INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

DEPARTAMENTO DE CIENCIAS E INGENIERÍA

Email Classifier with Naive Bayes

Alberto Castañeda Arana A01250647

Intelligent Systems Project

Querétaro, QRO **November 2021**

Abstract

With over more than two decades of becoming one of the mainstream communication channels, emails are subject to huge volumes of spam which most end up hidden thanks to spam filtering detection technology everyday. In this document we will go into detail about creating an email classifier using supervised learning classification algorithm Multinomial Naive Bayes by scratch and comparing it to other existing framework classifiers.

Content

Context of the problem

With technology that has arisen from computacional advances from the last century we have achieved global communication for everyone with access to the internet. While there are many methods of online communication nowadays, one of the oldest one yet still most used are emails. According to a statistical report published in 2021, the average number of legitimate email messages sent over the internet daily is around 22.43 billion emails *[1].* While this is an impressive amount of emails already, the wording of the previous sentence implies the existence of non-legitimate email messages. Non-legitimate emails are considered emails that pretend to be something or someone that it is not, to sabotage or for personal gain. A more colloquial term to classify these non-legitimate emails would be using the modern term ‘Spam’. The term ‘Spam’ comes from a *Monty Python* sketch in which the name of the canned pork product Spam is ubiquitous, unavoidable, and repetitive *[2]* just like spam emails can be.

Going back to our initial statistics, it is said that spam emails account for nearly 85% of all emails. The previous would amount to a total of 122.33 billion emails daily around the globe. If you have ever used email it is most likely that you have encountered a spam email. However, usually we don't get nearly as much spam emails in our inbox as the statistics imply and the reason behind this is thanks to Spam filtering technology. Spam filtering technology, as the name implies, allows detection of email that is most likely to be spam so that it can be hidden. Computers sometimes make mistakes, as we humans do, so the email marked as spam is not deleted but marked and hidden in case it is a case of a False Positive (email marked as spam when in reality it is a legitimate email).

Developing accurate spam filtering technology is important not only because we are able to hide useless spam emails, but because some of these spam emails can be highly dangerous. According to Nucleaus Research 2012, email spam costs businesses around $20.5 billion every year and estimates suggest that damage might rise to a staggering $257 billion in a few years. The reason spam emails can lead to huge losses like previously described is that spam emails maliciously often contain Phishing links or other kinds of attacks including malware attachments, spyware, ransomware, and other kinds of virus attacks. Most of these attacks require human interaction, which makes disguised emails the perfect tool to make people fall into these traps.

That being said, while it is important to detect and hide spam email it is also important to not hide legitimate email mistakenly. Legitimate emails can be very important messages or invitations which can be highly annoying or even costly if it were to be hidden by error. For this report we will be using the term 'Ham' to describe the opposite of spam emails: legitimate emails.

In the following section we will develop an approach for a spam filtering technology using machine learning concepts.

Solution

Going deeper into the mentioned Spam filtering technology from the previous chapter, one the most commonly used ways to achieve this is using machine learning. Using machine learning, we can create software applications that can be trained to classify a given email as ham or spam. Classification in machine learning and statistics is a supervised learning approach in which the computer program learns from the data given

to it and makes new observations or classifications *[3]*. In the context of email classification, we could use a classification algorithm to be able to predict if a given email is either spam or ham.

There are many classification algorithms that we can use, yet there is no “best one” to choose for a specific problem which is why we must analyze each one of them. For this project I have decided to implement the Naive Bayes Algorithm from scratch in order to create a hypothesis model. Naive Bayes algorithm was chosen due to its speed, which is important when we consider the volume of emails that would need to be classified in practice. Another reason that Naive Bayes is a good candidate is that it works on dependent events and the probability of an event happening in the future can be detected from the previous occurrence of the same event. For example, if some words occur in spam but not in ham, it is highly likely that the email is spam. Further on this section we will go more in detail on how this algorithm works and we will also be comparing the results with other classifier algorithms. For now, we must choose and analyze the data that we will use to train this model.

The dataset used for this implementation was obtained from Kaggle, based on a sample from the public Enron dataset *[4]*. The Enron dataset is a public corporation email dataset initially released in 2004 that is commonly used as a training and testing source for email filtering *[5]*. It is important to note that the Enron dataset is preprocessed to replace every sensitive data like names, addresses, phone numbers, etc. The dataset contains the content of each email, along the classification given to the email (spam or ham).

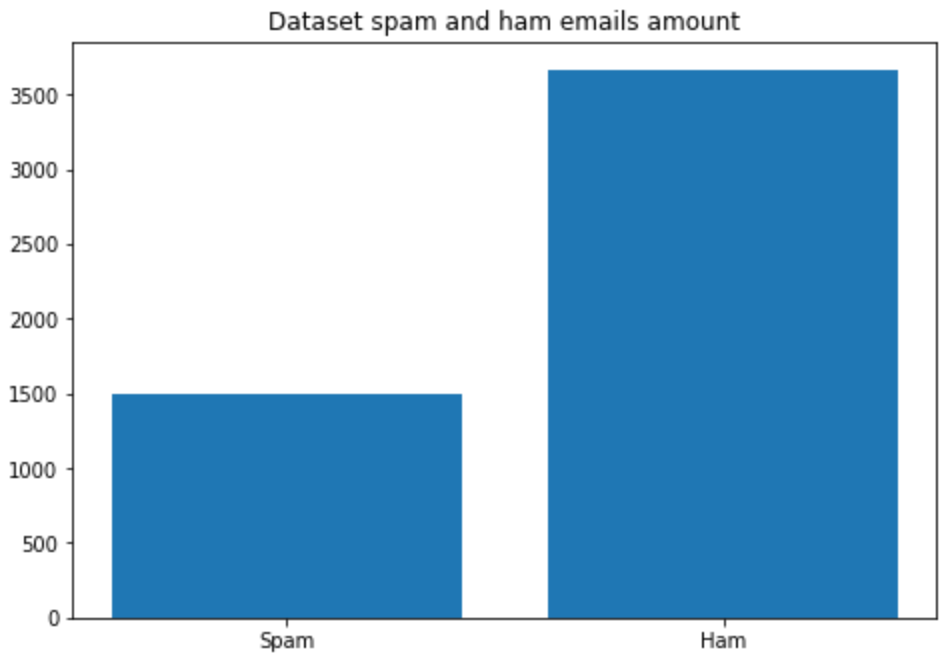

Figure 1. Amount of emails in the dataset by their classification.

Doing some exploratory data analysis on this dataset, we can create a word cloud from the email contents to find some patterns.

Figure 2. Word cloud of spam and ham emails vs cleaned spam and ham emails

A word cloud represents the frequency of a word in a list of words (in this case, our email contents list) in relation to it's visual size. As you can see from the top images of Figure 2, there are a lot of words that do not really give too much information about common words found in ham or spam emails. The reason behind this is that the dataset contains connector words, name replacements with Enron, special characters, and other replacement words that the Enron dataset uses to replace sensitive data. In order to get more information, we can apply some preprocessing for our data to exclude words that do not give us too much information. The following pre-processing was done in order to get the bottom images from Figure 2:

- Replace enron specific dataset words:
    - Subject
    - Enron
    - Etc

    - Ect
    - Hou
    - Nbsp
    - Cc
    - Will
- Lowercase every email
- Remove single letter words. Example: “a”
- Remove number words: Example: “23”
- Remove special characters: Example “$”, “.”

Doing this pre-processing, we got the bottom images from Figure 2 which indicate some command words found in ham and spam emails. By doing this pre-processing, we can remove words that do not give too much information and thus improve our model’s predictions.

An additional step that can be made in order to improve both accuracy and performance is by applying stemming or lemmatizing. Stemming and lemmatization are text normalization techniques used to group words by their base form.

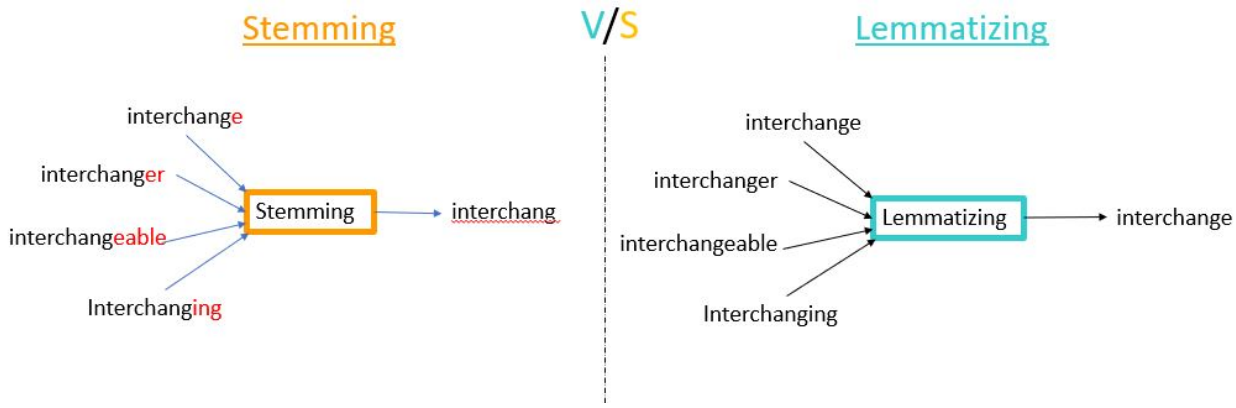

Figure 3. Stemming vs Lemmatizing *[6]*

As seen on Figure 3, while similar they do have some differences. Lemmatizing is more accurate, but it is more performance heavy. Applying one of these techniques would make our training faster and may slightly improve the accuracy of the predictions, however by applying these techniques we would be restricting our training data to be strictly in the english language. To avoid constraining the dataset to the english language, and to simplify the implementation of this algorithm stemming and lemmatization will be out of the scope for this project.

Now that we have our pre-processed and cleaned data, we are now able to dive into the Naive Bayes algorithm as promised. The formula for the Bayes' theorem is as follows:

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

Figure 4. Bayes' theorem formula *[7]*

Where:

- P(A | B) - The probability of event A occurring, given event B has occurred
- P(B | A) - The probability of event B occurring, given event A has occurred
- P(A) - The probability of event A
- P(B) - The probability of event B

Using this formula as a base for spam filtering,we want to find the probability an email is spam, given it's content words. To do this, we must find the probability that each word in the email is spam, and then multiply these probabilities to get the total chance of the email being spam and equally do the same for ham emails. Using the following formulas, we can obtain the probabilities of each classification and see which one has a higher probability that we can choose as a result:

$$P(Spam \mid word_1, word_2, word_3, \ldots, word_n) = P(Spam) * \prod_{i=1}^{n} P(word_i \mid Spam)$$

Figure 5. Probability of a email being spam

$$P(Ham \mid word_1, word_2, word_3, \ldots, word_n) = P(Ham) * \prod_{i=1}^{n} P(word_i \mid Ham)$$

Figure 6. Probability of a email being ham

That means, if the result of Figure 5 is greater than Figure 6 then the email is classified as Spam.

Expanding the formulas, we also have the following:

$$P(word_i \mid Spam) = \frac{N_{word\ i \mid Spam} + \alpha}{N_{Spam} + \alpha * N_{Vocabulary}}$$

Figure 7. Probability of a word being spam

$$P(word_i \mid Ham) = \frac{N_{word\ i \mid Ham} + \alpha}{N_{Ham} + \alpha * N_{Vocabulary}}$$

Figure 8. Probability of a word being ham

Where:

$$N_{word\ i \mid Spam} = number\ of\ times\ word_i\ occurs\ in\ spam\ emails$$
$$N_{word\ i \mid Ham} = number\ of\ times\ word_i\ occurs\ in\ ham\ emails$$
$$N_{Spam} = total\ number\ of\ words\ in\ spam\ emails$$
$$N_{Ham} = total\ number\ of\ words\ in\ ham\ emails$$
$$N_{Vocabulary} = total\ number\ of\ words\ in\ the\ vocabulary\ (all\ distinct\ words)$$
$$\alpha = 1$$

As you can see, there is a slight modification to the original Bayes' Theorem by adding a smoothing variable alpha. The reason behind this is to handle the case that the previous variables make the formula result in 0.

In theory, these formulas could let us classify the emails. However, there is a slight problem when transitioning to a computer environment. As defined in Figure 5 and Figure 6, our total probability is essentially a multiplication of N probabilities. The bigger our dataset becomes, these probabilities can become smaller and smaller and the problem comes when attempting to save these values on computer memory due to single-precision floating-point format. As of this writing, most computers nowadays can store a maximum of 32 significant digits with float variables *[8]*. For this reason, attempting to multiply very small probabilities can lead to a numerical underflow where the variable can not store the full precision of the percentage and it is saved as a zero. If both of our probabilities were equal to zero, we wouldn't be able to choose which probability is bigger and our model would be constrained to small datasets. Thankfully, what we can do is transform our probability multiplications to logarithmic space. By summing the log of the probabilities, we could equally discern which classification has a bigger probability without risking an underflow problem. Our resulting formula by doing this would change to the following:

$$P(Spam \mid word_1, word_2, word_3, \ldots, word_n) = ln\, P(Spam) + \sum_{i=1}^{n} ln\, P(word_i \mid Spam)$$

Figure 9. Probability of a email being spam in logarithm space

$$P(Ham \mid word_1, word_2, word_3, \ldots, word_n) = \ln P(Ham) + \sum_{i=1}^{n} \ln P(word_i \mid Ham)$$

Figure 10. Probability of a email being ham in logarithm space

As specified by the Naive Bayes algorithm, a vocabulary must be created in order to classify emails. Considering the proposed dataset only contains the email messages, a transformation must be done in order to create the vocabulary and word count per message. The following diagram would represent the transformation to create the final training set:

Label	Text
spam	CONGRATULATIONS! You are the 1,000,000th visitor!
ham	Hello! How are you today!
spam	Click here to claim your price

Vocabulary

Label	Text	congra	you	are	the	visitor	hello	how	today	click	here	to	claim	your	price
spam	CONGRATULATIONS! You are the 1,000,000th visitor!	1	1	1	1	1	0	0	0	0	0	0	0	0	0
ham	Hello! How are you today!	0	1	1	0	0	1	1	1	0	0	0	0	0	0
spam	Click here to claim your price	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 11. Transformation of data set to training set and vocabulary

As Figure 11 indicates, the vocabulary would be equal to all word count column names.

Algorithm and transformation techniques established the model training and testing can be put in action. A testing sample and a test sample will be used from the dataset for the models training and testing respectively.

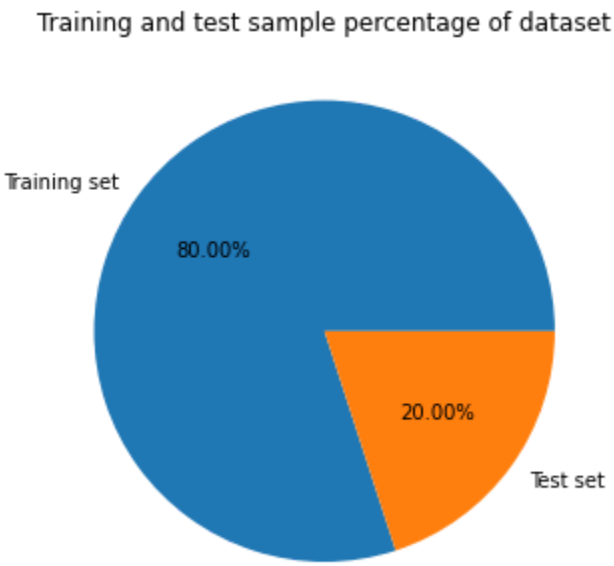

Figure 12. Training and test set distribution

As mentioned during the beginning of the chapter, we will be comparing the Naive Bayes model with Scikit's Decision Tree Classifier in Scikit's version of Naive Bayes classifier using the same training and test data set to compare the results between these three.

Results

Once each model was trained and tested, the accuracy metrics are now plotted to Confusión Matrix plots in order to calculate precision and recall of each algorithm:

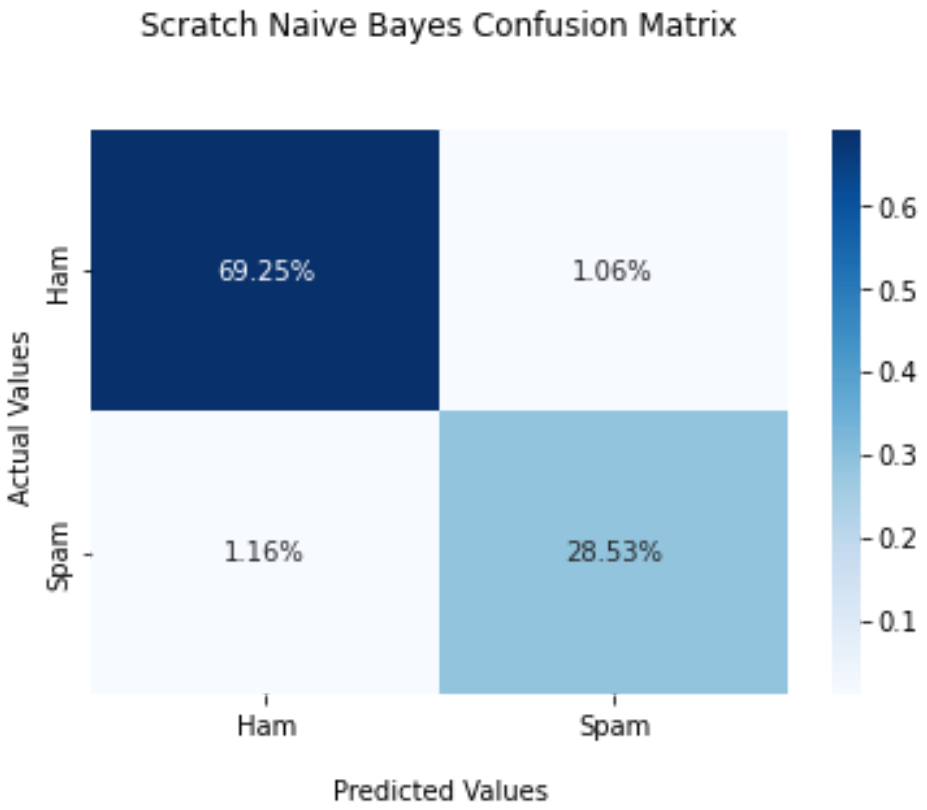

Figure 13. Scratch Naive Bayes Confusion Matrix

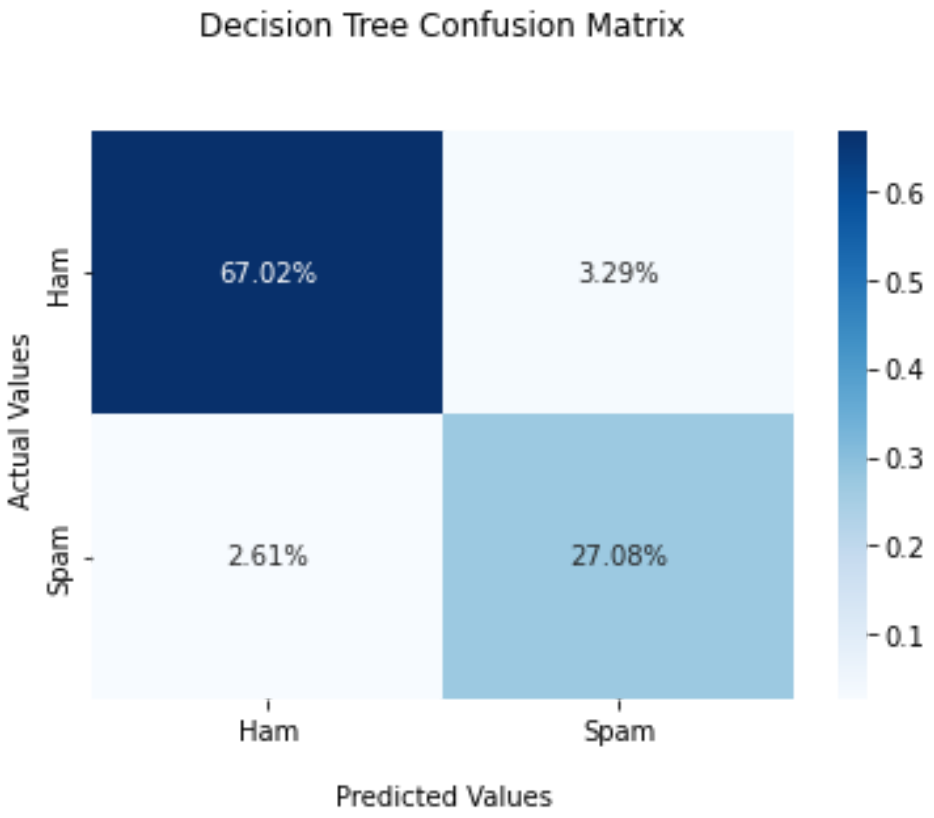

Figure 14. Decision Tree Confusion Matrix

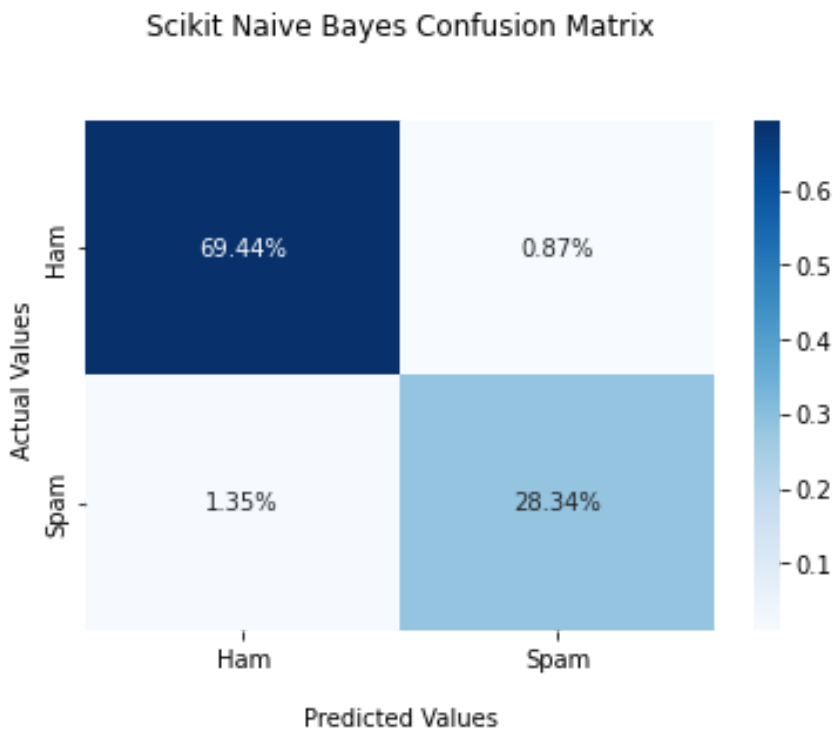

Figure 15. Scikit Naive Bayes Confusion Matrix

Additionally, the following training time was recorded:

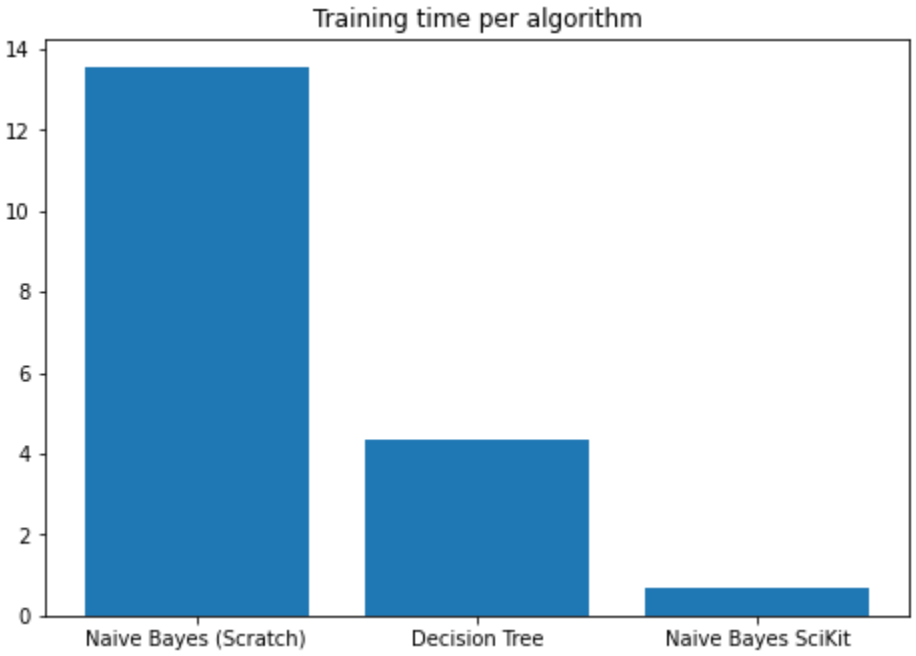

Figure 16. Training time in seconds per algorithm

Condensing our results into one single table would be as follows:

Algorithm	Precision	Recall	Accuracy	Training Time
Scratch Naive Bayes	97.88666 %	97.88666 %	96.9052 %	13.5691 seconds
Scikit Decision Tree	95.2 %	95.2 %	93.0367 %	4.3287 seconds
Scikit Naive Bayes	98.6666 %	97.6253 %	97.2920 %	0.69298 seconds

As we can observe from our results, not only did the scratch Naive Bayes algorithm approximately matched the SciKit implementation's accuracy, recall, and precision but it also surpassed the Decision Tree. One noticeable drawback that the scratch Naive Bayes implemented has is the slow performance on training time considering that the Naive Bayes algorithm should be fast as seen on the SciKit implementation which is approximately 20 times

faster. Looking at the implementation, it is clear that the training could be sped up if we used the parallelism paradigm on the training, which I suspect that the Scikit algorithm uses internally.

Additionally, due to the high amount of data loaded into RAM on training, there were occasional computer hiccups that freezes the computer until it is optimized. While it has been optimized to use less resources, it is highly recommended to do the training on a high end computer, or use less data set samples for the training.

By making more training iterations and seeing the result of confusión matrix plots, there is a tendency where Naive Bayes tend to mark real spam emails as ham (False Negative) more than viceversa False Positives. On the other hand, the Decision Tree model tends to the contrary.

While the models have a high accuracy, it is important to note that they can be biased towards emails similar to the ones to the Enron corp dataset, so it is important to keep training with more varied data in order to improve accuracy on real email queries.

Bibliography

1. *Cvetićanin, N., Jovanović, B., & Vojinovic, I. (2021, February 11). What's on the other side of your inbox - 20 spam statistics for 2021. DataProt. Retrieved November 20, 2021, from https://dataprot.net/statistics/spam-statistics/*
2. *Merriam-Webster. (n.d.). Spam definition & meaning. Merriam-Webster. Retrieved November 20, 2021, from https://www.merriam-webster.com/dictionary/spam*
3. *Waseem, M. (2021, July 29). Classification in machine learning: Classification algorithms. How To Implement Classification In Machine Learning? Retrieved November 20, 2021, from https://www.edureka.co/blog/classification-in-machine-learning*
4. *Garnepudi, V. (2019, January 23). Spam mails dataset. Kaggle. Retrieved November 20, 2021, from https://www.kaggle.com/venky73/spam-mails-dataset*
5. *Cohen, W. W. (2015, May 8). Enron Email Dataset. Enron email dataset. Retrieved November 20, 2021, from https://www.cs.cmu.edu/~enron*
6. *Singhal, G. (2020, October 5). Gaurav Singhal. Pluralsight. Retrieved November 20, 2021, from https://www.pluralsight.com/guides/importance-of-text-pre-processing.*
7. *https://corporatefinanceinstitute.com/resources/knowledge/other/bayes-theorem/*
8. *Bayes' theorem. Corporate Finance Institute. (2021, July 8). Retrieved November 20, 2021, from https://corporatefinanceinstitute.com/resources/knowledge/other/bayes-theorem/*
9. *Log-sum-exp trick to prevent numerical underflow. | Log-Sum-Exp Trick to Prevent Numerical Underflow. (2013, July 16). Retrieved November 20, 2021, from http://wittawat.com/posts/log-sum_exp_underflow.html*