BUILDING A SPAM FILTER USING MACHINE LEARNING

Team: Amey Arya and Nikhita Singh

PROBLEM OVERVIEW:

With the entire world moving towards a digital age, emails have become an integral part of our lives. Since, most, if not all the information important to us is communicated through emails, it has become an arduous task to keep track of what is more important. This is where spam filters come in. These filters classify the emails as spam and non-spam to make our lives easier.

We are trying to tap into this problem and understand how spam filters work using Machine Learning techniques. We plan to build models using different classifiers, and train them so that they predict if an email is spam or not with some substantial degree of accuracy.

WHY SPAM FILTERS?

Not only are these spams emails time consuming, they may also contain links that lead to phishing web. These emails can also carry malware and viruses that can compromise company security and data. They may host sites that are hosting malware - or include malware as file attachments.

PROBLEM STATEMENT:

Inputs:

The dataset that we are going to use is a pre-processed subset of the Ling-Spam Dataset, provided by Ion Androutsopoulos. It contains spam messages and messages from the Linguist list.

Features:

- Important or key words from the email
- Frequency of each key word in the dataset
- Basic (stop) words like “and” and “is” are removed. The words are also stemmed to remove prefixes and suffixes
- The length of an email

Algorithms:

We plan to implement various classifiers to build filters so that it predicts if an email is spam or not with some substantial degree of accuracy:

- Naïve Bayes
- Decision Tree
- Random Forest

- Support Vector Machine
- k Nearest Neighbor

Output:

- Classify an email as spam or non- spam
- To develop an intuition for real-world problems
- An analysis of how each technique differs from the other and which one gives a better result.

DATASET (Preparation and Pre-processing):

- **Cleaning and scraping the dataset:**
  We are using the Ling-Spam Dataset, provided by Ion Androutsopoulos which is available publicly. It is unfiltered and requires cleaning.

  The filters we applied are:

  - **Lower-casing**: The entire email is converted into lower case, so that capitalization is ignored (e.g., IndIcaTE is treated the same as Indicate)

  - **Stripping HTML**: All HTML tags are removed from the emails. Many emails often come with HTML formatting; we remove all the HTML tags so that only the content remains

  - **Normalizing URLs**: All URLs are replaced with the text "httpaddr"

  - **Normalizing Email Addresses**: All email addresses are replaced with the text "emailaddr"

  - **Normalizing Numbers**: All numbers are replaced with the text "number"

  - **Word Stemming**: Words are reduced to their stemmed form. For example, "discount", "discounts", "discounted" and "discounting" are all replaced with "discount". Sometimes, the Stemmer strips off additional characters from the end, so "include", "includes", "included", and "including" are all replaced with "includ"

  - **Removal of non-words**: Non-words and punctuation have been removed. All white spaces (tabs, newlines, spaces) have all been trimmed to a single space character.

As an example, we take the 3-380msg4.txt from the above-mentioned dataset. The message looks something like this:

Subject: postings

hi , i ' m working on a phonetics project about modern irish and i ' m having a hard time finding sources . can anyone recommend books or articles in english ? i ' , specifically interested in palatal (slender) consonants , so any work on that would be helpful too . thanks ! laurel sutton (sutton @ garnet . berkeley . edu

Post cleaning the data we obtain a result as follows:

posting hi m work phonetics project modern irish m hard source anyone recommend book article english specifically interest palatal slender consonant work helpful too thank laurel sutton sutton garnet berkeley edu

In this way we have cleaned the dataset and categorized it into four broad categories, namely, non-spam train dataset, spam train dataset, non-spam test and spam test dataset.

- **Generating Dictionary:**

There are a lot of possible ways to choose the words to create a dictionary. In our case, we chose the first 2500 most frequent words across all the emails. The algorithm used for this is as follows:

1. Load and concatenate all the emails in one string
2. Split the emails by white space so we can count the number of occurrences of each of the words
3. Build cell array of unique words and counts
4. Save the word count in dictionary.txt file

After creating the dictionary, it looks something like this:

1. email 2163
2. order 1648
3. address 1645
4. language 1534
5. report 1384
6. mail 1364

…

- **Generating features:**

The gist of this step is to generate features so that the resulting structure is ready to apply the three algorithms namely, Naïve Bayes, Random Forest and Decision Tree. The dictionary that we already created contains all the 2500 words (features) based on which we created the prediction model. Here we counted the number of occurrences of each word from the dictionary in the emails.

A sample of this generated feature is as follows:

1 7 1

1 12 2

1 19 2

1 22 1

1 25 1

The first column is the document sequence number, the second is the sequence number for word in the dictionary and the third column is the frequency of that word in a single email. For example, the first row means that document is one which according to our dataset is 3–380msg4.txt. We segregated this data into 2 text files, namely, features_train.txt (containing structured data for training the model) and features_test.txt (containing structured data for testing the model).

- **Generating the Machine Learning Model:**

Since, now we had set up the structured data set up, we used our three algorithms so that we get the prediction model.

Naïve Bayes Classifier:

The Naïve Bayes algorithm is a simple *probabilistic classifier* that calculates a set of probabilities by counting the frequency and combination of values in each dataset. It counts the number of occurrences of a w (word of a dictionary) in all the c (sum of all occurrences of the dictionary words). This probability will be calculated separately first on spam and then on non-spam emails.

$$\hat{P}(w_i|C) = \frac{Count(w_i, C) + 1}{\sum_{w \in v}(Count(w, c) + 1)}$$

We implemented the Naïve Bayes algorithm on our structured dataset.

Decision Tree Classifier:

A decision tree is a straightforward method of classifying something. It organizes a series of test questions and conditions into a tree like structure. It is based on yes/no questions. Each question has either a True or False answer that splits the node. Based on the answer to the question, a data point moves down the tree.

Gini Impurity: The Gini Impurity of a node is the probability that a randomly chosen sample in a node would be incorrectly labeled if it was labeled by the distribution of samples in the node.

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

Random Forest Classifier:

The Random forest is a meta-learner which consists of many individual decision trees. Each tree votes on an overall classification for the given set of data and the random forest algorithm chooses the individual classification with the most votes. Each decision tree is built from a random subset of the training dataset, using what is called replacement, in performing this sampling. That is, some entities will be included more than once in the sample, and others won't appear at all. In building each decision tree, a model based on a different random subset of the training dataset and a random subset of the available variables is used to choose how best to partition the dataset at each node. Each decision tree is built to its maximum size, with no pruning performed. Together, the resulting decision tree models of the Random forest represent the final ensemble model where each decision tree votes for the result and the majority wins. Calculating the total spam score which is the minimum score required to mark a message as spam. If category is zero email classified as non-spam & if category is one email classified as spam.

Support Vector Machine:

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyper-plane. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class. Support vectors are the elements of the training set that would change the position of the dividing hyperplane if removed.

Tuning Parameters: In real world application, finding perfect class for millions of training data set takes lot of time. Hence, we have three tuning parameters. Varying these parameters accordingly can achieve a considerable amount of accuracy in a reasonable amount of time. These parameters include:

1. Kernel: This plays a role in learning the hyperplane. A kernel takes as input two points in the original space, and directly gives us the dot product in the projected space. A couple of examples for kernels are as follows: linear kernel, rbf kernel, polynomial kernel, exponential kernel.
2. Regularization Parameter (C): The Regularization parameter tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger margin separating hyperplane, even if that hyperplane misclassifies more points.
3. Gamma: The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible separation line are

considered in calculation for the separation line. Whereas high gamma means the points close to plausible line are considered in calculation.

k-Nearest Neighbor:
kNN is a model that classifies data points based on the points that are most like it. It uses test data to make an "educated guess" on what an unclassified point should be classified as The algorithm then works by finding the distance between a data point and the test data point (new introduced point). The distance is calculated using Euclidean Distance which is given by the formula:

$$d(p,q)=d(q,p)=\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

- **Deciding the train-to-test ratio:**
  Upon cleaning the dataset and further discussions, we decided on a train to test ratio of 9:1. This will allow us to properly train our ML model and also will leave the 10% part for proper testing of the model.

RESULTS:

The results below were gathered using 2602 emails to train (2170 - non-spam and 432 - spam) and 291 emails to test (49 – spam and 242 – non-spam) so as to maintain a 9:1 ratio of test:train.

To perform the analysis, we use a confusion matrix. This matrix is represented as: The analysis of each classifier is done based on three parameters:

	PREDICTED (TRUE)	PREDICTED (FALSE)
ACTUAL (TRUE)	True Positive	False Positive
ACTUAL (FALSE)	False Negative	True Negative

	Naïve Bayes	Decision Tree	Random Forest	SVM	kNN
Confusion Matrix	[240 2] [12 37]	[231 11] [8 41]	[247 0] [15 29]	[221 17] [16 37]	[201 16] [29 45]

1. Accuracy: It is the most intuitive performance measure. It is simply defined as the ratio of correctly predicted observations to the number of total observations.

$$a = \frac{true\ positive + true\ negative}{true\ positive + true\ negative + false\ positive + false\ negative}$$

2. Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positive observations.

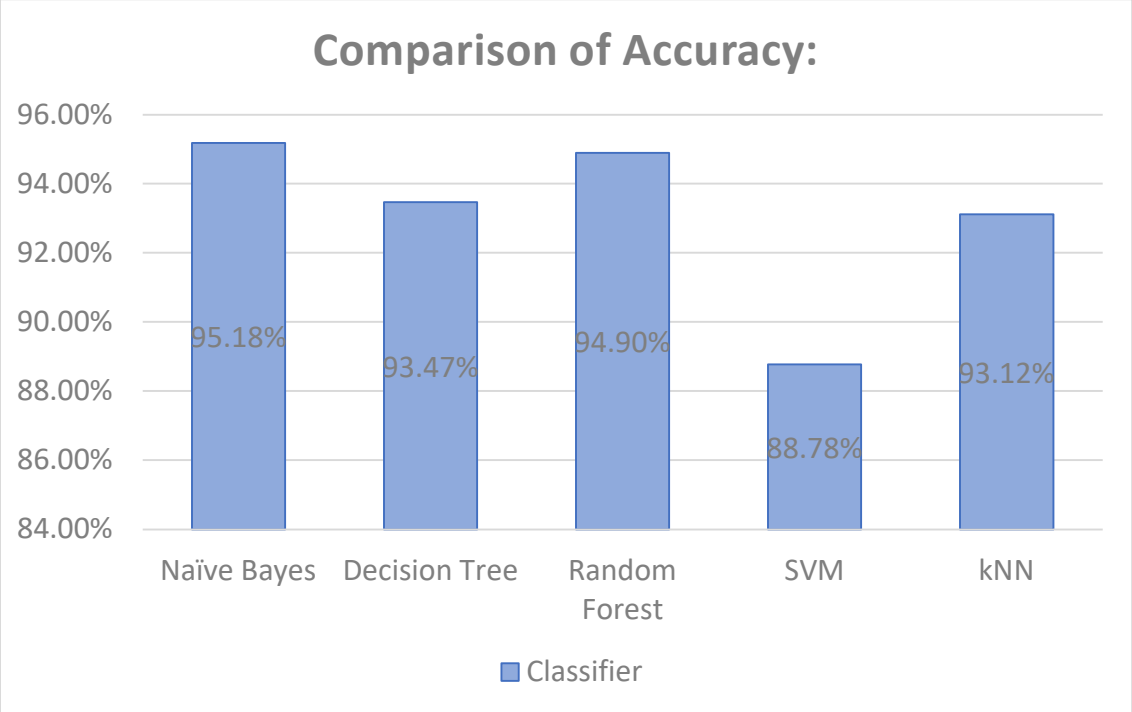
$$p = \frac{true\ positive}{true\ positive + false\ positive}$$

3. Recall: Recall is the ratio of correctly predicted positive observations to the all observations in actual class.

$$r = \frac{true\ positive}{true\ positive + false\ negative}$$

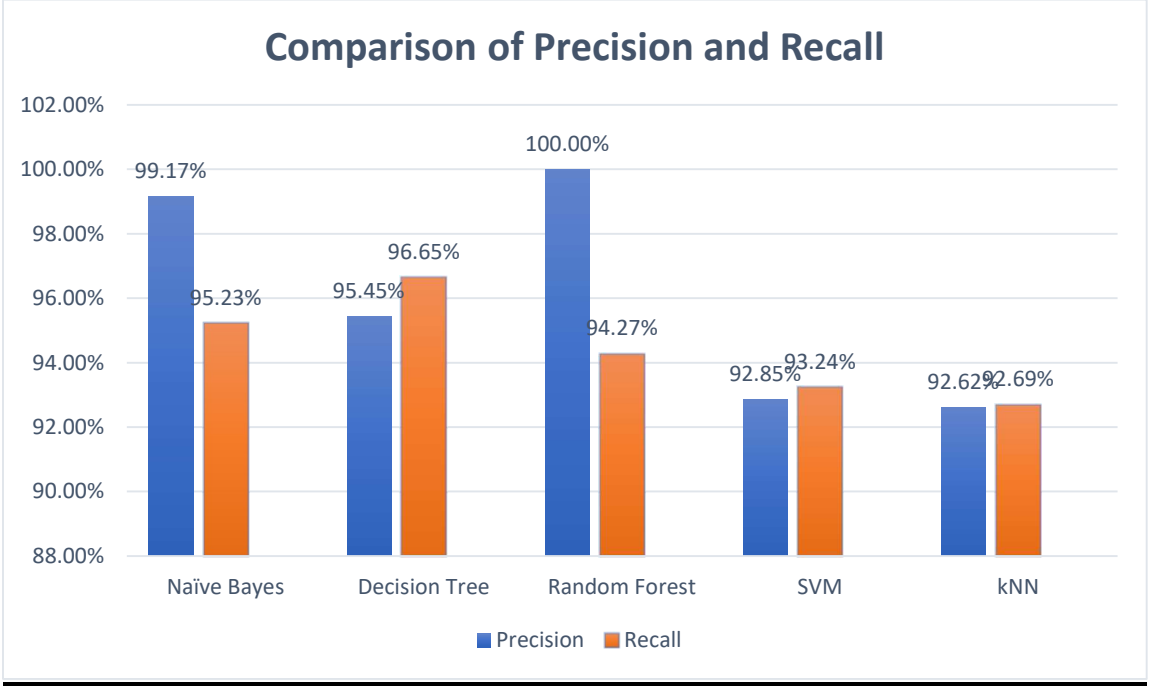
The analysis of all five classifiers are done based on the mentioned parameters. The results are as follows:

- **Comparison of Accuracy:**



It is observed that Naïve Bayes classifier gives the maximum accuracy followed by Random Forest classifier. This states that both Random Forest and Naïve Bayes are good choices for spam filtering. The SVM classifiers give the minimum accuracy amongst all the classifiers. This is clearly evidenced in the Confusion Matrix of all models.

- **Comparison of Precision and Recall:**



Coming to our problem of classifying an email as spam or non-spam, it is critical that an important non-spam mail does not end up in a spam folder. However, it might not be as critical that a spam mail shows up as a non-spam email. So, it is important to select a model based on all three parameters, i.e., accuracy, precision and recall with the right balance. This balance is greatly achieved in Random Forest and Naïve Bayes classifiers.

ANALYSIS:

Comparison between these machine learning models:

TIME:

Right from the start, it was clear to us that the decision tree was very slow to train. It took exponentially longer to train as we believe it was overfitting the dataset. Naive Bayes and Random Forest, both were comparably faster at classifying but as expected started to take much longer as we increased the size of the dataset.

In order to tackle this problem, we tried to use an efficient pickling technique such as pkl/NumPy n-D array. But, the prediction accuracy of our models is taking a significant hit. The accuracy of each model came down by around 15%-20%. The accuracy of the most effective models namely, Random Forest and Naïve Bayes came down to 86.75% and 79.31% respectively. We further tried

to explore other pickling techniques in JSON and CSV file formats. Although, these improved the time taken to train the models, the accuracy results of the models were not as good. Therefore, we stuck with the conventional approach to train and test these models without pickling.

CLASSIFICATION ACCURACY:

In general, all the algorithms performed very similarly with the Naive Bayes usually outperforming the other classifiers. However, what caught our attention was that the Random Forest had a trend of outperforming the decision tree by a couple of percent only. We had expected there to be larger differences between the different classifiers, so we were quite surprised that the different algorithms performed quite similarly.

What was more interesting to look at was both how the algorithms performed based on the data fed into them. At first, we started off by just feeding in the dataset and noticed that the model would become a little more accurate as we increased the number of emails to train with. But there was an inherent problem with this method. There were many emails in the dataset that were non-spam compared to the number of emails that were spam. The dataset consists of 2602 emails in total, out of which 2170 are non-spam emails. That is, about 83.4% of total emails are non-spam.

Therefore, at this point, we wanted to try to look at how the models would perform given an even number of spam and non-spam emails. When we did that, the performance was a lot lower, averaging between 75-80% classification accuracy.

We believe the model performed better when given a natural slice of the dataset maintaining the original ratios because our dataset inherently had a lot more non-spam emails than spam. Therefore, our model became a lot better at classifying emails that weren't spam than classifying emails that were spam.

Therefore, to fix this issue, given more time, we would train our model with the whole dataset we had after being cleaned in order to give it more exposure to emails that are spam, clean our dataset better, or create a dataset ourselves that had more spam emails to train our model. We believe this is the right way forward because from limited testing given the time we had, we noticed that as we increased the number of emails we trained with, the model performed slightly better.

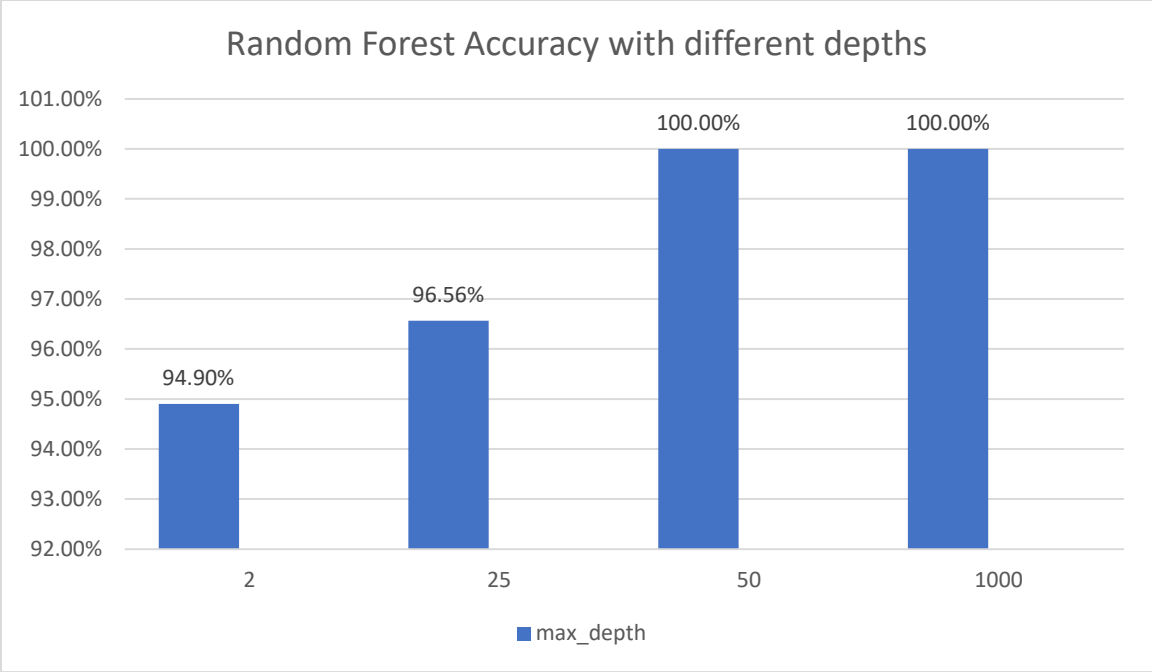
For example, when only given 864 emails, the models accuracy drops. The Naïve Bayes classifier (our best performing model) classified with an accuracy of 89%, but with the increase in the dataset size, the accuracy shoots to a high of 95%. The primary reason behind this, we believe is the fact that with the increase in size of dataset, the model learns better about classifying tricky words such as "banking" or "communications". While we spot these words quite often in non-spam emails, there is also a possibility that they may be in a spam email as well.

Overall, we think there are many ways we can improve our results given more resources and time.

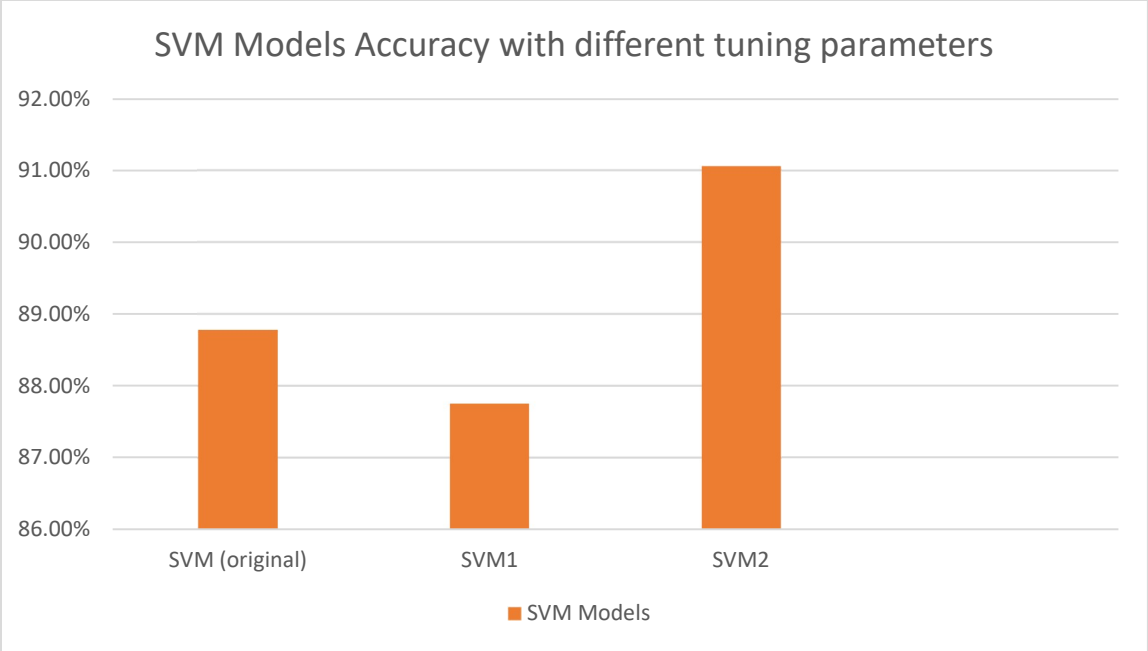
FURTHER ANALYSIS:

We furthered our analysis by tweaking some parameters for the given classifier:

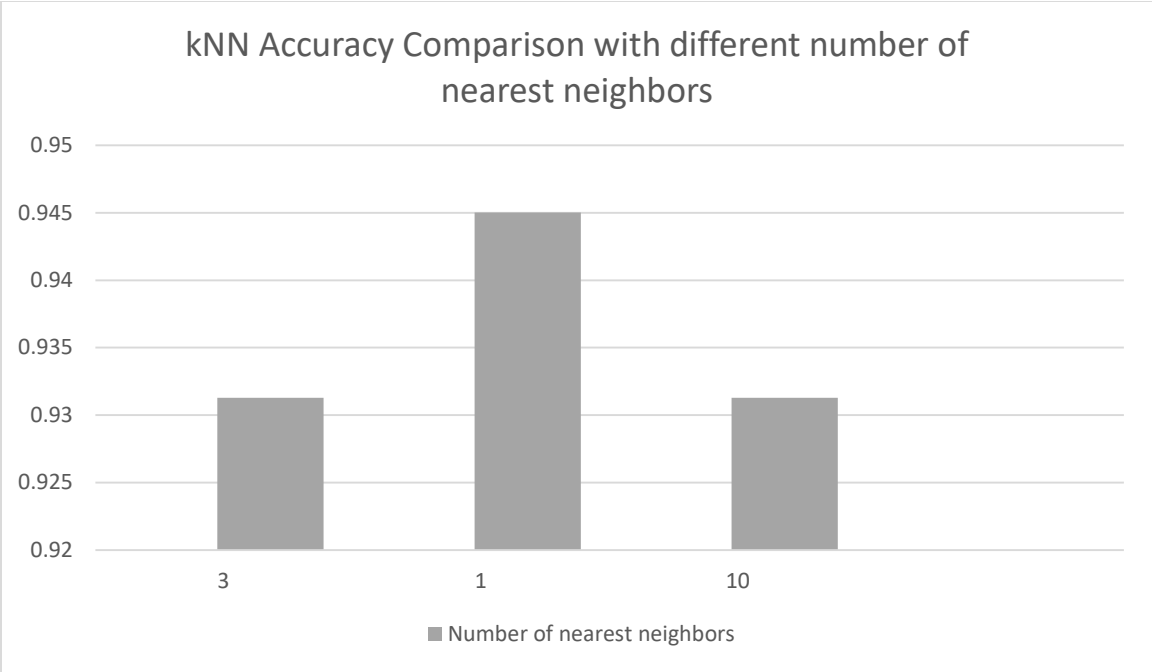
1. **Random Forest:** Initially, max_depth = 2. We tweaked this value to 25, 50 and 10]00. It is observed that as the depth of the classifier is increased, the accuracy also increases. This classifier reaches a threshold of 100 % around the depth 50. Post this, the accuracy remains constant.



2. **SVM:** Initially, the kernel used for the SVM classifier was RBF with C value as 10000. We created a combination of these values to create two new SVM classifiers. SVM1 where, kernel = RBF, C = 1. SVM2 where, kernel = linear, C=10000. It is observed that the best accuracy given by our original model. This value plummets drastically as the value of C (regularization parameter) is decreased. This is because for large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger margin separating hyperplane, even if that hyperplane misclassifies more points.



3. **kNN:** Initially, the number of nearest neighbors were chosen as 3. This value was tweaked to 1 and 10 in order to observe the accuracy changes. It is evident that the accuracy is quite high when the value of n is 1. However, the accuracy becomes stagnant when n reaches the value of 3.



FUTURE DIRECTIONS:

Improving the Random Forest Classifier:

As evidenced by our results, Random Forest classifier is the most efficient for spam filtering. Hence, we would like to explore this further by using effective boosting methods such as AdaBoost (Adaptive Boost) or Gradient Boost. Random Forest is a bagging method, which means that it trains a bunch of individual models in a parallel way, each model being trained by a random subset of data. AdaBoost trains a bunch of individual models sequentially, with each model learning from the mistakes of the previous one. In random forest, each of the decision trees gets an equal vote in the final classification of the phrase, but with AdaBoost, each of the trees are weighted according to their error rates, and their weights are considered when determining how much their vote counts towards the final label. We believe this could lead to a more adaptive and intuitive classifier.

Using More Features:

In addition to the word frequency mapping, we can use the following features as well:

- Character frequency: 6 continuous real features can be selected in which frequency of few spam indicative CHARACTERS (;, (, [, !, $ and #)
- Average length of capital letters: The average length of uninterrupted sequences of capital letters is calculated and used as feature in spam filtration
- Longest length of sequence of capital letters: The longest length of uninterrupted sequence of capital letters is calculated and used as feature in spam filtration.
- Total length of Capital letters: Total number of capital letters in the e-mail are also used as feature.

Recurrent Neural Network:

In the future, we would also like to compare our results with the Naive Bayes Classifier and the Random Forest Classifier against a Recurrent Neural Network (RNN). Unlike any of the previous models we have trained, an RNN would factor in that the sequence of the words itself can impact whether an email is spam or not. Instead of assuming that all of the given features are independent, RNNs perform the same task for every element of a sequence (list of features) and the output is dependent on previous computations. Because of this, RNNs are very well suited to a lot of language related machine learning problems. However, the nuance of this makes them much more effective when working with extremely large datasets, so it is possible that our dataset will not be enough to properly train it.

CONCLUSION:

Both the **Naïve Bayes** classifier and the **Random Forest** classifier are good choices for spam filtering. However, there are weaknesses to both approaches. A Naïve Bayes classifier uses frequencies of words to detect spam- if spammers avoid using words that are more prone to be a spam message, the classifier is weakened.

The weakness of the support vector machine is that it cannot perform well when many feature vectors are present. The complexity of the system increases drastically as inputs increase; therefore, it takes more time and memory to perform. Spam filtering in general is a tough task mainly because there aren't many public datasets available for use. The abbreviations and structure of sentences also makes spam detection difficult.

As an ML engineer, who wants to solve variety of real-world problems, it is important to realize the importance of each model and parameters for analysis (accuracy, precision and recall) The aim for the project was to develop such an intuition.

CITATION:

- Language Processing and Python: https://www.nltk.org/book/ch01.html
- Converting Words to Features with NLTK:

https://pythonprogramming.net/words-as-features-nltk-tutorial/

- NLTK Naive Bayes Classifier Training for Sentiment Analysis:

https://stackoverflow.com/questions/20827741/nltk-naivebayesclassifier-training-for-sentiment-analysis

- Understanding Random Forest Classifiers in Python:

https://www.datacamp.com/community/tutorials/random-forests-classifier-python

- Basic Ensemble Learning (Random Forest, AdaBoost, Gradient Boosting) - Explained:

https://towardsdatascience.com/basic-ensemble-learning-random-forest-adaboost-gradient-boosting-step-by-step-explained-95d49d1e2725

- Bag of Words Algorithm in Python Introduction:

http://www.insightsbot.com/blog/R8fu5/bag-of-words-algorithm-in-python-introduction