CS 4700: Artificial Intelligence

Yik Yak Analyzer: A Study of College Campus Sentiment

Demitri Tzitzon, Peter Blanco, Rahul Shah

December 18, 2014

1 Mission Statement

Understand and compare the varying stress levels between Ivy League universities and State schools. This will be accomplished by using Geo-Centric social media data from an up and coming social media service known as Yik Yak.

We will use a Naive Bayes classifier to train data and then apply it to various universities and compare stress. The classifier will make use of negation handling, laplacian smoothing, n-grams, feature selection. Furthermore, in order to avoid re-inventing the wheel, we will make use of academic studies in Sentiment Analysis (and cite relevant work). Finally, we will also create a front-end visualization tool that can be used to compare stress levels between the various schools.

2 Software Overview

From a high-level, our analyzer breaks down the Yak (post on Yik Yak) word by word, running an Enhanced Naïve Bayes Model to determine sentiment. From the paper Fast and Accurate Sentiment Classification Using an Enhanced Naïve Bayes Model, we build our own algorithm, utilizing Laplacian Smoothing, negation handling, bi-grams, and feature selection. To train the algorithm, we used a csv of 80,000 hand ranked positive/negative tweets. To collect the Yaks, we wrote a script by reverse engineering the iOS Yik Yak API, and downloading the top 100 Yaks every hour for two nights from each selected school. This data was then passed into our analyzer, classified, and then stored to be utilized by our web based analytics dashboard.

2.1 Training

Or initial plan was to train our data by hand classifying a collection of retrieved Yik Yaks. To do so, we wrote a python script that enables the ease of classification by present a Yak to the person classifying and allowing them to enter p or n to classify. However there was a pitfall, after classifying around 300+ Yaks, we realized that our training data had many more negatives then positives to train on. After a discussion, we decided to search for a twitter sentiment analysis corpus to use as our training data because of the similarity in structure of a tweet and a yak. Through the use of a corpus of twitter training data acquired from Sanders Analytics with around 80,000 human classified tweets, we were able to accurately train our algorithm to detect positive and negative sentiment within textual input.

2.2 Yik Yak Scrapper

To acquire the Yik Yak data, we followed a blog post The Yak is a Hack detailing the structure and endpoints of the iOS API. Following this, we created a script in python to create new users, switch locations, and download the top 100 Yaks every hour from each selected school.

2.3 Classifier

The Classifier runs modified naïve bayes on our corpus of Yaks, applying Laplacian Smoothing, negation handling, bi-grams, and feature selection.

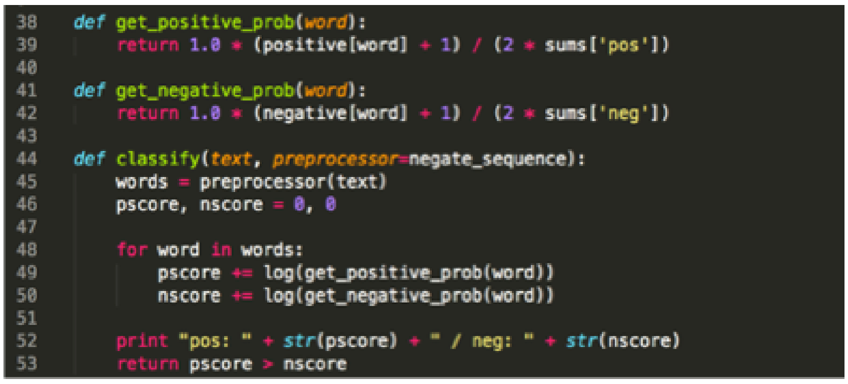
2.3.1 Naive Bayes Approach

We chose to use the Naive Bayes classifier after reading about it in the textbook, *Ariticial Intelligence - A Modern Approach*, and doing some basic research online and seeing that a Naive Bayes model is often cited as the most efficient and promising method for Sentiment classification. As the book explains, a Naive Bayes method is meant to quantify the relationship in the ?causal direction? (p. 496), almost as a diagnosis tool. In this case, we have we had prior data of classified textual documents (movie reviews and classified tweets) and were looking to use them to diagnose the sentiment of Yik Yaks.

As the book explains, a Naive Bayes classifier is rooted in Bayes Law, below:

$$\Pr(Cause|Effect) = \frac{\Pr(Effect|Cause)\Pr(Cause)}{\Pr(effect)} \tag{2.1}$$

The corresponding code in our implementation is as follows:



```python
def get_positive_prob(word):
    return 1.0 * (positive[word] + 1) / (2 * sums['pos'])

def get_negative_prob(word):
    return 1.0 * (negative[word] + 1) / (2 * sums['neg'])

def classify(text, preprocessor=negate_sequence):
    words = preprocessor(text)
    pscore, nscore = 0, 0

    for word in words:
        pscore += log(get_positive_prob(word))
        nscore += log(get_negative_prob(word))

    print "pos: " + str(pscore) + " / neg: " + str(nscore)
    return pscore > nscore
```

2.3.2 Additional Features

1. **Laplacian smoothing**

   Because it is likely our classifier will encounter a word that is not seen in the training set yet, the classifier applies Laplacian smoothing to avoid naïve Bayes assigning the probability of 0 to the word. What this does is give a word equal probability to be in either class, positive or negative, if it has not been seen yet.

2. **Negation handling**

One of the major problems in sentiment classification is the handling of negated words. For example "not happy" and "happy" could be classified as both positive. To help litigate these issues, the classifier applies negation handling to all terms by adding their negated version to the model. While iterating through the tokens in a textual input, an internal negation state is maintained. The negation state is activated when a negation word such as "not" or "no" appears. The following word or group of words is then indicated as negated. This resolves the issue of "not good" vs. "good" in sentiment analysis. Additionally, for each word that was categorized as positive or negative, its negation was then categorized as the opposite. For example, if we observed ?beautiful? as positive in our training set, we would also observe "not beautiful" as negative.

3. **N-grams**

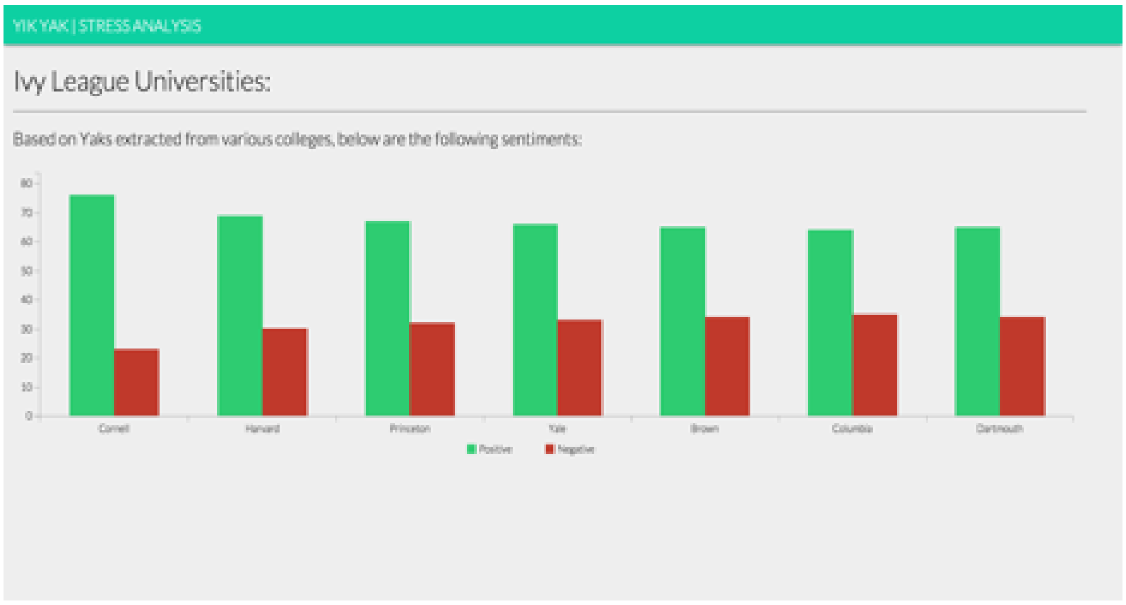
   The classifier captures both unigrams and bigrams (consecutive pair of words) to help with the classification of terms that have adjectives and adverbs. We found that bigrams improved performance over unigrams by about 5%. Unigrams did not provide a sufficient amount of contextual information and we found too many ambiguous words classified as highly negative or positive. When adjectives and adverbs are classified alone, they do not hold much value, but when applied to another word, their meaning can become blatantly obvious. For example "very" as a unigram does not give much information to the classifier, but the bi-gram "very excited" does. This is what allowed bigrams to improve on performance to such and extent. We also incorporated trigrams. However, we found that trigrams did not improve performance. This is likely due to the nature of Yik Yak posts. They are restricted to 200 characters, which causes the likelihood of three word groupings to be too low to be useful.

4. **Feature selection**

   With the use of bi-grams and negation handling, the number of features in our model begin to become very large. To reduce the size of the features, improving performance time, the classifier iterates through the classification model after being trained and filters out terms that occur only once. We also limit our feature set in order to reduce the number of features necessary to store and compute on. This improves both time and space efficiency. We do this by computing the mutual information for each term, and saving only the terms with the highest mutual information.

2.4 Data Analytics Dashboard

In order to visualize our results, we prepared a front-end using the C3.js data visualization library in order to compare the various schools stress levels. A screenshot of the frontend is seen below:

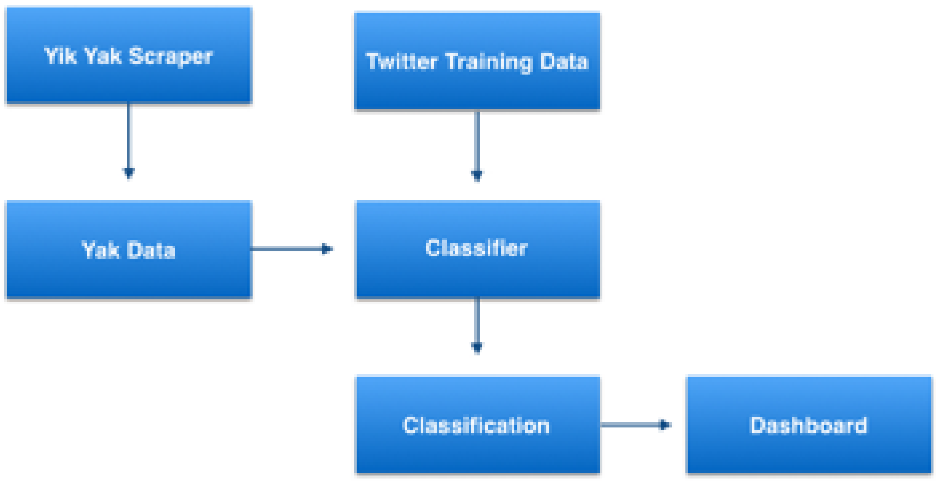


YIK YAK | STRESS ANALYSIS

Ivy League Universities:

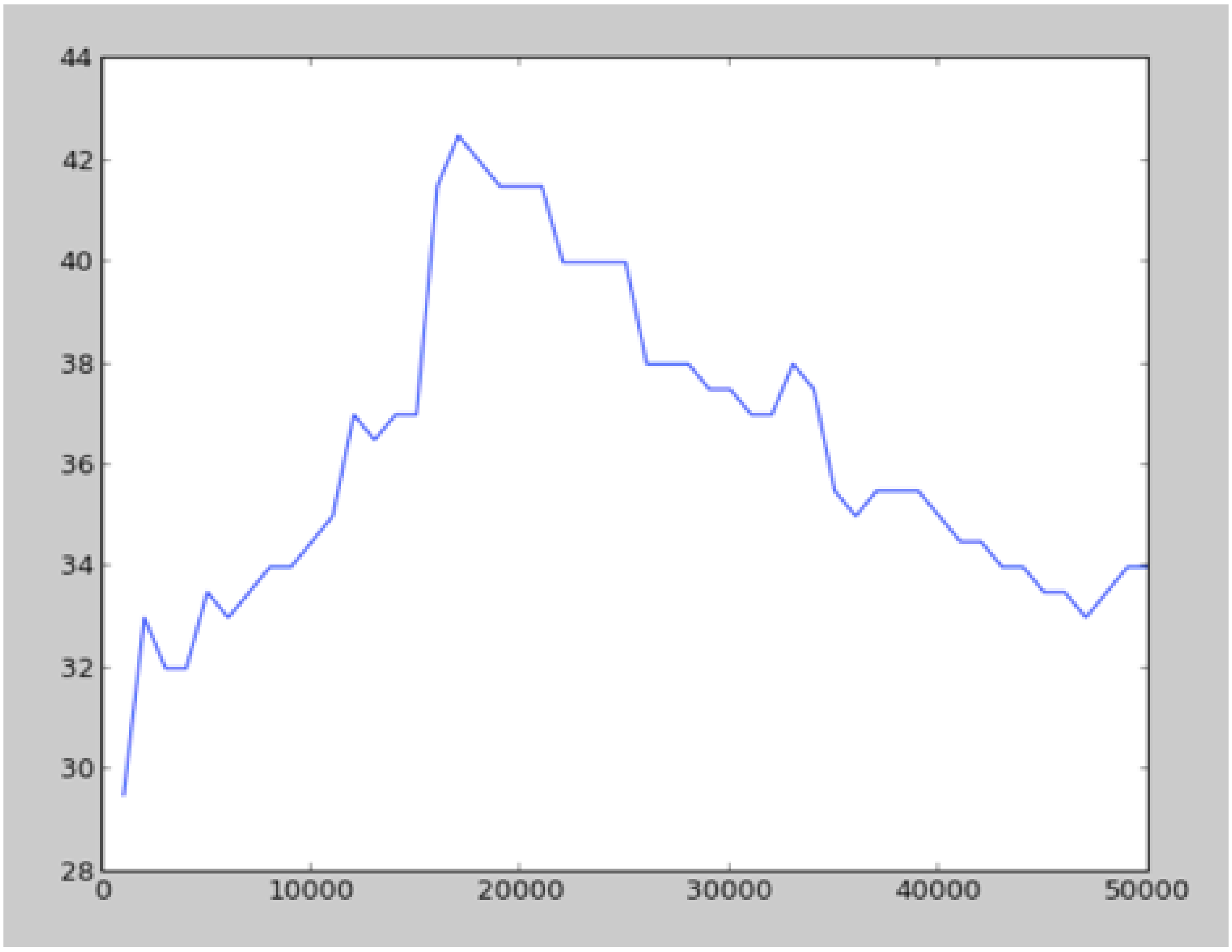
Based on Yaks extracted from various colleges, below are the following sentiments:

2.5 Process Flow Diagram



3 Development Process and Attempts

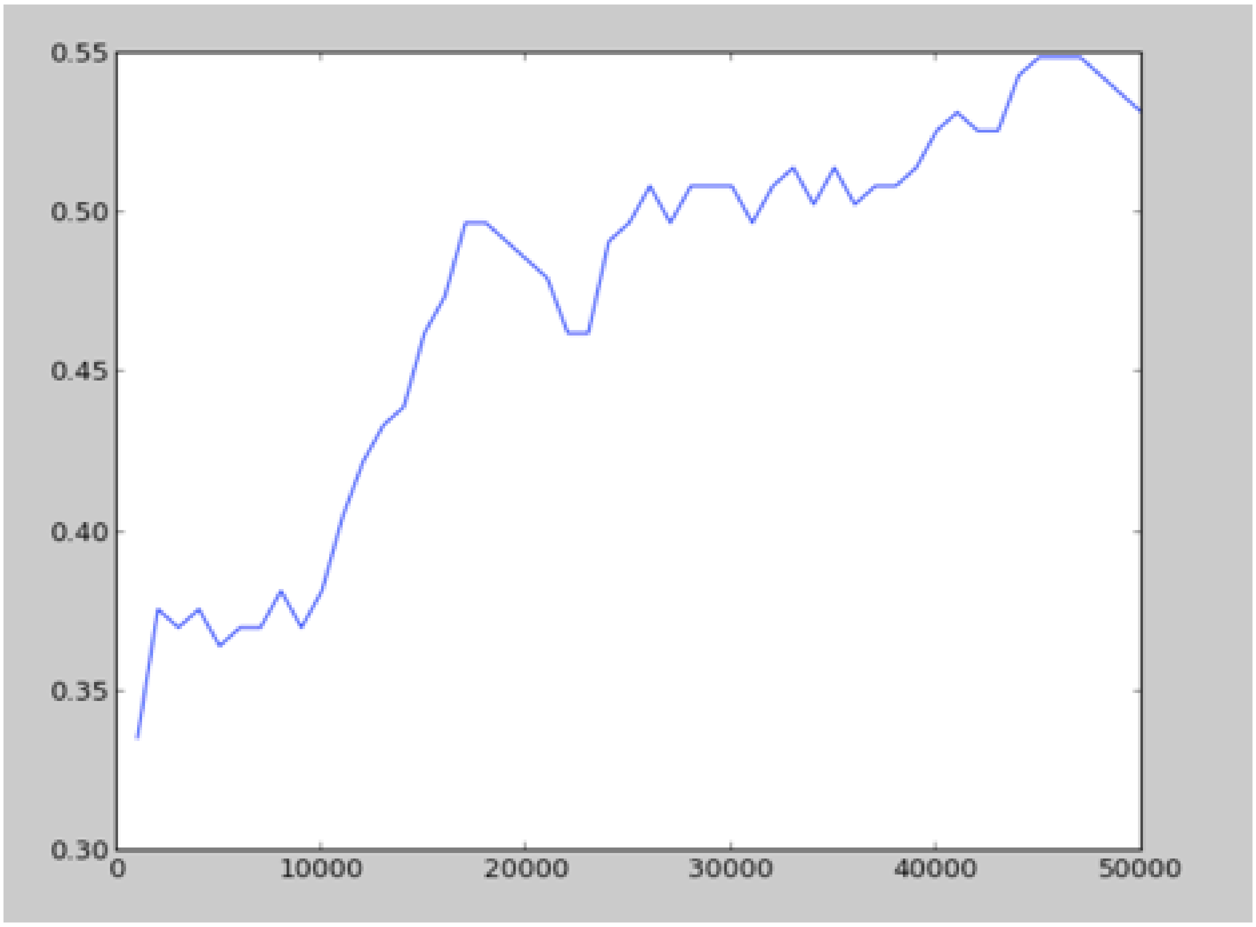
Attempt 1: Movie Rating with Naïve Bayes, Negation Handling, and LaPlacian Smoothing



Our first attempt was to train based on Movie Ratings from the ?Movie Review Data? dataset provided by Cornell (http://www.cs.cornell.edu/people/pabo/movie-review-data/), which includes 1000 positive and 1000 negative processed reviews.

As is evident by the graph, the results from this data were very weak; in fact, it was not able to beat 50% (a coin toss). We believed the training set of movie reviews to be effective due to the opinionated nature of movie reviews as well as the fact that the movie reviews were of medium length. Therefore, each word would have held positive or negative sentiment, but the length would be long enough to train a sizable number of features. We expect that the classifier performed worse than a coin flip when trained on this movie data due to the high level of sarcasm in Yik Yak posts.

Attempt 2: 20k Twitter Data with Naïve Bayes (Negation Handling, and LaPlacian Smoothing)

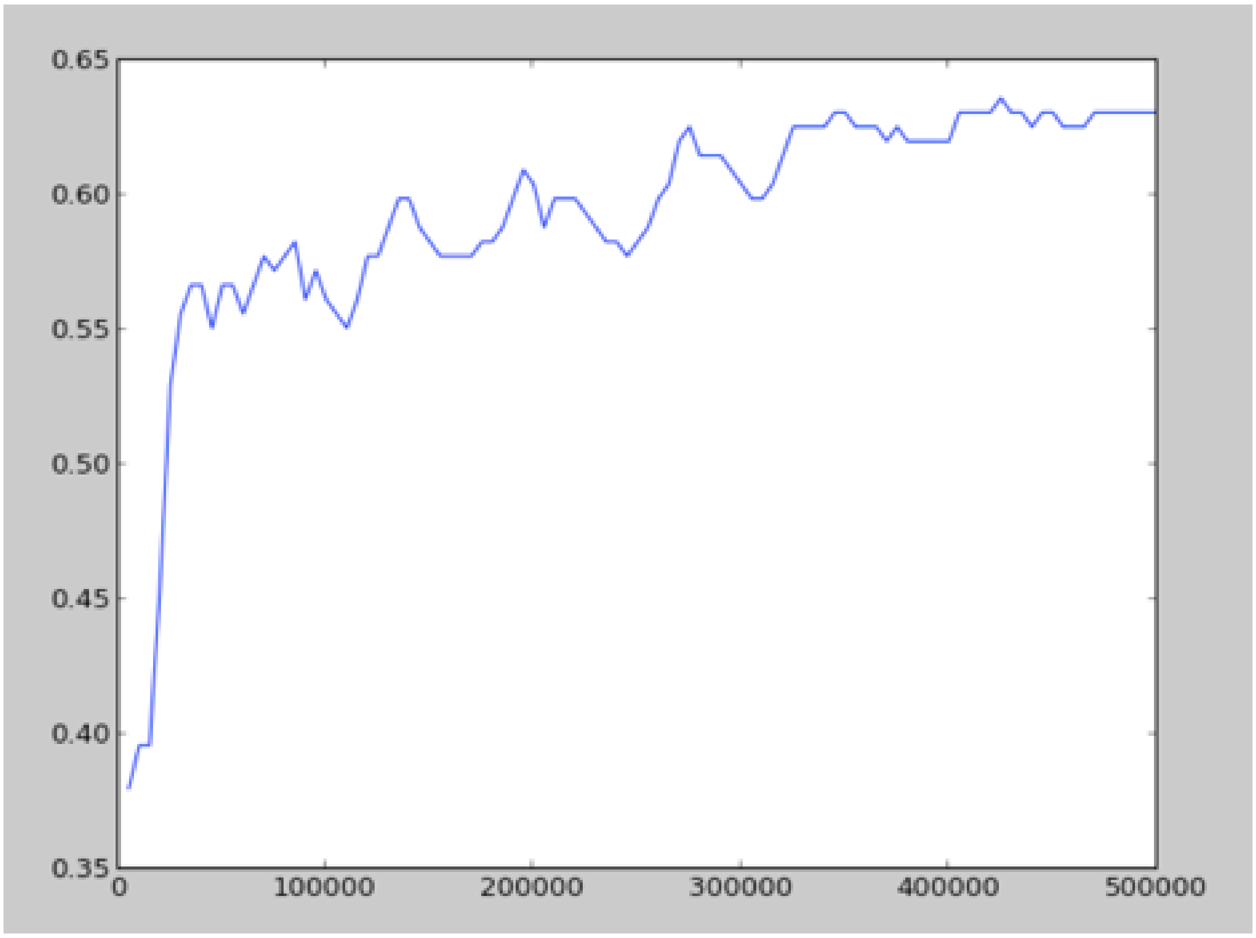


We suspected that the major reason that the previous approach was incorrect was because the difference between the type of language in the training data and the type of language in Yik Yaks. Therefore, we decided to find another training set that more closely resembled our use case: Twitter data.

We suspected that the major reason that the previous approach was incorrect was because the difference between the type of language in the training data and the type of language in Yik Yaks. Therefore, we decided to find another training set that more closely resembled our use case: Twitter data.

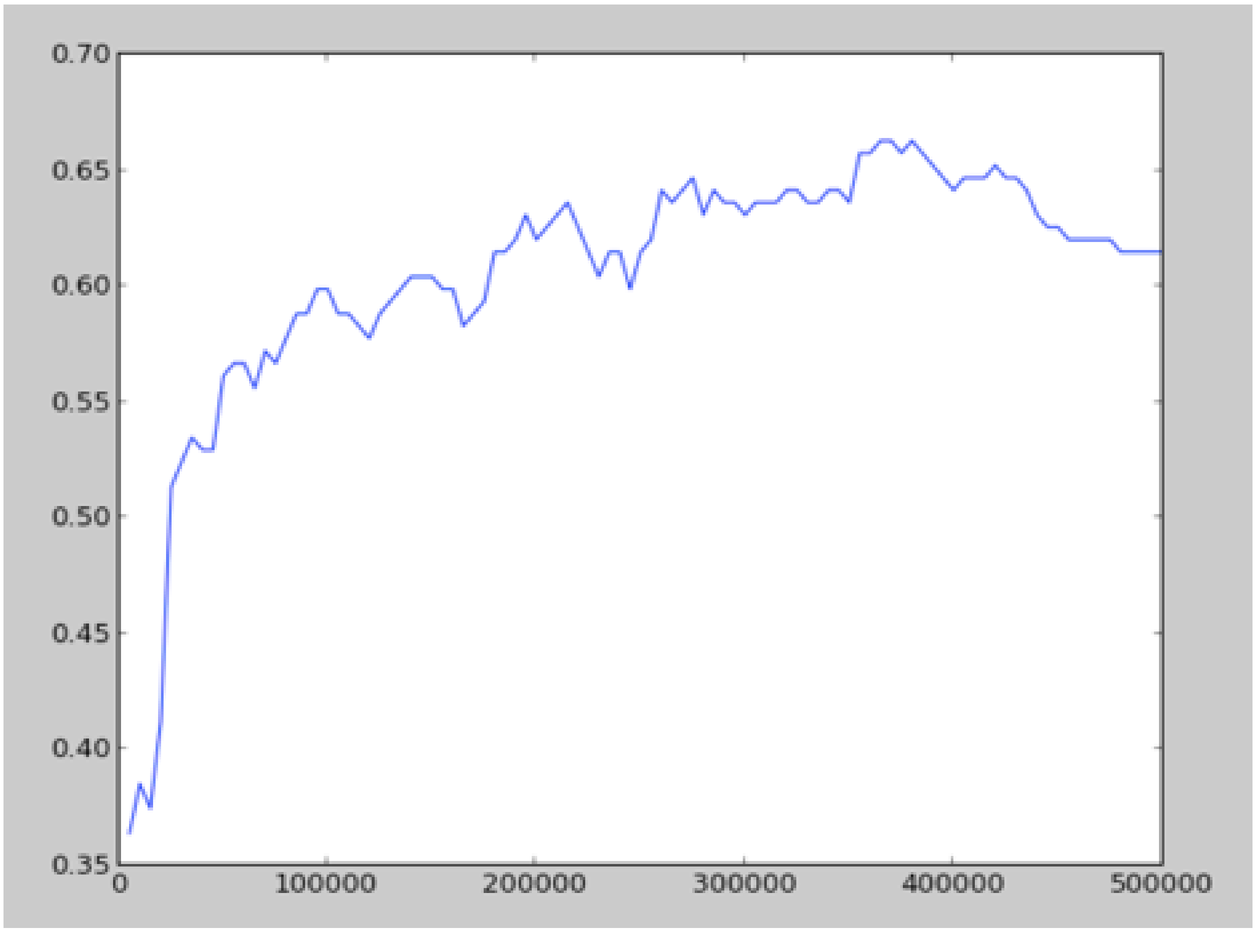
We were able to find a public twitter dataset with preprocessed tweets that were marked with positive or negative sentiment. The results were able to beat a coin-toss, but not by a significant amount. We discovered that this was likely due to the short nature of the tweets. Although they are similar in language and size to Yik Yak posts, the size of the tweets limits the number of features we were able to train on. We determined that we needed more features, and that we needed to train on a greater number of Twitter posts.

Attempt 3: 70k Twitter Data with Naïve Bayes (Negation Handling, and LaPlacian Smoothing)



In order to further improve the results, we first attempted to run the system through more training data. We found that performance started to plateau once trained on 70,000 tweets. Training on additional tweets was able to improve performance by almost ten percentage points. The similarity between the tweets and the Yik Yak posts combined with a high number of features was enough to achieve a better than random sentiment categorization.

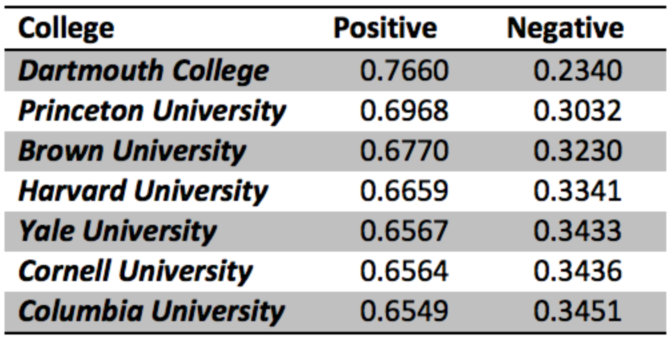
Attempt 4: 70k Twitter Data with Naïve Bayes (Negation Handling, and LaPlacian Smoothing, and Bigram Handling)



The final adjustment that was attempted was to add bigram analysis. This meant that instead of just looking at individual words, we were analyzed groups of 2-pairs to analyze train and classify with. This allowed us to capture statistically significant phrases as well as statistically significant words. While bigrams improved performance by about 5%, we found that Trigrams had virtually no performance impact. This was likely due to the short nature of Yik Yak posts and the limited number of statistically significant trigrams.

4 Result

Our results show that Dartmouth College is the most positive Ivy League. It is a full .11 higher than Columbia University who was ranked as the most negative school. Cornell was ranked as the 2 most negative Ivy League, which we feel is an accurate score. While these results are interesting, we cannot call them conclusive because of the varying number of yaks retrieved for each location. For example, Dartmouth had 36 unique top yaks to test after multiple hours of pulling data, while Cornell had over 460 unique top yaks, downloaded over the same time frame. This effect can be attributed to the usage of Yik Yak at certain schools. Given the limited use at some universities, would need to wait for the app to become more popular at the schools for us to have a more accurate comparison, a task that can be completed in the future.



College	Positive	Negative
Dartmouth College	0.7660	0.2340
Princeton University	0.6968	0.3032
Brown University	0.6770	0.3230
Harvard University	0.6659	0.3341
Yale University	0.6567	0.3433
Cornell University	0.6564	0.3436
Columbia University	0.6549	0.3451

5 Conclusion

The Biggest outcome from the project was the creation of a sentiment analysis classifier that adequately judged our inputted Yak data in linear time, determining positive and negative sentiment, achieving 67% accuracy. Also our classifier is is not limited to Yaks sentiment analysis. The program can be applied to any inputted text. What is exciting is our group is one of the first to apply sentiment analysis to Yik Yak. Yik Yak provides a unfiltered anonymous feed, directly from the students, and represent the current mindset on campus. If we take a moment to think about further applications of our approach, opportunities are present.

Imagine if the system ran every day, all day and night. It would continue to build profiles for schools, measuring the current sentiment of the University, presenting us with information on the days/weeks where negativity is highest for their students. Schools can then take action to help relieve the students of stress, and view the effects of their campaigns live through the data collected on Yik Yak. What this creates is a live feedback loop for universities to use to help students maintain positive attitudes on campus, and keep student moral high. While this is one application, there are many more that can be imaged.

As AI becomes ubiquitous, applications that require understanding human sentiment and empathy will increase in accuracy and speed. With this, comes the possibility to apply these applications to new data sets gaining a broader range of knowledge about a group?s sentiment. These application will enable the discovery of human trends and trends we have yet to find. To say exactly where affective computing will go is impossible, but we are sure it will be a subject of great interest for both enterprise, research, and CS 4700 students in the coming years. We?re excited (or classification: ?Positive?) to see where it will lead us.