Spam email Classification using Naive Bayes Algorithm

Aldrin C. Racusa
Department of Physical Sciences and Mathematics
College of Arts and Sciences
University of the Philippines Manila
Manila, Philippines
acracusa@up.edu.ph

Lorenz Timothy Barco Ranera
Department of Physical Sciences and Mathematics
College of Arts and Sciences
University of the Philippines Manila
Manila, Philippines
lbranera@up.edu.ph

***Index Terms*—Naive Bayes, Spam Classification, Email, Bayes Theorem, Machine Learning**

I. Introduction

Spam, referring to the sending of unsolicited email, is a crucial and increasing problem for both home users and companies. [1]. Recipients of spam often have had their email addresses obtained by spambots, which are automated programs that crawl the internet looking for email addresses. [2]

While there are many ways to prevent spam, one of the common techniques used is the Spam Filtering. A spam filter is a program that is used to detect unsolicited and unwanted email and prevent those messages from getting to a user's inbox. [3]

Naive Bayes is a simple, yet effective and commonly-used, machine learning algorithm. Naive Bayes classifiers have been especially popular for text classification, and are a traditional solution for problems such as spam detection. [4]

This paper aims to present a model using Naive Bayes Algorithm that would classify whether a specific email is a spam email or not.

II. Dataset

The dataset the is used in this paper is *TREC 2007 Public Corpus*. There are 75,419 email messages with 50199 being a spam email and 25220 for not spam email (ham).

III. Preprocessing

In Natural Language Processing, there are various preprocessing techniques implemented as preliminary procedures in handling text data, that may be words or documents, before representing them into the vector representations, such as bag of words model, TF-IDF, and document embedding. The rationale for this is because some words that are insignificant may appear in our corpus that could negatively affect the training of our dataset and eventually the model and its accuracy.

A. Removal of Insignificant Words

There are various texts that are insignificant however appears frequently like the string *"x x x"*. The possessive moneme " *'s* " may frequently appear in emails but it does not give any significant unique meaning to the email document. Other examples would be strings like *(a)*, *(b)*, *(c)*, *(1)*, *(2)*.

B. Tokenization

Tokenization is the process where a text, which is a string, is broken down into words and punctuation. Each unit is called *tokens* [5] [6]. One may think of it as decomposing a large string into an array of string, which are words of the document. This step is necessary because texts are converted into an array of words in NLP as a data structure.

For example, let us consider a short document which contains the text *"I am a Computer Science student"*. Tokenizing the said document will result in a [*'I'*,*'am'*, *'a'*, *'Computer'*, *'Science'*, *'Student'*, *"."*].

There is a python implementation of this available from the Natural Language Toolkit (NLTK) package. There is also a R programming language implementation of this available from the Text Mining (TM) package.

C. Lowercasing

Lowercasing is a preprocessing technique in which all words which is capitalized or has a capitalized character is converted in its lowercase form. This technique avoids the the case where words with different forms of cases are treated differently [7].

For example, let us consider a short email which contains the text *"Know what? Thanks to Anatrim, my marriage was luckily saved! I fell down into this circle, depression more eating more depression. My wife was about to leave me as I was turning in overweight psycho."* Without the use of lower casing, the words *"Anatrim"* and *"anatrim"* are treated differently. Applying lowercasing, the document is now *"know what? thanks to anatrim, my marriage was luckily saved! i fell down into this circle, depression more eating more depression. my wife was about to leave me as i was turning in overweight psycho."*

D. Removal of Non-alphabetical symbols

HTML tags, Non-UTF characters, Foreign language characters such as Japanese and German, strings with numerical values embedded, and removal of non-alphabetical symbols, mostly numbers and punctuation marks, are done as a preprocessing technique to the corpus. Punctuation symbols removed because they do not infer any significant meaning or contribute to the document's content at all depending on the domain of the document. In this study, all punctuation symbols, HTML tags, Non-UTF characters, and Foreign language characters are removed. [7] [**?**]. The same case can be said to numbers.

E. Removal of Stop-words

There are some words that appear almost in every text documents but does not provide any significant meaning. These words may also affect the performance of the model when employing Machine Learning and other NLP techniques. These words are called *stop words*. In effect, the vocabulary size of unique words in a corpus will also decrease leaving out all significant words [5] [6].

Linking verbs, conjunctions, and pronouns are examples of stop words in the English language. These words such as *"is"*,*"he"*, and *"and"* frequently appear in almost all of documents. In computing similarity between two documents, the two documents may be considered *"similar"* but it just so happens that both contains a huge frequency of these kinds of stop words. The same can be argued in classification of documents [5]. In topic discovery, these words cannot provide any meaningful *"topics"* in analyzing the distribution of topics in a documents. Overall, stop words are not significant.

There is a collection of English stop words that can be imported from NLTK in Python. There is also a collection of English stop words that can be imported from the TM package in R.

F. Stemming

Stemming is a preprocessing technique which converts or reduces words into its simplest form. This technique helps reducing the number of the features in a corpus [5] [6] [8]. This is not problematic since the various forms of words have similar meanings even if they are in different forms [9]. The different forms of a particular word will be treated as different words without stemming.

For example, let us consider the words *"runs"*, *"ran"*, and *"running"*. The three mentioned words will be converted into the simplified word *"run"*.

There is a python and R implementation of this available from the NLTK package/TM package respectively with different algorithms like the Porter Stemmer algorithm.

G. Removal of Less Occurring Words

Words that occurs in less than a number of a particular number of documents are removed in the document. This will remove the cases where words are misspelled which has a significant small number of occurrence. This technique will also remove rare words that may not significant context enough [**?**]. In R, there is a function called removeSparseTerms under the TM package that computes for the sparse rating of a certain word. For example, a term that appears say just 4 times in a corpus of say size 1000, will have a frequency of appearance of 0.004 =4/1000.

This term's sparsity will be (1000-4)/1000 = 1 - 0.004 = 0.996 = 99.6%. Therefore if sparsity threshold is set to sparse = 0.90, this term will be removed as its sparsity (0.996) is greater than the upper bound sparsity (0.90).

In this study, we will be using a sparsity threshold of 0.997 or 99.7% for the General Vocabulary, and 0.95 or 95% for the Reduced Vocabulary.

IV. Preprocessing Implementation

There are two kinds of vocabulary used in this study: *General* and *Reduced*. Only *stop-words* and *non-alphabetical symbols* are removed to create the *general* vocabulary while all preprocessing techniques discussed are used to generate the *reduced* vocabulary. There are 3370 unique words in the *general* vocabulary and 237 unique words in the *reduced* vocabulary.

V. Naive Bayes Algorithm

Naive Bayes is one machine learning technique that uses Bayes' theorem of probability in order to create a model that would classify labels based on a set of features. Let y_i be a particular *label* and X be a vector which contains all of the values for every *feature*. The probability of $p(y_i|X)$ is shown below.

$$p(y_i|X) = \frac{p(X|y_i) \cdot p(y_i)}{p(X)} \tag{1}$$

where $p(X)$ is defined below

$$p(X) = \sum_{i=0}^{m-1} \Big[p(X|y_i) \cdot p(y_i) \Big] \tag{2}$$

where there are m labels. $p(y_i)$ is defined below as

$$p(y_i) = \frac{n_{y_i}}{n} \tag{3}$$

where n_{y_i} is the number of dataset with label y_i and n is the number of dataset. $p(X|y_i)$ is defined below as

$$p(X|y_i) = \prod_{j=0}^{k-1} p(x_j|y_i) \tag{4}$$

where k is the number of features and $p(x_j|y_i)$ is defined below as

$$p(x_j|y_i) = \frac{n_{x_j,y_i}}{n_{y_i}} \tag{5}$$

where where n_{x_j,y_i} is the number of dataset with feature x_j and label y_i and n_{y_i} is the number of dataset with label y_i.

Applying Laplace smoothing will result in a change of the formulas presented before. Let λ be any constant.

$$p(y_i) = \frac{n_{y_i} + \lambda}{n + \lambda \cdot m} \tag{6}$$

$$p(x_j|y_i) = \frac{n_{x_j,y_i} + \lambda}{n_{y_i} + \lambda \cdot k} \tag{7}$$

where m is the number of labels and k is the number of features [10].

The data with feature values x is classified to label y_* where $max\{p(y_i|x)\}$ when $y_i = y_*$.

VI. Machine Learning Implementation

Naive Bayes is applied to both *general* and *reduced* vocabulary. Laplace smoothing is also applied with lambda value of 1 and is compared to without Laplace smoothing. The Naive Bayes Algorithm is implemented from scratch using R.

In total, four models are generated and compares in this study:

1) Naive Bayes *without* Laplace smoothing using *general* vocabulary dataset
2) Naive Bayes *with* Laplace smoothing using *general* vocabulary dataset
3) Naive Bayes *without* Laplace smoothing using *reduced* vocabulary dataset
4) Naive Bayes *with* Laplace smoothing using *reduced* vocabulary dataset

VII. Results and Discussion

The following are the results of the four models. Cross-validation was performed where 30% (22626 emails) of the dataset are used for testing and 70% (52793 emails) are used for training.

The first model is Naive Bayes (NB) without Laplace smoothing (LS) using general vocabulary dataset. The model got an accuracy of 0.6275082, precision of 0.3418803 and a recall of 0.1392982. The below in Table I and the confusion matrix is shown below in Table II.

TABLE I
NB w/o LS using general vocabulary

Accuracy	0.6275082
Precision	0.3418803
Recall	0.1392982

TABLE II
Confusion Matrix

	Actual: Ham (H)	Actual: Spam (S)
Pred: H	1040	2002
Pred: S	6426	13158

The second model is Naive Bayes with Laplace smoothing using general vocabulary dataset. The model also has an accuracy of 0.4981879, precision of 0.3285714 and a recall of 0.4990624. The results are shown below in Table III and the confusion matrix is shown below in Table IV.

TABLE III
NB w/ LS using general vocabulary

Accuracy	0.4981879
Precision	0.3285714
Recall	0.4990624

TABLE IV
Confusion Matrix

	Actual: H	Actual: S
Pred: H	3726	7614
Pred: S	3740	7546

The third model is Naive Bayes without Laplace smoothing using reduced vocabulary dataset. The model also has an accuracy of 0.6520817, a precision of 0.3445274 and a recall of 0.03634217. The results are shown below in Table V and the confusion matrix is shown below in Table VI.

TABLE V
NB w/o LS using reduced vocabulary

Accuracy	0.6520817
Precision	0.3445274
Recall	0.03634217

TABLE VI
Confusion Matrix

	Actual: H	Actual: S
Pred: H	277	527
Pred: S	7345	14477

The fourth model is Naive Bayes with Laplace smoothing using reduced vocabulary dataset. The model also has an accuracy of 0.6523469 , precision of 0.3439898, and recall of 0.03529257. The results are shown below in Table VII and the confusion matrix is shown below in Table VIII.

TABLE VII
NB w/ LS using reduced vocabulary

Accuracy	0.6523469
Precision	0.3439898
Recall	0.03529257

TABLE VIII
Confusion Matrix

	Actual: H	Actual: S
Pred: H	269	513
Pred: S	7353	14491

The fourth model is the best model among the four because it has the highest accuracy. This is mainly due to the fact that various preprocessing techniques were done for the fourth model that resulted in a lesser but more significant features (words) before performing the machine learning algorithm.

VIII. Conclusion

It is possible to classify *Ham* and *Spam* emails using Machine Learning specifically Naive Bayes classifier technique. In this study, we are able to generate a model with accuracy of 65.23% at best. Various and intense preprocessing techniques should be performed first in the corpus in order to generate a model with that accuracy.

References

[1] M. Siponen ; C. Stucke *Effective Anti-Spam Strategies in Companies: An International Study*. Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)

[2] "Email spam ". Retrieved in https://searchsecurity.techtarget.com/definition/spam

[3] "Spam Filter". Retrieved in https://searchmidmarketsecurity.techtarget.com/definition/spam-filter

[4] "Introduction to Naive Bayes Classification". Retrieved in https://towardsdatascience.com/introduction-to-naive-bayes-classification-4cffabb1ae54

[5] V. Gurusamy and S. Kannan, "Preprocessing techniques for text mining ," October 2014

[6] A. I. Kadhim1, Y.-N. Cheah, and N. H. Ahamed, "Text document preprocessing and dimension reduction techniques for text document clustering," in 2014 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology, 2014.

[7] M. J. Denny and A. Spirling, "Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it," 2017.

[8] S. Vijayarani1, M. J. Ilamathi, and Nithya, "Preprocessing techniques for text mining - an overview," International Journal of Computer Science and Communication Networks, vol. 5, no. 1, pp. 7-16.

[9] A. G. Jivani, "A comparative study of stemming algorithms," International Journal of Computer Technology and Applications, vol. 2, no. 6, pp. 1930-1938, 2011.

[10] P. Naval, "CS 280: Learning as Inference"