Multiclass Sentiment Analysis of Movie Reviews

Robert Chan & Michael Wang
CS229, Stanford University

Abstract

We perform sentiment analysis on a new multiclass dataset. We apply various machine learning techniques to this dataset, including Naive Bayes and the Stanford CoreNLP. We then consider various methods for optimizing these algorithms. Finally, we conclude that incorporating syntactical information in our models is vital to the sentiment analysis process.

1 Introduction

Previous studies on sentiment analysis consider a bi-classification problem where only polarized examples are considered (Pang et al, 2002; Maas et al, 2011). While these studies achieve high accuracy, they do not consider neutral examples, or the magnitude of the sentiment (i.e. somewhat positive). However, sentiment is often not unequivocal, and a bipolar view of sentiment has only limited application. In our study, we consider the problem of multiclass classification sentiment analysis. 5 labels are considered: negative, somewhat negative, neutral, somewhat positive, and positive. The Stanford Recursive Neural Tensor Network (Socher et al, 2013) provides a baseline for our study. We consider methods for optimizing basic text classification algorithms, as well as the Stanford CoreNLP package, in order to maximize performance on a new multiclass dataset provided by Kaggle.

2 Data Sources

Data is publicly available to Kaggle users under the competition titled "Sentiment Analysis on Movie Reviews". In the training file, there are 156,060 rows and 4 columns: Phrase Id, Sentence Id, Phrase, and Score (class). A phrase is a concatenation of words separated by a space, and is assigned a score of 0 through 4. In the testing file, there are 66,293 rows. It has the same columns as the training dataset, except for Score. Data pre-processing was not necessary when we evaluate the Stanford CoreNLP library because the model consumes the text corpus as is.

3 Research Question

Formally, the research question we seek to answer is: "given any phrase X, what is the sentiment score Y? ". Namely, we wish to learn some hypothesis *h* that maps from a phrase X to its sentiment score Y.

4 Naive Bayes and Support Vector Machine

For our initial analysis, we analyze the performance of two basic classification learning algorithms : multivariate naive bayes, and support vector machine.

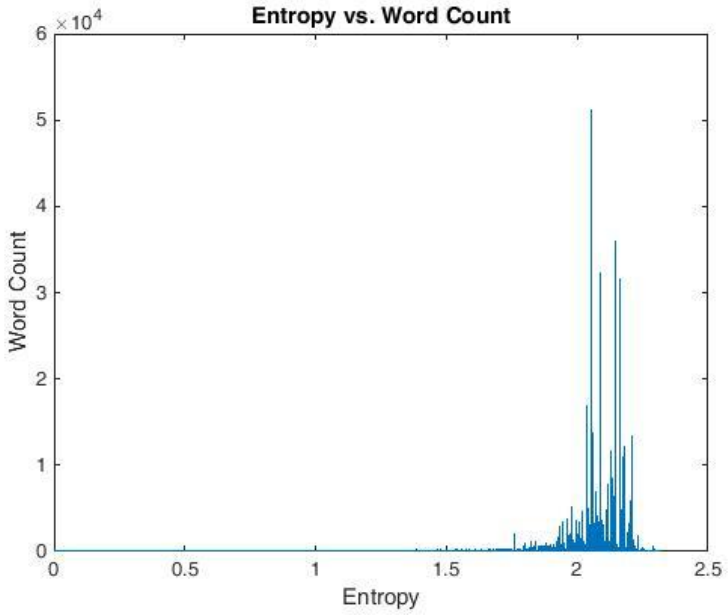
Data Preprocessing

For our initial stage of analysis, we wanted to see if we can determine the sentiment of a phrase based on the occurrence of certain words. Therefore, we decide to use a collection of single words as our feature set, and construct a word-count matrix with words as columns and phrases as rows. This generates 16,510 unique features. We then perform a simple feature selection process based on the

entropy of a word's occurrence and its sentiment. We then select the first 2000 words with lowest entropy as our feature set. Our calculation of entropy is given as follows:

$$H(\mathcal{S}) = -\sum p_i \log_2 p_i$$

where p_i is the probability of i.



SVM

Multiclass vs. Two-class SVM

We first experiment with a multiclass-SVM classification model. There are many ways to handle multiclass classification in SVM, but to keep thing simple, we chose the one-versus-all approach where we decomposed our problem into training five binary-classifiers and had each classifier generating probability estimates for our training set. Then, we assigned a sentiment score to a phrase based on the highest probability estimate. The result was less than promising, with an average accuracy of 14.9%.

Next, we simplified our problem. We observed that about half of our training data are classified as 2 (neutral sentiment), with the other four classes making up the rest of the labels. We wanted to know, given our feature set, if SVM could distinguish phrases with strong sentiment from neutral ones; therefore, we converted the training labels such that all non-2 classes took on the value -1, while class 2 became 1. With this reformulation of our problem, we still achieved a disappointing 49% accuracy.

Out-of-Sample vs. In-Sample Testing

So far, we had been performing 10-fold cross validation to obtain our out-of-sample testing error and accuracy. We wanted to know how well our classifier would do if we ask it to classify data that it has already seen. The in-sample testing accuracy is only 50.99%.

Naive Bayes

We performed multivariate naive bayes with LaPlace Smoothing. This algorithm achieved 47% accuracy on the test set.

The multinomial event model assumes that any example picked from the distribution is first randomly determined to be of a certain sentiment, and then is composed by running through a dictionary and deciding whether to include a word independently and according to some conditional probability. However, we believe this view of how phrases are generated is not a reasonable approximation. Namely, we believe that the naive bayes assumption does not hold for sentiment analysis. For example, given a phrase of positive sentiment of a certain length, the presence of the word "good" is conditional on the previous words chosen, as well as the words that come afterwards; for a phrase of length 2 and a sentiment score of 5, the probability of the word "good" as the second word will be very low in the case that the first word is "not", and high in the case the first word is "very". Therefore, if we wish to use multivariate naive bayes, we need to consider another feature set where the naive bayes assumption holds.

Improved Naive Bayes

We believe that the feature vectors we are using to train our learning algorithms have little to no correlation with the labels. While the presence of certain words is probably indicative of sentiment (i.e. "good", "bad", etc), their effect on the sentiment is likely conditional on the presence of other "inversion" words before or after these indicative words ("not", "although", "but", etc.). We perform pre-processing on the dataset in order to create a more relevant feature vector. We start by trying to identify "inversion" words (i.e. words that indicate a polar switch in sentiment). We select a set of 50 words in the dictionary that we believe have strong sentimental value (for example, "good", "bad", "well") and assign to each a polar sentiment score (i.e. "good" or "bad"). For each of these sentimental words, we calculate the conditional probability of the other words in the dictionary given the presence of a sentimental word and the opposite strong sentiment score (i.e. for a word that we assign to be "good", we consider sentiment score of 0). We are then able to create a list of 10 words from the dictionary that have highest probability of inverting sentiment. We then process the dataset such that we create feature vectors that contain the number of inversion words present, as well as the word frequency of the sentence snippet that is after the last inversion word. For example, the sentence "This movie could have been good, but did not achieve its potential" becomes a feature vector containing the number 1 (since there is one inversion word "not"), and the word frequency vector of "achieve its potential" (the sentence snippet that is after the last inversion word not). Logically, such a feature vector considers that the sentiment of a sentence is dependent only on the number of inversion words, and the words that appear after the last inversion word. This logic can be verified by consider the simple sentence "It's not good". Such a sentence probably has a negative sentiment, and it's negative sentiment is a function of the presence of an inversion word "not" and the word "good" that appears afterwards. We then performed naive bayes with LaPlace Smoothing using these new feature vectors, and achieved 54% accuracy on the test set, an increase of 7% from naive bayes performed on the word frequency vectors alone.

5 Stanford CoreNLP

Another approach to sentiment analysis that is different from SVM and Naïve Bayes is the use of natural language processing. This method incorporates linguistic information such as word order, part of speech, and named entity to understand a corpus, thus allowing it to better infer sentiment of a given text. For this paper, we use the Stanford CoreNLP library as our tool, which comes bundled with state-of-the-art models for the natural language processing pipeline. More importantly, the sentiment analysis portion of this tool is the direct result of Socher's work on the Recursive Neural Tensor Network, which is of special interest to us because we want to explore the concept of inversion words and Socher claimed in his work that "it is the only model that can accurately capture the effects of negation and its

scope at various tree levels for both positive and negative phrases" (Socher et al, 2013).

To use Stanford CoreNLP for sentiment analysis requires the propagation of text through a chain of processes that form a pipeline. Options are available at each stage of the pipeline that can be configured to suit the body of text being processed. Without a deep understanding of each stage of the pipeline and all the options available to us, we selected a few parameters that we believe are applicable to our dataset, or that the literature suggest may improve performance. Table 1 summarizes the options we used, the reason we chose them, and their corresponding training and testing error.

Options	Reason	Training Error	Testing Error
Baseline	Default setting of CoreNLP are: tokenize.whitespace = false; ssplit.eolonly = false; pos.model = left3words; ner.model = 3class, 7class, MISCclass; parse.model = PCFG	27.810%	35.357%
tokenize. whitespace	We set this option to true because the dataset provided by Kaggle is already pre-processed such that each phrase contains tokenized words separated by spaces; therefore, we don't want CoreNLP tokenizer to introduce new tokens.	27.802%	35.344%
ssplit. eolonly	Similar to tokens, we set this option to true because our dataset is already pre-processed such that each line is a phrase; we don't want CoreNLP to further split our phrases.	27.804%	35.342%
pos. model	We experimented with bidirectional dependency network tagger because it is said to outperform "previous single automatically learned tagging result" (Toutanova et al, 2003)	27.815%	35.375%
ner. model	We experimented with conll.distsim.iob2.crf.ser.gz because it is said to perform better than the models being distributed with Stanford CoreNLP. (http://nlp.stanford.edu/software/crf-faq.shtml#k, Accessed 12 Dec, 2014)	27.804%	35.342%
regexner. mapping	CoreNLP allows us to define our own named entities, so we created our own list of movie titles , thinking that it'll help the sentiment analyzer to not be influenced by words appearing in titles.	27.804%	35.342%
parse. model	We experimented with Neural-Network Dependency Parser (RNN) because it has the highest F1 score compared to other available parsers (http://nlp.stanford.edu/software/srparser.shtml, Accessed 12 Dec, 2014)	27.838%	35.402%

Table 1. Various options we experimented with in Stanford CoreNLP and their results.

It is worth noting that since all the processes form a pipeline, the effect of each option being tested is cumulative, unless they don't improve the overall performance (highlighted in yellow), or they increase the testing error (highlighted in red). Therefore, the final model that we use to analyze our dataset has the options *tokenize.whitespace* and *ssplit.eolonly* set to true (highlighted in green).

5 Discussion

We show that we can achieve higher accuracy achieved by simple text classification algorithms by optimizing the features that are considered. Specifically, by using basic sentence structure logic, we can identify the words in the dictionary that are most likely to invert sentiment, then by considering only the words that appear after all inversion words, we were able to create a set of features that improved the performance of Naive Bayes. We believe that by doing more sophisticated processing using sentence structure, we can identify other categories of words that have high correlation to the sentiment. For example, we can divide each sentence segment into noun, verb, and object segments, then considering how each segment relates to the overall sentiment score (i.e. the presence of "smart" and "provocative" in the noun segment have high correlation to a positive sentiment, but may not have high correlation when present in the object segment).

Although our preliminary result from using Stanford CoreNLP for sentiment analysis already

outperformed SVM and Naive Bayes, the task is far from complete and the result is far from its potential. For one, we never used the training data provided by Kaggle to train our own model. An attempt was made to process the data into the Penn Treebank (PTB) format, which is required by the Stanford CoreNLP library. However, the way in which the sentences were broken down into phrases in the training dataset was not consistent, making the task of converting it to PTB format error prone and time consuming. Another task that would substantially improve the performance of this method is to get a better understanding of each stage of the natural language processing pipeline and to utilize all the options available to fine tune the model. As the saying goes, a tool is only as good as the person who is using it.

6 Acknowledgement

We want to thank the CS229 Course Staff at Stanford for suggesting that we incorporate some syntactical information in our models, as well as to motivate us to explore Stanford CoreNLP.

7 Sources

Accurate Unlexicalized Parsing, Dan Klein and Christopher D. Manning, Proceedings of the 41st Meeting of the Association for Computational Linguistics 2003, pp. 423-430

A Fast and Accurate Dependency Parser using Neural Networks, Danqi Chen and Christopher D Manning, Proceedings of EMNLP 2014

Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network, Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer, Proceedings of HLT-NAACL 2003, pp. 252-259.

Incorporating Non-local Information into Information Extraction Systems, Gibbs Sampling. Jenny Rose Finkel, Trond Grenager, and Christopher Manning, Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.

Learning Word Vectors for Sentiment Analysis, Andrew L. Mass, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts, Proceedings of the 49th Annual Meeting of the Association for Computation Linguistics 2011

LIBSVM : a library for support vector machines, Chih-Chung Chang and Chih-Jen Lin, ACM Transactions on Intelligent Systems and Technology 2011, 2:27:1--27:27

Nlp.stanford.edu, (2014). The Stanford NLP (Natural Language Processing) Group. [online] Available at: http://nlp.stanford.edu/software/crf-faq.shtml#k [Accessed 12 Dec. 2014].

Nlp.stanford.edu, (2014). The Stanford NLP (Natural Language Processing) Group. [online] Available at: http://nlp.stanford.edu/software/srparser.shtml [Accessed 12 Dec. 2014].

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng and Chris Potts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).

Thumbs up? Sentiment Classification using Machine Learning Techniques, Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, Conference on Empirical Methods in Natural Language Processing (EMNLP 2002).