# Textual and Sentiment Analysis of Movie Reviews

DS-501 Case Study Team#6- Praneeth Nooli, Yousef Fadila & Rahul Ghadge

# **Introduction**

# Watching movies is a very common pastime all over the world. It is a favorite form of relaxation and entertainment. Before many of us choose a movie, however, we often seek out the opinions of others who have already seen it. There was a time when this process was rather limited- local newspapers would print a movie review in which a film critic described and ranked a given film. But today, Internet allows us to read movie reviews from critics and non-critics alike. Websites such as IMDB or Rotten Tomatoes offer entire collections of reviews for current and past films. Countless more movie “reviews” are readily available in much less structured forms: social media platforms are full of the thoughts and opinions of movie watchers.

# In fact, these informal “reviews” are the method by which many now shape their decisions to watch a movie. When headed out to the theater, many will check what their Facebook friends think of a new movie, as opposed to seeking out an official critic’s review. Thus it is our comprehension of this unstructured text that we use to form our decisions. As humans, understanding whether text in a passage is positive or negative is a skill we spend a lifetime developing. This process becomes more interesting on a large-scale, where the unstructured film reviews, authored on social media, are available to movie production companies and other industry players. Understanding the sentiment behind the text can help these corporations understand the popularity status of their film, as well as help shape their marketing strategies, and future directions. However the volume of these opinions is massive: it would be implausible for movie studios to hire employees to simply read and judge movie opinions. On the other hand, a machine learning method, which can process and reliably extract the sentiment of unstructured movie-related text, could be hugely beneficial. This will be the focus of this report: using machine learning techniques to understand the sentiment of unstructured film text. We will approach this problem through four objectives.

# **Objective 1:** Preliminary sentiment analysis on movie reviews. We will acquire a data set, appropriate for considering this question, and begin to explore it analytically.

# **Objective 2:** Explore the sci-kit learn TfidfVectorizer class. One of the most useful tools for text analysis is the TfidfVectorizer class available in Python’s sci-kit learn package. We will define the ideas behind this class, and explore some of the relevant parameters.

# **Objective 3:** Machine learning algorithms. Having completed objectives 1 and 2, we will try to use machine learning techniques to classify movie reviews as “positive” or “negative.” We will focus on two classification techniques: K-Nearest Neighbors and a linear Support Vector Classifier.

# **Objective 4:** Finding the right plot. If one could determine a two-dimensional plot in which positive and negative reviews are separated, this would make classification rather straightforward and offer a reasonable method of predicting the sentiment of new text. We attempt to determine such a plot.

# **Preliminary sentiment analysis on movie reviews**

**Data**: This study utilizes the movie reviews of the v2.0 Polarity Dataset, available at <http://www.cs.cornell.edu/people/pabo/movie-review-data>. This dataset consists of 2000 .txt files, each containing a movie review. Out of which 1000 are “positive” and 1000 “negative” reviews.

The following three problems are presented as Exercise 2 in the “Working with text Data” tutorial of the scikit-learn documentation [http://scikit learn.org/stable/tutorial/text\_analytics/working\_with\_

text\_data.html]:

1. Write a text classification pipeline to classify movie reviews as either positive or negative
2. Determine a good set of parameters using grid search
3. Evaluate the performance on a held-out test set.

In order to solve these three problems, we modify the solution provided in the scikit-learn documentation so that it can be run in an iPython notebook. First, we added a code which downloaded the data directly from the source website, and stored it in a local directory. Then we randomly split the 2,000 movie reviews into two groups, 75% for training (1500 records) our pipeline, and 25% for testing (500 records). Using the TfidfVectorizer, we built a vectorizer-classifier pipeline that filtered out tokens that were too rare or frequent, and fit a linear support vector classifier with relatively high penalty.

Using grid search, we determined a set of tokens to consider within our reviews: words (1-grams) or words and pairs of words (1-grams and 2-grams). We combined this grid search with our classification pipeline on the *training* data to perform grid search cross-validation, finding the following (mean cross-validation) scores:

**Grid Search CV scores**

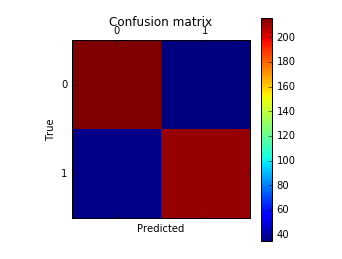
|  |  |
| --- | --- |
| **ngram\_range** | **score** |
| (1 , 1) | 0.83 |
| (1 , 2) | 0.84 |

While ngram\_range will be explained fully in the following objective, what these grid scores indicate is that on the *training* data, the linear SVC pipeline performs more accurately when it considers both words and pairs 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. We obtained the following classification report:

**Classification Report**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **Precision** | **Recall** | **f1-score** | **Support** |
| Negative | 0.85 | 0.86 | 0.86 | 251 |
| Positive | 0.86 | 0.85 | 0.85 | 249 |

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 can further evaluate performance using the confusion matrix, which can be plotted as follows:



Therefore the number of false negatives and false positives are both small compared to the number of true positives and negatives. Combined with the classification report given above, this indicates that this model performed quite well on our test data set.

# **Explore the sci-kit learn TfidfVectorizer class**

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

Term frequency–inverse document frequency, also known as TF-IDF, is a statistic that measures how important a word is to a document in a particular collection of documents[[1]](#footnote-1). It is the product of two individual term statistics: term frequency and inverse document frequency. The *term frequency* of a term *t* in a particular document *d* simply measures the frequency of *t* in *d*. While there are many ways to define this frequency, the simplest is the raw frequency: the number of occurrences of *t* in *d* divided by the total number of words in *d*. The *inverse document frequency* tries to estimate the information content of a given word: common words (such as “stop words”) will appear in most documents, and thus do 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. 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*:

The TF-IDF is a measure of importance of a term *t* to a document *d*, among a collection of documents *D*. It is the product of the term frequency of *t* in *d* and the inverse document frequency of *t* in *D*. A word will have a high TF-IDF value if it’s mentioned very frequently in that specific document, but not in a large number of documents in the collection. A word which occurs in a large number of documents will have a low IDF value, thus decreasing the TF-IDF statistic of that word in any document: this serves to filter-out common words.

*Run the TfidfVectorizer 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. 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.

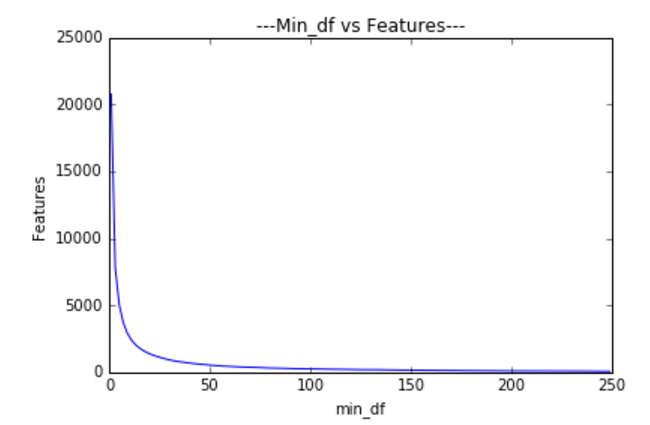
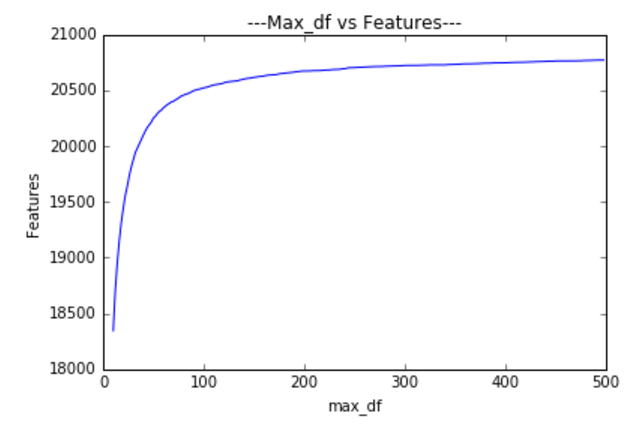
*Explore the min\_df and max\_df parameters of TfidfVectorizer . What do they mean? How do they change the features you get?*

When running the TfidfVectorizer class on a collection of documents, we first build a vocabulary of terms contained in all documents, for which we will compute the TF-IDF statistic. The parameters min\_df and max\_df are used in determining which terms to include in this vocabulary.

We ignore all terms that have a document frequency less than min\_df: therefore *all* terms in our vocabulary have a frequency of *at least* min\_df in all documents in our collection. On the other hand, we also filter out all terms with a document frequency greater than max\_df: therefore all terms in our vocabulary have a frequency *at* most max\_df in every document. The max\_df parameter is often used to filter out stop words.

Using our collection of movie review documents, we ran the TfidfVectorizer class for a range of min\_df and max\_df values ranging between 0 and 1, and computed the number of features retained in the vocabulary. Plots of the number of features versus these parameter values are shown below

**Min\_df vs Features of TfidfVectorizer Max\_df vs Features of TfidfVectorizer**

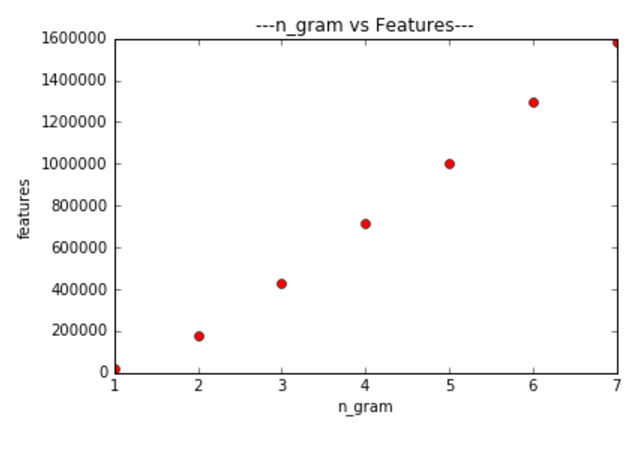
 

These plots exhibit a clear relationship between the number of features in our vocabulary and the values of min\_df and max\_df. The size of our vocabulary is inversely related to the parameter min­\_df: as min\_df increases, the number of words in our vocabulary decreases. Moreover, this decay appears to be exponential. On the other hand, the size of the vocabulary is related to the parameter max\_df: as max\_df increases, the size of the vocabulary increases as well. This growth seems to be hyperbolic in nature: the vocabulary size grows rapidly with small values of max\_df, and then increases more slowly as max\_df grows larger. This directly affects the features considered: a non-zero value of min\_df immediately eliminates from consideration all terms which don’t occur in *all* documents of our collection. On the other hand, choosing a reasonable value of max\_df eliminates from consideration all words which occur a large number of times in all documents: thus a good choice of max\_df can be used to filter-out stop words.

*Explore the ngram\_range parameter of TfidfVectorizer . What does it mean? How does it change the features you get?*

Another relevant parameter of the TfidfVectorizer class is the ngram\_range. In the world of of textual analysis, an n-gram is a sequence of “n” consecutive words in a document. Thus the 1-grams of a document is the collection of words, the 2-grams are the collections of pairs of consecutive words, and so on. The ngram\_range parameter determines the sets of n-grams that will be included in the vocabulary of the TfidfVectorizer Given as a tuple, ngram\_range = (min\_n, max\_n), the vocabulary for the TfidfVectorizer class is built from *all* n-grams (for min\_n ≤ n ≤ max\_n) of each document in our collection. This fundamentally changes the nature of the features considered in our vocabulary. Choosing ngram\_range = (1,1) creates a vocabulary of all *words* in the document. On the other hand, choosing ngram\_range = (1 , 2), the vocabulary is now constructed of all words and all consecutive pairs of words in the document. In this case we must use care, as these features are no longer comparable in a sense: some features represent words while others represent pairs of words. What is more, this range also affects the *number* of parameters in our vocabulary. Using ngram ranges of the form (1, 1), (1, 2), (1, 3) … (1, 10), we ran the TfidfVectorizer class on our collection of movie reviews, and counted the number of features in the vocabulary. A plot of the number of features versus ngram\_range tuples of the form (1, ngram) are shown in figure. 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,ngram) vs. Features of TfidVectorizer**



# **Machine learning algorithms**

In this objective, we use two machine learning algorithms, K-Nearest Neighbors and Linear Support Vector Classifier (SVC), to predict the polarity of movie reviews. We use a training and testing set with the same split as above (75% training, 25% testing), which is *different* from that used previously, to avoid data-snooping.

*Based upon Problem 2 pick some parameters for TfidfVectorizer .*

For all of the following analysis we choose min\_df = 1, max\_df = 0.95, and ngram\_range = (1, 2) as the parameters for the TfidfVectorizer class. We now use the fit-transform properties of the TfidfVectorizer class to turn our Training and Testing documents into a pair of matrices. To ensure values of these matrices correspond to the same text tokens, we use the same TF-ID-weighted class derived from the *training* documents, to transform both the training and testing documents. In this manner we compute “Xtrain” i.e. a TF-ID-weighted document-term matrix corresponding to the training documents and “Xtest” i.e. the same matrix, now corresponding to the testing documents. We may now use these two weighted document-term matrices to train and evaluate machine learning algorithms.

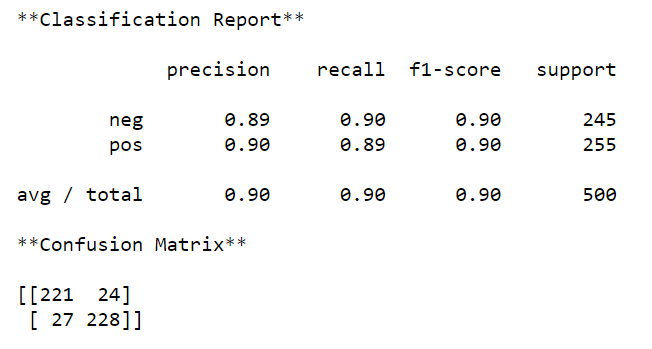
*Examine two classifiers provided by scikit-learn*.

1. **Linear Support Vector Classifier**

First we used our training matrix, Xtrain, to develop a set of linear support vector classifiers. We used a number of different values for the “penalty” parameter ({0.01, 0.1, 0.5, 1, 10, 100}) and “tolerance” parameter ({0.0001, 0.01, 1, 10}) and computed the testing accuracy of each model which is explained in the below table.

|  |  |  |
| --- | --- | --- |
| C | Tolerance | Mean\_test\_score |
| 0.01 | 0.0001 | 0.61 |
| 0.01 | 0.01 | 0.61 |
| 0.01 | 1 | 0.51 |
| 0.01 | 10 | 0.59 |
| 0.1 | 0.0001 | 0.81 |
| 0.1 | 0.01 | 0.81 |
| 0.1 | 1 | 0.81 |
| 0.1 | 10 | 0.55 |
| 0.5 | 0.0001 | 0.83 |
| 1 | 0.0001 | 0.83 |
| 10 | 0.0001 | 0.83 |
| 100 | 0.0001 | 0.84 |

Conclusion: - Based on the mean cross validation scores from the above table we observe that SVC works well when the parameters are set to C=100 and Tolerance = 0.0001 (default value). The classification report and confusion matrix for this model are shown below.



From the above mentioned confusion matrix, it clearly demonstrates a small number of false positives and false negatives in comparison to a much larger amount of true positives and negatives. Thus, in addition to the testing accuracy of about 85% shown above, this indicates that a Linear Support Vector classifier does a good job of predicting the polarity of movie reviews (based on a TF-ID matrix)

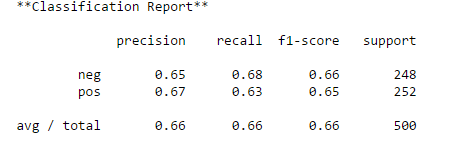
1. **K-Nearest Neighbors**.

Next we used our Xtrain, to develop a set of K-NN classifiers. We used a number of different values for “neighbors” parameter, k ({1, 2, 3, 4, 5, 6, 7}) along with the Power Parameter P for the Minkowski metric (in which P =1 corresponds to manhattan distance and p =2 corresponds to euclidian distance) and computed the mean cross validation scores of each model which is shown in the below table.

|  |  |  |
| --- | --- | --- |
| K | P | Mean\_test\_score |
| 1 | 1 | 0.50 |
| 1 | 2 | 0.66 |
| 2 | 1 | 0.50 |
| 2 | 2 | 0.65 |
| 3 | 1 | 0.51 |
| 3 | 2 | 0.67 |
| 4 | 1 | 0.52 |
| 4 | 2 | 0.67 |
| 5 | 1 | 0.50 |
| 5 | 2 | 0.65 |
| 6 | 1 | 0.52 |
| 6 | 2 | 0.67 |
| 7 | 1 | 0.52 |
| 7 | 2 | 0.66 |

Judging by the above table, the model with k = 4 and p =2 (Euclidian) performed the best.

We can evaluate the performance of this model using a confusion matrix and classification report. The confusion matrix for this model is given with values and classification report is given by



From the above mentioned confusion matrix, it clearly demonstrates a small number of false positives and false negatives relative to the amount of true positives and negatives. Thus, in addition to the testing accuracy of about 66% shown above, this indicates that a KNN Vector classifier does a decent job of predicting the polarity of movie reviews.

The result from LinearSVC was way better than KNNeighborsClassifier a good LinearSVC precision imply that there is a good line separate the reviews, it could be that many instances are close to that hypothetical line from both sides, so when running the KNNeighborsClassifier, instances near the line will have close neighbors from instances of the other hypothetical line side. This would cause a high error rate in KNNeighborsClassifier that can't fit well to such problem

*For a particular choice of parameters and classifier, look at 2 examples where the prediction was incorrect.*

1. Finding False Positive (Actual Value- Negative, Predicted Value-Positive)

We could conjecture that the model falsely classify this review as positive because it includes some positive structures. even the 1st sentence appears very positive like- *“i read the new yorker magazine and i enjoy some of their really in-depth articles about some incident frequently i get the feeling that the article sounded exciting for even so good an actor as plummer to play him convincingly have been enthralling”*

We have used with (1,2) n grams, so positive terms of two words like "so good", "i enjoy", "sounded exciting" and single words "exciting", "enthralling", "good" which could occur more frequent in true positive reviews rather than the negative ones contribute to misclassify this review as Positive.

1. Finding False Negative (Actual Value- Positive, Predicted Value- Negative)

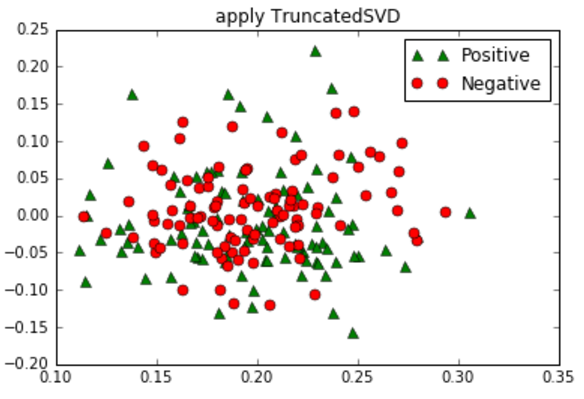
Consider the following review “When king is screwed out of his title by a corrupt promoter, gordie and sean take it upon themselves to find their fallen hero and restore his glory. The hook of the movie is that gordie and sean are just too stupid to realize that. none casting complaint however : rose mcgowan as a sexy dancer ? ”We could conjecture that the model falsely classify this review as negative because it includes good amount of (1,2) n grams of negative terms like "silly", "corrupt promoter" , "screwed out", "too stupid", "complaint ","silly factor" etc. These terms may be more frequent in true negative reviews rather than the positive ones so they contribute to that the model finds it similar to negative reviews rather than positive and hence misclassified.

Irrespective of the cause, there will likely always be some level of error in our machine learning algorithm. In addition to tweaking features to improve the model, we pose the following question for machine learning in general: **Are false positives more expensive, or are false negatives?** For example, if a negative movie review is incorrectly classified as positive is this more acceptable than if a positive movie review is classified as negative? It is difficult to tell – it really depends on the case. For example, a false negative in a life or death situation is extremely expensive (not running away from a hungry lion) while a false positive is cheap (running away from a rock you *thought* was a hungry lion). Overall, defining criteria for success is extremely important.

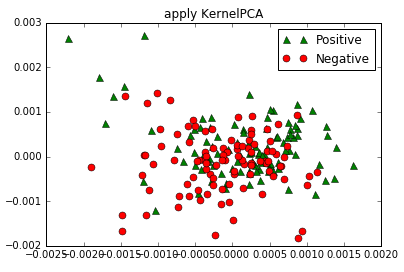
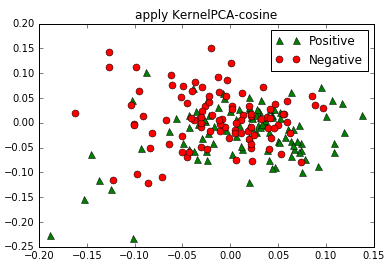
# **Find the right plot**

If one could determine a two-dimensional plot in which positive and negative reviews are separated, this would make classification and prediction rather intuitive. In this objective, we attempt to find such a plot.

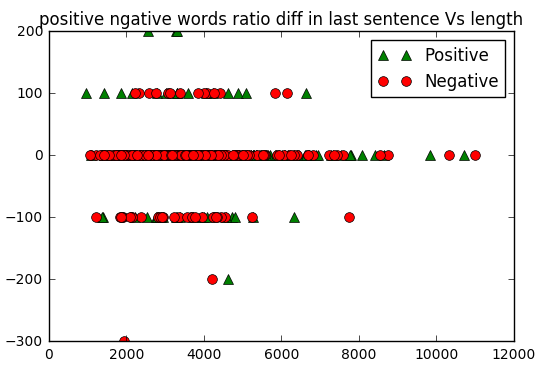
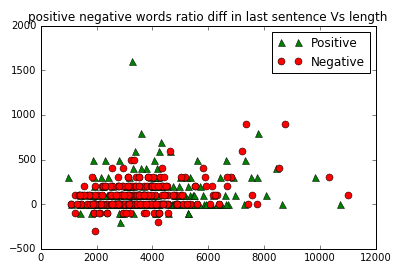
This problem seems inherently difficult: considering only 1-grams in the previous objectives gives a large number of features. Transforming these thousands of features into just two numbers seems daunting at best. So in order to check how PCA works on this data, we started applying TruncatedSVD (equivalent to PCA as PCA does not support sparse data) with its default linear kernel by reducing the number of components(features) whose result can be seen from the below figure



So from the above plot we can figure out that TruncatedSVD did not perform well on our data. So we tried to change the kernel of PCA to Polynomial and cosine, but the results obtained were similar to the above one and thus not able to separate the positive and negative reviews.

Hence we pose the following conjecture: **Can the polarity (positive/negative tendencies) of a given text be determined based on the *structure* of the text itself?** For example most of the Positive reviews contains words such as good, awesome, excellent, pleasant, surprising etc. On the flip side, most of the negative reviews contains words such as horrible, unpleasant, awful, disgusting, disappointing, boring, uninteresting etc. So we created an array of positive patterns as well as negative patterns. By using these pattern array in regular expression, we calculated ratio of positive words and negative words in each movie review. Then we took difference of two ratios which would represent the overall sentiment of a review. We plotted this against length of a movie review.



**Sentiment ratio for all text of a review Sentiment ratio for first and last 90 characters of a review.**

Later in order to improve the results we decided to calculate positive and negative ratio only for first and last 90 characters. This is because mostly people would right reviews with actual sentiment at the beginning or in conclusion at the end. Again we plotted the newly calculated ratio difference against the length of the review. Here we can see different clusters are formed but all clusters are mix of both positive and negative reviews so we need to again identify more features which would help in clearly distinguishing positive and negative review in each of those clusters for which we may have some common feature or different set features per cluster.

Another thing that we tried was to check if more occurrence of WH question words, question mark, words like *not, doesn’t, didn’t* etc. help to classify a review as negative. But it was observed that even in that case these words are present in both kind of reviews especially as many movie reviews have the movie plot explained.

# **Business Case**

A machine learning method which can process and reliably extract the polarity of unstructured movie-related text could be beneficial. Specifically, it can indicate to production companies’ specific areas, ages, genders, and demographics in which their film is performing well (and performing poorly).

As a result, such data analysis can help to shape business decisions. Understanding demographic popularity can assist in advertising decisions: to which gender should more marketing be targeted? Understanding the age groups for which a movie performs well can help determine future re-release of a film: if a film performs well in younger age groups, perhaps it is beneficial to release it on streaming platforms soon after its theater run. On the other hand, production studios can collect (and analyze) similar data for the movies of *other* studios. Analyzing demographic performance of similar movies is also beneficial. For example, suppose studio B releases a superhero movie, and studio A is planning to release a superhero movie in the near future. Understanding demographic performance of Studio B’s films can shape Studio A’s decisions: are there changes (to the script, casting etc.) that can be made during production to boost the films popularity amongst demographics which did not enjoy Studio B’s movie? An understanding of demographic performance can be gained, at least in part, via comprehension of the unstructured movie reviews posted on social media platforms.

# **Conclusions**

In this report, we studied the polarity (positive/negative) of unstructured movie reviews, with a particular interest in prediction via machine learning algorithms. Using a dataset of 2,000 movie reviews, we first performed preliminary sentiment analysis via a text classification pipeline. Parameters were chosen via grid search, and performance was evaluated using a validation set. Next we explored the scikit-learn TfidfVectorizer class. Understanding of the relevant statistics and parameters then allowed us to train two classification models: K-NN and linear SVC. Parameters were chosen via cross-validation, and the models were evaluated with a validation set. The linear SVC performed quite will on the testing set, while K-NN performed poorly, particularly on the test set of negative reviews.

Finally, we tried to determine a 2-dimensional plot which separated the negative and positive reviews. We conjectured that the polarity of a review could be determined based on the structure of the test, and thus computed a new set of structure-based predictors on our movie reviews. Using these predictors, after a number of unsuccessful visualization attempts, we tried to represent the positive and negative reviews. While this did not allow us to successfully separate the reviews. It does not disprove our conjecture, but (extensive) further analysis is required to prove the conjecture in the affirmative. In the end, we did not find anything *surprising* in the data (other than just how difficult textual analysis can be!) In particular, principled classification methods can perform overwhelmingly poorly. Much further (and computationally-expensive) analysis could certainly reveal surprising trends in this data set.

1. “TF-IDF” <http://en.wikipedia.org/wiki/Tf%E2%80%93idf>  [↑](#footnote-ref-1)