Eric Leung, Mohammad Asfour

May 11th, 2020

CS 4563 - Intro to Machine Learning

Professor Christopher Musco

Sentence Sentiment Classification

Our project was on text sentiment analysis, specifically to classify sentences as being either positive or negative. The main question we wanted to answer was what is the best way to predict and classify the sentiment of a short text. Another question was how to extract the most important words from a sentence and if those key words are enough to classify a sentence.

One of the datasets[1] we used consisted of 3,000 Amazon, Yelp, and IMDb reviews labeled with a 0 if it were negative and a 1 if it were positive. We also used two other datasets consisting of Amazon reviews on arts, crafts, and sewing[2] and fine foods[3] which had each had around 500,000 reviews. These two additional data sets had review scores ranging from 1-5 so to make it more consistent, we gave them labels of 0 and 1 instead of negative and positive respectively. Reviews that had low scores of 1-2 were given 0 while reviews with high scores 4-5 were given 1. We chose to do this because reviewers usually explain the reasoning behind the low or high score so there should be enough words to do sentiment analysis on. To make everything consistent, we narrowed down all this data into two different datasets and all of the testing was done on them. The first being the Amazon review dataset on fine foods concatenated with the Amazon, IMDb, and Yelp data from UCI which we'll call Food dataset. The second being the Amazon review dataset on arts, crafts, and sewing which we'll call Arts dataset.

However, one thing we noticed was a large majority of the Amazon reviews were positive (as seen in figure 1). This class imbalance could have resulted in the negative sentiment classifier not performing as well. This is because there is a higher probability that the sample belongs to the positive sentiment class. Another possibility is there is a much greater number of

[1] http://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences
[2] https://nijianmo.github.io/amazon/index.html
[3] https://snap.stanford.edu/data/web-FineFoods.html

positive sentences to train on. With more time, we would have found more balanced datasets or done more in terms of adjusting the various classifiers.

To process the data, we removed punctuation and stop words such as “the” and “is.” Removing stop words was appropriate because words that can be used to predict sentiment are left. On the other hand, words with less meaning are removed.

In order to turn this text data into something quantifiable, we used word embeddings. First, we tried to train our own word2vec model using gensim. To do that, we used the gensim Phrases package to automatically detect common phrases (bigrams) from a stream of sentences. Then, using these sentences, we set up the parameters of the model and built the vocabulary from the bigram sentences and finally trained our model. It’s worth noting that changing the amount of sentences and some of the parameters of the model made a good difference. Although we did not finish playing around with the parameters, we managed to get an increase of about 5% accuracy by only changing some of the parameters. For example, changing the windows parameter from 2 to 10 increased the accuracy of about 2.9%.

Also, we noticed that increasing the data would make a huge difference; using roughly ten thousand sentences yielded bad results (as shown in figure 2). Furthermore, since this model was trained using the Foods dataset, the model reflected it. For example, in figure 3, the most similar words to the word “amazing” were “weak” and “too sweet.”

```
w2v_model.wv.most_similar(positive=["amazing"])

/usr/local/lib/python3.6/dist-packages/gensim/mat
  if np.issubdtype(vec.dtype, np.int):
[('look', 0.9999381899833679),
 ('used', 0.9999381303787231),
 ('nothing', 0.9999378323554993),
 ('all,', 0.9999376535415649),
 ('short', 0.9999374747276306),
 ("it's", 0.9999369978904724),
 ('experience.', 0.9999368190765381),
 ('up', 0.9999368190765381),
 ('their', 0.9999366998672485),
 ('see', 0.9999358654022217)]
```

Figure 2. Trained model on 10,000 sentences

```
w2v_model.wv.most_similar(positive=["amazing"])

INFO - 06:27:30: precomputing L2-norms of word weig
/usr/local/lib/python3.6/dist-packages/gensim/matut
  if np.issubdtype(vec.dtype, np.int):
[('weak', 0.9999464750289917),
 ('too_sweet.', 0.9999410510063171),
 ('bland', 0.9999404549598694),
 ('too_sweet,', 0.9999401569366455),
 ('wonderful.', 0.9999399185180664),
 ('fantastic', 0.9999390840530396),
 ('truly', 0.9999364614486694),
 ('delicious!', 0.9999364018440247),
 ('bitter,', 0.9999357461929321),
 ('Tastes', 0.999934732913971)]
```

Figure 3. Trained model on 100,000 sentences

In order to have something to compare the results of our model with, we also used a pre-trained Google News word2vec model[4]. This model was trained on Google News articles so the most similar words to the word "amazing" were "incredible" and "awesome." A factor that could have improved our model would be better processing of punctuation and training on more data. The relatively small dataset of 100,000 reviews could have led the vector similarities between words with different meanings to be too large.

```
1 model.most_similar('amazing')[:5]
```

```
[('incredible', 0.9054000973701477),
 ('awesome', 0.8282865285873413),
 ('unbelievable', 0.8201264142990112),
 ('fantastic', 0.778986930847168),
 ('phenomenal', 0.7642048001289368)]
```

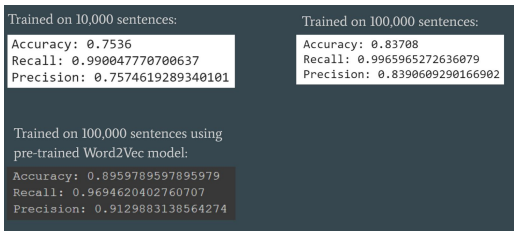
With these word2vec models ready, we decided to average all the word vectors in a piece of text together in order to embed a review. We chose this method due to its simplicity and because our review texts were of variable lengths and generally weren't too long. There were some other methods that we looked into that scaled the importance of a word based on its frequency in the document, but because the reviews already had stop words removed and were relatively short texts, we felt it was not necessary.

Another method we tried was using Google's universal sentence encoder[5]. This is a new model released in 2018 that embeds text into vectors similar to word2vec. However, one huge advantage is that it is trained and optimized for greater-than-word length text.

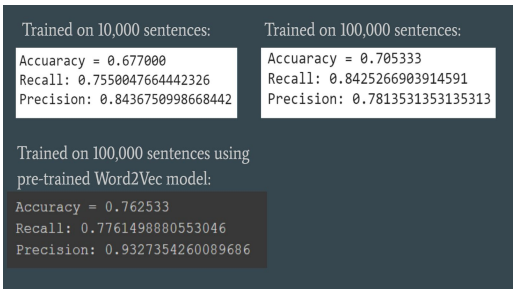
After the embeddings were complete, many different classification methods were used in order to classify the data as positive or negative. All of these were done using smaller random subsets of 100,000 and 200,000 reviews for the Food and Arts dataset respectively in order to save time. To judge performance, we split our data into train and test data. Then, with each method, we measured the accuracy, precision, and recall. As mentioned before, the majority of the data were labeled 1 for positive so our simple baseline was labeling everything as positive. This baseline of all 1's achieved accuracies of 84.37% and 94.29% for the Food and Arts dataset respectively.

[4] https://github.com/3Top/word2vec-api Google News
[5] https://tfhub.dev/google/universal-sentence-encoder/4

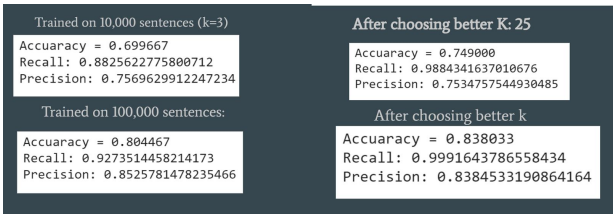
The first method used was logistic regression. Using the pre-trained word2vec model, this method yielded accuracies of 89.53% and 95.11% for the Food and Arts dataset respectively.

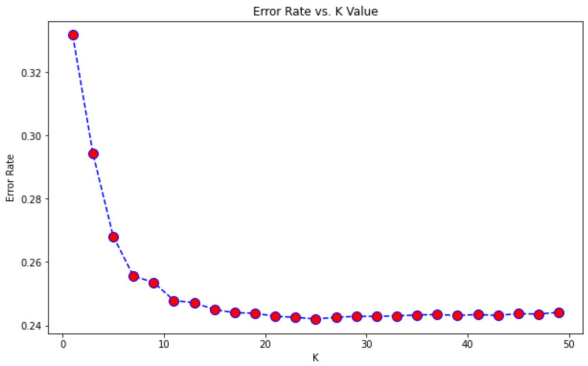


The second method used was Naive Bayes, specifically with a gaussian distribution. This yielded results of 70% accuracy for our trained model while giving 76% accuracy for the pre-trained model both on the Food dataset. This method performed the worst with the lowest accuracy.

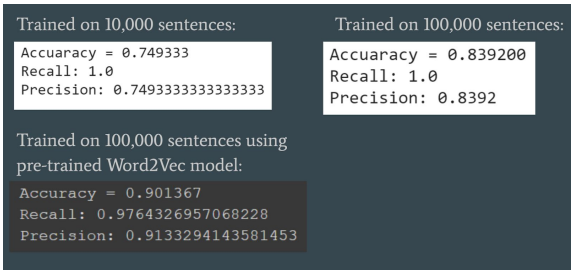


The third method used was K-nearest neighbors. This yielded results of 83.8% for our trained model on the Food dataset. An increase of about 13% from what we've got before choosing an optimal k and increasing the data.





Lastly, we used support vector machines. Using the pre-trained word2vec model, SVMs yielded accuracies of 90.14% and 95.49% for the Food and Arts data respectively. Ultimately, this method worked best for both of our datasets with a decent increase in accuracy compared to the



baselines. As a result, we chose to also test the performance of Google's universal sentence encoder with the SVM method. This resulted in an accuracy of 96.66%, recall of 99.28%, and precision of 97.22% which is the best we've seen yet.

This bump in performance is probably due to advantages of sentence embeddings directly over averaging all the word vectors in a sentence. Even after removing stop words, there are inherent words that have more meaning towards sentiment analysis. For example, a sentence with the words "product" and "good" will average the two of these vectors together even though "good" is a better predictor if a review is positive. Another limitation of averaging word vectors is oftentimes reviews explain both what's good and bad about a product. Even if the number of positive things said in a review far outnumbers the number of negative things, averages are greatly affected by outliers leading to the averaged vector being misclassified. Another advantage of Google's sentence encoder is that the authors claimed the encoder was trained on "a number of natural language prediction tasks that require modeling the meaning of word sequences rather than just individual words." The ordering of words is not something taken into account when averaging word vectors together so a lot of the sentence's meaning could be lost.

In conclusion, our results were decent and could be improved with more fine-tuning and larger datasets. If time permitted, we could have tried other methods of getting sentence embeddings from word embeddings. One potential method is to try using single value decomposition to concatenate a fixed number of top principal components instead of averaging all the word vectors. Another potential method is to group the words into fixed lengths and use these n-grams to find sentiment rather than just using individual words. Lastly, bag-of-words could have been an option to quantify the data without the need of word2vec altogether. However, this would have needed much larger datasets and longer text. One way we could have improved was by finding a dataset with many negative reviews since as mentioned above, the all positive baseline guess was very accurate due to our imbalanced classes. Going beyond our project, a new question that we could have asked is could we change the labels and classify the text beyond positive/negative sentiment. For example, for detecting if something is hate speech or if something is spam.