Sentiment Analysis on Rotten Tomatoes Movie Reviews

Yuliya Astapova, William Murdy, Alexander Tairbekov

The Problem

Our project is to take a one-sentence movie review and classify it based on how positive/negative it is. We had a training set of movie reviews that all have a sentiment value based on how positive/negative the movie review is, with 0 being the most negative value, 4 being the most positive, and 2 being a neutral score.

Feature Extraction

We extracted features from the data in the form of a bag of words. To learn the bag of words, we used Scikit-Learn. We first took the sentences from the training set and removed the non-letter characters. Then, we looked at each word in the sentence and removed any stopwords (a, the, is, not etc.). We then turned each sentence into a vector representation. Finally, we took those vectors and created a vocabulary based off all the words that appear in the training set.

Our Naive Bayes Implementation

For our own implementation of naive bayes, we used the multinomial model. The model we used is $P(Y, W_1 \ldots W_n) = P(Y) \prod P(W_i \mid Y)$, where Y is the sentiment value Y = {0, 1, 2, 3 ,4}; $W_1 \ldots W_n$ are the words in the sentence to be classified; and P(Y) is the prior probability, which is calculated by taking the number of sentences in each category and dividing by the total number of sentences in the training set. $P(W_i \mid Y)$ is the probability of the i^{th} word in the sentence being in each category of Y.

To train the classifier we, first calculated the priors. This is an array of five values that correspond to the probabilities of a sentence being in each of the 5 categories. Next, we calculate the probabilities for each word. To do that, first we separated the training data into 5 different sets based on the sentiment value. Then, we go through each word in the vocabulary and and calculate the probability of that word showing up in a sentence for each sentiment value. So, at the end of the training, we have a dictionary that, for each word in the vocabulary, has an array of five values that are the probabilities that that word is in a sentence of that sentiment value.

We used Laplace Smoothing to smooth the data. Laplace Smoothing works by adding a “fake” instance of each word. This is used to prevent any word from having a zero probability, which will cause problems later when trying to classify a sentence. We tried multiple values for the Laplace Smoothing, and found the best value to be one or two.

After training the classifier, we are ready to start classifying movie reviews. To classify a movie review, we create an array that will hold the probabilities of each sentiment value. Following the equation $P(Y, W_1 \ldots W_n) = P(Y) \prod P(W_i \mid Y)$, we first take the prior probability. Then, we multiply the probabilities of each word in the sentence. We actually take the log of each probability and add them together to prevent any problems that could happen if the probability gets very small. And we classify the sentence according to the sentiment that has the highest probability.

Naive Bayes Results

To train the classifier, we used a training set with 8463 reviews. When we extracted the features from the training set, we had a vocabulary with 5000 words. We used a test sets of 49 and 347 sentences. For our own implementation of Naive Bayes, we got an accuracy of 40.8163%. We tested multiple values for the Laplace Smoothing value, and settled on k = 1.

K-Nearest Neighbors

The K-Nearest neighbors is a possible future implementation that relies on using a parse tree structure to identify the dimensionality of the sentiment phrase. Identifying such dimensionalities is difficult due to sentence structure. Here, we identify one possible way of constructing such a feature set.

To train the K-Nearest Neighbors classifier, first we load the training data into a database. Then, for each sentence, we create a parse tree. Each phrase is then hashed and assigned a sentiment value. The feature vector is determined by the structure of the parse tree. To classify a sentence, we first construct a parse tree for the sentence. Each leaf of the parse tree is either found in the database or, if it's not found in the parse tree, then it is assigned a neutral value (2). Then, we select all the subtrees from the training data that match the parse tree from the sentence we want to classify. We use the sentiment values as features to run KNN. The code for loading the KNN data is provided, but the code for running KNN is not included yet. This requires an installation of the Redis in-memory store and Python, as well as the redis-py library.

Conclusions

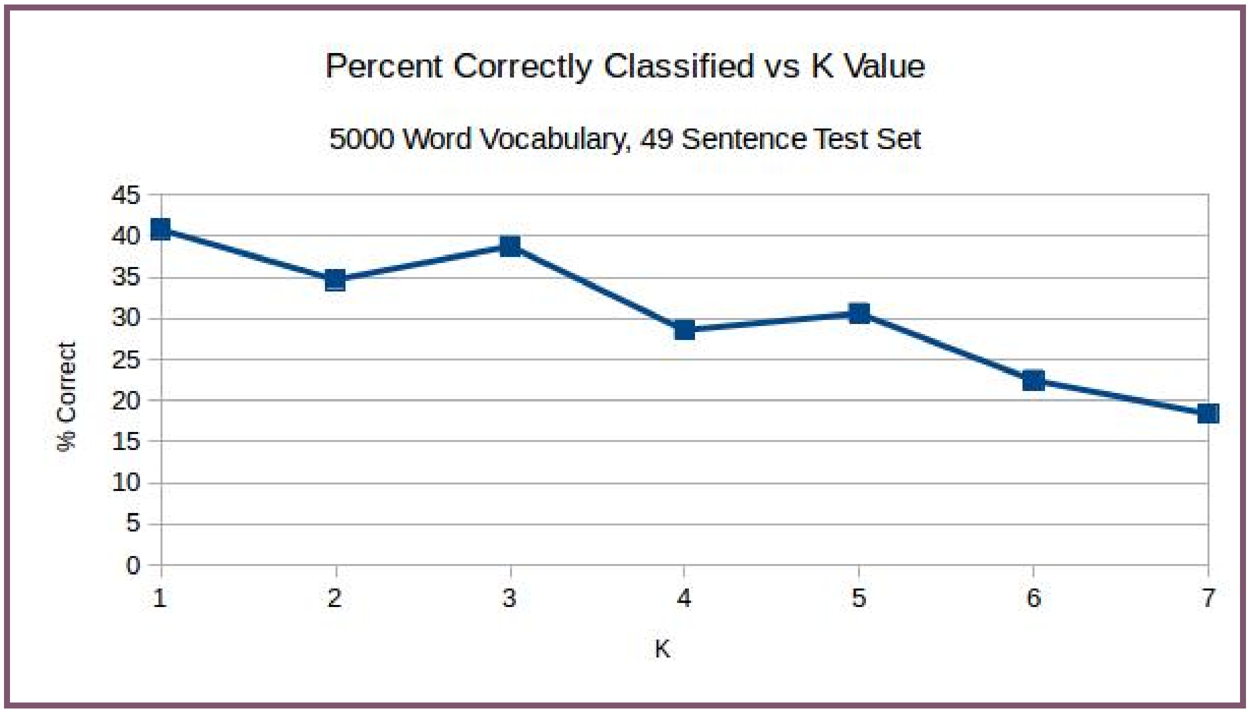
In the end the scikit-learn's implementation outperformed ours. The scikit-learn's implementation got an accuracy of 48.9796% correct, while our naive bayes implementation got a 40.8163%. We also learned that the accuracy increased with the vocabulary size. It's also possible that there wasn't enough crossover in the vocabulary in the sentences.

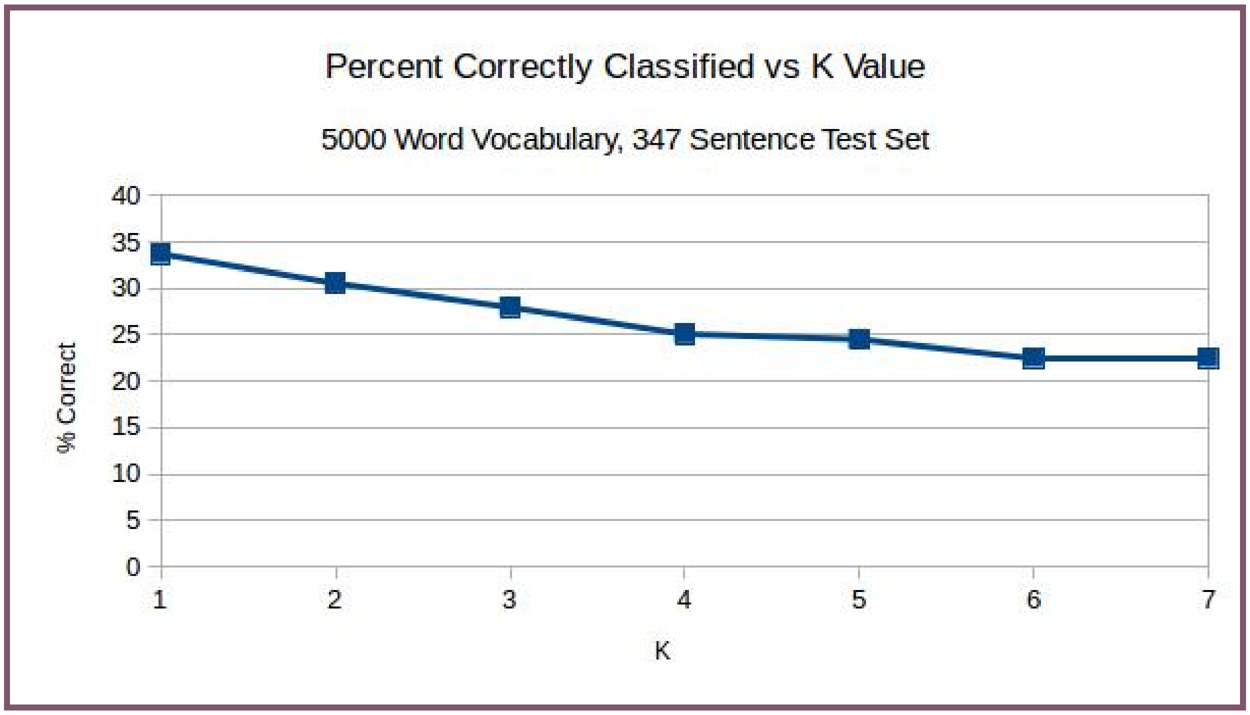
Running our Classifier

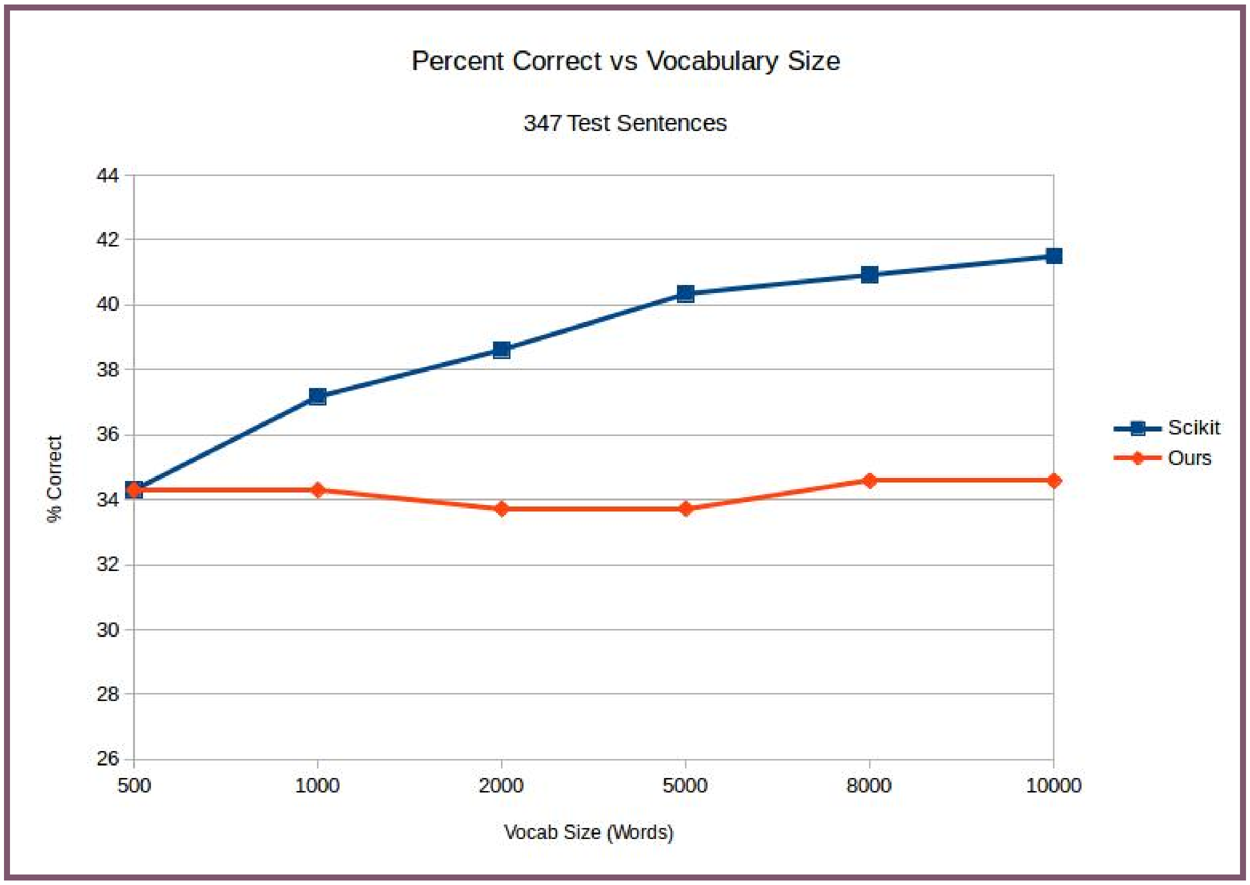
To run our classifier, you will need pandas (http://pandas.pydata.org/), numpy (http://www.numpy.org/), nltk (http://www.nltk.org/), and scikit (http://scikit-learn.org/stable/) for the naive bayes method. For the KNN method, you will need redis (https://pypi.python.org/pypi/redis).

For naive bayes, after you have all the packages installed, all you need to do is to run Classifier.py and that will create the bag of words, create and train the scikit-learn naive bayes classifier, and classify the test set. It will then create and train our own classifier. It will show the results and accuracy for each step of the Laplace Smoothing for k = 0 to k = 9. Classifier.py is set by default to train the classifier on a vocabulary of 5000 words and to test the it on the set of 347 test sentences. To change the vocabulary size, change the value for max_features on line 41. To change which of the test sets the classifier uses, change which of either line 60 or 61 is uncommented.

Graphs of Results






Percent Correct vs Vocabulary Size
347 Test Sentences
% Correct
44
42
40
38
36
34
32
30
28
26
500
1000
2000
5000
8000
10000
Vocab Size (Words)
Scikit
Ours