MCO3: Machine Learning Models Report

Submitted By:

Go, Daphne Janelyn

Lejano, Enrique Rafael

Manlises, Maria Monica

Wong, Krizchelle Danielle

Date Submitted:

April 17, 2023

I. Introduction

In recent years, the Philippines has experienced a rise in the number of spam messages being sent to mobile phone users. In 2021, the telecommunication company Globe had to block 1.1 billion messages sent to its clients. That number doubled to 2.72 billion in 2022 (ABS-CBN News, 2023). This is a huge problem for both companies and clients. For one, the sheer number of messages that a user receives from these spam messages is incredibly annoying and can clutter up one's device. Moreover, these messages are usually a scam to trick people into giving sensitive information to the scammers, and susceptible Filipinos have lost millions of dollars due to this phenomenon (Cordemo, 2022). Thus, developing effective ways to flag spam messages is important for the safety of users.

To tackle this problem, the group created a machine learning pipeline that identifies whether or not a message is spam. As this is a natural language processing problem, natural language processing techniques such as pre-processing the dataset is necessary prior to feeding the data into the machine learning model, especially given the nature of the data—spam messages tend to be noisy. Two models are then trained and tested on this data; namely, Multinomial Naive Bayes and Logistic Regression. For a comprehensive comparison of the two models that were developed, multiple evaluation metrics were used.

The development of this machine learning pipeline to detect spam messages would be helpful for both clients and companies. Users will be able to filter their own messages, and telecommunication companies would be able to block spam messages before it even gets to their clients. With the reduction of spam messages, it will also be easier for users to identify legitimate messages, which will make them less likely to be victims of scams.

II. The Dataset

'SMS Spam Collection Dataset' is a corpus taken from kaggle.com that contains text message data. The messages are classified into two classes: either spam or ham. Spam messages are the unnecessary and unsolicited messages sent via text message, and ham messages are all messages that do not fall into the spam category.

Upon scanning the corpus, there are two major features in the input data that may determine the label of each data. First, common words such as 'free' or 'claim' tend to appear in spam messages, as the messages tend to be offering some sort of product to the receiver. Second, the messages seem to have more numerical values than the ham messages. These values often represent some monetary value used to encourage the user to reply or click on some link in the message. Each message in the data set is labeled as either spam or ham according to the presence of these features, or the lack thereof.

The corpus contains 5772 documents in total, 87% of these documents are labeled as ham messages and the other 13% are spam messages. Thus, there is a huge imbalance in the data, and this may be an issue in terms of both training and testing the model. When the model is trained on a data set with a huge imbalance, it will tend to be biased towards whichever is the majority class. In this case, the model will be biased towards the majority data set, and this may mean that predictions for the minority class will be more susceptible to error. The group has addressed these concerns to the best of their ability in order to make decisions for the methodology that are informed and justified.

III. Methodology

Pre-processing Methods

For each document in the set, cleaning and noise reduction of the data was done to make it more structured. Furthermore, all the words were made to be lowercase to reduce the sparsity of the extracted features. The spam messages tend to have plenty of variety in casing, wherein some words are all uppercase, some are all lowercase, and some have a mix of uppercase and lowercase. All these words have the same meaning but are treated differently due to the casing, and this consequently creates a sparse set of features, with many features having low or zero frequency counts. The lowercasing of the words is not explicitly shown in the source code as the feature extractor used already defaults to lowercasing all the letters.

Another pre-processing technique that was considered was to remove the stop words, which are frequently or words that do not add significant meaning to the text. The stop words, in this context, are pre-defined by the vectorizer that was used. Ultimately, the group decided to keep the stop words in because it negatively affected the performance of the model. This is further discussed in the results and analysis section.

Prior to feature extraction, the data set was split into training and testing data using scikit-learn's train_test_split class. 30% of the data is for testing and 70% of it is for training. Due to the imbalance in the data set labels, the stratify parameter was used to ensure that the proportion of the labels for the training and testing data is similar to the one in the original data set. This was done so as not to further imbalance the data.

After the splitting of the data, features were then extracted from the training data. The group used the Bag-of-Words model, which is a model that extracts all the words in each document and simply counts its frequency in the document to create a term-document matrix (Great Learning Team, 2022). For this model, the location of the words in relation to other words does not matter, only the frequency. This model was implemented using the CountVectorizer class from Python's scikit-learn library.

The TF-IDF model, which is similar to the Bag-of-Words model but assigns importance to certain words through weights, was also tested and implemented through the TfidfVectorizer class. The group first hypothesized that this model would perform better than CountVectorizer due to the weights it adds. However, CountVectorizer turned out to perform better due to the documents only containing short texts and plenty of unique words (Kaplan, 2022). Thus, CountVectorizer was the final feature extraction method that the group used for the machine learning pipeline.

Since each document only contains short text, the feature extraction uses a unigram to reduce the risk of overfitting the test data to the model. Moreover, as discussed earlier, one of the primary features of the data set is the presence or lack thereof of words such as 'free' or 'claim'. Complex relationships between these words do not need to be determined in order to properly label a message as ham or spam. Thus, a unigram would suffice for this classification task.

After all the necessary pre-processing steps were conducted, the group then trained and tested two different algorithms on the data: Logistic Regression and Multinomial Naive Bayes.

Logistic Regression

Logistic Regression is a discriminative and supervised machine learning algorithm that predicts the relationship between a categorical dependent variable and one or more independent variables. Unlike Linear Regression which is used for determining a value for a continuous variable, Logistic Regression is used for categorizing data. The model has three different types: binomial, multinomial, and ordinal. Binomial Logistic Regression is used when the dependent variable only has two possible outcomes. Multinomial Logistic Regression is used when the dependent variable has three or more possible outcomes. Lastly, Ordinal Logistic Regression is used when the outcomes have an order such as rating scales (IBM, n.d).

For spam detection, Binomial Logistic Regression is most appropriate since the task is a binary classification. However, the default implementation of Logistic Regression in scikit-learn supports both binomial and multinomial Logistic Regression. Although this is worth nothing, it does not influence the outcome of the model whatsoever.

Logistic Regression makes use of an S-shaped function called the sigmoid function to perform its predictions. The function takes in an input and outputs a probability that the positive instance will occur. The positive instance, in this case, is that the message is spam. When the probability is greater than 50%, then the input is classified as spam. Otherwise, it is classified as ham.

In order to determine the importance of a specific variable to the prediction, Logistic Regression assigns it a weight or coefficient. The most optimal set of coefficients is calculated using a statistical tool called the maximum likelihood estimation (MLE) (Addagatla, 2021). The trained model will then use the learned coefficients on the independent variable and use the function to estimate its probability. From the probability, the data is then classified (Sharma, 2018).

Logistic Regression was chosen for this study because it is less likely to overfit in datasets with a low number of features such as text messages. Moreover, its computation is not complex, therefore the model can be trained quickly and efficiently. Although Logistic Regression does not perform well on datasets with complex relationship features, this does not significantly affect the model's performance for the spam classification task as relationships between features are not that important (Grover, n.d).

Multinomial Naive Bayes Algorithm

Naive Bayes is a non-discriminative and supervised machine learning algorithm that performs classification tasks as well (IBM, n.d). As the name suggests, the algorithm is heavily based on Bayes' theorem which states that "the conditional probability of an event, based on the occurrence of another event, is equal to the likelihood of the second event given the first event multiplied by the probability of the first event" (Hayes, 2022). To put succinctly, the theorem calculates the likelihood an event will occur based on other conditions. Additionally, Naive Bayes assumes that a feature is independent of the other features, hence why it is said to be naive (IBM, n.d).

Multinomial Naive Bayes is a specific version of Naive Bayes that is primarily used for text classification. It is important to differentiate this from other variants of Naive Bayes as they are used for entirely different purposes.

With any Naive Bayes algorithm, there is a zero probability issue. This refers to an instance wherein a given word or token does not occur for any of the training data for a specific class. This will result in a likelihood of zero for the entire class and will cause computational problems (Jurafsky & Martin, 2023). To address this, scikit-learn's Multinomial Naive Bayes class implements Laplace Smoothing. Laplace Smoothing simply adds one to each frequency count to avoid any zeroes.

After calculation of probabilities for the input data, the model will take whichever class has the higher probability, and that is the label it assigns the data (Sriram, 2022).

Multinomial Naive Bayes was one of the machine learning algorithms used for this classification task because it performs well on sparse data sets. Despite the initial pre-processing methods done to try to reduce sparsity, the term-document matrix is still sparse (Wärnling & Bissmakr, 2017). Moreover, it is even more time and space efficient than logistic regression (IBM, n.d.).

Testing Machine Learning Models

F1 score, accuracy, precision, and recall were all used in assessing the models' performances; however F1 score was deemed to be the best evaluation metric to utilize for the models.

Accuracy measures the total number of correct predictions the model makes, precision measures the number of correct positive predictions there are, and recall measures how many actual positives the model is able to predict (Dauria, 2019). These three evaluation metrics tend to be susceptible to bias, especially since the dataset is heavily imbalanced. Their susceptibility is discussed and explored further in the results and analysis section.

F1 score, the primary evaluation metric the group decided to use, is a combination of the precision and recall scores of a model. F1score takes the harmonic mean of the two metrics. Thus, it is able to handle imbalanced datasets better than the other three metrics (Educative, n.d.).

To further handle the imbalanced dataset, a macro-average was used instead of a micro-average for all evaluation metrics. The micro-average treats every document equally, consequently giving more weight to whichever class is the majority. Macro-average will evaluate the metrics separately for each class before getting the overall average, thus treating both classes equally despite the ratio. When a dataset is balanced, the two will result in the same score for all metrics. In the case of this dataset, if micro-average were to be used, the majority class would dominate the evaluation metric and may lead to inaccurate results. Because the spam class is smaller but more important than the ham class, macro-average is the better option as it gives equal priority to both classes (Knowledge Transfer, 2021).

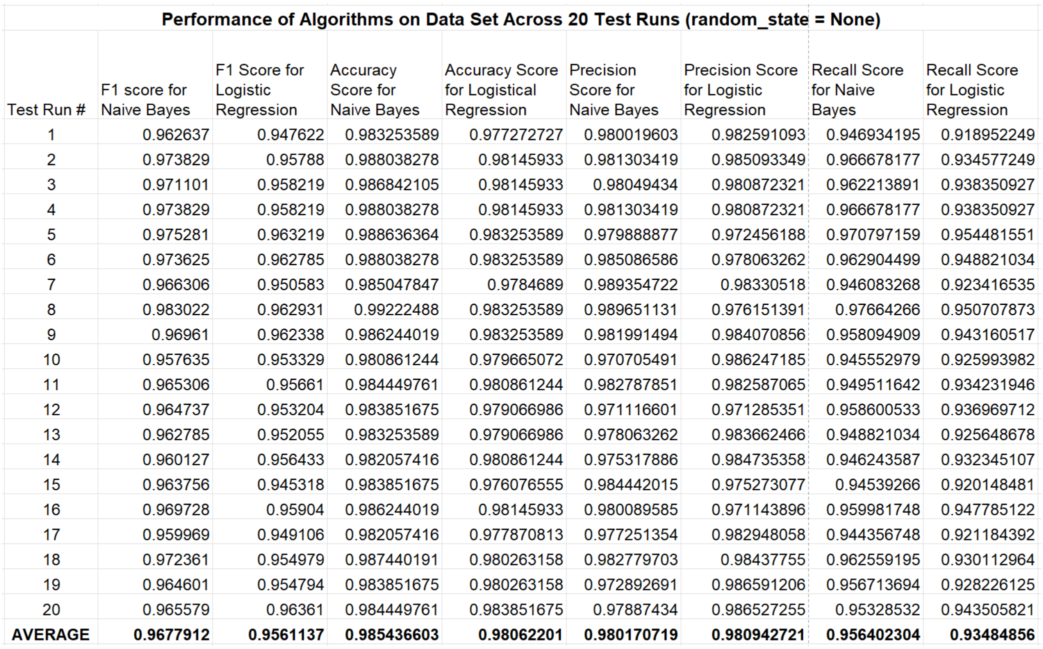
IV. Results and Analysis (Mon & Daph)

Shuffling and Random State Value Assignment

The performance of the two classification models, namely Multinomial Naive Bayes and Logistic Regressions, are evaluated using the four common metrics in machine learning model performance: precision, recall, accuracy, and F1 score. This is crucial in identifying the strengths and weaknesses of the algorithms implemented.

In evaluating the model, shuffling the data set through the **`random_state`** parameter is essential to reduce overall overfitting of the model. Moreover, the learning model is trained and tested on different data subsets by altering the value of the **`random_state`** parameter. This is to make sure that the model is not overfitting to a specific subset of training data. This aids in reducing bias in prediction values.

The **`random_state`** parameter can be assigned in two ways, which include equating the **`random_state`** variable to None or an integer of choice. **`random_state = None`** is the default setting in which each run of the entire program will produce different results with no control over how the shuffling process randomizes the data per execution. Whereas, if **`random_state = <Integer Value>`**, each execution of the program will produce the same results, unless the integer value is modified (Pramoditha, 2022).

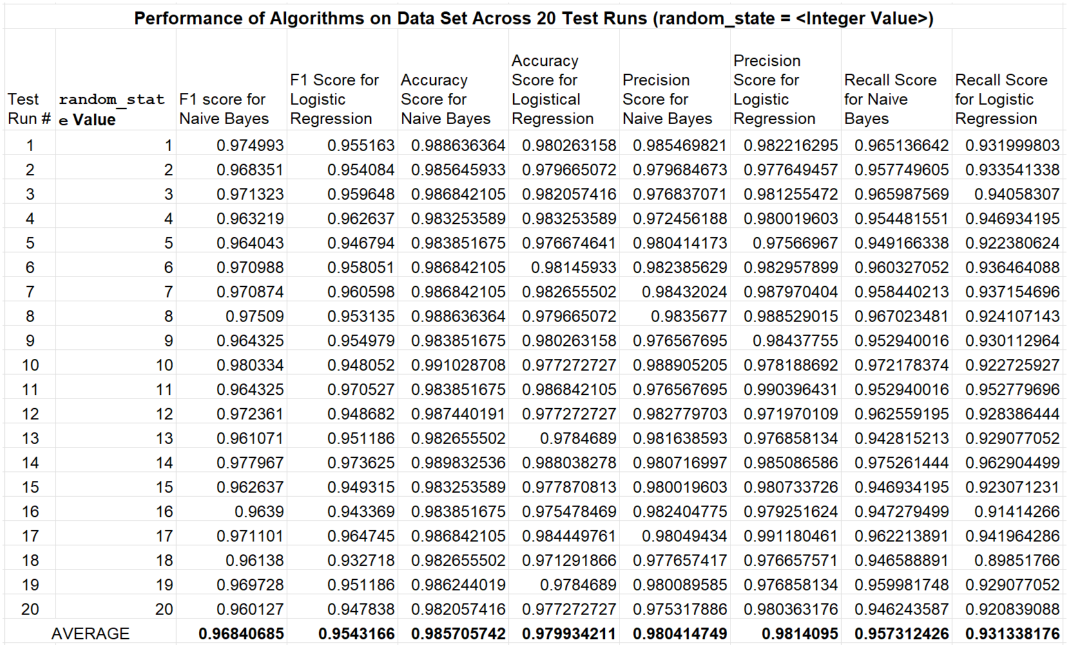


Performance of Algorithms on Data Set Across 20 Test Runs (random_state = None)

Test Run #	F1 score for Naive Bayes	F1 Score for Logistic Regression	Accuracy Score for Naive Bayes	Accuracy Score for Logistical Regression	Precision Score for Naive Bayes	Precision Score for Logistic Regression	Recall Score for Naive Bayes	Recall Score for Logistic Regression
1	0.962637	0.947622	0.983253589	0.977272727	0.980019603	0.982591093	0.946934195	0.918952249
2	0.973829	0.95788	0.988038278	0.98145933	0.981303419	0.985093349	0.966678177	0.934577249
3	0.971101	0.958219	0.986842105	0.98145933	0.98049434	0.980872321	0.962213891	0.938350927
4	0.973829	0.958219	0.988038278	0.98145933	0.981303419	0.980872321	0.966678177	0.938350927
5	0.975281	0.963219	0.988636364	0.983253589	0.979888877	0.972456188	0.970797159	0.954481551
6	0.973625	0.962785	0.988038278	0.983253589	0.985086586	0.978063262	0.962904499	0.948821034
7	0.966306	0.950583	0.985047847	0.9784689	0.989354722	0.98330518	0.946083268	0.923416535
8	0.983022	0.962931	0.99222488	0.983253589	0.989651131	0.976151391	0.97664266	0.950707873
9	0.96961	0.962338	0.986244019	0.983253589	0.981991494	0.984070856	0.958094909	0.943160517
10	0.957635	0.953329	0.980861244	0.979665072	0.970705491	0.986247185	0.945552979	0.925993982
11	0.965306	0.95661	0.984449761	0.980861244	0.982787851	0.982587065	0.949511642	0.934231946
12	0.964737	0.953204	0.983851675	0.979066986	0.971116601	0.971285351	0.958600533	0.936969712
13	0.962785	0.952055	0.983253589	0.979066986	0.978063262	0.983662466	0.948821034	0.925648678
14	0.960127	0.956433	0.982057416	0.980861244	0.975317886	0.984735358	0.946243587	0.932345107
15	0.963756	0.945318	0.983851675	0.976076555	0.984442015	0.975273077	0.94539266	0.920148481
16	0.969728	0.95904	0.986244019	0.98145933	0.980089585	0.971143896	0.959981748	0.947785122
17	0.959969	0.949106	0.982057416	0.977870813	0.977251354	0.982948058	0.944356748	0.921184392
18	0.972361	0.954979	0.987440191	0.980263158	0.982779703	0.98437755	0.962559195	0.930112964
19	0.964601	0.954794	0.983851675	0.980263158	0.972892691	0.986591206	0.956713694	0.928226125
20	0.965579	0.96361	0.984449761	0.983851675	0.97887434	0.986527255	0.95328532	0.943505821
AVERAGE	**0.9677912**	**0.9561137**	**0.985436603**	**0.98062201**	**0.980170719**	**0.980942721**	**0.956402304**	**0.93484856**

Figure 1. Consolidation of Algorithm Performance Across 4 Metrics with Random State Value Set to None

Note. Figure 1 details the performance metric scores of the model across 20 different runs with its random state value set to None. The creation of different subsets of training and testing data through the random state variable allows the fine-tuning of the learning model (Pramoditha, 2022). However, in terms of evaluating performance metrics, after multiple executions, there may be cases that two test cases may have the same set of training and test data, resulting in duplicate performance metrics which may impact the averaged metric score.



Performance of Algorithms on Data Set Across 20 Test Runs (random_state = <Integer Value>)

Test Run #	`random_state` **Value**	F1 score for Naive Bayes	F1 Score for Logistic Regression	Accuracy Score for Naive Bayes	Accuracy Score for Logistical Regression	Precision Score for Naive Bayes	Precision Score for Logistic Regression	Recall Score for Naive Bayes	Recall Score for Logistic Regression
1	1	0.974993	0.955163	0.988636364	0.980263158	0.985469821	0.982216295	0.965136642	0.931999803
2	2	0.968351	0.954084	0.985645933	0.979665072	0.979684673	0.977649457	0.957749605	0.933541338
3	3	0.971323	0.959648	0.986842105	0.982057416	0.976837071	0.981255472	0.965987569	0.94058307
4	4	0.963219	0.962637	0.983253589	0.983253589	0.972456188	0.980019603	0.954481551	0.946934195
5	5	0.964043	0.946794	0.983851675	0.976674641	0.980414173	0.97566967	0.949166338	0.922380624
6	6	0.970988	0.958051	0.986842105	0.98145933	0.982385629	0.982957899	0.960327052	0.936464088
7	7	0.970874	0.960598	0.986842105	0.982655502	0.98432024	0.987970404	0.958440213	0.937154696
8	8	0.97509	0.953135	0.988636364	0.979665072	0.9835677	0.988529015	0.967023481	0.924107143
9	9	0.964325	0.954979	0.983851675	0.980263158	0.976567695	0.98437755	0.952940016	0.930112964
10	10	0.980334	0.948052	0.991028708	0.977272727	0.988905205	0.978188692	0.972178374	0.922725927
11	11	0.964325	0.970527	0.983851675	0.986842105	0.976567695	0.990396431	0.952940016	0.952779696
12	12	0.972361	0.948682	0.987440191	0.977272727	0.982779703	0.971970109	0.962559195	0.928386444
13	13	0.961071	0.951186	0.982655502	0.9784689	0.981638593	0.976858134	0.942815213	0.929077052
14	14	0.977967	0.973625	0.989832536	0.988038278	0.980716997	0.985086586	0.975261444	0.962904499
15	15	0.962637	0.949315	0.983253589	0.977870813	0.980019603	0.980733726	0.946934195	0.923071231
16	16	0.9639	0.943369	0.983851675	0.975478469	0.982404775	0.979251624	0.947279499	0.91414266
17	17	0.971101	0.964745	0.986842105	0.984449761	0.98049434	0.991180461	0.962213891	0.941964286
18	18	0.96138	0.932718	0.982655502	0.971291866	0.977657417	0.976657571	0.946588891	0.89851766
19	19	0.969728	0.951186	0.986244019	0.9784689	0.980089585	0.976858134	0.959981748	0.929077052
20	20	0.960127	0.947838	0.982057416	0.977272727	0.975317886	0.980363176	0.946243587	0.920839088
AVERAGE		**0.96840685**	**0.9543166**	**0.985705742**	**0.979934211**	**0.980414749**	**0.9814095**	**0.957312426**	**0.931338176**

Figure 2. Consolidation of Algorithm Performance Across 4 Metrics with Random State Value Set to an Integer Value

Note. Figure 2 details the performance metrics of the learning model with the random state values set to integer values in the range of 1 to 20. Setting a specific value to the random state parameter is done for consistency and reproducibility (Pramoditha, 2022). All metric scores can be reproduced by simply assigning the random_state to its corresponding value in the table through this function statement: **`x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 20, stratify=y)`**.

Accuracy, Precision, and Recall as Performance Metrics of Proposed Learning Model

As shown in Figures 1 and 2, the scores for the performance metrics are within the range of 0.93 to 0.98 which was tested by repeatedly running the model using the two methods of **`random_state`** parameter assignment. With this, it may be possible to infer that due to the high metric score obtained throughout the tests done, the model is performing well as it is able to make relatively accurate predictions. However, it is important to note that the use of any of the three performance metrics – accuracy, precision, and recall – as the sole performance metric for the proposed learning model is not recommended.

As mentioned earlier, these metrics may be susceptible to bias, especially for imbalanced data sets. Consider an instance of a test data set with 90 samples classified for one class with the remaining 10 samples for the other class and a training data set with the same ratio. Even though the model will label that every document is of the first class, the model will still obtain a 90% accuracy score (Kanstrén, 2020). Based on the main data set used for this project, the number of messages classified as spam amounts to only 747 in contrast to ham messages amounting to 4825. This imbalance seen in the data set may therefore result in high susceptibility towards a bias for ham messages in prediction values generated by the model.

F1 Score as Primary Performance Metric of Proposed Learning Model

F1 score, also known as F-score and F-measure, as a quantitative metric is calculated as a harmonic mean of precision and recall. The use of this performance metric addresses the issue of imbalance and overfitting. As mentioned above, in cases where the data set utilized has imbalanced classes in which one class is classified more than the other, a model that is able to predict the dominant class will acquire a high accuracy performance metric despite its inability to classify the other class correctly. This implies that it does not take into consideration the performance of each individual classification. Meanwhile, the F1 score also accounts for precision and recall being two metrics that come at the cost of one another. Increasing the proposed model's threshold value will produce more stringent predictions, which improves precision. Whereas, decreasing the threshold value will imply more leniency in model predictions, allowing for better recall (Rohit, 2023). Hence, the F1 score combines both precision and recall, allowing for a more balanced evaluation of imbalanced data sets.

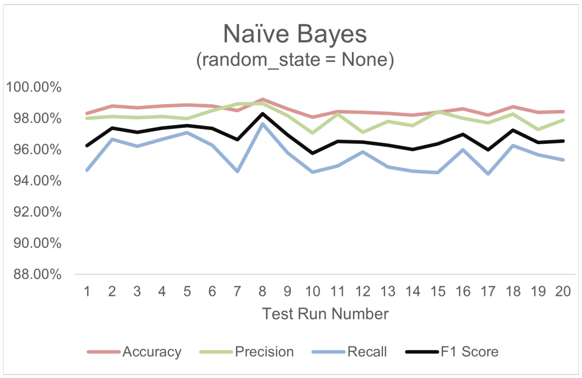


Figure 3. Variability of Performance Metrics by Multinomial Naive Bayes on Data Set with Random State Value Set to None

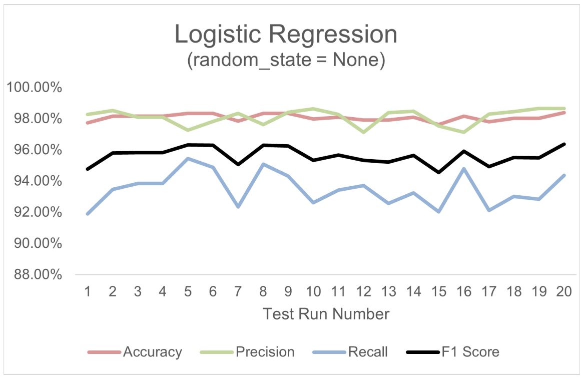


Figure 4. Variability of Performance Metrics by Logistic Regression on Data Set with Random State Value Set to None

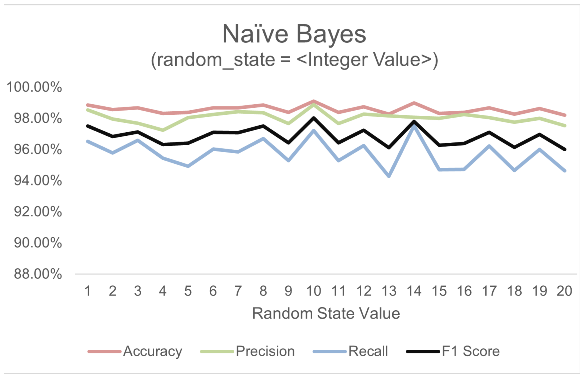


Figure 5. Variability of Performance Metrics by Multinomial Naive Bayes on Data Set with Integer Random State Value

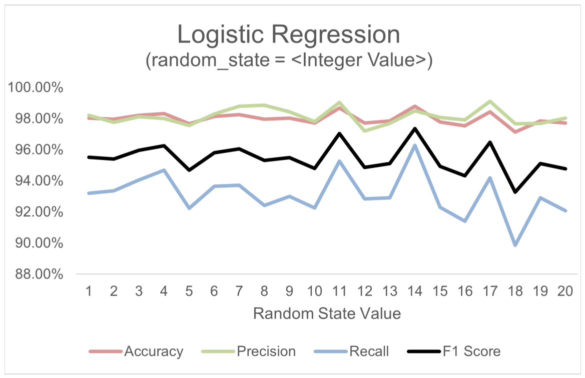


Figure 6. Variability of Performance Metrics by Logistic Regression on Data Set with Integer Random State Value

In Figures 3, 4, 5, and 6, the trend of the accuracy performance metric on the two models, Multinomial Naive Bayes and Logistic Regression, is relatively the highest among the four performance metrics and is always much higher compared to the trend of the recall metric. This may be indicative of the bias in prediction values. The recall performance metric tends to be low as the imbalance causes the models to classify messages as "ham" more often than "spam" even if they are actual spam messages. Thus, accuracy alone may not be indicative of the actual performances of the models. In relation to this, trendlines of F1 scores are situated between the precision and recall trendlines, which in turn may be an indication of a more balanced metric that takes into account both precision and recall metrics in its evaluation. As such, the F1 score will serve as the basis of the analysis of the performance of the algorithms in this paper.

Performance Evaluation of the Algorithms

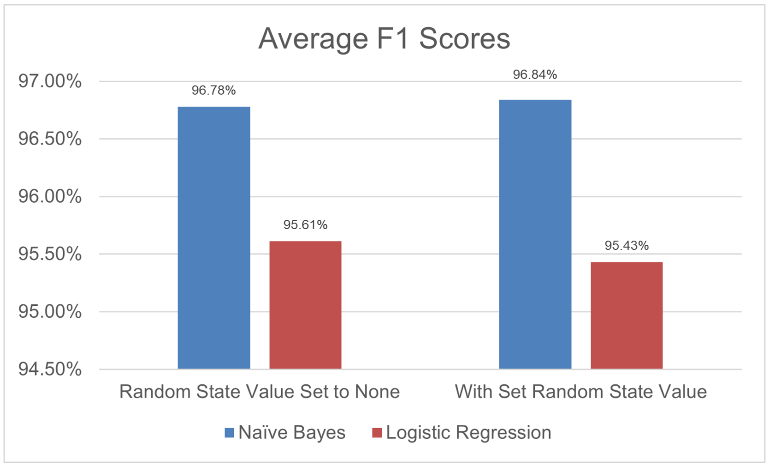


Figure 7. Comparison of F1 Scores of Multinomial Naive Bayes and Logistic Regression Algorithms with Different Random State Value Assignments

The mean F1 score of the two classification models across the 20 test runs was calculated to provide a balanced and objective evaluation. Based on Figure 7, high mean F1 scores are observed for both Multinomial Naive Bayes and Logistic Regression. With a performance metric score ranging from 95 to 98%, as seen in Figures 1 and 2, the obtained F1 score is indicative of how both models are able to classify spam messages accurately while minimizing false positives and negatives for spam classification subsequently.

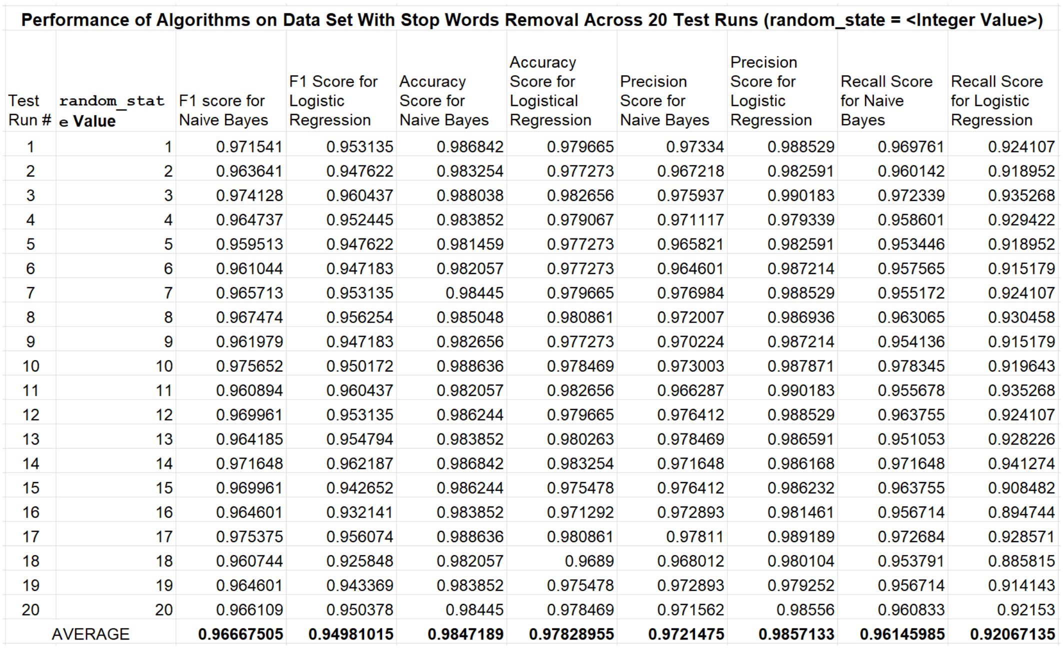
Comparative Analysis of the Results of the Algorithms

As observed in Figure 7, in the context of this particular data set and feature classification, Multinomial Naive Bayes is better-performing than Logistic Regression in classifying spam messages. The average F1 score achieved by Multinomial Naive Bayes surpassed that of Logistic Regression by around 1%. It is important to note that there is no baseline as to whether this percentage difference between the two classification models is significant. However, further analysis may be done to ascertain the reason behind the performance of the algorithms.

The Multinomial Naive Bayes Algorithm's performance for this data set can be attributed to its capability to assume independence among features and to handle sparse data. The use of the Bag-of-Words model often creates feature representations that are high-dimensional and sparse, where the majority of the features have a value of zero (Yse, 2021). This can be handled well by Multinomial Naive Bayes, which can also effectively model the joint probability distribution of the features. To perform at its best, however, Logistic Regression may need extra regularization or feature selection methods as the algorithm may find it difficult to deal with high dimensional sparse data (Li et al., 2015). As a consequence, the sparsity of the feature representation and Multinomial Naive Bayes' applicability to the data set may have led to its better performance over Logistic Regression in this context.

Stop Words Removal as an Additional Pre-processing Method

Removing stop words as an additional pre-processing step in text vectorization was considered for this implementation, however, its effectivity could not be theoretically determined by the group (Khanna, 2021). As such, another set of tests was performed on the data set adding this pre-processing method. However, since the importance of manually setting the random state value had already been established, tests were no longer performed with the random state value set to None. The results of this can be seen in Figure 8 and were also summarized in Figures 9-10.



Performance of Algorithms on Data Set With Stop Words Removal Across 20 Test Runs (random_state = <Integer Value>)									
Test Run #	random_state Value	F1 score for Naive Bayes	F1 Score for Logistic Regression	Accuracy Score for Naive Bayes	Accuracy Score for Logistical Regression	Precision Score for Naive Bayes	Precision Score for Logistic Regression	Recall Score for Naive Bayes	Recall Score for Logistic Regression
1	1	0.971541	0.953135	0.986842	0.979665	0.97334	0.988529	0.969761	0.924107
2	2	0.963641	0.947622	0.983254	0.977273	0.967218	0.982591	0.960142	0.918952
3	3	0.974128	0.960437	0.988038	0.982656	0.975937	0.990183	0.972339	0.935268
4	4	0.964737	0.952445	0.983852	0.979067	0.971117	0.979339	0.958601	0.929422
5	5	0.959513	0.947622	0.981459	0.977273	0.965821	0.982591	0.953446	0.918952
6	6	0.961044	0.947183	0.982057	0.977273	0.964601	0.987214	0.957565	0.915179
7	7	0.965713	0.953135	0.98445	0.979665	0.976984	0.988529	0.955172	0.924107
8	8	0.967474	0.956254	0.985048	0.980861	0.972007	0.986936	0.963065	0.930458
9	9	0.961979	0.947183	0.982656	0.977273	0.970224	0.987214	0.954136	0.915179
10	10	0.975652	0.950172	0.988636	0.978469	0.973003	0.987871	0.978345	0.919643
11	11	0.960894	0.960437	0.982057	0.982656	0.966287	0.990183	0.955678	0.935268
12	12	0.969961	0.953135	0.986244	0.979665	0.976412	0.988529	0.963755	0.924107
13	13	0.964185	0.954794	0.983852	0.980263	0.978469	0.986591	0.951053	0.928226
14	14	0.971648	0.962187	0.986842	0.983254	0.971648	0.986168	0.971648	0.941274
15	15	0.969961	0.942652	0.986244	0.975478	0.976412	0.986232	0.963755	0.908482
16	16	0.964601	0.932141	0.983852	0.971292	0.972893	0.981461	0.956714	0.894744
17	17	0.975375	0.956074	0.988636	0.980861	0.97811	0.989189	0.972684	0.928571
18	18	0.960744	0.925848	0.982057	0.9689	0.968012	0.980104	0.953791	0.885815
19	19	0.964601	0.943369	0.983852	0.975478	0.972893	0.979252	0.956714	0.914143
20	20	0.966109	0.950378	0.98445	0.978469	0.971562	0.98556	0.960833	0.92153
AVERAGE		**0.96667505**	**0.94981015**	**0.9847189**	**0.97828955**	**0.9721475**	**0.9857133**	**0.96145985**	**0.92067135**

Figure 8. Consolidation of Algorithm Performance Across 4 Metrics on Data Set With Stop Words Removal and with Random State Set to Integer Values

Note. Figure 8 details the performance metrics of the learning model after preprocessing the data using text vectorization integrating stop words removal as hyperparameters and setting the random state to integer values in the range of 1 to 20. Setting a specific value to the random state parameter is done for consistency and reproducibility. All metric scores can be reproduced by simply assigning the random_state to its corresponding value in the table through this function statement: **`x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 20, stratify=y)`**.

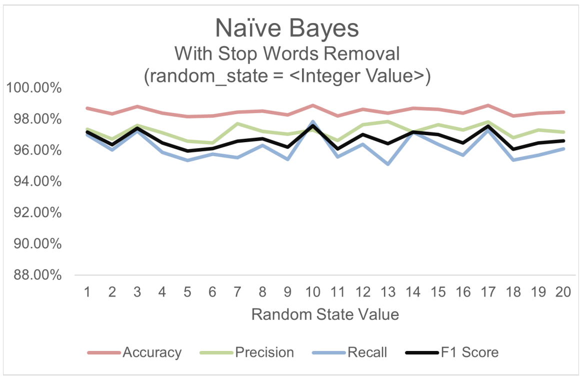


Figure 9. Performance of Multinomial Naive Bayes Across 4 Metrics on Data Set with Integer Random State Value

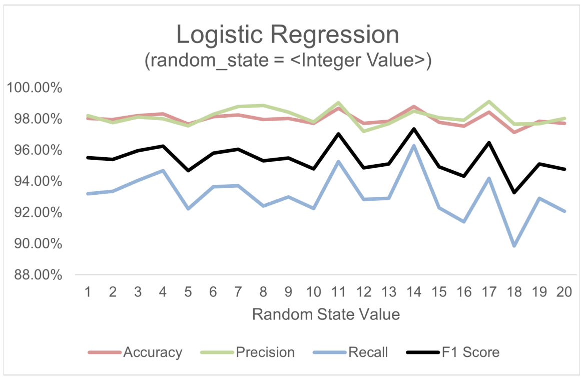


Figure 10. Performance of Logistic Regression Across 4 Metrics on Data Set with Integer Random State Value

It can be observed in Figure 11 that removing stop words led to lower average F1 scores for both algorithms. This means that the pre-processing step may not be appropriate for this particular data set. The slight lowering of performance when stop words were removed may be attributed to the loss of crucial features that may have been helpful in text classification. For example, one spam message in the data set involves the phrase "Please call our customer service representative", and since "call" is part of the stop words, its removal in the entire data set may have affected the model as typically spam messages may involve calling or texting certain numbers (Khanna, 2021).

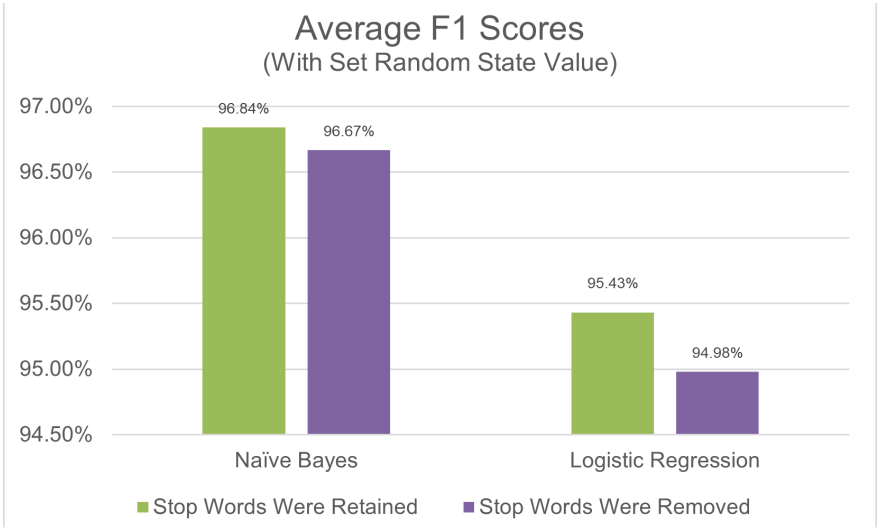


Figure 11. Comparison of Algorithm Performance with and without Stop Word Removal Pre-processing

V. Conclusions and Recommendations

Summary

With the prevalence of spam messages in the recent decade, this project proposes a learning classification model that aims to be able to classify spam and ham messages. Pre-processing of the acquired data set through data cleansing and text vectorization was done to ensure that the data is formatted in a manner that can be processed easily by the machine learning models. These methods have a significant impact on the overall performance of the learning models.

Two classification algorithms, namely, Multinomial Naive Bayes and Logistic Regression, were utilized to train models through the spam and ham data set. The performance of the models needs to be evaluated to ascertain their credibility as classification models.

With this, the performances were evaluated using four performance metrics: accuracy, precision, recall, and F1 score. However, due to the imbalance seen in the data set with 87% labeled as ham messages and the other 13% as spam messages, accuracy, precision, and recall were deemed inappropriate metrics due to their high susceptibility towards a bias for ham messages in prediction values generated by the model. With this in mind, the F1 score was used as the main metric for performance evaluation due to its relative objectiveness despite the imbalance in class proportions.

Both models achieved high F1 scores at a range of 95 to 98%, which is indicative of how both models are able to classify spam and ham messages accurately. Moreover, Multinomial Naive Bayes was observed to be performing better than Logistic Regression with a 1 to 3% difference. This was attributed to the sparsity of the feature representation and Multinomial Naive Bayes' applicability to the data set compared to Logistic Regression. Overall, the proposed models are able to predict relatively accurately which messages are spam versus ham. To further studies in this context, larger-scale data may be further utilized to evaluate its effectiveness.

Recommendations

For the project's further improvement, using k-fold cross-validation is recommended to produce a more comprehensive performance metric analysis of the proposed model. K-fold cross-validation involves dividing the data set into k subsets, wherein one fold may be used as test data and the remaining k-1 folds as training data. This is then done in k-iteration, in which each fold becomes the testing set at least once and is subsequently averaged across all runs to obtain the needed performance metric. This in turn addresses the issue of the model overfitting to a selected data set and maximizes the use of available data (Pandian, 2022).

More so, in this implementation, testing of the data was manually performed and visualized using external software (i.e., Microsoft Excel and Google Sheets). However, for reproducibility and consistency, integration of the testing and visualization into the notebook using the sci-kit library Scikit-Plot may be considered to shorten the testing duration and process and reduce inconsistencies and errors that may be encountered while testing (Solanki, 2022).

Additionally, the imbalance among model features was a prominent concern throughout the proposed implementation. Hence, mitigating class imbalance using techniques in downsampling and oversampling may be considered to address the problem. Synthetic Minority Over-sampling Technique (SMOTE) or Adaptive Synthetic Sampling (ADASYN) are some techniques for oversampling the feature with fewer data which may improve recall of the model. Furthermore, downsampling the feature class with more data through techniques such as Tomek links or NearMiss may also improve the precision and recall metric consequently (Brownlee, 2020).

Lastly, to contextualize the data set better, other natural language processing techniques may also be applied. Word embeddings may be used as vector representations that are able to capture semantic features and relationships. Models such as Word2Vec and GloVe may be considered for word embeddings (Dhruvil Karani, 2018).

VI. References

ABS-CBN News. (2023, January 24). *Globe says 2.72B spam, scams SMS blocked in 2022*. https://news.abs-cbn.com/business/01/24/23/globe-says-272b-spam-scam-sms-blocked-in-2022

Addgatla, A. (2021, April 26). *Maximum likelihood estimation in logistic regression.* Medium, https://arunaddagatla.medium.com/maximum-likelihood-estimation-in-logistic-regression-f86ff1627b67

Brownlee, J. (2020, January 14). *Random Oversampling and Undersampling for Imbalanced Classification*. Machine Learning Mastery. https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification

Cordemo, T. (2022, October 4). *Filipinos lost 'millions of dollars' due to text scams - CICC.* GMA News. https://www.gmanetwork.com/news/topstories/nation/846936/filipinos-lost-millions-of-dollars-due-to-text-scams-cicc/story/

Dauria, E. (2019, December 9). *Accuracy, recall, & precision.* Medium. https://medium.com/@erika.dauria/accuracy-recall-precision-

Dhruvil Karani. (2018, September). *Introduction to Word Embedding and Word2Vec*. Medium; Towards Data Science. https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

Educative. (n.d.). *What is the F1-score?*. https://www.educative.io/answers/what-is-the-f1-score

Great Learning Team. (2022, October 24). *An introduction to a Bag of Words (BoW) | What is Bag of Words?*. https://www.mygreatlearning.com/blog/bag-of-words/

Grover, K. (n.d). *Advantages and disadvantages of Logistic Regression.* Open Genus. https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/

Hayes, A. (2022, March 1). *Bayes' Theorem: What it is, the formula, and examples*. Investopedia. https://www.investopedia.com/terms/b/bayes-theorem.asp

IBM. (n.d.). *What are Naïve Bayes classifiers?*. https://www.ibm.com/topics/naive-bayes

IBM. (n.d.). *What is logistic regression?*. https://www.ibm.com/topics/logistic-regression

Jurafsky, D. & Martin J. (2023, January 7). *Naive Bayes and Sentiment Classification*. Speech and Language Processing. https://web.stanford.edu/~jurafsky/slp3/4.pdf

Kaplan, D. (2022, October 6). *Machine learning 101: CountVectorizer vs TFIDFVectorizer.* Enjoy Machine Learning. https://enjoymachinelearning.com/blog/countvectorizer-vs-tfidfvectorizer/

Kanstrén, T. (2020, October 31). *A Look at Precision, Recall, and F1-Score*. Medium. https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec

Khanna, C. (2021, February 10). *Text pre-processing: Stop words removal using different libraries*. Medium. https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a

Knowledge Transfer. (2021, August 29), *Micro and Macro Averages for imbalance multiclass classification.* https://androidkt.com/micro-macro-averages-for-imbalance-multiclass-classification/

Li, X., Ling, C. X., & Wang, H. (2015, November 1). *The Convergence Behavior of Naive Bayes on Large Sparse Datasets*. IEEE Xplore. https://doi.org/10.1109/ICDM.2015.53

Pandian, S. (2022, February 17). *K-Fold Cross Validation Technique and its Essentials*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/

Pramoditha, R. (2022, May 2). *Why do we set a random state in machine learning models?* Medium. https://towardsdatascience.com/why-do-we-set-a-random-state-in-machine-learning-models-bb2dc68d8431

Rohit, K. (2023). *F1 Score in Machine Learning: Intro & Calculation*. Www.v7labs.com. https://www.v7labs.com/blog/f1-score-guide

Sharma, N. (2018, May 5). *Spam detection with Logistic Regression.* Medium. https://towardsdatascience.com/spam-detection-with-logistic-regression-23e3709e522

Solanki, S. (2022). *Scikit-Plot: Visualize ML Model Performance Evaluation Metrics by Sunny Solanki*. Coderzcolumn.com. https://coderzcolumn.com/tutorials/machine-learning/scikit-plot-visualizing-machine-learning-algorithm-results-and-performance

Sriram (2022, October 3). *Multinomial Naive Bayes explained: function, advantages & disadvantages, applications in 2023.* upGrad. https://www.upgrad.com/blog/multinomial-naive-bayes-explained/

Wärnling, O. & Bissmark, J. (2017, June 5). *The effect of sparse data on the Naïve Bayes algorithm*. https://kth.diva-portal.org/smash/get/diva2:1111045/FULLTEXT01.pdf

Yse, D. L. (2021, July 3). *NLP | Text Vectorization*. Medium. https://lopezyse.medium.com/nlp-text-vectorization-e472a3a9983a

I. Contributions of Each Member

Go, Daphne Janelyn L.	Testing of the Program (60 test runs) Paper – Results and Analysis Paper – Conclusions and Recommendations Paper – References
Lejano, Enrique Rafael A.	Coding and Testing of the Program Paper – Introduction Paper – The Data Set Paper – Methodology
Manlises, Maria Monica	Paper – Results and Analysis Paper – Conclusions and Recommendations Paper – Figures Creation and Formatting
Wong, Krizchelle Danielle A.	Coding and Testing of the Program Paper – Introduction Paper – The Data Set Paper – Methodology Paper – References