The Reviews Are In

Textual and Sentiment Analysis of Movie Reviews

Qiuyi (Helen) Hong, Yanhong (Haley) Huang, Tom Meagher, and Tyler Reese

INTRODUCTION– In the modern age of technology, the Internet puts a seemingly infinite collection of information at our fingertips. Facts and opinions are produced at the "speed of thought" and consumed by Internet users in massive quantities and at extraordinary rates. What is more, as individuals we may harness this stream of information in order to make informed decisions.

The viewing of films and movies is a popular pastime the world over: it is a favorite form of relaxation and entertainment which spans generations, genders, and geography. Before many of us choose a movie, however, we often seek out the opinions of others who have already done so. 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. Today, the Internet allows us to read movie reviews from critics and non-critics alike. Websites such as IMDB or RottenTomatoes 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 film decisions. 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 inform 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 latent 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 TfidVectorizer class. One of the most useful tools for text analysis is the TfidVectorizer 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.

OBJECTIVE 1: 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. A thousand of these reviews are classified as "positive" and 1,000 as "negative."

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 in can be run in an iPython notebook. First, we added a script which downloaded the data directly from the source website, and stored it in a local directory. Then we had to modify a few of the package imports for compatibility with scikit-learn. From there, we proceeded as in that solution.

We first randomly split the 2,000 movie reviews into two groups, 75% for training our pipeline, and 25% for testing. Using the TfidVectorizer class (which will be explored further in objective 2), 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. (We use the values of min_df and max_df that were given in the documentation which we are modifying.)

Using grid search, we determined a set of tokens to consider within our documents (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:

Table 1.1: Grid Search CV scores

ngram_range	score
(1 , 1)	0.82533
(1 , 2)	0.84733

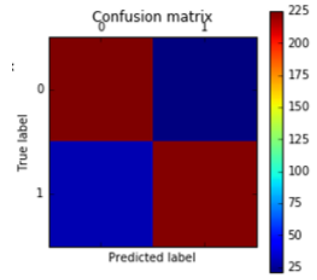
While ngram_range will be explained fully in the following objective, what these grid scores indicate is that on the *training* data, the linear SVC pipeline performs more accurately when it considers both words and pairs of words contained in our document. Using these preferred parameters determined with 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:

Table 1.2: Classification Report

Class	Precision	Recall	f1-score	Support
Negative	0.84	0.87	0.85	247
Positive	0.87	0.84	0.85	253

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, $\begin{bmatrix} 214 & 33 \\ 41 & 212 \end{bmatrix}$ which can be plotted as follows:

Figure 1.1: Confusion Matrix



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.

OBJECTIVE 2: Explore the scikit-learn TfidVectorizer 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]. 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*:

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

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 (or "stop") words.

Run the TfidVectorizer class on the training data.

The TfiDVectorizer 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 (words or strings of words determined by the n_gram input to the class, which will be described below) contained in those documents. The entries of the matrix are the TF-IDF statistic for each term in each document.

[1] "TF-IDF" http://en.wikipedia.org/wiki/Tf%E2%80%93idf

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

When running the TfidVectorizer 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 TfidVectorizer 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 in Figures 2.1 and 2.2.

Figure 2.1 Min_df vs Features of TfidVectorizer

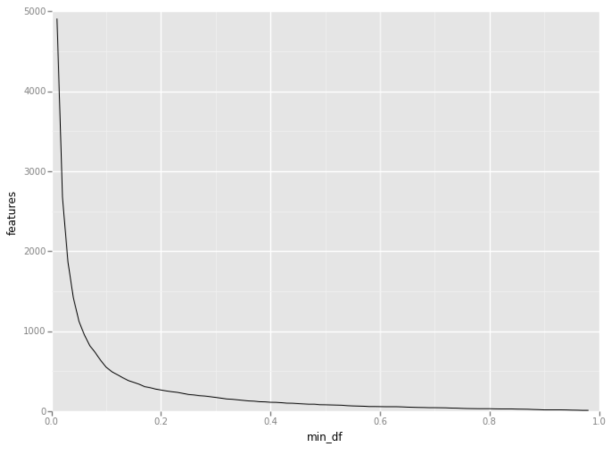
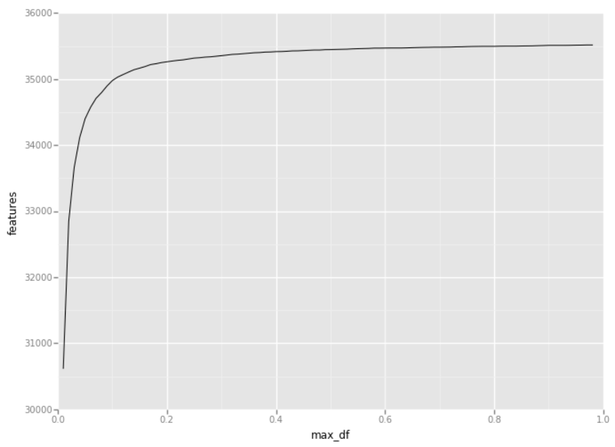


Figure 2.2 Max_df vs Features of TfidVectorizer



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 TfidVectorizer. What does it mean? How does it change the features you get?

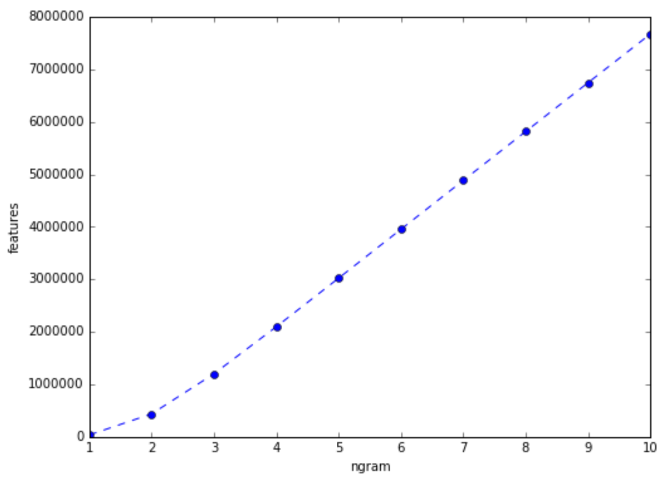


Figure 2.3: ngram_range = (1,ngram) vs. Features of TfidVectorizer

Another relevant parameter of the TfidVectorizer class is the ngram_range. In the realm 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 TfidVectorizer. Given as a tuple, ngram_range = (min_n, max_n), the vocabulary for the TfidVectorizer 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 TfidVectorizer 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 2.3. 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.

BRIDGE TO OBJECTIVE 3.

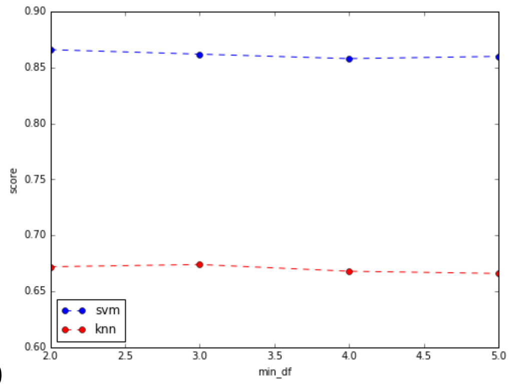
In objective three, we will fit two machine learning algorithms, K-Nearest Neighbors and linear Support Vector Classifier, on our set of movie reviews utilizing the TfidVectorizer class. Therefore before moving on, we consider how these three TfidVectorizer parameters- min_df, max_df and ngram_range- affect the performance of those algorithms. We consider the following ranges of each parameter:

Table 2.1: Ranges of TfidVectorizer Parameter values

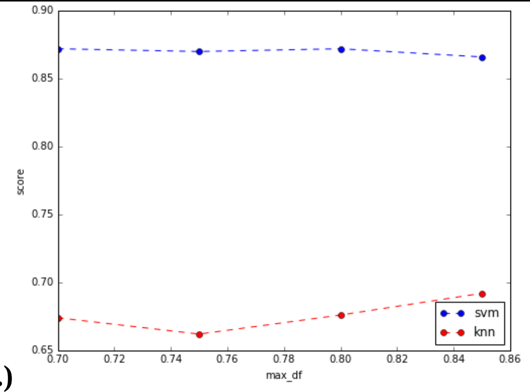
TfidVectorizer Parameter	Range of Values
Min_df	{1,3,5,7}
Max_df	{0.4,0.5,0.6,0.7}
Ngram_range	{(1,1) , (1,2)}

Holding two of these parameters at their default value, we run the TfidVectorizer class for each value of the third parameter in the range given above. By using both the fit and transform attributes of the TfidVectorizer class, we used the resulting transformed (training data) to train both KNN and Linear SVC classification models. We then compute the *testing* accuracy for each such model, in the form of an accuracy score. This allows us to compare the values of our parameters to the accuracy of the resulting models, as can be seen in Figure 2.4.

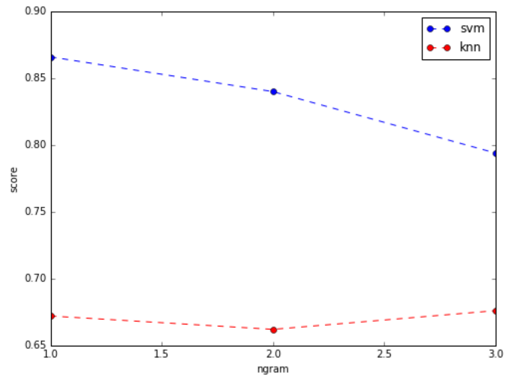
Figure 2.4 Impact of Min_df, max_df, and ngram_range on accuracy of KNN and LinearSVC



(a.)



(b.)



(c.)

(a). Testing Accuracy vs. min_df
(b). Testing Accuracy vs. max_df
(c). Testing Accuracy vs. ngram_range,
ngram_range = (1 , ngram)

Note that we use a testing error to choose good parameters for Objective 3. In order to avoid data-snooping, we use a re-partitioned set of training and testing data in Objective 3. By evaluating model performance versus the Tfid parameters, it allows us to determine appropriate values to choose in the proceeding section. In particular, we see that both K-NN and linear SVC perform (comparatively) well when min_df = 3, max_df = 0.85, and ngram_range = (1,1). Therefore, we chose (and fixed) these parameter values throughout Objective 3.

OBJECTIVE 3: 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.

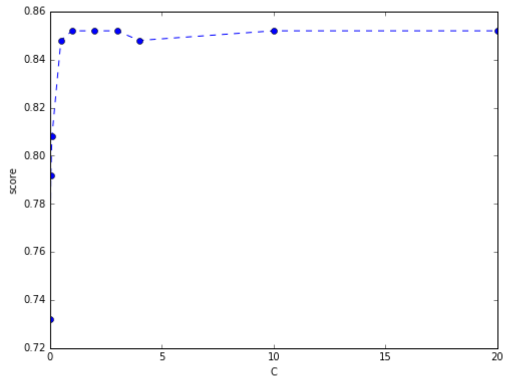
Based upon Problem 2 pick some parameters for TfidVectorizer.

As described in the preceding "Bridge to Objective 3," for all of the following analysis we choose min_df = 3, max_df = 0.85, and ngram_range = (1,1) as the parameters for the TfidVectorizer class. We now use the fit-transform properties of the TfidVectorizer 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, that derived from the *training* documents, to transform both the training and testing documents. In this manner we compute "Xtrain" (a TF-ID-weighted document-term matrix corresponding to the training documents) and "Xtest" (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.

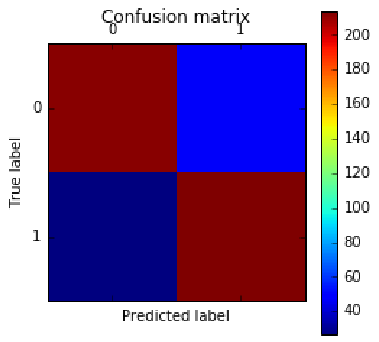
1. Linear Support Vector Classifier. First we used our training matrix, Xtrain, to develop a set of linear support vector classifiers. We used a number of different values for the "penalty" parameter ({0.01, 0.05, 0.1, 0.5, 1, 2, 3, 4, 10, 20}) and computed the testing accuracy of each model. A plot of the testing accuracy versus the penalty (C) is shown below in Figure 3.1.

Figure 3.1: Testing Accuracy vs. Penalty, Linear SVC



Judging by this plot, the model with penalty C = 10 performed the best on our testing matrix, Xtest. We can evaluate the performance of this model using a confusion matrix. The confusion matrix for this model, with values $\begin{bmatrix} 212 & 47 \\ 27 & 214 \end{bmatrix}$ is given by

Figure 3.2: Confusion Matrix, Linear SVC model with C = 10



Clearly, this confusion matrix 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)

2. K-Nearest Neighbors. Next we used our TF-IDF training matrix, 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, 8, 9, 10}) and computed the testing accuracy of each model. A plot of the testing accuracy versus the number of neighbors (k) is shown below in Figure 3.3.

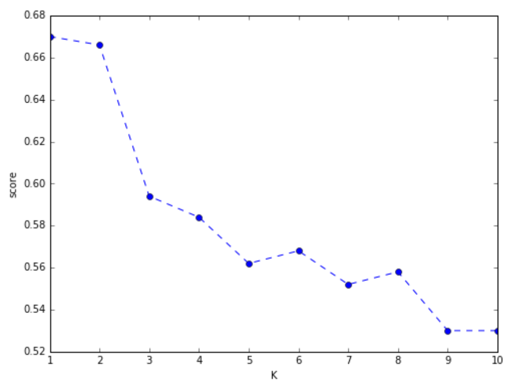
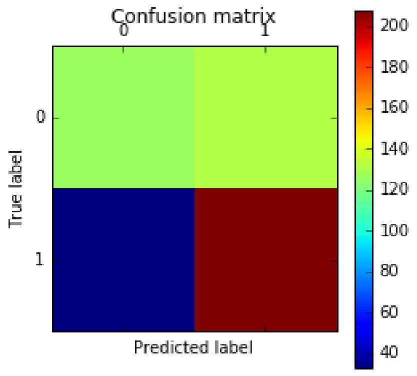


Figure 3.3: Testing Accuracy vs. k, KNN

Judging by this plot, it the model which considers only a single neighbor, k = 1 performed the best on our testing matrix, Xtest. We can evaluate the performance of this model using a confusion matrix. The confusion matrix for this model, with values $\begin{bmatrix}127 & 132\\ 33 & 208\end{bmatrix}$ is given by:

Figure 3.4: Confusion Matrix, KNN with k = 1



This confusion matrix reveals quite a lot about the performance of this K-NN model. With a testing accuracy of about 67%, one may guess that this model is doing a fairly good job. The confusion matrix tells a different story: on the testing set, this model has *more* false positives than it does true negatives: That is, of all of the reviews which are *actually* negative, the model predicted that *more than half of those* reviews were positive. Thus this model performed poorly on negative reviews (in fact, a random model which guessed "positive" or "negative" with equal probability would actually perform *better* on the negative test data). On the other hand, this model performed quite well on the positive reviews, producing only 33 false negatives compared with 208 true positives. Thus despite having a testing accuracy of 67%, this model performed well on half of the data, but poorly on the other half.

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

For all of the models considered here, the parameters of the TfidVectorizer were fixed (chosen based on the "bridge to objective 3"). Therefore the only parameters which vary among these models are those inherent to the method: the penalty C in linear SVC, and the number of neighbors k in K-NN. Based on figure 3.1, it is clear that for small penalty values, a higher penalty produces a model with a higher testing accuracy: that is, a model that "works better." This matches intuition: a model with little or no penalty is more lenient to choosing linear separators which mis-classify training points. On the other hand, by Figure 3.3 we see that as k, the number of neighbors considered in K-NN increases, the testing accuracy decreases. That is, K-NN models which "work better" have lower values for parameter k. This says that letting each test point "see" more of its nearby neighbors does not improve our prediction accuracy. This is the case if each point is "near" to a number of other points from both classes: this foreshadows the difficulties which we will encounter in Objective 4.

On the other hand, comparing Figures 3.1 and 3.3 it is clear that the linear SVC classifier is performing much better than K-NN: the *worst* SVC had a testing accuracy of 70%, whereas the *best* K-NN model had testing accuracy of about 67%. Moreover, the best K-NN model performed *worse than a random classifier* on half of its data. This could occur for any number of reasons. While K-NN is a flexible, non-linear classification method it is susceptive to overfitting. In particular 1-NN, though it had the highest testing accuracy, is especially apt to over-fit the training data. This over-fitting may be revealing itself in the form of inferior testing performance. Alternatively, there is likely not enough data for K-NN to perform well. In the presence of unlimited data, K-NN offers the best possible classification. However, very large amounts of data are needed to even approach this optimality: the given set of 1,500 training data is likely insufficient for K-NN to have high accuracy. For these reasons, a less flexible classification scheme such as linear SVC can out-perform K-NN.

In performing further experiments, the success rate of KNN did not see much improvement. Using 10-fold cross validation to evaluate our models, we used grid search to determine good parameter values for k, and the Tfid parameters (min_df, max_df, ngram_range). The *most* successful of these models had a prediction accuracy of only 70%. One reason why this accuracy does not be improve may be our feature selection methods. While we used the "Term-Frequency" "Inverse-Document_Frequency" statistics, other feature models in textual analysis include "Information Gain," "Mutual Information" and "Chi-Squared" feature selection. KNN performance may be improved by using these methods or, alternatively, using a different distance metric in the algorithm.

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

We extracted two movie reviews which were incorrectly classified by our most successful model, the linear SVC with penalty C = 10. We chose one negative and one positive review (which were predicted as positive and negative, respectfully). The text of these reviews can be found in Appendix I.

Review 1. (True Polarity: Negative. Predicted Polarity: Positive) We conjecture that this review was mis-classified for a rather simple reason. Many of the reviews in this collection feature ratings: numerical scores on a 1-10 scale. Positive reviews will feature high ratings, thus machine learning algorithms can likely recognize a correlation between a "positive" classification and the presence of high numeric values (such as 8, 9, and 10) corresponding to these ratings. This review, though negative, is for a movie titled "Session 9." Therefore the number 9 occurs a number of times within the text. While algorithms can determine a correlation between positive ratings and high numeric values, they fall short in determining the *meaning* of such values. Thus while *most* large numbers correspond to ratings, in this review the number 9 is part of the title: a distinction missed by the algorithm, leading to mis-classification.

Review 2. (True Polarity: Positive. Predicted Polarity: Negative) This review is for a movie entitled "Ghosts of the Mississippi." We conjecture that it was mis-classified because of its organization. About half of the review is dedicated to plot and character description (as opposed to the reviewer's opinions). Moreover, the movie itself involves a rather dark premise. The movie summary includes phrases like "legal battle," "racist," "Imprison," "slave-ship," "Miseries," and "serious and weepy". These words all carry a negative connotation (and may well be used as such in other reviews), but they have *no relation* to the reviewers opinions. Once again, while a machine learning algorithm may recognize word patterns, it is much more difficult to discern the *purpose* of those words, likely falling short in distinguishing the usage of negative terms in a summary setting as opposed to opinionated review.

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 bear) while a false positive is cheap (running away from a rock you *thought* was a hungry bear). Overall, defining criteria for success is extremely important.

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 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. Instead 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, perhaps (out of disgust) negative reviews always end with a short final sentence. On the flip side, perhaps positive reviews always begin with a rather long (and glowing) sentence. Do confused viewers (thus writing negative reviews) ask more questions? Do positive reviews use more words in general? The *structure* of the text itself may reveal the sentiment behind the text, without considering all of the tokens (be it words or n-grams) within the text. What is more, these structure-based predictors can be computed for *all* reviews (regardless of their word content), can be restricted to a much smaller, relevant number, and are also easy to interpret. Therefore we defined the following new set of predictors, which are computed for each review in the data set.

Table 4.1: Set of new, structure-based predictors

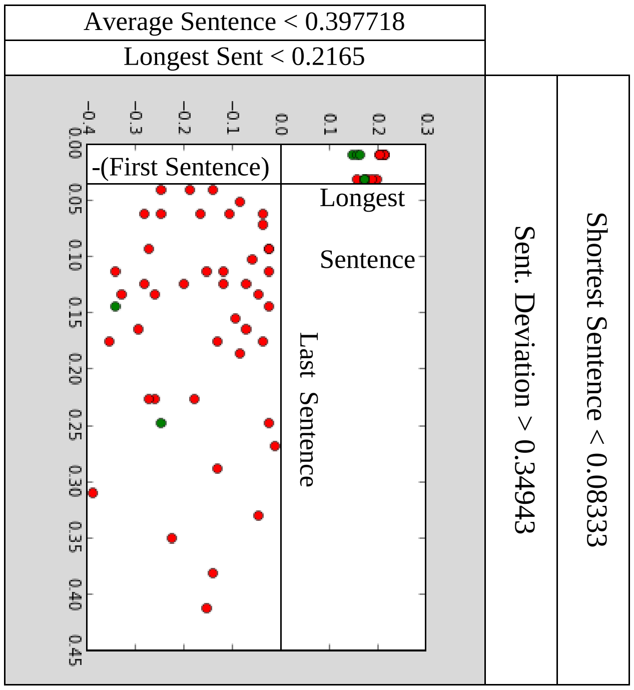
Predictor	Description
Total Words	Total Number of Words
Total Sentences	Total Number of Sentences
Number of "not" Contractions	Total Number of words containing "n't"
Total Number of "Not"	Total Number of Contractions + Total occurrences of "Not"
Last Sentence Length	Number of Words in the last sentence
First Sentence Length	Number of Words in the first sentence
Longest Sentence	Maximum number of words in a sentence
Shortest Sentence	Minimum number of words in a sentence
Sentence Deviation	Standard Deviation in length (in words) of all sentences
Number of Punctuation	Total count of parentheses, ellipsis, and question marks
Number of Contractions	Total number of words which contain an apostrophe
Number of Negative Prefixes	Number of words beginning with dis- , un- , in- , il- , im- , sub- , under- , non-
Total "You"	Total Number of occurrences of the word "you"
Closest "Not"	Location (in characters) of the usage of the word "not" (or a "not" contraction) which is closest to either the beginning or the end of the review

We included a few predictors which considered specific word structures in the review. Key components of language and syntax are modifying words and prefixes. For example, we may expect that negative reviews use the word "not" (or words ending in the contraction "n't") more frequently. Similarly, we also count the number of words which begin with a prefix carrying a negative connotation. One may also observe that we counted the total number of uses of "you" in each review: this arises from a conjecture that dissatisfied movie viewers, in trying to impress their disappointment on the reader, tend to address the reader more directly- thus using the word "you" more frequently.

Note: After computing these new predictors for all documents, each predictor was normalized to have values between 0 and 1 for all documents [For each predictor, we normalized using (value – min)/(max – min)]

Using these new predictors, we began to examine a number of 2-dimensional plots. Taking the full set of (now 15) predictors, we performed Kernel PCA, using the kernel functions provided by scikit-learn (linear, radial basis, and cosine, for example). Plotting the first two principal components of this data did not separate the positive and negative reviews. We also generated a number of combinations of these predictors–using both sums and products–and generated pair plots and again used Kernel PCA. Once again, these methods were unsuccessful. In light of these (many, many) unsuccessful attempts, we thought of the following: can we take a data structure which is inherently higher than two-dimensional, and represent it in two dimensions? This led us to the following novel (and somewhat successful approach) to this problem.

Decision trees are an excellent (and widely-used) tool in classification problems. They are very flexible, and easy to interpret. Perhaps a decision tree can point to subsets of the predictor space in which a two-dimensional scatter plot separates the positive from negative reviews. That is, can we use a classification tree to construct a <u>set</u> of two-dimensional scatter plots which separate the positive and negative reviews? What follows is our attempt at this idea. Using the whole data set (with these new predictors) we trained a decision tree of depth <u>six</u> on this data. Using this tree, we then developed series of two-dimensional scatter plots. The full plot can be found on page 15: here we will present one sub-plot of this whole do demonstrate what this plot means. Our plot contains 14 subplots: each such subplot corresponds to 4 splits beginning at from the top of the decision tree. These four splits are denoted by the side-bars corresponding to each section of the plot. For example, consider the following sub-plot:



Therefore this particular subplot contains exactly those points such that (**Average Sentence** < 0.3977) and (**Shortest Sentence** < 0.08333) and (**Longest Sentence** < 0.2165) and (**Sentence Deviation** < 0.34943). We use the scatter plot to denote *two* additional levels in our decision tree. The vertical axis represents the predictor split at the fifth-level of this particular branch. The horizontal axis then represents (simultaneously) *both* predictors split on at the sixth level of this tree. This deserves special attention, as it is not intuitive. In the example above, at the *fifth* level of this branch, our classification tree splits our predictor space into (**Last Sentence** ≤ . 035) and (**Last Sentence** > 0.035). Now in the *sixth* level of the tree, when (**Last Sentence** ≤ 0.035), our tree splits on the predictor **Longest Sentence**. Therefore scatter **Last Sentence** vs. **Longest sentence** for this subset of the data. On the other side of the 5th level split, when (**Last Sentence** > 0.035), our tree splits on the predictor **First Sentence**. As before, we wish to scatter-plot **Last Sentence** vs. **First Sentence**. In order to avoid graphical ambiguity, we construct this scatter plot on the *other* side of the (**Last Sentence**) axis: that is, we plot **Last Sentence** vs. –(**First Sentence**). Therefore the subplot that appears above actually contains *two* scatter-plots. Separating the horizontal axis (at 0) to represent two different predictors allows us to accomplish this. While this may seem over-complicated (and indeed it is to a point) this separation: using one pair of axes to represent two scatter plots, allowed us to add an additional level to our tree. This additional level lends greater flexibility to our decision tree, which will (hopefully) lead to better separation in the plots.

The full rendering of a 6-level tree as such a system of 2-dimensional scatter plots appears on Page 15. Green points represent positive reviews and red points represent negative reviews. It should be noted that each review in our data set appears *exactly once* in the system of plots. Given the rendering description above, the 14 subplots actually represent 28 scatter plots: each such scatter plot represents a distinct region of our predictor space. Looking at the full set of plots, some have them have mostly red points, or mostly green points, and thus do a good job of "separating" the reviews. Many of the plots do not separate the data at all, however. Thus despite a rather convoluted method of producing a "2-dimensional" plot, we would evaluate this method as somewhat successful. In some regions of our predictor space, our 2-dimensional scatter plots do an excellent job of separating the data. This is an improvement over all other methods we tried: when producing scatter plots of the whole data set the positive and negative reviews were always sufficiently "mixed up." Thus although this method is not an overall success, it is the most successful plot that we were able to produce.

Business Intelligence and Decision Making

As mentioned in the introduction, countless informal movie reviews are authored on social media every day. Though presented as unstructured text, understanding the sentiment expressed can be extremely value to movie production studios. While a small number of official reviews by "critics" are released for each film, the volume of movie opinions offered freely via social platforms is massive. 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).

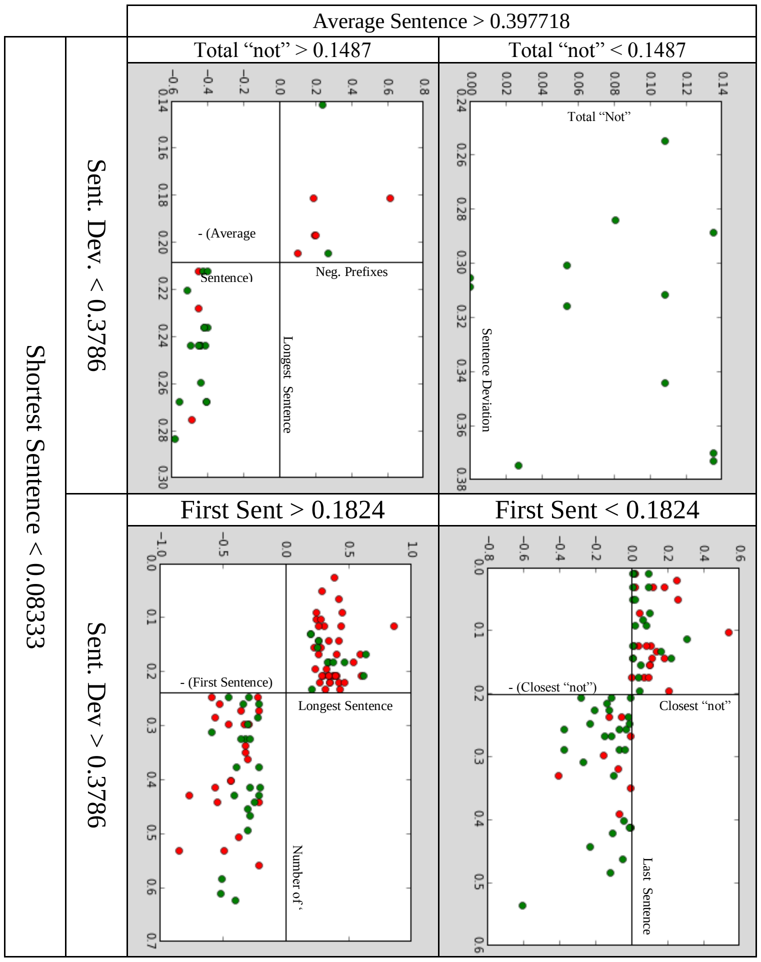
As a result, such data analysis can help to shape business decisions. Understanding demographic popularity can assist in advertising decisions: to which gender should more marketing be targeted? Understanding the age groups for which a movie performs well can help determine future re-release of a film: 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 superhero 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

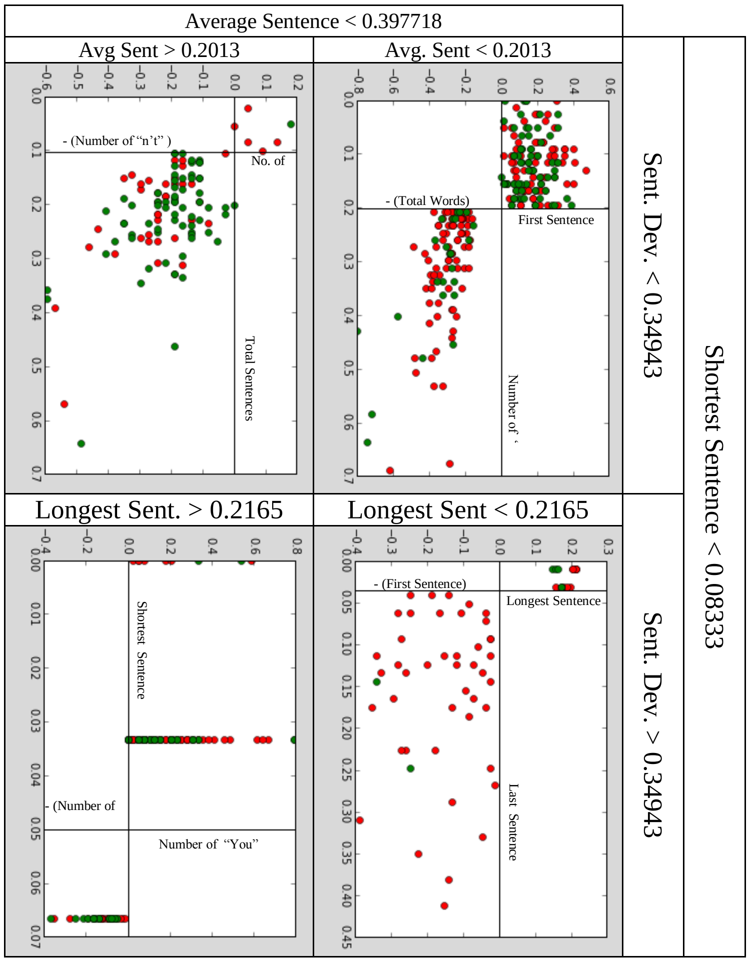
In this report, we studied the polarity (positive/negative) of unstructured movie reviews, with a particular interest in prediction via machine learning algorithms. Using a dataset of 2,000 movie reviews, we first performed preliminary sentiment analysis via a text classification pipeline. Parameters were chosen via grid search, and performance was evaluated using a validation set. Next we explored the scikit-learn TfidVectorizer 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.

Additional experiments were performed beyond those presented here. We tried a number different split rates (testing-training data) and cross-validation techniques (such as 5- and 10- fold cross validation) to evaluate our algorithms. Those results offered little improvement over those presented. Moreover, we took care in observing the run-times of machine learning algorithms: an important factor in practical implementation. In all cases, KNN required significantly longer computation time than did linear SVC (10 minutes compared to about 1 minute). Using 5-fold cross validation in place of validation-set for evaluation also increased run-times by a factor of about 3.5.

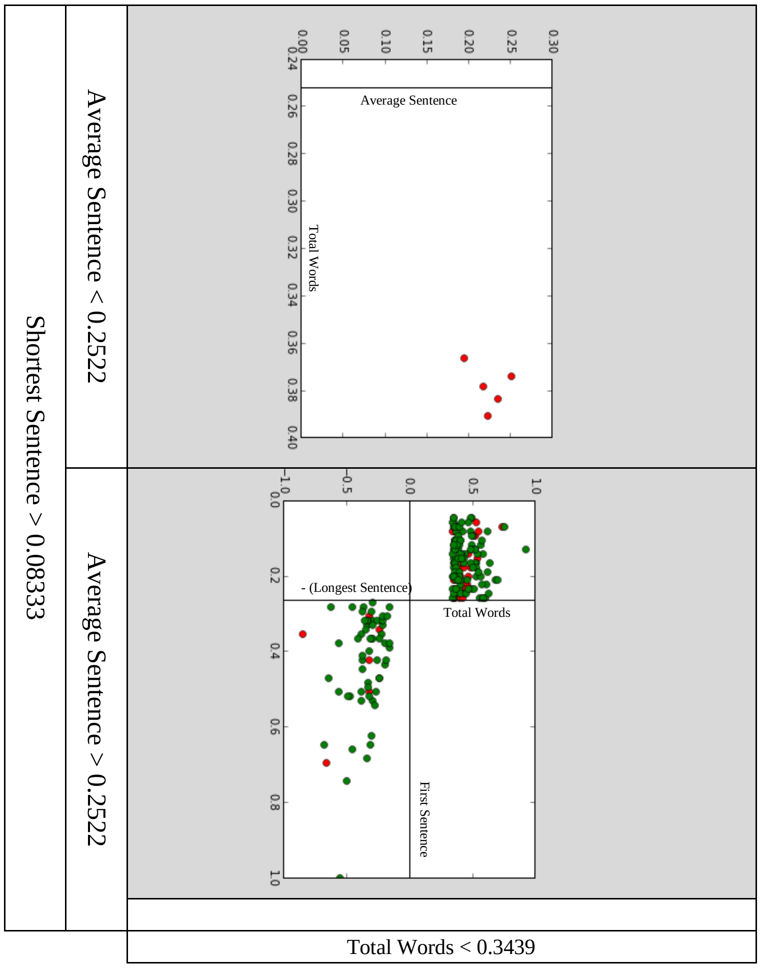
Finally, we tried to determine a 2-dimensional plot which separated the negative and positive reviews. We conjectured that the polarity of a review could be determined based on the structure of the test, and thus computed a new set of structure-based predictors on our movie reviews. Using these predictors, after a number of unsuccessful visualization attempts, we tried to represent a 6-level classification tree via a (disjoint) set of 2-dimensional scatter plots. While this allowed us to successfully separate the reviews in some subsets of the predictor space, it did not in all. This 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. Our “best” KNN model performed worse than a random coin-flip classifier on a set of negative test reviews. Much further (and computationally-expensive) analysis could certainly reveal surprising trends in this data set.



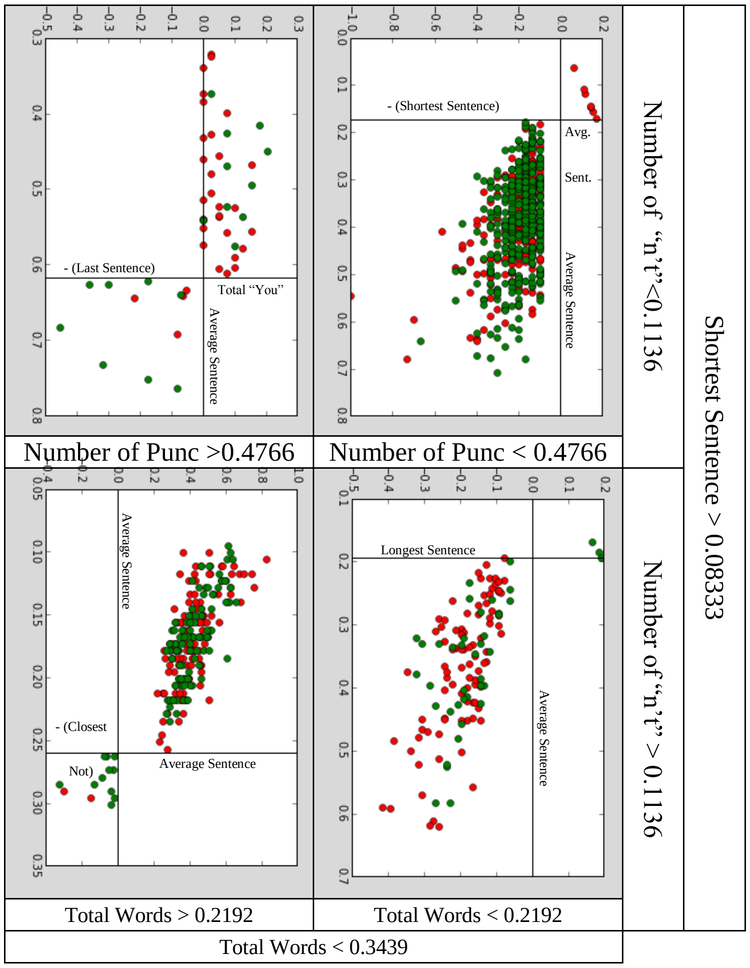
Average Sentence > 0.397718
Total "not" > 0.1487
Total "not" < 0.1487
First Sent > 0.1824
First Sent < 0.1824
Sent. Dev. < 0.3786
Sent. Dev > 0.3786
Shortest Sentence < 0.08333
- (Average Sentence)
Neg. Prefixes
Longest Sentence
Total "Not"
Sentence Deviation
- (First Sentence)
Longest Sentence
Number of '
- (Closest "not")
Closest "not"
Last Sentence



Average Sentence < 0.397718
Avg Sent > 0.2013
Avg. Sent < 0.2013
Longest Sent. > 0.2165
Longest Sent < 0.2165
Sent. Dev. < 0.34943
Sent. Dev. > 0.34943
Shortest Sentence < 0.08333
- (Number of "n't")
No. of
Total Sentences
- (Total Words)
First Sentence
Number of '
Shortest Sentence
- (Number of
Number of "You"
- (First Sentence)
Longest Sentence
Last Sentence



Average Sentence < 0.2522
Average Sentence > 0.2522
Shortest Sentence > 0.08333
Average Sentence
Total Words
- (Longest Sentence)
Total Words
First Sentence
Total Words < 0.3439



- (Last Sentence)
Total "You"
Average Sentence
- (Shortest Sentence)
Avg. Sent.
Average Sentence
Number of Punc >0.4766
Number of Punc < 0.4766
Average Sentence
- (Closest Not)
Average Sentence
Longest Sentence
Average Sentence
Number of "n't"<0.1136
Number of "n't" > 0.1136
Shortest Sentence > 0.08333
Total Words > 0.2192
Total Words < 0.2192
Total Words < 0.3439

APPENDIX I: Incorrectly CLASSIFIED REVIEWS.

Review 1. (True Polarity: Negative. Predicted Polarity: Positive)

```
susan granger's review of " session 9 " ( usa films )
sometimes you just get more than your bargained for . . . like when boston-based hazmat elimination , run
by scottish actor peter mullan and his trusty assistant , david caruso , assures a town engineer ( paul
guilifoyle ) that they can remove insidious asbestos fibers from a victorian hospital facility in a week .
erected in 1871 , deserted and decomposing since 1985 , the danvers mental hospital , is one of the most
malevolent " locations " ever chosen for a film .
the structure is so massive - with its labyrinth of rubble-strewn corridors , collapsing floors , stagnant
pools of water , isolation cells , and ominous surgical chambers where experimental pre-frontal lobotomies
were performed - that their task seems impossible within that time frame .
and each member of their inexperienced crew ( stephan gevedon , brandon sexton iii , and josh lucas ) is
coping with his own personal demons as , one by one , their minds seem to be affected by the grim areas in
which they're working .
the film's title is derived from salvaged reel-to-reel audio-recorded sessions involving the demonic
possession of a young woman who is suffering from multiple personalities .
by the time session 9 occurs so do dreadful disasters .
filmmaker brad anderson obviously envisioned this as a gruesome chainsaw-massacre-type ghost story but the
script lacks structure and isn't particularly scary .
the conclusion is more ludicrous than convincing .
on the granger movie gauge of 1 to 10 , " session 9 " is a dark , gloomy 4 . silly me . . . at
first , i thought that the original name of the danvers lunatic asylum bore some reference to mrs .
danvers , the creepy housekeeper played by judith anderson in alfred hitchcock's truly terrifying "
rebecca " that also involved a cavernous mansion called manderley .
```

Review 2. (True Polarity: Positive. Predicted Polarity: Negative)

```
this summer , one of the most racially charged novels in john grisham's series , a time to kill , was made
into a major motion picture .
on january 3 of this year , director rob reiner basically re-released the film under the title of ghosts
of mississippi .
based on the true story of 1963 civil rights leader medgar evars' assassination , ghosts of mississippi
revolves around the 25-year legal battle faced by myrlie evars ( whoopi goldberg , sister act ) and her
quest to have her husband's obvious assassin and racist byron de la beckwith ( james woods , casino )
jailed .
so she turns to assistant district attorney and prosecutor bobby delaughter ( alec baldwin , heaven's
prisoners ) to imprison the former kkk member .
ghosts sets its tone with an opening montage of images from african-american history , from slave-ship
miseries to life in the racist south of the 1960's .
but all too soon , the white folks take over , intoning lines like " what's america got to do with
anything ?
this is mississippi ! "
as beckwith , james woods , with his head larded with latex most of the time as an old man , teeters
between portraying evil and its character .
meanwhile , goldberg turns in a very serious and weepy performance as the wife who wouldn't let her
husband's death rest until she got the conviction .
both deserve serious oscar-consideration .
this brings us to the dull performance of baldwin .
let's face it , trying to match matthew mcconaughey's wonderful acting in a time to kill is basically
impossible .
and baldwin is living proof of this , as no emotions could be felt .
it seemed as if he actually had to struggle to shed a single tear .
either poor acting or poor directing , but something definitely went wrong .
another strange mishap was the fact that goldberg's facial features didn't change , as she looked the same
in the courtroom as she did holding her husband's dead body 25-years earlier .
yet woods' was plastered with enough make up to make him look like goldberg's father .
at least the make-up was realistic .
with some emotional moments in the poorly written script , ghosts of mississippi lacked in heart , when
its predecessor , a time to kill , brought tears to everyone's eyes .
don't get me wrong , the movie wasn't all that bad , but if you've seen grisham's masterpiece , then don't
expect this one to be an excellent film .
```