Introduction to Data Science

DS501 Case Study 3 Team 5

Textual Analysis of Movie Reviews

Submitted By:
Harsh Nilesh Pathak,
Jidapa Thadajarassiri,
Krushika Tapedia,
Pitchaya Wiratchotisatian,
Prince Shiva Chaudhary.

Supervisor:
Dr. Randy Paffenroth

Overview and Motivation

Movie reviews are the valuable source for any movie company since many useful information can be extracted in order to understand customers' need and their perspective of movies. This will be help a movie company wisely make a business plan and eventually increase company's benefit. However, analysing movie reviews is not a trivial subject; many detail tasks need to be applied and well taken care especially data preprocessing.

In this study, we aim to explore good methods that can be used for textual analysis of movie reviews. Among several methods that will be explored, we first study sentiment analysis since it is the most popular method in this area. We then apply TfidfVectorizer, a very useful function from scikit-learn, to transform raw data into a matrix. LinearSVC and KNeighborsClassifier provided by scikit-learn are also used for classification task. Lastly, we will figure out the way of making a 2D plot that can classify positive reviews and negative reviews.

Data Description

Polarity dataset v2.0, used in this study, is a collection of movie-review documents that is availble online at http://www.cs.cornell.edu/people/pabo/movie-review-data/. This dataset contains processed movie reviews that separated into 1,000 positive and 1,000 negative reviews.

Part 1: Sentiment Analysis on movie reviews

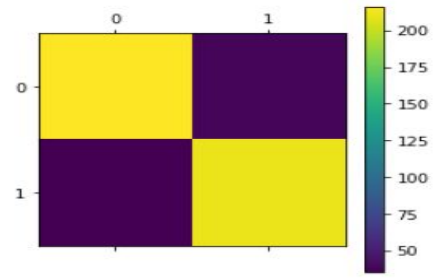
In this part, a code for doing sentiment analysis on movie reviews is studied. We basically follow the code from doc/tutorial/text_analytics/solutions/exercise_02_sentiment.py. It starts by using train_test_split to split our data into training data, training labels, testing data and testing labels following by applying pipeline available in scikit-learn to easily call ordered classifiers. Then GridSearchCV is used to help tuning best parameters set for an algorithm; therefore the prediction of testing data can be done by applying grid_search.predict. Finally, confusion_matrix is constructed as a tool for evaluation. Several evaluating score can be calculated based on this confusiton_matrix such as overall error rate, overall accuracy rate, precision, recall and f1-score.

```
             precision    recall  f1-score   support

        neg       0.86      0.85      0.86       254
        pos       0.85      0.86      0.85       246

avg / total       0.85      0.85      0.85       500

[[216  38]
 [ 35 211]]
```



Part 2: Explore the scikit-learn TfidfVectorizer class

Term frequency–inverse document frequency, TF-IDF, is a widespread used statistic in text mining. This statistic is a great capture of how important a word is since it reflects not only on how frequent any word occur in a document but also offset if that word commonly appears in the whole corpus. In this part, we will begin with the definition of TF-IDF and then explore a useful function "TfidfVectorizer" from scikit-learn that automatically convert a collection of raw text documents to a matrix of TF-IDF features.

TF-IDF Definition:

For a document d in a corpus containing N documents, the term frequency–inverse document frequency (TF-IDF) statistic shows how important a term/word t is. The formula for TF-IDF is

$$TF{-}IDF(t,d,N) = tf(t,d)*idf(t,N)$$

where

$tf(t,d)$ - term frequency - is the frequency of term t in a document d calculated by

$$tf(t,d) = \frac{the\ number\ of\ word\ t\ in\ a\ document\ d}{the\ number\ of\ total\ words\ in\ document\ d}$$

$idf(t,N)$ - inverse document frequency - is the adjusted weight of term t in the corpus. This weight reflects how common that term is in the whole corpus. It is calculated by

$$idf(t,N) = log\ \left(\frac{N}{|\{d \in N : t \in d\}|}\right)$$

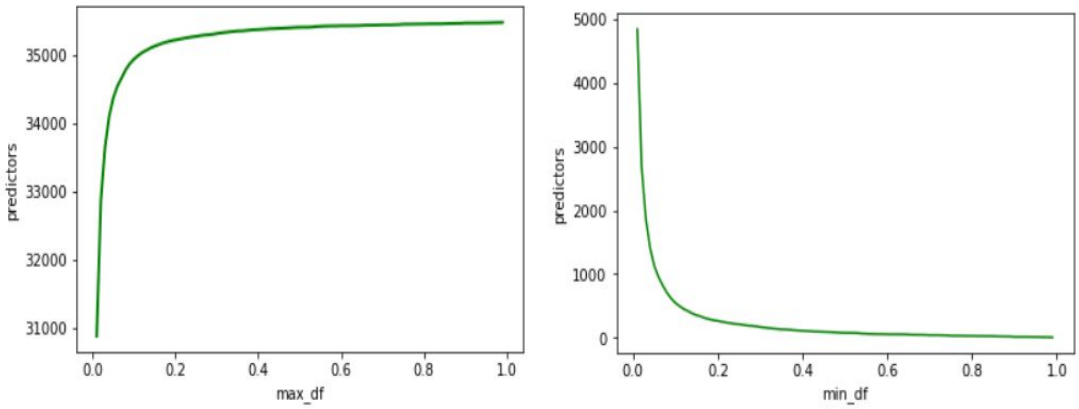
However, in order to avoid the case that there is no document containing that term t, it is common to add 1 into the denominator. For a high TF-IDF, it can come from either high frequency of term t in a document d (high $tf(t,d)$) or high weight since the term t is rare in the whole corpus (high $idf(t,N)$).

Explore the TfidfVectorizer

We explore TfidfVectorizer by running it on training data. This function converts training data to a TF-IDF matrix with approximately 35,000 features. Several parameters can be easily set to get a desired output. Three useful parameters (min_df, max_df and ngram_range) are further explored as following.

Explore the min_df and max_df parameters of TfidfVectorizer

Min_df is a minimum threshold of the frequency of documents containing term t in a corpus. Max_df is a maximum threshold of the frequency of documents containing term t in a corpus. For both of them, identifying as an integer means the absolute counts of documents containing term t in a corpus; while identifying as a float means a proportion of documents containing term t in a corpus (range = [0.0,1.0]).



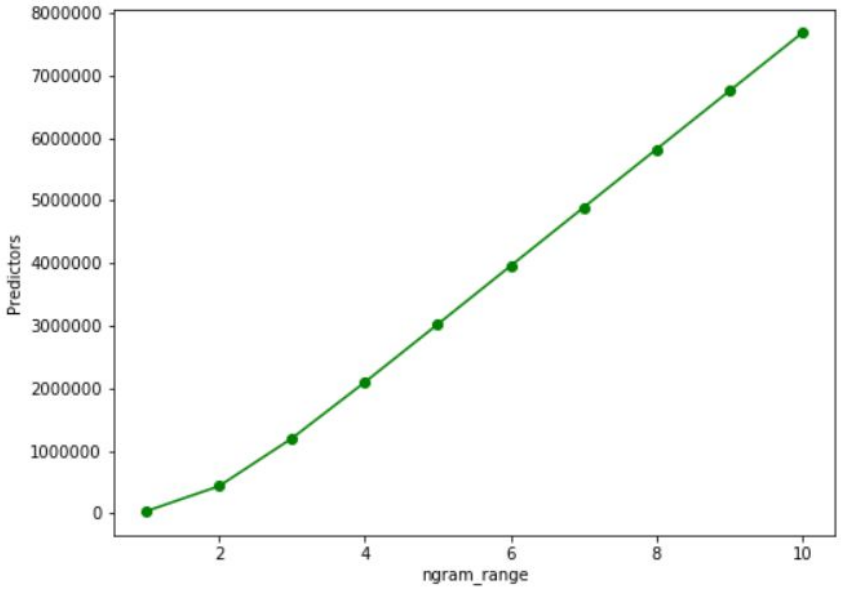
Left figure : variation of the number of predictors with max_df
Right figure : variation of the number of predictors with min_df.

These parameters fill out the terms that the number of documents containing them is either less than min_df or greater than max_df. Therefore, the number of features in the matrix of TF-IDF will be less when setting greater min_df or lower max_df.

Explore the ngram_range parameters of TfidfVectorizer

ngram_range is the lower-bound and upper-bound thresholds of values for n-grams, combinations of adjacent words, that are used for extracting features from raw text documents.

Feature extraction will be processed according to the specified range in this parameter. For example, if ngram_range = (1,2), all unigram (1-word) and bigram (2-word) will be considered as the features. When either the range is greater or the upper-bound is bigger, the number of features in the matrix of TF-IDF will be much higher.



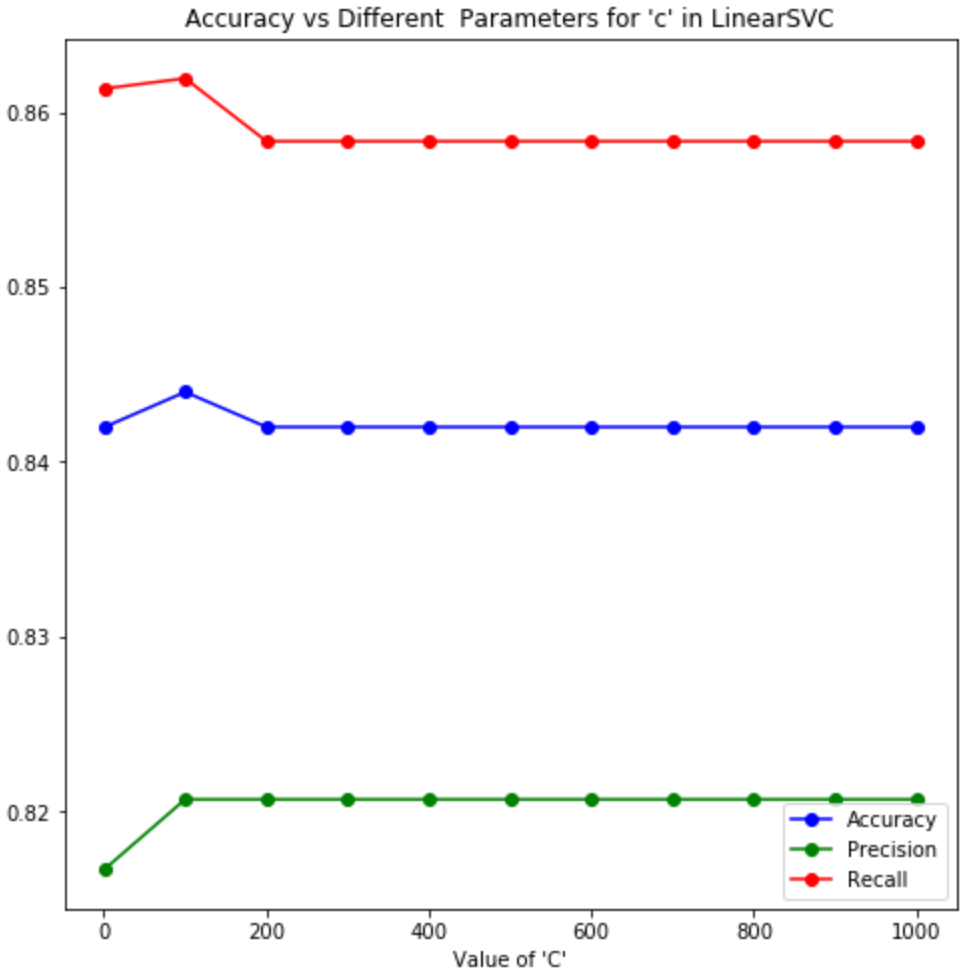
Part 3: Machine learning algorithms

Two machine learning algorithms are explored in this section, LinearSVC and KneighborsClassifier. A number of different parameters settings are tested in each algorithms. Then their performances are compared using evaluation score from confusion matrix.

LinearSVC

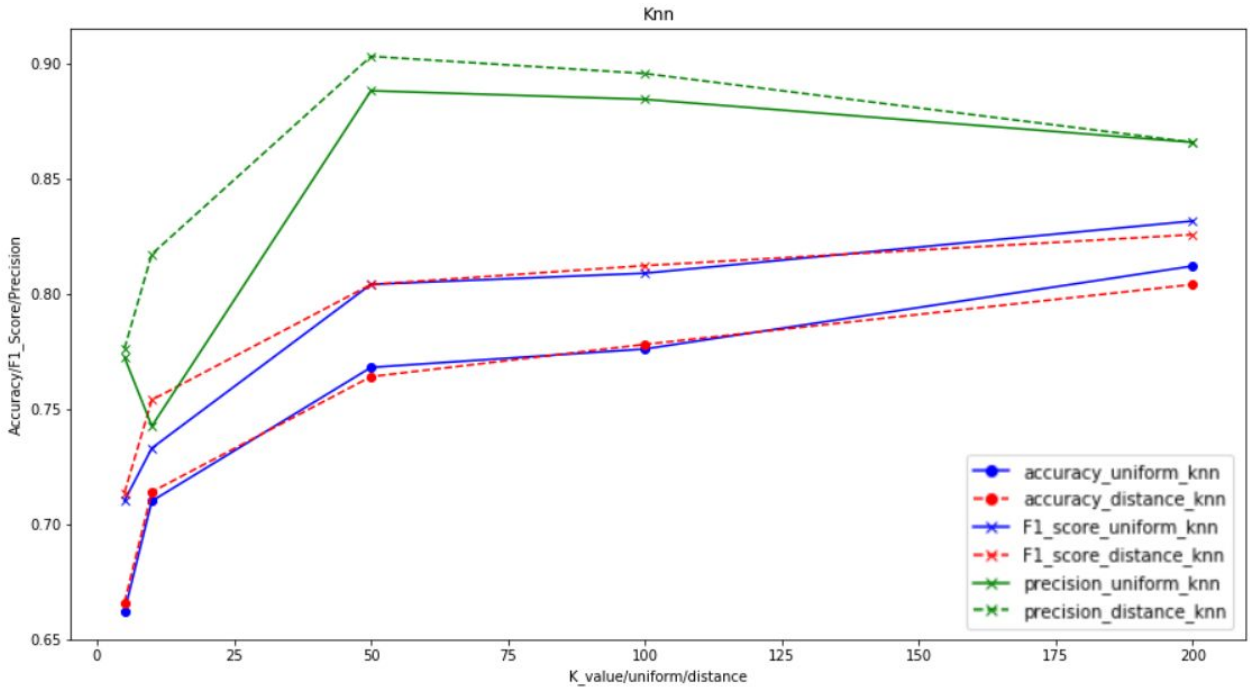
Firstly, we will perform LinearSVC on the dataset after applying the TdidfVectorizer with the parameters set as min_df = 2, Max_df =0.95 and ngram_range = (1,2). Now building a LinearSVC Classifier whose parameter 'c' we are going to .

The parameter 'C' tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. For very tiny values of C, you should get misclassified examples, often even if your training data is linearly separable.



KneighborsClassifier

Created our test and training dataset after applying TfidfVectorizer with min_df = 2, Max_df = .95, and ngram_range = (1,2). After dataset creation, used knn classifier to fit and predicted output class for test dataset for different value_of_k = [5,10,50,100,200]. We then compared overall accuracy, f1_score, and precision for each parameter set on same plot as shown below. We noticed that for $k \geq 50$, the precision of all parameters sets tend to decrease while overall accuracy and f1-score slightly increase.



Does one classifier, or one set of parameters work better?

Yes, because from confusion matrix there are several evaluation metrics can be calculated and used to compare results from different classifiers or different sets of parameters such as overall error rate, precision recall and f-score. We can choose one of them as the criteria to select the better model depending on which matrix evaluation is more reasonable to the domain.

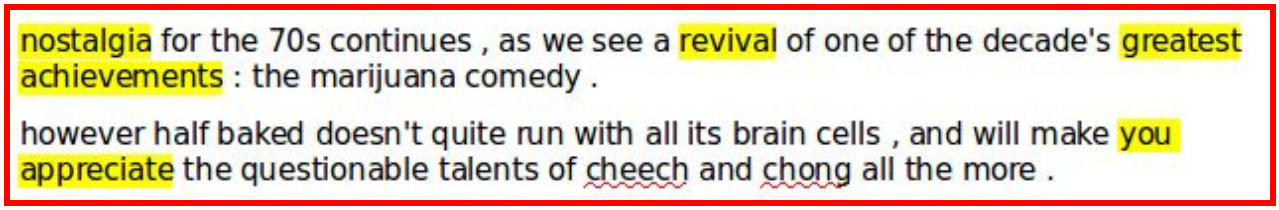
For example, from KNN Classifier, we ran several experiments with different parameters sets. If our domain concerns more about the precision, we will use precision as our criteria here. From the graph, we noticed that the model with k = 50 and weight = distance produced highest precision among other models.

For a particular choice of parameters and classifier, look at 2 examples where the prediction was incorrect. Can you conjecture on why the classifier made a mistake for this prediction?

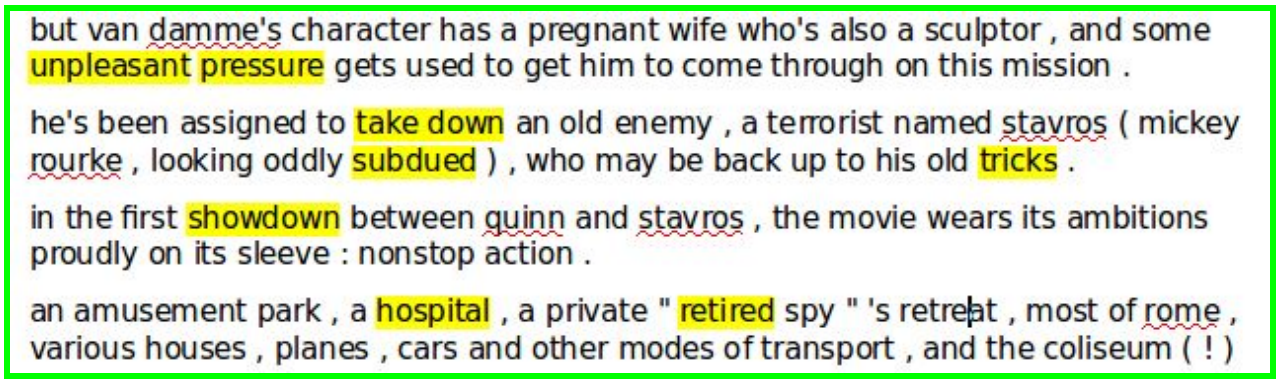
Looking at the confusion matrix we know that there we had 29 documents in each class that were misclassified in LinearSVC model. So we drew out 2 examples and performed misclassification analysis on those documents.

	True Pos	True neg
Pred Pos	203	29
Pred neg	29	239

Below, snapshot of a **negative** document **classified** as **positive** by our classifier, because of excessive use of sarcasm, using positive words but denying the fact that review was actually negative.



Below, snapshot of a **positive** document **classified** as **negative** by our classifier, because the user in this review explains really well about action movie explains about the fights in the action movie which increases the negative word count in the review and hence our classifier misclassified it as negative.



Part 4: Finding the right plot

Finding the right plot for the data set was challenging mainly because of the the overlapping features (words) that are presenting in both positive and negative documents. To tackle the problem, we applied many clustering techniques in various combinations and further decomposed it using different techniques explained step by step below.

Step 1. Data Preprocessing

Main advantage of this step was to reduce the size of features and we succeeded in that. We were able to reduce the feature size from 10k to approximately 6k.

- Stop-words and punctuations removed
- Stemming of the words to its root words using nltk.stem.snowball
- Finally creating TF-IDF Vector matrix.

Step 2. Feature Selection

After preprocessing the tf-idf matrix, we obtained over 38,000 features. Since the result matrix is sparse, we wanted to select only some features that are important for the analysis. We then performed multiple dimensional reduction techniques in scikit-learn. The following three techniques were applied to the data:

- linear_model.LogisticRegression with l1 penalty: We performed a cross validation to select the turning parameter. Applying lasso regularization resulted in a model with around 75% accuracy and around 50 nonzero coefficients. To make it easy to call and follow in the report, we will call the selected features from this technique as **LogRegLassoIndex**.
- linear_model.Lasso: Similarly to above, now we applied Lasso in scikit-learn. We obtained only 5 nonzero coefficients from our optimized model. We will call the selected features from this technique as **LinearLassoIndex**.
- decomposition.TruncatedSVD: We performed PCA on the sparse tf-idf matrix and selected the top-2 principal components (PCs). The first two PCs explained variance around 60% of the data.
- We also experimented with the full tf-idf matrix and also the matrix result from TruncatedSVD choosing n_components = 256, since they explain over 75% variance in the data.

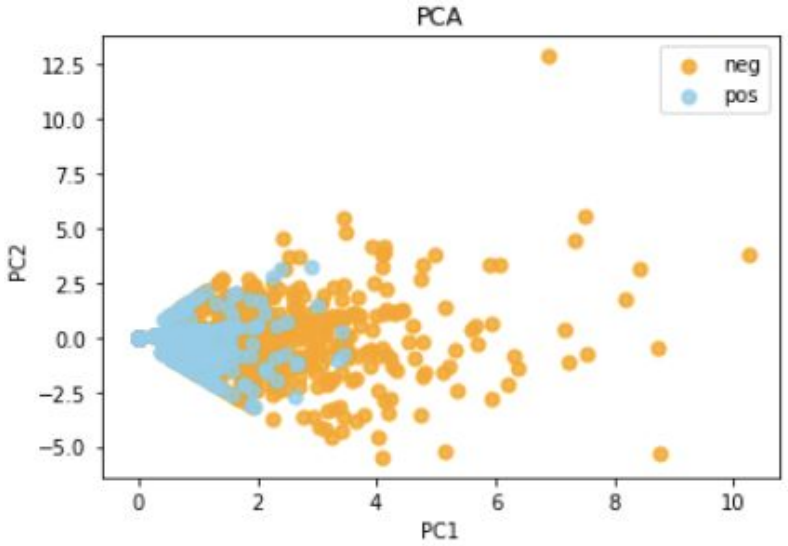
Step 3. Methodologies

1. Feature Selection: TruncatedSVD, LinearRegression and LogisticRegression with Lasso
2. Clustering: K-Means clustering, Hierarchical clustering
3. Ensemble Methods: RandomTreesEmbedding
4. Manifold Learning: MDS, Isomap, Spectral decomposition, Locally Linear Embedding, t-SNE

Step 4. Results and Plots

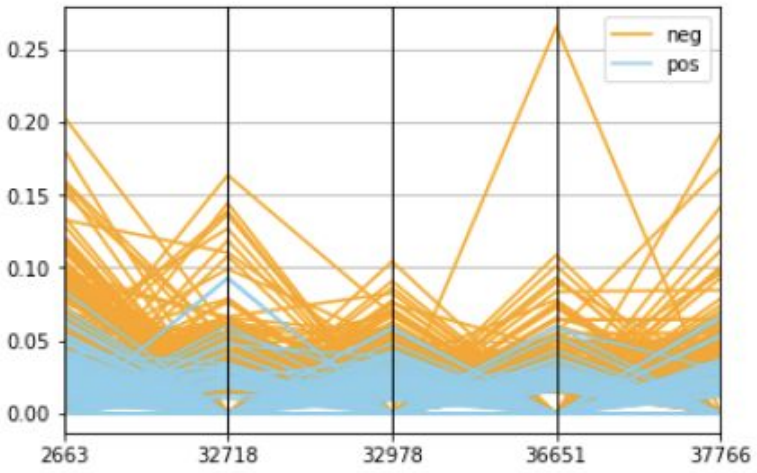
1. **Feature Selection**

1.1. TruncatedSVD



We applied TruncatedSVD with features obtained from LinearLassoIndex. The two classes are overlapping. However, we can see the variation of positive reviews (in blue) is less than negative reviews (in orange) since the points are close to each other. While the negative reviews span to the right of the plot.

1.2. Lasso



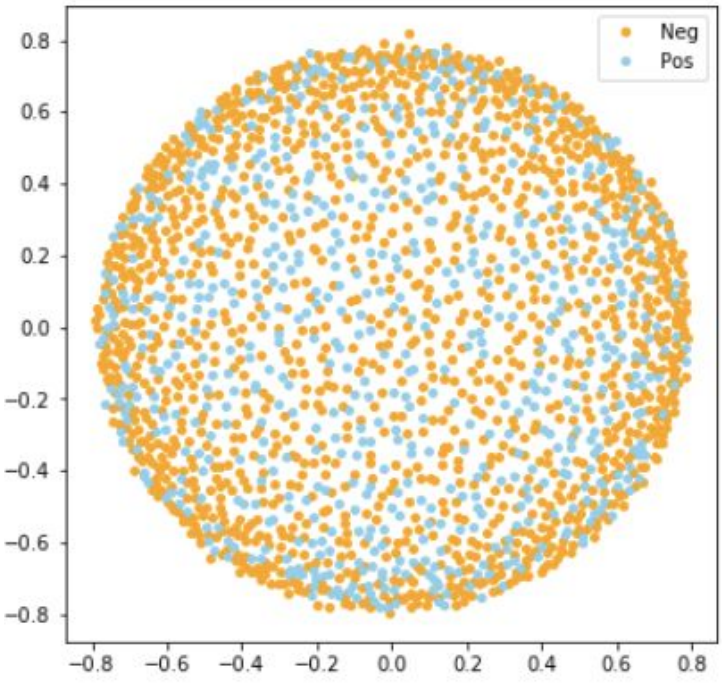
We created a parallel coordinates plot based on LinearLassoIndex. The result does not show any clear pattern, however, on average for every selected features, the negative reviews (in orange) has larger values of tf-idf compared to the positive reviews (in blue).

2. Clustering:

2.1. K-Means clustering

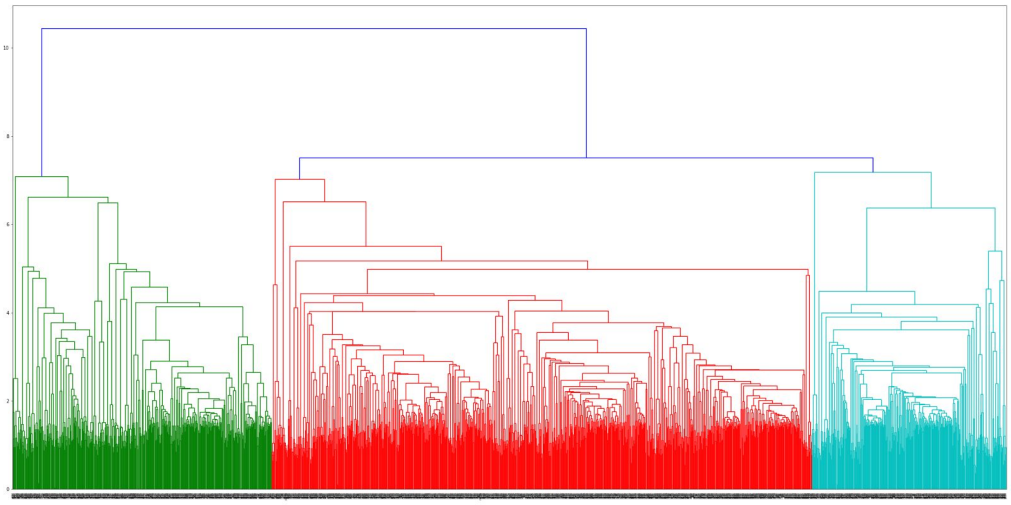
We first compute the pairwise distances between each review using cosine similarity. Then we apply manifold.MDS with 2 components to transform data into 2-dimension.

distance = 1 - cosine_similarity(tf-idf_matrix)



2.2. Hierarchical clustering

We create a dendrogram from the distance matrix computed previously using cosine similarity. We only try with the ward's minimum variance method for the linkage.



The result shows 3 main clusters. All clusters contain indistinguishable amount of negative and positive reviews as shown below. Up to this point, we still cannot classify reviews by 2D plots.

```
['p434', 'p819', 'p823', 'p232', 'p329', 'p415', 'p613', 'p75', 'p10', 'p398', 'p505', 'p199', 'p785',
 'p707', 'p958', 'p425', 'n67', 'p136', 'p403', 'n726', 'p5', 'n548', 'p403', 'p505', 'p370', 'p696',
'p179', 'p1000', 'p265', 'p322', 'p188', 'p837', 'p749', 'p79', 'p283', 'p965', 'n365', 'n935', 'n390',
'p534', 'n370', 'n582', 'n151', 'n23', 'n134', 'n347', 'n947', 'n220', 'n349', 'n887', 'n716', 'p197',
'p473', 'n313', 'n847', 'p31', 'p600', 'p476', 'n188', 'n322', 'n265', 'p579', 'n494', 'n449', 'n984',
```

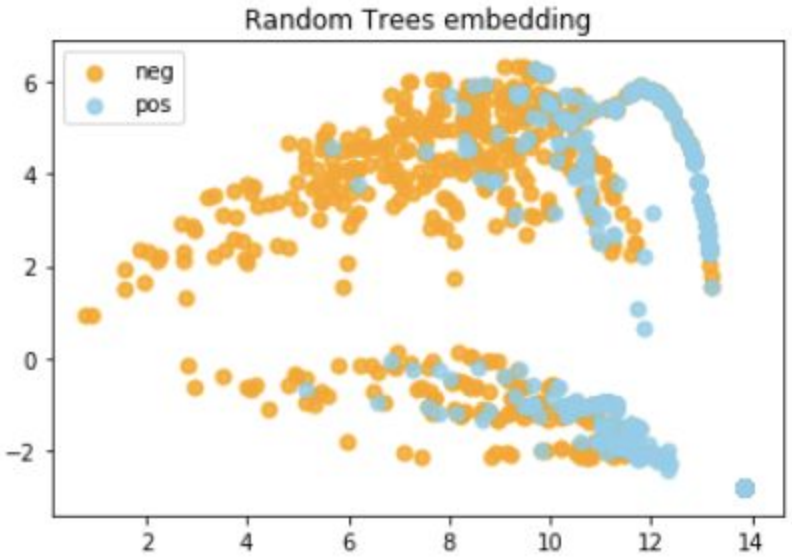
```
['p412', 'p954', 'p225', 'p62', 'p825', 'p715', 'p971', 'p38', 'p458', 'p989', 'n604', 'n927', 'p947',
 'p668', 'p147', 'n87', 'p32', 'p273', 'p346', 'p86', 'n558', 'p298', 'p212', 'p487', 'p512', 'n92',
'p607', 'p649', 'p290', 'n261', 'p22', 'n768', 'n444', 'p816', 'n541', 'p26', 'p452', 'n580', 'n773',
'n80', 'n957', 'n527', 'n538', 'n389', 'n662', 'n295', 'n72', 'p945', 'p538', 'n136', 'n101', 'n903',
 'n642', 'n71', 'n626', 'p784', 'n177', 'p933', 'n441', 'n16', 'n6', 'n129', 'n330', 'n27', 'n368',
'n416', 'n593', 'n438', 'n711', 'n170', 'n190', 'n18', 'p493', 'p573', 'p694', 'p50', 'p893', 'p349',
```

```
['n401', 'n999', 'n945', 'p612', 'n760', 'p647', 'n249', 'n457', 'p832', 'p496', 'p78', 'n764',
'p433', 'n52', 'p341', 'n243', 'n731', 'n819', 'p168', 'p752', 'p66', 'p130', 'p326', 'p543', 'n195',
'p17', 'n302', 'p589', 'p727', 'n590', 'p140', 'p365', 'p922', 'n13', 'n921', 'n294', 'n566', 'n247',
'n673', 'n738', 'n143', 'p575', 'n89', 'n230', 'n407', 'n343', 'p238', 'n397', 'n204', 'p495', 'p779',
'p134', 'p858', 'p531', 'n520', 'n854', 'n914', 'p210', 'p160', 'p566', 'p13', 'n364', 'n394', 'n617',
```

3. Ensemble learning: RandomTreesEmbedding

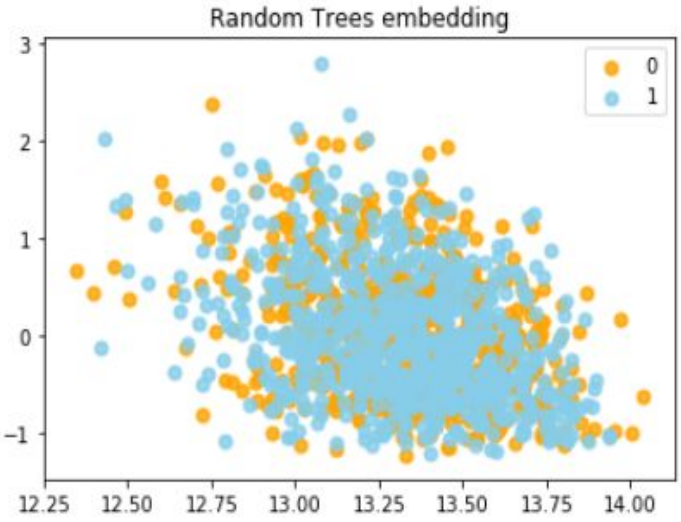
Approach I :

We apply an ensemble based method of RandomTreesEmbedding, with n_estimators = 200 and max_depth = 5, to the dataset with features in LinearLassoIndex. The result shows an overlapping of the two review-types



Approach II:

We apply an ensemble based method of RandomTreesEmbedding, with n_estimators = 200 and max_depth = 5, to the dataset with features in TruncatedSVD(256).The result shows more overlapping of the two review-types from the previous :



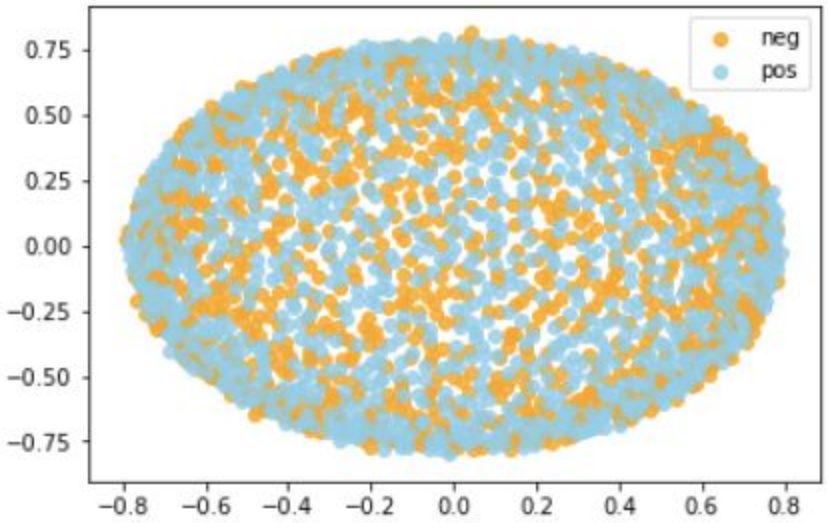
4. **Manifold Learning:** Manifold learning seeks a low-dimensional representation that preserves certain relationships within the the high-dimensional data.

4.1. MDS

Approach I:

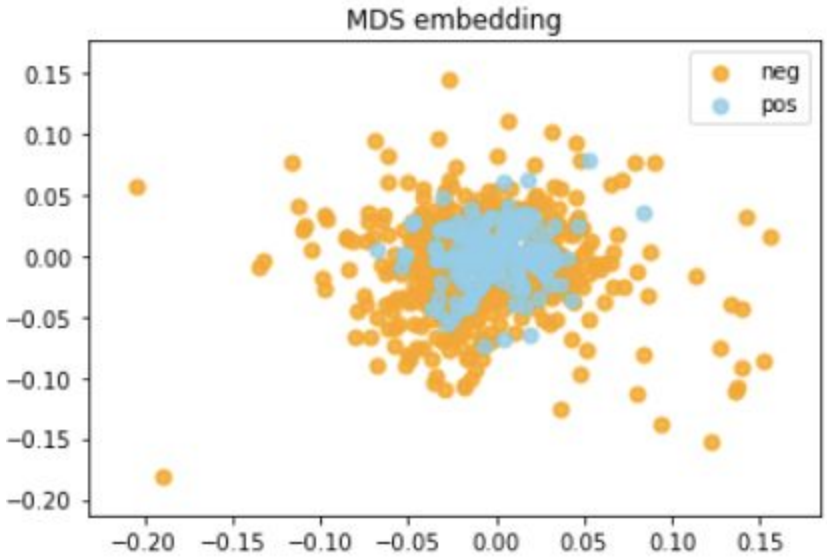
Using cosine similarity to compute pairwise distances between each review. Then we apply manifold.MDS with 2 components to plot the result.

distance = 1 - cosine_similarity(tf-idf_matrix)



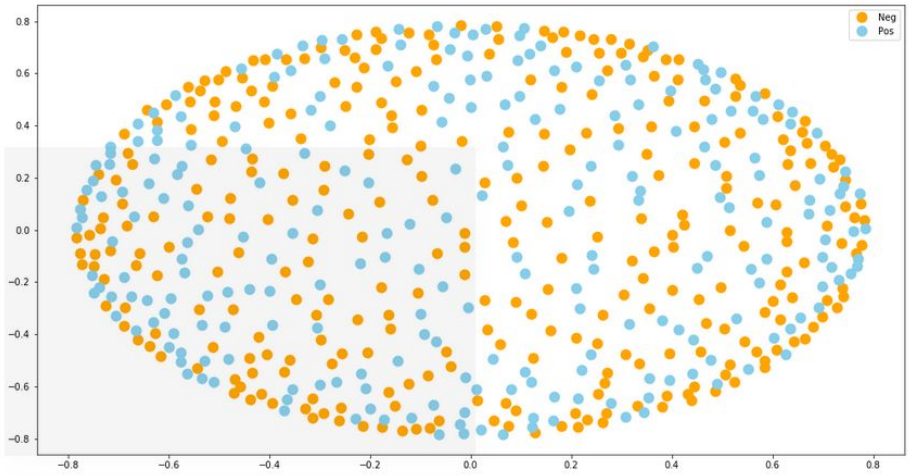
Approach II:

We apply MDS with n_components = 2, n_init = 1, and max_iter = 100 to the data with features in LinearLassoIndex. The result shows the overlap of two types of the reviews, however, positive results group closer to each other, while negative reviews are more spreading. **This is one of the best results we got.**



Approach III:

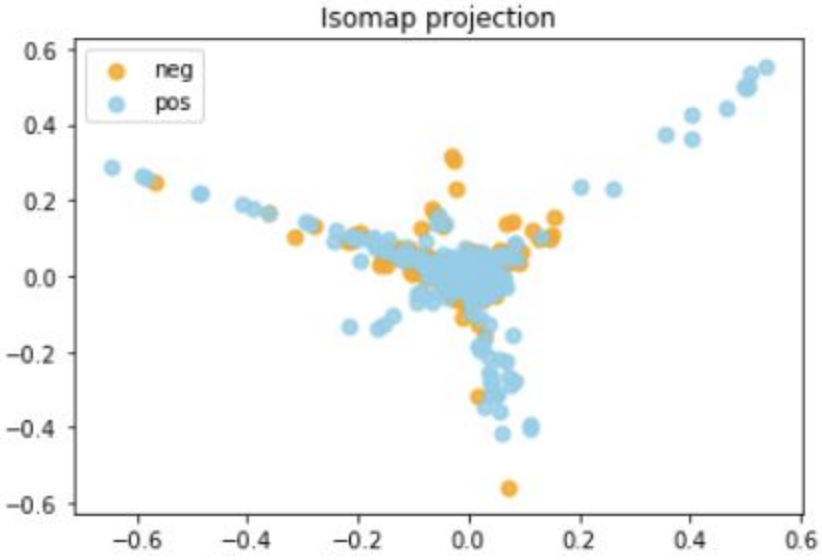
We apply MDS with n_components = 2, n_init = 1, and max_iter = 100 to the data with features in TruncatedSVD(n_components = 256). The result shows the overlap of two types of the reviews, however, the number of features have reduced.



4.2. Isomap

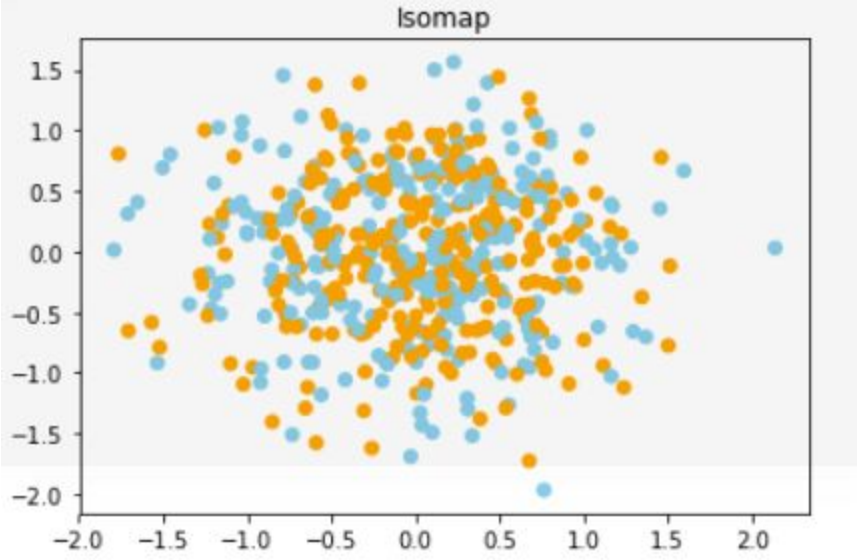
Approach I:

We apply Isomap projection with n_neighbors = 100 and n_components = 2 to the data matrix with features in LogRegLassoIndex. The result shows that data points form a 3-edge star shape. Negative and positive reviews are overlapping.



Approach II:

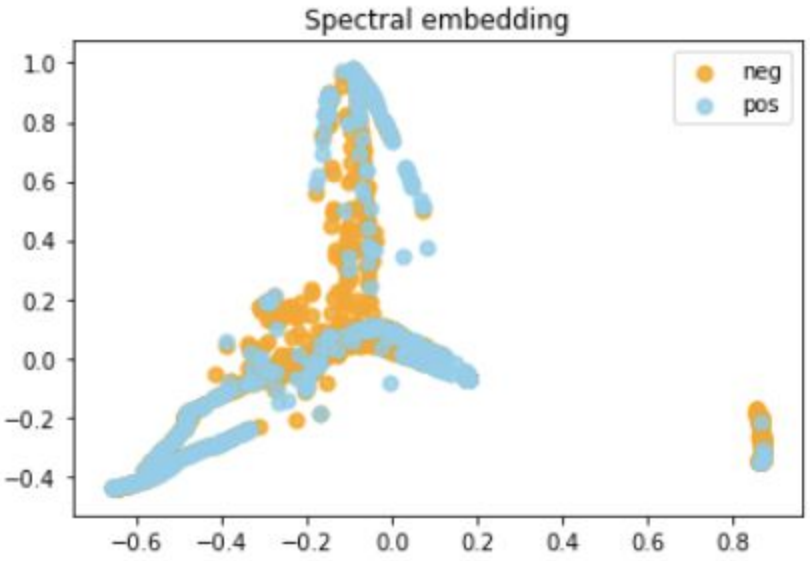
We apply Isomap projection with n_neighbors = 100 and n_components = 2 to the data matrix with features in TruncatedSVD(n_components=256). The result shows that data points form a 3-edge star shape. Negative and positive reviews are overlapping.



4.3. Spectral Embedding

Approach I:

We apply SpectralEmbedding with eigen_solver = ‘arpack’ to the data matrix with features in LinearLassoIndex. The result shows a small cluster at the bottom-right of the plot and a large cluster in the form of a 3-edge star, both containing negative and positive reviews.



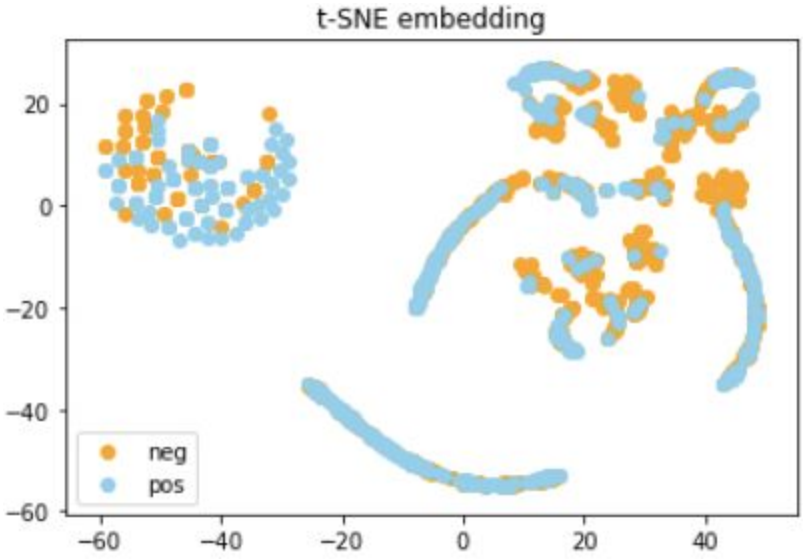
Approach II:

4.4. Locally Linear Embedding

4.5. t-SNE

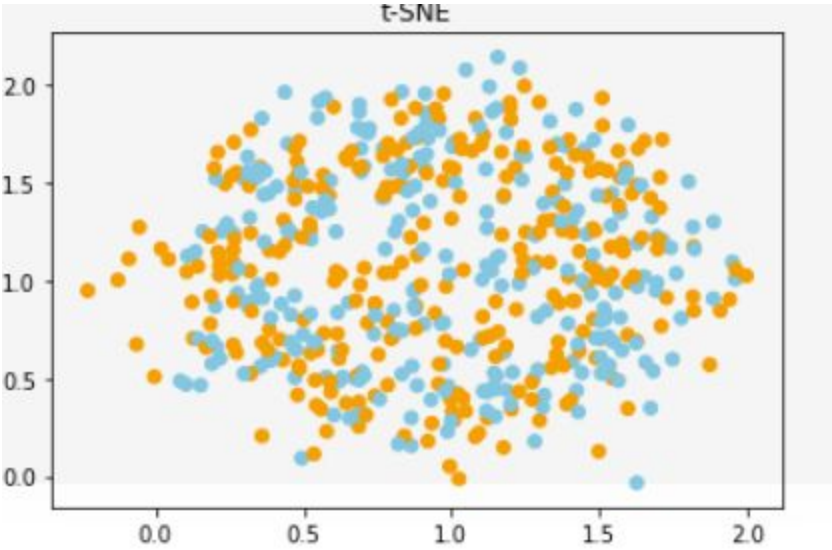
Approach I:

We apply t-SNE with init = 'pca' to the data matrix with features in LinearLassoIndex. The result looks like there are multiple clusters in the data. We still cannot separate reviews using 2D plot.



Approach II:

We apply t-SNE with to the data matrix with features in TruncatedSVD(256). We still cannot separate reviews using 2D plot.



Summary

In this case study we learned how to analyze text data. How text data is used with the existing machine learning algorithms like classification and clustering. TfidfVectorizer is used to transform the raw text documents into a matrix of TF-IDF features. Once we extract these we can further classify the documents into different classes. For Classification we explored LinearSVC, KNN and applied GridSearchCV to further tune the model using cross validation across parameters. In order to get the right 2d plot, four main methods, which are feature selection, clustering, ensemble learning and manifold learning, were applied to the data. Unfortunately, all plots cannot provide an obvious separation between negative and positive reviews. However, we observe that most positive reviews seems denser to each other than negative reviews. We think plotting in higher dimension with interactive technique or applying other classification methods might provide better visualization.