COMP 6721 Project2 Report

1 Baseline experiment

1.1 No preprocess of the training set and test set

For class ham:

	Actual: ham	Actual: spam
Test outcome: ham	TP: 394	FP: 44
Test outcome: spam	FN: 6	TN: 356

Accuracy is: 0.9375
Precision is: 0.8995
Recall is: 0.985
F1 score: 0.94

For class spam:

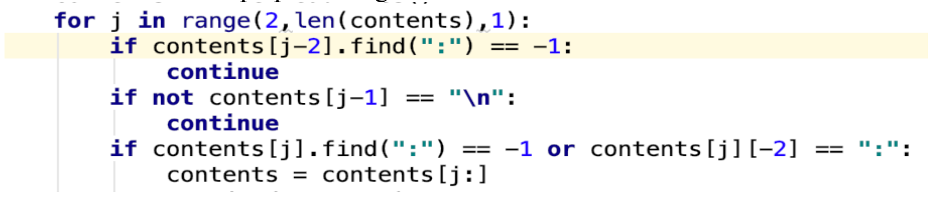
	Actual: spam	Actual: ham
Test outcome: spam	TP: 356	FP: 6
Test outcome: ham	FN: 44	TN: 394

Accuracy is: 0.9375
Precision is: 0.9834
Recall is: 0.89
F1 score is: 0.9344
Because the results of TN + TP are the same for each class. The accuracy is the same for each class. For the precision, FP in ham class is larger than the FP in spam class. So the precision in spam is larger than that in class ham. Similarly, the recall of ham is larger than spam class [1][2].

1.2 With preprocess of the training set

When I saw the training set, I found that each file had a long header in front of the actual message. I want to see if the accuracy is higher if there is no header. So I judge whether it is the actual message part according to the colon and a blank line. (The reason for writing the third if is because most of the actual message in training set files are after the colon and the blank line, but in the file “train-ham-00002.txt”, there is a colon in the first line of actual message [3].

This is the code to do preprocessing.



```
    for j in range(2,len(contents),1):
        if contents[j-2].find(":") == -1:
            continue
        if not contents[j-1] == "\n":
            continue
        if contents[j].find(":") == -1 or contents[j][-2] == ":":
            contents = contents[j:]
```

The result with preprocess:
For class ham:

	Actual: ham	Actual: spam
Test outcome: ham	TP: 388	FP: 32
Test outcome: spam	FN: 12	TN: 368

Accuracy is: 0.945
Precision is: 0.9238
Recall is: 0.97
F1 score is: 0.9463

According to the comparison of the two data, it is not hard to found that the accuracy can be higher after the header is removed. I compared the vocabulary that processed by the same file “test-ham-00277.txt”.

This is the first three lines of vocabulary of preprocess:
{'hit': 1, 'or': 1, 'miss': 1, 'groundhog': 1, 'day': 1, 'is': 1, 'going': 2, 'to': 2, 'be': 1, 'one': 1, 'heck': 1, 'of': 1, 'a': 3, 'show': 1, 'book': 1, 'early': 1, 'avoid': 1, 'the': 1, 'rush': 1, 'hey': 1, 'meybee': 1, 'by': 1, 'then': 1, 'bush': 1, 's': 1, 'grand': 1, 'plan': 1…}

This is the first three lines of vocabulary of no preprocess:
{'from': 8, 'fork': 18, 'admin': 4, 'xent': 18, 'com': 24, 'thu': 4, 'jul': 9, 'return': 1, 'path': 1, 'delivered': 2, 'to': 7, 'yyyy': 1, 'localhost': 7, 'netnoteinc': 2, 'received': 7, 'by': 8, 'phobos': 2, 'labs': 1, 'postfix': 3, 'with': 6, 'esmtp': 3, 'id': 6, 'c': 4, 'd': 3, 'for': 6, 'jm': 3, 'edt': 1, 'imap': 1, 'fetchmail': 1, 'single': 1, 'drop': 1, 'ist': 1…}
We can see that the vocabulary of no preprocessing is very messy, many of them are not even a word. Many of the words are from the email address separated by regular expression. These words are hardly appearing when processing the actual message. And because we want to improve the efficiency of the classifier, we need to use the dictionary as a way to store vocabulary. If there are too many useless words, it will take up a lot of memory.

2 stop-word filtering experiment

For class ham:

	Actual: ham	Actual: spam
Test outcome: ham	TP: 396	FP: 47
Test outcome: spam	FN: 4	TN: 353

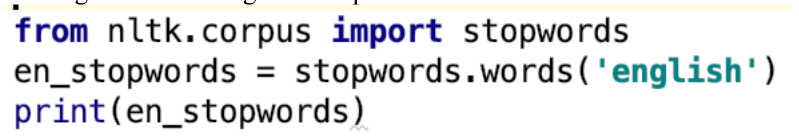
Accuracy is: 0.936
Precision is: 0.894
Recall is: 0.99
F1 score is: 0.9396

For class spam:

	Actual: spam	Actual: ham
Test outcome: spam	TP: 353	FP: 4
Test outcome: ham	FN: 47	TN: 396

Accuracy is: 0.936
Precision is: 0.9888
Recall is: 0.8825
F1 score is: 0.9326

Compared the result with the baseline experiment. I found that the result is almost the same. However, the result after removing the stop word should be better. So I try to find more stop words on the website. I found that the NLTK library and some website contains English stop words.
The following is the code to get the stop words from NLTK



```
from nltk.corpus import stopwords
en_stopwords = stopwords.words('english')
print(en_stopwords)
```

The new result with the new stop words for class ham:

	Actual: ham	Actual: spam
Test outcome: ham	TP: 387	FP: 32
Test outcome: spam	FN: 13	TN: 368

Accuracy is: 0.94375
Precision is: 0.9236
Recall is: 0.9675
F1 score is: 0.9451
The accuracy is better than the previous one. I compared the difference between stop words on Moodle and NLTK. I found that the stop words in NLTK are less than that on Moodle. But I added some stop words to my own stop word file that I found on the website. So if we use better stop words, we can get better results [3].

3 Word-length filtering experiment

For class ham:

	Actual: ham	Actual: spam
Test outcome: ham	TP: 393	FP: 7
Test outcome: spam	FN: 37	TN: 363

Accuracy is: 0.945
Precision is: 0.9825
Recall is: 0.914
F1 score is: 0.9469

For class spam:

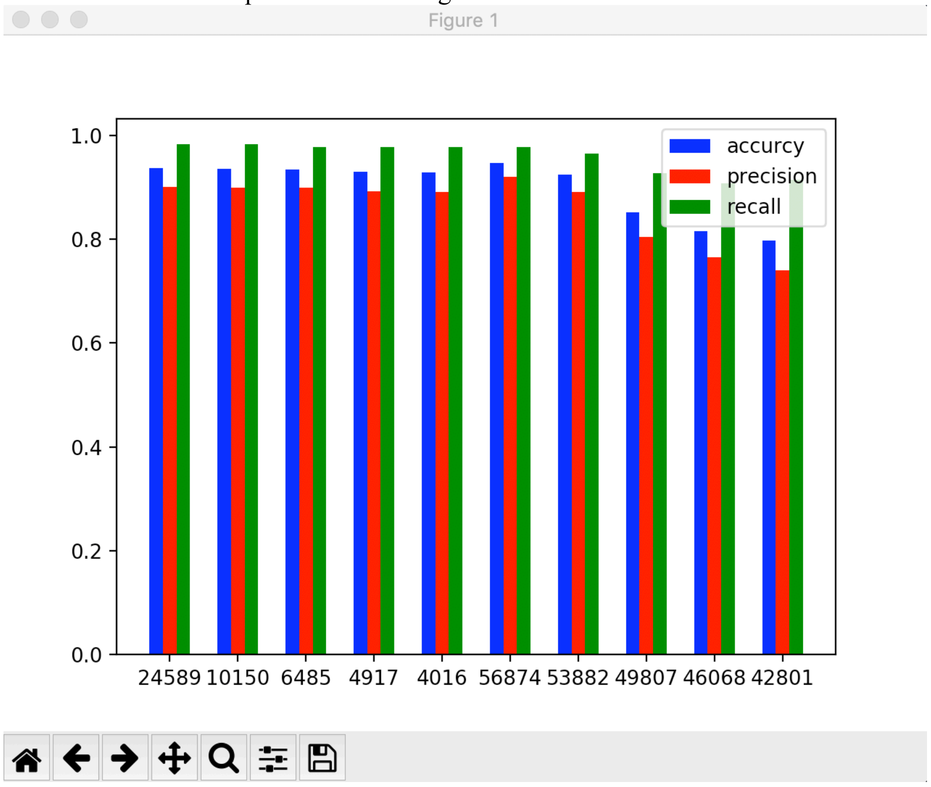
	Actual: spam	Actual: ham
Test outcome: spam	TP: 363	FP: 37
Test outcome: ham	FN: 7	TN: 393

Accuracy is: 0.945
Precision is: 0.9075
Recall is: 0.9811
F1 score is: 0.9431

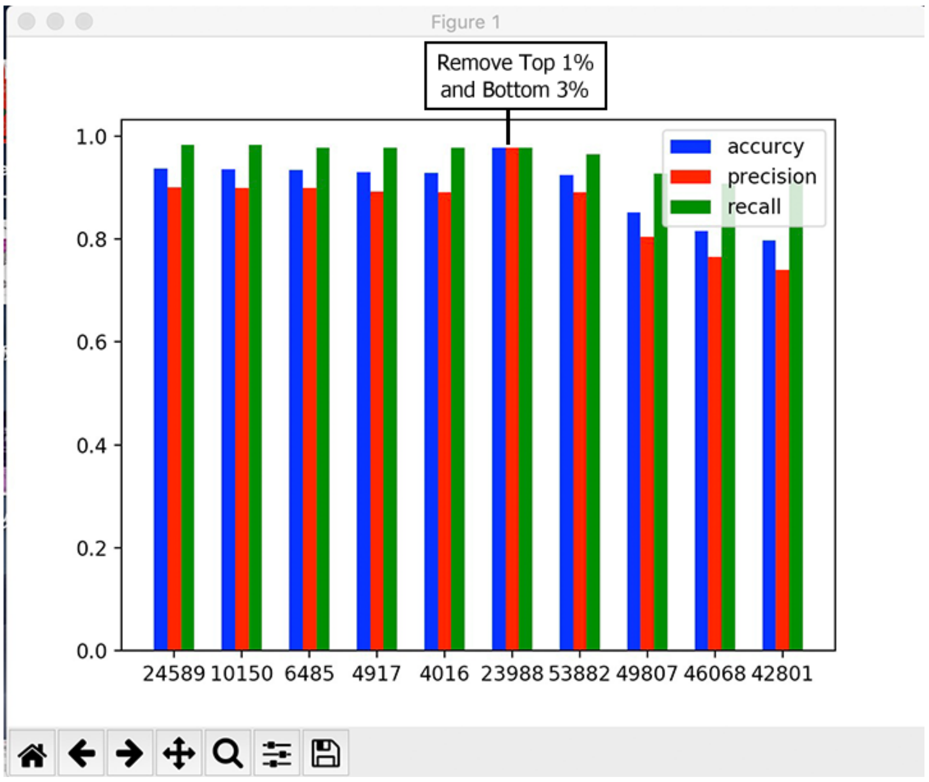
Compared with the result of the baseline experiment, the accuracy of word-length filtering is higher than the baseline experiment. Because most of the words are between 2-9 in length, after removing words less than 2 and longer than 9, we can filter out the useless words in the files when analyzing the test set. It can give us better accuracy and precision. Because the value of FN is large, the result of recall is low.

4 Infrequent word filtering

The bar result of infrequent word filtering:



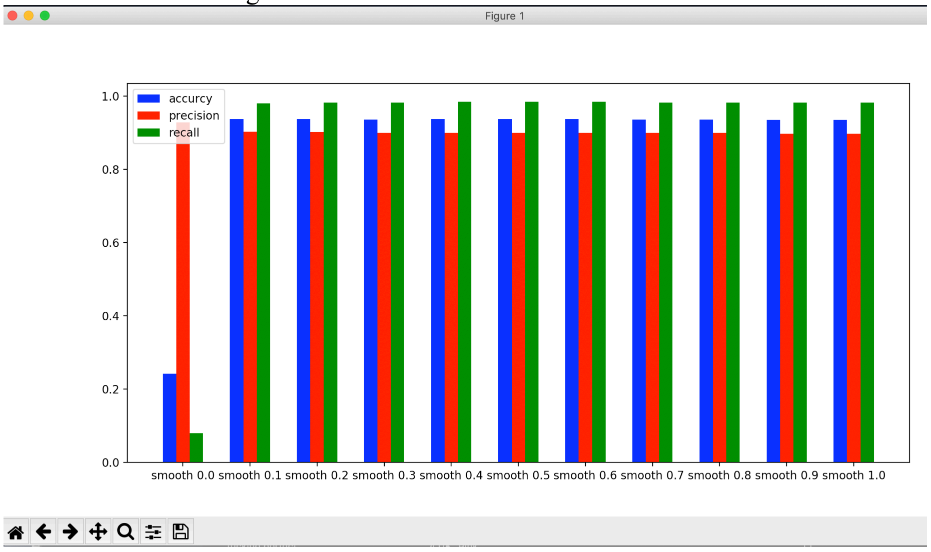
This chart shows the result of infrequent word filtering. The first bar is to remove frequency = 1 word, the second is to remove the words with frequency<=5, the following is frequency<=10, frequency<=15 and frequency<=20, remove the top 5% most frequent words, the 10% most frequent words, 15%, 20%, and 25% most frequent words. The best result is from removing the top 5% most frequent words.
I also have a look at the vocabulary after remove top 5%, I found there are some words like (frn, maeo, asptown, ster, netnaples …) these works don't make any sense. So I try to remove bottom 3% words based on remove top 1% most frequent words. The following chart is the result after removing the top 1% and bottom 3% most frequent words.



We can see that recall is almost the same, but both accuracy and precision have improved a lot. It is because if we remove the top 5% frequent words some useful words are removed also, and these words appear in the test set, which affects the classifier's classification effect on the test set. The bottom 3% of the words may also have some effect on the results. Most of these words are meaningless words. If these words appear in the test set. They will also affect the results, so I also removed the bottom 3% frequency words.

5 Smoothing

The result of smoothing:



The first one is smoothing = 0 because I didn't add smoothing for this step, the probability of most words is log(0), log(0) is negative infinity. So when we calculate the probability of the sentence, we add the probabilities of all words together. If there is one negative infinity, the total probability is negative infinity. If both classes have a word didn't appear in the vocabulary, the total results are both negative infinities so we can't compare the result. In my code, if the result of spam and ham are both negative infinities, the output is a draw. So the result of the first bar is the worst. The rest is almost the same.

6 Compare and discuss the result

The best result is removing the top 1% frequent word, stop word and frequent < 6. The accuracy is 0.98. The worst result is smoothing = 0.

For the baseline experiment:
The accuracy of the baseline experiment is 0.9375, but I found that removing the header is a way to improve accuracy. After removing the header of the test set, the accuracy can be increased to 0.945.

For the stop-word filtering experiment:
The accuracy of this is 0.936. After I added some stop words, the accuracy reached 0.94375.

For the word-length filtering experiment:
The accuracy is 0.945 after removing all words with length <= 2 and all words with length >= 9.

For the infrequent word filtering experiment:
The best result is from removing frequent top 5%. But after removing the top 1% and bottom 3% most frequent words. I got better accuracy and precision.

For smoothing experiment:
The result is almost the same, except the smoothing = 0, because we can't compare two negative infinity, so the result is not good. The rest is almost the same.

The table below is the result I tried to improve the accuracy:

	accuracy	precision
Baseline experiment	0.9375	0.8995
Stop word experiment	0.936	0.894
Word-length experiment	0.945	0.9825
Infrequent filter (frequent = 5%)	0.94265	0.92
Smoothing (smoothing = 0.1)	0.9375	0.903
Infrequent filter (remove frequent top 1% and bottom 3%)	0.9775	0.9775
Infrequent filter (top 1%, bottom 3% and stop word)	0.97875	0.98
Infrequent filter (top 1%, stop word and word frequent < 6)	0.98	0.9848

The best accuracy is removing top 1% frequent word, stop word and frequent < 6. For the baseline experiment, it just adds smoothing=0.5, so there are some words like stop words and length >= 9 that have an effect on the results. So the result is not good. When we remove the word length <= 2 or length >= 9. The accuracy is better. When I do the infrequent filter, I found the best result is remove top 1% frequent word, stop word and word frequent < 6.

7 Describe difficulties

7.1 remove the headers

When I do the task1, I found that in each file it has a header. But the header has no contributed to the classifier result. So I plan to remove the header before creating the vocabulary. I read most of the documents, I found that the actual message of all the files follows a rule. Before the actual message, there is a blank line, before the blank line, there is a colon. I use the following code to do the preprocessing.

```
for j in range(2,len(contents),1):
    if contents[j-2].find(":") == -1:
        continue
    if not contents[j-1] == "\n":
        continue
    if contents[j].find(":") == -1 or contents[j][-2] == ":":
        contents = contents[j:]
```

The first if is used to detect colon. The second is used to detect a blank line. When I saw the "train-ham-00002.txt" file. I add the third if, it is used to detect the first line of the actual message. If there is no colon in this line or the last word is a colon, because each line ends with "\n". So I use contents[j][-2] to detect the last word. If I do the preprocess for each file, the result is better.

7.2 Improve the accuracy

When I finish the rest experiments, I found that accuracy doesn't improve much. So I try to improve accuracy. I print the vocabulary after removing top5% frequent word. I found that some useful words had been removed as well. So I tried to adjust the top 5% to 3% and 1%. I found the accuracy is better. Because the result of removing the stop word had been also improved. I tried to remove the stop word based on removing the top 1% frequent words. The final best result is that accuracy reaches 98%.

7.3 Debugging difficulties

Because the training set and test set are huge. If I found that the program has errors, it is not easy to track the errors, so I built a small training set. After testing all the functions, I started using the training set downloaded on Moodle.

8 Future work

If I continue working on this project, I think I can try the sentence tokenize and word tokenize in NLTK lib instead of the simple regular expression. Because when I look at the vocabulary, I find that there are a lot of words that are not even a word. If I use word tokenize in NLTK, we may be able to avoid this problem.

The second thing I want to try is Naïve Bayes classifier in NLTK, I can add different features in this classifier, like cosine similarity, we can calculate the cosine result of the file with ham class and compare the result with the result of spam class. We can also use a bag of words model, use array instead of collection data structure to save vocabulary frequency information.

The third thing I want to try is different classifiers. Because in this project we are only allowed to use Naïve Bayes classifier. I don't know if we can get a better result with the different classifier. If you had the time I want to try SVM, decision tree and random forest classifier [5][6].

References

1. Towards Data Science. (2019). Beyond Accuracy: Precision and Recall. [online] Available at: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c [Accessed 10 Apr. 2019].
2. Exsilio Blog. (2019). Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures - Exsilio Blog. [online] Available at: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ [Accessed 10 Apr. 2019].
3. Colab.research.google.com. (2019). Import nltk and download the stopwords data. [online] Available at: https://colab.research.google.com/drive/1tNt0Ifom-h4OnFBBZpLndYCEPDU598jE [Accessed 10 Apr. 2019].
4. Scikit-learn.org. (2019). Naive Bayes. [online] Available at: https://scikit-learn.org/stable/modules/naive_bayes.html [Accessed 10 Apr. 2019].
5. Kumar, V. (2019). Email Spam Filtering : A python implementation with scikit-learn. [online] Machine Learning in Action. Available at: https://appliedmachinelearning.blog/2017/01/23/email-spam-filter-python-scikit-learn/ [Accessed 10 Apr. 2019].
6. Towards Data Science. (2019). Applied Text-Classification on Email Spam Filtering. [online] Available at: https://towardsdatascience.com/applied-text-classification-on-email-spam-filtering-part-1-1861e1a83246 [Accessed 10 Apr. 2019].
7. Datascienceassn.org. (2019). Natural Language Processing With Python. [online] Available at: http://www.datascienceassn.org/sites/default/files/Natural%20Language%20Processing%20with%20Python.pdf [Accessed 10 Apr. 2019].