A Comparison of Naïve Bayes and Extreme Learning Machine for Spam Detection

Ethan Massingill
Department of Computer Science
University of South Carolina Upstate
Spartanburg, South Carolina, USA
massinge@email.uscupstate.edu

ABSTRACT

The task of detecting spam emails is crucial for maintaining the productivity and security of modern communication systems in an increasing digitally connected world. This paper compares the performance of two popular machine learning algorithms, Naive Bayes and Extreme Learning Machine (ELM), on the task of spam detection through a standard confusion matrix and runtime for performance comparison. In this study, the algorithms are implemented in two different ways. One using Python machine learning library, the other directly in the code. They use a Kaggle spam dataset of 5,500 emails collected from the Spam Assassin Corpus. The accuracy, precision, recall, and specificity of each algorithm are evaluated and compared to each other with the purpose of determining which algorithm is better for creating a model that can if it is spam or not spam. Likewise, each algorithm is timed from the start of its training phase to the conclusion of its testing phase to determine the speed in which it completes the task. The results indicate that Naïve Bayes slightly outperforms Extreme Learning Machine in F1-Score and precision and greatly outperforms ELM in run time. The study's results reflect that with the simplicity of the Naïve Bayes algorithm might be better suited for spam detection with the size of the dataset used and the hardware it is ran on. However, the results obtained in this study would need to be confirmed by a larger dataset for each model. With this in mind the results of this study do reflect previous research but would be more conclusive if preformed on a large dataset.

Keywords

Computer Science, AI, Artificial Intelligence, ML, Machine Learning, Extreme Learning Machine, ELM, Naïve Bayes, NB, Neural Networks, Spam Detection, Computer Security

1. INTRODUCTION

With the world growing more connected ever since the pandemic so to has the need to safeguard user interactions. One of the primary ways to communicate with each other is through email. With the increasing reliance on this communication, the problem of spam has become a major concern for both individuals and organizations. Spam is defined as unwanted and unsolicited messages, including emails, text messages, and social media posts that are sent out in large numbers to recipients. Despite this unwanted communication spam itself makes up over 40% of emails according to [8]. These messages can contain malicious content, such as malware, scams and viruses, or irrelevant information and obscene content that can harm originations if it is not properly filtered. This makes it essential to develop effective detection methods of dealing properly with spam while allowing regular communication to be handled correctly.

Detecting spam through machine learning algorithms is essential handle the large variety of email types and filtering them correctly. These algorithms can be trained on a dataset of labeled messages and can then be used to classify new messages in real-time classifying the message as legitimate or spam. The success of spam detection algorithms is largely dependent on the accuracy of the classifier, which can be measured in terms of various performance metrics such as accuracy, precision, and recall. Likewise setting up the machine learning model itself can factor in with how much time it takes to train the model on datasets as time passes and the filters themselves need to be updated. Naïve Bayes (NB) and Extreme Learning Machine (ELM) are two popular machine learning algorithms for spam detection. While Naïve Bayes has been widely used, ELM has been gaining popularity as both hardware and neural networks become widely adopted. The objective of this research is to compare the performance of Naïve Bayes and ELM in the metrics of accuracy, precision/recall, and speed of training the models.

The results of this comparison will provide insights into the strengths and weaknesses of each model, as well as their suitability for use in the context of spam detection. Usage of these models for research can help push further the understanding of utilizing machine learning within By comparing the performance of these two models this study aims to provide a comprehensive evaluation of each approach and determine which is suitable for practical use with the onset of greater hardware and the need for strong detection models. The results will be valuable for the

research into computer security and machine learning and help advance the development of more effective and efficient methods for spam detection.

2. Literature Review

With the growth of Artificial Intelligence research and application it should come to no surprise that there is research on Naïve Bayes and Extreme Learning Machine models leveraged for spam detection.

Yilmaz Kaya et. Al conducted research to determine how Extreme Learning Machine preformed when utilized for spam detection. In this study ELM was leveraged to determine if an email was spam from the UCI machine-learning repository of 4601 emails. This was conducted on the metrics of accuracy of determining if the email was spam, and speed of that classification. This study showed that ELM showed a higher accuracy and speed than other models. For accuracy it was able to show a 91.66% rate, while classification time of .05 seconds[5]. When compared to Naïve Bayes classification, which had an accuracy of 78.26 and speed of .28 seconds, it showed great improvement. The authors concluded that ELM outperformed the other models on the same datasets[5]. This study provided useful research into how ELM models preform both in accuracy and training time compared to other popular models such as MLP, SVM, NB, J48, and PART. Furthermore it also provided a good look into how the ELM model on spam or not and how fast the training can be compared to similar machine learning models. Some drawbacks of this study could be with the dataset that was used for it. While the dataset was large and using real world messages the data itself is quiet old dating back to 1999. Furthermore, this testing could have been improved if multiple datasets were used to help validate the accuracy of the model on unseen testing data

Zhijie Zhang et. Al utilized a different type of text classification with the Extreme Learning Machine model [4]. In their study they looked at social media, specifically Twitter, to attempt to determine spam based on characteristics including text content similar to emails. With this study however they utilized a unique form of ELM called Fuzzy-Kernal-regularized Extreme Learning Machine (FK-ELM). With FK-ELM the key difference is that it uses a fuzzy kernel function which maps the data into higher-dimensional feature space. This function is a nonlinear function that calculates the similarities between the data points, which allows more flexible regularization process and results in improved generalization performance compared to a traditional ELM [4]. Aponador dataset, which is a public dataset from Brazil, was used along with a dataset harvested from the Twitter4J library which covered 43 million tweets from 2017. The research determined that high performance in the accuracy of detecting the twitter spam, specifically scored almost 10% higher than a traditional ELM [4]. Overall this study was good for exploring what an ELM was able to do against a large spam dataset. It provided a good model and visualization of its results. One of the drawbacks of this study however is that instead of implementing a more tradition ELM, they instead implemented a more specific style that may not translate well to other types of spam such as email despite its strength in text classification.

Jeremy Eberhardt analyzed two different forms of the Naïve Bayes algorithm for spam detection and text classification. They utilized the traditional Naïve Bayes model along with Multinomial Naïve Bayes (M-NB)[6]. With the main difference between traditional Bayes theorem and Multinomial Naive Bayes being the latter makes a simplifying assumption about the independence of features, which allows for a computationally efficient implementation [6]. This was trained on a dataset of 100,000 LinkedIn accounts classified as either spam or ham for the validation set[6]. The metrics they used were precision and recall for each of their models. They were able to show the “full” algorithm of n=5 was able to out preform the light weight one, of n=3, on larger datasets. However, when it came to memory usage and speed on smaller datasets the light weight one came out ahead. Likewise, it had a low false positive rate of 3.3% [6]. This research on how Naïve bayes can be slightly transformed depending on what type of data it needs to process. With the light one outperforming the more complex model on smaller data sets they were able to trade memory for speed while still maintaining higher accuracy compared to the complex model. With its large dataset it preformed quiet well against unique data. One of the drawbacks to this study is perhaps the machines themselves they ran the experiment on which could reflect the results more than the actual performance of the algorithms.

Vangelis Metsis et Al conducted research using the Enron dataset to analyze different Naïve Bayes models on realistic emails[7]. They looked at specifically the Multi-variate Bernoulli NB, Multinominal NB, and Flexible Bayes. They were compared using recall on both legitimate emails, classified as ham, and spam messages, classified as spam[7]. They compared the results using ROC curves to show the different NB versions between true positives and true negatives. With this large real world dataset they concluded that two NB versions that were not used often at the time preformed best, specifically Flexible Bayes and multinominal. [7] This study has a massive positive of being used on a large realistic real world data set. Allowing the models to be able to intake a large amount of data. Likewise they used multiple different types of Naive bayes leading to diverse group results to compared each other with. The only drawback could be that the Enron corpus itself is quiet old and spam could have changed significantly enough that these results might not reflect the same on current real world examples.

Nurul Fitriah Rusland et. Al researched naïve Bayes across two different email datasets[3]. Likewise they used the WEKA tool to handle their datasets. The datasets used in this study was the Spam Data set that has over 9000 emails, and the SPAMBASE dataset which has over 4500 emails. The metrics they used between each data set were accuracy, precision, recall, and F-measure. They concluded through their research that Naïve Bayes preformed better on the SPAMBASE dataset. This could be due to the SPAMBASE dataset is multivariate from a single email account while the Spam dataset collected from many email accounts. Likewise, it was able to build the model faster because of this discrepancy in how the data was collected. Which this intern shows that the type of emails collected, and instances of the dataset influence the results.[5] With the usage of two different data sets of both larger size it helps this study a lot with validating its results and testing it on different emails. This is a huge plus compared to some of the other studies. The only drawback could be what they found in their discussion and results which is the SPAMBASE data set results could be throwing off the results due

to it being from a sole email account instead of from a variety of sources.

3. METHODOLOGY

3.1 Extreme Learning Machine

The Extreme Learning machine functions as a single-hidden-layer feedforward neural network (SLFN) with random input weights first randomly generated between the input layer and hidden layer. Then, the activation function is applied to the weighted sum of inputs, which generates outputs for the hidden layer to solve classification, regression, and pattern recognition tasks.[1] For this experiment the outcome of the model is if an email is spam or not. In the experiment the input is the contents of an email, while the output is the binary classification of '0' if the email content is spam, or '1' if the it is determined to not be spam. An example of an Extreme Learning Machine is shown below.

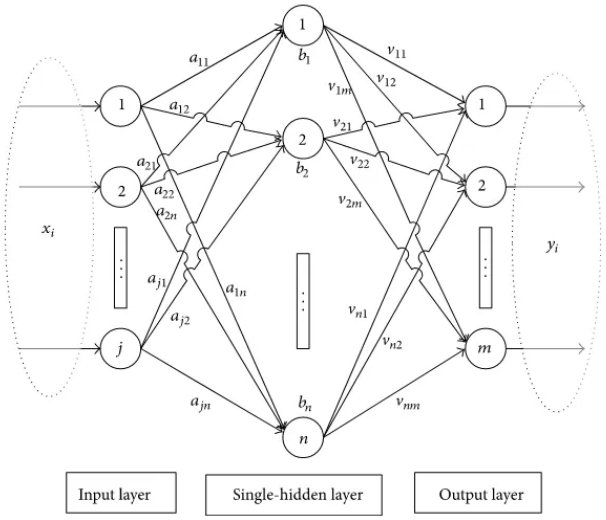


Figure 1.

In the Extreme Learning Machine's case the first layer is the input layer, where it maps the intake data onto a set of hidden nodes. Each node in the hidden layer has a randomly assigned weight and bias, which determines how strongly it responds to the input data. Likewise, that layer is connected to the hidden layer nodes, which is an assigned variable, which are then connected to the output layer.[9] Inside the hidden layer the weights between it and the input layer are randomly generated and fixed. Unlike the weights between the hidden layer and the output layer which are trained using a linear regression approach. What separates it between a standard Neural Network is how the weights between the hidden layer and output layer are handled. Here they are a special matrix, matrix beta, because of pseudo-inverse.[9]

The formula is as follows:

$$\sum_{i=1}^{\tilde{N}} \beta_i f_i(x_j) = \sum_{i=1}^{\tilde{N}} \beta_i f(a_i \cdot x_j + b_i) = t_j, j = 1, \ldots, N$$

Figure 2. [9]

Or

Y_i = beta * g(W *X_i +b)

Where Y_i is the predicted output, beta is the weight matrix between the hidden layer and the output layer, g is the activation function, w is the weight matrix between the input and hidden layer, X_i is the input data, and b is the bias vector. The input layer creates matrix X_i consisting of n samples, where each sample has d features. The hidden layer is the signature feature in this model. It consists of m neurons, where m is the typical should be smaller than n samples. The weights between the input layer and the hidden layer are randomly generated and fixed. H is the ouput of the hidden layer which can be computed as the following:

H = g(W * X_i +b)

Where:

G is the activation function, in this case sigmoid which can be defined as the following:

G(Z) = 1/(1 +exp(-z))

Where exp(x) is the exponential function, which is defined as e raised to the power of x. The input z to the sigmoid function is the weighted sum of the input features, with bias added: z = (W * X_i) +b

W is the weight matrix between the input and hidden layer, which is randomly generated and fixed. It has damsons m x d. b is the bias vector, which is also randomly generated and fixed. It has dimensions m x 1. The output of the hidden layer H is a matrix with dimensions m x n, where each column represents the output of one hidden neuron of one input sample.

The output layer produces the output predictions through weights between the hidden layer and the output layer being trained using linear regression. Let Y be the output matrix, where each row represents the output for one input sample. The wight matrix beta can be calculated as:

Beta = pinv(H) *Y

Where:

Pinv(H) is the pseudo-inverse of H. This calculates the weight matrix beta. Once that weight matrix has been calculated predictions can be made with the starting equation found below figure 2.

The positives of this approach compared to traditional neural networks is that it is able to be ready for predictions in one pass of the training data, instead of the traditional iterative looping. Likewise, it has shown high accuracy metrics scores compared to

traditional neural networks.[1] It only requires one attribute to tune, the number of hidden neurons, compared to traditional neural networks. Likewise, its algorithm structure makes it simple and efficient compared to complex traditional Neural Networks.[1] It can also be used for more than just classification tasks unlike Naïve Bayes.

However, it does come with some possible drawbacks. With the increase in the number of neurons added to the hidden layer comes with an increase of hardware demand unlike Naïve Bayes. [1] Likewise, it can be sensitive to the choice of activation function selected. [9]

3.2 Naïve Bayes

Naïve Bayes is a probability machine learning algorithm based on the Bayes' theorem, which estimates the probability of a specific event, or class, given input features. At the heart of the algorithm is the assumption that the features are independent given the class, which means that the presence or nonappearance of one does not depend on the presence or nonappearance of other features.[10] For this experiment the features of each email body, such as the appearance of certain words or characters are used to make predictions resulting in a binary value of either spam or not spam. This can be defined as the following:

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

Figure 3 [10] Fi

Which also can be represented by the following:

P(y | x) = P(x | y) * P(y)/P(x)

Where P(y | x) is the posterior probability, or updated probability after prior information, of class y given the input value of x. P(x | y) is the likelihood of the input x given class y. P(y) is historical probability of class y, and P(x) is the constant that ensures the probabilities sum to 1. The naïve portion comes from the assumption that the features are conditionally independent from each other which is defined as the following for binary classification specifically:

P(x_j | y) = theta_i^x_i*(1-theta_i)^(1-x_j)

Where (x_j | y) is the univariate probability distribution for the binary classification.

P(y), prior probability, can be estimated as the frequent occurrence of class item in the training data. With P(x) being the normalization constant. This can be defined as the following:

P(x) sum_y P(x | y) * P(y)

Where the sum takes over classes of y. When making predictions for new data the highest-class posterior probability gets chosen:

y_highest = argmax_y P(y |x)

Once passed through training data the algorithm is able to make predictions on any further. In the case of this experiment y would be denoted as the binary values of spam or not spam, while x would denote the email content with each word independent of each other. The history of those words in use for spam would determine the probability of how the email is classified.

There are a few advantages to this approach. One is that it does not involve neural networks and instead takes a statistical approach making it less hardware intensive and simpler to implement [2]. Likewise, it is a quick model compared to many complex machine learning models making it able to process high amounts of data quickly. Additionally, it has been a proven model with a rich history in spam detection.[2]

It does however have some drawbacks. The model itself is unable to understand the context of word association between each other in a sentence, allowing it to possibly be front loaded and beaten depending on how well trained the model is [2]. Additionally, it is highly dependent on being trained on exact features of spam, meaning it must be trained on large data sets to be efficient.

4. IMPLEMENTATION

4.1 Technology

These models, both Naïve Bayes and Extreme Learning Machine, were written in Python 3.9 in the Pycharm IDE. Additionally multiple libraries were used to help conduct research, interpret data, and data visualization for metric comparison. Here the following libraries were used: Scikit-learn for built in Naïve Bayes model, metric evaluation, and data splitting. Pandas for dataset reading. NumPy for mathematically functions. Matplotlib was used to build a confusion matrix around the test data for each model while Seaborn was used for the graphical design of it. Time was used to time each model while it ran the data set from training to testing output. The program was ran on a Ryzen 7 3750H 4 core processor with a NVIDIA GeForce GTX 1660 Ti with Max-Q GPU.

4.2 Data

The data that was used in this research was pulled from Kaggle. It is a sample preprocessed dataset from the SpamAssassin corpus containing 5,500 emails. The data set was split into 2 columns one containing the email body and the other one containing spam or ham depending on how the email was labeled. The email body column is labeled Message while the identifier is labeled key. The emails themselves would either be regular email messages or blatant spam advertisements, inappropriate content, phishing attempts, or random characters and symbols. This is what the models were trained on to predict unseen emails.

5. EXPERIMENTAL SETUP

5.1 Data Preprocessing

5.1.1 Data format

Inside this data set the column named key was used to label an email spam or ham. For my setup I determined it would be easier if I first made a new column called Label and have it in numerical format 1 or 0 for spam and ham respectively corresponding with the key column. After that the key column was removed.

5.1.2 Train-test split

The data was then split into two different sets at random selection. This randomizer was set to 44. The training set was used to train both models while the testing set was unseen emails to test the models against several metrics. Testing set consisted of 40% of the emails while the training set consisted of the remaining 60%. Both models used the exact same training and testing set.

5.1.3 Vectorizing text data

To prepare the data to be processed by the models it needed to be vectorized properly so that it was assigned to numerical format. This was implemented with the CountVectorizer from scikit-learn library. For this implementation the data was transformed using the bag of words technique. This transformed it into a sparse matrix of numbers where the row of each corresponds to the email, and each column corresponds to a unique word in the vocabulary. This further needs to be pushed into the form of a sparse matrix using the array call.

5.1.4 Scaling the data

To further prepare the data to be in the proper format for matrix format the test set needed to be scaled to the range of the training set features. This is one using the MinMaxScaler(). Without this it could throw matrix out of bounds errors

5.2 Algorithm Implementation

5.2.1 Extreme Learning Machine

The extreme learning machine model was implemented directly into python through tree functions, sigmoid, hidden nodes, and predict. In this case X is considered the email body matrix while y is considered the Label 0 or 1 depending on if the email is not spam or spam respectfully. It did need to implement the numpy library for specific mathematical calculations dot product and Moore-Penrose pseudoinverse. However, the rest was defined in python. A timer is first started and assigned to a variable to keep track of this metric. The input layer size is defined by taking the training data matrix size's value. Then the hidden size is determined by a fixed integer, in this case 1500. Meaning there is 1500 neurons in the hidden layer. Next the input weights and biases are randomly generated using numpy. The input weights are a matrix of size (input, hidden) with random values drawn numpy, and the biases are a vector of size (hidden) with random values using numpy.

Then the sigmoid activation function is defined which computes the hidden layer using the sigmoid formula passing in the parameter X. The hidden node function is then defined and takes the input data inside the function labeled X, and computes the dot product of it with the input weights, adds the bias, and applies previous sigmoid function and returns the hidden layer.

The output weights are calculated using Moore-Penrose pseudoinverse of the hidden layer, by calling numpy's pinv, and the values of why for the training set.

When this is completed the predict function is next which takes the parameter X and calls the hidden_node function and computes the dot product of the hidden layer and the output weights to obtain the predicted change of it being spam or not spam. And then returns its prediction.

The program then calls the predict function and passes the training matrix X through which pulls the other functions together to end up with a prediction on if it is spam or not spam. Once this test has been completed the timer is stopped and then calculated how long the algorithm ran for

5.2.2 Naïve Bayes

The Naïve Bayes model was implemented through the built in scikit-learn library unlike the Extreme Learning Machine. Before it is called though a timer must start and assigned a variable to keep track while the algorithm runs. It then grabs the vectorized training set X called X_train_vec and the corresponding y values called y_train.

The program calls the MoltinominalNB() function which creates an instance of the naïve bayes model and assigns it to nb. It then calls the fit() method passing it the corresponding X and Y value of X_train_vec, and y_train respectively. After doing so it follows the naïve bays algorithm without needing any further instructions. However it is important to point out that after it is trained through this function call it needs to be called again and the test data passed to it. This is handled with the .predict() call passing in the parameter of X_test_vec and then assigned to nb_prediction. Once this test has been completed the timer is stopped and then calculated how long the algorithm ran for.

5.3 Evaluation metrics

5.3.1 Confusion Matrix

For comparison of the models each will be scored on its testing set using a standard confusion matrix. This will indicate True Positives, True Negatives, False Positives, and False negatives. With the values inside the matrix indicated where each testing email was determined by the model. Likewise this model will be summarized with percentages to determine the Accuracy, precision, recall, and F-1-Score. This will help shed light on how the model preforms across the board and not just on accuracy. This will be plotted onto a heat map and then compared to the runtime of the model.

5.3.2 Algorithm runtime

This metric will be scored based on how long the algorithm runs from its start of training to the end of its testing using python built in time library. This will indicate exactly how the model preforms on a sizeable dataset and then compared to each other plus its confusion matrix. Doing so will show how time effect the model in in comparison to its actual ability to correctly classify spam emails.

6. Results

After the ELM model and the NB model was generated, predictions were made on the test set. After a confusion matrix was generated along with the time calculated for the runtime of each model.

The confusion matrix for the ELM was:

Table 1. Confusion Matrix for Extreme Learning Machine Model

	Negative (0)	Positive (1)
Negative (0)	1921	7
Positive (1)	56	245

According to this confusion matrix for the ELM model correctly protected 1921 emails as not containing spam (0 values) as true negatives. The model was able to also correctly predict 245 emails contained spam (1 values) that were truly positives. It incorrectly predicted 7 emails as spam that were not spam, thus predicting them as positives while they were true negatives. Likewise, it incorrectly predicted 56 emails as legitimate emails but they were spam, thus predicting them as negatives while they were true positives.

The confusion matrix for the artificial neural network model was:

Table 2. Confusion Matrix for Naïve Bayes Model

	Negative (0)	Positive (1)
Negative (0)	1926	2
Positive (1)	23	278

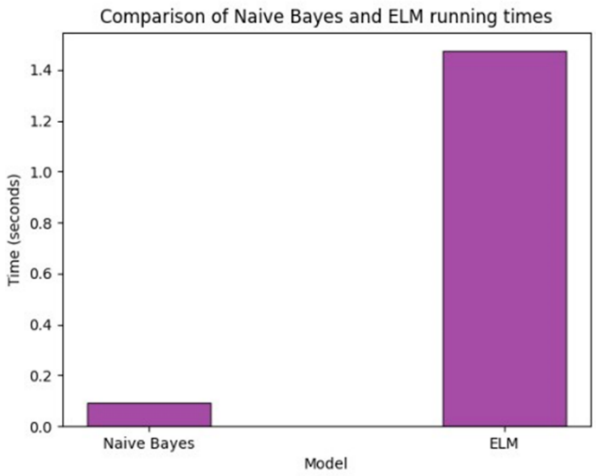
According to this confusion matrix for the NB model correctly protected 1926 emails as not containing spam (0 values) as true negatives. The model was able to also correctly predict 278 emails contained spam (1 values) that were truly positives. It incorrectly predicted 2 emails as spam that were not spam, thus predicting them as positives while they were true negatives. Likewise, it incorrectly predicted 23 emails as legitimate emails but they were spam, thus predicting them as negatives while they were true positives.

The runtime comparison of ELM and NB was:

Naïve Bayes: 0.1425682834 seconds

Extreme Learning Machine: 1.4634873921 seconds

Table 3 Comparison of Naïve Bayes and ELM running times.



According to this metric the ELM model took longer to train and test the model by 1.3209191087 seconds.

7. Discussion

The purpose of this study was to determine if extreme learning machine or naïve bayes better suited spam detection for emails and to judge them based on confusion matrix comparison along with run time. It can be noticed that both in table 1 and table 2 that the algorithms both preform a high accuracy for spam detection. However, it is worth noticing that Naïve bayes had and overall higher F1 score, of 95.70%, and precision, 99.29% compared to Extreme learning machine values of 88.61% and 97.22% respectfully. Furthermore, Naïve Bayes out preformed Extreme learning machine by 1.3209191087 seconds. This could be likely due to the limitations of the training data's size as a larger dataset could increase the F1 score and precision of the ELM model. Likewise for the runtime for Naïve Bayes could be attributed to the nature of the statistical algorithm versus a neural network. It is worth noting further that the ELM performance could be factored as a reflection of the hardware conducted in this study.

8. Conclusion

Both models preformed well against the dataset. Naïve Bayes was able to achieve high rates on confusion matrix slightly outperforming the Extreme Learning Machine model. When it came to runtime Naïve Bayes greatly outperformed the Extreme Learning Machine. This seems to reflect previous studies on the nature of how fast Naïve Bayes is for detecting spam. For the Extreme learning machine, the accuracy and speed could be reflective of the size of the training data set.[12][11] Overall, the model was able to preform well as a neural network and shows its capabilities at learning and processing data quickly. The problem for its false positives could be attributed to over fitting with the size of the hidden layer. However, both models seem to reflect similar conclusions as discussed in the literature review section. A more definitive conclusion could be arrived at with a larger dataset when comparing both models, however it is worth considering that Naïve Bayes runtime should not decrease because of how the statistical model functions.[13]

REFERENCES

[1] Wang, Jian, et al. "A Review on Extreme Learning Machine - Multimedia Tools and Applications." SpringerLink, Springer US, 22 May 2021, link.springer.com/article/10.1007/s11042-021-11007-7#Sec45.

[2] Webb, Geoffrey I., Eamonn Keogh, and Risto Miikkulainen. "Naïve Bayes." Encyclopedia of machine learning 15 (2010): 713-714.

[3] Rusland, Nurul Fitriah, et al. "IOPscience." *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, 1 Aug. 2017, iopscience.iop.org/article/10.1088/1757-899X/226/1/012091.

[4] Zhang, Zhijie, et al. "Directory of Open Access Journals." IEEE Access, IEEE, 1 Jan. 2020, doaj.org/article/4ca7c055a470444893367ec06baf02c3.

[5] Kaya, Yilmaz, et al. *An Expert Spam Detection System Based on Extreme Learning Machine*. Computer Science - Research and Development, July 2014, www.researchgate.net/publication/268025901_An_Expert_Spam_Detection_System_Based_on_Extreme_Learning_Machine.

[6] Eberhardt, Jeremy. "Bayesian Spam Detection ." *Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal*, University of Minnesota , Mar. 2015, digitalcommons.morris.umn.edu/cgi/viewcontent.cgi?article=1028&context=horizons.

[7] Metsis, Vangelis, et al. "Spam Filtering with Naive Bayes – Which Naive Bayes?" *Conference: CEAS 2006 - The Third Conference on Email and Anti-Spam*, CEAS , Jan. 2006, www.researchgate.net/publication/221650814_Spam_Filtering_with_Naive_Bayes_-_Which_Naive_Bayes.

[8] Dixon, s. "Global Average Daily Spam Volume 2021." *Statista*, Statista, 28 Apr. 2022, www.statista.com/statistics/1270424/daily-spam-volume-global/.

[9] Erdem, K. E. (burnpiro). (2020, May 29). *Introduction to extreme learning machines*. Medium. Retrieved March 16, 2023, from https://towardsdatascience.com/introduction-to-extreme-learning-machines-c020020ff82b

[10] C.D. Manning, P. Raghavan and H. Schütze (2008). Introduction to Information Retrieval. Cambridge University Press, pp. 234-265.

[11] Huang, G., Huang, G.-B., Song, S., & You, K. (2015). Trends in extreme learning machines: A Review. *Neural Networks*, *61*, 32–48. https://doi.org/10.1016/j.neunet.2014.10.001

[12] Rong, Hai-Jun, Yew-Soon Ong, Ah-Hwee Tan, and Zexuan Zhu. "A Fast Pruned-Extreme Learning Machine for Classification Problem." *Neurocomputing* 72, no. 1-3 (2008): 359–66. https://doi.org/10.1016/j.neucom.2008.01.005.

[13] Almeida, T. A., Almeida, J., & Yamakami, A. (2010, December 2). Spam filtering: How the dimensionality reduction affects the accuracy of naive Bayes classifiers -

journal of internet services and applications. SpringerLink. Retrieved April 20, 2023, from https://link.springer.com/article/10.1007/s13174-010-0014-7