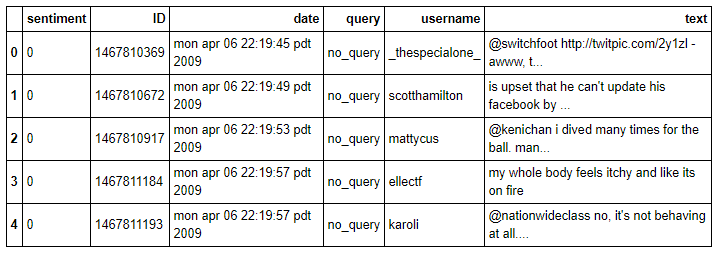
**Introduction**

In our project, we decided to try to work on a sentiment analysis method. Especially in today’s social-media focused world, automated sentiment analysis on blogs, tweets, and other types of social platform media becomes increasingly important for public figures, political campaigns, or a slew of other uses.

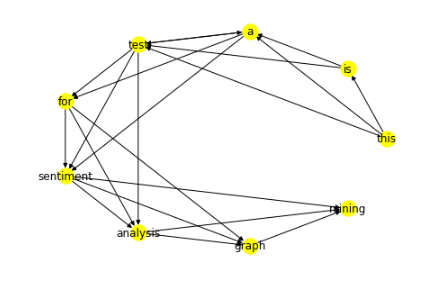
Although there are many existing methods to perform sentiment analysis, from simpler map-reduce methods to complex machine learning algorithms, one area which is not as explored is graph mining. Thus, we chose to attempt to implement sentiment analysis using graph mining. We used a dataset consisting of 1.6 million tweets, which were pre-classified into positive and negative tweets. (Although the dataset we sourced these tweets from claimed that the tweets were separated into three classes- positive, negative, and neutral, we did not find any tweets that were classified as neutral.) We constructed several different graph models that are representative of positive tweets, as well as another set of graph model representative of negative tweets. We also generated graph models of individual tweets in order to train a classifier, of which we used several, such as k-nearest-neighbors, support vector machine, ID3 decision tree algorithm, and also logistic regression. To train the classifier, we used several different methods to generate vectors representing each graph model of each tweet, which we will discuss in further detail later.

**Methodology**

 Our data came in the form of a CSV file, which was loaded into a Pandas data frame. We made sure each letter was lowercase, and ultimately our data, before any of the text was processed, looked like this:

However, attempting to run our project with 1.6 million tweets proved challenging, as we were limited by computing power. Some of the functions which we tried to run, such as building a graph model, took over two days and had not yet completed before we decided to try a different route. We used a keyword likely to garner both positive and negative tweets (“food”), and ended up with 8440 tweets, which is number that is much more easily digestible. Given that we have enough time at the end of project design, we could complete writing all the functions necessary to analyze data, select the best method of analysis, and apply to a larger dataset.

We split our data 80/20 into a training set as well as a testing set. The training set was further split—60% of the training set developed our positive and negative representative models, and 40% of the training set trained our data mining classifier. The set used to train our positive/negative model was also split into positive and negative tweets, which was already classified in the given dataset.

 Before we generated any graphs, we pre-processed the text in each tweet, removing stop words to improve run time, as well as removing any characters that are not letters, words, or spaces. In order to represent a string of words in a graph format, we looked at each word in the string, as well as the words that appeared in its neighborhood, which we called the frame. If we set the frame equal to two, then our algorithm would direct an arrow from the first word to the second word, as well as the first word to the third word. The frame would then slide one word to the right, treating the second word as the current first word, and linking the next two words in a similar manner. If a node representing a word already exists, then our algorithm draws an arrow to and/or from the existing node, without creating a new one.

The diagram on the right shows an example of what the graph of the sentence “this is a test a test for sentiment analysis graph mining” with a frame of four looks like. Note that “a test” is written twice in the above sentence to demonstrate how the algorithm handles existing nodes.

**Methodology- Graph Similarity**

Following, we define several different functions representative of similarity between graphs. In all of these following functions, the higher the number is in comparing a tweet to a graph model, the more similar the tweet is to that graph model.

Edge Similarity: Counts the number of common edges between the tweet graph and the graph model and normalized by the minimum number of nodes in either graph.

The maximum common subgraph, or MCS, is generated by taking the largest connected component generated by finding all common edges between the tweet graph and the graph model, and can treat edges as directed or undirected, depending on the parameter. The MCS is referenced in the following three functions.

MCS Node Similarity (MCSNS): Counts the number of nodes in the MCS, normalized by the minimum number of nodes in either graph.

MCS Undirected Edge Similarity (MCSUES): Here, common edges in the MCS are generated if the edge connects the same two nodes in both graphs, regardless of direction. The edges are then counted and normalized by the minimum number of nodes in either graph.

MCS Directed Edge Similarity (MCSDES): Here, common edges in the MCS are generated only if the edge connects the same two nodes in both graph in the same direction. The edges are then counted and normalized by the minimum number of nodes in either graph.

Term Frequency Inverse Document Frequency (TF-IDF):

TF-IDF is a generated statistic that measures the frequency of a term in a document, discounted by the number of documents that have the word. Its formula is as follows:

In our case, we treated each edge in the tweet as a term, and in evaluating each tweet model, we had three documents- the tweet model, the positive graph, and the negative graph. Each edge also had an associated weight in it, which represented how many times the edge was represented in generating the graph model. Each edge received a TF-IDF score. If an edge appeared in both the positive and the negative graphs, regardless of the term frequency, the edge was discounted because IDF is equal to the total number of modes over the number of models with the edge which is log(1) = 0. This leaves us with only the edges that were unique in either the positive or negative graph. The scores of each edge in the tweet model were then aggregated using some function, generating a positive or negative vector for the tweet. We used the following TF-IDF functions in our analysis:

Max TF-IDF: The maximum score for the set of positive and negative vectors generated for each edge was used as representative for the entire tweet graph model.

Average TF-IDF: The average score for the set of positive and negative vectors generated for each edge was used as representative for the entire tweet graph model.

Modified TF-IDF: Finally, we designed a simple algorithm of our own. Although we called it TF-IDF, it diverges from the TF-IDF algorithm. Instead of simply testing whether or not the graph model had the edge, we took into account the actual weight of the edge in the graph. For example, if we are evaluating how similar a term is to a positive graph, it will generate a higher score if the edge has a weight of 20 in the positive graph and only 1 the negative graph, versus the vice-versa situation. Additionally, the weight of the edge in the tweet graph also factored into the equation. The final equation is as follows:

**Methodology- Classification**

Lastly, we used several classification methods in order to classify the positive/negative vectors generated from each tweet and identify whether the tweet is positive or negative:

k-Nearest-Neighbors: Existing, labeled datapoints are plotted. For each new, unlabeled datapoint, the label is assigned by taking the “majority vote” of the k-nearest neighbors on the graph by Euclidian distance. This classifier was chosen due to intuitiveness of model.

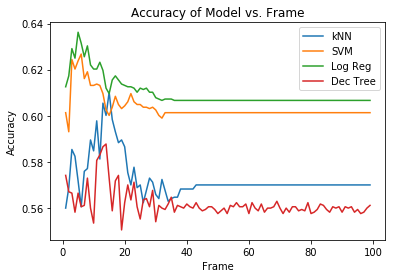
|  |  |
| --- | --- |
| **Kernel** | **Score** |
| Linear | 0.613151 |
| RBF | 0.609004 |
| Poly | 0.549763 |
| Sigmoid | 0.506516 |
|  |  |

Support Vector Machine: This is a discriminative classifier that uses a separating hyperplane. We decided to use a linear kernel, as our two-dimensional data is likely to have a linear separator. In preliminary analysis, a linear kernel performed the best.

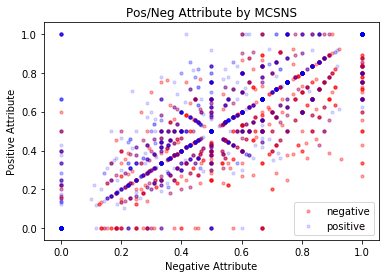
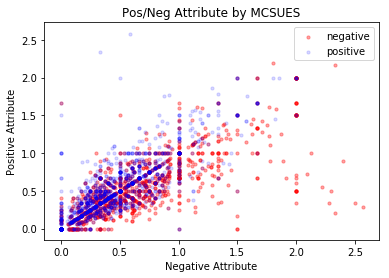
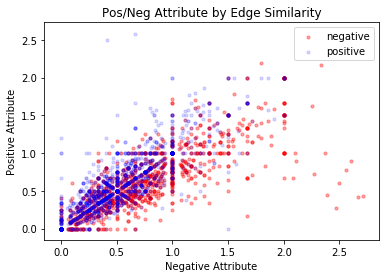
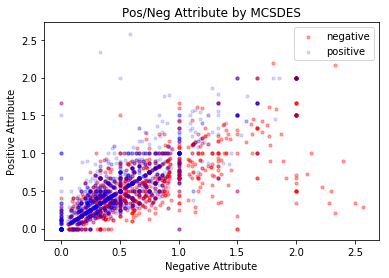
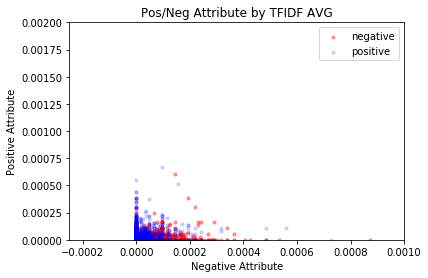
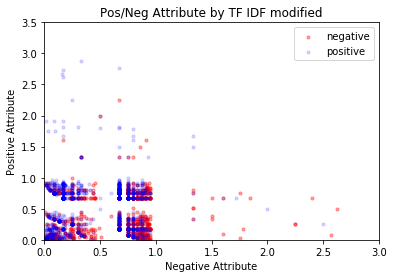
Decision Tree (ID3): We wanted to explore how using an information gain method may work- this was out of curiosity and not founded on a basis of confidence it would work well. In fact, because of the numerical and potential minute fluctuations in data that would classify a tweet as positive or negative, we did not expect this method to work out well.

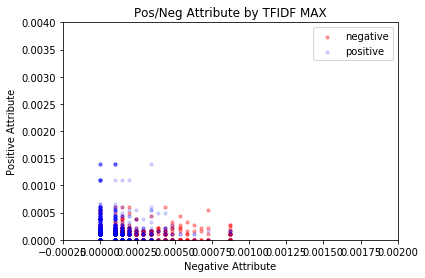
Logistic Regression: Similar to the support vector machine kernel, because our data is likely to have a linear separator, we wanted to attempt the simplest method and analyze results produced by it.

**Analysis of Data**

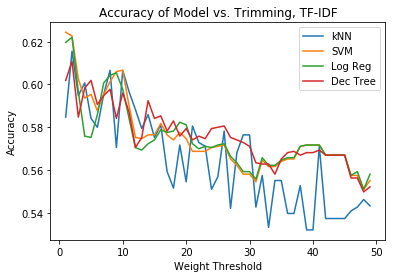
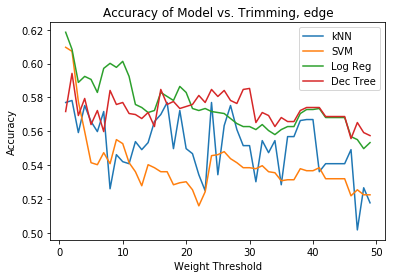
 Because of limitations in computational power, we decided to attempt to analyze our data beginning with only 0.5% of our entire dataset, or a total of 8000 tweets.

Initially, we wanted to see how difference in frame when generating our positive or negative word models changes the outcome of our results. We expected, in general, for the accuracy to increase as frame went up because of an increase in density of data, and to plateau and decrease after a certain point where the graphs were too densely connected and started to become similar. Interestingly, the accuracy of our SVM and logistic regression peaked at around a frame of 3-4 which was much smaller than what we expected, whereas the accuracy of our kNN and decision tree models peaked at around a frame of 10. After approximately a frame of 40 Additionally, SVM and logistic regression almost consistently performed better than kNN and decision tree, with decision tree consistently performing the worst.

 Following, we wanted to see how well our similarity measures separated tweets of different classes based on evaluating the similarity between the tweet and the positive or negative model. Below are graphs of how each similarity measure performed. Blue points are positive tweets, and red points are negative tweets. The x-scale is the negative score, and the y-scale is the positive score in each graph.



Ideally, positive tweets have a high positive attribute and a low negative attribute, meaning that blue (positive) points will cluster at the top left corner. Similarly, red (negative) points will cluster in the bottom right corner. However, it’s clear from these scatterplots that, regardless of similarity measure, the boundary between positive and negative tweets is very blurred. Even so, one can see that the TF-IDF measures do a better job of separating the positive from the negative tweets, with the modified method seemingly performing the best.

 Finally, we wanted to see if trimming edges with low edge weight would make a difference to our analysis. Since edges with low edge weight mean that the edges are not really represented in the majority of positive or negative tweets, trimming them may make computations easier as well as potentially reduce confounding calculations (eg, an edge between the words “not good” may be represented once or twice in positive tweets, but thousands of times in negative tweets. Yet, we don’t want these to carry equal weight in calculations where edge weight is not taken into account.) It quickly became clear that trimming models affected the accuracy of all classifiers, in both similarity measures that took into account edge weight as well as those which did not. We did not pursue trimming edges any further.

**Results**

Below is a table of our results. For some of the 50% of entire dataset calculations, the program ran for over 5 hours. Thus, we were unable to continuously refine our results, or test 100% of the dataset, as some of our algorithms are NPC.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0.5% of All Data** | | | | | | | |
|  | **Edge** | **MCSNS** | **MCSUES** | **MCSDES** | **Max TFIDF** | **Avg TFIDF** | **Mod TFIDF** |
| **kNN** | 0.585308 | 0.544431 | 0.568127 | 0.568127 | 0.542654 | 0.587677 | 0.595971 |
| **SVM** | 0.624407 | 0.576421 | 0.595379 | 0.595379 | 0.481614 | 0.479265 | 0.634478 |
| **Dec Tree** | 0.565165 | 0.546801 | 0.562203 | 0.562203 | 0.627962 | 0.595379 | 0.613151 |
| **Log Reg** | 0.629146 | 0.574644 | 0.606042 | 0.606042 | 0.479265 | 0.480265 | 0.639811 |
| **20% of All Data** | | | | | | | |
| **kNN** | 0.598102 | 0.557566 | 0.591406 | 0.591406 | 0.605267 | 0.624891 | 0.625615 |
| **SVM** | 0.648285 | 0.585421 | 0.622135 | 0.622135 | 0.598123 | 0.584612 | 0.679152 |
| **Dec Tree** | 0.636947 | 0.591801 | 0.618109 | 0.618124 | 0.615781 | 0.605495 | 0.619625 |
| **Log Reg** | 0.649359 | 0.594451 | 0.627125 | 0.627125 | 0.657513 | 0.651549 | 0.680216 |
| **50% of All Data** | | | | | | | |
| **kNN** | 0.607318 | 0.542089 | 0.574942 | 0.574942 | Calculations did not complete, took over 1.5 days and still running | | 0.654135 |
| **SVM** | 0.641659 | 0.575107 | 0.624312 | 0.621312 | 0.681352 |
| **Dec Tree** | 0.640243 | 0.583823 | 0.622583 | 0.622658 | 0.625435 |
| **Log Reg** | 0.642891 | 0.583147 | 0.625838 | 0.625838 | 0.701354 |

**Other Applications: Sentiment Analysis of 427 Homework Sentiments**

After analyzing tweet sentiment, we were curious as to whether using graph classification would perform better than using the trigrams model in hive from CSE427’s final project predicting homework sentiments. In order to input homework sentiments into the model, we first preprocessed all the homework reviews text files into a data frame and assigned integer values according to sentiment (4 = positive, 0 = negative). Similar to the tweet sentiment model, we used 80 percent of homework reviews to train the model and 20 percent to test it. After training the model and running the model for the test data, we found that the graph classification model had an accuracy of 90.40% and outperformed the trigrams algorithm implemented last year by about 27%.