COMP 6721 Project 2 Report

Zhiyi Ding[1] and Jun Shao[2]

[1] ID-26963013 dingzhiyi22@gmail.com

[2] ID-40077592 shao12jun@gmail.com

1 Introduction

The main task of this project is to build a Naïve Bayes classifier for email classification, classifying an email as 'ham' or 'spam'. In the family of machine learning, Naïve Bayes classifier is a kind of probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features [1]. In the case of our project, features are words appeared in the emails.

There are three general tasks to complete in this project. The first one is to build a baseline Naïve Bayes model based on the "Train-set" email corpus. Then, the performance of the model will be tested in a test set based on accuracy, precision, recall and F1-measure. After that, some interesting experiments will be conducted to improve the performance of the model, including stop-words filtering, word length filtering, infrequent and most frequent words filtering as well as smoothing. The effectiveness of these adjustments will be discussed in the following Experiment and Analysis section.

2 Method

2.1 Bayes Theorem [2]

The Naïve Bayes Classifier is built up based on applying Bayes Theorem with strong independence assumption between the features. This theorem can be used to compute the conditional probability, which is the possibility of an event by giving a known true condition. Bayes Theorem is illustrated in equation (1), where H stands for Hypothesis(event) and E stands for Evidence (true condition):

$$\mathrm{P}(H|\mathrm{E}) = \frac{\mathrm{P}(E|\mathrm{H})*\mathrm{P}(\mathrm{H})}{\mathrm{P}(\mathrm{E})} \quad (1)$$

The theorem may arise the idea that we can compute the desired probability of $\mathrm{P}(H|\mathrm{E})$ by the equivalent of right-hand side calculation $\frac{\mathrm{P}(E|\mathrm{H})*\mathrm{P}(\mathrm{H})}{\mathrm{P}(\mathrm{E})}$, which is much easier for collecting the data and calculating the result.

2.2 Naive Bayes classifier [1]

By applying Bayes theorem, A Naïve Bayes classifier is established to predict the type of an email. In a real application, we usually drop off the denominator P(E) in the formula (1) as for each candidate H_i the P(E) used is the same, it has no impact on the final classification. In our project, we only need to compute the prior probability $\mathrm{P}(\mathrm{C_i})$ and the condition probability $\mathrm{P}(w_i|c_i) = \mathrm{P}\left(\frac{total\ number\ of\ w_i\ in\ c_i}{total\ words\ in\ c_i}\right)$. We can also smooth the model by adding a small value, so the formula of conditional probability is converted to:

$$\mathrm{P}(w_i|c_i) = \mathrm{P}\left(\frac{total\ number\ of\ w_i\ in\ c_i + smoothing\ value}{total\ words\ in\ c_i + smoothing\ value * size\ of\ vocabulary}\right) \quad (2)$$

The impact of different smoothing value will be studied in the Experiment section.

After setting up this model, we use the function $\mathrm{P}(c_i) * \prod_{i=1}^{n} \mathrm{P}(w_i|c_i)$ to evaluate each test email and calculate the score for each category. Another detail needed to mention here is that a logarithm calculation is used to prevent the possible numerical underflow instead of simply multiplying all the words probabilities together.

2.3 Evaluation metrics

In order to evaluate the performance of our model, we take accuracy, precision, recall and F1-measure as evaluation metrics.

Accuracy is the percentage of instances in the test set that the model has correctly classified. Recall is the proportion of the instances in a special class *C* are labelled correctly. Precision is the proportion of instances predicted to be class *C* are actually correct. F-measure is a weighted combination of precision and recall [4].

3 Experiment [3]

3.1 Baseline experiment

In this project, we use the email corpus available on Moodle [5] to fit and test our classifier. The dataset contains emails that are already classified into 2 categories: ham and spam. In training dataset, there are 1000 emails are classified as ham, and 997 emails are classified as spam. In test dataset, both the number of emails for ham and spam are 400.

At first, we use the training data to build our classifying model. To preprocess the text, we open the text with latin-1 encoding, fold all strings to lowercase, then tokenize the words with the method re.split ('[^a-zA-Z]',aString).

After preprocessing of training data, we use them to build the model. For each word w_i in the training set, we save its frequency and calculate its conditional probability with add-0.5 smoothing for each class. See equation 3 for add-0.5 smoothing.

$$P(w_i|C_j) = \frac{(\text{frequency of wi in } C_j + 0.5)}{(\text{total number of words in} C_j + 0.5 * \text{size of vocabulary})} \quad (3)$$

Finally, we use the test data to test our Naive Bayes classifier. Before testing, we preprocess the testing text with the same method described above. To avoid arithmetic underflow, we calculate the score of each class in $\log_{10}$ space.

3.2 Stop-word Filtering

Stop words are commonly used words (such as “the”, “a”, “an”, “in”) but are useless for the semantic analysis. In preprocessing of training data, a major task is to filter useless data, thereby saving the space and time for the subsequent processing [6].

To study the effect of stop-word filtering, we redo the task in experiment 1 but remove the stop words from the vocabulary. Generate a new model and test it with the same test set as the baseline experiment.

3.3 Word Length Filtering

In the preprocessing of a corpus, it is a good practice to filter short strings that mainly come from parts of abbreviations (such as “t” from “can’t”, “d” from “I’d”.), as well as too long strings that do not make sense.

In this experiment, we redo the task in experiment 1but this time remove all words with length <=2 and all words with length >=9. Generate a new model and test it with the same test dataset.

3.4 Infrequent and highly frequent Word Filtering

According to Zipf-Mandelbrot Law [7], the distribution of words ranked by their frequency in a random text corpus is approximated by a power-law distribution, which means most word types are very rare, with a small number of word types make up the majority of word tokens that you see in any corpus. Thus, it is necessary to study the influence of removing infrequent and highly frequent words from the corpus.

In this experiment, we use the baseline experiment, and gradually remove from the vocabulary words with frequency= 1, frequency <=5, frequency <=10, frequency <=15 and frequency <=20. Then gradually remove the top 5% most frequent words, 10%, 15%, 20% and 25% most frequent words. At the first time, we calculated the number of top 5% most frequent words based on the original vocabulary length of the baseline model but found there were no words left after removing words with frequency <=20 and top 5% most frequent words. Therefore, we calculate the number of top n% most frequent words based on the length of vocabulary after removing words with frequency <=20.

Plot the performance of the classifier against the number of words left in your vocabulary.

3.5 Smoothing

Smoothing is a common practice in natural language processing, as it prevents the existence of probability 0. To study the effect of smoothing, we use the baseline experiment and change the smoothing value δ gradually from 0 to 1 in steps of 0.1. Then plot the performance of the classifier against the smoothing value.

4 Results and analysis

4.1 Experiment 1

Table 1. Performance of Naive Bayes Classifier

Predicted result	The True label		Precision	Recall	F1_measure	Accuracy
	ham	spam				
Ham	394	64	0.860	0.985	0.918	0.913
Spam	6	336	0.982	0.840	0.906	

As shown in Table 1, the overall performance of the baseline model is good. The accuracy of the model for the whole test set is 0.913. As a ham/spam classifier, we take recall of ham and precision of spam more seriously, because we would rather receive spam than miss important ham. The recall for ham and the precision for spam is 0.985 and 0.982, respectively. The F1-measure for both ham and spam are over 0.9 (0.918 and 0.906 respectively).

The result of this experiment can be used as a baseline, to study the effect of some word filters in the following experiments.

4.2 Stop-word Filtering

Table 2. Performance of Naive Bayes Classifier with stop-word filtering

Predicted result	The True label		Precision	Recall	F1_measure	Accuracy
	ham	spam				
Ham	394	63	0.862	0.985	0.919	0.913
Spam	6	337	0.983	0.843	0.907	

The existence of “senseless” stop words in the vocabulary may impact the performance of the model as intuitively, the change of the number of these frequent stop words in an email/text would make a bigger influence than that of a word with low conditional probability. However, the result of the stop-words filtering experiment (see Table 2.) does not show significant improvement in the performance of the model compared with the baseline model. The overall accuracy is 0.913, with a ham recall 0.985 and spam precision 0.983.

The F1-measure value for ham and spam are 0.919 and 0.907, which is in the same level with that of baseline model.

One possible reason is that there may be little difference between the conditional probability of stop-words in class ham and spam. When applying our model to a text, though we calculate the score for two classes, the number and types of stop-words are the same for the text. Another possible reason may be related to the limited number of stop words we used (about 160). As shown in subsequent experiment, the effect of frequent words filtering usually depends on the number of words filtered and the length of the vocabulary.

Nevertheless, filtering out stop-words is always a good practice, as it cuts down the length of the vocabulary, thereby reducing the cost of computation.

4.3 Word Length Filtering

Table 3. Performance of Naive Bayes Classifier with word length filtering

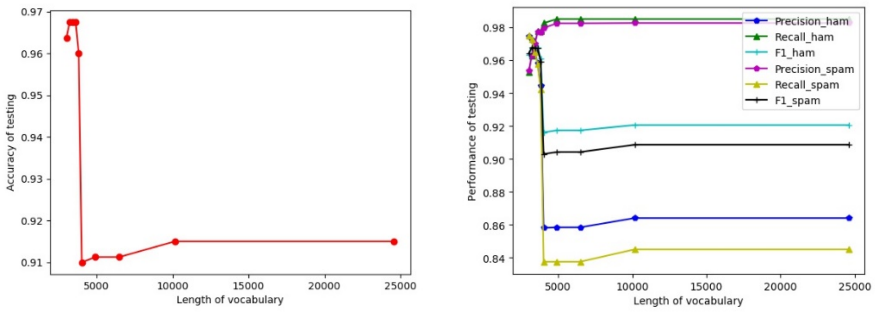
Predicted result	The True label		Precision	Recall	F1_measure	Accuracy
	ham	spam				
Ham	393	57	0.873	0.983	0.925	0.92
Spam	7	343	0.980	0.858	0.915	

As shown in Table 3., when filtering words with length less than 2 and greater than 9, the overall accuracy improved by 0.7% compared with the baseline model. Both the F1-measure values for ham and spam rise, by 0.7% and 0.9, respectively.

It is obvious that tokens less than 2 are more likely to be stop-words such as ‘is’, ‘as’, ’I’. They can also be parts from abbreviations like “m” from “I’m” or “t” from “don’t”. Considered the average length of English words is 6.19 [8], tokens with the length greater than 9 are more likely to be senseless strings such as “aaaaaaaaaaaaaa” or “aaaaaaabagadbbefehmuiravicjaqte”, which are picked out from our baseline vocabulary.

Although there is a rise in performance, it is not significant. We still need a more typical words filter for the improvements.

4.4 Infrequent and top frequent Word Filtering



(a) (b)

Fig. 1. Accuracy of Naive Bayes Classifier for testing dataset based on length of vocabulary

The original vocabulary length of our baseline model is 59953. When removing infrequent words gradually from the vocabulary for frequency =1 to frequency <=20, the length of vocabulary decreases dramatically from 24588 to 4016 (see Fig.1). It is in accordance with the Zipf-Mandelbrot Law [7], which implicates

most word types in the vocabulary are very rare. Although infrequent words have low frequency, they possess certain information for the classification of text. Thus, gradually removing of infrequent words results in the continuous loss of testing accuracy, as shown in Fig.1(a).

In almost all corpus, there are many words appear with high frequency but possess less information, we call them stop-words, such as “the”, ”is” and ”that”. The existence of stop-words will cost the computational time of our model, impact the conditional probability of meaningful words. Thus, it is a good practice to filter frequent stop-words during the preprocessing of the corpus. As shown in the figure above, when removing the top 5% most frequent words from the vocabulary after removing words with frequency <= 20, the testing accuracy for our model dramatically increased by 5% from 91% to 96 %. When continually removing the top 10%, 15% and 20% most frequent words, the testing accuracy peak at 96.8% and keep stable. However, when more words are removed, just as removing the 25% most frequent words, the testing accuracy for our model began to drop (see Fig.1(a)).

The same trend as that of accuracy was spotted for the precision of ham, recall of spam and F1-measure for both ham and spam, as shown in Fig.1(b).

The result of this experiment shows that filtering enough number of top frequent words from a suitable length of vocabulary can significantly improve the performance of the model. However, as the number of top frequent words filtered continually increased, more useful information will be lost, resulted in a drop in the accuracy of the model.

4.5 Smoothing

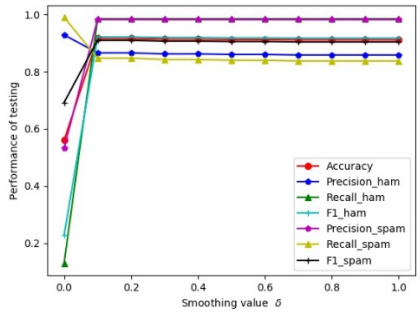


Fig. 2. Accuracy of Naive Bayes Classifier for testing dataset based on different smoothing values δ

As shown in the Fig.2., without smoothing ($\delta = 0$) the testing accuracy for the model is only 0.55, which is close to a random choice. As without smoothing, a lot of words in the vocabulary has a conditional probability of 0. When calculating the product of probability, it is easy to get 0 for both classes, or negative infinity when summing $\log_{10}$ of the product of probability. With the same score, the model can only choose randomly. Thus, the final accuracy is close to 0.5. Concretely, without smoothing our model labels most of the emails as spam, as shown in Fig.2. the Recall_spam is close to 1.0 while Recall_ham is lower than 0.1.

On the other hand, after adding smoothing, the performance of the model soars up to over 0.92 and keep stable for different smoothing values changing from 0.1 to 1.0.

5 Discussion

In this project, we build a Naive Bayes Classifier for email classification. After that, a series of experiments are conducted to improve the performance of the model, including stop-words filtering, short and long words filtering, infrequent and top frequent words filtering.

The results show that stop-words filtering, short and long words filtering can improve the performance of our model compared with the baseline model, but the individual effect of these methods is not obvious. When gradually remove infrequent words from the baseline vocabulary, the accuracy of the model declines slightly due to the loss of useful information. However, if we change to continually remove top frequent words based on vocabulary that has already removed words with a frequency less than 20, a dramatic rise of accuracy will be spotted. This result justifies the impact of highly frequent words on the product of conditional probabilities for different categories as well as the performance of our model.

Besides, we also study the impact of smoothing with the smoothing value δ varying from 0 to 1 with a step of 0.1. The result shows that without smoothing ($\delta = 0$), the model labels most of the emails as spam. After adding a smoothing value ($\delta \neq 0$), the accuracy of the model soars to over 0.92. The role of adding- δ smoothing is to assign a certain probability from frequent words to rare words. It is simple to deploy this method, but how to choose the value of δ is difficult as it depends on the size of vocabulary and types of corpus [9]. In this project, the variation of δ between 0.1 and 1 does not make a significant difference in the accuracy of the model.

The main challenge of this project is to remove the infrequent words and then remove top frequent words gradually. The base for the top frequent words removing experiment is not clear. At first, we calculated the number of the top frequent words based on the original vocabulary from baseline experiment but found no words left in the vocabulary. Later, we calculated the number of top frequent words based on the infrequent words removed vocabulary and found a good trend between the performance of the model and the length of vocabulary.

The language model we used in this project is a unigram model or bag-of-word model, as we only count the frequency of words without considering the word order. Next step, we can build a bigram or trigram model which will take the order of words into account.

Besides, by looking through the vocabulary of our word length filtering model, we find a lot of tokens that do not represent an available English word. If we can filter out these tokens based on an available English dictionary, our model may be more efficient.

What's more, we find lots of tokens in the vocabulary belong to the same stem such as the token "earn", "earned", "earni", "earning" and "earns", which represent the same meaning. We can represent all these words with a single lemma, thereby reducing the length of vocabulary further. We can deploy the methods PorterStemmer() or WordNetLemmatizer() from NLTK[10] to complete this task.

All methods we have discussed above are machine learning methods and need to tune the features (words in the vocabulary) manually. In the future, we can also use word2vector to represent our words and apply the neural network to this classifying task.

6 Team members and Contributions

Contributions:

Zhiyi Ding:

- Main coder.
- Report the section of introduction, method, result and analysis of experiment 1 ~ 3.

Jun Shao:

- Main coder.

- Report the section of experiment, results and analysis of experiment 4 ~ 5, discussion.

Note: Though Jun's code was submitted, both team members contribute to the whole programming. We programed individually, then compared the result with each other.

References

1. Naive Bayes classifier, https://en.wikipedia.org/wiki/Naive_Bayes_classifier
2. Bayes' theorem, https://en.wikipedia.org/wiki/Bayes%27_theorem
3. COMP 472/6721 Project 2 in Winter2019, https://moodle.concordia.ca/moodle/pluginfile.php/3445782/mod_label/intro/COMP_472___2019_Winter___Proj ect_2.pdf
4. Artificial Intelligence: Machine Learning -1, https://moodle.concordia.ca/moodle/pluginfile.php/3467565/mod_label/intro/472-7-ML-1-Winter2019.pdf
5. COMP 472 NN, COMP 6721 NN 2184 https://moodle.concordia.ca/moodle/course/view.php?id=112054
6. Removing stop words with NLTK in Python, https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
7. Zipf–Mandelbrot law, https://en.wikipedia.org/wiki/Zipf%E2%80%93Mandelbrot_law
8. Length of words – average number of characters in a word, https://diuna.biz/length-of-words-average-number-of-characters-in-a-word/
9. Language Modelling: Smoothing and Model Complexity, https://www.cs.mcgill.ca/~jcheung/teaching/fall-2016/comp599/lectures/lecture4.pdf
10. Natural Language Toolkit, https://www.nltk.org/