Lab 10 Report

Adam Lewis

May 2016

1 Introduction

My task is to create and improve a textual entailment system that will predict, given a string of text and a hypothesis, whether the hypothesis entails, contradicts or is neutral with respect to the given string of text. As a baseline, I have implemented a machine learning based approach with a perceptron that accepts feature vectors for each of the premise-label pairs.

Currently this produces poor results on the testing data, giving an accuracy of 0.3229. This is because it does not yet use features from the two sentences and their label so it is entirely random (3 states: 'entailment', 'neutral', 'contradiction' therefore 1/3). This clearly has large scope for improvement.

To improve this, I plan on using the perceptron based approach and improving upon my feature vectors. The Improvements I will implement are as follows:

- I will implement cross uni-grams across the premise and the hypothesis and use these as features to predict the label.
- I will also implement cross-bigrams across all the bigrams in the premise and the hypothesis. By combining this with the cross uni-grams approach, I hope to achieve much better results.
- I will experiment with the sparse array created by the cross n-gram models and instead of simply doing a 1 hot encoded array, I will make the value relative to the counts of each token.
- I will use word2vec[2] to create the feature vector for each premise and hypothesis. I then plan to append this to my feature vector for each element in the list.

By taking advantage of these improvements, I hope to improve my classifiers accuracy to at least 60 percent.

2 System Description

Bowman et al.[1] manages to attain a staggering 78.2 percent accuracy on a test set of data. In order to achieve this, he uses a mixture of cross uni-grams, cross-bigrams and a lexicalized classifier. In my implementation, I will make use of cross uni-grams and cross-bigrams as well and take note of the classifier performance.

To implement the use of cross uni-grams in my classifier as a feature, I use scikit's DictVectorizer class to create a sparse one hot encoded array for each of the premise-hypothesis pairs in the training set. For each pair of words between the premise and the hypothesis, I combine them with an underscore and use this as one 'feature'.

To further improve upon this feature set, I use gensim's word2vec[2]. I use the Google news vectors as my word2vec model. Although this model is not necessarily linked with textual entailment, It is trained on a huge amount of data that is far more than I could ever train on and will likely provide better results because of the fact that is trained on such a huge data set.

In my continuous word representation approach using word2vec, I combine the word vectors for each token in the premise by summing them using my `sentence_to_vec` function. I then do the same for the hypothesis and take the difference between the two. The result is a 300 length vector representing the difference between the hypothesis and the premise. I append this to my feature set which is then passed in to the classifier.

3 Experiments

Improving on the baseline didn't take much as my baseline was merely the random allocation of labels. However, I believe that by using cross uni-grams, I have improved upon this vastly.

By using the sparse array obtained from the DictVectorizer as the feature set, it produces good results when passed to the perceptron (Figure 2). The cross uni-grams as features are combined with the rest of the features to be used in the classifier. Implementing cross uni-grams alone, without any other features improves the accuracy by about 40 percent. However, the use of cross bi-grams only further improves accuracy by approximately 1 percent on the development data set.

Baseline	**Cross Uni-grams**	**Cross Bi-grams**
0.3229	0.7014	0.7132

Figure 1: Accuracy comparisons between the baseline, cross uni-grams, and cross uni-grams with cross bi-grams on the development data set

Baseline	**Cross Uni-grams**	**Cross Bi-grams**
0.3159	0.6986	0.7165

Figure 2: Accuracy comparisons between the baseline, cross uni-grams, and cross uni-grams with cross bi-grams on the testing data set

Using the testing data set, which is much larger, you can see that my classifier still attains a relatively high accuracy. It's worth noting that when working with the testing data set, cross bi-grams made a more significant improvement of almost 2 percent.

To create my sentence vectors, I took the word2vec features (of length 300) of each word and took the element-wise addition of these in sequence. This gave me a sentence vector of length 300. I applied this function to the premise and the hypothesis of each pair and took the element-wise difference. I then appended this difference feature vector to the rest of the features.

With this implementation, the accuracy was worsened to 0.5243 percent. This is likely because I had to train and test on smaller sets. The reason is because calculating the 300 length feature vectors for each premise-hypothesis pair was too intensive and took too long on my machine.

4 Plots

The following plots were done from 300 to 3000 training data elements in increments of 300. This was done to test the classifiers on smaller amounts of training data and observe their performance.

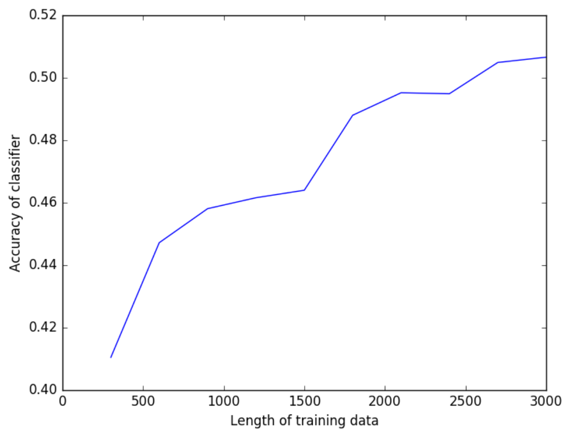

Figure 3: The learning curve of the cross uni-grams implementation

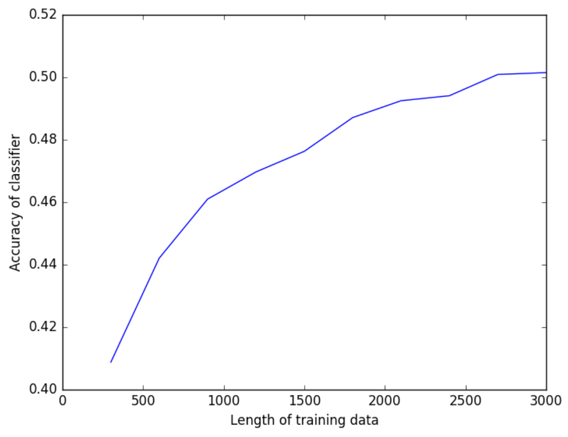

Figure 4: The learning curve of the cross uni-grams and cross bi-grams implementation

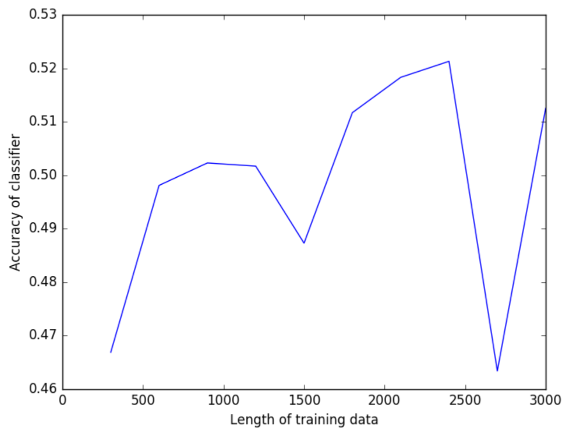

Figure 5: The learning curve of the cross uni-grams, cross bi-grams, and the word2vec implementation

As you can see, the clear winner here is the cross uni-grams and cross bi-grams approach. The learning curve for the cross uni-grams, cross bi-grams, and word2vec approach seems to be fairly inconsistent. The cross uni-grams and cross bi-grams implementation seems to have a smooth learning curve with good accuracy when the training data reaches around 3000 in length.

5 Pitfalls and Improvements

It seems my word2vec approach didn't nearly perform as well as I thought it would. Not only does it provide worse accuracy measures, it also is very inconsistent on smaller training data sets. To improve upon this, I alter my word2vec approach to perform another operation on the sentence vectors instead of simply taking the element-wise difference. I think that perhaps taking the cosine difference or even the euclidean difference between the premise and hypothesis vectors could show promise.

To improve my classifier even further, I would look to using the part of speech (POS) tags to help classify the premise-hypothesis pair. As Bowman et al.[1] gathers the cross uni-grams and bi-grams, they only use those that share a POS tag. As seen in their results, this is a large improvement over my classifier; roughly 8 percent.

I could also potentially improve my classifier by going to greater values of n in my n-grams. Perhaps tri-grams or quad-grams would be sufficient to improve the classifier further. By comparing the accuracies on larger data sets and the learning curve graphs of the uni-grams and the bi-grams, it looks as though this would only provide a slight improvement and likely smooth the learning curve further. Bowman et al.[1] talks in their paper about using 3 to 4 gram models but mainly focuses on uni-gram and bi-gram models so I only implemented these.

The python code referenced in this document can be found at the following url:
https://github.com/mr-adam-lewis/nlp-final-lab

References

[1] Christopher Potts Christopher D. Manning Samuel R. Bowman, Gabor Angeli. A large annotated corpus for learning natural language inference. Technical report.

[2] Radim Řehůřek. gensim: models.word2vec – deep learning with word2vec, 2016.