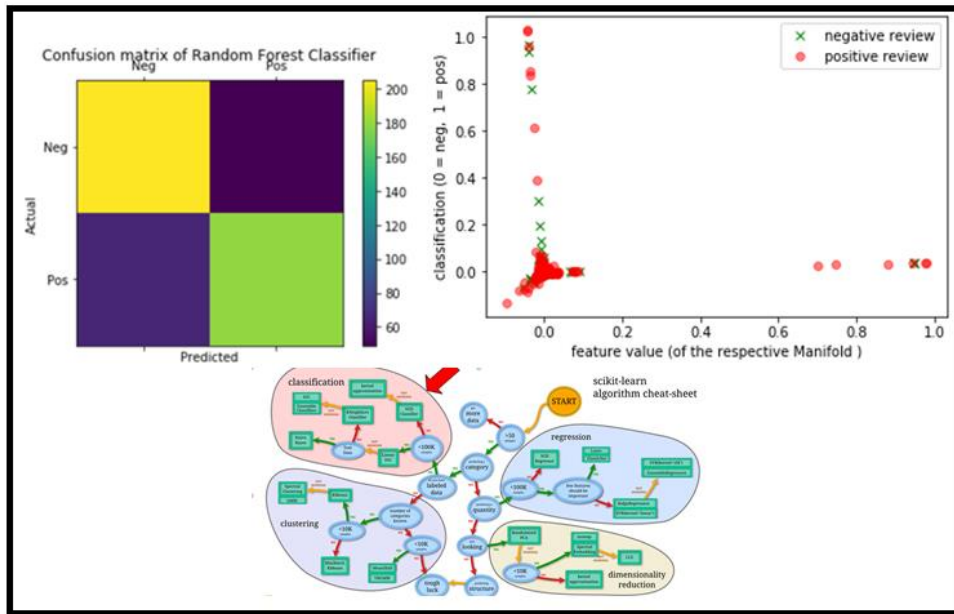


Analysis of v2.0 polarity dataset consisting of 2,000 textual movie reviews



Submitted By:

Todd Hay

Mukund Khandelwal

Zeling Lei

Brandon Werner

Zhe Lyu

Submitted To:

Prof. Randy Paffenroth

Instructor: DS501

WPI

11/16/2017

Contents

| | |
|---|----|
| Abstract | 3 |
| Motivation..... | 3 |
| Goal..... | 3 |
| Introduction..... | 3 |
| Problem 1: Sentiment Analysis on movie reviews | 4 |
| Problem 2: Explore the Scikit-learn TfidfVectorizer class | 6 |
| Problem 3: Machine learning algorithms..... | 9 |
| Problem 4: Finding the right plot..... | 15 |
| Business Application | 19 |
| Conclusion | 20 |
| Scope of Improvement..... | 20 |
| Citations | 21 |

Analysis of v2.0 polarity dataset consisting of 2,000 textual movie reviews

Todd Hay, Mukund Khandelwal, Zeling Lei, Brandon Werner, Zhe Lyu

Abstract

Our team (Team 14) proposes that useful information can be extracted from a dataset of 2,000 textual movie reviews. We first performed exploratory data analysis (EDA) to better understand the primary characteristics of the movie review data set. Guided by the information collected through EDA, we made various conjectures on the interactions and relationships between movie reviewers and their assessments of movie quality to identify areas of risk and opportunity.

Next, we performed sentiment analysis using Scikit-learn. Then we ran the TfidfVectorizer class on the training data and explored the min_df, max_df and ngram_range parameters of TfidfVectorizer. In problem 3 we fit our TfidfVectorizer using docs_train. Then we computed Xtrain and Xtest and we examined two classifiers provided by scikit-learn, LinearSVC and KNeighborsClassifier with some bonus explorations of classifiers Multinomial Naïve Bayes, Random Forest and LinearSVC using Polynomial kernel. Finally, in problem 4 we experimented with manifold learning to see if we could separate the reviews using a two-dimensional graph without feature engineering beyond the TfidfVectorizer or using another library.

Motivation

Our team believes there is actionable information hidden in the dataset that can be analyzed by a model. It is our desire to unlock the value through the application of statistical rigor to create a machine learning model that can analyze textual movie reviews.

Goal

Our team's goal is to collect the data, analyze the reviews using Scikit Learn, identify best classifier and make effort to visualize positive and negative reviews.

Introduction

As a starting point our team will convey some basic details regarding the v2.0 polarity dataset from <http://www.cs.cornell.edu/people/pabo/movie-review-data/>. This dataset holds two subfolders which contain .txt format written reviews of movies divided into positive and negative reviews by two-thousand movie reviewers. Specifically, we analyzed the polarity dataset v2.0, when combined, collectively contained “1000 positive and 1000 negative processed reviews” [1.1].

This dataset contains significant insights how movie goers analyze and describe the movies they watch. Team 14 will explore these areas and share our findings in this paper. As a first step, we will do some exploratory data analysis and present some basic details of the data our team collected and analyzed.

Problem 1: Sentiment Analysis on movie reviews

The perfunctory step was to import and merge the two files that make up the two-thousand textual movie reviews dataset.

First we installed the Scikit-learn which is a “machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.” [1.2]

Next, we downloaded the movie review dataset using the script provided in the case study, we split the dataset into training and test sets. We assigned 75% of the reviews to the training dataset and we assigned the remaining 25% of reviews to the testing dataset.

Next, we implemented an example vectorizer / classifier pipeline that filtered out words that appear too frequently or too infrequently. The vectorizer used was TfidfVectorizer and the classifier used was LinearSVC. Using the vectorizer, we set the parameters to try the ngram_ranges of (1, 1) and (1, 2) with the min_df parameter explicitly set to 3 and the max_df parameter explicitly set to 0.95. We calculated the mean and standard deviation for each grid search candidate along with the parameter settings for all the candidates explored by grid search.

| Metrics for example LinearSVC Classification | | | | |
|--|-----------|--------|----------|---------|
| 0 params - {'vect_ngram_range': (1, 1)}; mean - 0.82; std - 0.01 | | | | |
| 1 params - {'vect_ngram_range': (1, 2)}; mean - 0.84; std - 0.01 | | | | |
| | precision | recall | f1-score | support |
| neg | 0.91 | 0.88 | 0.90 | 254 |
| pos | 0.88 | 0.91 | 0.90 | 246 |
| avg / total | 0.90 | 0.90 | 0.90 | 500 |

We found the best score produced by the grid search was:

```
# find best score from grid search for LinearSVC
grid_search.best_score_
```

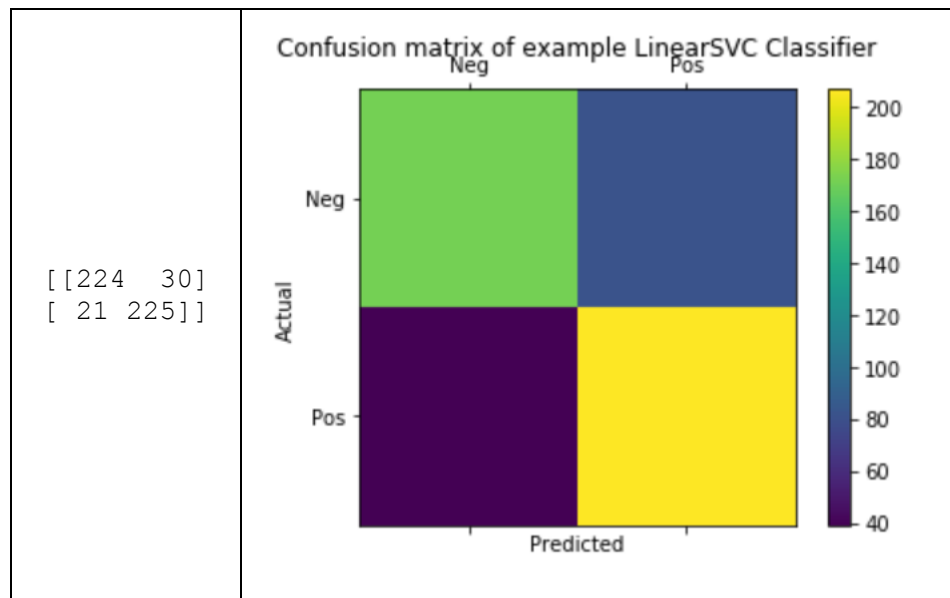
```
0.8386666666666667
```

The parameters used to produce the best score were:

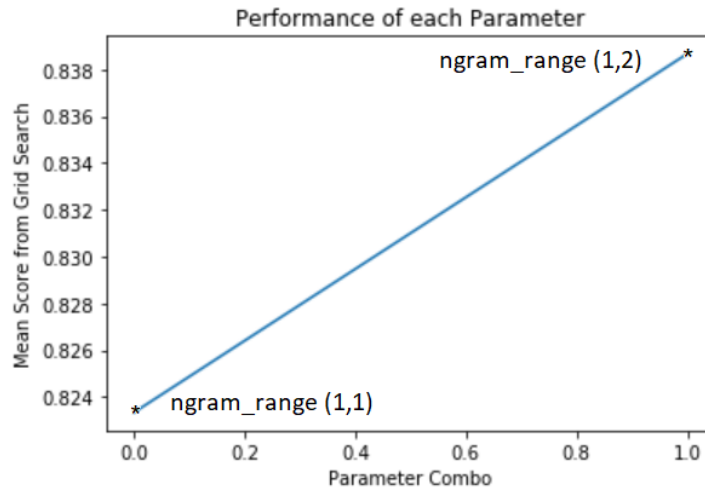
```
# find best parameters from grid search for LinearSVC
grid_search.best_params_
```

```
{'vect__ngram_range': (1, 2)}
```

We also generated a confusion matrix. A confusion matrix is a table that's frequently used to define the performance of a classification model on a set of test data where the true values are known. For this example LinearSVC classifier, we found that the classifier achieved 224 correct negative review classifications and 30 negative reviews were incorrectly classified as positive. We also found that the classifier achieved 225 correct positive review classifications and 21 positive reviews were incorrectly classified as negative. (see below)



Lastly for problem 1, we calculated and graphed the performance of each parameter. (see below)



This graph shows us that the `ngram_range` of (1, 2) offers better performance than the `ngram_range` of (1, 1). In other words, bigrams (one word and two words) leads to better LinearSVC classification performance when compared to unigrams (only one word).

Conjectures

We compared three models, LinearSVC, using the same `ngram` values of (1,1), (1,2) out of these three LinearSVC and Logistic Regression performed much better than Decision Tree Classifier. The reason why Logistic Regression works good is because in two categories cases, when the classes are not well-separated, the parameter estimates for the logistic regression model are stable. **(Results of Logistic Regression and Decision Tree Classifier in the Python Notebook)**

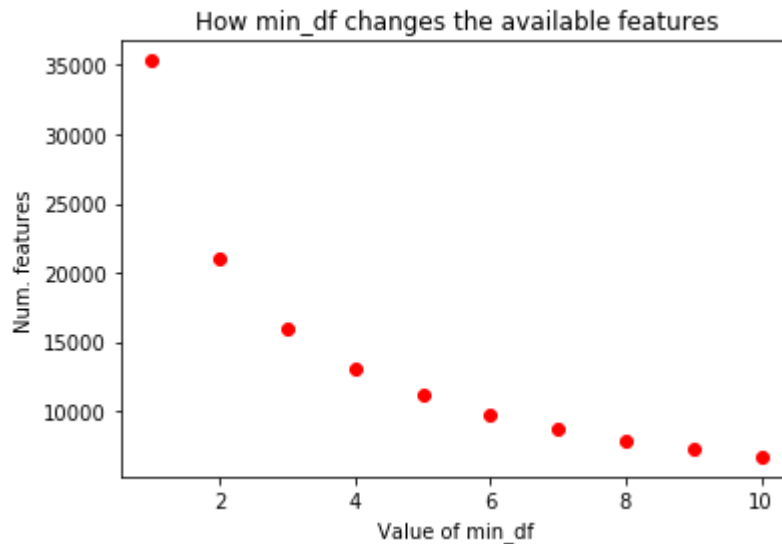
Problem 2: Explore the Scikit-learn TfidfVectorizer class

Term Frequency–Inverse Document Frequency (TF-IDF) is a “numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval, text mining, and user modeling. The TF-IDF value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general.”[2.1]

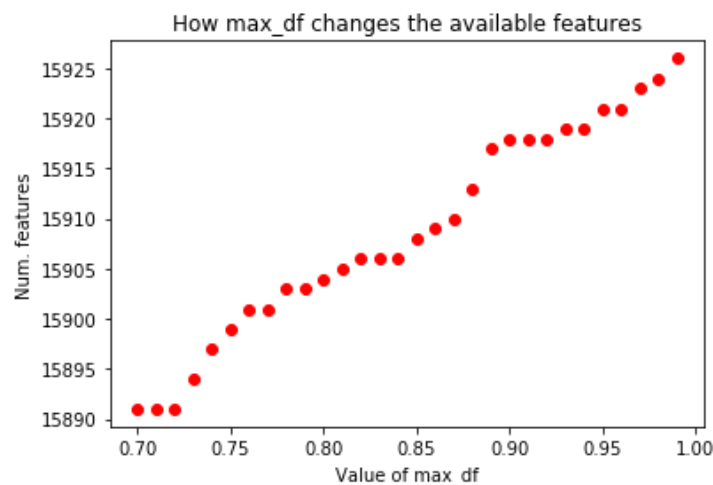
We ran the `TfidfVectorizer` class on the training data set using default parameters and we found this yielded a sparse matrix consisting of 1,500 rows and 35,321 columns which contained the tokenized features (TF-IDF) of the words in our training data set.

Next, we explored the the “`min_df`” and “`max_df`” parameters of the `TfidfVectorizer`. The parameter `min_df` is the minimum word frequency in the document and any vocabulary terms that have a document frequency lower than the given threshold should be ignored. If float, the

parameter represents a proportion of documents, integer represents absolute counts. The min_df parameter is ignored if vocabulary is not None. Using the Matplotlib library we graphed the relationship between the min_df parameter (using integer values from 1 to 10 and stepping by 1) and its effect on the number of available features in a given document. In layman's terms, the higher the min_df, the more words that are excluded as features. (see plot below)

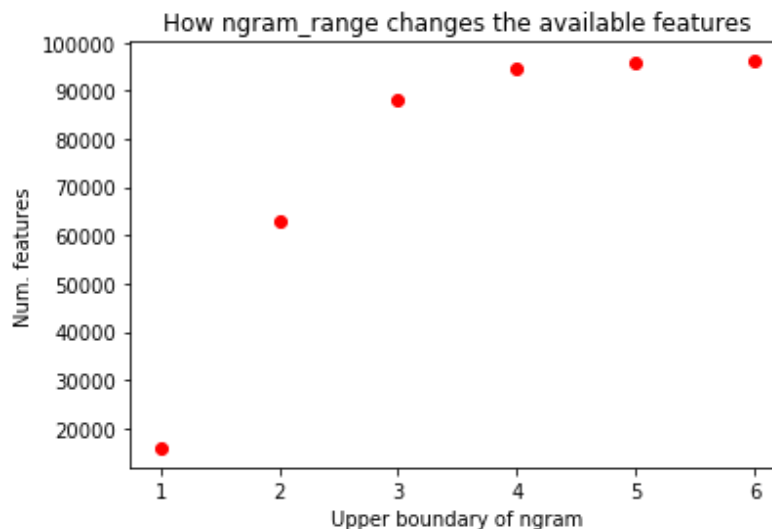


The parameter max_df is the maximum word frequency in the document and any vocabulary terms that have a document frequency higher than the given threshold should be ignored. If float, the parameter represents a proportion of documents, integer represents absolute counts. The max_df parameter is ignored if vocabulary is not None. Using the Matplotlib library we graphed the relationship between the max_df parameter (using float values from 0.7 to 0.99 and stepping by 0.01) and its effect on the number of available features in a given document. In layman's terms, the lower the max_df, the more words that are excluded as features. (see plot below)



Last, we explored the “ngram_range” parameter of TfidfVectorizer. “In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application.”[2.2] In this exploration, we limited our analysis to ‘words’.

Using the matplotlib library we graphed the relationship between the ngram_range parameter (using integer values from 1 to 7 to set the upper boundary) and its effect on the number of available features in a given document. In layman’s terms, as the upper boundary of the ngram range increased so did the number of document features, but with diminishing returns seen at the ngram range of (1, 4). (see plot below)



Conjecture

- As you increase the value of min_df that is from 1 to 10 (number of documents) the number of features reduce from 35,000 to almost 7,500.
- For the second conjecture, as you increase the value of max_df from .6 to .99, in the step of (0.01), the number of features increases slightly from 15890 to 15925.
- The third conjecture is about the n_gram range from 1 to 6. We see that the number of features increase from 20000 to almost 100,000. However, as you increase the upper bound from 4 to 6 in the increment in the number of features compared to the number of features that increase from n_gram range of 1 to 3 is extremely small.

Problem 3: Machine learning algorithms

For problem 3, we computed Xtrain and Xtest. Based on the knowledge gained from Problem 2, we first applied some parameters to TfidfVectorizer and fit it using the training data we previously stored as 'docs_train'. The TfidfVectorizer parameters were set explicitly to use 'word' for analyzer, the ngram_range was set to (1, 2), the min_df was set to 3 and the max_df was set to 0.95.

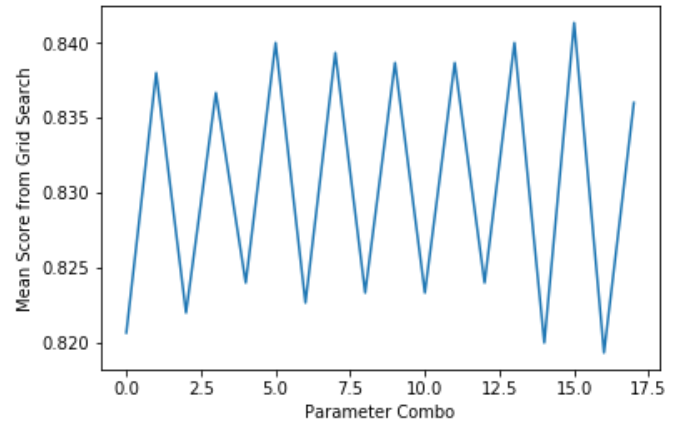
XTrain was computed using "count_vect.fit_transform(docs_train)" and XTest was computed using "count_vect.fit_transform(docs_test)". The results of these statements were checked by examining the shape and type of the computed objects. XTrain is a sparse matrix consisting of 1,500 rows and 63,008 columns. XTest is also a sparse matrix consisting of 500 rows and 24,345 columns.

Next, we examined two classifiers provided by scikit-learn by applying a number of different parameter settings for each and judged their performance using a confusion matrix. First, we applied the LinearSVC classifier on the dataset and executed the metrics.classification function to generate a text report showing the main classification metrics. (see below)

| Metrics for LinearSVC Classification – Team 14 | | | | |
|--|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| neg | 0.91 | 0.89 | 0.90 | 254 |
| pos | 0.89 | 0.91 | 0.90 | 246 |
| avg / total | 0.90 | 0.90 | 0.90 | 500 |

Then we graphed the results of our various parameter combinations for LinearSVC. (see below)

```
parameters = {
    'vect_ngram_range': [(1, 1), (1, 2)],
    'vect_min_df': [2, 3, 4],
    'vect_max_df': [0.98, 0.95, 0.90],
```



We found the best score produced by the grid search was:

```
# find best score from grid search for LinearSVC
grid_search.best_score_

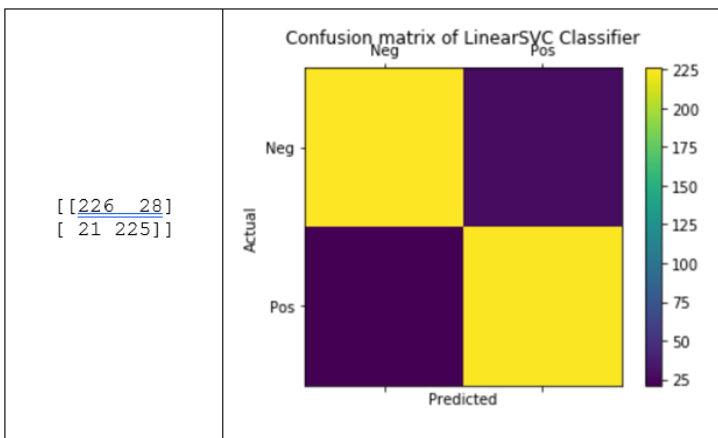
0.8413333333333333
```

The parameters used to produce the best score were:

```
# find best parameters from grid search for LinearSVC
grid_search.best_params_

{'vect_max_df': 0.9, 'vect_min_df': 3, 'vect_ngram_range': (1, 2)}
```

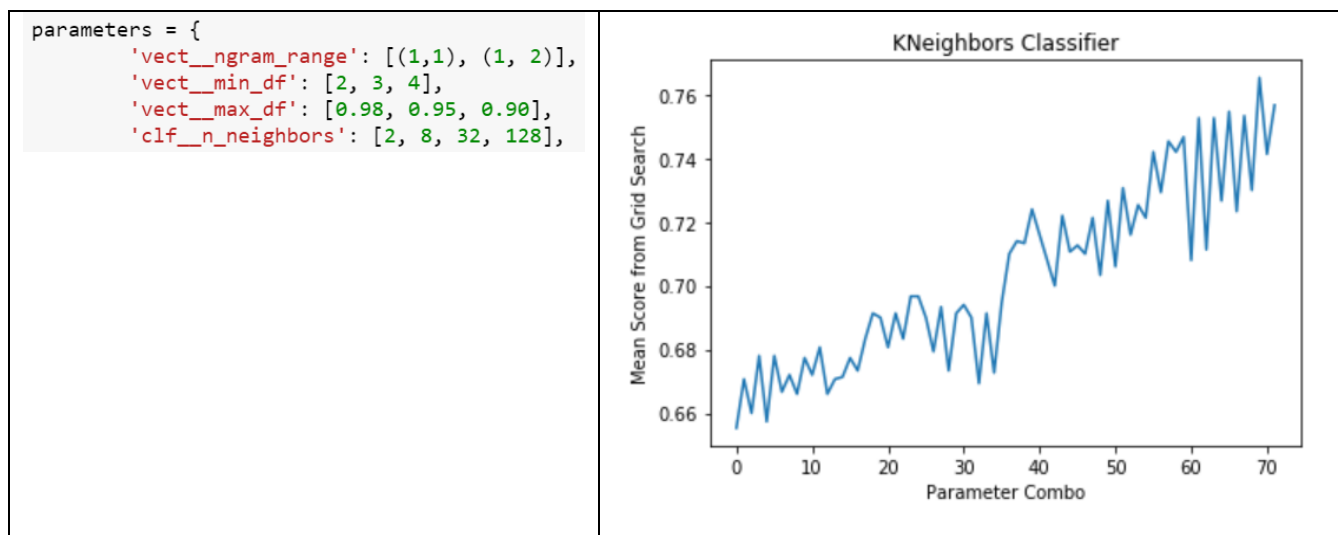
Last, we calculated the confusion matrix for LinearSVC classifier. (see below)



Next, we applied the KNeighbors classifier on the dataset and executed the `metrics.classification` function to generate a text report showing the main classification metrics. (see below)

| Metrics for KNeighbors Classification – Team 14 | | | | |
|---|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| neg | 0.82 | 0.68 | 0.74 | 254 |
| pos | 0.72 | 0.84 | 0.77 | 246 |
| avg / total | 0.77 | 0.76 | 0.76 | 500 |

Then we graphed the results of our various parameter combinations for KNeighbors. (see below)



We found the best score produced by the grid search was:

```
# find best score from grid search for KNeighbors
grid_search.best_score_

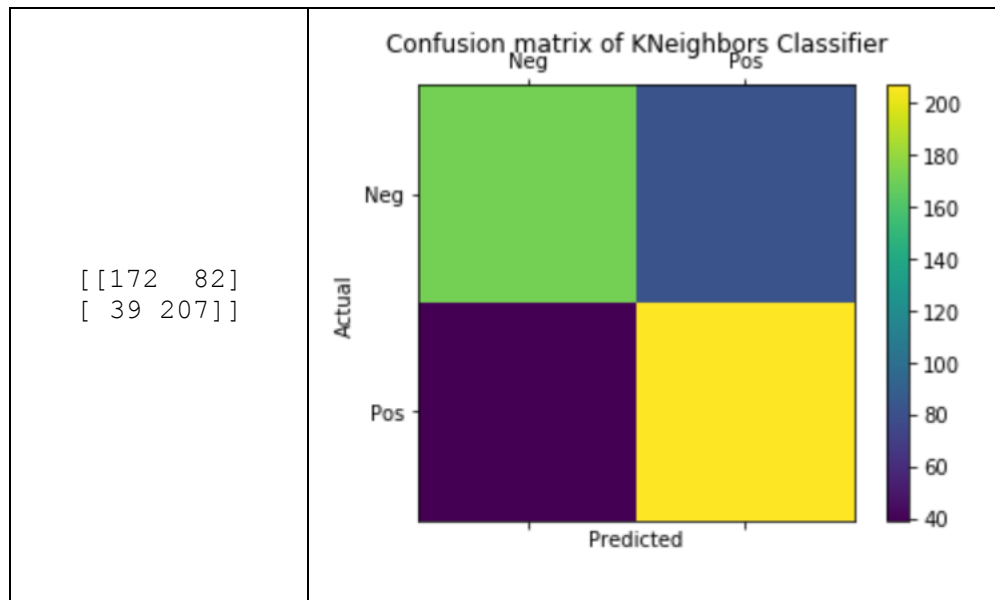
0.7653333333333333
```

The parameters used to produce the best score were:

```
# find best parameters from grid search for KNeighbors
grid_search.best_params_
```

```
{'clf__n_neighbors': 128,
 'vect__max_df': 0.9,
 'vect__min_df': 3,
 'vect__ngram_range': (1, 2)}
```

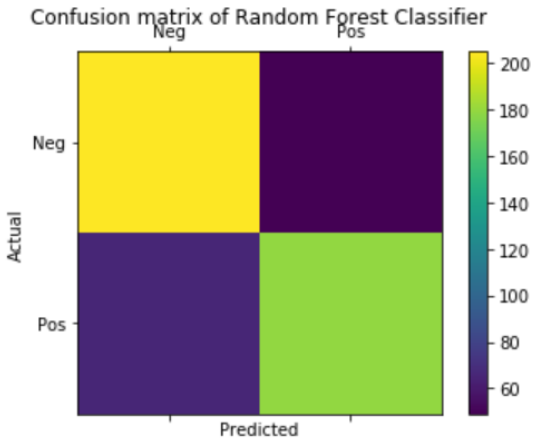
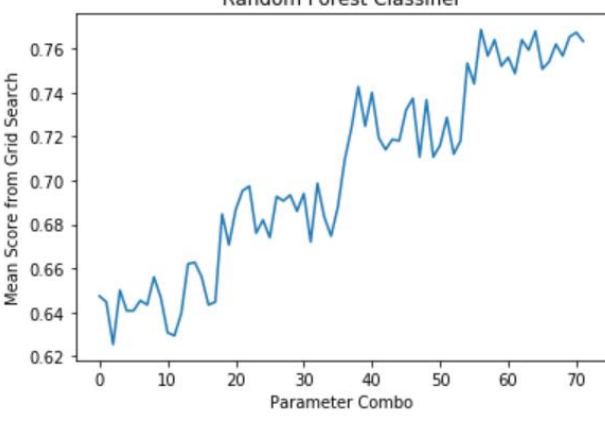
Last, we calculated the confusion matrix for the KNeighbors classifier. (see below)



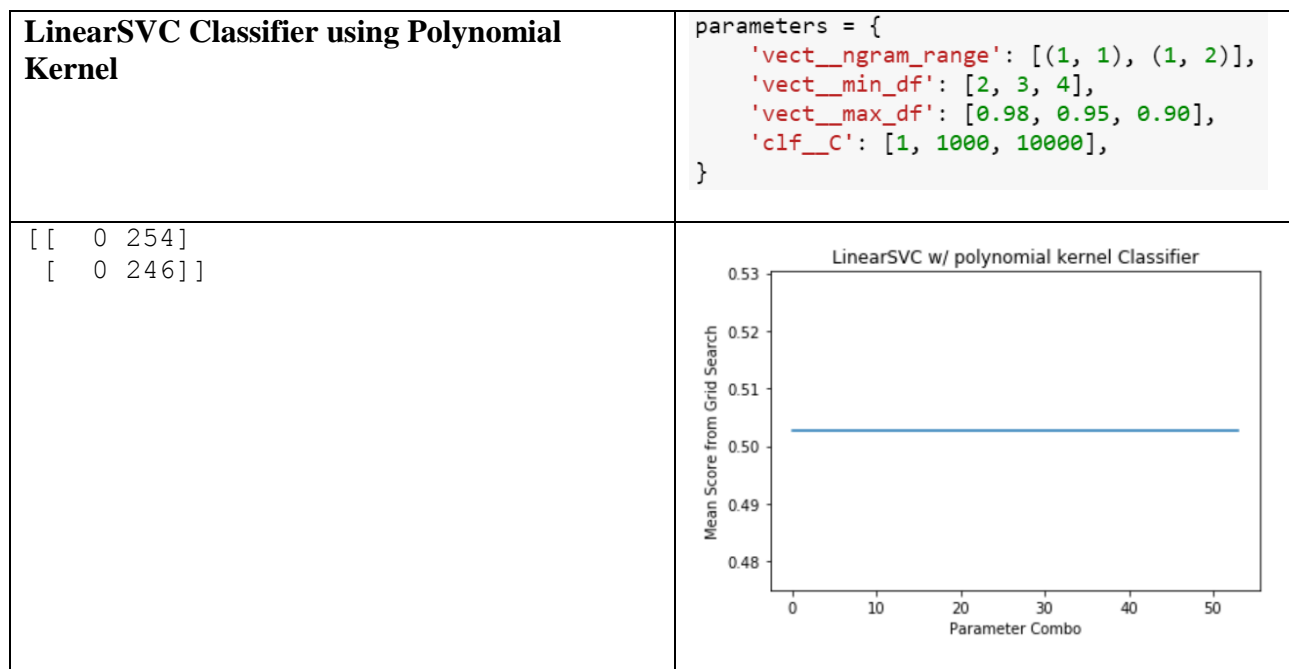
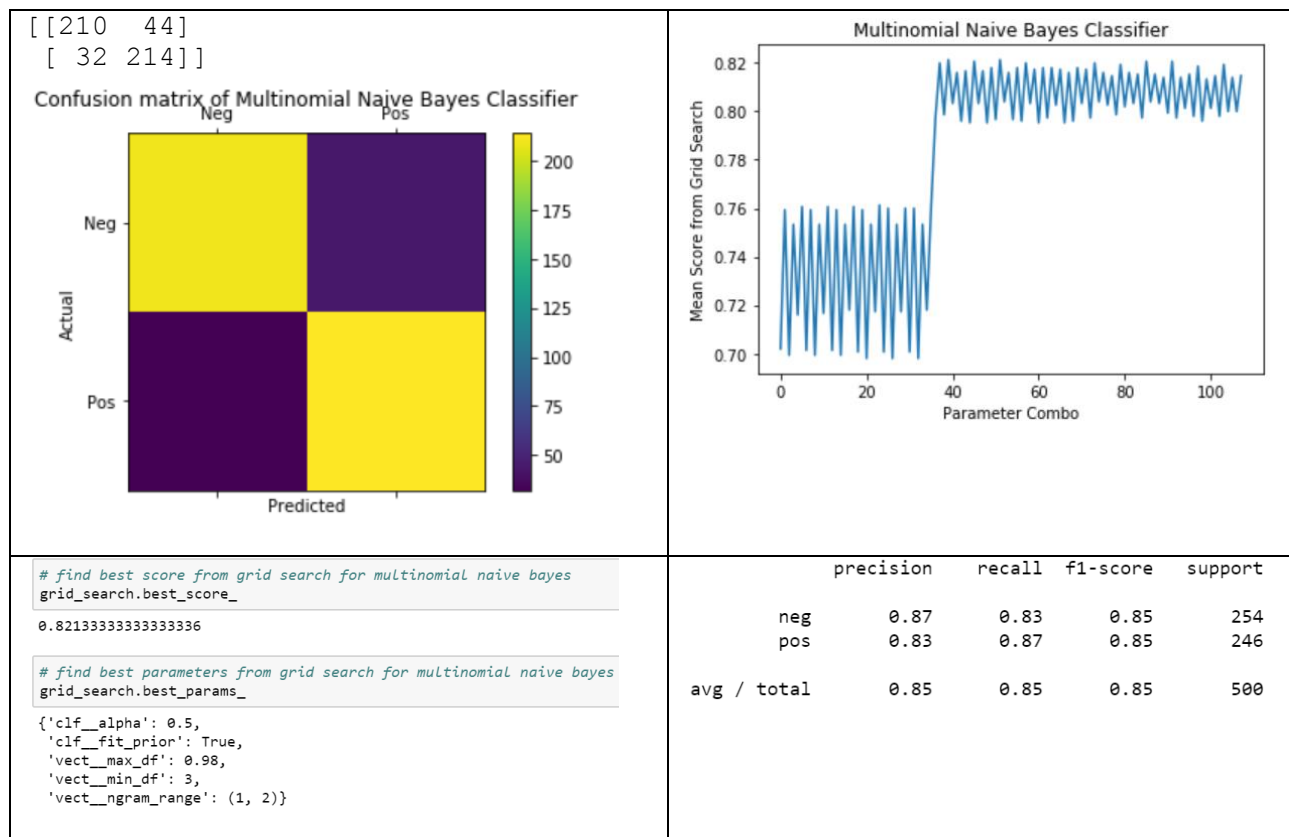
Intrigued by the findings for the LinearSVC and KNeighbors classifiers, we consulted the scikit-learn algorithm cheat-sheet [3.1] to see what other classifiers and/or methods were recommended.

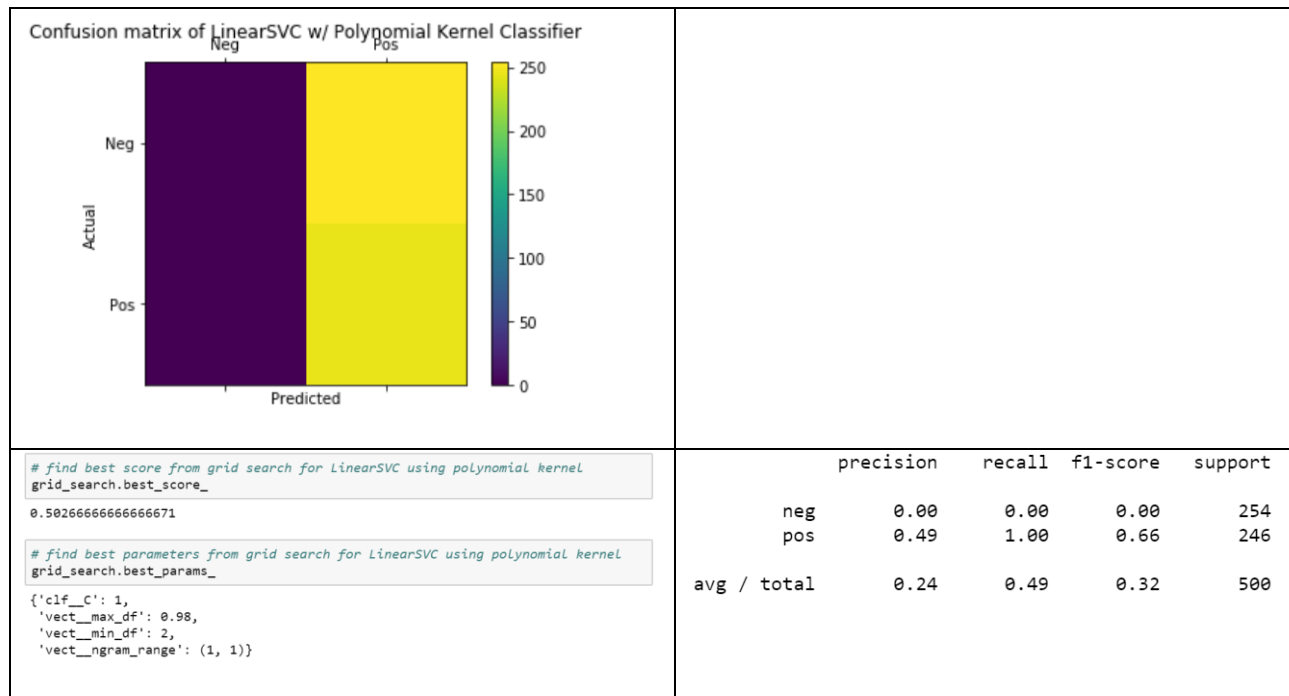
Focusing on the classification bubble, we noted that additional classifiers were available and applied various parameters to each of them to see if it was possible to achieve a better score than the LinearSVC classifier.

We present the results of each classifiers' performance in the series of tables below. We observed that none of these additional classifiers achieved a "best score" higher than the LinearSVC best score of 0.8413333333333333

| <h1>Random Forest Classifier</h1> | <pre>parameters = { 'vect__ngram_range': [(1, 1), (1, 2)], 'vect__min_df': [2, 3, 4], 'vect__max_df': [0.98, 0.95, 0.90], 'clf__n_estimators': [8, 16, 32, 64], }</pre> | | | | | | | | | | | | | | | | | | | | |
|--|---|--------|-----------|---------|----------|---------|-----|------|------|------|-----|-----|------|------|------|-----|-------------|------|------|------|-----|
| <pre>[[205 49] [66 180]]</pre> <p>Confusion matrix of Random Forest Classifier</p>  <p>Actual</p> <p>Predicted</p> | <p>Random Forest Classifier</p>  <p>Mean Score from Grid Search</p> <p>Parameter Combo</p> | | | | | | | | | | | | | | | | | | | | |
| <pre># find best score from grid search for Random Forest grid_search.best_score_ 0.76866666666666672 # find best parameters from grid search for Random Forest grid_search.best_params_ {'clf__n_estimators': 64, 'vect__max_df': 0.98, 'vect__min_df': 3, 'vect__ngram_range': (1, 1)}</pre> | <table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>neg</td><td>0.76</td><td>0.81</td><td>0.78</td><td>254</td></tr><tr><td>pos</td><td>0.79</td><td>0.73</td><td>0.76</td><td>246</td></tr><tr><td>avg / total</td><td>0.77</td><td>0.77</td><td>0.77</td><td>500</td></tr></table> | | precision | recall | f1-score | support | neg | 0.76 | 0.81 | 0.78 | 254 | pos | 0.79 | 0.73 | 0.76 | 246 | avg / total | 0.77 | 0.77 | 0.77 | 500 |
| | precision | recall | f1-score | support | | | | | | | | | | | | | | | | | |
| neg | 0.76 | 0.81 | 0.78 | 254 | | | | | | | | | | | | | | | | | |
| pos | 0.79 | 0.73 | 0.76 | 246 | | | | | | | | | | | | | | | | | |
| avg / total | 0.77 | 0.77 | 0.77 | 500 | | | | | | | | | | | | | | | | | |

| | |
|---|--|
| Multinomial Naïve Bayes Classifier | <pre>parameters = { 'vect__ngram_range': [(1, 1), (1, 2)], 'vect__min_df': [2, 3, 4], 'vect__max_df': [0.98, 0.95, 0.90], 'clf__alpha': [0, 0.5, 1], 'clf__fit_prior': [True, False] }</pre> |
|---|--|





Conjectures

First we compared LinearSVC with KNN and LinearSVC performed better than KNN. The specific feature of LinearSVC that performed better is max_df of 0.9, min_df = 3, and ng = (1,2). For other models, we used RandomForest, Multinomial Naïve Bayes Classifier, LinearSVC using Polynomial Kernel with max_df of 0.5 and all of the stop words removed. As we compare LinearSVC with the max_df 0.5 and stop words removed, we only see a slight decrease in accuracy, which means a lot of the words don't add weight to the performance of the classifier.

Problem 4: Finding the right plot

Unfortunately, we were not able to find a two-dimensional plot in which the positive and negative reviews are **clearly separated**. Manifold and PCA do not reduce the high dimensionality to point where a meaningful 2D plot can be constructed, **although some intersection was obtained between positive and negative reviews**.

To find a right plot to categorize positive and negative reviews, we implemented dimension reduction techniques of Manifold Learning and Principal Component Analysis, to try and draw two dimensional plots in which the positive and negative reviews can be separated.

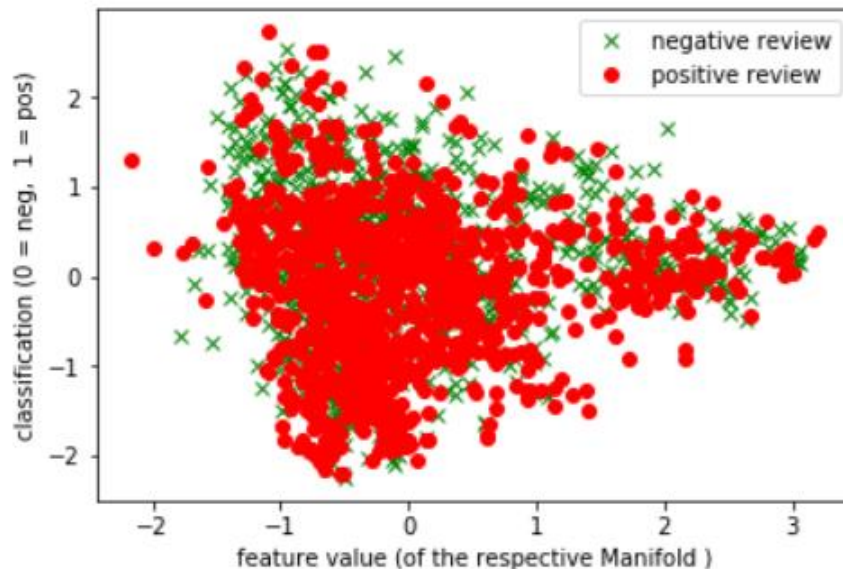
The reason why we chose Manifold Learning was that Manifold learning is an approach to non-linear dimensionality reduction. Algorithms for this task are based on the idea that the dimensionality of many data sets is only artificially high. [4.5] And with thousands of dimensions in the problem, this was our priority algorithm.

We tried three different Manifold Learning techniques:

- Isometric Mapping
- Locally Linear Embedding
- Spectral Embedding

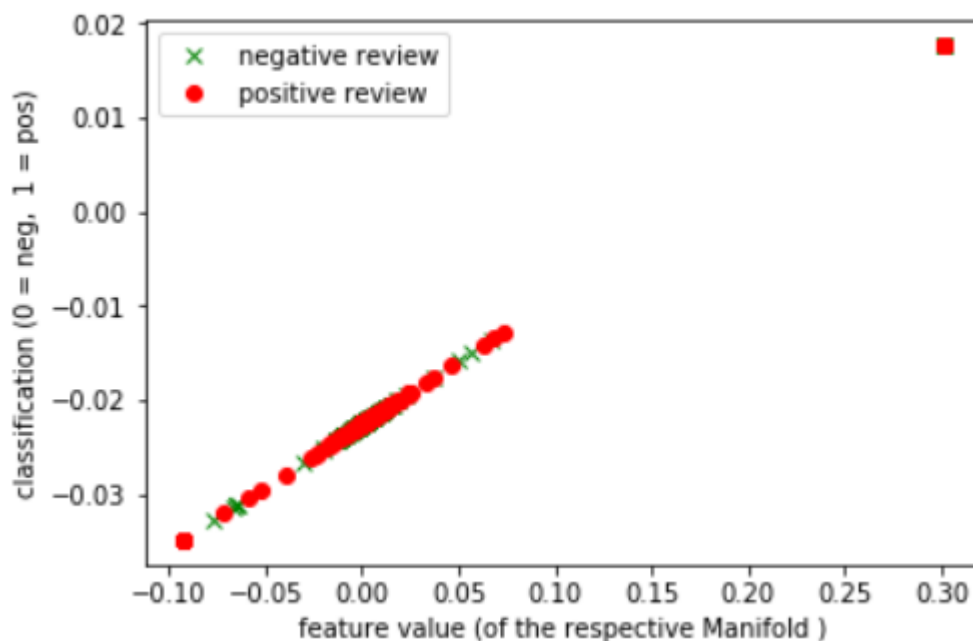
The first algorithm is Isometric Mapping(IsoMap). For high-dimensional data from real-world sources, isometric mapping (IsoMap) seems to generally lead to more meaningful embeddings. IsoMap creates a graph by connecting each instance to its nearest neighbors, then reduces dimensionality while trying to preserve the geodesic distance between the instances [4.1]. The geodesic distance between two nodes in a graph is the number of nodes on the shortest path between these nodes.

The following graph is plotted with 8 neighbors to consider for each point. The green point is the negative review towards a movie while the red dot is a positive review



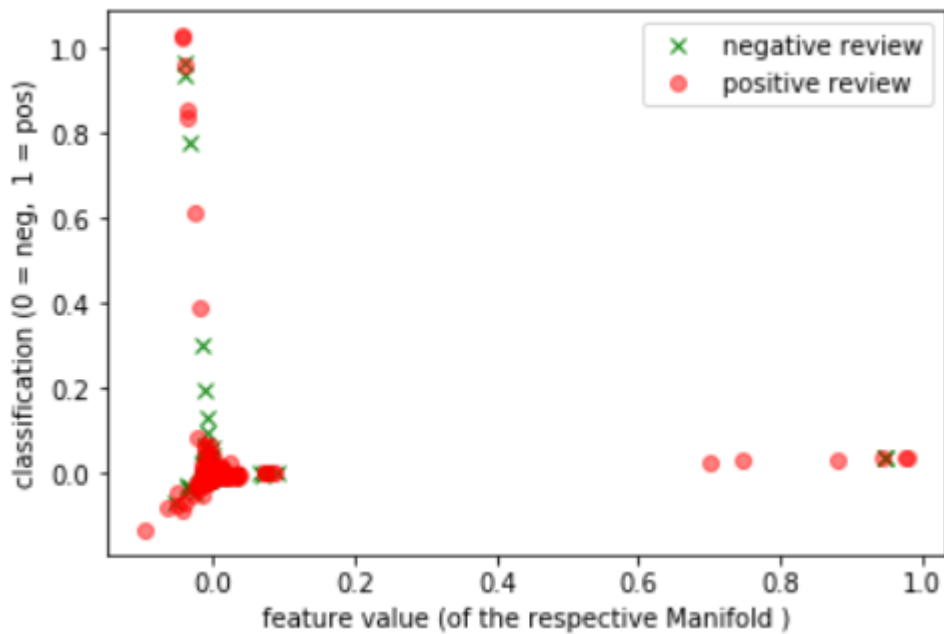
The second one is Locally Linear Embedding (LLE). LLE is a Manifold Learning technique that does not rely on projections. In a nutshell, LLE works by first measuring how each training instance linearly relates to its closest neighbors, and then looking for a low-dimensional representation of the training set where these local relationships are best preserved (more details shortly). This makes it particularly good at unrolling twisted manifolds, especially when there is not too much noise. [4.2] Locally linear embedding (LLE) seeks a lower-dimensional projection of the data which preserves distances within local neighborhoods. It can be thought of as a series of local Principal Component Analyses which are globally compared to find the best non-linear embedding.

The following heat map is plotted with 8 neighbors to consider for each point. The green point is the negative review towards a movie while the red dot is positive review.



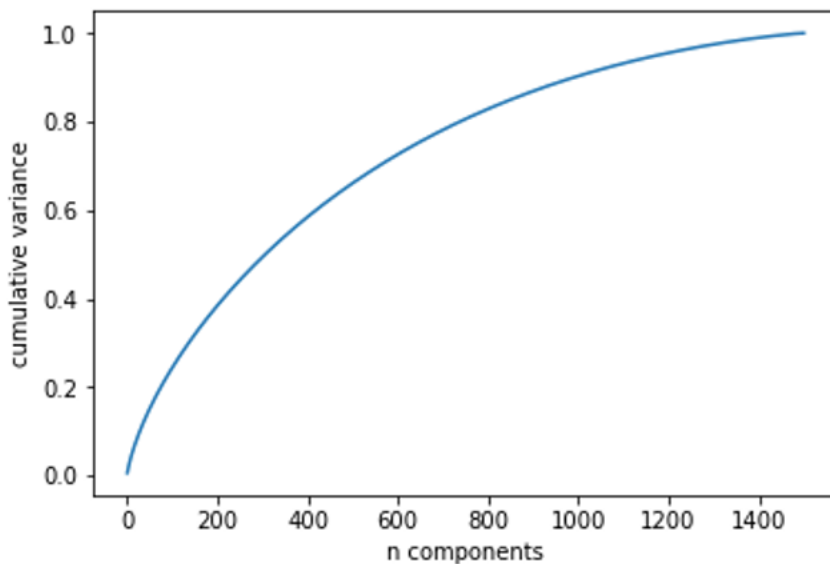
The third one is Spectral Embedding. Spectral embedding is used for non-linear dimensionality reduction. It works by Forming an affinity matrix given by the specified function and applies spectral decomposition to the corresponding graph Laplacian. The resulting transformation is given by the value of the eigenvectors for each data point [4.3].

The following heat map is plotted by constructing affinity matrix by knn graph. The green point negative review towards a movie while the red dot is positive review.



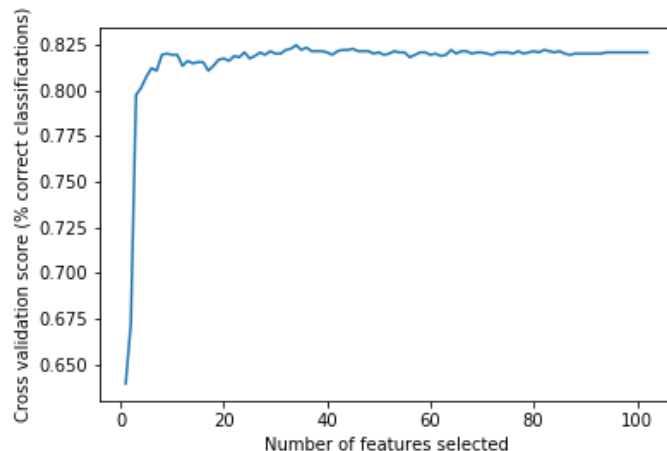
To compare our methods of Manifold Learning with another Dimension Reduction Algorithm, we tried Principal Component Analysis(PCA).

PCA is by far the most popular dimensionality reduction algorithm [4.4]. First it identifies the hyperplane that lies closest to the data and then it projects the data onto it. This time we try to find out the minimum number of dimensions to preserve 90% of the variance in our data.



We see that for this data, nearly 1000 components are required to preserve # 90% of the variance: this tells us that the data is intrinsically very high dimensional — it can't be described linearly with just a few components.

Additionally, we derived a conjecture RFE to create a visualization to demonstrate the optimal number of features, which is 20,168.



Business Application

Our idea of Developing a Minimum Viable Product (MVP) in Case Study 2 and recommending Action Genre for movies could be validated through our present analysis of Case Study 3 as well as some further analysis.

- Simply pick a start year, say 1946, and obtain list of all action movies obtained till 2004 (As the dataset was released in 2004)
- Based on the data obtained of the list of action movies, we can create two lists of distinct words, one from all positive reviews and the other from all negative reviews after removing the stop words.
- In the next step, we could compare the words present in all the positive reviews to the list of action movies obtained, and compare the matches. We perform a similar step for negative reviews and compare the matches.
- If more action movies are rated in positive reviews, as compared to negative reviews, we can support our claim of Case Study 2 that Action movies are more liked by the audiences, in general.

Conclusion

We tried manifold learning and PCA. Using manifold learning we tried separating the data points, but we were unsuccessful to clearly separate it. However, we were successfully able to reach to a stage where we created two dimensional plots, that showed positive and negative reviews, but in overlapped regions.

When we used PCA, we learned that more than 1000 features are required to preserve 90% of the variance. Through this, we learned that the data is intrinsically very high dimensional.

Scope of Improvement

- Learn about other Dimensionality Reduction techniques and implement them: Graph Based Kernel PCA, Linear Discriminant Analysis
- Experiment with different combinations of the method we tried and see if we can generate a 2-D Plot that separates the reviews.
- Since reducing it to two 2 dimensions was not giving a clear demarcation, we can try reducing it to more than 2 dimensions and use plots like Parallel Coordinates plot to categorize reviews.

Citations

- [1.1] <http://www.cs.cornell.edu/people/pabo/movie-review-data/>
- [1.2] <https://en.wikipedia.org/wiki/Scikit-learn>
- [2.1] <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>
- [2.2] <https://en.wikipedia.org/wiki/N-gram>
- [3.1] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
- [4.1] <https://jakevdp.github.io/PythonDataScienceHandbook>
- [4.1] Book: Machine Learning with Scikit-Learn and TensorFlow. Page 224
- [4.2] Book: Machine Learning with Scikit-Learn and TensorFlow. Page 221
- [4.3] <http://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html>
- [4.4] Book: Machine Learning with Scikit-Learn and TensorFlow. Page 211
- [4.5] <http://scikit-learn.org/stable/modules/manifold.html>