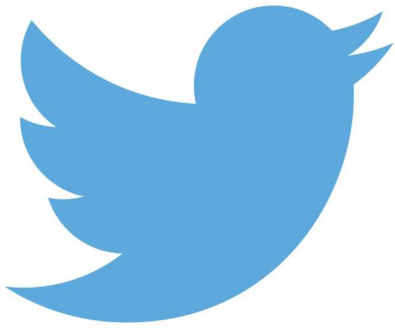
Cpts 315 Project Report:

Text Sentiment Classification Using Tweet Data



Trent Bultsma

I Introduction

Often it is useful for the things people say on the internet to be categorized into the attitude behind their words. This type of data analysis could be in something like a political campaign, being able to monitor how people feel about a certain candidate to influence where or how to target advertising in certain areas, or in things like product reviews or the way people talk about merchandise online. Companies can use this data to identify shortcomings in their products and reach out to customers who had problems with it to research how to improve their designs. The question here is: "how do we tell what people feel about something by what they say?" This is a topic I am interested in because it is a way that computer science overlaps with the social aspect of being in a society and the way that people express themselves, which has always been something that has intrigued me. Through this project I wanted to bridge the gap between people and computers and create something that could mirror the human experience back to the user a little bit. It was challenging to get past the learning curve of the machine learning library I used, scikit-learn, because it was my first experience with the library, but I found lots of helpful documentation and examples online that helped me through it. The result of my project was that I was able to create a system which identified the testing data I provided with around 97% accuracy. Also, I produced a system where I could then input my own phrases manually to see how people feel about those words using my machine as a reflection of human expression, which is fascinating to me.

II Data Mining Task

The premise of this project from a technical perspective is to classify the sentiment of text, namely a tweet, as either being positive, negative, litigious (meaning like a complaint or accusation), or unknown. The input data I used to solve the problem included blocks of text which were each tagged with one of these four labels as well as a label for the language of the tweet. Since there were four distinct categories to sort the data into, this is known as a multiclass classification problem. In addition to categorizing the data, I also set out to examine three different methods of categorization, those being a support vector machine, a naïve Bayes approach, and using a decision tree, to determine which was the most effective in categorization. The output of data mining included a percentage accuracy of predictions on the 20% of the tweet data I set aside for testing as well as the time it took to perform the calculations. I also included an interactive user interface to submit manually input test data that would be categorized and displayed to the user which category their sentence or phrase best fit into. The data mining questions I set out to investigate or examined along the way included asking how to categorize data into the categories of sentiment, which machine learning method is the most effective, whether it is helpful to remove artifacts of the text data coming from tweets (like URLs or hashtags), and if removing stop words such as "and" or "then" would prove beneficial. Some of the key challenges I faced were learning the classes/functions of the scikit-learn library and determining how to preprocess the training data in an efficient way.

III Technical Approach

I solved the data mining task by first preprocessing the input data, splitting it up into training and test data, looping through the different classifiers I was testing, training the classifier, testing the classifier with the test data, then displaying the results. After this, it took the best classifier and prompted the user input to classify via the console. Here is some pseudocode to illustrate this process:

```
read input data from csv
remove from the input data all words starting with "@" or "#" or "http" using regexes
filter out only the English tweets
training_data, test_data = split the preprocessed input data randomly 80% 20%
best_classifier = none
best_classifier_accuracy = 0
for classifier in SVM, NB, decision_tree:
        start_time = current_time()
        classifier.train(training_data)
        accuracy = classifier.predict(testing_data)
        total_time = current_time() – start_time
        print(accuracy and total time)
        if accuracy > best_classifier_accuracy:
                best_classifier = classifier
```

```
                best_classifier_accuracy = accuracy
while forever:
        sentence = ask user for a sentence to test
        print(best_classifier.classify(sentence))
```

Regarding the challenges I had to overcome for this project, I learned to use the scikit-learn library by reading documentation and looking at code examples on articles online. The much more difficult challenge was to filter out or sanitize my input data in a time effective manner. Initially, I had intended to go through every tweet and remove words that started with "@," "#," or "http," using loops and string comparison functions. This proved to be far too inefficient though, and after a few minutes only got through a few thousand of the roughly one million tweets so I knew that would not be sufficient. I eventually landed on a solution of reading the whole csv file containing the data in at once and removing those words using regular expressions, which were able to clean up the whole dataset in only a matter of seconds.

IV Evaluation Methodology

The dataset used for this project was gathered off Kaggle, which is an online platform that stores datasets for data mining projects. The dataset is named "Sentiment Dataset with 1 Million Tweets" on the platform and I didn't run into any specific challenges with it besides struggling to read it as a string until I realized I needed to use utf-8 encoding, which wasn't a very severe issue. The main metric I used to evaluate the solution to the data mining task was the accuracy of predictions. This was acquired by asking the classification machine to classify each of the test data points and then comparing those classifications to the labels of those test data points from the original dataset to see if the machine got the prediction right. Evaluating based on accuracy is important because if data is not labeled correctly, the classification system is not useful. A secondary metric for evaluating the success of the project was the time it took for the program to run. If a solution took more than a few minutes, it was considered unresponsive and ineffective. Differences in time efficiency in the matter of seconds were considered relatively inconsequential, but a big percentage improvement in time was seen as indicative of improvement that would be seen with a larger dataset, so it was considered somewhat when evaluating the solution. Time isn't a huge concern in the real world as long as the time commitment is reasonable and not taking days and days because data only needs to be processed once as long as the dataset is the same, but it is important for me practically because I don't have all day to work on my project sitting around waiting for it to run.s

V Results and Discussion

To start the project, I began by downloading the dataset from online and loading the csv file data into a python script. Then I started playing around with the naïve Bayes classifier built into scikit-learn, trying to figure out how to train the machine. I realized I needed to split up my data into test and training data, so I found a function to do that included in the library. Next, I researched how to input sets of strings into my classifier since typically classifiers work based on numbers. I discovered that I needed to use a count vectorizer for the classifier to efficiently create a bag of words style feature vector (like the one we created for the fortune cookie programming assignment). Doing it without the count vectorizer would require creating

almost a million vectors of a size equal to the number of unique words in all the tweets, which would require a huge amount of memory and isn't feasible. Next, I went about setting up and testing a bunch of other classifiers and settled on two others to use for my project, those being a support vector machine and decision tree classifier since they were the ones that were able to run on my data without taking at least ten minutes. Once I got a basic solution that was able to make predictions with these three classifiers, I went through the data and experimented with some methods of preprocessing the dataset. First, I looked at removing hashtags, tagging people, and URLs. I thought this may be beneficial because often users people reply to have random names that don't really have anything to do with the sentiment of the tweet and also I wanted to decouple the data a little bit from being linked to twitter by removing those artifacts, allowing it to be a more general solution to other types of text input. After implementing the change, I ran tests in quick succession on the same machine without changing any processes being run, a few times in a row with times being within +/- 0.5 seconds and the same accuracy (due to randomization seed being the same). With sanitization, the time improved by an average of around 17%, dropping the SVM accuracy by less than a percent, increasing the naïve Bayes approach by over 2 percent, and not really impacting the decision tree. I decided to use this sanitization method because of the pretty significant time improvement with changes to the accuracy being only slightly detrimental to a little bit beneficial. I also experimented with stop word removal, or the removal of words like "and" or "then," which don't impact the meaning of the sentence. The experiment was carried out in the same way as before but wasn't found to be beneficial. This is because the SVM got 1.5% worse in accuracy with a negligible improvement to time, the naïve Bayes saw an almost 3% hit to accuracy with negligible improvement to time, and then the decision tree did see a good time improvement of 16% but its awful accuracy was unaffected. I decided not to stick with this method of data preprocessing because of the negative impact to accuracy on the two actually good classifiers and negligible impact to time.

No preprocessing (except filtering out only the English tweets)

Classifier	**Accuracy (percent)**	**Time (seconds)**
Support Vector Machine	97.51	44.9
Naïve Bayes	80.45	31.19
Decision Tree	38.35	41.11

Twitter artifacts removed

Classifier	**Accuracy (percent)**	**Time (seconds)**
Support Vector Machine	96.86	38.63
Naïve Bayes	82.77	25.42
Decision Tree	38.33	33.00

Stop words removed

Classifier	Accuracy (percent)	Time (seconds)
Support Vector Machine	95.37	38.48
Naïve Bayes	79.93	24.02
Decision Tree	38.33	27.56

VI Lessons Learned

Through this project, I learned to use the scikit-learn library and realized that doing so is much more efficient than writing straight python code (gone are the days of 5-to-9-hour program runtimes like I experienced for one of the previous homework assignments). The program optimization is very good on scikit-learn's built in stuff, so I plan to use it for future data mining projects. I also learned the meaning of the term "litigious," which was one of the labels for the tweet data and refers to statements that are accusatory in nature, often including news related tweets. In hindsight, I wish I would have spent more time testing code on my desktop computer instead of going to the library to work on the project. I discovered that the improvement in performance on my desktop compared to my laptop does actually make a big difference and I would have saved a lot of time had I not been using my laptop for as long.

VII Acknowledgements

I used the following websites for assistance in completing the project:

https://www.kaggle.com/datasets/tariqsays/sentiment-dataset-with-1-million-tweets

https://www.datacamp.com/tutorial/naive-bayes-scikit-learn

https://scikit-learn.org/stable/modules/naive_bayes.html

https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

https://docs.python.org/3/library/time.html

https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f

https://medium.com/swlh/multi-class-text-classification-using-scikit-learn-a9bacb751048

https://stackoverflow.com/questions/3451111/unzipping-files-in-python