Spam email Classification using Support Vector Machine

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*—Support Vector Machine, Spam Classification, Email, Support Vector, 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]

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. [4]

This paper aims to present a model using Support Vector Machine 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* [8] [9]. 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 [10].

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. [10] [**?**]. 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 [8] [9].

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 [8]. 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 [8] [9] [11]. This is not problematic since the various forms of words have similar meanings even if they are in different forms [12]. 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 [**?**].

IV. Support Vector Machine

Support Vector Machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. [5]

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N being the number of features) that distinctly classifies the data points. [6]

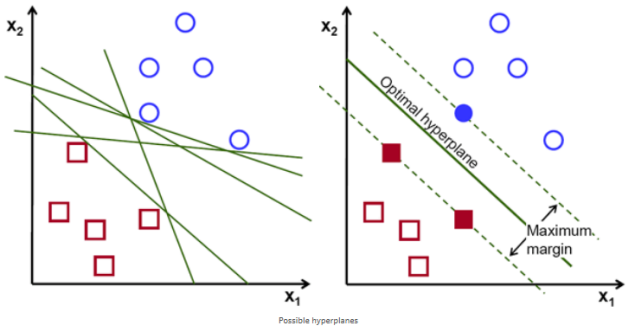

Fig. 1. Hyperplane

Some datasets are difficult to be seperated by a plane hence sometimes it is necessary that they are translated in another dimension. This technique is called *kernel trick*. There are four common kernel types used in SVM: *Linear*, *Radial Basis Function (RBF)*, *Polynomial*, and *Sigmoid* [7].

The most common parameters of SVM are *gamma* and *cost*. *Gamma* is the coefficient used in kernel types: *RBF, Polynomial* and *Sigmoid*. The gamma value influences correctness and fitness of the hyperplane in its dataset.

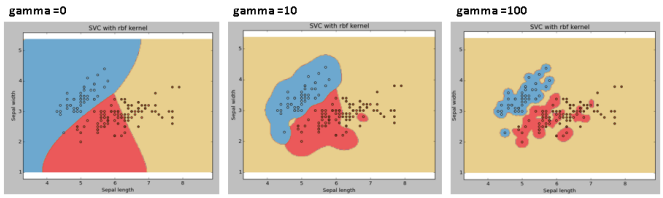

Fig. 2. Gamma parameter

Cost is the parameter that is used for the error term. It both influences the smoothness of the hyperplane and its correctness in classifying the labels.

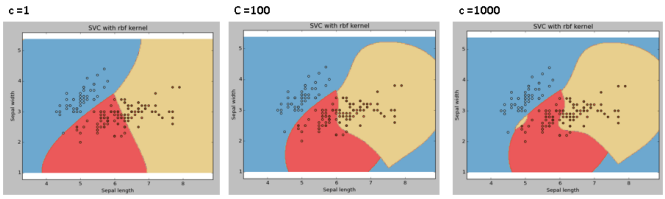

Fig. 3. Cost parameter

V. Methodology

The dataset that was preprocessed has been turned into a TF-IDF format using the *TfidfTransformer* in *sklearn* package.

The python library *pandas* was used in order to read the csv file and the library *sci-kit learn* was used to implement SVM, and a Pipeline to group the following: *tfidfTransformer*, *svm.SVC*, and *CountVectorizer*.

We then used a hyper-parameter tuning technique called *Grid Search* in order to get the best parameter that can produce the highest accuracy.

In order to implement the Grid Search, the dataset has been reduced to 1000 rows. Because the drawback of Grid Search is it is computationally expensive. Because it exhaustively get all possible combinations of the given parameter list in order to find the "*best*" parameters.

Cross-validation was performed where 25% of the dataset are used for testing and 75% are used for training in pipeline.

The dataset for the Pipeline without Grid Search has a 53991 rows for training set, and 17997 for the test set. While the Pipeline with Grid Search has a 675 rows for training and 225 rows for testing for a total of 1000 rows.

There is a total of 81 SVM models generated and analyzed in this study.

VI. Results and Discussion

The following are the results of the 48 models. Cross-validation was performed where 25% (17997 emails) of the dataset are used for testing and 75% (53991 emails) are used for training.

The first model is Support Vector Machine with a parameter of *gamma* = auto, *kernel* = linear, and *Cost* = 1. The model got an accuracy of 0.996, precision of 1.0 and a recall of 1.0. The below in Table I and the confusion matrix is shown below in Table II.

TABLE I
SVM WITH KERNEL=LINEAR, GAMMA=AUTO, COST=1

Accuracy	0.996
Precision	1.0
Recall	1.0

TABLE II
CONFUSION MATRIX

	Actual: Ham (H)	Actual: Spam (S)
Pred: H	6131	56
Pred: S	22	11788

The table in appendix are Support Vector Machines with Grid Search performed on 1000 observations in order to find the "best" parameter.

the best model among the 47 is with the *gamma* = 1, *kernel* = linear, and *cost* = 1 because it has the highest accuracy (0.99555).

VII. Conclusion

It is possible to classify *Ham* and *Spam* emails using Machine Learning specifically Support Vector Machine technique. In this study, we are able to generate a model with accuracy of 0.996% at best with the dataset of 75419 and a model with an accuracy of 0.995% at best with the dataset of 1000 observations. Various and intense preprocessing techniques, and a "good" parameter should be considered 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] "Support Vector Machine". Retrieved in http://www.statsoft.com/textbook/support-vector-machines

[5] "SVM wiki". Retrieved in https://en.wikipedia.org/wiki/Support-vector$_machine$

[6] "Support Vector Machine Introduction". Retrieved in https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[7] S. Ray. "Understanding Support Vector Machine algorithm from examples (along with code)," 2017. Retrieved in https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

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

[9] 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.

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

[11] 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.

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

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

VIII. APPENDIX

	accuracy	precision	recall	gamma	kernel	cost
0	0.9955555555555555	0.9955809523809525	0.9955555555555555	1	linear	1.0
1	0.9555555555555556	0.9579710144927536	0.9555555555555556	1	rbf	1.0
2	0.88	0.8961194029850745	0.88	1	poly	1.0
3	1.0	1.0	1.0	1	sigmoid	1.0
4	0.9955555555555555	0.9955809523809525	0.9955555555555555	1	linear	0.75
5	0.9422222222222222	0.9462388591800357	0.9422222222222222	1	rbf	0.75
6	0.8666666666666667	0.8862745098039215	0.8666666666666667	1	poly	0.75
7	1.0	1.0	1.0	1	sigmoid	0.75
8	0.9822222222222222	0.9826217228464419	0.9822222222222222	1	linear	0.5
9	0.8933333333333333	0.9062626262626262	0.8933333333333333	1	rbf	0.5
10	0.8044444444444444	0.843914373088685	0.8044444444444444	1	poly	0.5
11	0.9733333333333334	0.9742222222222222	0.9733333333333334	1	sigmoid	0.5
12	0.92	0.9275	0.92	1	linear	0.25
13	0.7911111111111111	0.8355354449472095	0.7911111111111111	1	rbf	0.25
14	0.7733333333333333	0.5980444444444445	0.7733333333333333	1	poly	0.25
15	0.9111111111111111	0.920274914089347	0.9111111111111111	1	sigmoid	0.25
16	0.9955555555555555	0.9955809523809525	0.9955555555555555	2	linear	1.0
17	0.8977777777777778	0.9097123519458545	0.8977777777777778	2	rbf	1.0
18	0.8888888888888888	0.9028475711892798	0.8888888888888888	2	poly	1.0
19	0.9955555555555555	0.9956410256410256	0.9955555555555555	2	sigmoid	1.0
20	0.9955555555555555	0.9955809523809525	0.9955555555555555	2	linear	0.75
21	0.88	0.8961194029850745	0.88	2	rbf	0.75
22	0.8888888888888888	0.9028475711892798	0.8888888888888888	2	poly	0.75
23	1.0	1.0	1.0	2	sigmoid	0.75
24	0.9822222222222222	0.9826217228464419	0.9822222222222222	2	linear	0.5
25	0.8177777777777778	0.8525271317829457	0.8177777777777778	2	rbf	0.5
26	0.8888888888888888	0.9028475711892798	0.8888888888888888	2	poly	0.5
27	0.9955555555555555	0.9955809523809525	0.9955555555555555	2	sigmoid	0.5
28	0.92	0.9275	0.92	2	linear	0.25
29	0.7733333333333333	0.5980444444444445	0.7733333333333333	2	rbf	0.25
30	0.8888888888888888	0.9028475711892798	0.8888888888888888	2	poly	0.25
31	0.9733333333333334	0.9742222222222222	0.9733333333333334	2	sigmoid	0.25
32	0.9955555555555555	0.9955809523809525	0.9955555555555555	3	linear	1.0
33	0.8533333333333334	0.8767149758454106	0.8533333333333334	3	rbf	1.0
34	0.8888888888888888	0.9028475711892798	0.8888888888888888	3	poly	1.0
35	0.9911111111111112	0.9914465408805031	0.9911111111111112	3	sigmoid	1.0
36	0.9955555555555555	0.9955809523809525	0.9955555555555555	3	linear	0.75
37	0.8311111111111111	0.8613836477987422	0.8311111111111111	3	rbf	0.75
38	0.8888888888888888	0.9028475711892798	0.8888888888888888	3	poly	0.75
39	0.9955555555555555	0.9956410256410256	0.9955555555555555	3	sigmoid	0.75
40	0.9822222222222222	0.9826217228464419	0.9822222222222222	3	linear	0.5
41	0.7822222222222223	0.8300747384155456	0.7822222222222223	3	rbf	0.5
42	0.8888888888888888	0.9028475711892798	0.8888888888888888	3	poly	0.5
43	0.9955555555555555	0.9955809523809525	0.9955555555555555	3	sigmoid	0.5
44	0.92	0.9275	0.92	3	linear	0.25
45	0.7733333333333333	0.5980444444444445	0.7733333333333333	3	rbf	0.25
46	0.8888888888888888	0.9028475711892798	0.8888888888888888	3	poly	0.25
47	0.9866666666666667	0.9868119164072922	0.9866666666666667	3	sigmoid	0.25