Sentiment Analysis Using Semi-Supervised Recursive Autoencoders and Support Vector Machines

BAHAREH GHIYASIAN and YUN FEI GUO

December 12, 2014

1 Introduction

Sentiment analysis is an important field that aims to extract opinion or other subjective information from sources such as text. With the advent of social networks, blogs, reviews, and online shopping, sentiment analysis has garnered significant attention from both the industry and the research community[3]. Being able to accurately identify sentiment from online text allows businesses to better understand their customers. It gives businesses invaluable insight about what were the pain points in their products and services, what potential improvements can they make on their products and services, and etc. Because of its potential to obtain high-level insight from large amounts of data, sentiment analysis has applications in various domains [3]. Sentiment analysis can be used to summarize user reviews on review-related sites [3]. It can be used as sub-component of software systems such as question answering systems. In organizations such as business and governments, it can also be used for reputation management and public relations [3].

Past work on sentiment analysis has often focused on classifying text into two classes, positive and negative [3]. Some recent work [6] has tried to classify text into more than two classes. In our experiments, we will classify data into five classes: very positive, positive, neutral, negative and very negative. We will make use of the sentiment analysis dataset on Kaggle [5], which contains phrases and sentences from Rotten Tomatoes movie reviews.

2 Dataset

Our dataset [5] consists of 8544 sentences which is converted to 156060 English phrases from movie reviews. We split our dataset randomly into a training set and a testing set, so that 70% of the phrases are in the training set and 30% are in the testing set. Our goal is to classify the phrases in the testing set into five categories (very negative, negative, neutral, positive, and very positive). All phrases from the dataset (both training and testing) are labelled with integers from 0 to 4, where 0 indicates very negative and 4 indicates very positive.

3 Features and Pre-processing

For the SVM the features are the count of each word in each phrase. For the neural network, the features for each phrase are a vector that is generated by the recursive autoencoder and encodes the semantic and syntactic information of the phrase.

4 Models

4.1 Support Vector Machines

One algorithm we used on the dataset is support vector machines with linear kernel. We used the package LIBLINEAR to train and test our dataset. To prepare our data, we found a list of all unique words in our dataset. We then created a matrix A, where each row is a training sample (a phrase) and each column is a unique word. For each row in

the matrix, we have the count of each unique word in that particular phrase. Initially we made our features case-insensitive so that both upper- and lower-case forms of a word would map to the same feature. But we found that this didn't give us good results. Also, we tried using the unique words from [6] as our features but we found that this didn't give us good results. So instead we used the original words with their case information as features. The number of unique words or features is 18227.

Because we have five label classes, we cannot run a simple SVM. Instead, we ran one-vs-all SVM. So for each label class, we let its training and testing samples be the positive class and all other training and testing samples be the negative class. We then run linear svm on the dataset and the modified labels.

4.2 Semi-Supervised Recursive Autoencoders

Another algorithm that we used was semi-supervised recursive autoencoders [6]. The algorithm consists of an unsupervised part and a supervised part. The unsupervised part is essentially a recursive autoencoder that creates a "code" or an N-dimensional vector that represents the phrase. To do this, each phrase in the training set is converted to a list of N-dimensional vectors where each vector represents a word. In our experiments, we build each vector by drawing each element in the vector randomly from a uniform distribution in the range of [-0.05, 0.05]. We also choose N to be 50, although N can be any positive integer. For each pair of adjacent words in the phrase, we calculate the following value:

$$p = f(W^{(1)}[c_1; c_2] + b^{(1)}) \tag{1}$$

where $W^{(1)}$ is an N-by-2N matrix, c_1 and c_2 are vectors corresponding to each word in the pair, and $b^{(1)}$ is an N-by-1 vector. The function f is an activation function such as the sigmoid function or tanh. In our case, we used tanh.

After computing p, we want to obtain the original word vectors from p by performing the following evaluation:

$$[c_1^{'}, c_2^{'}] = W^{(2)}p + b^{(2)} \tag{2}$$

where $W^{(2)}$ is a 2N-by-N matrix and $b^{(2)}$ is a 2N-by-1 vector. The goal of the autoencoder is the minimize the reconstruction error:

$$E_{rec}([c_1; c_2]) = \frac{1}{2}\left\|[c_1; c_2] - [c_1^{'}; c_2^{'}]\right\|^2 \tag{3}$$

For each pair of adjacent words in the phrase (e.g. x_1 and x_2 followed by x_2 and x_3), we calculate the reconstruction error. We pick the pair with the lowest reconstruction error and replace both word vectors with their code p. We then perform the same reconstruction error calculation on the new list of word vectors again and find the pair with the lowest error. We continue in this greedy fashion until there is only one vector left in the list. This vector is the "code" that represents the phrase.

After obtaining an N-dimensional code for each phrase from the unsupervised recursive autoencoder, we wish to use supervised learning to classify this vector into a particular class. Many standard supervised learning algorithms such as naive bayes and support vector machines can be used here. In the case of multi-class classification, one can choose to use softmax regression. In our case, we chose to run one-vs-all logistic regression. For each of the five label classes, we label its training and testing examples as being the positive class and all other training and testing examples as being the negative class. We then ran logistic regression on this data to determine what percentage of the testing examples were correctly classified.

5 Results

5.1 Support Vector Machines

For the linear SVM we have the following results:

Positive Class	Accuracy	Number of Correct Labels
Very Negative	95.5%	44734
Negative	83.3%	38999
Neutral	74.1%	34688
Positive	80.3%	37605
Very Positive	95.0%	44367

The first column indicates the class that was used as the positive class in the one-vs-all SVM. So the data for the first row results from treating training and testing examples with label very negative as 1 and all other training examples as 0. The second column in the table is simply the percentage of testing examples that were correctly labelled. The third column is the actual number of testing examples that were correctly classified. The total number of testing examples is 46818. The average accuracy of all five classes is 85.6%. It took about 600 iterations for the SVM to converge.

5.2 Semi-Supervised Recursive Autoencoders

For the Semi-Supervised Recursive Autoencoder we have the following results:

Positive Class	Accuracy	Number of Correct Labels
Very Negative	95.6%	44758
Negative	82.9%	38812
Neutral	72.9%	34130
Positive	80.4%	37642
Very Positive	94.6%	44290

The table has the same format as SVM table. The average accuracy of five classes is 85.3%. The maximum iteration was 70, and 4 sub-models reached the maximum, "very positive" model stoped at 34.

6 Discussion

6.1 Impact of Data Format

In our experiments, the neural network has a slightly lower performance than the SVM. We hypothesize that this may not be the result of the neural network being less effective in sentiment analyses. When we manually scanned the training and testing examples, we found that many of the labels didn't make sense. For instance, the phrase "introspective and" has the label "positive", while the phrase "introspective" has the label "neutral". Also, the example "is worth seeking ." has the label "positive" while the example "is worth seeking" has the label "very positive". Our theory is that the SVM works better for examples similar to the latter because it can easily recognize that a dot contributes to increasing the positivity scale. The neural network is more proficient in recognizing the semantic and syntactic relationship within the phrase, and therefore would be weaker in recognizing one-off cases like these. Given that the dataset labels are hard to understand even by human standards, it is fair to say that the neural network's worse performance on this dataset is actually an indication that it is a more effective algorithm overall.

Another reason why the neural network performed worse might be because of the size of each phrase. The dataset used in the paper [6] contains training and testing examples with large amounts of text. Our dataset consists of phrases that were in comparison much shorter in length. We hypothesize that the neural network works better when the size of the phrases/sentences are large. The figure 1 shows the distribution of the size of each phrase in data set for different labels.

6.2 Improvements on Support Vector Machines

The SVM can be improved in several ways. In our experiments we used a linear kernel.We can try using another kernel such as the Gaussian kernel in future experiments. (However, since the size of the

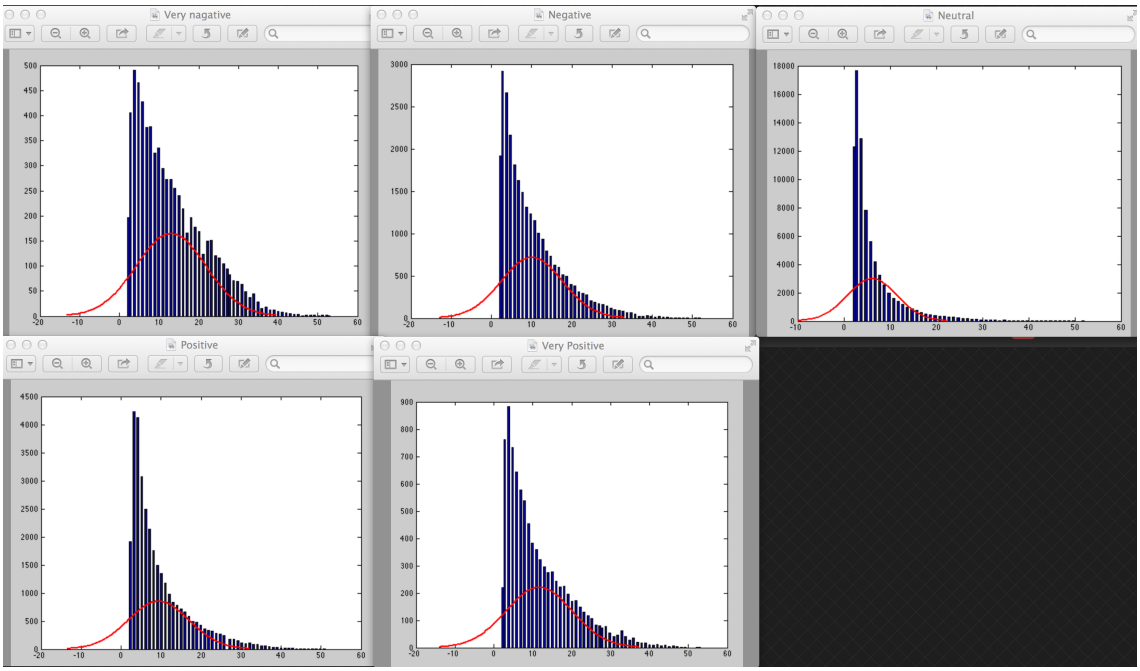

Figure 1: Distribution of the size of each phrase in words in training and testing data sliced by the label.

feature set (18227 words) is large, we doupt that kernel could improve the performance). We can also try to adjust the regularization parameters. We used a large number of features. It is possible that many of these features were not effective in predicting sentiment. Therefore one can perform forward search or other algorithm to select only the features that are useful. We can also spend some time trying to develop meaningful features that correlate strongly with certain sentiment labels.

6.3 Improvements on Semi-supervised Recursive Autoencoders

In our method, we used randomized numeric vectors of dimension 50. While they are sufficient for giving us decent results, they don't capture the fact that some words are more similar to each other in terms of meaning or the concepts they represent. For instance, the pair of words apple and orange is more similar than the pair of words apple and car. To obtain this information, one can use pre-trained word vectors such as [4] that are trained over a large corpus of text such as Google News articles or Freebase. Words that tend to co-occur in the corpus would have vectors that are very close to each other in terms of distance.

One thing we noticed while running the neural network is that for several of the label classes the program did not terminate until the max number of iterations (70) was reached. Therefore it is possible that if we set the max iterations variable to be a higher number, the neural network would perform better.

We believe that if we had used softmax regression instead of one-vs-all logistic regression, the neural network performance would have been better because the labels are mutually exclusive. This is because softmax regression gives a probability distribution over the set of label classes (the sum of probabilities will be 1), which means that there's a

definitive answer as to whether a particular example is from a particular class (class with the largest probability). However, in one-vs-all logistic regression it is possible that more than 1 sub-model return true for the same sentence: like both poitive and very positive would return true for the same sentence; in this case, there is at least one error. So one-vs-all logistic regression introduces unnecessary errors.

Another thing that one can try is to alter the embedding size, or the size of the word vectors. In our experiments we used 50, but there is no reason why this should be the number. There might be other embedding sizes that yield better results. Similarly, one can try to tweak various parameters in the recursive autoencoder to see if one can yield better results. For instance, one can try to alter α, the parameter that controls the relative weighting between the reconstruction error, error of the unsupervised part, and the cross-entropy error, error of the supervised learning part.

We could try increasing the training dataset size. Because we believe recursive neural network is a more complex algorithm (non-linear), it needs more training data to reach a certain upper bound on the error with high probability. Therefore, we hypothesize that if we increase the size of the training data, the marginal improvement of neural network would be more than SVM's.

7 Conclusion and Future Work

On our dataset, the neural network performs slightly worse than support vector machines. Based on our discussion, we believe that this may not be an indication that the neural network is a less effective algorithm, but that its lower performance might be due to the nature of the dataset. More rigorous work in the future will be needed to determine under what conditions are neural networks more effective than conventional algorithms and by how much.

Lastly, much of the work done in sentiment analysis has been to categorize text into groups that range in posivity [2]. Few works have been done to classify text in terms of subjective categories that are not positive or negative. It would be interesting for future study to try to classify text into categories that have varied meanings. For instance, for online forums that address customer concerns, one may try to classify the posts as feature request, software bug, account issue, payment issue, and etc.

References

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. 2011. Learning Word Vectors For Sentiment Analysis. In *ACL*.

[2] B. Pang and L. Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *ACL*, pages 115124.

[3] B. Pang and L. Lee. 2008. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2(1-2):1135.

[4] Google. 2014. word2vec. https://code.google.com/p/word2vec/.

[5] Kaggle. 2014. Sentiment Analysis on Movie Reviews. https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews.

[6] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. 2011b. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *EMNLP*.

[7] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng and C. Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *EMLP*.

[8] Stanford. 2014. UFLDL Tutorial. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial.