NLP Lab 4

Moritz Bohm, 140164022

Evaluation of results:

Binary Perceptron:

The binary perceptron produces results with an accuracy of 50%, which is just as successful as randomly guessing the label of each document as this is a binary classification problem. The main reason as to why this form of perceptron is so unsuccessful is due to the fact that the perceptron will only predict the label wrongly half of the time, meaning that the weights from the training function remain relatively unchanged. Furthermore, as the training data is not shuffled, the weights will remain unchanged as the first half of the dataset is processed. As the negative documents are extracted first due to the folder structure, the perceptron will classify all negative documents wrongly, meaning that the weights are updated to favour positive documents, leading to further misclassifications. This continues until the first positive document is processed, upon which the perceptron classifies the document correctly, and then will classify all other positive documents correctly towards the weights' positivity bias.

Perceptron improvements:

As shown by the results produced from the binary perceptron, it is not very successful. In order to improve the accuracy, the three improvements are implemented, and the results will be evaluated as follows:

Shuffling of training data:

Shuffling of the training data resolves the issues discussed in the binary perceptron evaluation, as now all documents will be processed in random order, the weights will be updated much more frequently due to an increase in misclassifications. This improves the accuracy considerably, bringing it up to 73.75%.

Multiple passes:

Multiple passes further improve the accuracy of the perceptron, due to the fact that the weights are calculated more than once, meaning that they become more refined. The implementation of multiple passes increase the accuracy of the perceptron to 80.5%, meaning that, even without implementation of weight averaging, the perceptron can be regarded as successful.

Weight averaging:

Averaging the weights improves the accuracy because, rather than taking the last computed weight for each word, it returns the average of the weights across all multiple passes. This gives a representation of the weights that better represents the change in weights across the multiple passes. The implementation of weight averaging increases the accuracy of the perceptron to 81.25%. While this increase isn't as significant as that brought on by the other improvements, it is important to note that the increase of accuracy according to the implementation of the perceptron will even plateau, as the results achieved no longer improve.

2 additional features:

Bigrams:

The first additional feature type which I implemented in my system is the extraction of bigrams for the documents. This means that instead of representing a document as a bag-of-words and its classification label, the document is represented as the words and bigrams in each document, stored in a Counter(), as well as the classification label. This improves the accuracy as bigrams provide additional context according to which the perceptron can classify documents. After the implementation of this feature, the accuracy of the perceptron increased to 81.75%.

Length of document:

The second feature which I implemented in order to improve perceptron accuracy is the inclusion of the length document as a weight. The logic behind this surrounds the fact that, during extraction of the document, I noticed that positive reviews contained more words in the bag of words than negative reviews. In order to prevent the perceptron from classifying the length as a word, I stored the length in each document Counter() under "$length", and simply treated it as another weight. The implementation of this feature led to an further increase in accuracy of the classifier to 82.75%. This increase suggests that the length of the document is a better classification feature than bigrams.

Most positively-weight features:

The 10 most positively weighted features for the classifier are as follows:

```
Highest weights: [(‘great', 154), ('most', 150), ('you', 147),
('movies', 136), ('see', 132), ('seen', 131), ('also', 124), ('always',
121), ('jackie', 120), (('to', 'the'), 118)]
```

The fact that ‘great’ is the single most positively-weighted feature indicates that the classifier is successful, as great is an extremely possible word. Furthermore, ‘most’, which is a superlative, also makes sense as being very positively weighted.

```
Lowest weights: [[('any', -124), ('boring', -136), ('why', -137), ('no',
-139), ('only', -141), ('have', -145), ('plot', -166), ('script', -170),
('worst', -173)]]
```

The most negatively-weighted feature is “worst”, which is a very negative word and as such indicates that the classifier processes the weights of the words correctly. Looking through this list of most negatively-weighted features, we can see that there are quite a few negative words, such as “boring” and “no”, which provide further evidence of correct classification.

Applying the classifier to a different domain would probably yield lower accuracy results. For example, the 4th highest-rated feature is ‘movies’ which would not be common at all in laptop or restaurant reviews. However, the fact that the classifier is trained in a very generalised way would imply that, given a different corpus, the weight vector will be tuned to whatever category of reviews that it is given.