Sentiment Classifier of Movie Reviews

LING 406 Spring 2017 Final Project

Yifeng Chu
ychu26@illinois.edu

May 6, 2017

Abstract

This project is an exploration on sentiment analysis by employing machine learning techniques. The data used in the project is movie review dataset from Cornell. We first propose a baseline approach and improve the performance by extracting more informative features from the text. Although multiple machine learning models are compared in this paper, Naive Bayes Classifier is more thoroughly discussed.

1 Introduction

Due to the popularity of Internet, people are provided with numerous on-line platforms for social activities, where the most common way for communications is through text. They can leave comments for movies they watched and books they read, or provide reviews of products they bought to help other customers make decision, or simply share things in their daily life with others. For various purposes, people would like to identify the objectivity of texts and emotions expressed in the opinions. The analysis result would be of great value, for instance, to improve customer services and develop potential market.

2 Problem Definition

Generally, sentiment analysis aims to determine the attitude of a speaker or writer, which may include the objectivity, emotion state, or intended emotion communication of the content[5]. In the context of the Computational Linguistics, we attach certain attributes to a given text and assign values to evaluate the intensity or polarity of the attributes.

In this project, I consider a simplified version of sentiment analysis: the polarity of emotion on movie reviews, i.e., given a comment on a movie, my system outputs a binary value indicating whether the review expresses the positive or negative emotions.

3 Previous Work

I discuss three papers on sentiment analysis task. Section 3.1 is a overview of the work authors have done. Then, difference among their work is discussed in the following section.

3.1 Overview

3.1.1 Learning to Identify Emotion in Text

The first paper "Learning to Identify Emotions in Text", by Strapparava and Mihalcea studies the sentiment analysis problem in the most general sense, since the systems authors build can identify six emotions: ANGER, DISGUST, FEAR, JOY, SADNESS, SURPRISE. The dataset they use consists of headlines from various news websites. They first show how to construct the dataset and methodology to evaluate the annotation result. Then they propose knowledge-based and corpus-based classification methods to identify the emotions of news headlines. Specifically, in knowledge-based approach, the baseline method is to extract subsets of words from lexical resources for each emotion category and given a document, the system assigns a score for every emotion class by checking the presence of corresponding emotion words in the document, where the score would be metrics of similarity between emotion words from lexicon and to-be-classified document. Inspired by this idea, they implement a better similarity measurement mechanism by using the variation of Latent Semantic Analysis (LSA), the advantage of which is the ability to represent emotion categories, generic terms, and concepts in a homogeneous form. In other words, both emotions and documents can be vectorized in LSA space so that the semantic similarity would be better measured. As for corpus-based approach, they train a Naive Bayes classifier on blog entries from LiveJournal.com [8].

3.1.2 Sentiment Analysis of Movie Review Comments

In Yessenov and Misailovic's work, "Sentiment Analysis of Movie Review Comments", they focus on two properties of review comments: subjectivity and polarity. The data for experiment is from Digg. In order to employ machine learning models, they extract a feature vector from each document. The feature extraction method is based on classic unigram language model: a subset of words in documents are selected as features. For a given document, each feature is assigned by a binary value indicating whether the frequency of occurrence of that certain word is above the threshold. However, they argue that this statistical representation alone does not capture the semantic and syntactic information of text. Therefore, with respect of semantics, they propose to represent semantic-similar words by a single feature with the help of lexical resources and the words are assigned by a numeric value measuring the similarity between the word in feature vector. Hence, the feature value, in this case, would be numeric. As for the syntactic part, due to assumption that the parts of speech such as adjectives and adverbs provide more information, they experiment the classification by only using adjectives and adverbs words as features. Additionally, to deal with the negation in the reviews, they attach a suffix "NOT" to word feature if negation words like "not" appear in the comments. The learning models they use include Maximum Entropy classifiers, Naive Bayes Classifier in supervised learning and K-Means clustering in unsupervised learning [10].

3.1.3 Detecting Sadness in 140 Characters

The last paper, "Detecting Sadness in 140 Characters", describes a specific task in sentiment analysis: detecting sadness from tweets. Authors use as dataset

the tweets on Michael Jackson's death. In the first part, they assign every word in texts a rating ranging from 1-9 along each of three dimensions of emotion affect: *valence*, *arousal*, and *dominance*. By analyzing these tweets, they find the word "sad" so frequently appear. By comparing the tweets on Michael's death with those on "everyday life", it is shown that the rating on word "sad" along the dimension valence from the former kind of tweets is indeed lower than the latter one. Therefore, they come up with the question: how relevant that the usage of "sad" is to the sad emotion. To achieve this, they make human determine whether the tweets containing word "sad" express sad emotion under certain categories, which are "Y"(yes), "N"(no), "M"(mixed), "U"(unclear). It turns out it is rather difficult for people to decide unanimously whether the tweet expresses sad emotion or not [9].

3.2 Comparison

The first two papers share more commonality than they do with the last one. The third paper actually considers a more fundamental problem: whether the text that contains emotions words expresses the sentiment those emotion words indicate. For instance, whether the tweet containing word "sad" indeed expresses sad emotion. The work on first two papers is conducted based on the assumption that expressed emotion is related to word usage.

As for the classification technique, all three papers more or less use a combination of statistical and linguistic feature of text.

4 Data

The dataset I use is the movie review comments from Cornell containing 2000 movie reviews, where the numbers of reviews that express *positive* and *negative* emotions are equal [6].

5 Approach

5.1 Baseline Approach

The following would present multiple baseline approaches and all of them are experimented in WEKA, since it is very convenient to get results in WEKA simply by clicking around in GUI interface [3]. Please refer to this video for instructions on how to format .txt files to .arff file, how to preprocess the text and convert strings to feature vectors on WEKA [4].

The first baseline method is using strings as features and apply Bayes Rule. Specifically, given a document consisting of bag of words, we would like to calculate the conditional probabilities $P(label|doc)$, the classification result we get is $\underset{label \in labels}{\text{argmax}}\ P(label|doc)$. Based on assumption that all features are independent, we can get the $P(label|doc)$ by Eq. 1, among which $P(word|label)$ can be obtained in the distributions of that word in different documents. *NaiveBayesMultiNomialText* can be used to realize this method. (See result in Table

1)

$$P(label|doc) = \frac{P(label)P(doc|label)}{p(doc)} = \frac{\prod_{word \in doc} P(word|label)}{p(doc)} \quad (1)$$

The second baseline method is very similar with the first except that the former extracts a feature vector for every document. Actually, it is the same method used in paper "Sentiment Analysis of Movie Review Comments" in Section 3.1.2. What worth mentioning here is that not all words are considered as features since in that way, it would cause some overfitting issues and increase computational complexity. There are more than 30,000 distinct words, including symbols in the corpus while only 1166 of them are used for features. The number "1166" is actually related to one parameter I set in preprocessing: the largest number of words to keep for every document. I assume how WEKA decides which words should be abandoned depends on the frequency distribution of words since that is what NLTK does.

The reason why I present two baseline approaches here is because *NaiveBayesMultiNomialText* is the only classification method available when the input is raw text. In that way, I cannot compare different learning models based on this input. Furthermore, the results from these two methods give me inspiration about how to improve the system, which will be discussed in Section **??**

After obtaining the features, I conduct experiments on various learning models: *NaiveBayes*, *DecisionTree*, and *SupportVectorMachine*. For convenience of discussion, I simply consider accuracy (percentage of correctly classified instances) as the evaluation metric for model choice. The complete results are presented in Table 1. The rank based on the accuracy is *NaiveBayes*, *SupportVectorMachine*, and *DecisionTree*. The performance of SVM is not bad. However, it takes long time to train the model.

As a result, I would like to focus on Naive Bayes learning model and try to improve the performance by exploring better feature sets for it. Note that this may be not a best choice, since perhaps given a better feature sets, Decision Tree model would perform better than Naive Bayes. As you can see, this is one of limitation of my work: I determine the model I use based on their performance in baseline approach.

5.2 Improved Approach

This section is more a reconstruction of my experimenting process and concrete explanation how I end up using the feature sets with which I get the best performance.

The idea behind this improved approach is actually very simple. Technically, it is still in the context of the unigram model. Despite its simplicity, the performance is improved to a considerable extent. There are two key observations.

There is a function in WEKA interface that you can select a subset of attributes or features that contribute most to the classification task before training the model. However, when I do this, there is little change in the performance.

One plausible explanation for this is that when I extract features from documents, some words with low frequency have been filtered and these words are possibly the "informative" features. Thus, I try to adjust the parameter so that WEKA can keep as many words as possible. Unfortunately, WEKA is very ineffective when dealing such a size of data and the major drawback is that you cannot control the number of words you want to choose based on rank of features' contribution, at least not I am aware of. Due to these factors, I decide to explore the better features in NLTK.

For any trained classifier, there is a classmethod that can return most informative features. However, only the *trained* classifier has such a method, which again causes some complexity problem since to find a informative feature sets for overall corpus, the size of feature vector for each document would be around thirty thousand. If WEKA is capable of ranking the contributions of features, there must be a way to do it in NLTK. When I read the documentation of the classmethod that returns most informative features, I find that *likelihood ratio* can resolve the problem. Actually, similar methods like *chi square*, *mutual information*, etc. are introduced in [7].

By using these carefully-selected features, the model performs a lot better, **??**. Then I wonder if I can abandon the absent words as the first baseline approach does, since the first baseline approach is slightly better than the second baseline approach. Before conducting the experiment, I try to explain the results in baseline approach and if the reasoning stands, I should get the consistent result using more informative features. The explanation is as follows:

Intuitively, how informative of a certain word is determined by measuring the dependency of that word on certain class. Take likelihood ratio as an example, which is easier to understand comparing to other measurements according to Manning [7].

If $P(word|'pos') \gg P(word|'neg')$, then the likelihood ratio of "pos" over "neg" would be very high. However, if we use entire feature vector for classification, the model would take those absent feature into consideration, which would "mitigate" the "informativeness" of informative words. Let's consider a special case here. Assume a review is "good". We simply compare $P('good'|'pos')$ and $P('good'|'neg')$ if we only consider the present word "good". Since "good" is already determined to be informative word, the difference between two probabilities would be significant. However, if we need to consider absent words as well, for example, a "pos" word "excellent". Then we are comparing $P('excellent' = false|'pos')+P('good' = true|'pos')$ and $P('excellent' = false|'neg')+P('good' = true|'neg')$. Because "excellent" is also informative, $P('excellent' = false|'pos')$ is would be small if "excellent" is frequently appear in "pos" documents. Then the difference between to-be-compared probabilities would decrease to a large extent, although it would not be necessarily the case if 'excellent' is a rare word in overall corpus. We can never know. Same reasoning applies to negative words. Thus, why risk adding uncertainty to the classification?

Then, I filter out the features whose value is false during *classification phase*. In other words, the absent words of input document are ignored by *trained classifier*. There is indeed a slight improvement, which verifies my hypothesis. Note that I would like to emphasize that in two methods (whether ignoring absent words or not) the trained model should be the same or the posteriors are the same, even if the absent words are ignored during feature extraction instead of filtering the absent words after feature extraction and the latter way

is undesirable due to the time complexity. Specifically, if words are ignored during feature extraction, the sizes of feature vectors would vary and the values of their features would be always true. The reason why it works can be found in documentation of NLTK: the missing feature value is dealt with an extra function [2].

Additionally, based on this, I also try the feature sets where negation is handled and document is lemmatized.

I would like to denote as nonuniform feature sets those that do not consider absent word. To summarize, the feature sets I have experimented are whether-uniform, whether-handle-negation, and whether-lemmatized. I also try different methods to rank the informative words. These results would be presented in next section.

6 Results

6.1 Comparison of Models in Baseline Approach

I will use standard evaluation metrics for following result analysis. For each class, we have *precision*, *recall*, *f-measure* and average values for them. For the overall classification, we have accuracy.

As Table 1 shows, Bayes model and SVM have some promising performance. However, SVM has higher computational complexity according to the running time. Therefore, as mentioned before, the results obtained in section 6.2 would be in the context of Bayes model.

6.2 Using Informative Features

From the results of Table 2, we can see that nonuniform features work slightly better than the uniform features. However, the difference of time complexity between is significant. For this reason, in the following experiment results, I would **always use nonuniform features**. As is shown in Table 3, if we deal with negation, the result would be slightly better. However, it seems lemmatization does not produce a positive influence on the results. From Table 4, we can find that results from those three feature rank methods are very close, especially for χ^2 and ϕ^2 (Actually the calculation method for these two is almost the same). Combining the results above, I decide to use the following feature sets to explore the effect of feature length on the classification task: nonuniform = True negation handled = True, lemmatized = False, rank method = likelihood ratio.

Since I use the standard evaluation method for both classes, it makes sense that the Figure 1 and 2 have a symmetric pattern. From the pattern, we may conclude that, when there are fewer informative features, the instance is more likely to be classified as positive reviews. As for the overall performance, if the number of features is between certain interval, the model has a better result as the number of features increases. However, when the number is larger than certain threshold, the overall performance decreases, which may due to the overfitting problem. In this case, if we look at Figure 3, when the number of features is around 10000, the accuracy reaches the peak point, approximately 0.95.

6.3 Summary of Projection Questions

Considering that I have contained all the answers among the sections above. I will give a brief summary of project questions we are required to answer.

Q1 : Two baseline approaches, both of which use unigram model.

Q2 : Bayes Model is quite suitable for this task, although Support Vector Machine is also promising according to Table 1

Q3 : combinatoric setting of best feature sets: use nonuniform feature vector; convert all words to lower case, handle negation; do not lemmatize words, use likelihood ratio method to rank feature's informativeness.

Q4 : Refer to Figure 1, 2, 3, 4.

Table 1: Different Models in Baseline Approach

		Precision	Recall	F-Measure	Accuracy
BNMT[1]	pos	0.859	0.759	0.806	0.817
	neg	0.784	0.875	0.827	
	avg	0.821	0.817	0.816	
NB[2]	pos	0.828	0.793	0.810	0.814
	neg	0.801	0.835	0.818	
	avg	0.815	0.814	0.814	
SVM[3]	pos	0.798	0.814	0.806	0.804
	neg	0.810	0.794	0.802	
	avg	0.804	0.804	0.804	
DT[4]	pos	0.601	0.745	0.6656	0.626
	neg	0.665	0.506	0.575	
	avg	0.633	0.626	0.620	

[1] BNMT-NaiveBayesMultinomialText
Features are simply strings
[2] NB-NaiveBayes
Uniform feature vectors (1166 words selectd by WEKA)
[3] SVM-SupportVectorMachine
Uniform feature vectors (1166 words selectd by WEKA)
Linear kernel
[4] DT-DecisionTree
Uniform feature vectors (1166 words selectd by WEKA)
[**] Extra Notes on Experiment Settings
command line to write .arff file from .txt file
import .arff file to WEKA
unsupervised attribute selection: StringToWordVector
supervised attribute selection: evaluator=CfsSubEval, search=BestFirst
choose classification models (cross-validation, 5-fold)

Table 2: Uniform Features versus Nonuniform Features

		Precision	Recall	F-Measure	Accuracy	Running Time
Uniform	pos	0.934	0.898	0.916	0.915	7min 39s
	neg	0.896	0.933	0.914		
	avg	0.915	0.915	0.915		
Nonuniform	pos	0.930	0.793	0.810	0.930	7.19s
	neg	0.941	0.912	0.927		
	avg	0.931	0.929	0.930		

** Exrta Notes on Experiment Settings:
most informative features = True
feature ranking method = likelihood ratio
number of features = 5000
lower case of word = True
negation handled = False
lemmatized = False
cross-validation = False

Table 3: Handle Negation & Lemmatize

		Precision	Recall	F-Measure	Accuracy
(False, False)*	pos	0.924	0.950	0.937	0.937
	neg	0.949	0.922	0.935	
	avg	0.936	0.936	0.936	
(False, True)*	pos	0.959	0.893	0.925	0.927
	neg	0.900,	0.962	0.930	
	avg	0.930	0.930	0.930	
(True, False)*	pos	0.928	0.952	0.940	0.939
	neg	0.951	0.926	0.938	
	avg	0.939	0.939	0.939	
(True, True)*	pos	0.924	0.955	0.939	0.939
	neg	0.953	0.922	0.937	
	avg	0.939	0.939	0.938	

* Binary value for parameter tuple (negation handled, lemmatized)
** Extra Notes on Parameter Setting:
most informative features = True
feature ranking method = likelihood ratio
number of features = 5000
lower case of word = True
cross validation = True (5-fold)

Table 4: Informative Feature Rank Methods

		Precision	Recall	F-Measure	Accuracy
likelihood ratio	pos	0.928	0.952	0.940	0.939
	neg	0.951	0.926	0.938	
	avg	0.939	0.939	0.939	
χ^2	pos	0.899	0.961	0.929	0.926
	neg	0.958	0.892	0.923	
	avg	0.929	0.927	0.926	
ϕ^2	pos	0.899	0.961	0.929	0.926
	neg	0.958	0.892	0.923	
	avg	0.929	0.9265	0.926	

[c] 5-fold cross validation
[*] Binary value for parameter tuple (negation handled, lemmatized)
[**] Extra Notes on Parameter Setting:
most informative features = True
number of features = 5000
lower case of word = True
negation handled = True
lemmatized = False
cross-validation = True (5-fold)

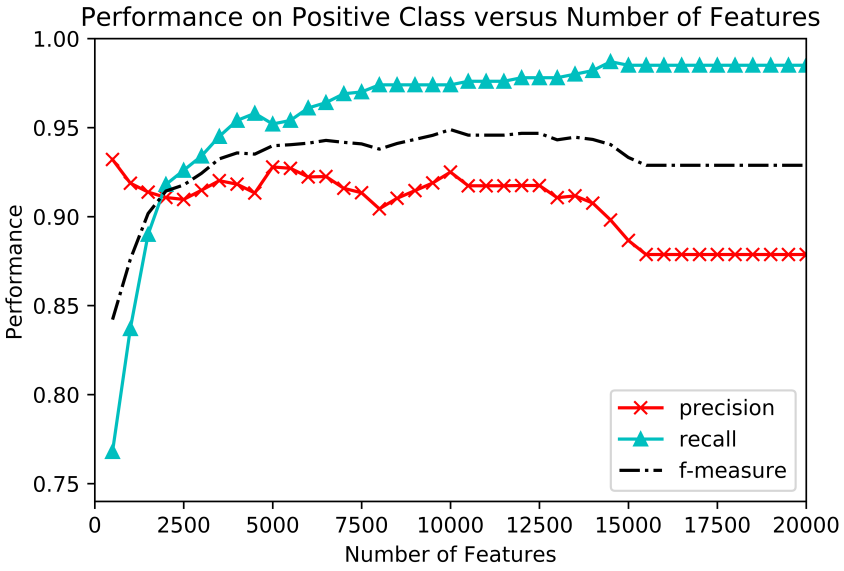

Figure 1: Performance on Positive Class

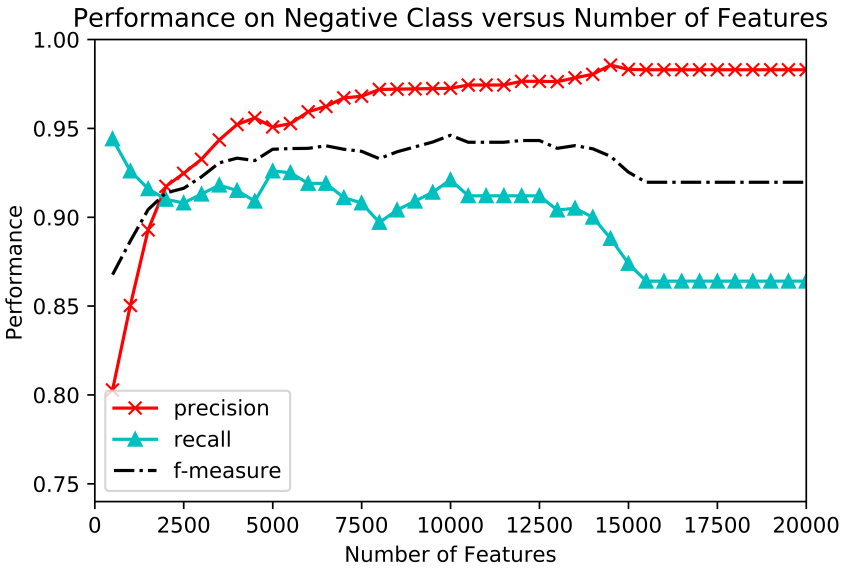

Figure 2: Performance on Negative Class

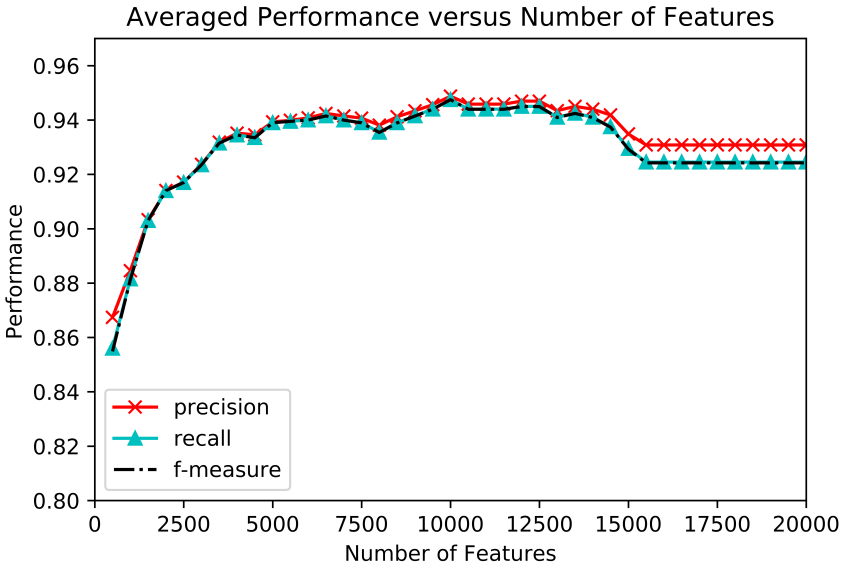

Figure 3: Averaged Performance

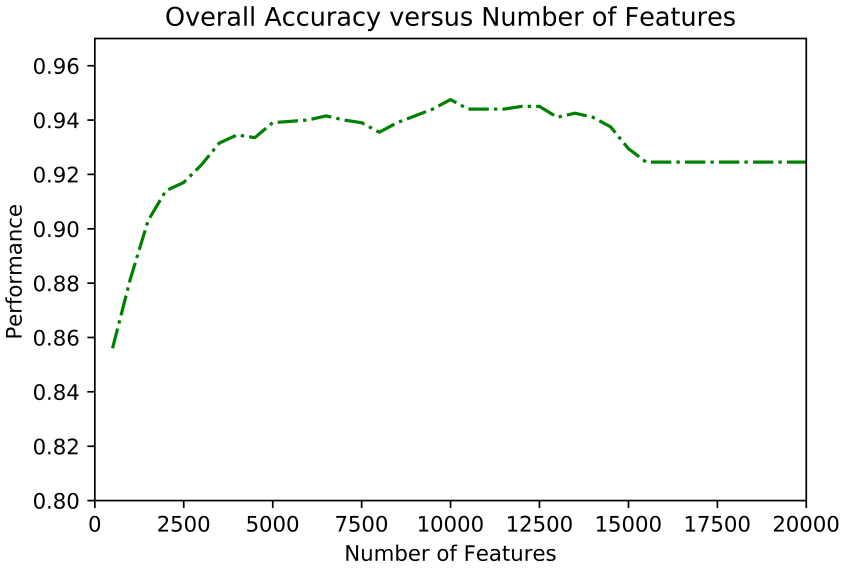

Figure 4: Overall Accuracy

7 Discussion and Conclusion

Overally, the result is satisfying considering that such as simple model is used. Compared with those three papers, I think I have not used texts' syntactic and semantic information that much. That is mainly because I spend a lot of time on understanding the different Bayes Models and how to determine the informative features. If I still have time, I may want to try the some other methods based on the work I have done. On the one hand, NLTK does not have Multinomial Naive Bayes model.

The classifier they implement in python is based on the Binomial distribution assumption, which means for certain word, it may be or may not be in the document. the second paper 3.1.2 mentions that it is reasonable since it is not common to have several same words in a movie review in their dataset. However, it is not necessarily the case in the dataset I am working on. Some of reviews are obviously very long. Thus, I might try Multinomial distribution on other machine learning libraries. On the other hand, I will use more lexical resources like WordNet to contain the semantic information in the feature. As far as I am concerned, one major problem with the statistical approach in this task is some rare words that the model never sees during training phase. With the help of lexical resources, I believe this problem would be more properly resolved.

References

[1] Natural language toolkit. http://www.nltk.org.

[2] Documentation of Naive Bayes Classifier in NLTK. http://www.nltk.org/_modules/nltk/classify/naivebayes.html

[3] Weka. https://weka.wikispaces.com/.

[4] Text Classification in Weka with .txt Files https://www.youtube.com/watch?v=IY29uC4uem8

[5] Definition of Sentiment Analysis:https://en.wikipedia.org/wiki/Sentiment_analysis

[6] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. 2002.

[7] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press. Cambridge, MA: May 1999.

[8] C. Strapparava and R. Mihalcea. Learning to identify emotions in text. *Proceedings of the 2008 ACM symposium on Applied computing*, March 16-20, 2008, Fortaleza, Ceara, Brazil.

[9] E. Kim and S. Gilbert. Detecting Sadness in 140 Characters: Sentiment Analysis and Mourning Michael Jackson on Twitter. *Web Ecology Project*, August 2009.

[10] K. Yessenov and S. Misailovic, Sentiment Analysis of Movie Review Comments. Massachusetts Institute of Technology, Spring 2009.