|  |  |
| --- | --- |
|  |  |

Hanoi University of Science and Technology  
School of Information and Communication Technology

EMAIL SPAM DETECTTION

|  |  |  |
| --- | --- | --- |
| **Supervisor:** | Nguyen Nhat Quang |  |
| **Academic year:** | 20212 |  |
| **Students:** | Group 8 |  |
|  | Nguyen Thanh Long | 20180128 |
|  | Ngo Viet Tung | 20184326 |
|  |  |  |
| **Date:** | June, 2022 |  |

Table of Contents

1 Introduction 2

2 Data preparation 2

2.1 Analysis dataset 4

2.2 Implementing Bag of Words 5

2.3 Training and testing sets 7

3 Model selection and preparation 7

3.1 Naïve Bayes Classifier 7

3.2 Multinomial Naive Bayes 10

4 Training and Evaluation 11

4.1 Accuracy 12

4.2 Precision 12

4.3 Recall(sensitivity) 12

4.4 F1 – score 12

4.5 Support 13

4.6 Confusion matrix 13

4.7 How to training and evaluation in program 13

5 Difficulties 15

6 Conclusion 16

7 Contribution 16

8 References 16

# 1 Introduction

Text mining (or deriving information from text) is a wide field which has gained popularity with the huge text data being generated. Automation of several applications like topic classification, text summarization, machine translation, ... has been done using machine learning models.

Even if people were not aware of it, classification in machine learning has undoubtedly already been utilized in a variety of areas of human existence. Spam will be immediately detected if someone uses a modern email system. To put it another way, the system will review all incoming emails and classify them as spam or not spam. To improve the effectiveness of its spam detection, the end-user will frequently be allowed to manually designate emails as spam or not. This method of machine learning uses examples of two different message types - spam and ham, which is the common word for "non-spam emails" to automatically categorize incoming emails.

Nowadays, everyone has a smartphone, and many individuals use one or two email accounts. Therefore, you must be familiar with the numerous emails promising huge sums of money, incredible lottery wins, wonderful gifts, and life's mysteries. We get dozens of spam messages every day unless you use well-trained filters. They could be dangerous, just annoying or space-consuming, but they could also carry viruses or fishing attempts. In any case, it is not the content we want to deal with. Thus, there is a constant need for effective spam filters.

Spam filtering is a beginner’s example of document classification task which involves classifying an email as spam or non-spam (normal) mail. Spam box in your Gmail account is the best example of this. So, we have started building a spam filter on a publicly available mail corpus.

# 2 Data preparation

The Email spam collection is a set of tagged messages that have been collected for Spam filter research. It contains one set of messages in English of 5728 messages, tagged according being ham (legitimate) or spam.

The columns in the dataset are currently named as you can see, there are 2 columns.

* The first column is the text content of the SMS message that is being classified.
* The second column takes two values, “0” which signifies that the message is not spam, and “1” which signifies that the message is spam.

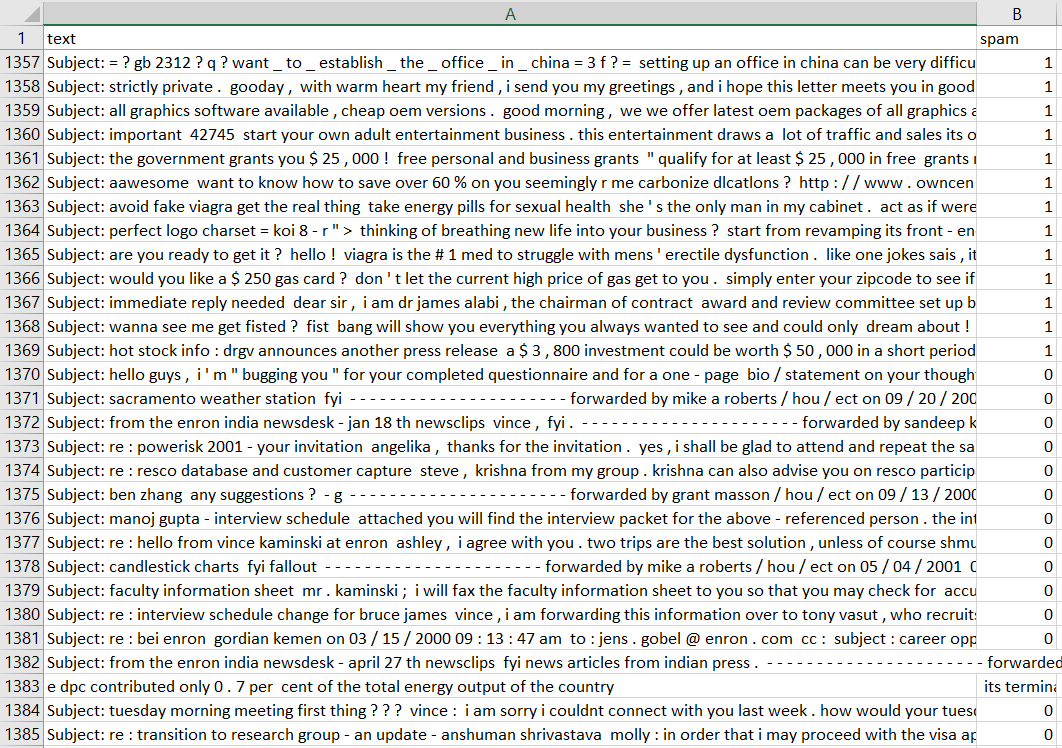


Figure 1: Data preparation

## 2.1 Analysis dataset

The mentioned dataset contains 5728 records of different messages together with 1368 spam messages.

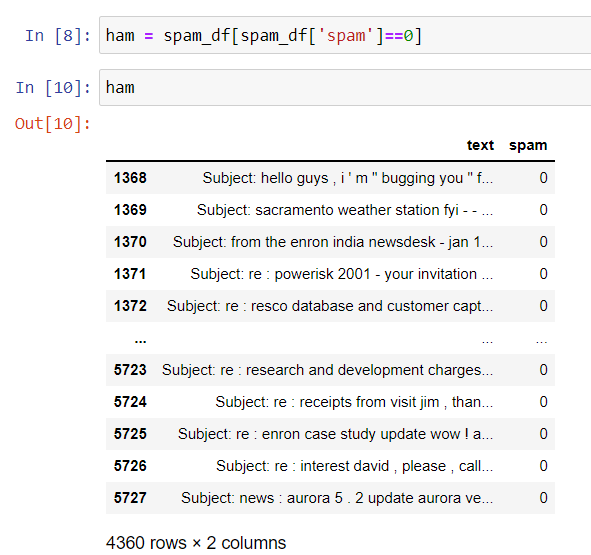
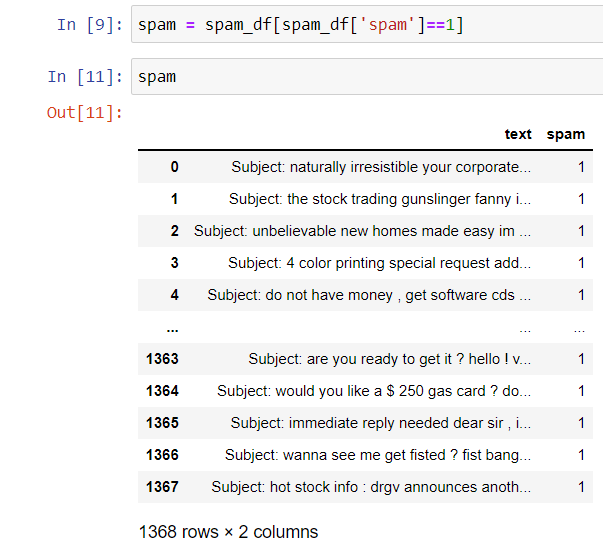
 

Figure 2: Group ham/spam email

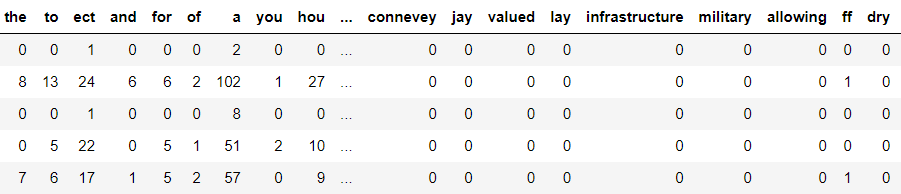


Figure 3: Countplot dataset

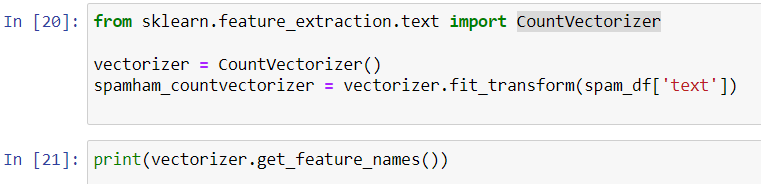
## 2.2 Implementing Bag of Words

We have a massive collection of text data in our data set (5,728 rows of data). Most Machine learning algorithms rely on numerical data to be fed into them as input, and email messages are often text-heavy.

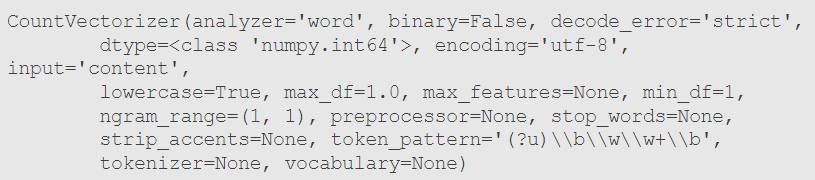
The basic idea of Bag of Words(BoW) is to take a piece of text and count the frequency of the words in that text. It is important to note that the BoW concept treats each word individually and the order in which the words occur does not matter.



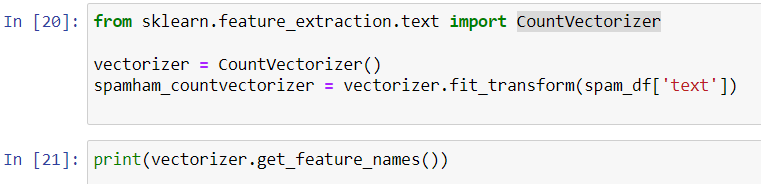
To implement the BoW, we would like to use sklearns *count vectorizer* method to complete this procedure quickly and cleanly.

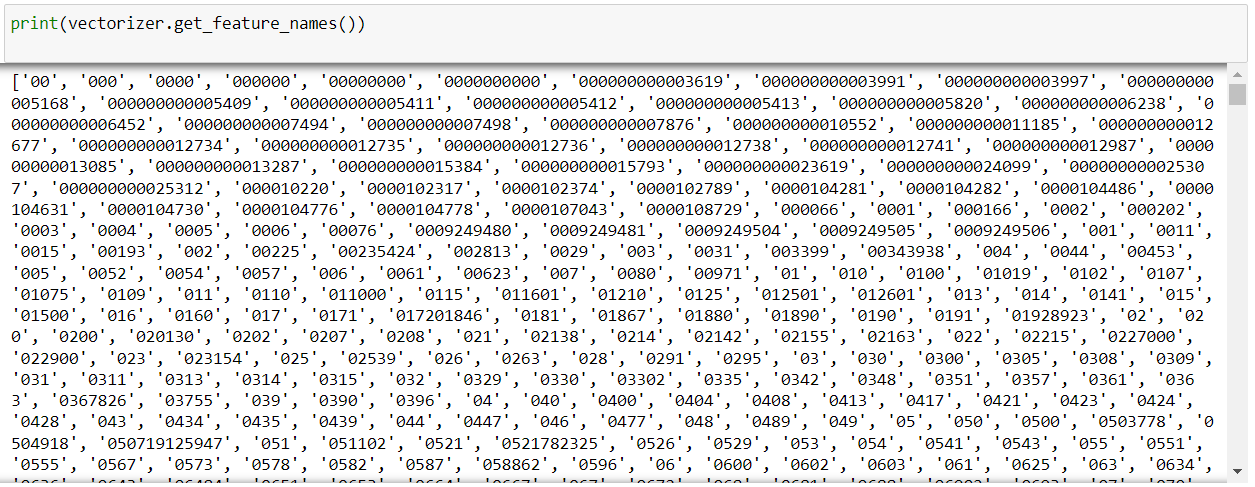


*CountVectorizer* will convert all the text into **lowercase**, will remove all the **punctuations** and all the **stopwords**.

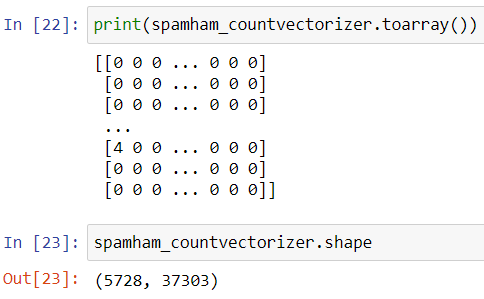


Convert our data into the desired matrix format.





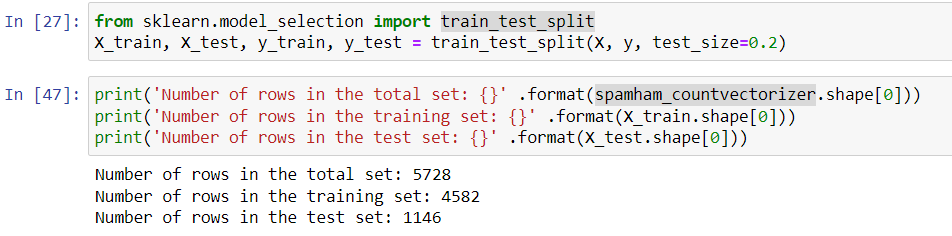
Now we need to have a clean representation of the documents in terms of the frequency distribution of the words in them.



## 2.3 Training and testing sets

Back to our dataset and proceeding with our analysis, we need to split it into a training and testing set so we can test our model later.

We will be using **train\_test\_split** from **sklearn.model\_selection** module.



# 3 Model selection and preparation

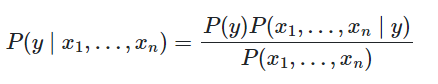
We have chosen so many algorithms to do classification tasks; the following are some state-of-the-art algorithms that we think we should give a try

## 3.1 Naïve Bayes Classifier

**Definition:**

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes’ theorem with the “naïve” assumption of conditional independence between every pair of features given the value of the class variable.

Bayes’ theorem states the following relationship, given class variable ***y*** and dependent feature vector ***x***1 through ***x***n.



𝑃(*y*): Prior probability of hypothesis (e.g., classification)

𝑃(x1,… xn): Prior probability that the data x1,… xn is observed

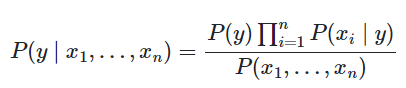
𝑃(x1,… xn | *y*): Probability of observing the data x1,… xn given hypothesis y

𝑃(*y*| x1,… xn): Probability of hypothesis y given the observed data x1,… xn

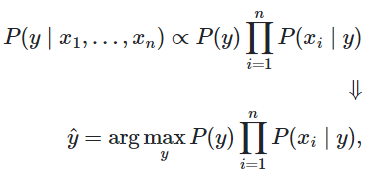
Using the naive conditional independence assumption that



for all ***i***, this relationship is simplified to



Since 𝑃(x1,… xn) is constant given the input, we can use the following classification rule:



and we can use Maximum A Posteriori (MAP) estimation to estimate 𝑃(*y*) and 𝑃(xi | *y*); the former is then the relative frequency of class *y* in the training set.

The calculation of 𝑃(xi | *y*) depends on the type of input data. There are three commonly used types: Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive.

**Advantages:**

* Despite their over-simplified assumptions, Naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering
* Naive Bayes requires only a small number of training data to estimate the parameters necessary for classification.[[1]](#footnote-1)

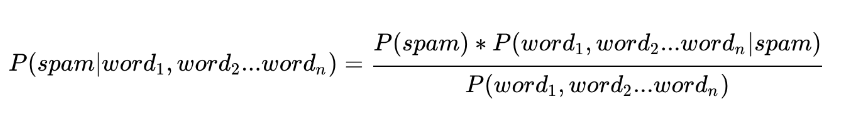
**Disadvantages:**

* If your test data set has a categorical variable of a category that was not present in the training data set, the Naive Bayes model will assign it zero probability and will not be able to make any predictions in this regard. ...
* This algorithm is also notorious as a lousy estimator.[[2]](#footnote-2)

**Apply in spam filter:**

In the context of the spam filter, we suppose that every word in the message is independent of all other words and we count them with the ignorance of the context.

We take the Bayes formula of the conditional probability and apply it to our task:

[[3]](#footnote-3)

𝑃(spam): Prior probability of spam emails over total number of emails.

𝑃(𝑤𝑜𝑟𝑑1,… wordn): Prior probability of words over total number of words.

𝑃(𝑤𝑜𝑟𝑑1,… wordn |𝑠𝑝𝑎𝑚): Probability of observing words in spam emails over total number of words in spam emails.

𝑃(𝑠𝑝𝑎𝑚|𝑤𝑜𝑟𝑑1,… wordn): Probability that an email is spam if its content contains words.

Calculating elements:

𝑃(𝑤𝑜𝑟𝑑1,… wordn |𝑠𝑝𝑎𝑚) =

(Multiply all the probabilities of each word in spam email)

𝑃(𝑤𝑜𝑟𝑑1,… wordn)

= 𝑃(𝑠𝑝𝑎𝑚) \* 𝑃(𝑤𝑜𝑟𝑑1,… wordn |𝑠𝑝𝑎𝑚) + 𝑃(ℎ𝑎𝑚) \* 𝑃(𝑤𝑜𝑟𝑑1,… wordn |ℎ𝑎𝑚)

= 𝑃(𝑠𝑝𝑎𝑚) \* + 𝑃(ham) \*

Taken everything together, the probability that the spam email is:

𝑃(𝑠𝑝𝑎𝑚 | 𝑤𝑜𝑟𝑑1,… wordn)

=

## 3.2 Multinomial Naive Bayes

**Definition:**

MultinomialNB is used in the text classification that *features vectors* calculated in Bags of Words(BoW). At this time, each document is represented by a vector of the length *d*, which is the number of features (in text classification, the size of the vocabulary). The value of the *ith* component in each vector is the frequency of it in that text.

**How does it work?**

In Bayes theorem, Multinomial Naive Bayes apply MLE, MAP to classify by: *maximize P(x**1,… xn | y)*

Then *P(xi | y)* is proportional to the frequency of the *ith* word (or *ith* feature for the general case) appearing in documents of class y. This value can be calculated by:

*P(xi | y) =*  [eq.1]

Where:

* *N*yi is the total number of times from the *ith* appearance in the text of the *y* class *y* in the training set *T*: *N*yi =
* *N*y is the total number of words (including repetition) that appears in the *y* class: *N*y =

However, if there is a word that has not appeared in class *y*, the calculation [eq.1] will be zero. This will lead to inaccurate results.

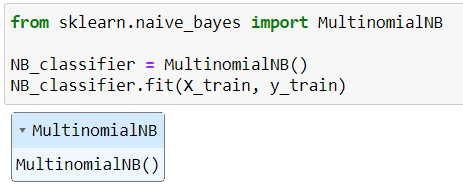
To solve this, a technique called Laplace Smoothing is applied:

𝜃yi *=* *P(xi | y) =*

With α is a positive number, usually equal to 1, to avoid the result equal to 0. The denominator plus α\**d* to ensure that = 1.

Thus, each class C will be described by the positive numbers with the sum of 1 𝜃y = {𝜃y1 *,….* 𝜃yn}

**How to use Multinomial Naive Bayes in the program?**

****

Specifically, we will be using the multinomial Naive Bayes implementation, from sklearns *sklearn.naive\_bayes* method, to make predictions on our dataset.

This classifier is suitable for classification with discrete features (such as in our case, word counts for text classification). It takes in integer word counts as its input.

On the other hand, Gaussian Naive Bayes is better suited for continuous data as it assumes that the input data has a Gaussian(normal) distribution.

# 4 Training and Evaluation

There are four ways to check if a prediction is positive or negative:

|  |  |  |
| --- | --- | --- |
|  | Actual Positive | Actual Negative |
| Predicted Positive | True Positive / TP | True Negative / TN |
| Predicted Negative | False Positive / FP | False Negative / FN |

We have made predictions on our test set; our next goal is to evaluate how well our model is doing. There are various mechanisms for doing:

## 4.1 **Accuracy**

It measures how often the classifier makes the correct prediction. It’s the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

**Accuracy =**

## 4.2 **Precision**

It tells us what proportion of messages we classified as spam, were spam. It is a ratio of true positives (words classified as spam, and which are spam) to all positives (all words classified as spam, irrespective of whether that was the correct classification), in other words it is the ratio of

**Precision =**

## 4.3 **Recall(sensitivity)**

It tells us what proportion of messages that spam classified by us as spam. It is a ratio of true positives (words classified as spam, and which are spam) to all the words that were spam, in other words, it is the ratio of recall:

**Recall =**

## 4.4 F1 – score

The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.

F1–score **=**

## 4.5 **Support**

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support does not change between models but instead diagnoses the evaluation process.

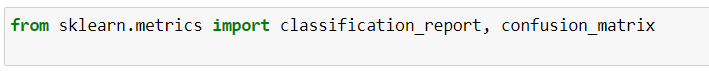
## 4.6 Confusion matrix

Compute confusion matrix to evaluate the accuracy of a classification. A confusion matrix *C is* such that *C*i,j is equal to the number of observations known to be in group *i* and predicted to be in group *j*.

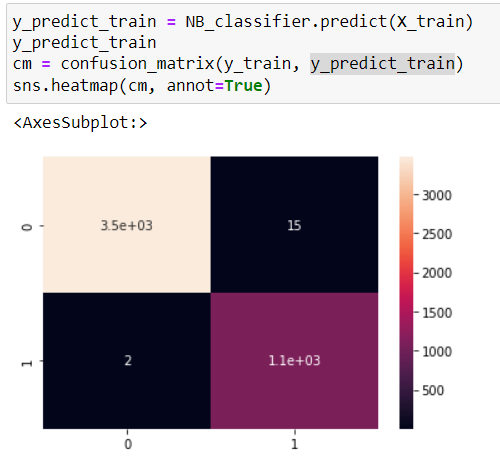
|  |  |
| --- | --- |
| Thus, in binary classification, the count of true negatives is *C*0,0, false negatives are *C*1,0, true positives are *C*1,1 and false positives is *C*0,1. |  |

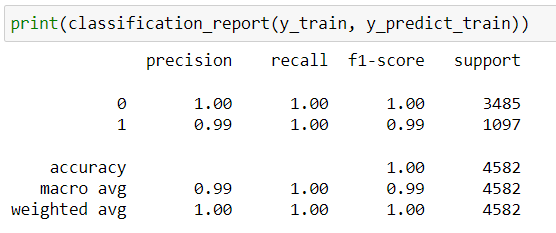
## 4.7 How to start training and evaluation in a program?

We have used *classification\_report*, *confusion\_matrix*, *accuracy\_score* functions in sklearn library to evaluate the performance of the system after the training and testing phases.

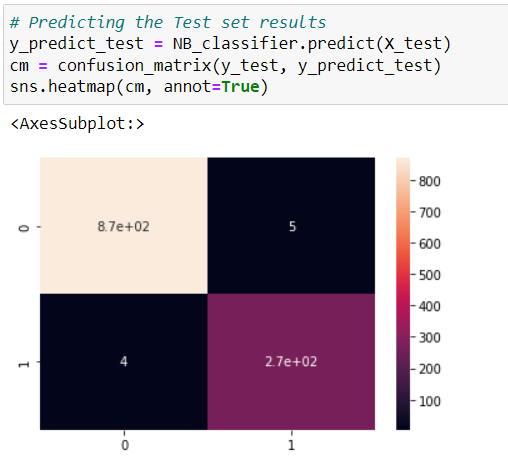


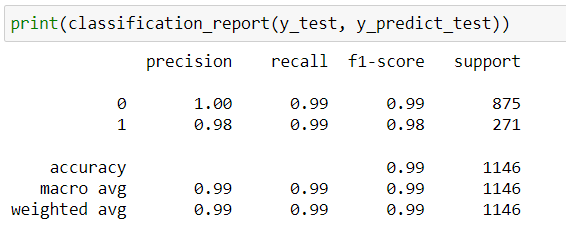
* **After training phase:**





* **After testing phase:**





# 5 Difficulties

* Choosing dataset  We have browsed through and compared many resources and then decided to choose the most appropriate dataset.
* Choosing parameters during the training process, ratio to split train, and test datasets. 🡪 We have read other resources[[4]](#footnote-4) [[5]](#footnote-5) for suitability to our dataset.

# 6 Conclusion

* From the experimental results above, we can see Naive Bayes handles an extremely considerable number of features well and its model training and prediction times are fast for data it can handle.
* In our case, each word is treated as a feature and there are thousands of different words. Also, it performs well even with the presence of irrelevant features and is unaffected by them.
* Additionally, we can observe that MultinomialNB's classification report produced results with good accuracy, recall, and precision.
* Furthermore, we need to choose some larger and more multiform of dataset, as well as implement classification algorithms in various places (Speech Recognition, Identifications of Cancer tumor cells, Drugs Classification, …) to get more perspectives on classification application.

# 7 Contribution

The table below shows the members’ contributions to our project.

|  |  |
| --- | --- |
| Data preparation | Nguyen Thanh Long (50%)  Ngo Viet Tung (50%) |
| Model preparation, Training and Evaluation | Nguyen Thanh Long (50%)  Ngo Viet Tung (50%) |

Table 1: Member contribution

# 8 References

*https://en.wikipedia.org/wiki/Naive\_Bayes\_classifier*. (n.d.).

*https://hands-on.cloud/implementing-naive-bayes-classification-using-python/*. (n.d.).

*https://machinelearningcoban.com/2017/08/08/nbc/*. (n.d.).

*https://machinelearningcoban.com/2017/08/08/nbc/*. (n.d.).

*https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397*. (n.d.).

*https://medium.com/towards-data-science/logic-and-implementation-of-a-spam-filter-machine-learning-algorithm-a508fb9547bd*. (n.d.).

*https://pythonmachinelearning.pro/text-classification-tutorial-with-naive-bayes/*. (n.d.).

*https://www.springboard.com/blog/data-analytics/naive-bayes-classification/*. (n.d.).

1. https://en.wikipedia.org/wiki/Naive\_Bayes\_classifier [↑](#footnote-ref-1)
2. https://www.upgrad.com/blog/naive-bayes-classifier/ [↑](#footnote-ref-2)
3. https://medium.com/towards-data-science/logic-and-implementation-of-a-spam-filter-machine-learning-algorithm-a508fb9547bd [↑](#footnote-ref-3)
4. https://www.springboard.com/blog/data-analytics/naive-bayes-classification/ [↑](#footnote-ref-4)
5. https://hands-on.cloud/implementing-naive-bayes-classification-using-python/ [↑](#footnote-ref-5)