Movie Review Sentiment Analysis Using A Neural Network

Justin Barry **Matthew Keeran**

Abstract

Neural networks (NN) are an important tool for tasks involving classification. In this paper we train a neural network to classify the sentiment of movie reviews as being either positive or negative. We use Stanford's IMDB review dataset [1] of 50,000 highly polar movie reviews. 25,000 are used for training the neural network while the remaining 25,000 are used to test the accuracy of the trained classifier. We then implement a simple random forest (RF) classifier with which to compare the efficiency and accuracy of the two models. Our experiment resulted in the RF producing slightly better results than the NN.

1 Introduction

Inferring sentiment from text is a common problem in everyday life and there exist many methods for attempting to quantify and identify the underlying sentiment of utterings. As sentiment analysis tasks are, at their heart, a classification problem, all traditional techniques that apply to classification in general, can be applied to sentiment analysis. Although in many situations sentiments can have a wide range from positivity to indifference to negativity and beyond, some contexts can be sufficiently modeled by restricting the classification of sentiment to the extremes, being either solely positive or negative. Such is the case in this paper where we are attempting to determine whether a person liked a movie or not by analyzing the words they used in their review of it.

Two methods used in classification tasks are NNs and RFs. Although there are many types of NNs and ways in which to implement them the most basic type, and the one we use in this paper, is a feed forward neural network with back propagation. A feed forward neural net sends information through the network in one direction, that is, there are no cycles in the connections between nodes at different layers as there are in recurrent neural networks. A RF is an ensemble of decision trees as first mentioned in [8]. For an input, each decision tree in a forest will classify the input and the result of the RF classifier will be the majority result of all the decision trees. RF classifiers are known to be fast and efficient. They don't overfit and they have few hyper parameters.

2 Background

2.1 Bag of Words

Bag-of-words (BOW) is a simplistic yet commonly used approach to word embeddings. A BOW representation of a sentence is simply a frequency count of all the words in that sentence. The model is simple, but it is not very sophisticated and doesn't capture much useful information. Furthermore, the relationships between words are lost because there is no sequencing captured in the model. It is simply a bag of words. We use this approach to vectorize the movie reviews and pass them to a NN. We will compare results to the other methods.

2.2 Word2Vec

Word2Vec is a language model that represents words as vectors, or embeddings, and was originally proposed by Mikolov et al [7, 8]. The model is created by a NN and provides two architectures for embeddings, continuous bag-of-words (CBOW) and skip-gram (SG). CBOW predicts the target word based on its surrounding words, and

SG predicts the surrounding words based on the target word. CBOW is faster to train while SG is more accurate and works better than CBOW on a smaller corpus. For this project, we only use CBOW with W2V. The CBOW vectors produced from Word2Vec can be used as inputs to binary operators to produce some interesting results. For instance, we can have relationships such as vec("king") - vec("man) = vec("queen") - vec("woman").

2.3 Previous Work

Our project was inspired by Stanford students using neural networks to classify movie reviews [5,6]. In paper [5], the authors use a variety of methods to classify the data. Among them is an RNN trained on parse tree data from the Stanford Sentiment Treebank. It was found that the Support Vector Machine (SVM) classifier had the highest accuracy. In paper [6], the authors again use a variety of methods including training a RNN on Parse Trees and training an SVM on W2V vector averages. While we intended on implementing an RNN of our own, as neither of us had ever written a neural network or used tensorflow before, this proved too difficult in the time we had, so we created a simple feedforward neural network to train on the same dataset instead.

3 Dataset and Preparation

IMDB Movie reviews, as mentioned in [1], are available for the specific purpose of sentiment analysis. The set is split into training set and a test set, both consisting of 25K labeled reviews. The training set and the test set do not contain similar reviews. The reviews have been curated to include polar reviews so as to simplify the task of classifying sentiment.

3.1 Equipment Used

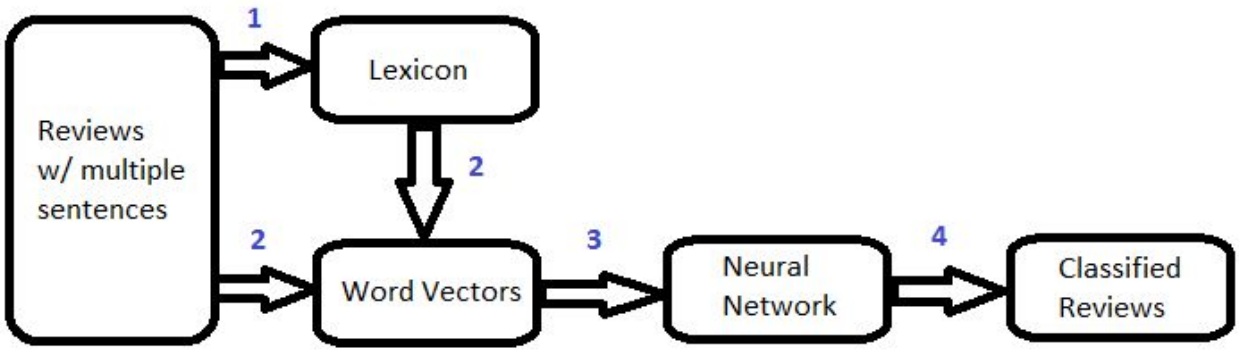
We used 2 computers to run experiments on, the first a laptop using a CPU and the second a desktop with a GPU. The laptop has an Intel i5 with 2.7 GHz CPU and 16 GB of memory, and was used to tune parameters with a subset of the data. The desktop has an Intel i7 with a NVidia Geforce GTX 1080 GPU and was used to compare the runtime efficiency between models.

4 External Software

All software was written in Python 3. Python libraries used: numpy, tensorflow, nltk, word2vec, gensim. For fast and efficient matrix operations, we used Numpy. We used NLTK to tokenize and lemmatize reviews, BeautifulSoup to remove html from the reviews, and Python's regex library for purging emoticons. We used tensorflow to train and optimize our NN, and Gensim for training the word2vec model and fitting a random forest classifier.

5 Implementation

Each review contains more than one sentence, so in the first NN model we treat the entire review (all sentences) as a bag of words and create a single vector containing the frequencies of all words in the review. We then feed these vectors into the NN and backpropagate, resulting in the trained model.



Using the Word2Vec approach, we use Gensim to train a CBOW model on the movie review dataset. After training a W2V model, for each word in our lexicon, we query the model to return it's vector representation. For each movie review we average the vectors of each word in the review to obtain our feature vectors. We then feed the feature vectors to a RF classifier and a NN and record the results.

6 Experiments and Results

6.1 Experiments

We test the accuracy and run time efficiency of our NN initially with a subset of all the data, 5,000 negative reviews and 5,000 positive reviews are used for training while 1,500 negative and positive reviews are used for testing (13,000 total). We then tune various parameters of the NN, using different numbers of hidden layers, nodes, and sizes of lexicon. We started our tests using 3 hidden layers with 500 nodes and trimmed the lexicon according to word frequency with the bounds $500 > x > 1$. We then run the model 5 times and record the range of accuracy, average accuracy and average run time. We then repeat this process using 2 and 4 layers, 250, 750 and 1000 nodes, and using word frequency bounds of 750 and 1000 for the upper bound and 5, 50, 75 and 100 for the lower bound. We then combine the best parameters and perform 5 runs on the full dataset.

The NN was also trained using word embedding averages as features vectors. We trained the Word2Vec model on the same corpus that we ran the our feedforward NN against. Before training the model, we removed emoticons, and HTML. Stop words were not removed before training the model as they were when creating a lexicon for BOW. After training, the model can be queried to return vector representations of a word. For each review in the training set, we created a feature vector for a review by taking the average of the word embeddings in the review. We then fed the feature vectors of each review to the NN in with a batch size of 5000 vectors per batch.

For the RF classifier, we used a W2V model to create averages of review vectors in the same way we did for the NN. After creating feature vectors, the RF classifier is then fit to the training data.

6.2 BOW NN

Combining Best	All data
training reviews	25,000
testing reviews	25,000
lexicon freq range	1000 > x > 75
lexicon size	4,369
nodes	1,000
hidden layers	3
accuracy range	69.4% - 71.5%
average accuracy	70%
average run time	1279 seconds
hidden layers	2
accuracy range	72.1% - 74.5%
average accuracy	73%
average run time	1125 seconds
hidden layers	1
accuracy range	74.1% - 75.1%
average accuracy	75%
average run time	1076 seconds

First we discuss the effects of the number of hidden layers, number of nodes, and size of the lexicon on accuracy and runtime efficiency. While increasing the number of hidden layers from 3 to 4 appears to decrease the run to run variability in accuracy, it does not change the average accuracy of all 5 runs and doing so increases the overall runtime of the algorithm. Additionally, when using all of the data, increasing the number of layers actually results in a decrease in accuracy. Conversely, reducing the number of layers from 3 to 2 increases the accuracy of the NN by 3%. Although increasing the number of nodes improves accuracy it requires more computation to do so and exhibits diminishing returns in the added accuracy above 1000 nodes. Reducing the number of nodes, while improving run time, reduces the accuracy. While the number of nodes in this case appears to improve the accuracy by 4% from the original parameters, reducing the number of hidden layers to 1 improves accuracy by 7% from the original parameters. However, by far the greatest gain in accuracy and decrease in runtime of the model comes from choosing the best bounds for the frequency of words used to trim the lexicon, this improved the accuracy of the model by 9% from the original parameters. Choosing the lower bound is by far the most important as most words are of low frequency and increasing the lower bound not only improves accuracy to a point, but as it reduces the size of the lexicon, this shortens the vectors for all reviews which in turn dramatically decreases the overall runtime of the NN. All other parameters held constant starting with bounds of $500 > x > 1$ yields an average accuracy of 59% whereas using bounds of $1000 > x > 75$ results in an average accuracy of 68%, the runtime also is reduced from 850 seconds to 337 seconds as a result of the size of the lexicon being reduced from 31,500 to just 2,242 words (using the subset of data to tune parameters on). Collectively using the best parameters combined and all the data with a 50/50 split between training and testing, the best accuracy achieved was 75%, a 16% gain from the original parameters.

6.3 Word2Vec NN

In general, the results of the NN using word-embedding averages performed slightly worse than the BOW implementation. The NN had 3 layers for each trial run. Each trial run uses 95% of the 50K reviews as training data and 5% as test data. To train the model, the resulting feature vector was limited to size 1000 and the window size was set to 10. For training the feedforward NN, the average node size for each run was 1000 per layer. Layers with nodes less than 1000 decreased the accuracy noticeably. Increasing the number of nodes past 1000 slightly lowered the accuracy. It seems 15-20 epochs is the range that gave the highest accuracy. Ultimately, the highest accuracy was 77.64 and came from a trial run that had 1000 nodes for each of the three layers, a feature vector of size 2000, and 25 epochs.

input layer size	750	1000	1000	1000	1000	500	1000
hidden layer size	750	1000	1000	1000	1500	500	1000
output layer size	750	1000	1000	1000	1000	500	1000
feature vector size	2000	3000	1000	2000	2000	2000	2000
num epochs	20	20	15	15	15	15	25
accuracy (%)	74	73	76	77.44	76	74	77.64

6.4 Random Forest

Interestingly, the RF gave the best results overall and it was by far the fastest. The highest accuracy was 83% and came from a feature vector size of 1000. A comparable run to the NN trial run with the highest accuracy, which is 77.64%, would be the run where the feature vector was size 2000. This completed in just over 2 minutes and had an accuracy of 80%. Results can be seen in the table below.

	1	**2**	**3**	**4**
feat size	300	1000	2000	3000
accuracy (%)	60	83	82	80
time (s)	20	67	87	150

7 Conclusions

Across all three approaches, the resounding theme seems to be that more is not necessarily better. There seems to be an optimal range for each parameter. Interestingly, a feature vector around size 1000 seems to yield the highest accuracies for our data and models. Perhaps this is because feature vectors of this size contain the most relevant information, whether for BOW model or CBOW model.

Our results demonstrate the difficulty in not only choosing an appropriate neural network and topology to use to perform sentiment analysis but additionally in choosing the optimal parameters for the network such as the size of the lexicon. In the future there are a number of ways in which the efficiency of our NN could be improved, including storing the reviews in RAM as we create the lexicon instead of rereading them reading them once to create the lexicon, thereby reducing the number of memory accesses. Additionally more structured approaches to analyzing the sentiment sentence by sentence, etc. may help to improve accuracy.

References

[1] Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. (2011, June). Learning word vectors for sentiment analysis. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1 (pp. 142-150).

[2] Arora, Liang, and Tengyu Ma. "A Simple But Tough-to-Beat Baseline For Sentence Embeddings." In Proceedings of ICLR 2017. https://openreview.net/pdf?id=SyK00v5xx

[3] Socher, Richard, et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank." Proceedings of the conference on empirical methods in natural language processing (EMNLP). Vol. 1631. 2013.

[4] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their composition-ality." Advances in Neural Information Processing Systems. 2013.

[5] Timmaraj and Khanna. "Sentiment Analysis on Movie Reviews using Recursive and Recurrent Neural Network Architectures." https://cs224d.stanford.edu/reports/TimmarajuAditya.pdf

[6] Pouransari and Ghili. "Deep learning for sentiment analysis of movie reviews." https://cs224d.stanford.edu/reports/PouransariHadi.pdf

[7] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv, 2013.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, Distributed representations of words and phrases and their compositionality, In Proceedings of International Conference on Neural Information Processing Systems, pp. 3111-3119, 2013.

[9] L. Breiman, Random forests, Machine Learning, vol. 45. Issue 1, pp. 5-32, 2001.

Appendix

Combining Best	
training reviews	10,000
testing reviews	3,000
hidden layers	3
lexicon freq range	1000 > x > 75
lexicon size	2,242
nodes	1,000
accuracy range	70.3% - 73%
average accuracy	72%
average run time	423 seconds
nodes	1,250
accuracy range	72% - 73.9%
average accuracy	73%
average run time	396.6 seconds

Number of Nodes	
training reviews	10,000
testing reviews	3,000
hidden layers	3
lexicon freq range	500 > x > 1
lexicon size	31,500
nodes	500
accuracy range	58% - 65%
average accuracy	59%
average run time	850 seconds
nodes	250
accuracy range	54% - 60%
average accuracy	58%
average run time	750 seconds
nodes	750
accuracy range	62% - 64%
average accuracy	63%
average run time	947.6 seconds

Size of Lexicon	
training reviews	10,000
testing reviews	3,000
hidden layers	3
nodes	500
lexicon freq range	500 > x > 5
lexicon size	15,022
accuracy range	59.5% - 62.3%
average accuracy	61%
average run time	555.5 seconds
lexicon freq range	500 > x > 50
lexicon size	2,929
accuracy range	62% - 64.7%
average accuracy	64%
average run time	353.4 seconds
lexicon freq range	750 > x > 50
lexicon size	3,082
accuracy range	66.2% - 66.9%
average accuracy	67%
average run time	350.2 seconds

lexicon freq range	1000 > x > 50
lexicon size	3,159
accuracy range	65.7% - 69.9%
average accuracy	68%
average run time	352.5 seconds
lexicon freq range	1000 > x > 100
lexicon size	1,709
accuracy range	65.5% - 69.6%
average accuracy	68%
average run time	325.5 seconds
lexicon freq range	1000 > x > 75
lexicon size	2,242
accuracy range	67.4% - 68.9%
average accuracy	68%
average run time	337 seconds