HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

MACHINE LEARNING AND DATA MINING
CAPSTONE PROJECT REPORT

Machine Learning for Email Spam Filtering

Authors:

Hoang Minh Tri - 20184315
Nguyen Tuan Minh - 20184294
Do Long Minh - 20184289

Cohort 63, ICT Program
SoICT, HUST

Supervisor:

Ph.D. Nguyen Nhat Quang

Information System Department
SoICT, HUST

Hanoi, Vietnam
June 28, 2022

Table Content

1. Introduction 01

2. Methodology 01

2.1. Dataset and pre-processing 01

2.1.1. Dataset 01

2.1.2. Pre-processing 03

2.2. Classifiers 04

2.2.1. K-nearest Neighbors 04

2.2.2. Naive Bayes 05

3. Result and Evaluation 07

3.1. Evaluation metrics 07

3.2. Testing result 09

3.3. Evaluation 09

4. Conclusion and Improvement 10

4.1. Conclusion and Obstacles 10

4.2. Improvement 10

5. Contributions 11

6. References 12

Abstract

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Recent machine learning methods are being used to detect and filter spam emails successfully. We present a systematic review of some of the popular machine-learning-based email spam filtering approaches. Our review covers a survey of the important concepts, attempts, efficiency, and the research trend in spam filtering. Our review compares the strengths and drawbacks of existing machine learning approaches and the open research problems in spam filtering. We recommended deep leaning and deep adversarial learning as the future techniques that can effectively handle the menace of spam emails.

1. Introduction

Email, electric email, is a method of exchanging messages between people using electronic devices. Email can contain documents, images, and attachments... and it is sent and received via the internet.

Spam email is an email referred to as junk email or simply spam, which is unsolicited messages sent in bulk by email. Sometimes, spam emails steal the information of the users for malicious purposes. Spam is caused by hackers who get the sender's email address leaked through the internet.

In this project, we implement two main classifiers to try to predict whether an email is spam or normal and compare them in many aspects. Besides, we have done pre-processing data, and tun some hyper parameters to improve the accuracy.

This project is conducted under the instruction of a Ph.D. Nguyen Nhat Quang.

2. Methodology

2.1. Dataset and pre-processing

2.1.1. Dataset

For this project, we use the spam mail dataset [1] collected from Kaggle. It consists of 5572 samples. In this data, there are 3 properties: *ID*, *Label*, and *Text*.

- ID: This property has a string type, representing each sample ID in the data set.
- label: *label* property has 2 main values: ham or spam. Ham stands for normal email and spam stands for spam email. In this project, we convert *ham* and *spam* into 0 and 1 respectively for easier processing.
- text: The document of the email is known as the content of the email (Subject, Body email...). We base on these to train the model; thus, the model can learn that email having its text is spam or normal.

The dataset contains 4825 ham and 747 spam messages.

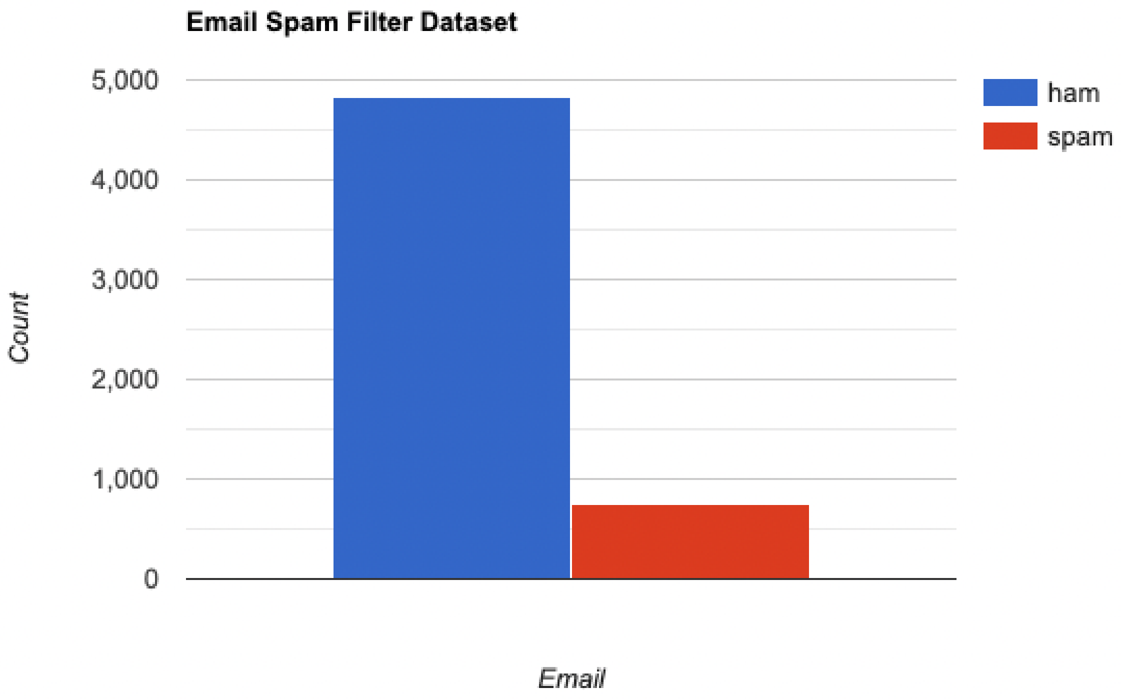


Figure 1: Overall the dataset

If we look further at the length of each email text; we can observe that spam emails are generally longer than ham emails. The bulk of ham has a length below 100, for spam, it is above 100.

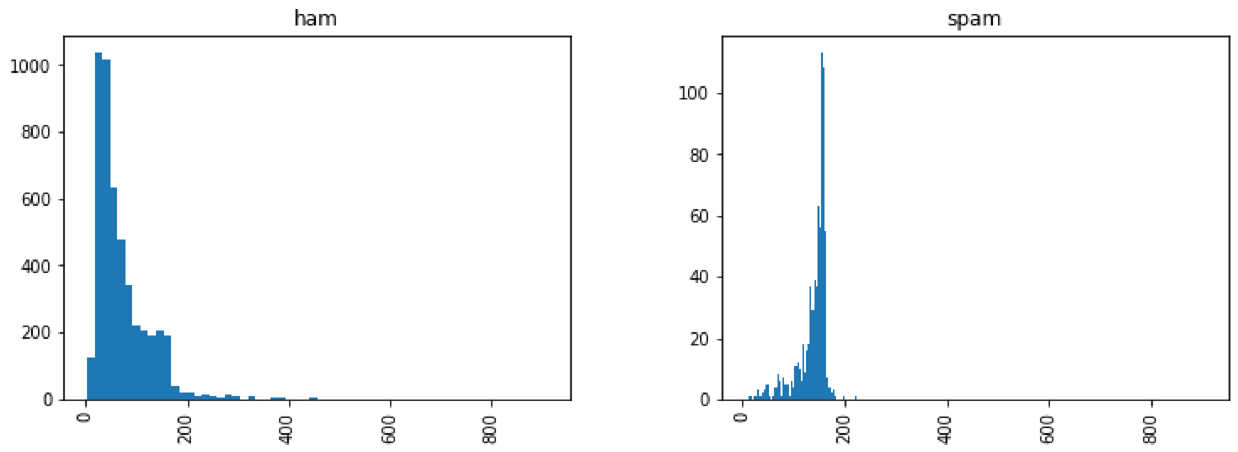


Figure 2: Email text length in ham and spam

2.1.2. Pre-processing

We choose both *label* and *text* for 2 main features. The *ID* is still included but it does not affect the result. For easier processing, we will convert the label into 0 and 1 corresponding to *Ham* and *Spam* values.

From the data, we can observe that there are many punctuation marks and stop words such as: "the", "is", "and"... which are not necessary for the learning of the model. Stop words are words like “and”, “the”, and “him”, which are presumed to be uninformative in representing the content of a text, and which may be removed to avoid them being construed as signals for prediction. Sometimes, however, similar words are useful for prediction, such as in classifying writing style or personality.

Hence, we decide to remove this redundancy by using a third-party nature langue process library supported in JavaScript: Winkjs.js. winkNLP [2] is a JavaScript library for Natural Language Processing (NLP). Designed specifically to make the development of NLP solutions **easier** and **faster**, winkNLP is optimized for the right balance of performance and accuracy. The package can handle a large amount of raw text at speeds over **525,000 tokens/second**. And with a test coverage of ~100%, winkNLP is a tool for building production-grade systems with confidence.

Together with the winkNLP library, we use the lemmatizer library [3]. In lemmatization, we try to reduce a given word to its root word.

Before	After

{
ID: '605',
text: 'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...',
": ",
status: 0
}
	{
	ID: '605',
	tokenization: [
	'jurong', 'point’, ‘crazy',
	'avail’, ‘bugis’, ‘n',
	'great’, ‘world’, ‘la',
	'e’, 'buffet', 'cine',
	'get’, 'amor', 'wat'
	],
	status: 0
	}

Table 1: Data before and after pre-processed

The last thing we try to pre-process the dataset is removing all emails with empty text content because an empty text will be converted into an empty array of tokenization which does not have any influence on the training progress. The dataset after filtering above contains 5560 records.

2.2. Classifiers

2.2.1. K-nearest Neighbors (KNN)

Theory

K-nearest neighbors [4], or KNN in short, is a supervised machine learning algorithm, that can be used for both regression and classification tasks. It is categorized as a “lazy” learning algorithm and an instance-based algorithm. Since our investigating problem belongs to classification problems, the reasoning of KNN here will mainly focus on its application in classification tasks.

KNN is one of the most intuitive supervised-learning algorithms (but has quite good efficiency in some cases). During the training process, it does not learn anything from the training data (therefore it is categorized as ‘lazy’), every calculation is made during the inference process with new data points (hence it must keep all the training data in the memory).

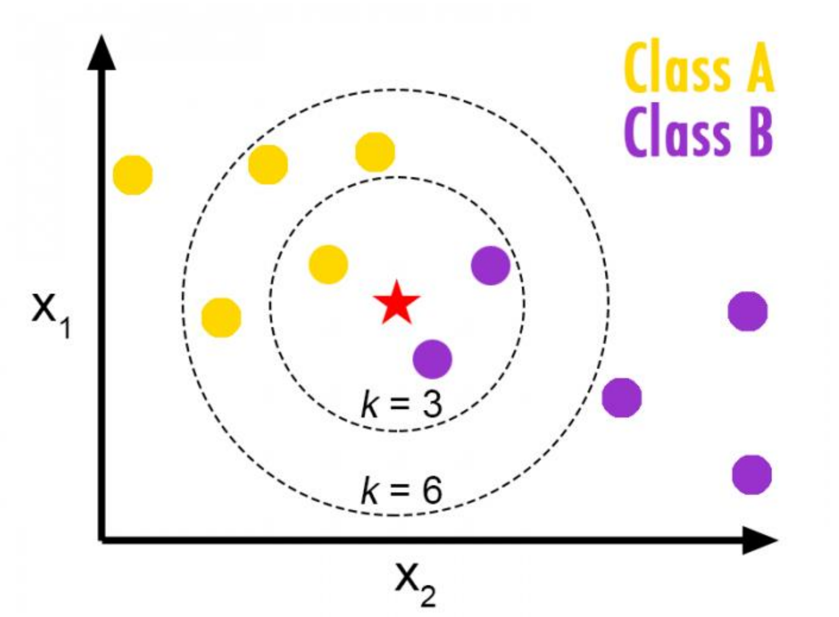


Figure 3: KNN illustration

The approach this method uses, instead of performing explicit generalization, compares new problem instances with instances seen in training, which have been stored in memory, therefore called an instance-based algorithm. In dealing with a 3 Figure 3: Illustration of KNN Model classification task, the label of one new data point is directly implied by K closest data points in the training set. It can be decided by major voting method among the K labels of its closest neighbors - with this method the weight of K neighbors is similar - meaning having similar impacts on the final decision. Another approach is to give weight to each label of the neighbor, this weight is chosen such that it is inversely proportional to the distance between the neighbor and the investigating point.

The basis of identifying the difference between the two data points in hyperspace is very important in the whole machine learning field and is also applied in the second method of identifying the label using KNN. To aid the calculation of this difference, the definition of the norm is established as a set of functions used to estimate the value - the length of a vector; and along with that length, another set of distance functions might be applied to calculate the distance.

Some common distance functions are widely used, namely:

- Euclidean distance
- Manhattan distance

Implementation

In this project, we only applied the Euclidean distance [5] for implementing the KNN.

Parameter: neighbors, weights, where:

- neighbors: the number of neighbors used as a reference to decide.
- weights: strategy used to measure the effects of each neighbor on results.

Steps:

- **Step 1:** Count the word frequency of one email content
- **Step 2:** Compute the Euclidean distance between two emails
    - **Step 2.1:** If a word appears in both train and test data, calculate the distance between their frequency
    - **Step 2.2:** If the word only appears in the train set, plus the distance with the square of the frequency. Delete common words in test data
    - **Step 2.3:** Looping through the rest test data, plus distance with the square of the word's frequency.
- **Step 3:** Sort data by ascending order and choose K neighbors that are nearest to the test data.
- **Step 4:** Evaluate the neighbors and decide.

2.2.2. Naive Bayes

Theory

The Naive Bayes algorithm is conducted based on Bayes's theory and is specifically suitable for filtering cases that require an amount of input. Bayes' theory is the outcome of Probability Theory, which refers to the condition of random variable A with knowing the distributed probability of A and B when A occurs. Filtering by Bayes's theory, the filtering method lies under a supervisor, is quite essential in consuming the human natural language. The theory is not only user-friendly (usage, installation) but also high efficiency.

To estimate the parameters for a feature's distribution, one must assume a distribution or generate nonparametric [6] models for the features from the training set.

Given the observed data X ($x_1, x_2, x_3, \ldots, x_n$), estimations of a probabilistic model's parameter C by Maximum a Posteriori (MAP) [7] is following:

$$C_{MAP} = argmax \; \underbrace{P(c \mid X)}_{posterior}$$

- Based on Bayes theorem and $c \in C$:

$$C_{MAP} = argmax \; \frac{P(X \mid c) \times P(c)}{P(X)}$$

- $P(X)$ is a constant and independent of C:

$$C_{MAP} = argmax \; \underbrace{P(X \mid c)}_{likelyhood} \times \underbrace{P(c)}_{prior}$$

Assumption: The attributes are conditionally independent given the classification.

$$P(x_1, x_2, x_3, \ldots, x_n \mid C) = \prod_{i=1}^{n} P(x_i \mid C)$$

Considering the evaluation of the probability above, the result may cause the underflow in the computer because the product is too small (the probability is smaller than 1). To avoid this scenario, we calculate log base 2 for both sides of the expression and get the final formula:

$$C_{MAP} \sim \sum_{i=1}^{n} \log_2 P(x_i \mid c) \times P(c)$$

Implementation

To implement Naive Bayes in this project, we must have a training set, where each training example is an object containing ID, text, and its label. The training set and test set are split from the dataset after pre-processing with the proportion of 0.7 and 0.3, respectively.

Based on each example's label, we divide the training set into 2 separated arrays: *rawSpamSet*, *rawHamSet*, and a *vocabulary*.

- *rawSpamSet* is an array containing all words in the spam class.
- *hamSpamSet* is an array containing all words in the ham class.
- *vocabulary* is an array containing all words that occurred in all email text without duplicating.

The model starts to train with the above arrays to get the probability for each corresponding word in the *vocabulary* where the class (ham or spam) is known. After training, the model now has a dictionary of words mapping to its probability in the given class.

The model predicts test email by applying the algorithm.

Note: Because there will be a word that only appear in ham or spam class the probability of that word will be 0 which causes the $P(c \mid X)$ will be 0. To avoid this scenario, we assume that each word will have appeared in class at least 1 time (add a black box to the features). Furthermore, the test email might have a word that does not exist in the *vocabulary*, this also makes the model break because its probability is undefined. We solve this problem by considering that word's probability in both classes will be 0.5. Adding both classes with the same amount of probability will not affect the result. The prior probability is not changed, it only depends on the training set.

3. Result and Evaluation

3.1. Evaluation Metrics

- Term:
  - TP: number of true positives (Ham emails are predicted as ham)
  - FP: number of false positives (Spam emails are predicted as ham)
  - TN: number of true negatives (Spam emails are predicted as spam)
  - FN: number of false negatives (Ham emails are predicted as spam)

	Predicted as Positive	**Predicted as Negative**
Actual: Positive	TP	FN
Actual: Negative	FP	TN

- In these metrics, we choose the *ham* class as positive and the *spam* class as negative.

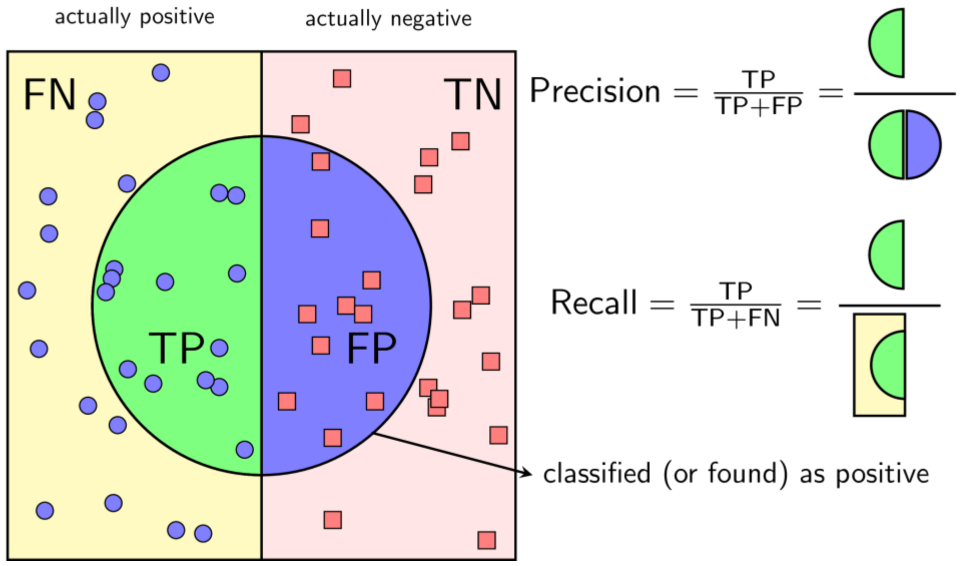
- Accuracy score:
  - The simplest and most used way is accuracy.
  - This evaluation simply calculates the ratio between the number of correctly predicted points and the total number of points in the test data set.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

- F1 score:
  - With the classification problem and the data set of the classes being very different from each other, there is an effective operation that is often used, which is Precision-Recall.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall}$$

  - $precision = \frac{TP}{TP + FP}$
  - $recall = \frac{TP}{TP + FN}$



actually positive
actually negative
FN
TN
TP
FP
Precision = $\frac{TP}{TP+FP}$ =
Recall = $\frac{TP}{TP+FN}$ =
classified (or found) as positive

Figure 4: Precision and recall illustration

3.2. Testing result

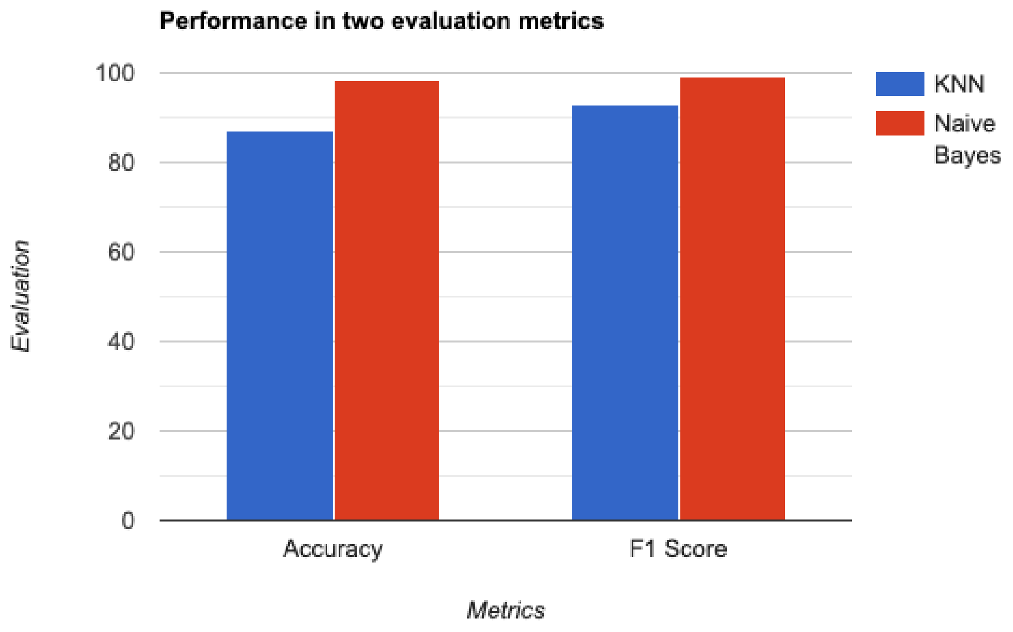


Figure 5: Performance illustration

Evaluate Metrics / Algorithm	Accuracy	F1 Score
KNN	86.93%	93.00%
Naive Bayes	98.44%	99.10%

Table 2: Testing results

3.3. Evaluation

The original dataset contains a lot of outliers as described in 2.1 and this affected a lot on the performance of the models. With original data for training and testing, the accuracy for all classifiers is quite moderate (accuracy around 73%, F1 score around 60%). After applying preprocessing with a very simple idea (describe in 2.1.2, the accuracy and F1 score are improved significantly (accuracy: increases 8-13%, F1 score: increases 20-22%). One more disadvantage of this dataset is that this is quite an imbalanced dataset. This approach needs more experiments and tunning for better accuracy.

According to the result of the 2 classifiers, Naïve Bayes is better than KNN, since the KNN is poor with the problems that have many dimensions to calculate (each distinct word is 1 dimension).

However, KNN without filling missing data has performed poorly due to its sensitivities with outliers, preprocessing, and oversampling, which can improve the result because the data now is more balanced and cleaner.

4. Conclusion and Improvement

4.1. Conclusion and Obstacles

The Naive Bayes algorithm brings unexpected results and a high recognition rate. The classification results are high but only in a specific condition and problem, not in all other models.

The model above only aims at the content of the email. Email also has other characteristics to classify as subject, sender address. Building a model with all the above characteristics is a big problem for our group.

One of our main concerns is the case that the need to be classified email having two probabilities at both classes nearly the same, leading to confusion in predicting. Then the model will be misclassified.

About the KNN algorithm, we see that K-NN is intuitive and simple: K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through the whole dataset to find out K-nearest neighbors. K-NN has no assumptions: K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN. Parametric models like linear regression have lots of assumptions to be met by data before it can be implemented, which is not the case with K-NN. No Training Step: K-NN does not explicitly build any model, it simply tags the new data entry-based learning from historical data. The new data entry would be tagged with the majority class in the nearest neighbor.

Otherwise, the K-NN slow algorithm: K-NN might be very easy to implement but as the dataset grows efficiency or speed of the algorithm declines very fast. Curse of Dimensionality: KNN works well with a small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data points. Outlier sensitivity: K-NN algorithm is very sensitive to outliers as it simply chooses the neighbors based on distance criteria.

4.2. Improvement

After experimenting with email spam filtering classifiers, we see that, generally, in machine learning, the training set is very important which may decide whether the Naive Bayes classifier model is good or not. The more valid data we collected (labeled email with the subject, sender address, etc.) which is suitable for our model, the better improvement our classification model will be.

5. Contributions

- Do Long Minh: Pre-process data with lemmatization, crawl data, implement Naive Bayes class, cross-validate, write the report, and make the presentation.
- Nguyen Tuan Minh: Pre-process data with tokenization, implement KNN class, cross-validate, write the report, and make the presentation.
- Hoang Minh Tri: Pre-process data, create Email model, run, and test model, write the report, create the PowerPoint presentation, and make the presentation**.**

6. References

Reference projects:

https://www.kaggle.com/code/dejavu23/sms-spam-or-ham-beginner/notebook
https://www.sciencedirect.com/science/article/pii/S2405844018353404

Denoted documents:

[1]: https://www.kaggle.com/code/dejavu23/sms-spam-or-ham-beginner/data

[2]: https://winkjs.org/wink-nlp/

[3]: https://npm.io/package/lemmatizer

[4]: https://towardsdatascience.com/spam-email-classifier-with-knn-from-scratch-python-6e68eeb50a9e

[5]: https://en.wikipedia.org/wiki/Euclidean_distance

[6]: https://en.wikipedia.org/wiki/Nonparametric_statistics

[7]: https://machinelearningcoban.com/2017/07/17/mlemap/#-y-tuong-1