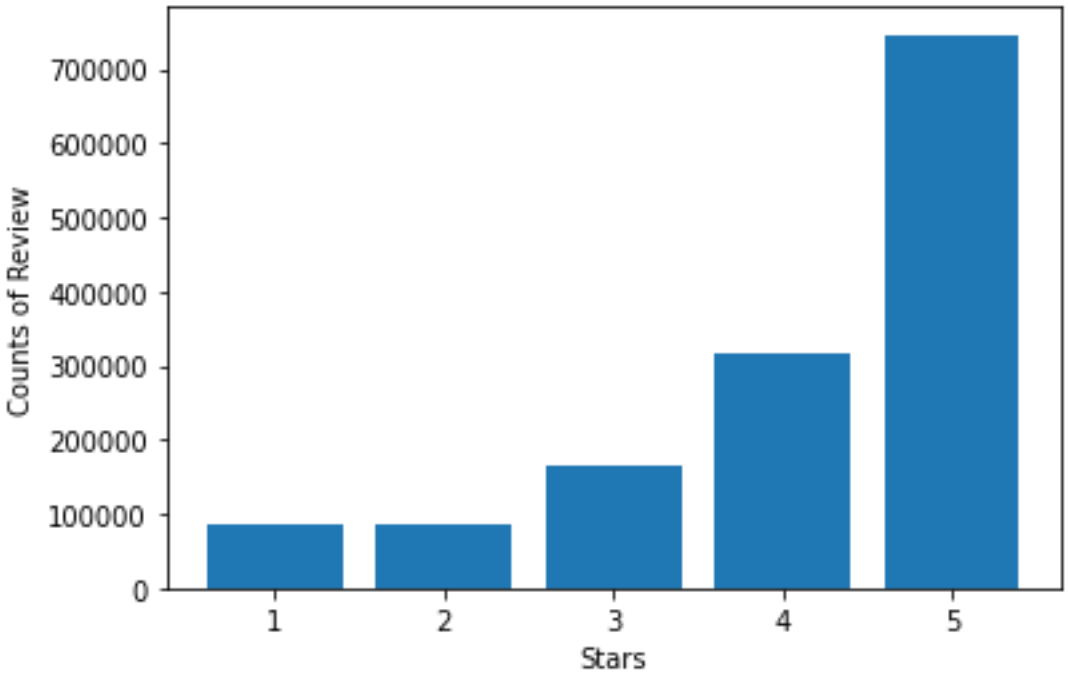
Thachathum Amornkasemwong
U55049935
March 16, 2021

CS506 – Kaggle Competition Write-up

There are altogether three jupyter files: “progress_work.ipynb”, “submit_work.ipynb”, “notebook.ipynb”. The “submit_work.ipynb” is the code I used to train the Naïve Bayes model that predicted the “submission.csv” that I submitted on kaggle. The “progress_work.ipynb” is the work-in-progress model that I did not have time to complete that might have create a model with better performance than the Naïve Bayes model trained. The “notebook.ipynb” is just the work I’ve done in testing out different classifiers and analyzing the data.

Initially I did an analysis on the training data, comparing between reviews of different ratings, to get a better understanding of the training data, which I plot onto a bar graph using matplotlib.pyplot:



The following graph demonstrates that the data is bias toward reviews with five stars rating; therefore, it should be keep in mind that it would be better to run the classifier with weights. However, intuitively, without analyzing the data, I run the Text column through the VADER sentiment analysis to get the compound VADER score, and is then ran through the kNN classification model with the “HelpfulnessNumerator” and “HelpfulnessDenominator”. However, the mean square error function returns a score of >1.5, which is almost the same score as kNN classifier with “HelpfulnessNumerator” and “HelpfulnessDenominator”. However, after further analysis of the Text column by counting the top

20 most common words of the reviews of different ratings, we can see that the reviews of different ratings have similar top most common words. Furthermore, looking through some samples of the Text, we could see that the sentiment analysis does not give the correct value corresponding to its connotation of the reviews. For example, one of the text uses “disgustingly” or other negative connotative words to describe the product in a positive way, which the sentiment analysis considered the text negative, which is wrong. Therefore, the VADER sentiment analysis is not helpful in classifying.

```
[129]: counts_5stars = Counter(list(itertools.chain.from_iterable(preprocessed_5stars)))
       print(counts_5stars.most_common(20))

       [('movie', 690284), ('one', 496403), ('film', 482891), ('great', 362394), ('like', 298343), ('good', 262921), ('love', 226647), ('time', 224599), ('story', 219123), ('dvd', 203204), ('see', 198185), ('wel
       l', 197418), ('really', 186829), ('first', 186542), ('would', 182716), ('also', 179641), ('best', 178716), ('get', 175050), ('series', 166571), ('show', 165241)]

[128]: counts_4stars = Counter(list(itertools.chain.from_iterable(preprocessed_4stars)))
       print(counts_4stars.most_common(20))

       [('movie', 368663), ('film', 323991), ('one', 251574), ('good', 199960), ('like', 182201), ('great', 132522), ('story', 128584), ('well', 116691), ('would', 115985), ('time', 114475), ('really', 112666),
       ('also', 103378), ('first', 103123), ('much', 102832), ('see', 100293), ('get', 96959), ('even', 91780), ('dvd', 83752), ('two', 79267), ('films', 75175)]

[126]: counts_3stars = Counter(list(itertools.chain.from_iterable(preprocessed_3stars)))
       print(counts_3stars.most_common(20))

       [('movie', 215819), ('film', 180331), ('one', 129359), ('like', 110555), ('good', 105508), ('would', 73641), ('story', 69799), ('much', 69522), ('really', 68174), ('time', 59408), ('get', 55742), ('first',
       54594), ('see', 54009), ('well', 53307), ('great', 53162), ('even', 53029), ('also', 48999), ('dont', 44606), ('little', 43261), ('better', 43192)]

[124]: counts_2stars = Counter(list(itertools.chain.from_iterable(preprocessed_2stars)))
       print(counts_2stars.most_common(20))

       [('movie', 125668), ('film', 92236), ('one', 69504), ('like', 63356), ('good', 45364), ('would', 39950), ('really', 38024), ('much', 36048), ('even', 35104), ('story', 34471), ('time', 30985), ('get', 3041
       7), ('see', 28466), ('dont', 26941), ('first', 26529), ('bad', 25169), ('could', 23939), ('better', 22907), ('well', 22825), ('two', 22755)]

[125]: counts_1stars = Counter(list(itertools.chain.from_iterable(preprocessed_1stars)))
       print(counts_1stars.most_common(20))

       [('movie', 124521), ('film', 64124), ('one', 59416), ('like', 53836), ('even', 36141), ('would', 36033), ('bad', 31314), ('good', 30041), ('time', 29345), ('dont', 28297), ('get', 27050), ('really', 2643
       6), ('see', 24406), ('movies', 24119), ('people', 22137), ('story', 21732), ('much', 21526), ('could', 21428), ('first', 21112), ('make', 20324)]
```

Another model that is recommend by sklearn is the Naïve Bayes model as increasing the k for kNN would increase the computation load without ensuring whether the kNN model would significantly decrease the mean square error; therefore, I switch to Naïve Bayes model that is simpler than kNN.

The Naïve Bayes model is train by vectorising the preprocessed texts with “CountVectorizor”. The texts are preprocess by removing punctuation, converting it to lowercase, and removing stop words through the “preprocess_text_nosent()” function. The Naïve Bayes gives a mean square error of around 1.3, which is the least error I get after running several other models. This work is save inside a jupyter notebook named “clean_work.ipynb”.

Another method that could significantly decrease the mean square error is by creating two models combined as one, where one is based on a Naïve Bayes model by processing the cleaned Text through TfidfVectorizer then passed it through the classifier with the Y as “HelpfulnessNumerator”. Then we need to find the cosine similarity matrix of user-features, which I did not have time to figure out.