Case Study 3

Textual Analysis of Movie Reviews

Team 9

Tingting Ma

Jiani Gao

Jinyan Lyu

Tianhao Guo

Mo Cheng

1. Motivation and Background

Getting feedback from customers and making improvement accordingly has always been one of the most important moves in the life cycle of a commercial product. And internet, without doubt, is the best place to collect feedbacks, where by posting text and uploading videos people share their opinions on everything. To collect and analyze these information, in 2017, machine learning is the go-to option. Since scikit-learn is one of the most efficient machine learning library in Python that features various classification, regression and clustering algorithms, in this case study, we explored several functions and parameters of scikit-learn and used it in the semantic analysis of movie reviews, so as to have a general view on textual analysis both theoretically and practically.

2. Sentiment Analysis on Movie Reviews

To analyze the data, we first loaded the dataset and split the data into 75% training data and 25% testing data. Then, we built a pipeline including Tfidf-vectorizer and Linear Support Vector Machine to filter and classify the training data's either positive or negative using different combination of parameters. After that, we used the grid search to fit the training data and predict the testing data. At last we obtained the scores for each parameter combination by this cross-validation process and evaluated the performance using a confusion matrix on a held out test set. As shown in figure 1, for a total number of 500 comments, 210 out of 256 negative and 213 out of 244 positive comments are correctly predicted.

```
n_samples: 2000
0 params - {'vect__ngram_range': (1, 1)}; mean - 0.83; std - 0.00
1 params - {'vect__ngram_range': (1, 2)}; mean - 0.86; std - 0.01
             precision    recall  f1-score   support

        neg       0.87      0.82      0.85       256
        pos       0.82      0.87      0.85       244

avg / total       0.85      0.85      0.85       500

[[210  46]
 [ 31 213]]
```

Figure 1: Optimal parameters selected by grid search and prediction results and confusion matrix of the prediction result.

3. Explore the scikit-learn TfidfVectorizer Class

The objective of this section is to explore three parameters of TfidfVectorizer: ***min_df***, ***max_df*** and ***ngram_range***. In order to figure out the meaning and correlation between the above parameters and features, below are several related terminologies and their definitions:

TF: Short for ***Term Frequency***, it presents how frequent a term appears in a document. Key point here is to understand "frequency". Documents might be varied in length, which makes it possible that the number of times a term appears in a long document to be significantly larger than a short document. To offset the effect of document length, term frequency are often divided by the measuring of document length (e.g. total number of terms in the document).

TF(t) = (# of times term t appears in document) / (Total number of terms in document)

IDF: Short for ***Inverse Document Frequency***, it measures the importance of an item. In the computation of term frequency, all terms are considered equal in weight, while in practise, certain terms such as "is", "an" and "the", which often have large numbers in appearance, share little weight in measuring the importance. In contrast, some terms that appear in small numbers of times might have strong meanings semantically, thus have heavy weights. To weigh down the frequent terms and scale up the rare ones, IDF does the computing as the following equation:

IDF(t) = log_e(Total # of documents / # of documents with term t in it).

TF-IDF: Short for ***Term Frequency–Inverse Document Frequency***, is a commonly used weighting technique in information retrieval and text mining. Being a statistical method, TF-IDF is used to assess the importance of a word in a fileset/file of corpus. The importance of a word increases proportionally to the number of times that a word appears in the document, while declines in inverse proportion to the frequency of appearance in the whole corpus. In practice, TF-IDF often used by search engines in scoring and ranking a document's relevance in user query. Knowing the calculation of TF and IDF, TF-IDF is calculated as follows:

tfidf(t, d, D) = tf(t,d) idf(t, D) = (# of times term t appears in a document d) / (Total # of terms in the documents) log_e(Total #of documents in the corpus / # of documents with term t in it)

min_df, max_df: When building the vocabulary, it will ignore the terms that have a document frequency strictly lower than the given *min_df* and those higher than *max_df*. Both parameters could be float, in range within [0.0, 1.0], representing a proportion of document; or integer, representing the absolute count number. Both parameter will be ignored if vocabulary is not None.

n-gram, ngram_range: *n-gram* is a terminology in computational linguistics and probability that means a contiguous sequence of n items from a given sequence of text or speech. Based on that, *ngram_range*, a tuple (min_n, max_n), represents the lower and upper boundary of the range of n-values for different n-grams to be extracted. The larger the upper boundary is, the larger the shape of vectorized result is and the more features we will get.

With terminologies clarified, we used control variable to explore the three TfidfVectorizer parameters. To be more specific, while changing one parameter, we kept the other two fixed. Thus we can observe and record the change of features accordingly.

Firstly, we performed a systematic screening with a changing *min_df* value from 0 to 99, while kept the value of *max_df* and *ngram_range* fixed. Figure 2 shows the plotting result. As we can see, the number of features decreases exponentially while min_df increases.

Secondly, we went through a similar screening process, applying *max_df* to range 0 to 1000 and fixed the value of *min_df* and *ngram_range*. The plotting result in figure 3 indicates suggests that the number of features increases exponentially while *max_df* increases.

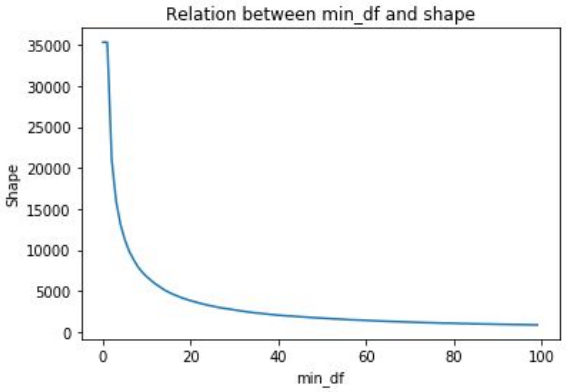

Figure 2: Screening of min_df values.

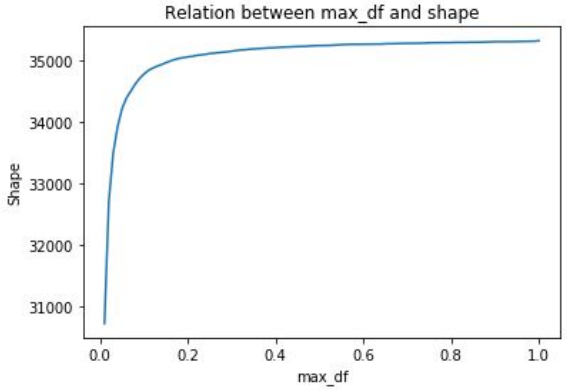

Figure 3: Screening of max_df values.

Both results in figure 2 and figure 3 are rational, because once *min_df*, which presents the minimum document frequency, increases, more features with document frequency lower than *min_df* are removed; whereas more features with document frequency no larger than *max_df* will remain in pool when *max_df* increases.

```
(1, 1): shape is 35443;
(2, 2): shape is 400307;
(3, 3): shape is 763139;
(4, 4): shape is 895933;
(5, 5): shape is 925941;
(1, 2): shape is 435750;
(2, 3): shape is 1163446;
(3, 4): shape is 1659072;
(4, 5): shape is 1821874;
(1, 3): shape is 1198889;
(2, 4): shape is 2059379;
(3, 5): shape is 2585013;
(1, 4): shape is 2094822;
(2, 5): shape is 2985320;
(1, 5): shape is 3020763;
```

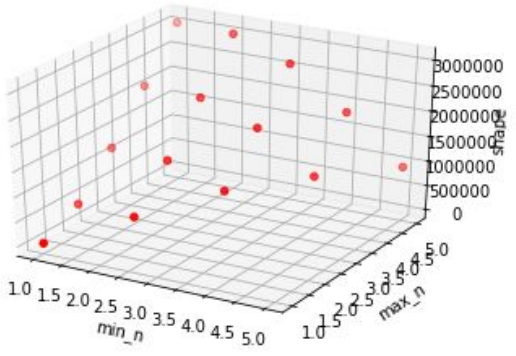

Figure 4: Exploration of ngram_range, datatable and 3D plotting.

When exploring the effect of ngram_range, a set of {(1,1), (2,2), (3,3), (4,4), (5,5), (1,2), (2,3), (3,4), (4,5), (1,3), (2,4), (3,5), (1,4), (2,5), (1,5)}tuple were assigned one by one while *min_df* and *max_df* were fixed. As figure 4 shows, more features are obtained when *ngram_range* (gap between the two numbers) increases. When we checked the results in detail, there's a pattern that, for example, the

number of features in range (1,2), which is 435,750 is the sum of range (1,1) 35,433 and range (2,2) 400,307, and this fits in the definition of ngram_range perfectly.

4. Machine Learning Algorithms

In this section, we examined LinearSVC and KNeighborsClassifier using the datasets we processed in problem 1 and tried different parameters to see whether the performance of one classifier or one set of parameters better.

The first step is to set up the parameters. For the two classifiers, we both choose the same set of parameters of TfidfVectorizer (Figure 4).

```
'vect__ngram_range': [(1,1),(2,2),(1,2),(1,3)],
'vect__max_df': [0.5, 0.75, 1.0],
'vect__min_df': [1,10,20,30],
```

Figure 4: Parameters of TfidfVectorizer

LinearSVC: Support Vector Machines (SVM) are supervised learning models used for classification and regression analysis of data. Here we use different parameters of LinearSVC to examine the performance of this classifier. We set the value of the penalty parameter C of the error term of LinearSVC as 1, 500 and 1000. After fitting the training data by grid search, we found the best score is 0.86 when the parameter sets are configured as in table 1. As the result shows, we can get the best score whatever penalty parameter C is 1, 500 or 1000, but all the values of min_df fixed at 10 and all the vect_ngram_range values fixed at (1, 3). Besides, in most cases, the value of vect_max_df is 0.75. So we speculate that the penalty parameter C has little effect on the result and when vect_min_df = 10, vect_max_df = 0.75, and vect_ngram_range = (1, 3) will give a better result comparing other parameter combinations.

penalty parameter C	vect__min_df	vect__max_df	vect__ngram_range
1	10	0.5	(1, 3)
1	10	0.75	(1, 3)
500	10	0.75	(1, 3)
1000	10	0.75	(1, 3)
1000	10	1.0	(1, 3)

Table 1

KNeighborsClassifier: K-nearest neighbors (K-NN) is a non-parameter method used for classification and regression. Here we use K-NN to classify our data. We set the n_neighbors parameter of K-NN as 1, 10, 20. And after fitting the training data by grid search, we found that the best score is 0.73 with the parameter sets configured as in table 2. According to the results, we can see that all the best results appeared when n_neighbors = 20 and the parameter combination n_neighbors = 20, min_df = 10, max_df = 0.5, ngram_range = (1, 3) tends to give a better result.

n_neighbors:	min_df	max_df	ngram_range
20	1	0.5	(1, 2)
20	1	0.5	(1, 3)
20	10	0.5	(1, 2)
20	10	0.5	(1, 3)
20	10	0.75	(1, 3)
20	30	0.75	(1, 3)

Table 2

Compare the prediction results of LinearSVC & K-NN

Next, we used the classifier to predict our testing data. Below are the results calculated. Clearly, the prediction result of LinearSVC is much better than that of K-NN.

	precision	recall	f1-score	support
neg	0.89	0.89	0.89	266
pos	0.88	0.88	0.88	234
avg / total	0.88	0.88	0.88	500

```
[[237  29]
 [ 29 205]]
```

	precision	recall	f1-score	support
neg	0.80	0.64	0.71	266
pos	0.67	0.82	0.73	234
avg / total	0.74	0.72	0.72	500

```
[[170  96]
 [ 43 191]]
```

Figure 5: Results of LinearSVC (left) and K-NN (right)

Random Forest Classifier: We did one more classifier algorithm to train the data. Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. We found the best score is 0.79 with the parameters: 'clf__n_estimators': 64, 'vect__max_df': 0.25, 'vect__min_df': 30, 'vect__ngram_range': (1, 3). Here's the confusion matrix with similar precision of K-NN.

```
                precision    recall  f1-score   support

           neg       0.72      0.88      0.79       251
           pos       0.84      0.66      0.74       249

avg / total       0.78      0.77      0.77       500

[[220  31]
 [ 84 165]]
```

Figure 6: Results of Random Forest Classifier

Two examples where the prediction was incorrect: We picked two wrong results predicted by K-NN. The first one is a review of movie *Topless Women Talk About Their Lives*. This is a negative review but predicted as a positive one. We found that although this is a negative review, the words used in this context are kind of vague. The reviewer thought the movie is not too bad so he/she didn't use too many negative words to describe. The review mainly described the content of the movie and also made some positive assessment of the good part. We thought that is the reason why the classifier was "confused" and made a mistake.

The second review is about the movie *The Jackal*, it is positive but predicted as a negative one. After reading this review, we found that this reviewer used too many negative words such as "not", "didn't" and "wasn't" when describing the content of the movie and also emphasized a lot that he/she wasn't a fan of this kind of movie before watching this one. Since the classifier could not tell the usage of these negative words, it made the mistake.

Conclusion: Based on our studies, we identified that the performance of LinearSVC is much better and provides a more accurate result. Probably it is because LinearSVC is more suitable for a large number of predictors compared to K-NN.

5. Open Ended Question: Finding the Right Plot

In this open-ended problem, we need to find a two-dimensional plot in which the positive and negative reviews are separated. So the key point is to transform thousands of features to only two features.

Since it's a little bit hard to find how to reduce the features, we decided to try two methods proposed by professor. First method is to plot the length of review versus the number of features of that review. Since we don't need to do the prediction, we use all 2000 data to train. We use TfidfVectorizer to fit and transform raw data. For each review, we get the length of review and get number of features using getnnz() of sparse matrix. In the figure below, yellow and purple points represent positive and negative reviews respectively. We noticed that these two colors are mixed together and all the points

look linear. This actually makes sense because with more words, there are more features after tfidf vectorizing. All reviews follow this pattern so we cannot separate positive and negative reviews.

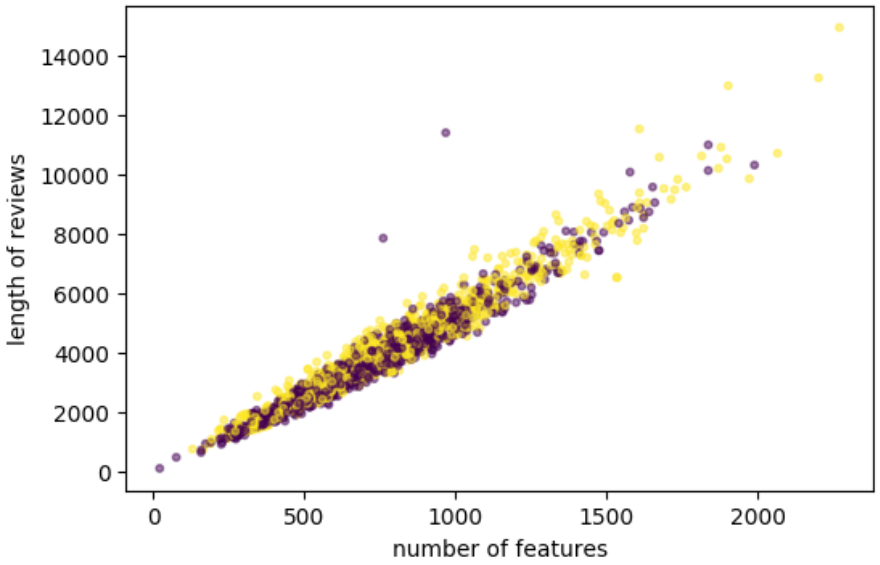

Figure 7: length of review versus the number of features

Next, we try to use Principal Component Analysis (PCA). This statistical procedure usually emphasizes variation and brings out strong patterns in a dataset. So we simply took all eigenvectors of the dataset's covariance matrix and reduced to two dimensions. However, since it's a sparse matrix, it cannot find two components with enough high variance. Most points in the following figure are mixed.

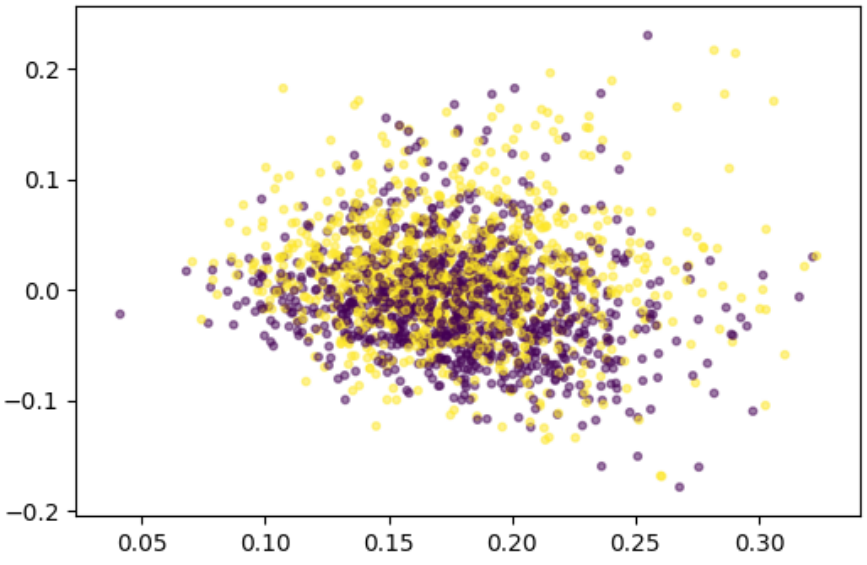

Figure 8: Principal Component Analysis (PCA)

To optimize the PCA method, we try to pick up a number of features using SelectKBest function provided in scikit-learn. We reduced number of features to 500 before using PCA, the final result is a little bit better. Some part of points are separated.

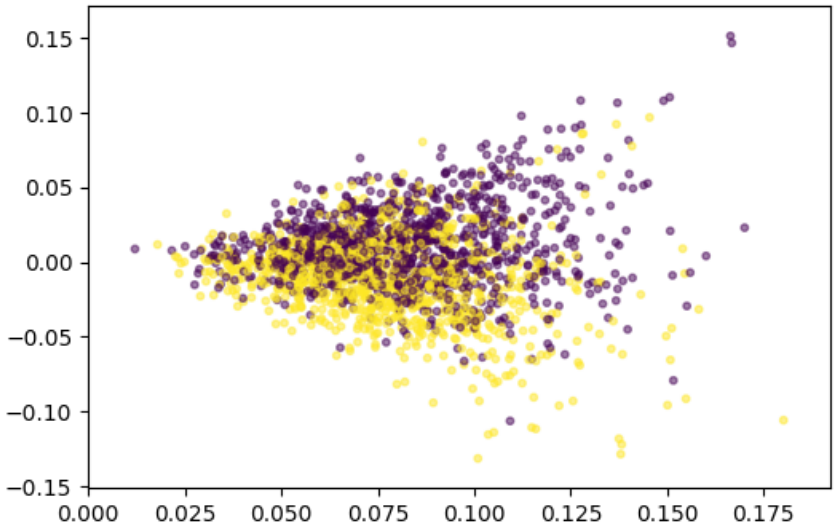

Figure 8: Optimized Principal Component Analysis (PCA)

Since we want to separate two kinds of points, we can use clustering algorithms. K-means is one of the most common algorithms which can cluster data to k centers. First we compute 2 centroids with all features. For each point, we compute the distance to 2 centroids and we get the following figure.

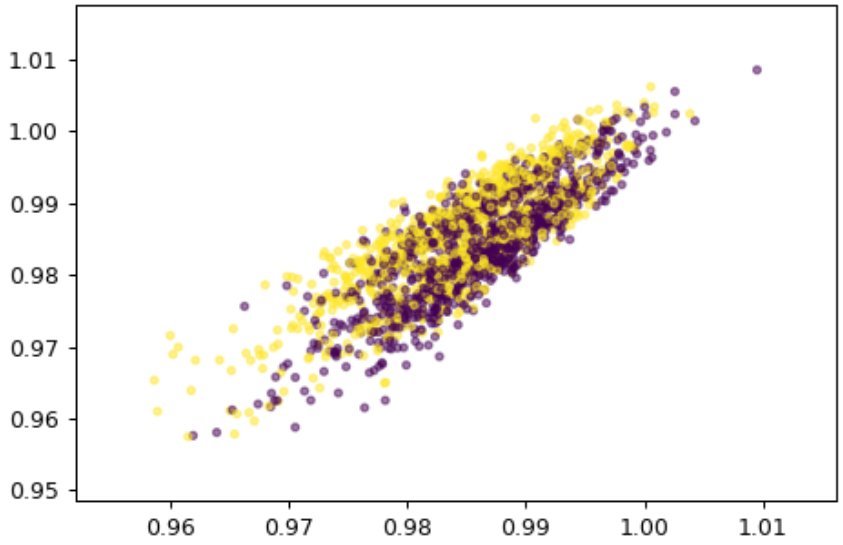

Figure 9: k-means

Then we optimized it using the same way we did on PCA. We pick up 20 features and compute the new centroids. We get a better result which is similar to optimized PCA.

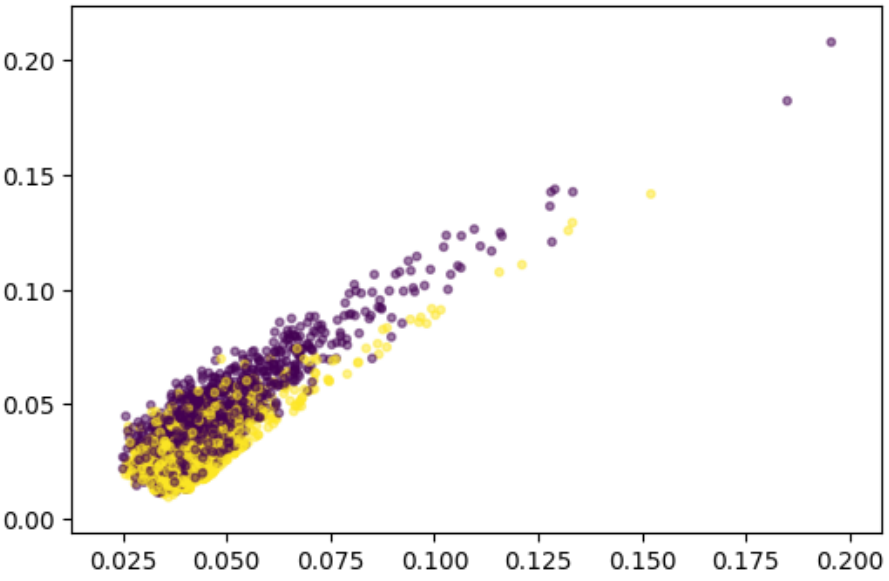

Figure 10: Optimized k-means

K-means is an unsupervised machine learning algorithm, but the data here is already labeled. So we tried to compute the centroids of the negative data and positive data without using k-means. Then we compute the distance again. This time, most points are separated and the bound of two parts is clear since we know all labels of data. This result is hard to achieve by unsupervised machine learning algorithm.

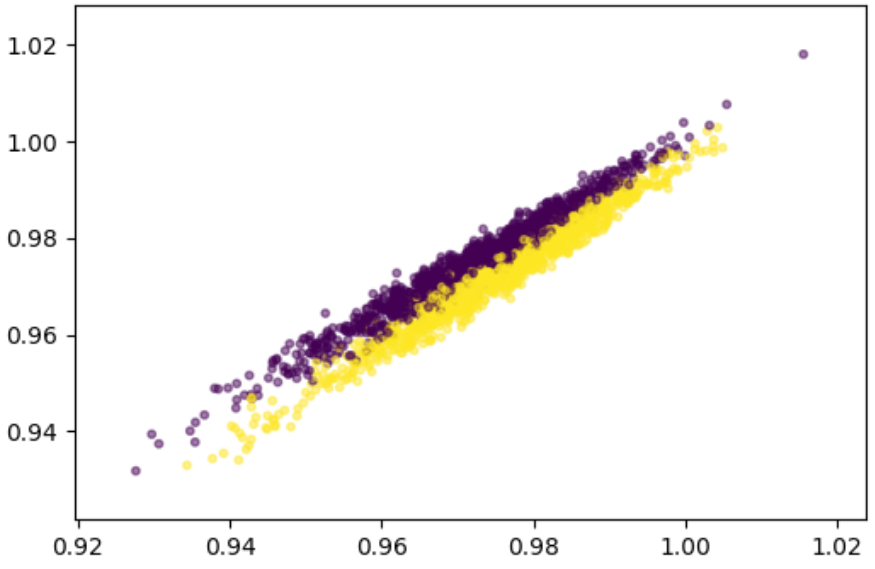

Figure 11: compute centroids without using k-means