**Naïve Bayes Twitter Sentiment Analysis**

**Introduction**

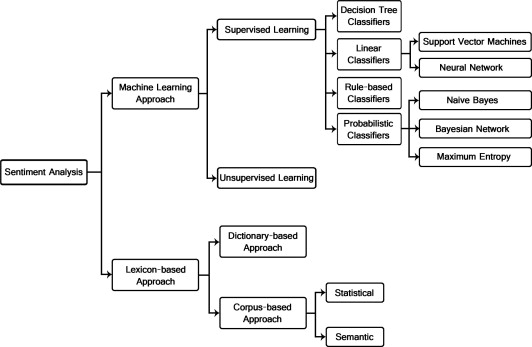
Sentiment Analysis, simply put, analyzes a String and tells whether or not that String has a positive or negative connotation attached to it.

This topic deserves attention because many companies use sentiment analysis tools on the web to track the public opinion of their products, services, or organization. Such tools could help businesses adjust and target markets where they do not have a strong foothold. In the end, knowing what tools to use and how to apply them could make or break a business.

Methods used for Sentiment Analysis

At first we looked into implementing a linear classifier, specifically, a neural network.[[1]](#footnote-1) However, after doing research on the topic, we concluded that, given our time constraint and seeing as everyone on our team has some probabilistic and statistical background, it would be more reasonable for us to use a probabilistic approach and implement the Naïve Bayesian classifier.

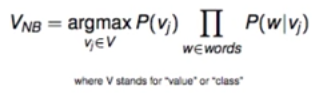
Using a supervised machine learning probabilistic classifier, our team shows that we are generally able to determine the sentiment of text. We say “generally” because no learning algorithm is 100% effective. That being said, our team trained our Naïve Bayes classifier on datasets comprised of a two class, positive and negative, system. By extending the Naïve Bayesian classifier to look at twitter we hope to find some clarity and determine the general consensus in the overabundance and oversaturation of news to which we are exposed.



**Naïve Bayes**

A Naïve Bayes classifier works by applying Bayes Theorem in a supervised manner. The classifier is trained on a pre-classified dataset to allow the classifier to learn what words comprise a positive block of text and what words comprise a negative block of text. These pre-classified samples allow us to build up a bag of words for each class presented in the training set.

Since the Naïve Bayes classifier calculates the probability that a given statement belongs to each class, the position of the words in the block of text and its immediate context is negligible. Ignoring the word’s position in the text may seem counterintuitive, but this approach actually works quite well. [[2]](#footnote-2) Each word, given a class, gets assigned a probability.



Naïve Bayes Formula

Stated above is the general formula that is used to classify new data. In this formula, *v* is the class (positive or negative) and *w* is the word in a given text. The value of the text is thus classified by the maximum, normalized, probability computed by aggregating the product of each word in the text, given each class.

We designed our Naïve Bayes classifier to have a few more features than just the bare-bones classifier. We gave the user the option of choosing whether or not they want the classifier to continue learning, to discriminate between what it learns and what it does not learn, to use an English language stemmer, or to write newly classified data back to its training set (so as to not lose any progress).

If the classifier continues learning, each word in the block of text is added as a key value pair to the given class’s vocabulary. Therefore, if this word appears in a new block of text, the newly learned context will add more weight to the given class. As a safety measure, to minimize the chance of learning a word incorrectly (say a bad word is used in a positive context), we decided to only add the word to the dictionary if the classifier is at least 90% certain of the classification. Our team chose 90% since it was empirically the best choice. Anything less than or greater than 90% had diminishing returns.

To deal with stop words and strange characters found in tweets, the classifier removes the stop words and symbols from the sentence so as to not skew the probability. Lastly, our team implemented an English language stemmer to converge multiple tenses of a given word to one word in the class vocabularies. [[3]](#footnote-3) This serves to strip away the unnecessary words, characters and suffixes to allow the vocabulary to only include relevant words.



Porter Stemmer Examples

This pre-processing of words greatly improved the accuracy of the classifier since the vocabulary was more revised.

**Twitter Integration**

The Sentiment Classifier is a standalone classifier that can work in many contexts. However, for this paper we integrated it with a simple PHP twitter aggregator, so that we have an easy way to collect relevant items to classify.

Using an open source PHP Twitter API Wrapper, we host a simple web app on an apache server to aggregate tweets by either a Twitter handle or Twitter hashtag. [[4]](#footnote-4) Along with choosing to search by a Twitter handle or hashtag, the user has the option to specify the number of tweets he or she wants sourced. Via the twitter API, we performed data filtering to ensure that the tweets we source have enough content for us to classify. This preprocessing allowed us to source tweets in English that are most relevant to the user’s query and ignore any Emojis/symbols (i.e. non-English characters). Once Tweets are sourced via the query in a web GUI, we populate a text file with just the contents of the Tweets. This text file is then passed into the Naïve Bayes classifier for classification.

**Results**

After revising our original proposal, we settled on aiming for an accuracy of 65% when classifying statements. At first we were aiming to accomplish at least 80%, but had no support to back up that claim.  After looking at similar projects and seeing how unrealistic our first goal was, we revised our definition of success. [[5]](#footnote-5)

Under ideal circumstances, the classifier has the capability of achieving ~77% accuracy. The set is ideal because it does not contain neutral text. [[6]](#footnote-6) Since the training set does not contain any neutral text, our classifier can perform quite accurately. Our Naïve Bayes classifier does not do well in the case where it is exposed to neutral texts because we only consider the classifications of positive or negative text. That is not to say that our classifier cannot handle the neutral class (given adequate data), but it has not been trained to do so.

It is worth stating, for clarity, that we used two distinct datasets for training and testing. Each of these two sets are pre-classified in a consistent manner. Therefore, our classifier can immediately check the classifier’s efficacy.

The training set consists of 3000 pre-classified data points, mainly consisting of movie related sentences. Whereas the testing set used for our accuracy analysis has 7000 pre-classified data points lines.

Checking the classifier’s efficacy against Twitter data is slightly more challenging. In real life, not every block of text can be neatly categorized as either positive or negative. There are many degrees between the two. That being said, our implementation of the Naïve Bayes classifier seems to empirically meet our goal of reaching at least 65% accuracy. Furthermore, we noticed that our classifier is more effective and accurate when it is set to the learning mode.

When using the classifier the user has the option to allow the classifier to keep learning as it encounters new sentences. It will update the probabilities for words already in it’s vocabulary. If new words reach a confidence threshold value, they will be added to the vocabulary too. As you can see below, the continued learning does improve proficiency slightly and works better as testing sets become larger.

The base case for testing accuracy was after training the classifier with the original training set, and evaluating the testing set with learning off. This configuration achieved a 75.90% accuracy. When this same test was performed with learning was turned on, we obtained an accuracy of 77.08%. Likewise, performing the same experiment after running hundreds of tweets through the classifier we achieved an accuracy of 77.25%.

**Project Evolution**

Currently, our Naïve Bayes classifier is only being trained on a two-class system - positive and negative. Koppel and Schler (2006) in a research paper presented their findings that a neutral class can improve overall accuracy of a classifier conducting sentiment analysis. [[7]](#footnote-7) Therefore, in the future, we would invest more resources into vetting more training data to cover all edge cases and incorporate all the classes necessary to adeptly represent real-life situations. In addition to expanding our dataset to a three-class system, we would also spend time expanding our dataset to include Emojis. Often times, Emojis are used to express sentiment more so than words. That being said, there are libraries that parse Emojis into text so that we can process them in Java.[[8]](#footnote-8) Therefore, on the Twitter interface portion of this project, our next step would be to include Emojis in the filtering.

Rather than improving on our Naïve Bayes classifier, our team could abandon the probabilistic approach and take the linear classifier approach instead. *The Stanford Natural Language Processing Group* achieved 85.4% accuracy using the same two-class (positive/negative) system that our team used, but implemented a Recursive Neural Tensor Network. [[9]](#footnote-9)

**Conclusions**

In conclusion, there still is room for improvement. The main areas that we could focus on improving are adding Emojis and a neutral class to the training set. However, since our goal is to maximize performance in sentiment analysis, adopting other approaches, such as linear classifiers (e.g. Artificial Neural Networks, Support Vector Machines), would yield better results.

With that said, our team was able to successfully create a Naïve Bayes classifier to accurately classify text in an ideal environment and adeptly classify text in a non-ideal environment, such as Twitter. We did so, surpassing our expectation of 65% by ~12% (i.e. achieving an accuracy score of ~77%) on a dataset from a University of Michigan sentiment analysis competition.

**Appendix**

* Abbeel, Pieter. “Lecture20: Machine Learning: Naive Bayes.” CS188Spring2013. Youtube. 10 April, 2013, accessed on 20 November 2016.

<https://www.youtube.com/watch?v=DNvwfNEiKvw&t=438s>

* Durmont, Vincent. "Vdurmont/emoji-java." *GitHub*. N.p., 15 Nov. 2016. Web. 16 Dec. 2016.

<https://github.com/vdurmont/emoji-java>

* Porter. M.F. “An algorithm for suffix stripping.” Version 3. 1980. Online. <https://tartarus.org/martin/PorterStemmer/def.txt>
* Mallison, James, “Simple PHP Wrapper for Twitter API v1.1 calls” (2015), GitHub repository

<https://github.com/J7mbo/twitter-api-php>

* Medhat, Walaa, Ahmed Hassan, and Hoda Korashy. "Sentiment Analysis Algorithms and Applications: A Survey." *Sentiment Analysis Algorithms and Applications: A Survey*. Ain Shams Engineering Journal, 27 May 2014. Web. 16 Dec. 2016.

<http://www.sciencedirect.com/science/article/pii/S2090447914000550>

* Kouloumpis, Efthymios. Wilson, Theresa. and Moore, Johanna. “Twitter Sentiment Analysis: The Good the Bad and the OMG!” 2011. Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media. Online. <http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/viewFile/2857/3251>
* Training Set, UMICH SI650 - Sentiment Classification, 2011, Online. Michigan University

<https://inclass.kaggle.com/c/si650winter11/data>

* Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. "Deeply Moving: Deep Learning for Sentiment Analysis." *Deeply Moving: Deep Learning for Sentiment Analysis*. Stanford, Aug. 2013. Web. 16 Dec. 2016.

<http://nlp.stanford.edu/sentiment/>

* Vryniotis, Vasilis. "The Importance of Neutral Class in Sentiment Analysis." Machine Learning Blog & Software Development News. N.p., 23 Sept. 2013. Web. <http://blog.datumbox.com/the-importance-of-neutral-class-in-sentiment-analysis/>

1. [Sentiment Analysis Approaches](http://www.sciencedirect.com/science/article/pii/S2090447914000550) [↑](#footnote-ref-1)
2. [CS 188 Lecture](https://www.youtube.com/watch?v=DNvwfNEiKvw&t=438s) [↑](#footnote-ref-2)
3. [Porter Stemmer](http://www.tartarus.org/~martin/PorterStemmer/index.html) [↑](#footnote-ref-3)
4. [Twitter API PHP](https://github.com/J7mbo/twitter-api-php) [↑](#footnote-ref-4)
5. [Twitter Sentiment Analysis](http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/viewFile/2857/3251) [↑](#footnote-ref-5)
6. Dataset is from [UMichigan](https://inclass.kaggle.com/c/si650winter11/data) [↑](#footnote-ref-6)
7. [Neutral Class - Sentiment Analysis](http://blog.datumbox.com/the-importance-of-neutral-class-in-sentiment-analysis/) [↑](#footnote-ref-7)
8. [Emoji Parser](https://github.com/vdurmont/emoji-java) [↑](#footnote-ref-8)
9. [The Stanford Natural Language Processing Group](http://nlp.stanford.edu/sentiment/) [↑](#footnote-ref-9)