Homework_2

Part_1

Sentence S = “I always like foreign films”
Prior probability:
P(pos) = 0.4
P(neg) = 0.6

P(S|pos)= P(pos)* P(“I”|pos)* P(“always”|pos)* P(“like”|pos)* P(“foreign”|pos)* P(“films”|pos)
= (0.4)*(0.09)*(0.07)*(0.29)*(0.04)*(0.08)
= 0.00000234

P(S|neg)= P(neg)* P(“I”|neg)* P(“always”|neg)* P(“like”|neg)* P(“foreign”|neg)* P(“films”|neg)
= (0.6)*(0.16)*(0.06)*(0.06)*(0.15)*(0.11)
= 0.0000057

Since P(S|pos)=000000234 < P(S|neg)=0.0000057, so **neg** class will be assigned to the sentence “I always like foreign films”.

Part_2

#a

Algorithm:

1) Preprocess training corpus and testing corpus.
   a) Lowercase all words and erase punctuation.
   b) Combine all documents from each corpus into a .NB file, with the format of each document as <u>one example per line; each line corresponds to an example; first column is the label, and the other columns are feature values</u>.

2) Implement Na¨ıve Bayes classifier with bag-of-word (BOW) features and Add-one smoothing.
   a) Read training, test ← Given input file path/name
   b) Write Parameter model, output ← Given output file path/name
   c) Train Na¨ıve Bayes classifier
      i) Read all document in training
      ii) Count word(W) frequency from every classes(C)
      iii) Compute parameters of each word P(W|C) and prior probability P(C)
         (1) P(W|C) = (count(W in C)+1) / ((count(C)+|V|)
         (2) P(C) = (#document with class C) / (total number of document)

d) Test classifier
   i) Read each document(D) in test file
   ii) Compute the probability of each class assign for the document, P(D|C), by the training parameters
   iii) assigned_class ← argMax(P(D|C)), get the highest probability class.
   iv) Compare assigned_class to actual class
   v) Compute the classifier accuracy

#b

The small corpus after preprocess:

```
{'action': {'fast': 1, 'furious': 1, 'shoot': 1}}
{'action': {'furious': 1, 'shoot': 2, 'fun': 1}}
{'action': {'fly': 1, 'fast': 1, 'shoot': 1, 'love': 1}}
{'comedy': {'fun': 1, 'couple': 1, 'love': 2}}
{'comedy': {'couple': 1, 'fly': 1, 'fast': 1, 'fun': 2}}
```

The parameters of the model train by the small corpus of movie reviews: (Significant parameter show below only, full parameters list in file `movie-review-small.NB`)

```
Prior probability:
P(action) = 0.6
P(comedy) = 0.4

Probability of each word:
P(love | action) = 2.233688489803212e-05
P(fun | action) = 2.233688489803212e-05
P(fast | action) = 3.350532734704818e-05
P(shoot | action) = 5.58422122450803e-05
P(fly | action) = 2.233688489803212e-05
P(furious | action) = 3.350532734704818e-05
P(couple | action) = 1.116844244901606e-05
P(love | comedy) = 3.3506075768406e-05
P(fun | comedy) = 4.467476769120801e-05
P(fast | comedy) = 2.2337383845604003e-05
P(shoot | comedy) = 1.1168691922802002e-05
P(fly | comedy) = 2.2337383845604003e-05
P(furious | comedy) = 1.1168691922802002e-05
P(couple | comedy) = 3.3506075768406e-05
```

#c

Test document: `{fast, couple, shoot, fly}`

- Probability for class ‘action’ = `2.801642519444326e-19`
- Probability for class ‘comedy’ = `7.471213615204606e-20`

The class ‘action’ has a higher probability, so it is more likely to be class ‘action’.

#d

Accuracy:

Out of 25,000 test document files, there are 15,909 correct predictions and 9,091 incorrect predictions in the first test. The accuracy rate reaches 63.64%.

By investigating all the incorrect predictions, I had found out that most of those incorrect predictions have an error probability computed. Since all the parameters used to compute the probability class for the documents are very small, multiplying them results in a numerical underflow error. Thus, I replaced the computed method to sum of logs.

Using the sum of logs to determine the probability of class, the accuracy is significantly improved. **There are 20277 correct predictions and the accuracy rate reaches 81.11%.**

Further Investigation:

By looking at the frequency of words in each document, I found that some trivial words like "a", "the", etc. were repeated a lot in one document. They make up a large part of probability measurement. However, those words don’t seem to be helpful in distinguishing the class of a document. Especially some of these words have a big difference probability in different classes, such as ‘or’.

```
P(or | neg) = -8.188980846614683
P(or | pos) = -8.642192610510786
```

Compare to the word ‘good’, which should a significant difference for different classes

```
P(good | neg) = -8.633811272003822
P(good | pos) = -8.62125785928197
```

Assume a document is just a combination of these two words, “good or good good good”. Obviously the class neg will be assigned to the document, even though it should be a pos class document by intuition. Although it’s an extreme case, it should somehow affect the prediction correctness. In this example case, even 10 ‘good’ can’t cover 1 ‘or’. Not to mention if any document has the word 'or' and 'her' repeated multiple times.

Therefore, I think I can make a STOP WORDS list, which contains the trivial, meaningless words, for the program. We are going to ignore any words found from the document in the STOP WORDS list. Perhaps this can improve the accuracy by filtering out the noise and focusing on the significant words. But, it’s hard to tell that all the words in a document must be meaningless. So, we can still count all the words in the documents, but make them as binary count only, either appear or not [1,0]. For example the sentence “I wish I can help”, we will count ‘I’ as 1 only, instead of 2 during the class probability computation.

Experiment:

1) Apply STOP WORDS list:

```
stop_words = {'a', 'about', 'above', 'after', 'again', 'against', 'all', 'am',
'an', 'and', 'any', 'are', 'as', 'at', 'be', 'because', 'been', 'before',
'being', 'below', 'between', 'both', 'but', 'by', 'can', 'cannot', 'could',
'did', 'do', 'does', 'doing', 'down', 'during', 'each', 'few', 'for', 'from',
'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here', 'hers',
'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in', 'into', 'is', 'it',
'its', 'itself', 'me', 'more', 'most', 'my', 'myself', 'no', 'nor', 'not',
'of', 'off', 'on', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 'same', 'she', 'should', 'so', 'some',
'such', 'than', 'that', 'the', 'their', 'theirs', 'them', 'themselves', 'then',
'there', 'these', 'they', 'this', 'those', 'thus', 'to', 'too', 'under',
'underneath', 'up', 'upon', 'us', 'use', 'used', 'uses', 'using', 'was', 'we',
'were'}
```

In this experiment, any words in the stop words list above will be ignored in the NB classifier. The result is below:

```
Total number of documents: 25000
Prediction: True=20658, False=4342
Accuracy: 82.63%
```

2) Binary count: (Reduced repetition count to 1)

In this experiment, any word has multiple repetition in the documents reduced to one. In other words, maximum word count is limited to 1, the word either appears or not in the documents. (1 OR 0) The result of this test is below:

```
Total number of documents: 25000
Prediction: True=20502, False=4498
Accuracy: 82.01%
```

Compared to the previous computed accuracy: 81.11%, both experiments have given a better accuracy result. The STOP WORDS method computes the best accuracy 82.63%. Moreover, I tried to apply both method at the same time, which give the result below:

```
Total number of documents: 25000
Prediction: True=20851, False=4149
Accuracy: 83.4%
```

Surprisingly to me, the accuracy improved to **83.4%**. In conclusion, applying a STOP WORDS list and erasing repeat count effectively increases the accuracy.