Attribute Based Sentiment Analysis

Bedi Rahatpal, Thakkar Vishwas, Ojas Thanawala

Northeastern University, Boston MA

Email: bedi.ra@husky.neu.edu | thakkar.vi@husky.neu.edu | thanawala.o@husky.neu.edu

Abstract

The aim of this project was to perform effective and reliable sentiment analysis using Natural Language Processing and Supervised Learning. Attribute Based Sentiment Analysis (ABSA) is the sentiment analysis of a particular aspect of a subject. For this paper we have used data from two sources: Twitter and Amazon. The idea is that, using twitter one can get the idea of public's general sentiment or opinion about any subject of discussion; while on other hand using Amazon reviews one can get an idea about how good a particular product is. Hence, after fetching the data from a relevant source (i.e. tweets through twitter API and amazon reviews from Amazon API) we use a Part-of-Speech (POS) tagger to split the given text in words and rank them according to their relevance using the stem module in the NLTK package. Then we use multiple classifiers to determine the sentiment in the text. Each classifier gets different result depending on their accuracies. We take the mode of all the results obtained by classifiers as the final result. The multiple classifiers are used to increase the efficiency and reliability.

Introduction[1]

People post real time messages on websites like twitter and Amazon about their opinions on variety of subjects, discuss, complain and express positive sentiments for products they use in daily life. These messages are a tremendous value to companies manufacturing products or anyone who needs to know the common sentiment about a subject. Hence, to detect and summarize a sentiment about something is a very important aspect of any industry.

The formal definition of Sentiment analysis is "Computationally identifying and categorizing opinions in a piece of text especially to determine the writer's attitude towards a topic, product, etc. is positive, negative or neutral". Sentiment analysis is widely used in product development and customer support by companies. Fluctuations in stock markets can be predicted using it and even upto some extent whether a politician who is running for presidency will win or not.

But sentiment analysis is not a complete solution to these problems. We only get an overall analysis of the subject i.e. we can only know if the subject, as a whole, is reviewed good or bad. What if we want to know the sentiment about a specific aspect of the subject? We have attempted to solve this problem in this project. Attribute Based Sentiment Analysis is the sentiment analysis on a specific attribute of the subject. For example: the user wants to buy a specific new phone. But he dont know how is the battery life of that phone. He/she can use ABSA to see the sentiments on battery of that phone and make a decision on whether to buy that phone or not.

In this paper, we built a model for classifying statements into positive, negative and neutral by using four types of classifiers: Naive Bayes classifier, Multinomial Naive Bayes, Logistic Regression and NuSVM. Among these classifiers, the highest accuracy was of NuSVM and lowest of Regular Naive Bayes. There are many papers with

models consisting only one classifier. But to improve the reliability and maximize the accuracy we have used several classifiers. For training, we used 10,000 manually annotated tweets and about 4,000 tweets for testing the model.

The remainder of the paper is organized as follows: The next section, section-2, contains the methodology used for data acquisition and classification algorithms being used. The section after that contains the experiments we performed and the results we got.

Methodology

In this project, our goal was to improve the traditional sentiment analysis and make the analysis more useful and reliable. To do this we have to give reliable data to the model for training and testing. In the following section we present our approach to collect the data and discuss the modelling techniques used.

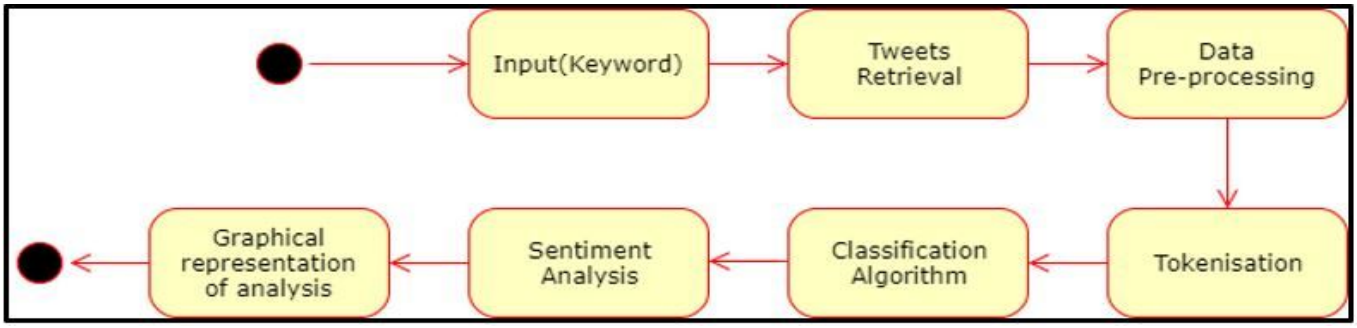


Figure-1 Overview of methodology behind model construction.

Data Source

We are using two primary sources of data:

1) The Twitter API
2) Amazon Product API

Using a keyword, one can fetch tweets and product reviews respectively from these APIs. The data will contain number of tweets related to the keyword entered by the user and on the other hand, in case of amazon API, the data will be the product reviews. For the training and testing of the model, we have used just the tweets.

Data Pre-processing

Extricating tags from a content archive includes no less than three stages: part the record into words, gathering variations of a similar word, and positioning them as per their significance. These three undertakings are completed individually by the Reader, Stemmer and Rater classes, and their work is assembled by the Tagger class.

A Reader item may acknowledge as information a report in some arrangement, play out some standardization of the content, (for example, transforming everything into lower case), investigate the structure of the expressions and accentuation, and return a rundown of words regarding the request in the content, maybe with some extra data, for example, which ones look like formal people, places or things, or are toward the finish of an expression. A clear method for doing this would be to simply coordinate every one of the words with a regular expression, and this is for sure what the SimpleReader class does.

The Stemmer endeavors to perceive the root of a word, so as to recognize marginally extraordinary structures. This is now a very entangled errand, and it's plainly language-explicit. The stem module in the NLTK bundle gives algorithms to numerous dialects and incorporates pleasantly with the tagger as appeared in the figure-2 underneath.

The Rater takes the rundown of words contained in the report, together with any extra data assembled at the past stages, and returns a rundown of labels (for example words or little units of content) requested by some thought of "relevance".

Things being what they are, simply chipping away at the data contained in the report itself isn't sufficient, in light of the fact that it says nothing regarding the recurrence of a term in the language. Therefore, a right on time "disconnected" period of the calculation comprises in

examining a corpus (for example an example of reports written in a similar language) to manufacture a lexicon of known words. This is taken consideration by the build_dict() work. It is encouraged to assemble your very own lexicons, and the build_dict_from_nltk() work in the additional items module empowers you to utilize the corpora incorporated into NLTK.

(characterized in the module additional items) rather than the standard one.

The multitag_size parameter in the Rater's constructor characterizes the greatest number of words that can establish a tag. Multi Tags are produced in the create_multitags() method; if extra data about the situation of a word in the expression is accessible (for example the

```
import nltk
# an English stemmer using Lancaster's algorithm
mystemmer = Stemmer(nltk.stem.LancasterStemmer)
# an Italian stemmer
class MyItalianStemmer(Stemmer):
    def __init__(self):
        Stemmer.__init__(self, nltk.stem.ItalianStemmer)
    def preprocess(self, string):
        # do something with the string before passing it to nltk's stemmer
```

Figure-2

So far, we may define the relevance of a word as the product of two distinct functions: one that depends on the document itself, and one that depends on the corpus. A standard measure in information retrieval is TF-IDF (*term frequency-inverse document frequency*): the frequency of the word in the document multiplied by the (logarithm of) the inverse of its frequency in the corpus (i.e. the cardinality of the corpus divided by the number of documents where the word is found). If we treat the whole corpus as a single document, and count the total occurrences of the term instead, we obtain ICF (*inverse collection frequency*). Both of these are implemented in the *build_dict* module, and any other reasonable measure should be fine, provided that it is normalised in the interval [0,1]. The dictionary is passed to the Rater object as the *weights* argument in its constructor. We may likewise need to characterize the principal term of the item in an alternate manner, and this is finished by overriding the rate_tags() technique (which naturally ascertains TF for each word and increases it by its weight).

On the off chance that we were not very exacting about the outcomes, these couple of bits would as of now make a worthy tagger. In any case, labels shaped just by single words are very constrained: while "Trump" and "Donald Trump" are both sensible labels (and it is very simple to treat cases like this so as to see them as equivalent), having "Elon" and "Musk" as two separate labels are certainly not satisfactory and misdirecting. Look at the outcomes on a similar archive utilizing the NaiveRater class

terminal individual from the class Tag), this should be possible in an increasingly precise manner. The rating of a multi tag is figured from the evaluations of its unit tags. As a matter of course, the combined_rating() method utilizes the geometric mean, with an extraordinary treatment of formal people, places or things if that data is accessible as well (in the best possible part). This method can be overridden as well, so there is space for experimentation.

With a couple of "presence of mind" heuristics the outcomes are enormously improved. The last phase of the default rating algorithm includes disposing of repetitive tags (for example tags that contain or are contained in other, less important tags).

Related Work

The research work in the area of Sentiment analysis has been vast and extensive. Sentiment analysis has been implemented as a Natural Language Processing task at many levels of granularity. From some of the earlier work where in it was a document level classification task (Turney, 2002; Pang and Lee, 2004), it was handled at the sentence level (Hu and Liu 2004; Kim and Hovy, 2004) and then at the phase level (Wilson et al ., 2005; Agarwal et al., 2009).

Some of the more recents results on sentiment analysis of Twitter data are by Go et al.(2009), (Bermingham and Smeaton, 2010) and Pak and Paroubek (2010). Go et al. (2009) used distant learning to acquire sentiment data. They used emoticons with tweets wherein emoticons such as ":)" and ":)" are taken as positive tweets and emoticons

like ":(" as depicting negative emotion. They built the models using Naive Bayes , MaxEnt and SVM classifiers, wherein they report SVM is better than all other classifiers. For the features they have used Unigram , Bigram along with Part-of-Speech (POS) tagging. They note bigram and POS tagging does not help and that unigram features outperforms all other models. They also performed some pre-processing of the data as part of modeling the preprocessing techniques used by them. They also performed text processing which included the removal of URLs, username references and repeated characters in words.

Another recent result by Pak and Paroubek (2010) collects data following a similar distant learning paradigm. They perform a subjective vs objective task which is a different classification. For the former they collect tweets ending with emoticons. Whereas for the objective data they crawl twitter accounts of popular newspapers like "Washington Post", "New York Times" etc.

They report that POS and bigrams both help (contrary to results presented by Go et al. (2009)). Both these approaches, however, are primarily based on n gram models. Moreover, the data used for training and testing is biased as it is collected by search queries. In contrast, we present features that achieve a significant gain over a unigram baseline. In addition to this we explore a different method of data representation and report drastic improvement over the unigram models. Another notable contribution of this paper is that we report results on data that is manually annotated and that does not suffer from any known biases.The data input is not data that is collected by using specific queries rather it is a random sample of streaming tweets. The size of our hand-labeled data allows us to perform cross- validation experiments and check for the variance in performance of the classifier across folds.

The sentiment classification on Twitter data by Barbosa and Feng (2010) is another significant effort.. They make use of polarity predictions from three websites wherein they use noisy labels to tune and train a model. They use 1000 manually labeled tweets for the former and another 1000 manually labeled tweets for training it. However they do not mention any ways by which they collect their test data. They propose the use of syntax features of tweets like retweet, hashtags, link, punctuation and exclamation marks in conjunction with features like prior polarity of words and POS of words. By combining prior polarity with POS, we extend their approach by using real valued prior polarity.. Our results show that the features that enhance the performance of our classifiers the most are features that combine prior polarity of words with their parts of speech. The tweet syntax features help but only marginally.

Gamon (2004) perform sentiment analysis on feedback data from Global Support Services survey. Analysing the role of linguistic features like POS tags is one their aims. They perform extensive feature analysis and feature selection and demonstrate that abstract linguistic analysis features contributes to the classifier accuracy. In this paper they performed extensive feature analysis and show the use of only 100 abstract linguistic features performs as well as a hard unigram baseline.

A survey report from Pang and Lee on Opinion mining and sentiment analysis [4] gives a comprehensive study in the area with respect to sentiment analysis of blogs, reviews etc. Algorithms used in the survey include Maximum Entropy , SVM and Naive Bayes.

Since twitter data is noisy as it has a lot of slangs and short words, some of the pre-processing techniques using the slang dictionary are mentioned in the paper Apoorva et al., along with it removal the technique of removing the stop words. They also use the emoticon dictionary which has been implemented in this project to be used in numeric features. They also implement a prior polarity scoring which scores many English words between 1(Negative) to 3(Positive). From the point of view of the algorithm they provide feature based models and a tree kernel. Unigram baseline model is combined with other features and modeled in this paper. Different combination of the features are selected. A list of emoticons and Part-of-speech tagging is used from wikipedia for the features

Modelling Techniques

After we apply the different steps of the preprocessing part, we then focus on the algorithms part. We have used 4 main algorithms in sentiment analysis to classify a sentence into positive or negative: Naive Bayes , Multinomial , SVM and Logistic Regression. SVM is known to give the best be the model giving the best results .

Naive Bayes

Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong independent assumptions between the features. Naive Bayes classifiers can be easily scaled, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum likelihood training can be done by evaluating a closed form expression (mathematical expression that can be evaluated

in a finite number of operations), which takes linear time. It is based on the application of the Bayes rule given by the following formula:

$$P(C = c|D = d) = \frac{P(D = d|C = c)P(C = c)}{P(D = d)}$$

We can simplify this expression by,

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$

In our case, a tweet d is represented by a vector of K attributes such as d = (w1, w2, ..., wk) . Naive Bayes introduces the assumption that all of the feature values wj are independent given the category label c as computing P(d|c) is not trivial and that is why. That is, for i = j , wi and wj are conditionally independent given the category label c . So the Bayes rule can be rewritten as:

$$P(c|d) = P(c) \times \frac{\prod_{j=1}^{K} P(w_j|c)}{P(d)}$$

Based on this equation, maximum a posterior (MAP) classifier can be constructing by seeking the optimal category which maximizes the posterior P (c|d) :

$$c* = arg\max_{c \in C} P(c|d)$$

$$c* = arg\max_{c \in C} \left\{ P(c) \times \frac{\prod_{j=1}^{K} P(w_j|c)}{P(d)} \right\}$$

$$c* = arg\max_{c \in C} \left\{ P(c) \times \prod_{j=1}^{K} P(w_j|c) \right\}$$

Note that we remove P(d) since it is a constant for every category c. In addition to this we also have used a variation to the naive bayes which is multinomial naive bayes.

The Multinomial Naive Bayes

Typically used for discrete counts. In text classification, we extend the Bernoulli model further by counting the number of times a word w_i appears over the number of words rather than saying 0 or 1 if word occurs or not.

SVM

SVM or Support vector machine is an algorithm which helps us classify both linear and nonlinear data. A support vector machine constructs a hyperplane or set of hyperplanes in a high-dimensional space such that the separation is maximum. This is the reason the SVM is also called the maximum margin classifier. The hyperplane identifies certain examples close to the plane which are called as support vectors. LinearSVC from sci-kit learn, which is a python package, is used to classify the tweets.

If the input data is linearly separable, the SVM that separates data of one class from another by searching for a linear optimal separating hyperplane (the linear kernel), which is a decision boundary. Mathematically, a separating hyperplane can be written as:

W·X+b=0, where W is a weight vector and W=w1,w2,...,w n. X is a training tuple. b is a scalar.

In order to optimize the hyperplane, the problem essentially transforms to the minimization of $\|W\|$, which is eventually computed as:

$\sum_{i=1}^{n} \alpha_i y_i x_i$, where α_i are numeric parameters, and y_i are labels based on support vectors, X_i .

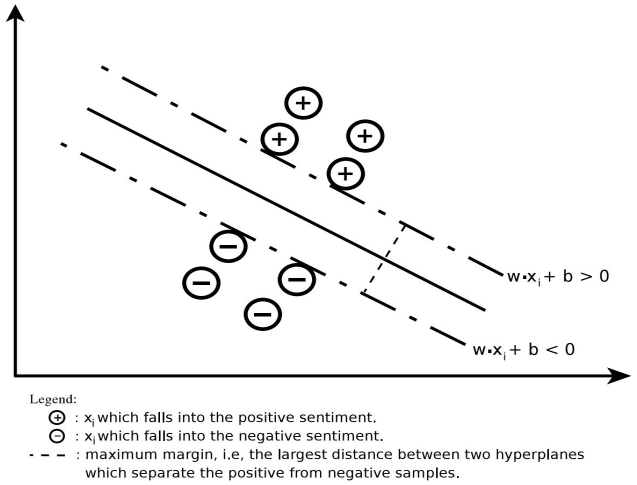
That is: if y_i=1 then

$$\sum_{i=1}^{n} w_i x_i \geq 1;$$

if y_i=−1 then

$$\sum_{i=1}^{n} w_i x_i \geq -1.$$

The figure below shows a representation of SVM method.



Logistic Regression

We also used logistic regression as it predicts the probability of an outcome that can only have two values

(i.e. a dichotomy). The prediction is calculated by the use of one or more predictors (numerical and categorical). The reason for choosing not choosing linear regression is for two reasons:

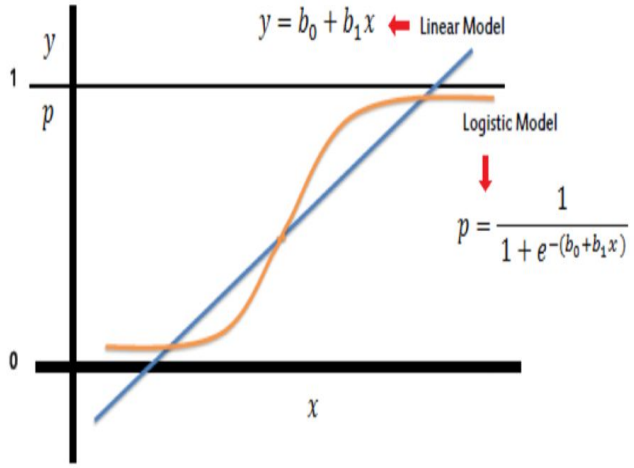
- A linear regression will predict values outside the acceptable range such as for example. predicting probabilities outside the range 0 to 1.
- Since the divided experiments can only have binary value ie one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line.

Whereas, a logistic regression produces a logistic curve, that is limited to values between 0 and 1. Logistic regression has similarities to a linear regression, but rather than the probability , the curve is instead constructed using the natural logarithm of the "odds" of the target variable. Moreover, unlike with linear regression the predictors do not have to be normally distributed nor is it required to have equal variance in each group.

Logistic regression makes use of maximum likelihood estimation (MLE) in order to obtain the model coefficients that relate the predictors to the target. After this initial function is calculated, the process is repeated until there isn't a significant change in LL (Log Likelihood).

$$\boldsymbol{\beta^1 = \beta^0 + [X^T W X]^{-1}.X^T(y-\mu)}$$

β - is a vector of the logistic regression coefficients.
W - is a square matrix of order N with elements $n_i \,.\, \pi_i(1-\pi)_i$ on the diagonal and zero elsewhere.
μ - its a vector of n elements $\mu_i = n_i \,.\, \pi_i$



Experiments

Implementation:

The various back-end technologies used for experimentation include Python Web Framework (Django), Machine Learning libraries and tools API, Twitter API, Amazon Product API (Product reviews), Rotten Tomatoes API for Movie Reviews (only for experiment purposes, not part of the project), NLTK Library, Scikit-learn Library, Tweepy/ Twython. These configurations and were implemented with a set of front-end technologies like HTML5, CSS3 (Bootstrap) and JavaScript (jQuery).

Our goal was to achieve maximum possible accuracy for each Algorithm. The user should be able to download the analysis report in various file formats. The user should also be able to access his or her search history. At the same time, the design should not downgrade the performance of the algorithms so as to maintain the accuracy of all our experiments.

Data:

The data dictionary that we used for our experiments include Amazon's Product Reviews dataset (labelled dataset), Twitter posts and comments (labelled dataset) and IMDB's Movie Reviews dataset (only for experimental procedure. This was done so as to improve the accuracy of our algorithms. An extra dataset was used to confirm whether the algorithms were voting correctly for different words and their polarity). The dataset from Twitter contained a minimum of 10,000 tweets. We started with a minimalistic design where only 1000 tweets were considered. Once the algorithms started improving their accuracies, we scaled up to 10,000 tweets. Although 10,000 tweets is a rather small dataset considering the number of tweets currently available, our algorithms and classifiers should work equally accurately for a bigger dataset. This was done just to improve accurate testing of the data.

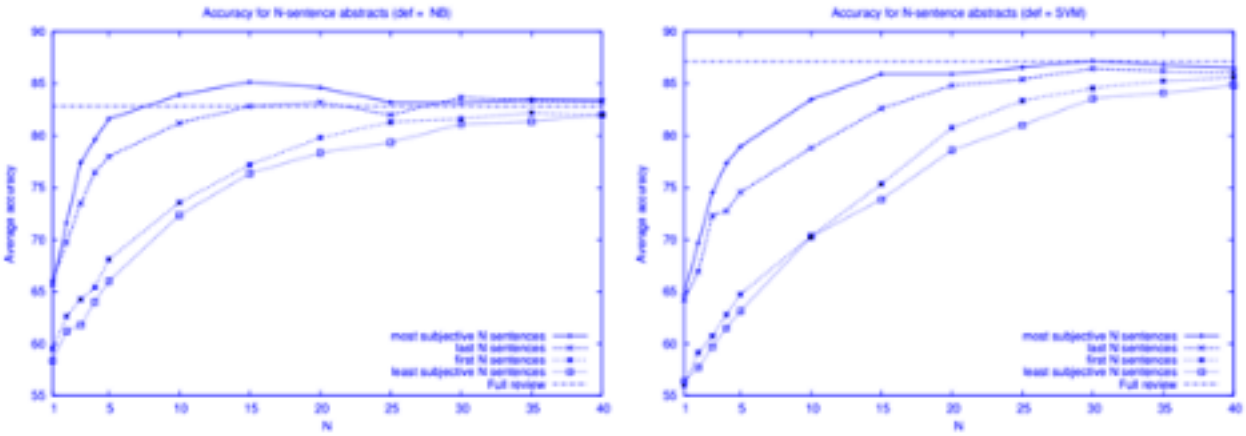


Figure-3: This diagram shows accuracies using n-sentence extracts for Naive Bayes (left) and nu-SVM (right) default polarity classifiers.

Results:

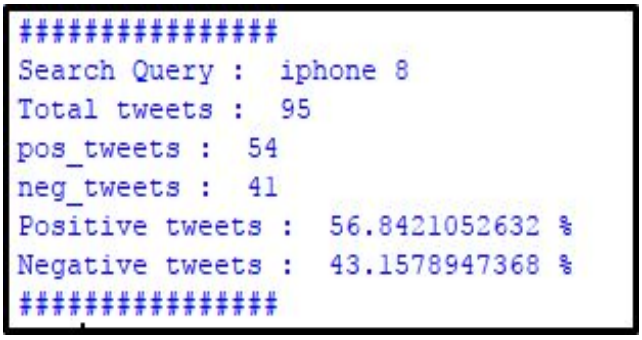
We achieved maximum accuracy using the Logistic Regression Algorithm. The nu SVM coupled with the Multinomial NB classifier also provided with a high accuracy rate. Even though implementing only Naive Bayes Algorithm resulted in a low accuracy rate, 4 other algorithms were used to improve the reliability of testing the data. Each algorithm was used to vote whether a trained dataset is positive or negative. An algorithm would evaluate to 1 if the dataset provided was positive, 0 if it was negative. 0 to 1 was kept as our rating limit. Each word would reside only in this limit.
This is the Diagram showing various informative attributes from a dataset:

We achieved the following accuracies for each of the algorithms used for implementation:

```
('Original Naive Bayes Algo accuracy percent:', 74.4)
('LogisticRegression_classifier accuracy percent:', 72.2)
('SGD_classifier accuracy percent:', 71.0)
('MNB_classifier accuracy percent:', 76.8)
('NuSVC_classifier accuracy percent:', 71.39999999999999)
```

A snippet of an attribute from the Twitter API shown below explains how the sentiment analysis is done for a specific feature. Here, “iphone 8” was the query. Tweets containing this query were fetched, trained and then passed to various classifiers. Each classifier voted if tweets related to this query were positive or negative. After combining results of all algorithms, a total negative and positive ratio was displayed. This snippet only contains the overall averaged ratio of all the classifiers. This does not contain a separate evaluation of each of the classifiers:

```
Most Informative Features
                   would = True              neg : pos    =      8.7 : 1.0
                    well = True              pos : neg    =      8.5 : 1.0
                   great = True              pos : neg    =      7.8 : 1.0
                     bad = True              neg : pos    =      7.6 : 1.0
                  script = True              neg : pos    =      7.5 : 1.0
                       - = True              neg : pos    =      5.1 : 1.0
                    plot = True              neg : pos    =      4.8 : 1.0
                    only = True              neg : pos    =      4.8 : 1.0
                    best = True              pos : neg    =      4.8 : 1.0
             performance = True              pos : neg    =      4.8 : 1.0
                  ending = True              pos : neg    =      4.8 : 1.0
                    even = True              neg : pos    =      4.7 : 1.0
                 feeling = True              neg : pos    =      4.5 : 1.0
                    game = True              pos : neg    =      4.1 : 1.0
                      10 = True              pos : neg    =      4.1 : 1.0
```



.

Future Enhancement:

The accuracy of the system will be improved in the future for more accurate analysis. Current attribute extraction methods assigns same weights for each word and ignores the structure of the sentence or context. To solve this issue, we will look deeper into learning vector representations of the words. Also, we want to know deeper in the logic gap between words on a psychological level.

Naive method isn't particularly compelling. We thought that the limited space permitted in tweets would allow for purely statistically based techniques, however the assumption seems to not hold this class of problem, we found a relatively less efficient performance. This indicates that our text representation isn't particularly the most optimal representation. A more intelligent method for attribute selection could lead to better results. We could possibly use unigram models for improving the efficiency of our system.

Conclusion

To summarise our work, we started out with a goal of providing a more detailed and accurate analysis of any product or tweet. We are providing segregation of reviews and tweets in terms of how different customers and users like or dislike an attribute.

The customers will now be able to get an appropriate analysis of the product based on the reviews by other users. Insight of the product alone with the quality of the individual attributes will be obtained, that will be beneficial to all the customers and stakeholders. This attribute based sentiment analysis will be helpful to a large number of users as they will be able to decide the quality of the product, or the correctness and validity of a tweet on the basis of product or tweet attributes.

There are a number of ways available for improving our methodology and accuracy. We can improve the manner in which we infer data from a review or a tweet, and the manner in which we signify the relevance of the topic of an attribute using labelled data.

References

[1]. Barbosa, L., Feng, J.: Robust sentiment detection on twitter from biased and noisy data. In: Proceedings of COLING, pp.36–44 (2010).

[2]. http://cs229.stanford.edu/projects2013.html

[3]. https://buffer.com/library/twitter-analytics

[4]. http://sentic.net/sentire2016ahlgren.pdf

[5]. http://www.cyberemotions.eu/rudy-sentiment-preprint.pdf

[6]. https://arxiv.org/ftp/arxiv/papers/1612/1612.01556.pdf

[7].https://thesai.org/Downloads/Volume6No9/Paper_30-Online_Paper_Review_Analysis.pdf

[8]. Mr. Akshay, A. Adsod, & Prof. Nitin R.C., "A review on web mining", International Journal of Engineering Trends and Technology (IJETT)", Volume 10 Number 3, 2014.

[9]. Dushyant, B.R., & Samrat, K., "A Review on Emerging Trends of Web Mining and IT's Application", International Journal of Engineering Development and Research | IJEDR, 2013.

[10]. Bing, L., Sentiment Analysis and Opinion Mining. Morgan & Claypool Publishers, 2012.

[11]. Yin, Z., Rong, J., & Zhi-Hua, Z., "Understanding Bag-of-Words Model: A Statistical Framework", International Journal of Machine Learning and Cybernetics, 2010.

[12]. Hang, C., Vibhu, M., & Mayur, D., "Comparative experiments on sentiment classification for online product reviews", American Association for Artificial Intelligence (AAAI Conference), 2006.