Sentiment Analysis with Naïve Bayes and Neural Network

Athena Mason Megan Ostby Neil Brommer

March 19, 2018

Abstract

We attempted to analyze the sentiment displayed in a collection of movie reviews, using three different approaches. Our initial approach was less successful than the subsequent attempts, with an accuracy no higher than 32% on a 0-4 rating system and 64% on a binary system. Later attempts proved more fruitful. The Naïve Bayes model achieved an average accuracy of 36% for the numerical classification and up to 74% for the positive/negative classification. The neural network results proved similar, with 39% and 77% accuracy for the respective rating systems.

1 Introduction

The idea of analyzing the emotion of a piece of text is not a new idea. Discovering the feeling behind a text using the processing power of computers is something that many different people have attempted over time. The challenge with this task is found in the fact that machines are not intuitively able to comprehend emotive intent as people are. Thus, it is necessary to train the computer to interpret the data it receives in order to perform sentiment analysis.

We used three different approaches to attempt to analyze the sentiment displayed in movie reviews. In our initial approach, discussed in section 3, we used Rapid-Miner Studio to classify the texts. This approach proved unfruitful, so we then moved on to using a Naïve Bayes classifier and a convolutional neural network, described in sections 4 and 5, respectively. These two approaches proved to be more successful than our initial attempt. In our final sections, we discuss the results and potential future work.

2 Related Work and Background

A considerable amount of research has been done on the subject of sentiment analysis. One group focused on analyzing tone of informal comments on the internet platform MySpace. They found success with their sentiment analysis by giving more weight to words that have influence, as well correcting spelling on words that were shortened with different slang terms[1].

Another group utilized appraisal groups, which focused on different categories such as “very good” or “not terribly funny” rather than positive or negative association. For this they used a bag of words method, which is fairly standard in terms of sentiment analysis, and appraised that to generate associations with their categories[2]. Although sentiment analysis has such a wide variety of methods approaching the problem, the vast majority of them involve creating a list of the words present in the document and giving each word an associated weight with the labels being analyzed.

3 Methodology and Design

Given a movie review, our goal was to classify how positive or negative it is on a scale of 0-4. We also tried using a classifying the reviews on a simple positive/negative scale to see how it would affect our results.

Our training data is a list of movie reviews with an associated 0-4 rating. For our positive/negative test we modified the given training data so that 0-2 became negative and 3-4 became positive. To check the accuracy of our models, we used split validation saving part of our training data to use as test data.

3.1 Initial Approach

We began by testing the various models and preprocessing options that RapidMiner Studio offers. We used RapidMiner's first party Text Mining & Sentiment Analysis extension. We tested the K-NN and Naïve Bayes models in combination with a variety of preprocessing options, such as filtering out stop words, filtering out words that were too short or too long, using word stemming, and using n-grams.

Our best result with five classes was a 32% rating from using a K-Nearest Neighbor model. In this test, we set k=9, and used word stemming, removing stop words, and filtering tokens (words) by their length to pre-process the data. However, the K-NN model takes a significant amount of time to run, approximately an hour with our data.

The Naïve Bayes model was the next most accurate model with 27% accuracy, but it took only a few minutes to run. We decided to use Naïve Bayes as our model of choice going forward due to its speed allowing us to work more quickly.

After testing the various models with the 0-4 rating system, we tested again using the positive/negative rating system. Using the most accurate K-NN model from the 0-4 test gave us a result of 64% accuracy.

At this point, we decided that it would be better to switch from RapidMiner to a different tool. RapidMiner was useful for quickly testing available options, but we would have more control with something else. We decided to use two different approaches in Python to work further with our model, one using a Naïve Bayes model and one using a convolutional neural network.

3.2 Naïve Bayes

The Natural Language Toolkit (NLTK) is package built for Python that was designed to work with data concerning the human language[3]. NLTK contains tools for all of the pre-processing that we did previously in RapidMiner, as well as many others, all with a focus on language processing. The package also had a built in Naïve Bayes classifier that was much more suited to interpreting the sentiment of words. RapidMiner's Naïve Bayes is a Gaussian classifier, meaning that it expects data with a normal distribution in order to interpret it. Unfortunately sentiment analysis does not lend itself to having a normal distribution. The NLTK classifier on the other hand utilizes the Bernoulli model of classification, which categorizes data based on whether or not it exists. This factor makes the Bernoulli model much more suited for text classification.

We are utilizing a bag of words model, where all the words in the data set are given a positive or negative association based on the number of times it occurs in a given review of that type. This bag of words, along with the data itself, is then sent through the NLTK classifier for Naïve Bayes and processed. The output result gives us a general accuracy of 70-74% when utilizing a binary negative/positive data set, and around 36% when using a 0-4 ranked data set. The model determines what words were the most informative for evaluating if the results were positive or negative. For example, words such as stupid, ugly, and offensive were more frequently found in negative reviews. A listing of the top 10 most informative words is included in Tables 1 and 2.

stupid = True	Neg:Pos = 16.4 : 1.0
ugly = True	Neg:Pos = 12.2 : 1.0
devoid = True	Neg:Pos = 12.2 : 1.0
beautiful = True	Pos:Neg = 11.9 : 1.0
manipulative = True	Neg:Pos = 11.1 : 1.0
sappy = True	Neg:Pos = 11.1 : 1.0
car = True	Neg:Pos = 11.1 : 1.0
future = True	Neg:Pos = 11.1 : 1.0
annoy = True	Neg:Pos = 10.1 : 1.0
pas = True	Neg:Pos = 10.1 : 1.0

Table 1: Most Informative Features: Pos/Neg

stupid = True	Neg:Pos = 16.4 : 1.0
ugly = True	Neg:Pos = 12.2 : 1.0
devoid = True	Neg:Pos = 12.2 : 1.0
beautiful = True	Pos:Neg = 11.9 : 1.0
manipulative = True	Neg:Pos = 11.1 : 1.0
sappy = True	Neg:Pos = 11.1 : 1.0
car = True	Neg:Pos = 11.1 : 1.0
future = True	Neg:Pos = 11.1 : 1.0
annoy = True	Neg:Pos = 10.1 : 1.0
pas = True	Neg:Pos = 10.1 : 1.0

Table 2: Most Informative Features: 0-4

3.3 Convolutional Neural Network

The final approach we tried for our project was classifying our data using a convolutional. Like the Naïve Bayes classifier, this was programmed in Python. We used Keras[4], an open source library for Python, to build our network. Keras sits on top of Google's Tensorflow[5] machine learning library and provides a simpler interface for building neural networks. In order to ensure a fair comparison between the two approaches to classifying the data, we pre-processed the data in the same way for the neural network as we did for the Naïve Bayes model.

The final layer or layers of a convolutional neural network operate the same way as a standard multi-layer perceptron. However, additional processing is performed through specialized layers before the data is passed to the fully-connected layers for final weighting and classification. These specialized layers help to identify features of the data, which can increase the accuracy of the neural network's predictions.

Convolutional neural networks were originally implemented for image recognition, but they can be adapted for use in text processing. When processing images, the network first translates the image to a matrix of pixels. A smaller matrix, called a kernel, passes over the entire matrix one section at a time and performs elementwise multiplication over the pixels in that section. The result is written to another matrix (called a feature map) and then the kernel advances to the next section[6]. This process is called convolution. Convolution can also be performed on text, in one dimension rather than two, by translating words into vectors called embeddings. We used Global Vectors for Word Representation (GloVe), an unsupervised learning algorithm developed at Stanford[7], to do this.

Our final model achieved an accuracy rating of up to 77%. The architecture of this model was as follows:

1. **Embedding Layer**: Translates the words into embeddings to be passed to the convolutional layer. We used embeddings of length 300.

2. **Conv1D Layer**: Performs the convolution on the received embeddings. In our most effective model, we used a softmax activation function, with 128 filters, and a kernel stride of 5.

3. **Max Pooling Layer**: Subsamples the outputs from the convolutional layer to reduce the number of nodes, which increases processing speed and reduces the risk of overfitting the data.

4. **Fully-Connected Layer**: Assigns weights in the same way as a layer in a multi-layer perceptron. In our most successful model, we used a ReLU activation function.

5. **Dropout Layer**: Forgets some of the learned information, in order to reduce the risk of overfitting. In our most successful model, we set a dropout rate of 0.3.

6. **Fully-Connected Layer**: Predicts the class.

We ran the model for three epochs, meaning the weights and errors were passed back through the model twice before the final predictions were given. With more epochs than this, we found that the data was being overfitted; the accuracy for the training data increased dramatically (over 90% in some cases), but the accuracy for the testing data tended to drop or see no marked improvement.

We built a variety of models, which included changes to the number and organization of layers and alterations to the other variables, such as activation functions, number of filters, et cetera. However, at best these models resulted in roughly the same accuracy with a similar run time, and at worst, they dramatically increased run time while reducing the accuracy of the model.

4 Results

Table 3 shows a summary of our best results for each of the above approaches.

	Rapid Miner	**Naïve Bayes**	**Neural Net**
Pos/Neg	64%	74%	77%
0-4	32%	36%	39%

Table 3: Results

Because we were using the split validation method to test our models, the results were somewhat variable. However, all results for the same model were within 3-4% of the maximum accuracy.

5 Conclusions and Future Work

Based on the results described above, we have determined that the Naïve Bayes classifier was ultimately more successful for this project. With the libraries available for Python, the Naïve Bayes model is very simple to implement, runs quickly, and involves very little guesswork. Conversely, even using Keras, which is designed to simplify the process of building machine learning models, the neural network was significantly more complicated to develop. Furthermore, the process of optimizing the neural network was essentially trial and error. The neural network also took several minutes to run for each model. Given the sheer number of models attempted, this added up to a significant time cost. While the neural network provided a small increase in accuracy for both the positive/negative and 0-4 classifiers (approximately 3 percentage points for both), this is not a significant enough increase to offset the added time and effort.

In the future, further improvement would focus on optimizing the data before passing it to a model. There is very little we could do to improve the Naïve Bayes model itself. Possible pre-processing approaches that we did not implement include filtering out words based on length, evaluating n-grams, and removing infrequently used words. Another method of optimizing the data requires adding artificial weights, manually making some words more important for classification than others. As machine learning technologies continue to improve, more potential approaches could develop.

References

[1] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas, "Sentiment strength detection in short informal text," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 12, pp. 2544–2558.

[2] C. Whitelaw, N. Garg, and S. Argamon, "Using appraisal groups for sentiment analysis," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, (New York, NY, USA), pp. 625–631, ACM, 2005.

[3] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

[4] F. Chollet *et al.*, "Keras." `https://github.com/keras-team/keras`, 2015.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[6] ujjwalkarn, "An intuitive explanation of convolutional neural networks." `https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`. Accessed: 2018-03-09.

[7] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–1543, 2014.

A Running Our Programs

A.1 Naïve Bayes

Syntax: `python SentimentAnalysisNaiveBayes.py data.csv`

"`data.csv`" requires two columns: Phrase and Sentiment. The sentiment values don't need to be any specific values. 0-4, positive/negative, or any other values will work.

This requires NLTK (`http://www.nltk.org/`) to be installed. When the program is run, NLTK may prompt you to download additional components.

A.2 Neural Network

Syntax: `python SentimentAnalysisCNN0-4.py`

This requires NLTK (`http://www.nltk.org/`), Keras (`https://keras.io/`), and TensorFlow (`https://www.tensorflow.org/`) to be installed. It also requires the `'glove.6B.300d.txt'` GloVe file (`https://nlp.stanford.edu/projects/glove/`). When the program is run, it may prompt you to download additional components.

Syntax: `python SentimentAnalysisCNNPosNeg.py`

This requires NLTK (`http://www.nltk.org/`), Keras (`https://keras.io/`), and TensorFlow (`https://www.tensorflow.org/`) to be installed. It also requires the `'glove.6B.300d.txt'` GloVe file (`https://nlp.stanford.edu/projects/glove/`). When the program is run, it may prompt you to download additional components.