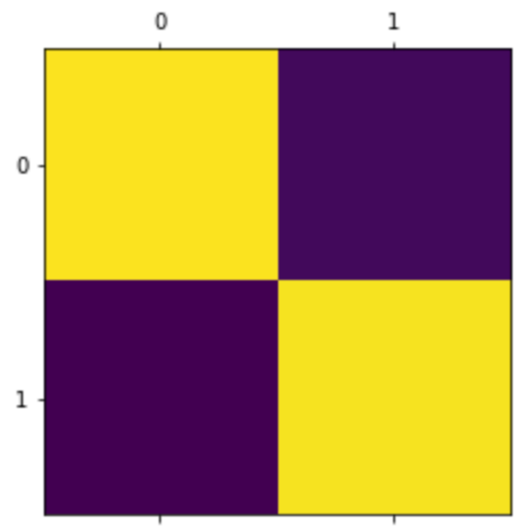
Can we get better prediction on longer review?

Group 3 Yao Chun Hsieh, Yuchen Shen, Yang Tao, Ying Fang, Nathan Hsu

Movie reviews have different lengths. Some are as long as over 10,000 words, some are less than 100 words. When applying prediction on whether the article is positive or negative, we suggest that the longer review will have better prediction since they have plentiful of information. We will use Linear SVC and KNN as our prediction model. And then we will discuss the relationship between the length of article and the prediction rate. First of all, we have to analyze our dataset.

Dataset Analyze:
There are 2000 samples in this dataset and we split 75% of them into training set and 25% into testing set. By using the given solution and changing parameters, we got a total 86% accuracy of our classification, in 500 test datasets, 216 out of 250 negative section and 214 out of 250 positive samples are correctly classified. We could find that using tf-idf methods and grid search can classify these reviews pretty good.



In order to make our result more precise from the review, we tried different methods: delete stop words, do stemming and lemmatization. Deleting stop words can ignore most useless words like ‘the, and’; Stemming and lemmatization can change words to their prototype, such as changing ‘doing’ to ‘do’ which can reduce many similar meaning words but on the other hand, this method can probably ignore some adjective word. We compared these methods to the review dataset without doing them, and found that removing stop words is useful. But stemming and lemmatization change only a little bit of the classification results and sometimes even make it worse; Adjective words are important when judging a review.

Then we needed to retrieve features from articles. We used TF-IDF, a numerical statistic, to reflect the importance of a word to a document in a collection or corpus. As a word appears in the document repeatedly, the tf-idf value will increase. We also used the frequency of the word in the corpus at the same time to offset the fact that some words appear too frequently in general.

Explanation of terms:
TF(Term Frequency):
The number of times a term occurs in a document is called its term frequency. We defined count(w, d) as the number of times a term occurs in the document(w) and size(d) as the total words of that document, then:

$$tf (w, d) = count (w, d) / size(d).$$

IDF(Inverse Document Frequency):
It is a measure of how much information the word provides. That is, whether the term is common or rare across all documents. We defined n as the total number of documents and docs(w, D) as the number of documents which contain word w, then:

$$idf = \log (n/docs(w, D)).$$

At first, we ran the TfidVectorizer class on the training data with default parameters. The min_df and max_df was 0 and 0.790586, the result was every single number or word. Then, we set min_df=0.1, max_df=0.7, the result was far less than before. Because min_df means when building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold. When we define min_df=0.1, which means our result will eliminate vocabularies whose document frequency is lower than 0.1. And max_df means when building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold. When we defined max_df=0.7, which means our result will eliminate vocabularies whose document frequency is higher than 0.7. Results like '00', '000', '0009f' in former part have been deleted, as whose document frequency must lower than 0.1 or higher than 0.7.

Later, we added parameters 'min_df=0.1, ngram_range=(0,4)' of TfidVectorizer. When ngram_range extracts the lower and upper boundary of the range of n-values for different n-grams. In this part, all 1<=values<=4 were used. Compare to the former results, we now have values with vocabularies’ length from one to four. We could find 'the rest', 'the rest of', 'the rest of the' in this part, which cannot be found in former parts, which had default ngram_range with only one vocabulary.

To clearly identify the difference of these parameters, we used wordcloud. We set min_df=0.10 and max_df=0.70 with ngram_range = (1,4) to ignore the words showing too less and too often. As a result, we got 754 words.

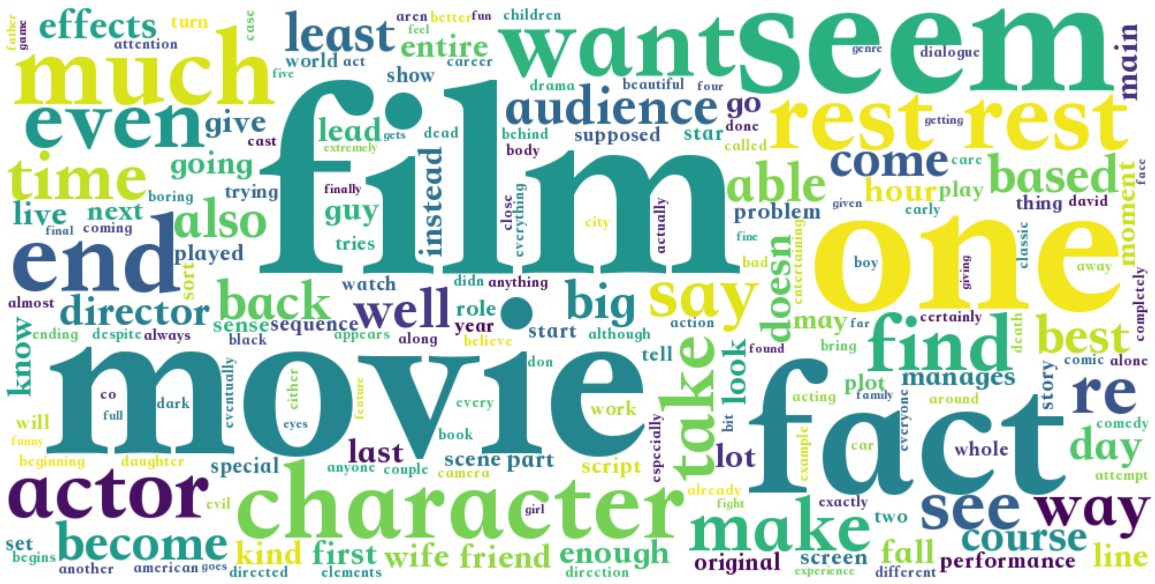
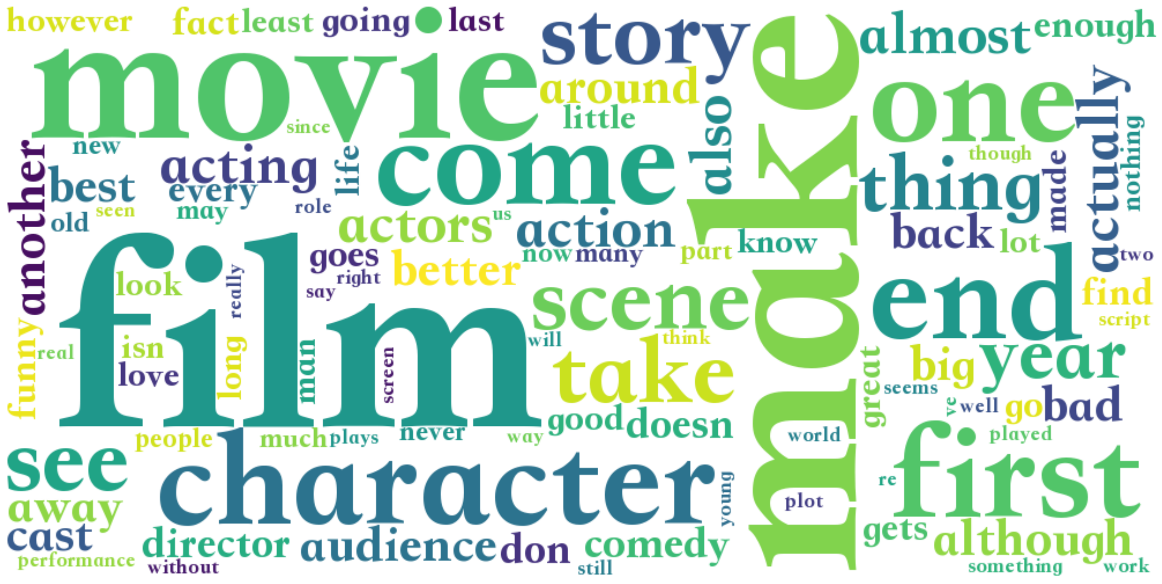


figure 1

After that, we set min_df=0.25 and max_df=0.60 also with ngram_range = (1,4) to eliminate more words which are more less or more often. This time, we got only 197 words in result.

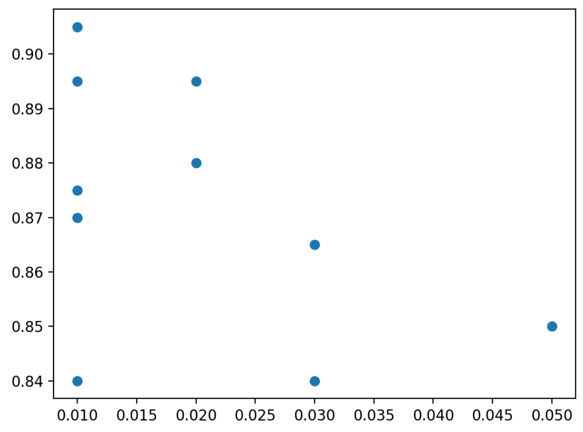


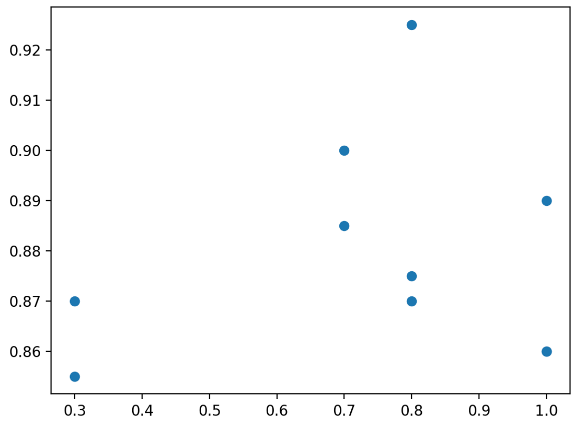
In these wordclouds, the importance of each tag is shown with font size or color. The words that are more important will be more obvious in the figure. Comparing with figure 1, figure 2 obviously has less words but with some same words in big size such as “film”, “movie”, “character” and “end”.

Optimize Parameters:
As we know the meaning of max_df, min_df and ngram_range, we make some guess about the values of them influencing the classification accuracy: First, a quite low min_df will include useful words and improve the classifier: when words have a suitable low appear rate, they are more likely unique words to represent the file and classifier can use them to distinguish positive review from negatives. Second, a large max_df can include useless words that appear in most files, but they will probably not worse the result much, because tf-idf and classifier will ignore those words. Third, taking phrases which contain two words into consideration will improve the classifier, because some single words will change from positive to negative after combine with the former one, and three words phrase always have unclear meaning.

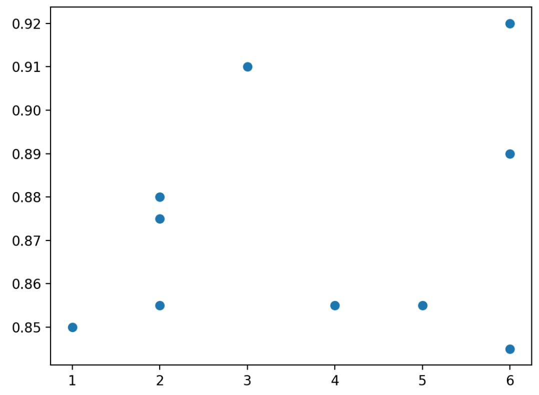
In order to make our result more precise and reliable, we applied 10-fold cross validation to the dataset which split the data into 10 parts, and used 1 part for test data and others for train data each time, repeat 10 times. This method can reduce the bias which result from splitting train_test data and make full use of dataset.

Pictures below: in our 10-fold cross validation, we choose the highest accuracy in each iteration and the correspond min_df and max_df. Using the result, we can find the suitable parameters for tf-idf method.





The result on the left shows taking the words that appears rate bigger than 10% into consideration will give a better classify accuracy. The result on the right shows that excluding word that appear rate bigger than 0.8 will increase accuracy, we choose the mean of cross validation 0.78 as our max_df and 0.01 as our min_df.



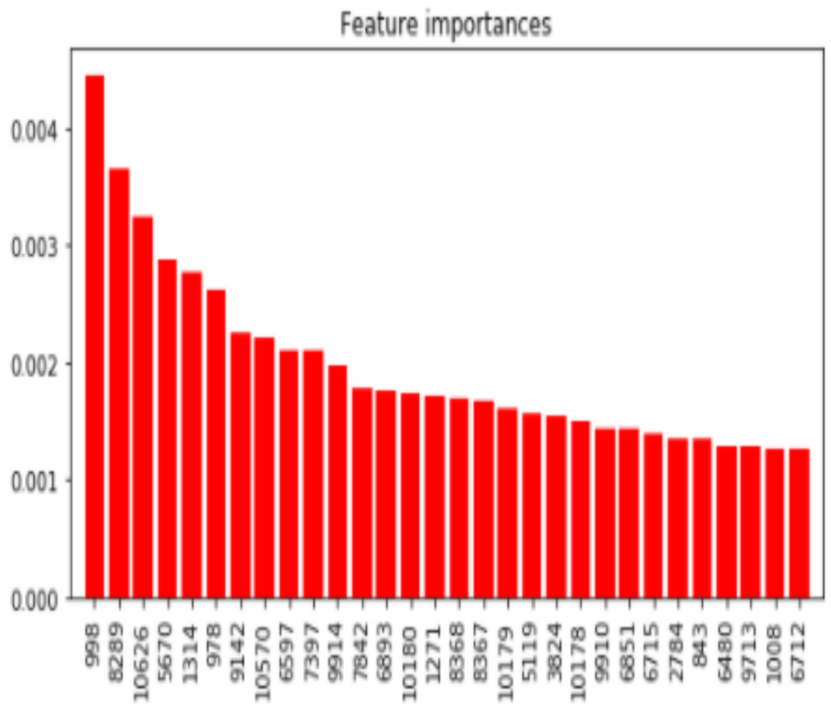
choosing different number of phrases from one to six, we find that two word phrases and six words phrases are more likely to result highest accuracy. Six words phrases actually have high accuracy, but two words phrases are far more stable.

In the following question, we chose 0.01 for min_df, 0.78 for max_df and include two words phrases to the given split using by the solution.

Decrease Dimensions and Visualization:

An attribute of a dataset can be seen as a dimension in space. That is, if a dataset has 2 attributes for each data, we can use an attribute as x-axis, the other one in y-axis, and plot the data in a 2-dimension plane. What about a dataset possessing a great number of attributes (in our case, more than 10 thousand attributes)? Generally, it's hard for human being to observe and explain patterns with data more than 3 dimensionality. Therefore, if we want to find potential trend or patterns simply with observation, dimensionality reduction method should be implemented.

Before jumping into advance dimensionality reduction methods, let's try if we can select feature manually and draw a desirable 2D plane. First, we analyze the importance of each data attributes. The result is as below. For all 10 thousand attributes in our dataset, attribute with column index 998 is the most important attribute, and attribute with column index 8289 is the second importance. Notice, the highest ratio is under 0.005, which means even the main attributes have no domination among others. If there are no significant attributes available, it is hard to find a good combination of attributes for the 2D plot directly by manual.

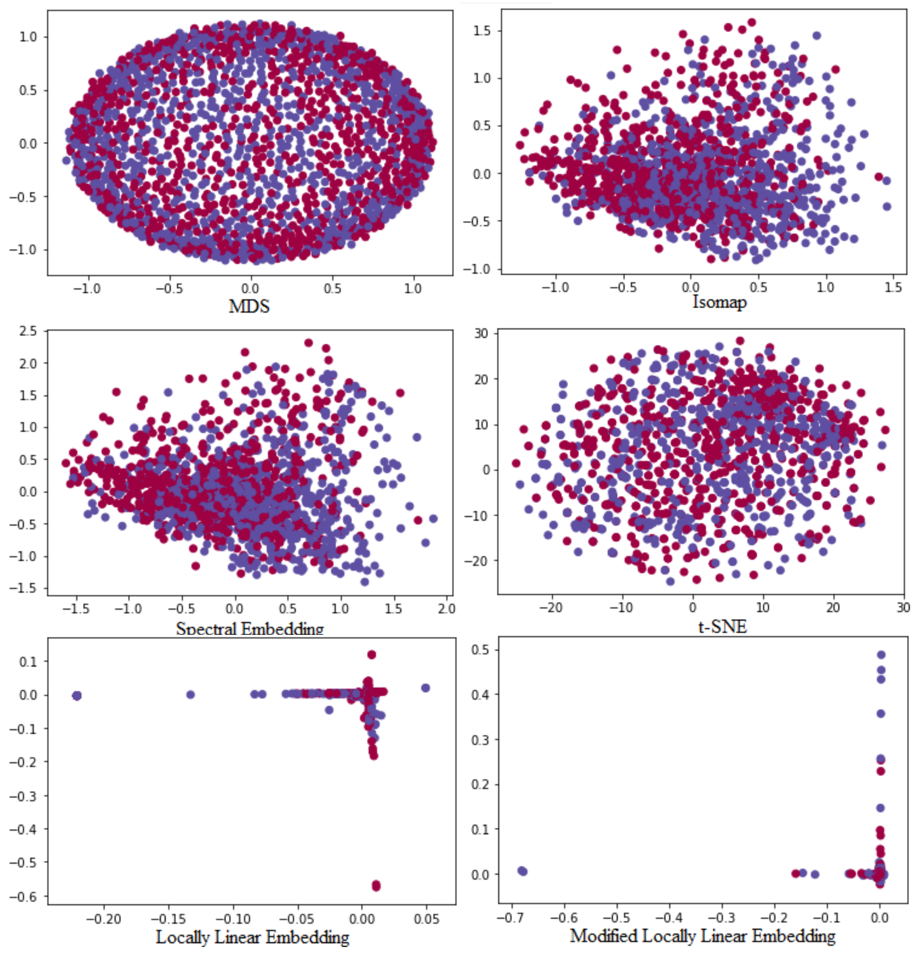


```
Feature ranking:
1. feature 998 (0.004455)
2. feature 8289 (0.003659)
3. feature 10626 (0.003255)
4. feature 5670 (0.002873)
5. feature 1314 (0.002774)
6. feature 978 (0.002628)
7. feature 9142 (0.002264)
8. feature 10570 (0.002219)
9. feature 6597 (0.002111)
10. feature 7397 (0.002105)
11. feature 9914 (0.001987)
12. feature 7842 (0.001790)
13. feature 6893 (0.001759)
14. feature 10180 (0.001735)
15. feature 1271 (0.001712)
16. feature 8368 (0.001703)
17. feature 8367 (0.001670)
18. feature 10179 (0.001618)
19. feature 5119 (0.001564)
20. feature 3824 (0.001538)
21. feature 10178 (0.001497)
22. feature 9910 (0.001442)
23. feature 6851 (0.001434)
```

There are many advance dimensionality reduction methods such as Principal Component Analysis or Manifold Learning that can be used for data visualization. Furthermore, whether a visualization method performs well or not depends on the data. If the data is linear, then linear projection method such as PCA works well. On the other hand, if the data is non-linear, methods like Manifold Learning is more desirable.

A manifold can be thought as the surface of an object, a 3-dimensional surface, and manifold learning is to flat the surface. Here we use an example to illustrate how it works. Given a real folded sheet, when manifold learning method flats it into a 2D plane, the learning method will unfold the sheet and let it lay on the plane flat instead of projecting the data in one direction directly.

Several manifold learning implementations are available with scikit-learn package, and we tried a few algorithms with our dataset to see if we can visualize our data in 2D plane. The 2D plot results are as below:



In all the result figures, there was no significant separation between the positive articles (denoted as blue dots) and negative articles (denoted as red dotes). Moreover, we've experimented several parameter combinations with each algorithm, yet none of a figure shows a clear separation between the two target groups. As a result, we concluded that our data has no significant characters for creating a meaningful 2D-plane visualization.

Applying Linear SVC and KNN:
After analyzing our data, we need to build our prediction model. We used Tfidf.fit to on our training dataset, and transform both training dataset and testing dataset. Then we could run the classifier algorithms. We applied Linear SVC and KNN.

The Linear SVC result is as follows:

Linear SVC	Precision	Recall	F1-Score	Support
Negative	0.90	0.87	0.89	254
Positive	0.88	0.89	0.89	246
Avg/total	0.89	0.89	0.89	500

Confusion matrix/Linear SVC		Predicted	
		Negative	Positive
True	Negative	223 (TN)	31(FP)
	Positive	26 (FN)	220 (TP)

We got good predictions on both positive and negative reviews. The average precision rate is 0.89, and F1 score is also 0.89. The positive predictive value is 0.88, and the negative predictive value is 0.90. The confusion matrix displayed the same result. Moreover, we applied KNN to compare different methods. In KNN, we used iterative loop trying to find the best K from 1 to 300. We use accuracy rate as our indicator. The k with the biggest accuracy rate is our target value. Accuracy rate is calculated by (TP+TN)/(P+N). When k grows larger than 100, the accuracy rate does not grow significantly. The best value of k is 228:

K=228

KNN	Precision	Recall	F1-Score	Support
Negative	0.79	0.85	0.82	254
Positive	0.83	0.76	0.79	246
Avg/total	0.81	0.81	0.81	500

Confusion matrix/KNN=228		Predicted	
		Negative	Positive
True	Negative	215 (TN)	39 (FP)
	Positive	58 (FN)	188 (TP)

As we increased k, we were trying to make the model simple. We have the best K equal to 228, and when K is small the accuracy rate is bad. This is reasonable, since our dimensions are very high, for each different review they will have very different pattern. As a result, we tend to find larger K so that the model won't be too flexible. However, when it's too big, the difference is not that obvious, so we don't need a very large K. We show a result when K=300.

K=300

KNN	Precision	Recall	F1-Score	Support
Negative	0.77	0.84	0.80	254
Positive	0.82	0.74	0.78	246
Avg/total	0.79	0.79	0.79	500

Confusion matrix/KNN=300		Predicted	
		Negative	Positive
True	Negative	213 (TN)	41 (FP)
	Positive	63 (FN)	183 (TP)

The precision rates only dropped slightly, overall, they are pretty much the same. We will stick to K=228 as our model.

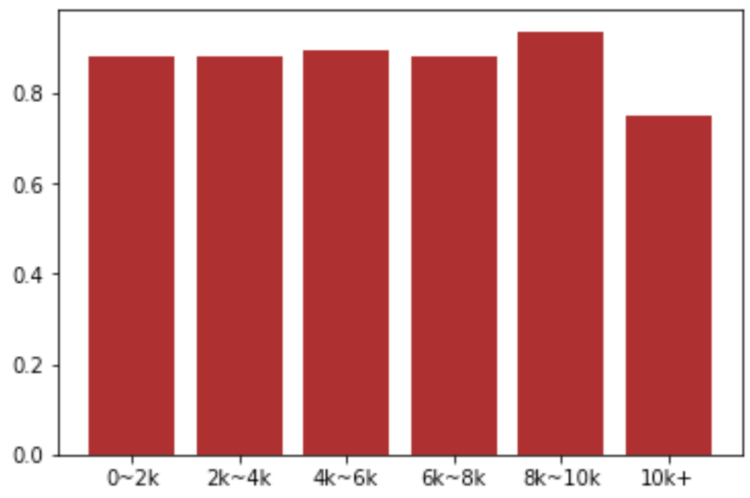
We can see that even we had optimized k, the precision rates are still worse than linear SVC. The average precision rate is 0.81. The positive predictive value is 0.83, and the negative predictive value is 0.76. That is, both model showed a better accuracy rate when the prediction is negative, however KNN has a much worse prediction rate on positive rather than negative, while they are close to each other in linear SVC.

After we had model, we want to analyze some cases that does not have the correct predictions. For example, we pick the article “cv303_27520”, and SVC prediction is negative, while the KNN gave positive which is the correct result. After reading the article, we found that the author spent most of the part describing the detail of the movie or the actors, and he tended to use metaphor in the sentence. For example, “keanu reeves (2000's " the watcher ") makes an utterly convincing backwoods meanie, so much so that it is difficult to believe he is an actor at all.”, this line should be a compliment, but the construction was very different from other reviews. In other word, the author is using some metaphor instead of using straight adjectives to describe the movie. When using KNN, since our K is very high, we can still find some similar neighbor that has the same content. As a result, the KNN still predicted the correct value. However, the Linear SVC cannot tolerant the metaphor features and tend to give wrong predictions.

Another example is “cv132_5618”, the actual value is positive, and Linear SVC gave the correct prediction, while KNN did not. One thing to be noticed is that this review is very short. Within a short review, many elements in the features vector is zero, which will make the review fall into a place without too many similar neighbors. Therefore, it may be hard for KNN to predict short articles. But for Linear SVC, it performs better, and maybe the reason is that it will not be affected by the length of review as KNN did. We will discuss more in the next part.

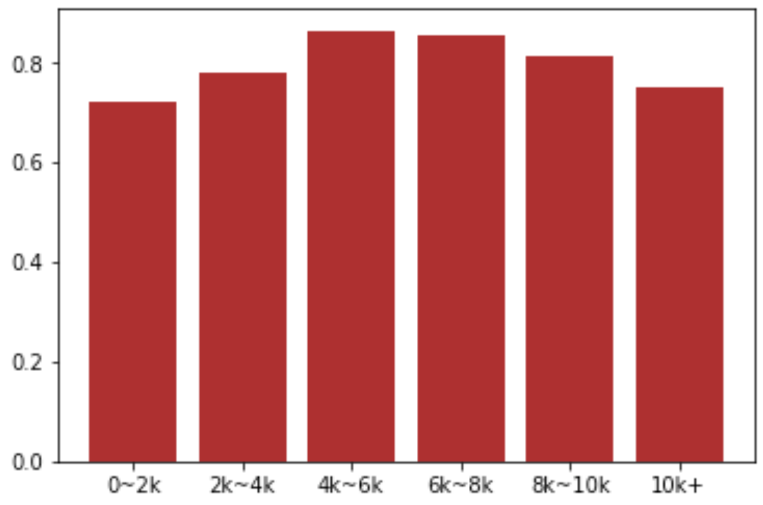
The Effect of Length of Reviews:
Will the longer reviews have better prediction rate? In order to get the result, we have to sort our testing dataset by the length of articles first. So, the dataset is different with the dataset we used in the previous page in order. After we sorted the testing dataset, we can apply Linear SVC and KNN to the dataset, and observe the output. If the prediction is correct, we set the value to be one; if not, then zero. After that, we categorized our dataset into 6 different sections, by using the length of reviews. They are: '0~2k','2k~4k','4k~6k','6k~8k','8k~10k','10k+'. They mean how many words are there in the articles.

From our conjecture, we though that for the longer length, we can have better prediction result. The reason is because the longer reviews contained plentiful of information, so the model can work well. Thus, we suppose to see a positive relationship. The result of Linear SVC is as below:



Different from our conjecture, the article with more than 10,000 words has the worst prediction rate, and it seemed stable when the lengths are short. We suggest that the negative effect of noise is greater than the useful information when the article is long. On the other hand, we don't see a significant drop when the length goes down. Generally speaking, the accuracy rates are quite stable over the length of reviews of Linear SVC. And, the longer does not assure better predictions.

The KNN result is as below:



In this picture, we noticed that same as Linear SVC, the length over 10k has lower accuracy rate, but the shorter sections performed worse. The differences between sections were still not very obvious, but we could say that when applying KNN, the shorter length of articles we have, the lower accuracy rate we get. However, it's not saying that we cannot predict short reviews using KNN, it's just the accuracy rate will be lower than longer reviews. As a summary, KNN performs worse than Linear SVC, and shorter reviews perform even worse in KNN models. And lengths did not make too much effect on Linear SVC, which is not the same as our conjecture.