COM6513: Text Classification with the perceptron

Registration Number: 180128251

February 17, 2019

1 Implementation

The application is developed with Python 3 and was tested on a computer with a 32GB RAM and an Intel Xeon Inside at 3.33GHz. It accepts a single command-line argument which must be the destination of our data. I employed the use of quite a few libraries such as **numpy** for high-performance multidimensional array computations and **matplotlib** for data and graph plotting. The program consists of 4 functions, namely: **build_dataset**, which accepts 3 arguments being: *pos_class*, a list of all positively sentiment documents, *neg_class*, a list of all negatively sentiment documents and *gram*, the value of 'n' for an n-gram language model to be used. (Default value is 1). The function begins by taking the first 800 documents from the positive and negative data-sets each and creating a bag-of-words list of that data. It then proceeds to build the data that will be used for training and testing the accuracy of our model. For n-gram language models greater that one(i.e. bigrams and trigrams) stoplists were used as a preprocessing method to reduce the size of the model and improve speed. The **train** function which is used to train the model of our perceptron by updating the weight values after every iteration where the predicted value is not equivalent to the actual value. The **test** function is used to compute the accuracy of our model and the **run_program** function runs the entire application for the different types of the n-gram language models by combining the above mentioned functions.

2 Evaluation

Using the bag-of-words approach with the unigram language model, we obtained an accuracy of 50% after training our model with our data been sorted. This resulted in our weight values not being regularly updated as the initial weight values were zero(0) causing the updates to be aligned with the training labels. After randomizing the data with a seed of 180128251, an accuracy value of 71.75% is obtained indicating that there was more change in our model and more weight updates that made the model predict better. We obtained a learning progress of the model by performing multiple passes over the training instances and can observe that our model gets better after a number of iterations as indicated in the figure below. However using the average of our weights we obtain an accuracy value of 85.75%.

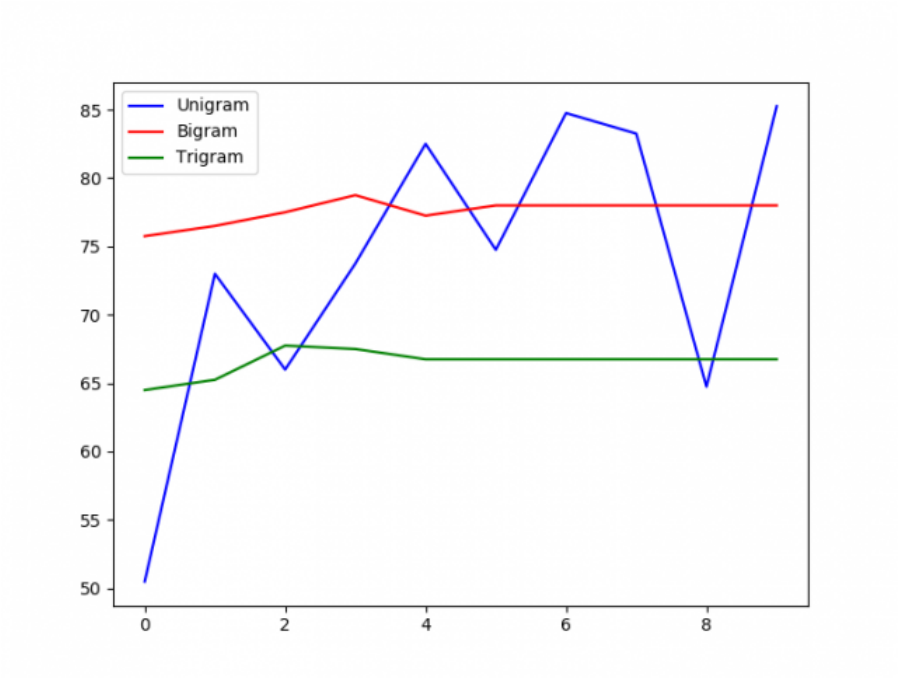

Figure 1: Learning progression for different learning progressions

In addition to the bag-of-words approach, Two different feature types namely: bigram and trigram language models were implemented. The bigram and trigram approaches involve the combination of two and three words respectively as data features when training the model. After training with the bigrams, An accuracy of less than 50% is obtained with the sorted data with a 74.5% after randomization. However, after performing multiple passes over the training instances, there isn't much accuracy increase compared to the unigram model and an eventual constant accuracy which indicates that the weight updates have seized. The accuracy using the average of all the weights is at 79% which is significantly lower than that of the unigram. A similar observation is made with the trigram where it has a relatively high accuracy of 56.75% with the sorted data but doesn't increase much when the data is randomized, having an accuracy value of 66.5%. It also seizes to update its weights after a few iterations and remains stable at a low accuracy value yielding an accuracy value of 65.75% when all weights are averaged.

Using the bag-of-words approach, we obtained the highly weighted features that are both positive and negative. In the table illustrated below, we can see the most positive and negative words in the data features as displayed on the left and right respectively. Words such as *bad*, *worst* and *unfortunately* are obvious words that are negative; likewise words like *great*, *best* and *well* can be characterized as positive. However, both tables include arbitrary words, that don't exactly positively or negatively affect the sentiment of a statement.

Word	Weight Value
bad	-315.07
plot	-160.01
any	-151.31
only	-148.44
worst	-140.54
unfortunately	-133.33
director	-130.12
if	-129.76
nothing	-128.29
boring	-123.15

Word	Weight Value
seen	156.63
great	145.73
very	133.63
see	132.96
quite	131.33
most	121.55
also	118.05
well	113.94
best	113.72
you	112.27

3 Conclusion

We can observe that our model is very biased to the data-set in which it is trained as it classifies domain-specific words such as *director* as highly sentimental words. The model will therefore not be appropriate for classifying data from other domains such as restaurant or hotel reviews as well as general sentences such as tweets. Regardless, the bag-of-words approach with a unigram model proves to provide a better solution for this type of classification task against other n-gram language models.