E-mail Spam Filter based on Naïve Bayes Classifier

Sanchit Deora
Department of Computer and Information Science Engineering
University of Florida
sanchitdeora@ufl.edu

***Abstract*— Email or electronic mail is a medium for exchanging messages or mails on a digital platform amongst users. In today's professional world, emails are considered to be the best means of communication in a professional or friendly manner. It helps deliver messages to the users in a private, allowing them to be secure and free to use. Businesses cannot function without the use of emails, may it be used for internal communication between colleagues or to reach out to new, or existing customers outside. Some of these businesses tend to send bulk emails to all their customers for several reasons. It could be for a brand promotion, or market and advertise a new product, or just to remind the users about their products. These emails lead to spamming of the user inboxes. Email Spams are junk emails sent out in bulk which are deemed unwanted and unsolicited by the users. These emails are helpful for the marketeers; however, it leads to a rise of unwanted emails, making it easier to miss out on higher priority emails. Therefore, email spam filter is a necessary tool for every user to have. It is provided by every email service provider, to help discriminate if the emails are a spam email or an important email. In this project, I will build an email spam filter using various methods to solve this problem.**

***Index Terms* — Spam Emails, Ham Emails, Text classification, CountVectorizer, Tfid vectorizer, Naïve Bayes Classifier.**

I. Introduction

EMAILS are the most widely accepted form of digital communication. People use emails on a daily basis for several reasons, it could be used for business needs, such as importing or exporting goods to consumers, or could be used for internal business communication, to network and connect with other likeminded people in the corporate world, to send or receive multimedia files, or communicating with a closed one. Emails have changed the way for digital communication for the better. According to Statista, in a survey conducted during 2017-2018, an estimate of 281 billion emails were sent and received in a single day in the year 2018, predicting it to rise to 347 billion emails by the year 2023. Emails are secure and personal, keeping the messages confidential as well as having no cost makes it a very reliable source of communication for the businesses. This results in businesses using email in bulk, since having no cost makes it economically viable to contact their customers, may it be new, existing or from the future to promote their brands, and market or advertise a new product. This leads to the customers receiving multiple emails from several marketeers. To avoid such situations, all email service providers use an email spam filter to discriminate these unwanted emails and distinguishes them from the important emails.

In this project, I am aiming to building an email spam filter using Naïve Bayes Classifier Algorithm, a collection of basic probabilistic classifiers which is implementing Bayes' theorem with naïve independent assumptions among the features. This project is a text classification problem to distinguish between the emails a user receives as *ham emails* or *spam emails*. To solve this problem, I will use Natural Language Processing or NLP, a domain of machine learning which analyzes, manipulates and handle textual or human language data. The term *Spam* is a class label used to classify the email received by the user is a spam email. Similarly, *Ham* is a class label used to classify the email received by the user is an important or a non-spam email. Fig 1. displays the basic overview of email spam filtering.

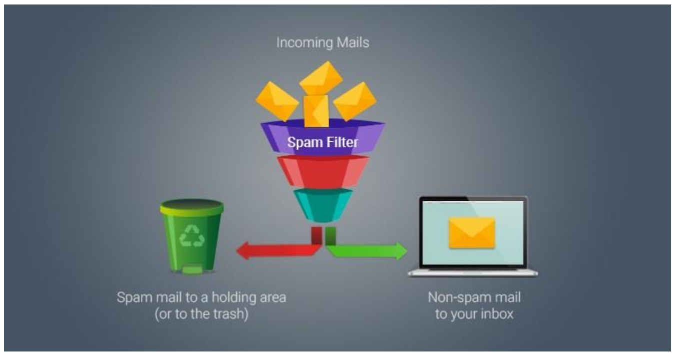

Incoming Mails
Spam Filter
Spam mail to a holding area (or to the trash)
Non-spam mail to your inbox

Fig 1. Email Spam Filter Overview

II. Description

Email Spams are junk emails sent out in bulk which are deemed unwanted and unsolicited by the users. The objective of these email spamming is usually for promotion and advertisement of products and brands since it is economically very viable. There may also be a malicious intent behind these email spams since they could be used for phishing attacks, contain viruses and risk your system or some other form of attacks. This email spamming leads to a rise of unwanted emails, making it easier to miss out on higher priority emails.

Email Spam Filter is a valuable tool which helps identify these spammed emails and segregates them into spam and ham emails category to avoid the problems mentioned earlier. The discussion about the dataset used for this project has been done in the next section.

A. *Dataset*

A dataset, as the name suggests, is a set of data. It consists of various formats such as tabular form with multiple rows and columns, an unstructured raw data, collection of articles or other text format, audio files, images, or videos. This data is used to obtain certain measurable attributes which could be used by the algorithms and models to perform the required actions. These attributes are called features.

In this project I used the dataset provided by Apache SpamAssassin [1]. This chosen dataset contains text-based emails. There is a total of around 4000 ham emails and around 2000 spam emails. The training set to test set ratio is around 3:1 or a split of 75% and 25% respectively, with each set consisting of a total of 4500 emails to 1500 emails. In each set the ratio of ham emails to spam emails is around 7:3 or a split of 70% and 30% respectively. Therefore, the training dataset consists of around 3000 ham emails and 1500 spam emails; and the testing dataset consists of around 1000 ham emails and around 500 spam emails.

The dataset has a corpus of spam and ham emails in separate dedicated folders. It consists of 3 categories of emails. Ham emails which are easily recognized are stored in the folder named ‘ham’ and the spam emails which are easily recognized are stored in the folder named ‘spam’. The third category are the ham emails which may be difficult to distinguish from the spam emails are stored in the folder named ‘toughHam’. I used these folder names to identify and create labels for the data. In the toughHam category, the label was given as Ham. The details for the label creation are provided in the data preprocessing phase.

B. *Exploratory Data Analysis*

Exploratory Data Analysis or EDA is the beginning step in the pipeline for this project. Exploratory data analysis is a technique to explore and analyze the data and understand its characteristics. This step is used to learn more about the data. To expand further, datasets could be of different types such as records, graphs, networks, multimedia contents like audio, video and images, or an ordered series of information such as transactional data or time-series data. It could also be of various formats such as structured data like relational database formatted content, semi-structured data like XML texts or unstructured data like multimedia contents and email or text messages. The data are also of various type such as nominal or categorical data, binary data, ordinal data, discrete or continuous numerical data, and interval or ratio data.

This chosen dataset is an unstructured data set which consists of emails, a textual formatted data. In this project, I stored the paths of the dataset in the program variables from the files. These data paths are segregated into folders named spam, ham and toughHam. These folder names are used to generate the labels for the dataset where the ham and toughHam emails are denoted by the class label ‘0’ and spam emails are denoted by the class label ‘1’. Now, the data paths are split into training data and testing data paths along with their corresponding labels in the ratio of 3:1 or a split of 75% and 25%. After splitting the training and testing dataset, I visualized these splits in the form of a bar and pie chart to better understand and explore the data shown below in Fig 2—5.

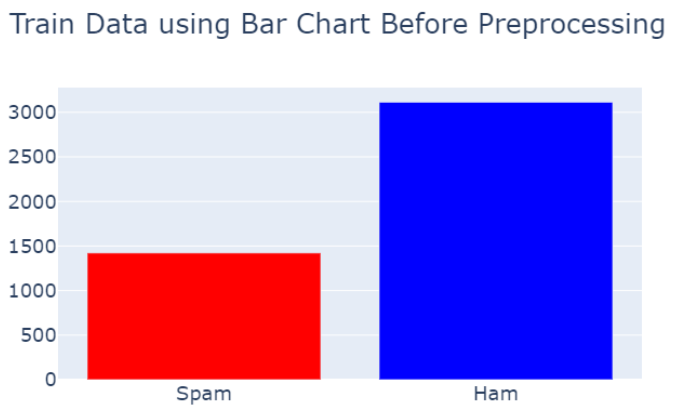


Fig 2. Label count for Training data using Bar chart

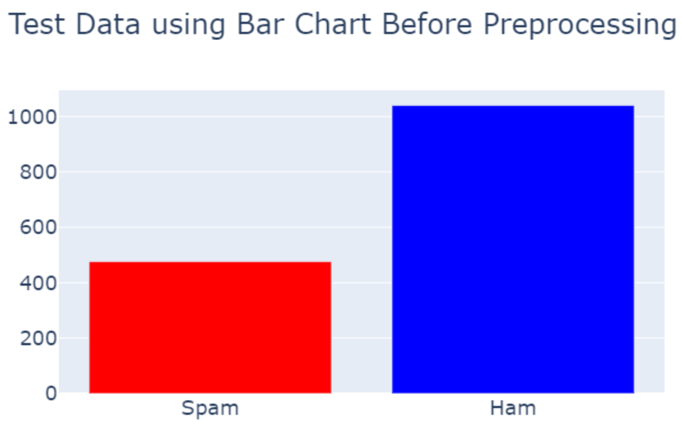


Fig 3. Label count for Testing data using Bar chart

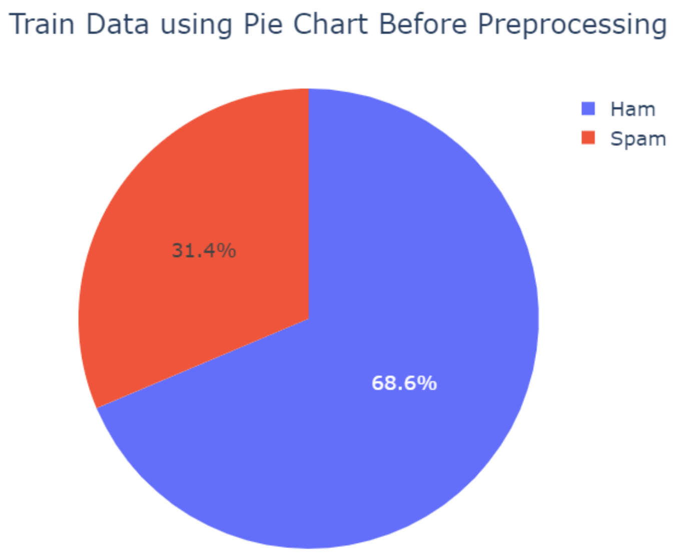


Fig 4. Label count for Training data using Pie chart

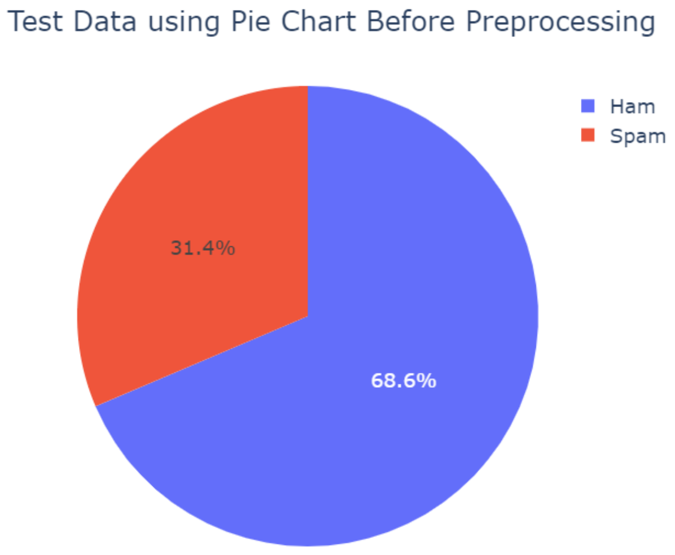


Fig 5. Label count for Training data using Pie chart

This training and testing data paths are used to obtain the email content in each of the file at those data paths. To obtain these emails from the file, python uses a library 'email' and stores it in 'x_train' and 'x_test' along with their labels stored in 'y_train' and 'y_test'. These training and testing data are now ready to be preprocessed.

C. *Data Preprocessing*

Data Preprocessing is the second step of the pipeline for this project. In every Data Science pipeline, Data Preprocessing is the phase which is used to clean and encode the data, to a state which the algorithm or model can interpret with ease. The datasets are sometimes been collected from several sources having various formats. This step is used to transform these datasets in a uniform manner to avoid facing errors and issues by the model. Data preprocessing involves several processes like data cleaning, feature subsampling, feature engineering and transformation and feature selection. Data cleaning involves handing missing values and inconsistent values and deleting duplicate values. To handle these missing values, it either deletes the entire feature or adds a default value. Feature subsampling is a process of choosing subsets of the dataset to avoid the overhead on the system. Feature engineering and transformation is a process of creating or transforming a feature into meaningful attributes from the existing features to provide more accurate information to the system. Feature selection is a process to selecting more meaningful attribute from the list of features provided in the dataset. These preprocessing methods provide a variety of techniques to improve the quality of the dataset to improve the results and accuracy of a model while avoiding unnecessary errors.

Data preprocessing of a dataset is an important phase of the project since the datasets tend to contain some unneeded noisy data that needs to be handled. In this project, I start with data cleaning on the dataset. The first step was to search for the dataset and deleting all null values in the dataset. Since this dataset is of unstructured text format, having null values in an email provides no meaning to the data and deleting is the only solution. This step was performed by iterating over all the emails in the training and testing dataset, and checking the condition, if the data is null it is removed from the dataset.

After the dataset is free of all null or empty emails, I start cleaning the data by removing the noisy characters in the email. I use string functions and regular expression 're' library available in Python. When I look at the emails, I am looking for important and frequent keywords in both spam emails and ham emails. To obtain these important keywords, I need to remove the other information in the email that seems less meaningful than those keywords, which can be considered as noisy data. These noisy data include hyperlinks, punctuations and extra white spaces. However, before checking for stop words, we also need to make sure that the words we are preprocessing are in lower case. This will ensure that the same words existing in different cases in the body of these emails are not treated differently. This will improve the quality of the data we output after this phase.

I created a pipeline of functions to remove these noisy data from the email. The program iterates over each email and follows these functions in the pipeline as shown in Fig 5. This pipeline consists of the functions given as follows:

- **removeHyperlink(word):** This function uses a regular expression or simply the package 're' to remove the hyperlinks from the emails. I remove all the words which starts with 'http'.
- **repalceNewline(word):** This function uses a simple string replace function available in python. It replaces the new line in an email with a simple whitespace. I perform this function using string.replace function and replace '\n' with a white space.
- **removeSpecial(word):** This function uses a regular expression to remove all the special characters in the emails. Special characters provide no meaning compared to the keywords in the emails which will be used for modeling and hence, should be removed. These characters include '@', '#', '$', '%' to name a few. To remove these mentioned characters, I check if they exist between the alphabets A-Z or a-z or if they are a number between 0-9.
- **removeWhitespace(word):** This function is used to remove excess whitespaces in the beginning or end of the word by using simple string strip function in python.
- **lowercase(word):** This function is used to convert the words in email to lowercase for uniformity in the email data. This step is important since after this step the word, 'Free' and 'free' will not be treated in a differently since 'Free' will be converted to 'free' itself. This function is performed by a simple lowercase string function in python.

After data cleaning, I start with sentence processing of the preprocessing phase. In sentence processing, I aim to manipulate the words in the emails to get a higher quality, concise keyword that would be used for the next phase. Sentence processing is a key step to perform word reduction to its most basic form. In the body of an email, there could be words of multiple forms which belong to the same word at its most basic form. Without sentence processing, these words would be considered as separate values making the list of keywords longer and redundant. These steps include removing stop words, word stemming and word lemmatization. To start the sentence processing, I first need to convert these strings into word tokens to easily parse through these word reduction algorithms. To perform this operation, I imported 'word_tokenize' from 'nltk' library available in python. This function allowed the email body to be tokenized into words. After performing word tokenizing, I start the sentence processing phase.

Stop words are collection of words which are used commonly in languages, which could be English or any other language as well and have less significance in comparison with other words. These usually include articles, prepositions, and conjunctions such as 'a', 'an', 'the', 'with', 'and', 'for', 'but', 'after', to name a few. These words are chosen to be removed from the dataset since it only returns unnecessary information.

Word stemming is a heuristic technique most commonly used in information retrieval and natural language processing in data science. This technique is used to reduce words to its basic word stem or root word. It removes the common suffixes or prefixes of the word in that language to return a basic stem. Let me explain with an example. Consider the word 'working'. Now using the stemming algorithms, it will identify its common suffix is '-ing' and remove that to return the basic stem word 'work'. The returned word stem is not always expected to be a word which is acceptable by the machine. For example, if we apply the stemming algorithm on the words, 'loved', 'loving', 'loves', it will detect the common suffixes like '-ed', '-ing', '-es' and return the stem word 'lov' which the machine considers to be the keyword for all the forms of 'love'. For this reason, stemming algorithms are very easy to implement and have a higher speed performance. They also return a good recall value.

Word Lemmatization, like Stemming is a heuristic technique most commonly used in information retrieval and natural language processing in data science. This is an algorithmic process used to detect the accurate part of the speech along with the tense and the meaning implied in the usage of that word. It is used to reduce the word to its basic lemma based on its targeted meaning. A lemma of a word is its basic form capturing the meaning of the words use. Let us consider the word 'working'. Now using the lemmatization algorithms, it will identify its common continuous tense that is '-ing' and remove that to return the basic lemma word 'work'. Unlike the stemming algorithms, in this algorithm the returned lemma are accurate words found in the dictionary since it understands the context the word was used in the email. For example, if we apply the lemmatization algorithm on the words, 'loved', 'loving', 'loves', it will detect the different tenses and part of speech the words were used in and remove suffixes like '-ed', '-ing', '-es' to return the lemma word 'love'. It will also return a lemma which is completely different from the word intended. For example, the word 'better' will return its lemma to be 'good'. These algorithms are more accurate and improves the precision than the stemming algorithms. However, for some applications where the accuracy of these reduced form of words does not affect the result, the lemmatization algorithms have a higher overhead and reduces the performance of the application.

I created a second pipeline of functions to process these sentences in the manner I described earlier. In this pipeline, the program iterates over each email and follows these functions as shown in Fig 6. This pipeline consists of three functions given as follows:

- **removeStopwords(words):** This function is used to remove all the stop words in the emails to obtain the keywords. To perform this function, we first need to collect a set of stop words to compare each word with. This set could be obtained from Natural language toolkit, or 'nltk' a python library available with functions to help with natural language processing. After importing 'stopwords' from 'nltk.corpus', I downloaded the English stop words and stored it in a set. Now that I have a set of stop words, I can iterate over the stop words and check if the current word which was input to the function matches them. If the word in the function is not matching to any of the stop words in the set, it is returned, else the function ignores the stop word.
- **wordStemmer(words):** This function is used to find the most basic stem form of the word passed to the function. Python 'nltk' library provides us with two different stemming algorithms. In this project, I used Porter's PorterStemmer algorithm to apply the word stemming technique. This function allowed the words in the emails to be transformed to its stem form to avoid having multiple forms of the same word to be considered as different words.
- **wordLemmatizer(word):** This function is used to find the most basic lemma form of the word passed to the function. Python 'nltk' library provides us with a WordNet Lemmatizer, which is a lemmatization algorithm that looks up the lemmas of the words in the WordNet database. This technique is used to return valid English words which the stemming algorithms fail to do.

Note that, I have mentioned both word stemming techniques and word lemmatizing techniques in the program, however, I will be using only one of them. Since the validity of the English word is not a criteria for the distinguishing of emails between ham and spam, just the frequency of the basic words, I choose to run the word stemming algorithm, over the lemmatization algorithm to improve the performance of the application.

This concludes the data preprocessing phase. This step of the project pipeline has returned a collection of processed keywords, which is free of hyperlinks, new line escape characters, whitespaces, special characters, stop words, and reduced to its basic form. I have performed an N-gram model visualization using bar chart to understand the data these processed keywords provide. This bar chart visualizes the frequency of shown in Fig 6.

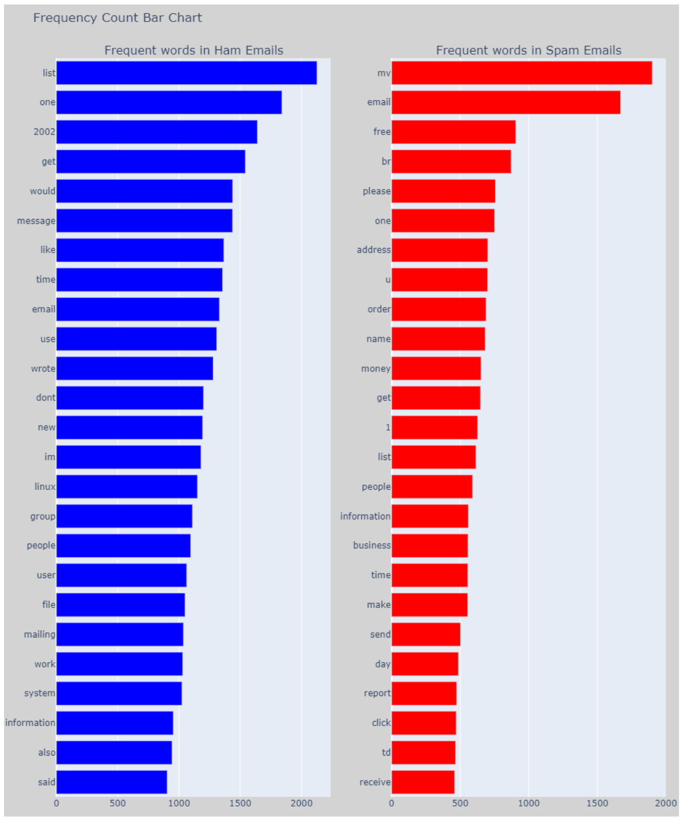


Fig 6. 1-gram model visualization of frequency of words in spam and ham

After this step, we must now convert these keywords to a numeric value for the computer program or model to interpret. This leads to the next step in the project, feature extraction.

Data preprocessing will start with data cleaning, where all the missing values are filled in and incorrect values are handled appropriately. For multiple sources of the data, it needs to be uniform for the model to classify the data accurately. Hence, data transformation is performed, where dataset has two columns, the email content and the target label, spam or ham.

D. *Feature Extraction*

Feature extraction is the third step in the project pipeline. It is a process of extracting features from the available data and making it more meaningful for your algorithm. It involves dimensionality reduction since it selects or reduces the feature vectors from the dataset.

In the previous step, data preprocessing phase returned a list of processed keywords. Since these data contains texts, we need to transform this information into something which is more friendly to the algorithm in this project. This feature extraction step will be used to convert these words which it receives from the data preprocessing step and will transform it into numeric features. These numeric values are acceptable by the Naïve Bayes Classifier which would help model this email spam filter. There are various techniques to transform this data into numeric features. In this project, I decided to use two bag of words approaches to perform feature extraction from the data ─ Count vectorizer and Tf-Idf vectorizer.

Count vectorizer is a text feature extraction technique which uses counts of tokens or words to calculate the importance of words. It uses a vocabulary dictionary of every word from the email content and stores their counts as values. Count vectorizer is a Python library which is available in the text sub-module of sklearn.feature_extraction.text. Using count vectorizer, my algorithm accepts a list of tokens. These tokens are just words from the training dataset of emails which were tokenized in the data preprocessing step. These training dataset tokens are used to train the count vectorizer algorithm using the function fit(). This algorithm uses the word tokens in the dataset to learn and create a vocabulary dictionary of words and generate a unique id for each word. This id used to keep a check of the count. It iterates over words and finds the id in the vocabulary dictionary. This id is incremented every time the corresponding word is found in the training dataset. After the vocabulary dictionary is completed and the algorithm is trained, it transforms these tokens to a document-term sparse matrix using the function transform(). This transform function is called for both training and testing dataset. Using the vocabulary dictionary created by the training dataset, count vectorizer is able to generate a sparse matrix for both training and testing dataset. These matrices are then converted to an array to have a contiguous one-dimensional input for the next step, Naïve Bayes Classifier.

Term frequency–Inverse document frequency vectorizer or Tf-Idf vectorizer is an alternative approach to transforming the processed tokenized words returned by the data preprocessing step to an array vector of numeric features to feed the model, Naïve Bayes Classifier. Similar to count vectorizer, Tf-Idf vectorizer, is a text feature extraction technique which is used to calculate the term frequency score and inverse document frequency score as the name suggests. Tf-Idf or Term frequency–Inverse document frequency is a popular statistic used in information retrieval to identify and understand the significance of a word corresponding to a document in the large corpus or dataset. It is used to weigh the importance of a particular word in search retrievals or modeling algorithms. The Tf-Idf score is proportionally increasing to the frequency of the word appearing in the email. However, it is downscaled by the word appearing across emails in the large dataset. This helps determine the importance of some words existing more often than others across emails in a dataset. Term frequency is used to calculate how frequent a word exists in a single document or email in this case. Inverse document frequency is used to calculate the significance of the word across all documents or emails. As the name suggests, it assigns the lowest inverse document frequency score to the most frequent term in across all documents.

Tf-Idf vectorizer is a Python library which is available in the text sub-module of sklearn.feature_extraction.text. Using Tf-Idf vectorizer, my algorithm accepts a list of tokens in an email. These emails are just a collection of word tokens generated by the data preprocessing step. The training dataset of emails which were processed earlier are used to train the algorithm of Tf-Idf using the function fit(). This algorithm uses the tokenized words in an email and use it to learn and create a vocabulary dictionary and calculates the inverse document frequency for all the words across all emails. After calculating the scores, it starts encoding each email. Each word in the vocabulary dictionary is assigned a unique id. Now using these unique ids, the inverse document frequency of each word is calculated and assigned to the id. Finally, each email is encoded as the words in each email is encoded by normalizing the inverse document frequency scores between the range of 0 to 1. After the training of this algorithm is complete, the transform() function is used to obtain a sparse matrix of Tf-Idf features. This transform function is called for both training and testing dataset. Using the encoded features from the vocabulary dictionary created by the training dataset, Tf-Idf vectorizer is able to generate a sparse matrix for both training and testing dataset. These matrices are then converted to an array to have a contiguous one-dimensional input for the next step, Naïve Bayes Classifier.

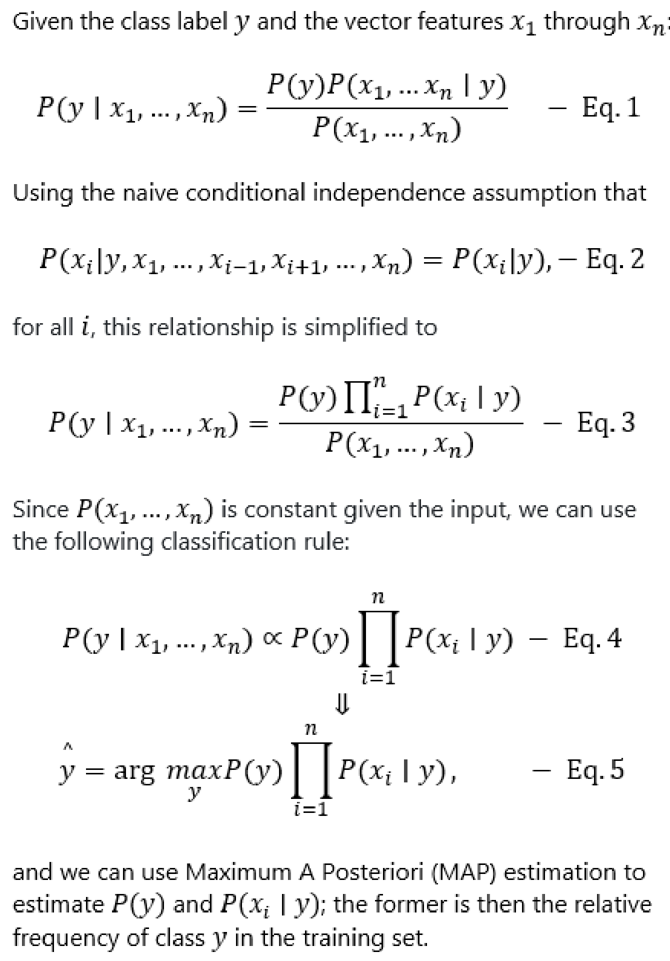
Tf-Idf vectorizer overcomes the shortcomings of count vectorizer since count vectorizer only keeps the total count, there could be a case where a word has a very high count in a single email while not being present in any other email. This would mislead the model to think that the word to be important due to the high count frequency. Tf-idf overcomes this by finding the importance of the word across all the email in the corpus.

These two text feature extraction techniques return a vector of numeric values which are real numbers. This vector is a transformation of the words that was originally a part of the email into real numbers representing those words. The output of this Feature Extraction step in the project pipeline will be fed to the model Naïve Bayes Classifier. This is explained in detail in the next phase.

E. *Model Training*

Model Training is the fourth step of the project pipeline. Model training phase is when you have the correct data for your model to start learning. In this project, I am using Naïve Bayes Classifier.

Naïve Bayes classifier is a family of probabilistic classifiers which uses Bayes' theorem involving a strong independence amongst the features. Naïve Bayes uses a simple conditional probability model in which each feature in a document is given a strong independent weight to calculate the probability which is used to predict the class label of that document. It makes a naïve assumption that each feature in the vector provides an equal and independent weightage to the result. These assumptions are usually only valid in an ideal world, whereas in the real world these features have an unequal impact on the given results. This is the reason why this classifier is called Naïve. This naïve classifier uses Bayes theorem to predict the outcomes and hence called the Naïve Bayes classifier. The relationship in Bayes Theorem is stated in Fig 7 with the help of Eq 1 — Eq 5.



Given the class label y and the vector features x_1 through x_n:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)} \quad - \text{Eq. 1}$$

Using the naive conditional independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y), - \text{Eq. 2}$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)} \quad - \text{Eq. 3}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \quad - \text{Eq. 4}$$

$$\Downarrow$$

$$\hat{y} = \arg \max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y), \qquad - \text{Eq. 5}$$

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

Fig 7: Bayes Theorem formulae

In machine learning, Naïve Bayes consists of a collection of supervised learning algorithms based on the Bayes' theorem. It is one of the most important baseline implementations in the sklearn module in Python which includes various types of Naïve Bayes classifiers like Gaussian Naïve Bayes classifier, Multinomial Naïve Bayes classifier, Complement Naïve Bayes classifier, Bernoulli Naïve Bayes classifier, Categorical Naïve Bayes classifier to name a few. To use different Naïve Bayes classifiers means that the probability distribution function of the Naïve Bayes classifiers will change for each type. Despite of these naïve and basic ideal world assumptions made by these classifiers, they have performed very well in the real world especially in the applications of spam filtering and document classification problems. Naïve Bayes are fast learners and requires only a limited quantity of training dataset to understand and adjust their required parameters from the features. Therefore, I have chosen the Naïve Bayes Classifier as my model for this project. They perform quite efficiently compared to other intricate machine learning algorithms. In this project, I have used Gaussian Naïve Bayes Classifier and Multinomial Naïve Bayes Classifier as my main models to classify the emails into two class labels, Spam and Ham.

Gaussian Naïve Bayes Classifier is a simple Naïve Bayes classifier which uses a probabilistic approach. It computes prior and posterior probability of the class labels and the test data given in the dataset. It is called Gaussian, simply because it calculates the probabilities using a Gaussian distribution, also known as normal distribution. It considers the real values which is corresponds with each value in the feature vectors to be in normal distribution. The maximum likelihood of these features is defined to be Gaussian. The formula for Gaussian distribution is given in Eq. 6 in Fig 8.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) - \text{Eq. } 6$$

where the parameters σ_y and μ_y are estimated using maximum likelihood.

Fig 8. Gaussian distribution

In my project, I have imported Gaussian Naïve Bayes classifier from the sklearn.naive_bayes module. I have a 1-dimensional array of features which we extracted from the tokenized words in the emails in the last phase. This array was transformed from the sparse matrix that the Tf-Idf vectorizer returns. I use the features extracted from the Tf-Idf vectorizer to train the model. Since Naïve Bayes classifiers are supervised learning algorithms, I feed the Gaussian Naïve Bayes model with these extracted features from the emails of the training dataset along with the class labels they belong to using the function fit(). It uses these real numbered features from the Tf-Idf vectorizer to perform the Bayes' theorem as explained above. It applies a normal or gaussian distribution to obtain the conditional probability of these features which is used to set the necessary parameters in the model. After the model is trained, it is now ready to make predictions. This model is able to classify emails if the emails are spam emails or ham emails. To start with the prediction and classification of spam emails in the testing dataset, I use the function predict() of this model. This function inputs the testing features returned by the Tf-Idf vectorizer, after transforming the tokenized words in the testing dataset. These extracted features which were converted to a 1-dimensional array from the sparse matrix returned by the vectorizer, is passed on to the predict function of the model to start the prediction of the emails in the testing dataset. This function returns an array of predicted class labels which classifies the emails as ham with the class label '0' and spam with the class label '1'.

Multinomial Naïve Bayes Classifier is one of the popular Naïve Bayes classifier used in classification of documents and texts. It uses a Multinomial distribution to calculate the conditional probabilities of the features in the vector. Hence, this classifier is called Multinomial Naïve Bayes classifier. The formula for the multinomial distribution is given in Eq 7 in Fig 9.

Vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y,
where n = size of the vocabulary and,
θ_{yi} = probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood:

$$\hat{\theta}_{y\text{i}} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \quad - Eq. 7$$

where $N_{yi} = \sum_{x \in T} x_i$ = number of times feature i appears in class y and,
$N_y = \sum_{i=1}^{n} N_{yi}$ = count of all features for class y.

For $\alpha \geq 0$, features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

Fig 9. Multinomial Distribution

In this project, Multinomial Naïve Bayes classifier was imported from the sklearn.naive_bayes module. S we recall, the last phase returned a 1-dimensional array of features which we extracted from the tokenized words in the emails. This array was converted from the sparse matrix that the count vectorizer returns. I use these extracted features from the count vectorizer for the algorithm to learn. Since Naïve Bayes classifiers are supervised learning algorithms, I provide the Multinomial Naïve Bayes model with these features from the emails of the training dataset along with their corresponding class labels to the function fit(). It utilizes these word counts from the count vectorizer to perform the Bayes' theorem as explained earlier in Fig 8. It uses a multinomial distribution to obtain the conditional probability of these features which is used to set the required parameters in the algorithm. After the training is complete, the model is capable of making predictions to distinguish emails if the emails are spam emails or ham emails. The prediction and classification process is performed by using the predict() function of this model. Predict function is passed with the test features that the count vectorizer returned and it was then transformed to a 1-dimensional array from the sparse matrix. The predict function uses this trained model to output an array of predicted class labels corresponding to the features belonging to the email in the test features.

This step completes the underlying objective of this project. However, note that the model is yet to prove that it is accurate. To check the performance of the model, various evaluation metrics were used in this project, and is discussed in detail in the Evaluation section.

III. Evaluation

Evaluation is the final step in the project pipeline. This phase is mainly concerned about understanding the performance of the project. It uses various metrics to measure the performance and analyze different aspects of the model. Evaluating these metrics, helps identify the shortcomings in a model and bridge the gap between these shortcomings to your intended goal.

There are various metrics available to evaluate a model in the Python libraries like accuracy, confusion matrix, F1 score, Log loss, Gini coefficient, ROC curve, Precision and Recall, Sensitivity and Specificity, Root mean square error to name few. These metrics provide a different perspective to your model and help identify the highs and lows this model has to provide. In this project, the evaluation metrics used are accuracy of the model, confusion matrix, precision, recall and F1-score on the outputs returned from the two models namely Gaussian Naïve Bayes classifier and Multinomial Naïve Bayes classifier to evaluate their performance and show the comparison of their results.

Accuracy of a model is a simple metrics which returns a percentage of accurately predicted classes. It is a ratio of the results that were predicted correct by the model to the total number of results as shown in Eq 8.

$$Accuracy = \frac{No.of\ correct\ predictions}{Total\ no.of\ predictions} \quad - Eq.8$$

The Naïve Bayes models in sklearn.naive_bayes module provide a score() function to calculate the accuracy of that model. The test accuracy comparison of the trained models used in this project are shown in Fig 10.

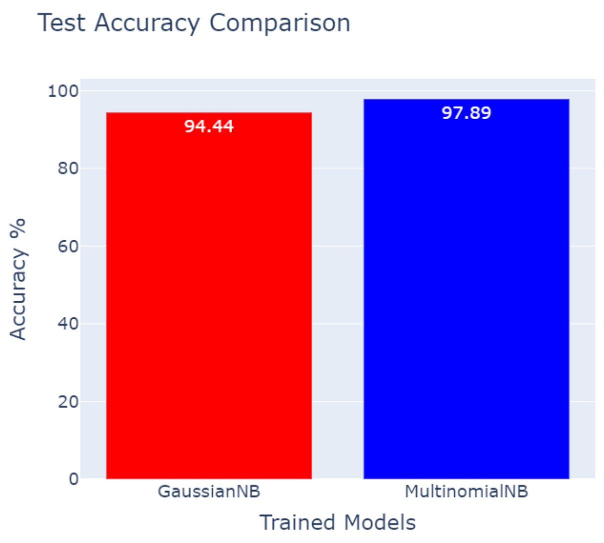


Figure 10. Accuracy comparison

Confusion Matrix is a matrix of size 2 × 2 since there are two class labels used in the model. To understand confusion matrix, let us first understand the necessary terms:

- **True Positive (or TP):** True positives are the results that are correctly predicted and labeled both as positives.

- **True Negative (or TN):** True negatives are the results that are correctly predicted and labeled both as negatives.

- **False Positive (or FP):** False positives are the results that are falsely predicted as positive when the label is negative.

- **False Negative (or FN):** False negatives are the results that are falsely predicted as negative when the label is positive.

Standard	Predicted Ham	Predicted Spam	Total
Label Ham	TP	FN	TP + FN
Label Spam	FP	TN	FP + TN
Total	TP + FP	FN + TN	TP + FN + FP + TN

Fig 11. Standard Confusion matrix

These four values make up the 2 × 2 confusion matrix as shown in the Fig 11, are used to get a better insight of the predictions and evaluate how many values are falsely classified and of what type. In this case, there is a difference in importance of the two class labels. The project aims to minimize spam emails from the ham emails, since ham emails are the. Hence the positives are ham emails which is labeled as 0, and negatives are spam emails which is labeled as 1. Confusion matrix can be imported from the sklearn.metrics module in Python. The confusion matrix for the two models used in this project are shown in Fig 12.

GuassianNB	Predicted Ham	Predicted Spam	Total
Label Ham	986	25	1011
Label Spam	46	221	267
Total	1032	246	1278

MultinomialNB	Predicted Ham	Predicted Spam	Total
Label Ham	1007	4	1011
Label Spam	23	244	267
Total	1030	248	1278

Fig 12. Confusion matrices for Gaussian NB and Multinomial NB

Precision and recall metrics identify the drawbacks of accuracy. Even though our target is to increase the accuracy of the model, they can be misleading at times. In the cases similar to this project, when one class has higher quantity than the other and it only predicts the class with higher quantity correctly. This results in a high accuracy even though the model is incorrectly predicting the lower quantity class. In this case, since the ham emails are higher in quantity than the spam emails, it could be possible that the model only predicts ham emails and wrongly predicts spam emails as ham. This is would still return a high accuracy. However, it would have not solved the purpose of the project.

In these situations, precision and recall come in handy. These metrics are used to calculate the ratio of true positives and negatives, and checks for the falsely predicted values as well. The formulas for precision and recall are given in Eq. 9 and 10.

$$Precision \;=\; \frac{TP}{TP \,+\, FP} \qquad -\; Eq.9$$

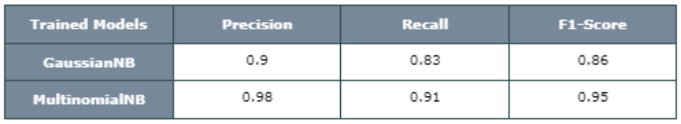
$$Recall \;=\; \frac{TP}{TP \,+\, FN} \qquad -\; Eq.10$$

Higher the values of precision and recall, better is the performance of the code. Precision requires the program to have low false positives which means that the precision is high when the positive predictions of the model has higher percentage of correct values. Recall requires the program to have low false negatives which means that the recall is high when the model has predicted the higher percentage of actual positives. Precision and recall can be imported from the sklearn.metrics module in Python. This module provides a function, precision_score and recall_score which is used to calculate the precision and recall score of the predicted values.

F1-score, in a classification problem is said to be a harmonic mean of the values mentioned previously. It uses both precision and recall to calculate a score.

$$F1 - Score \;=\; 2 \cdot \frac{Precision \,\cdot\, Recall}{Precision \,+\, Recall} \quad -\; Eq.11$$

F1-score are sometimes called the balanced F-measure since it gives a balanced score of precision and recall. It uses a harmonic mean over arithmetic because harmonic mean is skewed more by the outlierss than the arithmetic mean. The F1-score formula is given in Eq 11. The best case in Harmonic mean results in the F1-score. Hence, higher the F1-score, better the performance of the model.



Trained Models	Precision	Recall	F1-Score
GaussianNB	0.9	0.83	0.86
MultinomialNB	0.98	0.91	0.95

Fig 13. Summary of Precision, Recall and F1-score of the trained models

From Fig 13, you can see that the Naïve Bayes classifier have a good overall performance in email spam filtering. Among the two model that I trained, it is evident that the Multinomial Naïve Bayes classifier with count vectorizer has a better performance than the Gaussian Naïve Bayes classifier with the Tf-Idf vectorizer. Multinomial Naïve Bayes had an almost perfect precision of 0.98 along with a high recall score of 0.91 and a F1-score of 0.95, the best case being 1as well.

IV. RELATED WORK

Email spam filtering is a popular problem which is being tackled by the big tech companies who are an email service provider, like Yahoo, Google with Gmail and Microsoft with Outlook. There has been some work done in this field by these major players. This problem can be solved used using various techniques other than Naïve Bayes classifiers such as Neural networks and Support vector machines (SVM). SVMs provide a similar result in comparison to the Naïve Bayes classifiers, however, take longer to train and perform. Deep Learning using neural networks is a popular approach which has recently helped improve the performances of various applications. They have been known to improve the email spam filter where these major email service providers utilize these neural networks in their email spam filters and have been researching further in this problem. There are various approaches used to improve the performance of a simple email spam filtering such as Content based filtering technique, Case base spam filtering method, Rule based spam filtering technique, Previous likeness-based spam filtering technique. Adaptive spam filtering technique which are popular among the popular email service providers.

Rule based spam filtering, is one of the approaches used by Google in Gmail. In this approach, it uses the already existing rules to analyze large quantity of patterns using regular expressions for the selected email. Adaptive spam filtering technique is also a popular technique which identifies the spam emails and groups them in different categories.

There are many approaches proposed for the email spam filtering problem which have been successfully implemented, and yet they always have more scope to improve.

V. CONCLUSION

Naïve Bayes classifier is a high performing group of classifiers for text classification and document analysis problems. Through this project, it is evident that even with a naïve ideal-world approach, Naïve Bayes classifiers provide a tremendous result. Among the Naïve Bayes classifiers, it is shown that the Multinomial Naïve Bayes classifier used with the count vectorizer text feature extraction technique provides a high performance with lower overhead in comparison with Gaussian Naïve Bayes classifiers paired up with Tf-Idf text feature extraction technique giving an accuracy of 97.89% along with an F1-score of 0.95.

REFERENCES

[1] Spamassassin.apache.org. 2020. Index Of /Old/Publiccorpus. [online] Available at: <https://spamassassin.apache.org/old/publiccorpus/>.

[2] Medium. 2020. How to Design A Spam Filtering System. Available at: <https://towrdsdatascience.com/email-spam-detection-1-2-b0e06a5c072>.

[3] Dada, E., Bassi, J., Chiroma, H., Abdulhamid, S., Adetunmbi, A. and Ajibuwa, O., 2019. Machine learning for email spam filtering: review, approaches and open research problems. Heliyon, 5(6), p.e01802.

[4] Pooja, Komal Kumar Bhatia, “Spam Detection using Naive Bayes Classifier,” International Journal of Computer Sciences and Engineering, Vol.6, Issue.7, pp.712-716, 2018.

[5] Rusland, N., Wahid, N., Kasim, S. and Hafit, H., 2017. Analysis of Naïve Bayes Algorithm for Email Spam Filtering across Multiple Datasets. IOP Conference Series: Materials Science and Engineering, 226, p.012091.