Lab 1 report：Text Classification with the perceptron

1. A brief description of the code：

At the beginning, I build a function called ‘Unzip’ to extract the folder ‘neg’ and ‘pos’ and a function called ‘get_files’ to take the files name from folder ‘neg’ and ‘pos’ into a list, at the same time sort the files name.

The second step is to create training data and test data, I take the first 800 files as training data with label '1'(positive) or '-1'(negative) and the rest of files as test data with label '1'(positive) or '-1'(negative).

Then, I build a function called ‘Perceptron’ to train model, i.e. weight of words. I not only use the features ‘unigrams’, but also use ‘bigrams’ and ‘trigrams’ to create bag of words. In this process, I record the correct rate every 50 files to plot the learning progress.

In order to get the more accurate weights, I build a function called ‘average_weight’ to get the average weight after many times iterations.

The function ‘top_features’ is to return the top 10 positively-weighted features for negative and positive.

The last function is ‘test_model’, which is used to test the accuracy of model(weights), I use the weight on the 400 test files to prediction their class and record the number of correct classification files.

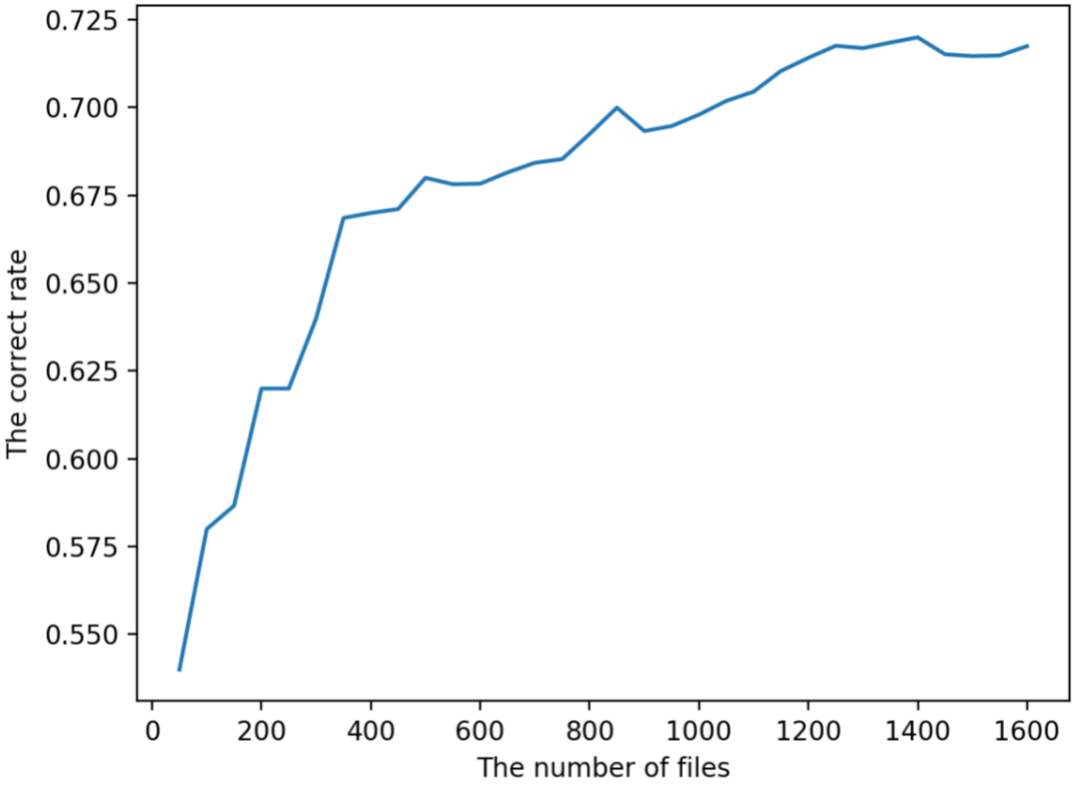
2. Result display：

```
When the feature type is : unigrams ,the correct rate is: 0.7975

When the feature type is : unigrams ,the top 10 positive features are: [('and',
83.9), ('will', 82.45), ('well', 82.3), ('most', 80.0), ('great', 79.55), ('very
', 78.8), ('life', 78.2), ('also', 73.5), ('seen', 70.7), ('you', 68.9)]

When the feature type is : unigrams ,the top 10 negative features are: [('script
', -73.05), ('so', -73.65), ('on', -75.15), ('if', -82.1), ('movie', -91.35), ('
plot', -94.15), ('only', -97.35), ('no', -100.9), ('have', -111.05), ('bad', -17
2.3)]


When the feature type is : bigrams ,the correct rate is: 0.845

When the feature type is : bigrams ,the top 10 positive features are: [('the bes
t', 20.8), ('he is', 14.8), ('many of', 14.6), ('which is', 13.55), ("and it's",
 12.9), ('even if', 12.75), ('one of', 11.95), ('is what', 11.45), ('and for', 1
1.35), ('with their', 11.25)]

When the feature type is : bigrams ,the top 10 negative features are: [('to do',
 -15.35), ('none of', -16.5), ('the only', -16.6), ('and then', -17.35), ('is so
', -17.5), ('at least', -17.6), ('supposed to', -19.1), ('have been', -19.8), ('
should have', -22.85), ('the worst', -24.05)]


When the feature type is : trigrams ,the correct rate is: 0.81

When the feature type is : trigrams ,the top 10 positive features are: [('of the
 best', 8.3), ('in the world', 6.1), ('saving private ryan', 5.9), ('many of the
', 5.8), ('as much as', 5.6), ('to keep the', 5.35), ('to see the', 5.3), ('the
film and', 5.25), ('of the series', 5.25), ('a lot more', 5.25)]

When the feature type is : trigrams ,the top 10 negative features are: [('to wor
k with', -5.35), ('the whole thing', -5.35), ('wild wild west', -5.5), ('have be
en a', -6.05), ('there is nothing', -6.15), ('for a film', -6.2), ('supposed to
be', -7.15), ('could have been', -7.55), ('should have been', -7.7), ('of the wo
rst', -10.5)]
```

3. Implement two feature types beyond bag-of-words. Discuss your choice of features. Does any of them help improve accuracy?

From above result, I not only use unigrams but also bigrams and trigrams as feature. I find that the best result is 'bigrams', whose correct rate reach at 0.845, then is the 'trigrams', which is 0.81, the worse is 'unigrams', which is only 0.7975. So these two feature types improve the accuracy.

4. Show the learning progress.



5. What are the most positively-weighted features for each class? Give the top 10 for each class and comment on whether they make sense (if they don't you might have a bug!). If we were to apply the classifier we learnt to a different domain such laptop reviews or restaurant reviews, do you think these features would generalize well? Can you propose better features for the new domain?

The most positively-weighted features for each class I have showed in above result. I don't think it will perform very well when it is applied the classifier to a different domain. Because different domains has different words to describe, some words are specialized, so we can't use this domain's model (weights) to prediction other domain's class. But we can use the same method(algorithm) to train different model, if we have other domain's training data, we can still get the model to predict the new data's class.