A Review of ML Spam Classification Algorithms

Michael Weiner
December 15, 2021

Abstract

Modern technology has made email a common method to share information between one or more parties. Any email falls into one of two categories: ham or spam. Ham emails contain genuine information that an individual should take the time to review. Spam emails are any unsolicited emails that are received by an individual that should not be viewed as they are of no significant value. The continued prevalence of spam emails in 2021 requires that every email service provider have a fast and accurate spam filtering system. Incorrect classifications of emails as spam or ham decreases user satisfaction, decreases productivity, and increases the computational resources and energy needed to correct mistakes. This paper examined the spam filtering performance of four machine learning (ML) classification algorithms: a Naive Bayes (NB), a Support Vector Classification (SVC), a Multilayer Perceptron (MLP), and a Decision Tree (DT). Each ML algorithm was analyzed based on its average performance over three trials on three different training-to-test dataset ratios. Every ML algorithm was given the same randomized training and test data on each trial for each dataset ratio to ensure each algorithm had access to the same training data. On average, the NB classifier and MLP were found to be $\geq 97\%$ accurate over all three dataset ratios, with the MLP taking substantially longer to train. The experiment's results can be used to determine a cost-effective, accurate spam filter for small datasets and the results can be extrapolated for datasets of substantially larger size.

Contents

Importance of Accurate Spam Detection

Unfortunately, spam emails continue to be a problem in the 21st century. According to Kaspersky's Q3 2021 Global Spam and Phishing Report, 45% of all emails sent globally in Q3 2021 were identified as being spam [13]. With the continued pervasiveness of this issue, spam detection filters for email services have had to continue to adapt and remain proficient at correctly identifying spam emails. The inability to correctly identify spam emails could lead to a significant decline in productivity for email users across the globe. It could also spell significant cybersecurity events on global infrastructure based on the threats that are often contained in spam emails.

As technology has advanced, basic spam detection has become a combination of filtering techniques that are applied in a sequence to parts of an email's content or header. A common technique within every spam detection tool is some form of machine learning. This could be a Naive Bayes Classifier or a form of neural network to examine the textual content of an email to classify it as spam or ham.

Although modern tools correctly identify and categorize over 99% of emails as being spam or not, spam emails are still able to bypass spam detection filers and some filters still incorrectly mark genuine email as spam [11]. The need to continue to adapt and refine existing spam detection tools remains incredibly important for all email users.

This project will compare several algorithms that can be used to detect spam in a textual format. Since text can be found in Google search results, social media posts, news articles, email, and more, understanding the results from this project can be used to determine which algorithm - or series of algorithms layered together - would potentially be the most effective to detect spam for different mediums of content that are viewed by people around the world every day.

A Review of Relevant Work

Modern, efficient, machine learning (ML) spam detection algorithms begin with some form of pre-processing of the textual content found within an email [1]. This pre-processing of text content is a component of Natural Language Processing (NLP). NLP is required for modern email communication as emails can be both large in length and contain a variety of sentence and word combinations. Research from the Technical University of Denmark has shown that if the raw text content of an email can be stripped of non-discriminative words, machine learning classifiers cannot only be more efficient but can also be more accurate in their classifications [6]. A handful of common non-discriminative words in the English language include: I, the, and, of, or. Other non-discriminative symbols include punctuation marks and numbers.

The team's research showed that, by removing these common occurrences of words in the English language that do not help to discriminate the text of genuine emails from that of spam emails, classifiers could be more accurate by up to 25% as compared to the classification of the same data that contained the non-discriminative content. Not only could the spam classifiers be more accurate, but they could be more efficient in making their classification as the number of words that would need to be parsed was

reduced.

Spam Detection using Bayes' Theorem

Spam Detection algorithms that use Bayes' Theorem to classify a portion of text as either being spam or ham are called Bayesian Classifiers. Bayes' Theorem is defined as [11]:

$$P(b|a) = \frac{P(a|b) \cdot P(b)}{P(a)} \tag{1}$$

A Naive Bayesian (NB) algorithm is a common ML algorithm used to classify emails as spam or ham. The NB algorithm conducts NLP on a given set of emails that have already been classified as spam or ham - usually called the training data. It then builds a knowledge base of the discriminative words that were present in the spam and ham emails and their respective probabilities that the word was found in spam emails and in ham emails. This knowledge is then used to help classify new emails that are passed into the NB classifier. This makes a NB classifier a relatively simple, effective spam filter.

A pitfall of the standard NB classifier is that it assumes that every word in a piece of text is unrelated to the words around it [5]. The rules of English grammar are in direct contradiction to this.

Many experiments have been run on standard NB classifiers. Research at Xi'an Jiaotong University has found that NB spam filtering with a 80:20 testing-to-training data ratio is about 92% accurate in classifying spam emails as being spam [16]. The relatively simple requirements of this algorithm combined with its ability to enhance its knowledgebase's accuracy as more emails are processed and filtered make it a strong candidate to be the starting point for any spam detection tool.

Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras have conducted research on various forms of NB classifiers that have shown the capabilities of Bayes' Theorem for use in multiple forms of text classification [8]. Comparisons were run on the different types of information that a NB classifier could use to build its knowledgebase. The team compared tracking the word count for each discriminative word found in the text content (to calculate probabilities of each word appearing), a binary flag tracking whether a word was present, a combination of both the frequency and binary flag, and a Bayesian classifier that allowed for multiple normal distributions of several categories (words). After running the different NB classifiers on several of the open-source Enron datasets, it was concluded that the traditional Multinomial NB classifier that tracks the probabilities of each word occurring in spam and ham emails provided the highest average recall of *both* spam and ham at 97% recall.

Karl-Michael Schneider has demonstrated actual and theoretical improvements that can be made to the standard NB classifier [12]. The improvements include enhancing the NB algorithm to gain insight into the grouping of words, accounting for words that have never seen before, and how to avoid removing non-discriminative words

from shorter textual content where the non-discriminative words may be critical in classifying the text content as spam or ham.

Spam Detection using a Neural Network

Using neural networks (NN) to do text classification for spam email filtering has been well researched as neural networks provide a stronger ability to work with higher dimensionality textual data, as compared to a NB classifier.

Research at Xi'an Jiaotong University examined using a feed-forward neural network with a single hidden layer [16]. This particular neural network uses weights for all words that have been found. For the words found in the text of the current email being evaluated, the weight is either multiplied by 1 (if the word is present) or 0 otherwise. Summing the weights of the words present in the text, the NN will return a value for that email between 0 and 1. The closer the email is to 1 the more likely that email is spam. It was found that this NN performed relatively well but was rather sensitive to the ratio of the training-to-test data it was given.

Additional research with Multilayer Perceptrons (MLPs) has supported the conclusion that artificial NNs have high degrees of accuracy in the classification of text emails as being spam or ham [14]. Although V.Christina was able to demonstrate the MLP's high accuracy in spam filtering ($\geq 99\%$), it required 138 seconds to train its model on a given dataset. The same dataset took only 0.15 seconds for a Naive Bayes' classifier to train on. As more hidden layers are added to the NN, the relative simplicity begins to fade for a relatively small increase in email filtering accuracy.

Although MLP NNs become increasingly complex as more hidden layers are added, this addition allows them to perform better on larger emails where the words that distinguish spam emails from ham emails may become limited [4]. Additional hidden layers of perceptrons in parallel begin to allow for more complex mathematical computation on larger textual input that has a higher dimensionality. This begins to separate the abilities of a relatively simple Naive Bayesian Classifier from that of a MLP NN.

Performance Evaluation

Upon review of numerous studies and evaluations of Naive Bayes' Classifiers and Neural Network Classifiers, there is a sweeping consensus on how different classifiers can be evaluated.

One form of evaluation is to compare the time it takes different solutions to train the model with the same ratio of training-to-test data. Consider the real-world implications of how long it takes to classify an email as spam or ham. A large email service provider, such as Google, might process hundreds of thousands of emails a minute. Imagine the differences between being able to classify each email being sent as spam or ham in 10 seconds as compared to 100 seconds. The longer it takes to classify emails, the longer a customer must wait to receive their email, more compute power is needed, more electricity is needed to power the servers running spam detection algorithms, more machines are needed to assist in classification, more staff are needed

to maintain said servers, etc. Classifying a relatively small number of emails as spam or ham is rather insignificant. However, the implications of this problem, *at scale*, carries tremendous economic implications.

Another common form of evaluation is to analyze the effectiveness of the spam filtering by evaluating the number of (1) true negatives (emails that are classified as ham that are ham), (2) false negatives (emails that are classified as ham that are actually spam), (3) true positives (emails that are classified as spam that are actually spam), and (4) false positives (emails that are classified as spam that are actually ham) [3]. Intuitively, this makes sense. A spam filter not only needs to filter emails into two categories, but it needs to minimize incorrect classifications.

Recall is a metric used to measure the accuracy of correctly filtering emails as being in one of the categories (spam or ham). Precision is another metric that can be used to determine what percentage of the emails that were marked as being of a certain category actually belonged to that category [2].

It is important to note that not all machine learning algorithms are an equal apples-to-apples comparison. Different ML algorithms have multiple ways in which they can be fine-tuned for better performance on a training and a test data set [15]. Since different algorithms have different types of optimizations available through different hyperparameters, any experiment comparing different ML algorithms will need to explicitly state the values of the hyperparameters being used for each algorithm. By explicitly stating these hyperparameter values, it can be demonstrated that the particular ML algorithm used does not have a single solution but almost an infinite number of possible solutions with different optimizations. One way that a comparison can be made relatively equivalent is by training and testing each algorithm on the same datasets. This will be important in establishing a baseline set of results to analyze.

Approaching the Problem of Spam Email

Computers cannot interpret written language the same way that humans can. Machines are incredibly literal and can detect even the smallest of differences between words in a textual format. To be able to effectively and correctly classify emails, the words that make up the content of an email need to precisely represent the contents of the email in a way that a machine can process it.

Selecting a Dataset

Since email spam remains so prevalent, there are numerous open-source datasets that contain a large number of emails to use as training and test data for different classification algorithms [13].

The experiments in this project use a version of the Enron email dataset that was collected during the United States' investigation into the Enron corporation [8]. This modified Enron dataset took six Enron employees that had the largest email mailboxes (by comparison of file size) and broken each employee's mailbox into its own dataset

(Enron1 - Enron6) with each dataset representing one of the six Enron employees. These datasets contained a portion of the Enron user's genuine emails that they sent to other people around the world. The only modifications made to *every* email in the datasets were:

1. Removal of emails containing viruses.
2. Removal of all email attachments.
3. Removal of all HTML tags from the email's content.
4. Removal of all non-latin encodable characters.
5. Removal of any duplicate emails to guarantee uniqueness.

Each Enron dataset (Enron1 - Enron6) then had a set of spam emails (with the same preprocessing applied) added to the set containing all of the emails for that employee. Each of the six Enron datasets had a different source of spam emails. Most interestingly, the Enron1 dataset had spam emails that were received by an individual *without* the use of any spam filtering to collect a real sample of spam emails moving through day-to-day email traffic.

The use of this particular open-source dataset allows for easy replication of the experiment defined below and prevents inconsistent initial pre-processing of the original dataset for future experimentations.

Processing Email Content from the Chosen Dataset

To be able to accurately determine the classification capabilities of different algorithms, insignificant differences and details need to be removed or unified throughout every piece of textual data. This preprocessing of textual data can include:

1. **Case Unification** - When the body of an email is being read by a computer, it would view the occurrences of `The`, `the`, `THE`, etc. as different words. Obviously, in the English language these are not different words. They are the same word. By making every character in an email lowercase or every character uppercase this incorrect and unnecessary distinction is avoided.
2. **Removal of Uncommon Characters** - Special characters such as various punctuation can be removed uniformly from a set of text data as punctuation is common in many, if not all, categories of text.
3. **Removal of Stop Words** - Every language has its own common words and phrases - these are generally referred to as stop words. In the English language these include words like: I, and, but, for, of, the, he, she, etc. These words don't provide any meaningful capability for a machine to distinguish a set of text from one category or another. The removal of stop words helps to ensure that an algorithm is only analyzing the most helpful words for a classification.
4. **Lemmatization of Words** - Many languages have multiple versions of the same word. The English language is no exception. As an example, `cat` and `cats` are two different words that refer to the same object. One is a singular reference,

and one is a plural reference. Another example is `has` and `had`. Each refers to the same verb, but in a different tense. Since a machine would view each of these words as being different, lemmatization works to take each word in a piece of textual data and reduce it to its root word [7]. This is a common form of natural language processing (NLP) that helps to reduce the number of words an algorithm needs to analyze.

This list is not exhaustive, but these are the common forms of preprocessing that can take place on all elements in a textual dataset before a set of training and test data are created and features are extracted.

It should also be noted that these forms of NLP preprocessing have both benefits and drawbacks. For example, lemmatization may potentially increase the effectiveness of an algorithm but the time it takes to lemmatize large pieces of textual data may be too large of a tradeoff for large scale text classification. In many instances, determining which combination of preprocessing to apply to a dataset starts with understanding attributes of the data itself.

Creating a Set of Training and Testing Data

Using the selected Enron dataset, the Enron1 data will be separated into two distinct sets [8]. One set will be comprised of a random selection of ham and spam emails used to train different classification algorithms for comparison. The second distinct set of data, the test dataset, will be a set of the remaining spam and ham emails that were **not** part of the training data. After each algorithm has been trained with the training dataset, the test dataset will be run through each of the different classification algorithms to compare how accurately emails that have never been encountered previously are classified.

It is important to note that in each trial of an experiment, each algorithm should be trained on the same set of data and each algorithm should be tested on the same set of test data. This ensures that each algorithm is given the same set of information during its training period and that the test data has not been encountered during the training of each algorithm.

Vectorization

Before a set of training and testing data can be passed into any algorithm, the textual data needs to be represented in a mathematical form that a machine can efficiently use to conduct classification.

One of the most common forms of text vectorization is called Count Vectorization. The principle is straightforward. Every unique word in an entire dataset (after preprocessing of the dataset is complete) is put into an array called the vocabulary. Each piece of text data (each email) in the dataset is then converted into an equivalent representation in the form of a vector of the same length as the vocabulary vector. Each index of the new vector stores the how many times the word that corresponds to that index in the vocabulary was found in the email.

Consider an example. Let's says that a dataset contains two sentences: "The quick brown fox." and "The fox is nice." The only preprocessing that will be done on this dataset is ensuring that every word has the same case (lowercase in this example). The vocabulary vector would be represented as ["the", "quick", "brown", "fox", "is", "nice"]. Table 1 tracks how many times each of the unique words in the vocabulary vector appears in each sentence that makes up the original dataset.

	the	quick	brown	fox	is	nice	.
the quick brown fox.	1	1	1	1	0	0	1
the fox is nice.	1	0	0	1	1	1	1

Table 1: Example count vectorization of two sentences.

Based on the results in Table 1, an equivalent mathematical representation of the sentence "The quick brown fox." is: `[1,1,1,1,0,0,1]` and an equivalent mathematical representation of the sentence "The fox is nice." is: `[1,0,0,1,1,1,1]`. These vectors would then be passed into an algorithm as part of a training dataset or a test dataset. This is how an algorithm would determine which words appear in a piece of text and would ultimately be used to determine which category a piece of text data should be categorized as. Potential limitations start to appear with a count vectorization as the word order is ignored in the vector representation and meanings of sentences don't necessarily coordinate with frequencies of individual words. The space complexity required by this approach can also become significant as the number of unique words in a set of text data increases.

A count vectorization is just one method of creating an equivalent mathematical representation of text data. Other forms of vectorization include tracking the probability of words appearing in a text, tracking relations and positions to nearby words, and simply tracking whether a word appears or not.

Experiment Design and Results

This experiment was designed to compare the accuracy and effectiveness of various machine learning (ML) algorithms as tools to classify emails as being spam or ham. The ML algorithms being compared include: a Naive Bayes (NB), a Support Vector Classification (SVC), a Multilayer Perceptron (MLP), and a Decision Tree (DT). These algorithms were selected to give broad coverage of the current ML algorithms that might be effective for this type of binary classification.

In addition to the general comparison of the classification results provided by each algorithm, the training-to-test data ratio was changed to determine the effects that a smaller or larger training dataset had on each classification algorithm's efficacy.

Procedure for Comparison

The Enron1 dataset was used throughout this experiment [8]. This dataset contains 5,172 emails, of which 3,672 emails are ham and the remaining 1,500 emails are spam.

The pandas Python library was used to read each ham and spam email from files stored on a local machine into a DataFrame [9]. Each row in the respective DataFrames were emails that were ready to be preprocessed.

In addition to the preprocessing that is already present in the Enron1 dataset, each email's case was uniformly changed to be lowercase. Stop words were not removed from these emails. The body content of each email was not lemmatized. These decisions were made for two reasons. The emails in the Enron dataset contain words such as `we've`. If the stop words were to be removed from this set of emails, `we've` would become `'ve`. This partial phrase would not be representative of the original email's content. Additionally, lemmatization can introduce a high computational time expense as the length of text data increases. This experiment is more concerned with the baseline performance of the listed ML algorithms. However, it should be noted that the four ML algorithms listed cannot have a true direct comparison. Each ML algorithm has its own specific hyperparameter(s) that can be used to tune the learning of the ML algorithm to improve the efficiency and accuracy of the algorithm given the specific data of the problem.

Once the preprocessing was completed, the DataFrames were broken into a DataFrame containing training data and a DataFrame containing testing data.

Using the scikit-learn open-source Python library, every email in the training and test datasets were transformed into a count vectorization to be passed into the different ML classification algorithms [10]. Scikit-learn was then used to build instances of each classification algorithm. The classification algorithms were then trained with the same training dataset, and then performance metrics were calculated when the test dataset was fit to each of the algorithms. Three different trails were conducted with each trial having a different training-to-test dataset ratio. Each of these three trials were repeated three times to calculate an average of the interested performance metrics.

During each trial, the overall accuracy, precision of spam and ham classification, recall of spam and ham classification, and F1 score were recorded for each algorithm. The time to train each ML classification algorithm was also recorded and averaged.

- Accuracy [0.0-1.0]: The the percentage of classifications that the algorithm made correctly.
- Precision [0.0-1.0]: The percentage of true positive[1] classifications that the algorithm made out of all positive classifications.
- Recall [0.0-1.0]: The percentage of true positives[2] that were correctly classified as true positives by the algorithm.
- F1-Score [0.0-1.0]: A mean of an algorithm's precision and recall scores.

[1] Positive instances could be either spam or ham depending on whether spam precision or ham precision is being measured.

[2] Positive instances could be either spam or ham depending on whether spam recall or ham recall is being measured.

Data

Several different trials were conducted using different training:test ratio data splits. In Trials 1, 2, and 3, an instance of a Multinomial Naive Bayes (NB) Classifier, a Support Vector Classification (SVC), a Multilayer Perceptron (MLP), and a Decision Tree (DT) were compared. The hyperparameters for each ML algorithm were as follows:

- NB: `alpha` = 1.0
- SVC: `C` = 1.0, `kernel` = Radial Basis Function (RBF), `gamma` = $\frac{1}{(\#\text{ of Features})(\text{Variance of Data})}$ `tolerance` = 0.001 (tolerance of decision boundary)
- MLP: `hidden layers` = 1 hidden layer with 100 perceptrons, `activation function` = Rectified Linear Unit (ReLU), `weight optimization` = Adam Stochastic Gradient-Based Optimizer, `tolerance` = 0.0001
- DT: `split decision` = Gini Impurity, `split strategy` = best possible split, `max tree depth` = Unlimited, `min number of splits` = 2

These ML algorithms' hyperparameters were **not** optimized or changed between Trials 1, 2, or 3. These configurations are just one of many different possible combinations of hyperparameters. These hyperparameters enable ML algorithms to be fine-tune their learning capabilities for a wide range of data. These values were selected not to optimize each algorithm but rather as a starting point to compare each algorithm's classification abilities with different amounts of training and test data.

Trial 1 used an 80:20 split of the Enron1 dataset. 80% of the data became training data that was used to train each algorithm. The remaining 20% of the data was then used to test how effectively the algorithm could classify emails that had never been seen previously.

Table 2: Results of Trial 1

Metric \ Algorithm	NB	SVC	MLP	DT
Avg. Time to Train (sec.)	0.011230	3.954668	78.62268	0.424103
Avg. Accuracy	0.980354	0.957165	0.976167	0.946537
Avg. Ham F1-Score	0.986403	0.969794	0.983481	0.962529
Avg. Ham Precision	0.982693	0.983884	0.980928	0.971308
Avg. Ham Recall	0.990145	0.956124	0.986092	0.953925
Avg. Spam F1-Score	0.964589	0.926328	0.957172	0.906702
Avg. Spam Precision	0.974300	0.895257	0.964446	0.886757
Avg. Spam Recall	0.955097	0.959744	0.950320	0.927674

Trial 2 used a 70:30 split of the Enron1 dataset. 70% of the data became training data that was used to train each algorithm. The remaining 30% of the data was then used to test how effectively the algorithm could classify emails that had never been seen previously.

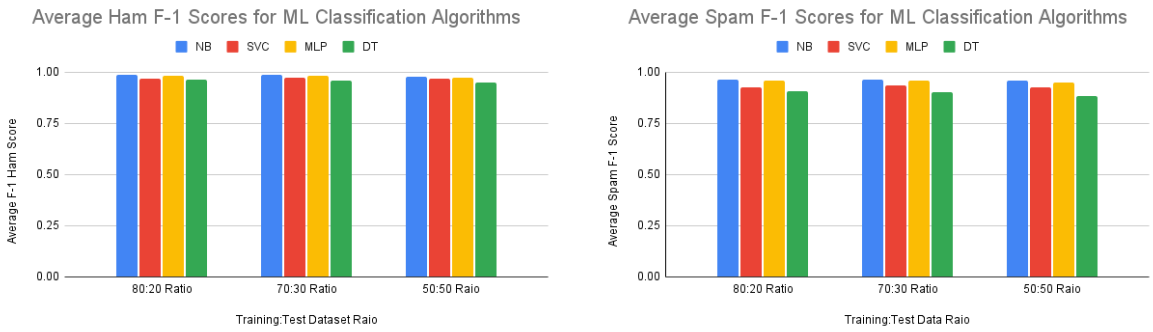
Table 3: Results of Trial 2

Metric \ Algorithm	NB	SVC	MLP	DT
Avg. Time to Train (sec.)	0.009825	3.252516	63.88128	0.440889
Avg. Accuracy	0.978522	0.959621	0.976374	0.943943
Avg. Ham F1-Score	0.984918	0.971088	0.983353	0.960417
Avg. Ham Precision	0.983428	0.988705	0.985723	0.964433
Avg. Ham Recall	0.986414	0.954101	0.981009	0.956469
Avg. Spam F1-Score	0.962666	0.933038	0.959294	0.903798
Avg. Spam Precision	0.966231	0.895913	0.953568	0.895215
Avg. Spam Recall	0.959134	0.973490	0.965184	0.912726

Trial 3 used a 50:50 split of the Enron1 dataset. 50% of the data became training data that was used to train each algorithm. The remaining 50% of the data was then used to test how effectively the algorithm could classify emails that had never been seen previously.

Table 4: Results of Trial 3

Metric \ Algorithm	NB	SVC	MLP	DT
Avg. Time to Train (sec.)	0.006654	1.723817	42.46689	0.224361
Avg. Accuracy	0.975380	0.955400	0.971642	0.930265
Avg. Ham F1-Score	0.982636	0.968343	0.979992	0.950484
Avg. Ham Precision	0.975806	0.967666	0.973812	0.951413
Avg. Ham Recall	0.989563	0.969058	0.986264	0.949813
Avg. Spam F1-Score	0.957705	0.924544	0.951318	0.881946
Avg. Spam Precision	0.974326	0.926335	0.966366	0.881678
Avg. Spam Recall	0.941648	0.922951	0.936808	0.883593



(a) Average F-1 Ham Score

(b) Average F-1 Spam Score

Figure 1: Comparison of the (a) Ham F-1 Scores and the (b) Spam F-1 Scores for the four ML classification algorithms.

Analysis of Results

The data collected in each trial helps to highlight several important ideas.

One of the most obvious correlations drawn from the data is that as the ratio of training data to test data decreased, the average time (in seconds) needed to train each algorithm with the set of training features decreased. This alone is unsurprising. As the amount of data that needs to be analyzed is reduced, every algorithm should be able to process a smaller amount of data faster. Between the 80:20 and 50:50 training:test data splits, each algorithm saw approximately a 40-50% reduction in the time taken to train on the given dataset.

The time needed to train each algorithm in this experiment should not be viewed as an absolute comparison but as a relative comparison. These trails were run on a consumer grade laptop without a dedicated graphics processing unit (GPU). For the size of the dataset being used, this did not present a challenge. However, on a real-world scale the binary classification of emails would be run on hardware dedicated and optimized for ML and Deep Learning. Another clear conclusion from the data presented is that the average time needed to conduct model training on the MLP classifier was several orders higher than any of the other classification algorithms. As the number of hidden layers in an MLP were to grow in number and size, it would not be surprising that learning would take longer. Training any ML algorithm is computationally expensive, but the learning of an entire dataset (in theory) would only have to happen once. Each of the four algorithms described in this research should be able to improve their classification accuracy as more emails are classified. Therefore, the time needed to train an ML algorithm should not be the sole decider of what algorithm would be "best" to classify a set of data.

At each ratio of training to test data, both the average spam and ham F-1 scores for the NB and MLP classifiers performed within 4-7 one-thousandths of each other. The NB classifier consistently edged out the MLP classifier. The SVC followed behind the NB and MLP classification algorithms by 1.5-2 percentage points, on average. The DT classifier came in 1.5-2 percentage points behind the SVC in each trial, on average. The precision and recall scores allow for the F-1 scores to be analyzed further. Results from each trial indicate, on average, each of the four classification algorithms had a higher ham recall than spam recall. Meaning that ham emails were easier to identify and classify correctly than spam emails were. Figure 1(b) demonstrates that this is an indicator of a potential area of optimization that each of the algorithms could benefit from - especially the DT classification algorithm.

The speed and high accuracy of the NB classifier needs context. A review of Bayes' Theorem shows that the math behind a Naive Bayes classifier is relatively trivial. It should not be surprising how quickly a NB classifier can train on a set of emails in each trial. The high accuracy and speed demonstrated by the algorithm might lead individuals to preemptively choose a NB classifier over any other classification algorithm. That could be problematic - depending on the data being classified and the scale at which the classification algorithm is used.

The dataset used in this experiment contains approximately 5,200 emails. Relatively,

that's small. Consider the Naive Bayes classifier that had 0.980354 accuracy with an 80:20 training:test data split and 0.975380 accuracy with the 50:50 training:test data split. That 0.004974 percentage point difference sounds miniscule. Now, assume that this algorithm would be scaled to a production email server classifying 1,000,000 emails a minute. That same 0.004974 percentage point difference *could* equate to 4,947 additional incorrect categorizations each minute. Compounded over an hour, or a day, and that small performance dip could have serious implications. The experiments conducted in this research do not provide any quantitative data to project performance of these algorithms on substantially larger data sets. What the collected data does say is that for relatively small data sets, these algorithms do not exhibit substantial performance decreases (or increases) with larger and smaller training:test data splits.

The method in which textual data is translated into a mathematical representation plays a large role in that computational complexity. In the trails conducted in this research a count vectorization was used. This representation can be space intensive during the learning process depending on the number of unique words in the dataset. The types and accuracy of NLP that occur in the preprocessing stage of data preparation can also contribute to the accuracy (or inaccuracy) of every classification. NLP becomes challenging as every language would potentially need slightly different forms of preprocessing. If every email that would need to be classified would have to go through several stages of NLP, the computational complexity and time required for classification would increase.

As mentioned, these ML algorithms have a variety of hyperparameters that can be used to fine-tune their learning performance. Where a NB and MLP classifier look to have "equal" performance, a MLP has the flexibility and complexity of adding hidden layers and different activation functions where a NB classifier does not. Outside research has already shown that a NB classifier could trivially be beaten by a genuine email that only contains unseen words or words that are all correlated with previous spam emails [12]. Although a MLP classifier takes longer to train on a set of data, it could provide more flexibility with its adjustability to learn on new datasets.

Most importantly the data in each of the three trials does not explicitly rule out any of the four classifiers. For a small dataset, a NB classifier appears to be a very strong choice. A MLP takes longer to train but has just as good efficacy in correctly classifying emails, and it has the opportunity for further optimization through its hyperparameters. The SVC and DT classifiers might need further optimization for this *specific* set of data before they can become competitive with a NB or MLP classifier. On the same hand, both the SVC and DT had over 90% classification accuracy without any specific optimizations made.

Conclusion

This experiment has provided strong insight into a comparison between four of the most commonly used ML classification algorithms for spam email filtering. Classification of textual data is not trivial. A combination of NLP to prepare text data for vectorization, the type and accuracy of vectorization, and optimizations made to each classification algorithm's learning capability are all critical components to

conduct binary classification such as spam filtering. The data collected and presented has shown that the ratio of training to test data passed into an ML algorithm can impact both the accuracy of classification and time needed to train each algorithm. The speed of the algorithms in this experiment can be deceiving given the relatively small size of the dataset used. The data collected does not explicitly rule out any of the four ML classification algorithms for use in spam email classification based on their raw performance. With that being said, the results presented are for the specific set of data used in this experiment. Research should be conducted on how each of the four ML algorithms analyzed perform on a wide range of email formats, lengths, languages, and content.

The challenging job of spam classification, for any high-quality classification algorithm, is the less than 1% of emails that might be incorrectly classified [11]. Spam email continues to be a prevalent issue in the modern era [13]. This fundamental comparison of these algorithms should be expanded in several ways.

Studies should be run on each of these four ML algorithms independently to analyze each hyperparameter that is available. These hyperparameters could demonstrate an ability for all four of these algorithms to have similar efficacy when classifying emails. Additional research should be conducted on the computational space complexity of different types of vectorizations of textual data that is passed into ML algorithms to determine their effect on classification speed.

The preprocessing of textual data is a major component of binary classification. This preprocessing takes time, compute power, and energy. Research should be conducted on how combinations of NLP such as lemmatization, unification of casing, removal of excess spacing and/or punctuation, etc. can impact the classification abilities of an algorithm. It could potentially be determined where the most cost-effective combinations of NLP exist. Studying whether different languages benefit from different combinations of NLP would be important to determine the implications of filtering text between multiple languages.

Modern email providers give users a method to report misclassified emails (false positives or false negatives). This serves as another piece of information for any ML algorithm. Research could be conducted to see how important this set of community feedback is to a classification algorithm's accuracy over time. Email header information such as the physical location of where an email is sent or the domain name that an email is sent from could also be used as an additional layer of information for a ML algorithm to filter on. Research could be done on how these layers of information could be used to improve overall classification accuracy while keeping spam filters cost effective and fast for end users.

Emails are more than just text. They contain attachments in the form of other documents, videos, etc. Those attachments could potentially provide information on the legitimacy of emails, and that should be researched further as well.

By layering efficient and effective NLP preprocessing, vectorization, and optimized classification algorithms, the gap can be closed on the misclassifications of emails.

References

[1] Isra'a AbdulNabi and Qussai Yaseen. Spam email detection using deep learning techniques. In *The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops*, volume 184, pages 853–858, 2021.

[2] Eman M. Bahgat, Sherine Rady, and Walaa Gad. An e-mail filtering approach using classification techniques. In Tarek Gaber, Aboul Ella Hassanien, Nashwa El-Bendary, and Nilanjan Dey, editors, *The 1st International Conference on Advanced Intelligent System and Informatics (AISI2015), November 28-30, 2015, Beni Suef, Egypt*, pages 321–331, Cham, 2016. Springer International Publishing.

[3] Hossam Faris, Ibrahim Aljarah, and Ja'far Alqatawna. Optimizing feedforward neural networks using krill herd algorithm for e-mail spam detection. In *2015 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*, pages 1–5, 2015.

[4] Aurangzeb Khan, Baharum Baharudin, Lam Hong Lee, and Khairullah Khan. A review of machine learning algorithms for text-documents classification. *Journal of Advances in Information Technology*, 1(1):4–20, 2010.

[5] Yuliya Kontsewaya, Evgeniy Antonov, and Alexey Artamonov. Evaluating the effectiveness of machine learning methods for spam detection. In *2020 Annual International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence: Eleventh Annual Meeting of the BICA Society*, volume 190, pages 479–486, 2021.

[6] R.E. Madsen, S. Sigurdsson, L.K. Hansen, and J. Larsen. Pruning the vocabulary for better context recognition. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 483–488, 2004.

[7] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Stemming and lemmatization*. Cambridge University Press, 2008.

[8] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes - which naive bayes? In *Third Conference on Email and Anti-Spam CEAS*, 2006.

[9] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education (US), 4th edition, 2020.

[12] Karl-Michael Schneider. Techniques for improving the performance of naive bayes for text classification. In Alexander Gelbukh, editor, *Computational Linguistics*

and Intelligent Text Processing, pages 682–693, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[13] Tatyana Shcherbakova and Tatyana Kulikova. Spam and phishing in q3 2021, Nov 2021.

[14] V.Christina, S.Karpagavalli, and G.Suganya. Email spam filtering using supervised machine learning techniques. *International Journal on Computer Science and Engineering*, 2, 12 2010.

[15] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[16] Bo Yu and Zong ben Xu. A comparative study for content-based dynamic spam classification using four machine learning algorithms. *Knowledge-Based Systems*, 21(4):355–362, 2008.