CASE STUDY 3: TEXTUAL ANALYSIS OF MOVIE REVIEWS

TEAM 12

DS 501: INTRODUCTION TO DATA SCIENCE

TEAM MEMBERS

Chu Wang

Saranya Manoharan

Rishitha Kiran

Di You

1. INTRODUCTION

Watching movies is a very common pastime, as it is a form of relaxation and entertainment for majority of the people. Before many of us choose a movie, we often seek out the opinions of other who have already seen it. A movie review is the opinion of someone who has seen a particular movie and written an article about the movie. In this case study, we perform "Textual analysis of movie reviews" from the polarity dataset from http://www.cs.cornell.edu/people/pabo/movie-review-data. Textual analysis is a method to describe and analyze contents, structures and characteristics of text comments. It has been widely used to reveal static information such as documentation, standards, or user guides of legacy systems. In this case, we studied and built sentimental analysis and polarity model to analyze a movie review by fitting linear classifier on features extracted from the text of the user messages, in order to guess whether the opinion of the author is negative or positive.

2. SENTIMENT ANALYSIS ON MOVIE REVIEWS

This case study uses polarity dataset v2.0, which consists of 2000 .txt files, each containing a movie review, out of which 1000 positive and 1000 negative processed reviews are available. We followed standard sentimental analysis procedure from the tutorial "Working with Text Data" of scikit-learn [http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html].

First, we added a code that performs downloading of the dataset directly from the source website and stored in the local directory. Then, we randomly split the dataset into two groups, 75% for training (1500 records) and 25% for testing (500 records). Then, a pipeline was built containing Tfid-vectorizer, which filters out tokens that are either too rare or too frequent in the dataset, and a classifier, where the Linear Support Vector Machine was chosen with default for training data. In this pipeline, an additional parameter 'vect_ngram_range', was set to be in the range of (1,1) or (1,2), which basically means a sentence was tokenized to both one word or two words combinations, unigrams or bigrams, (more detailed explanation refer Problem 2). With the pipeline set, a GridSearchCV function followed by fitting the pipeline on the training set using grid search for the parameters to evaluate which parameter combination would afford best fittings. After cross-validation process, we thus obtained the mean cross-validation scores for each parameter set as explored by the grid search (Figure 1).

```
0 params - {'vect__ngram_range': (1, 1)}; mean - 0.83; std - 0.02
1 params - {'vect__ngram_range': (1, 2)}; mean - 0.84; std - 0.01
```

Figure 1: Optimal parameters selected by grid search

These mean scores indicate that, on the training data, the linear SVC pipeline performs more accurately when it considers both words and pair of words contained in our review. Using these preferred parameters determined by grid search, we use our SVC pipeline to predict the class of each review in our held-out testing set. The classification report is given below in Figure 2.

	precision	recall	f1-score	support
neg	0.86	0.89	0.87	247
pos	0.89	0.85	0.87	253
avg / total	0.87	0.87	0.87	500

Figure 2: Classification report

The high precision values on both classes (in addition to a nearly equal number of samples of each class) indicates that our model performed relatively well on this test set. We further evaluated the performance using the confusion matrix. The prediction result represented by a confusion matrix (Figure 3) indicates that 219 out of 247 negative comments and 216 out of 253 positive comments were predicted correctly. And the plotted confusion matric is shown in Figure 4.

```
[[219  28]
 [ 37 216]]
```

Figure 3: Confusion matrix of prediction result

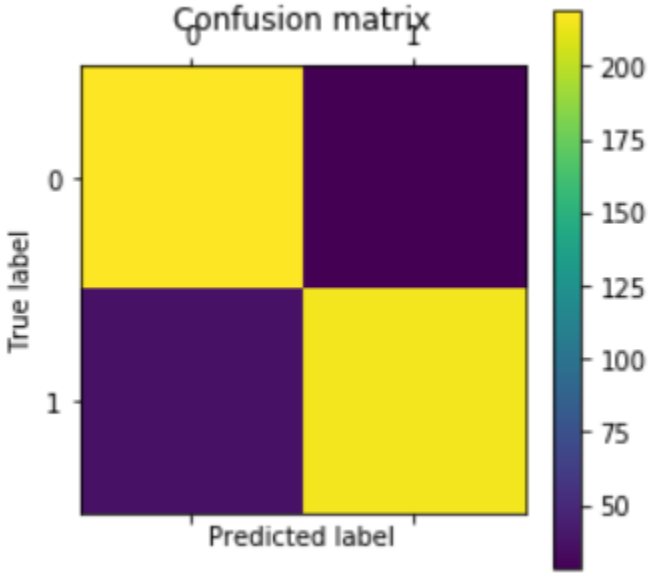

Figure 4: Confusion matrix

Confusion matrix is to evaluate the quality of the output of a classifier. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions. Combined with the classification report given above, this indicates that this model performed quite well on our test data set.

3. EXPLORE THE SCIKIT-LEARN TFIDVECTORIZER CLASS

In this section, our aim is to explore the effect of three parameters: min_df, max_df and ngram_range. First, we define few terms and then run TfidVectorizer on training data. Finally, we explore three parameters.

3.1. Define the term frequency–inverse document frequency (TF-IDF) statistic.

Term Frequency–Inverse Document Frequency (TF-IDF): The TFIDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus [3]. The td-idf weight is a weight often used in information retrieval and text mining. The importance increases proportionally to the number of times the word appears in the document but is offset by the frequency of the word in the corpus. The tf-idf is the product of two statistics, term frequency and inverse document frequency.

Term frequency (TF): It measures the number of times *t* a term occurs in a document *d*. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. While there are many ways to define this frequency, the simplest is the raw frequency: the number of times term *t* appears in the document *d* divided by the total number of terms *t* in document *d*.

Inverse Document Frequency (IDF): The inverse document frequency is a measure of how much information the word provides, that is, whether the term is common or rare across all documents. While computer term frequency, all terms are equally important. However, it is known that certain terms such as "stopwords", may appear in most of the documents, and thus not carry much information. More document-specific words are less likely to occur across a collection, and thus may be considered as carrying more "information" in that document. Thus, we need to weigh down the frequent terms while scale up the rare ones. For a term *t* and collection of documents *D*, the standard definition of inverse document frequency is the (logarithmically scaled) number of documents in *D* divided by the number of documents containing term *t*:

$$\log \frac{|D|}{|\{ d \in D : t \in d \}|}$$

Min_df: The vocabulary ignores terms that have a document frequency strictly lower than the given min_df value. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None [1].

Max_df: The vocabulary ignores terms that have a document frequency strictly higher than the given max_df value. If float, the parameter represents a proportion of documents, integer absolute counts. This parameter is ignored if vocabulary is not None.

Ngram_range: The lower and upper boundary of the range of n-values for different n-grams to be extracted. All values of *n* such that *min_n* <= *n* <= *max_n* will be used. When upper bound of ngram_range is larger, the shape of vectorized result is larger and we will get more features.

3.2. Run the TfidVectorizer class on the training data.

The TfidfVectorizer class in Python converts a collection of raw documents into a matrix of TF-IDF values for a set of features. Transforms text to feature vectors that can be used as inout for estimators. The rows of this matrix are indexed by the documents in the collection, and the columns correspond to a vocabulary of terms i.e. words or strings of words determined by the n_gram input to the class, contained in those documents. The entries of the matrix are the TF-IDF statistic for each term in each document.

3.3. Explore the min_df and max_df parameters of TfidVectorizer. How do they change the features you get?

In order to explore the result of three parameters is done by changing one parameter and fix the other two so that we are able to observe the change of features accordingly. Firstly, max_df and ngram_range were fixed and a systematic screening of min_df from 0 to 20 was performed. The plotting result shown in Figure 5 clearly indicates that the number of features decreases as the threshold increases.

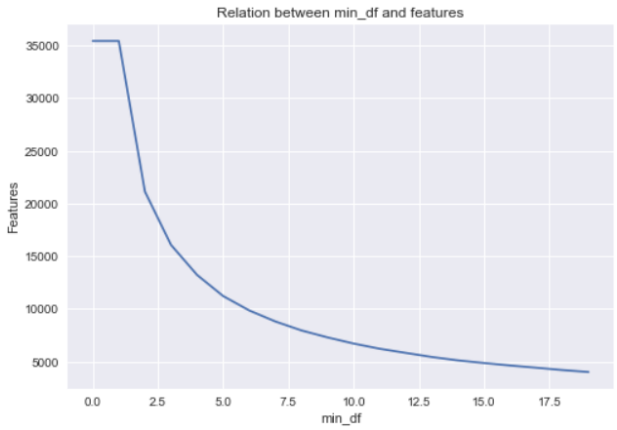

Figure 5: Screening of min_df values

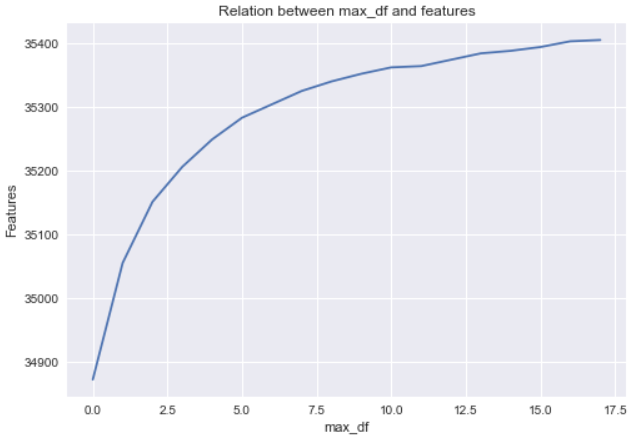

Figure 6: Screening of max_df values

The similar process from 0.1 to 1.0 was applied to max_df by fixing min_df and ngram_range accordingly. The plotting result shown in Figure 6 clearly indicates that the number of features increases exponentially after max_df decreases. The results thus obtained is rational because once min_df increases, more features with document frequency lower than min_df has been removed from features, whereas less features with document frequency higher than max_df will be removed when max_df increases.

3.4. Explore the ngram_range parameter of TfidVectorizer. How does it change the features you get?

Initially, we used a combination set of {(1,1), (2,2), (3,3), (4,4), (1,2), (2,3), (3,4), (1,3), (2,4), (1,4)} was searched one by one with fixed values of min_df and max_df. The results in Figure 7 show that more features are obtained as ngram_range is increased. This result clearly fits the definition of ngram_range, because, the feature number 435214 in range (1,2) is the sum of 35429 in range (1,1) and 399785 in range (2,2).

```
ngram_range - (1, 1); shape - 35429;
ngram_range - (2, 2); shape - 399785;
ngram_range - (3, 3); shape - 761536;
ngram_range - (4, 4); shape - 893589;
ngram_range - (1, 2); shape - 435214;
ngram_range - (2, 3); shape - 1161321;
ngram_range - (3, 4); shape - 1655125;
ngram_range - (1, 3); shape - 1196750;
ngram_range - (2, 4); shape - 2054910;
ngram_range - (1, 4); shape - 2090339;
ngram_range - (1, 5); shape - 3013843;
```

Figure 7: Exploration of parameter ngram_range

We also explored ngram_range using combination set of {(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7)}. A plot of the number of features versus ngram_range tuples of the form (1, ngram) are shown in Figure 8. As expected, it is clear that the number of features in the TdifVectorizer vocabulary increases as ngram is increased in ngram_range tuples of the form (1, ngram). Moreover, this growth appears to be roughly linear.

```
ngram_range - (1, 1); features - 35429;
ngram_range - (1, 2); features - 435214;
ngram_range - (1, 3); features - 1196750;
ngram_range - (1, 4); features - 2090339;
ngram_range - (1, 5); features - 3013843;
ngram_range - (1, 6); features - 3942754;
ngram_range - (1, 7); features - 4871884;
```

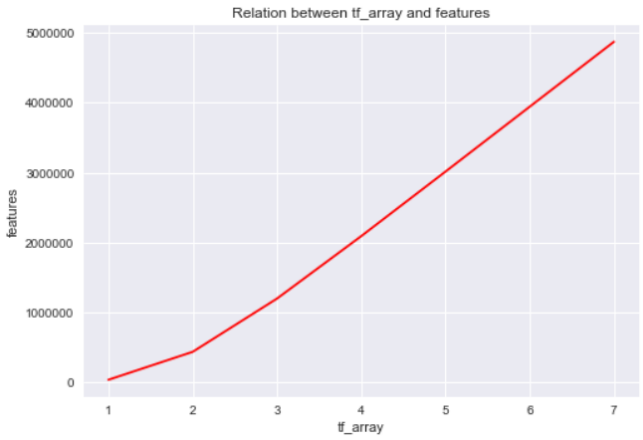

Figure 8: Exploration of paramter ngram_range of form (1, ngram)

4. MACHINE LEARNING ALGORITHMS

4.1. Pick parameters for TfidfVetorizer

In order to pick the parameters for TfidfVectorizer, we ran cross-validations using grid search on different values of min_df, max_df, and ngrams.

Min_df

First, we ran cross-validations on min_df with the range of 20, other variables being equal. The default value of is 1, which means "ignore terms that appear in less than 1 document", that is not ignore any terms. According to grid search, the min_df value with the highest precision is 5. Also, the plot of relationship between min_df and precision of prediction shows (Figure 9) that in

general, higher min_df value tends to result in lower precision scores. Thus, it is reasonable to conclude that setting a relatively higher infrequency threshold would probably impair the overall prediction performance.

```
best params {'vect__min_df': 5}
             precision    recall  f1-score   support

        neg       0.82      0.82      0.82       184
        pos       0.84      0.85      0.85       216

avg / total       0.83      0.83      0.83       400
```

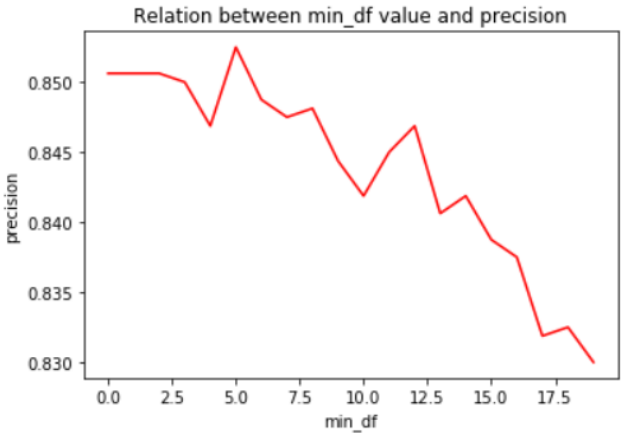

Figure 9: Exploration to find best Min_df value

Max_df

Keeping other two parameters the same and we ran cross validation on max_df, we set its type of value as float in this experiment. The plot below (Figure 10) indicates that the precision remains at a high level when max_df between 0.4 and 0.9. The idea of default value of max_df is to not ignore any terms at all. However, according to the result we obtained, it seems that including the words that appear in almost all document is really meaningless and irrelevant, as features will make the model's performance worse. The best max_df value chosen by grid search is 0.55

```
best params {'vect__max_df': 0.55000000000000016}
             precision    recall  f1-score   support

        neg       0.89      0.82      0.85       209
        pos       0.82      0.88      0.85       191

avg / total       0.85      0.85      0.85       400
```

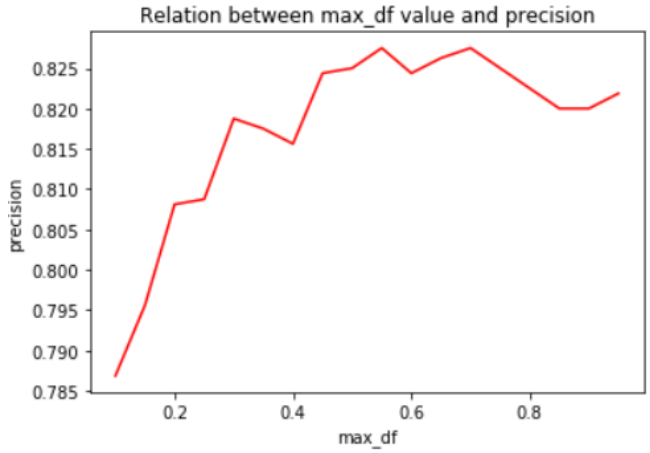

Figure 10: Exploration to find best Max_df value

N_grams value

For ngram, we experimented 11 different range of ngram values. We expected to capture the language structure from the statistical point of view, such as some words is likely to follow the given one. The grid search result shows below. Surprisingly, increasing the range of ngram, though

adding more features to the model, does not lead to a higher precision. According to the plot (Figure 11), ngram range of (1, 2) even has a higher precision then (1, 5)

```
best params {'vect__ngram_range': (1, 2)}
             precision    recall  f1-score   support

        neg       0.92      0.87      0.89       217
        pos       0.86      0.91      0.88       183

avg / total       0.89      0.89      0.89       400
```

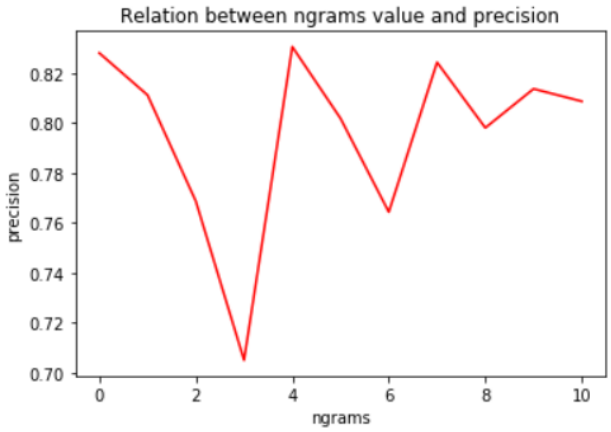

Figure 10: Exploration to find best N_gram value

Interaction Effect

We also hypothesize that those parameters would interact with each other. Due to the constraint of hardware, we only run experiment on limited combination of parameters max_df and min_df. It turns out we got different result of parameters and it do improve the precision of classification by 3% on average, comparing to use best value of parameters respectively.

```
best params {'vect__max_df': 0.67999999999999972, 'vect__min_df': 3}
             precision    recall  f1-score   support

        neg       0.84      0.86      0.85       194
        pos       0.87      0.84      0.86       206

avg / total       0.85      0.85      0.85       400
```

Figure 11: Exploration to find best combination of parameters

4.2. Pick parameters for Classifiers

LinearSVC

We run experiment on parameter C, which is the penalty parameter of the error term, with parameters of TfidfVectorize fixed as follow: min_df = 5, max_df = .55, ngram_range(1,2). According to the result of 3 values we picked, which is 0.1,0.5 and 1.0, the precision tends to be higher when C is larger. The precision of LinearSVC classifier reaches 0.88 when the value of C equals to 1.0.

```
0 params - {'clf__C': 0.1}; mean - 0.82; std - 0.01
1 params - {'clf__C': 0.5}; mean - 0.84; std - 0.01
2 params - {'clf__C': 1.0}; mean - 0.85; std - 0.01
best params {'clf__C': 1.0}
             precision    recall  f1-score   support

        neg       0.84      0.84      0.84       184
        pos       0.86      0.87      0.86       216

avg / total       0.85      0.85      0.85       400
```

Figure 12: Exploration to find best parameter for LinearSVC (other parameter fixed)

KNeighborsClassifier

Fixing the parameters of TfidfVectorizer as follow: min_df = 5, max_df = .55, ngram_range(1,2), we run experiment on 13 different K values. The generalization ability of KNN classifier increasing as the value of K increase. However, its stability begins to drop after K exceeds 250, and the precision also begins to drop after K over 350.

```
0 params - {'clf__n_neighbors': 3}; mean - 0.65; std - 0.01
1 params - {'clf__n_neighbors': 5}; mean - 0.67; std - 0.01
2 params - {'clf__n_neighbors': 10}; mean - 0.69; std - 0.00
3 params - {'clf__n_neighbors': 20}; mean - 0.71; std - 0.00
4 params - {'clf__n_neighbors': 50}; mean - 0.76; std - 0.01
5 params - {'clf__n_neighbors': 80}; mean - 0.78; std - 0.01
6 params - {'clf__n_neighbors': 100}; mean - 0.79; std - 0.02
7 params - {'clf__n_neighbors': 150}; mean - 0.80; std - 0.02
8 params - {'clf__n_neighbors': 200}; mean - 0.80; std - 0.02
9 params - {'clf__n_neighbors': 250}; mean - 0.80; std - 0.01
10 params - {'clf__n_neighbors': 300}; mean - 0.80; std - 0.03
11 params - {'clf__n_neighbors': 350}; mean - 0.80; std - 0.02
12 params - {'clf__n_neighbors': 400}; mean - 0.79; std - 0.03
best params: {'clf__n_neighbors': 250}
             precision    recall  f1-score   support

        neg       0.79      0.82      0.80       194
        pos       0.82      0.79      0.81       206

avg / total       0.81      0.81      0.81       400
```

Figure 13: Exploration to find best parameter for KNN (other parameter fixed)

Interaction with parameters

Also, we hypothesize that the parameter of classifier would probably have interaction relationship with parameters in TfidfVectorizer. In this case, we tried several combinations of all kinds of parameters and get the best result as below, which is a little different from former result:

```
best params {'clf__C': 0.5, 'vect__max_df': 0.59999999999999998, 'vect__min_df': 9, 'vect__ngram_range': (1, 2)}
             precision    recall  f1-score   support

        neg       0.90      0.85      0.87       203
        pos       0.86      0.90      0.88       197

avg / total       0.88      0.88      0.87       400
 best params: {'clf__n_neighbors': 250, 'vect__max_df': 0.5, 'vect__min_df': 9, 'vect__ngram_range': (1, 2)}
              precision    recall  f1-score   support

         neg       0.80      0.81      0.81       204
         pos       0.80      0.80      0.80       196

 avg / total       0.80      0.80      0.80       400
```

Figure 14: Exploration to find best parameter for KNN (combining with other parameters)

4.3. Classification results

Based on the previous cross-validation result, a set of parameters and classifier is chosen as follows: min_df = 9, max_df = 0.60, ngram_range(1,2), linearSVC(C=0.5). The combination of these parameters shows great stability and relatively high performance within the results of our more than 1000 experiments. The test result of when using these parameters is as below:

	precision	recall	f1-score	support
neg	0.90	0.85	0.87	203
pos	0.86	0.90	0.88	197
avg / total	0.88	0.88	0.87	400

Figure 15: Confusion Matrix of chosen Classifier

In general, LinearSVC classifier outperforms KNN classifier. Moreover, precision of LinearSVC classifier is stable and keeps at a high level, which is not sensitive to change of C; while on the other hand, precision of KNN classifier changes in a wide range with the change of N.

4.4. Reflection of Incorrect Classification Instance

This model has predicted 67 wrong answers among 400 tests. We first looked at a case when the model has predicted it to be positive, yet it is really negative.

```
number of wrong answers:  67
predicted anwser:  1
answer:  0
```

Figure 16: Prediction results

terrence malick made an excellent 90 minute film adaptation of james jones' world war ii novel .
unfortunately , he buried it within an overlong and overreaching 3-hour long pseudo-epic .
*this is a **shame** because the film features an outstanding performance by nick nolte .*
the best scene is when nick nolte's character , lt . col . tall , is forced to deal with the direct refusal by capt . staros (elias koteas) to execute an order .
nolte's reaction and transformation may be the best work of his career .
had terrence malick concentrated on the great performances of nolte and koteas as well as those by sean penn , woody harrelson , and john cusack , he could have made a truly great film .
instead , malick saddled the film with plodding pacing , unnecessary flashbacks , and a voice-over narration all designed to telegraph the great philosophical underpinnings of the story .
*the narration was especially **annoying** as much of it sounded like very **bad** high school poetry .*
with a lot of editing , the core story could be transformed into a truly classic war film.

hopefully , the dvd version of this film will feature options to suppress the narration, and perhaps will even provide for an alternate , shorter version of the film .
i give this film .

From this result we could understand that the model got this one wrong. The author clearly expressed his disappointment towards the movie, using words such as "shame", "annoying", "bad" etc. But we noticed that the review praised the performance of Nick. So, although negative over all, the review has some positive sentiments which makes it hard to categorize.

Here is another example that is positive but predicted to be negative:

i had been expecting more of this movie than the less than thrilling twister *.*
twister was good but had no real plot and no one to simpithize with .
but twister had amazing effects and i was hoping so would volcano
volcano starts with tommy lee jones at emo .
he worrys about a small earthquake enough to leave his daughter at home with a baby sitter .
there is one small quake then another quake .
then a geologist points out to tommy that its takes a geologic event to heat millions of gallons of water in 12 hours .
a few hours later large amount of ash start to fall .
then . . . . it
starts .
the volcanic eruption . . . .
i liked this movie . . . but
it was not as great as i hoped *.*
i was still good none the less .
it had excellent special effects .
the best view . . . the
helecopters flying over the streets of volcanos .
also . . . there were interesting side stories that made the plot more interesting .
so . . . it was good ! !

Throughout this review, the author criticizes the movie as it did not meet his expectations saying "i had been expecting more of this movie than the less than thrilling twister", "it was not as great as i hoped." However, the author still thinks it is a good movie after all. The harsh criticism might have confused the model, especially it is using keywords or cluster of keywords to understand sentiment. In this case, negative keywords greatly outnumber the positive keywords which makes it extremely hard to analyze the sentiment using statistical approach.

5. OPEN ENDED QUESTION: FINDING THE RIGHT PLOT

5.1. Principle Components Analysis (PCA)

For this question we attempt to find a 2-dimensional plot where the positive and negative reviews can be separated. The first attempt involves using Principle Components Analysis (PCA),

a method of identifying the vectors describing the greatest variance within a dataset. This vector space can be used as a means of classifying other vectors in the original space according to their similarity. We accomplished this by simply taking all the eigenvectors of a dataset's covariance matrix and reducing to two-dimensions. Since it is a sparse matrix, none of features can cover a large percentage of features, thus the percentage of variance explained by each of the selected components are not high enough for separating (explained variance ratio of first two components. A scatter plot in Figure 17 shows the result of PCA.

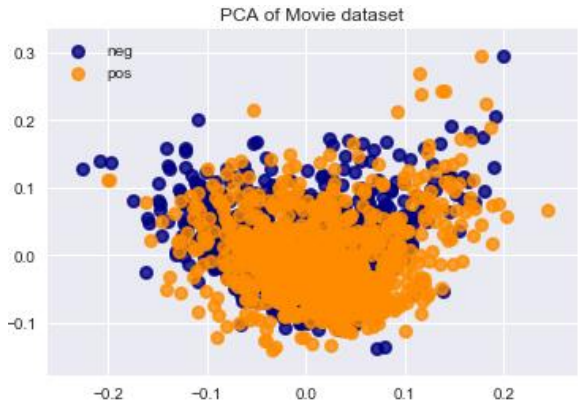

Figure 17: Plotting PCA after using all features in data

5.2. SelectKBest

We then picked up a number of features chosen by SelectKBest function in Scikit-learn before doing PCA. A series of parameters for SelectKBest was tested, and 600 features was eventually used to produce a much better separating result. The result is shown below Figure 18.

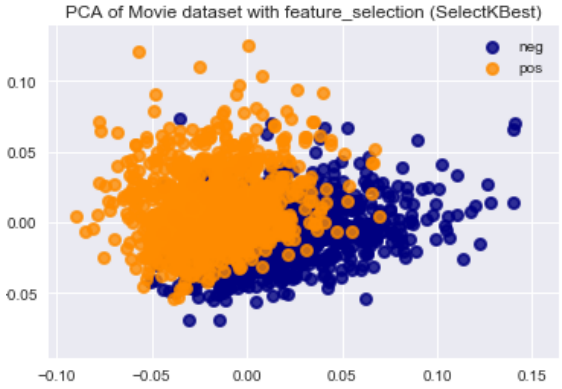

Figure 18: Plotting PCA using 600 features selected by SelectKBest function

5.3. TruncatedSVD

We then tried applying TruncatedSVD with its default linear kernel by reducing the number of components (features) to obtain the plot (Figure 19). From the plot we see that TruncatedSVD also did not perform well on our data.

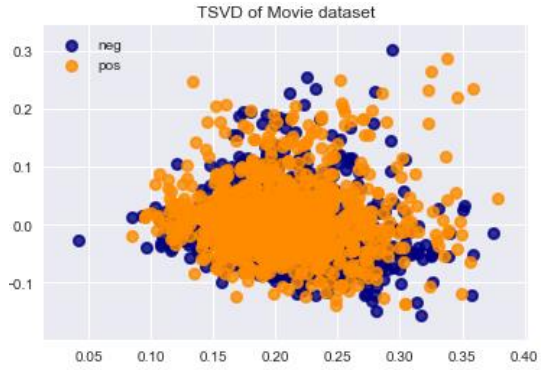

Figure 19: Using Truncated SVC

5.4. KernelPCA and Incremental PCA

As TruncatedSVD did not perform well on our data. So, we tried to apply KernelPCA, where we applied three values for kernel parameter and analyzed the results. The results are shown below "KernelPCA with default value" (Figure 20a), "KernelPCA with cosine value" (Figure 20b), "KernelPCA with sigmoid value" (Figure 20c), "KernelPCA with poly value" (Figure 20d) and Incremental PCA (Figure 21) to the data set, but the results obtained were similar to the above one and thus not able to separate the positive and negative reviews.

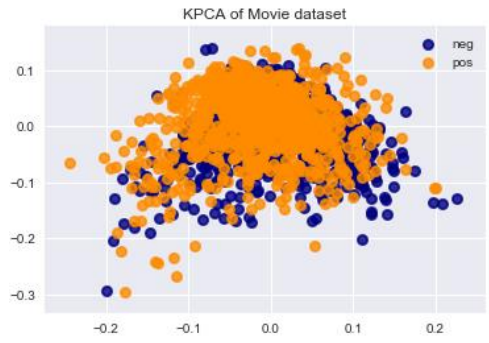

Figure 20a: Using KernelPCA

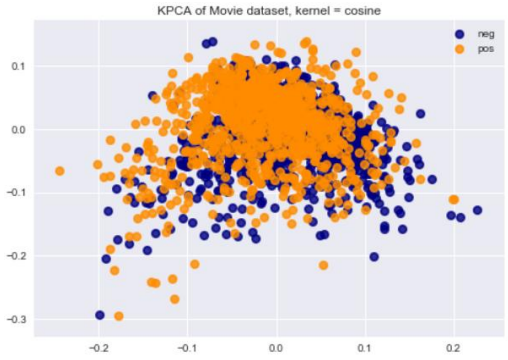

Figure 20b: Using KernelPCA (cosine)

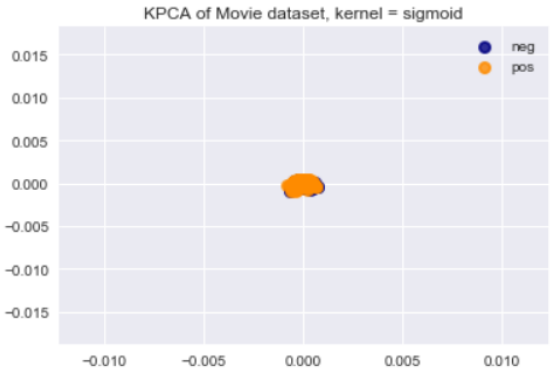

Figure 20c: Using KernelPCA (sigmoid)

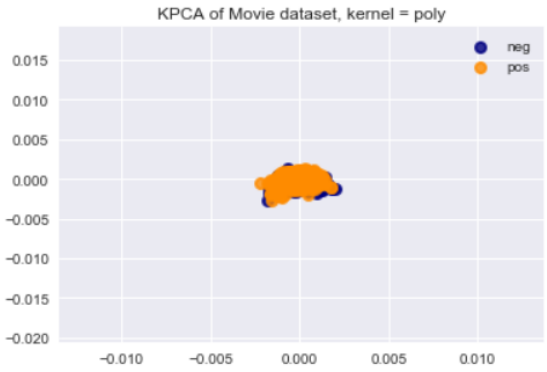

Figure 20d: Using KernelPCA (poly)

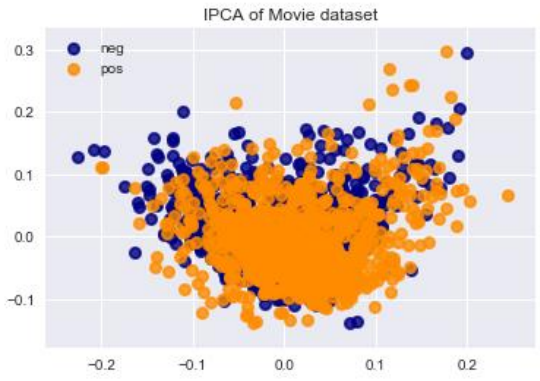

Figure 21: Using Incremental PCA

5.5. Plotting of the number of word "good" and “bad” vs length of the review

In an attempt to separate the negative and positive reviews in a two-dimensional plot, we counted the number occurrences of the words “bad” and “good” that appear in all of the reviews and then plotted three graphs: the number of “bad” versus length of the reviews, the number of “good” versus the number of “bad”, and the number of “good” versus the length of the reviews.

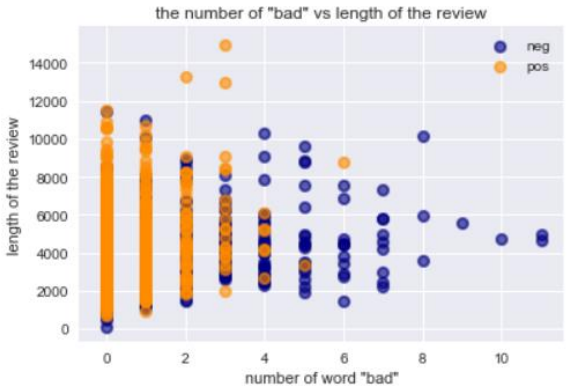

Figure 21a: Number of bad vs length

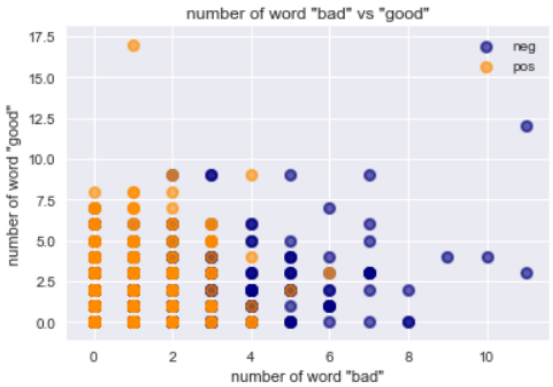

Figure 21b: Number of word bad vs good

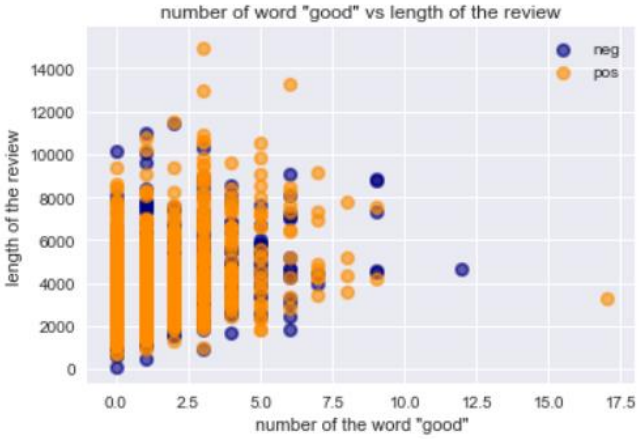

Figure 21c: Number of word good vs length

5.6. Manifold Learning methods

Few methods in Manifold learning were implemented and the results were analyzed. Below plots shows the results of different methods.

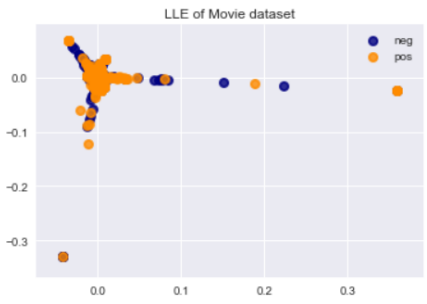

Figure 22a: Locally Linear Embedding

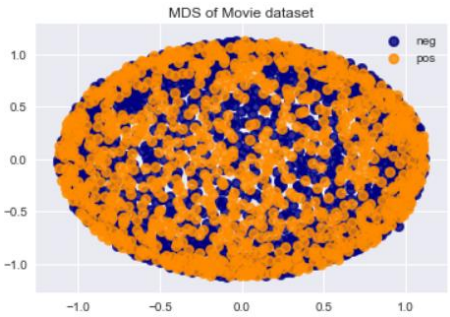

Figure 22b: MDS

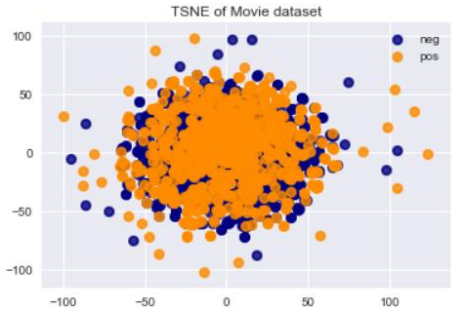

Figure 22c: TSNE

Figure 22d: Isomap

Figure 22e: GaussianRandom Projection

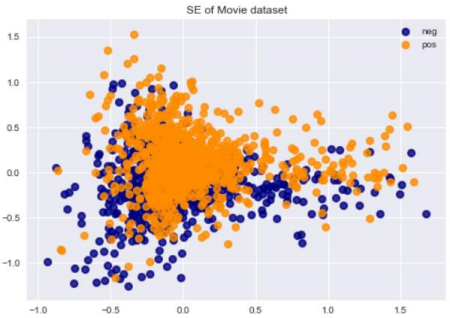

Figure 22f: Spectral Embedding

Figure 20g: SparseRandomProjection

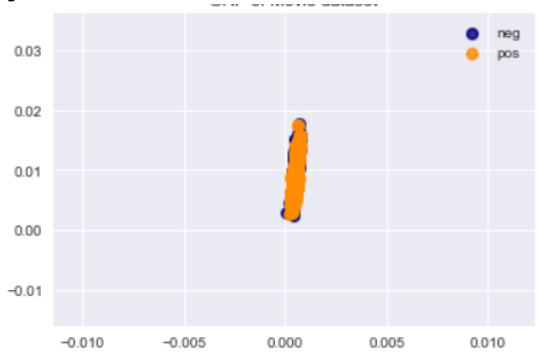

Figure 20h: FeatureAgglomeration

REFERENCES

[1] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[2] http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

[3] https://en.wikipedia.org/wiki/Tf%E2%80%93idf