Intelligent Systems Report:

Spam Email Classification using Machine Learning

Jazer Barclay

STUDENT ID: 20837308

CI512 – INTELLIGENT SYSTEMS

20TH JANUARY 2022

Table of Contents

1. Introduction

1.1. Report aim

This report covers classification models used in machine learning and the final model used to classify spam and non-spam emails within my selected dataset for use within a business setting.

1.2. Problem outline

A rapidly growing company that corresponds with its customers via email are receiving an increasing influx of spam emails from random addresses across their entire email system. This increase in spam has resulted in a significant loss of revenue due to staff spending more time sorting emails rather than replying. In addition, many customers are unhappy with the response times and are looking to other, more responsive businesses.

1.3. My solution

My solution will classify new emails as either spam or not spam using a classification algorithm which can help filter out undesirable emails automatically.

1.4. Performance measures

For this task, I will select an algorithm that prioritises accuracy and precision to prevent false positives, which could result in legitimate customers not being seen, or false negatives where a significant number of spam emails will remain in staff inboxes.

I aim to be filtering at least 80% of spam while retaining over 95% of non-spam emails to ensure minimal loss in productivity for all staff members.

2. Test data

Due to the nature of emails being words of varying length and nature, a dataset that can apply to emails in a business setting and accommodate new emails received is vital.

I have selected a sizable dataset containing just over 5000 datapoints with 3000 features for this task. This dataset is from an existing business' mail server, which has each datapoint classified as either spam or non-spam as its final feature containing either a 0 for not spam or a 1 for spam.

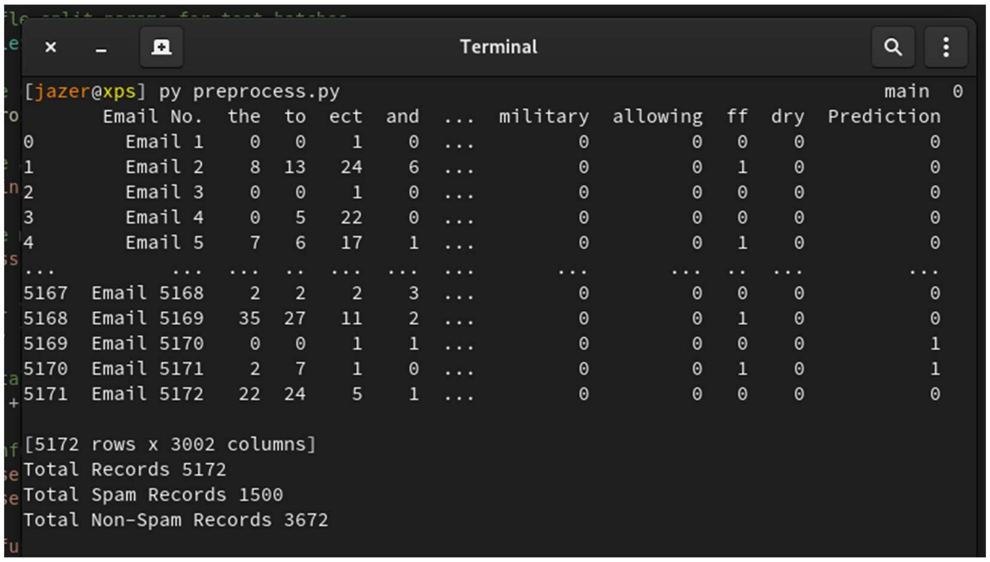
This dataset has been cleaned and can be applied to the machine learning algorithms that we have covered over the past semester; however, selecting the best model will need careful testing and consideration of its environmental variables, as will be covered in the next section.

For all of my testing, I will be using a random seed to ensure reproduceable results with each test running multiple times within its total runtime. This will ensure multiple different results while maintaining that results can be reproduced if required. All results will be stored as both its standard output to the console into a text document and values extracted into an excel spreadsheet where graphs can be created to clearly show how each model performed.

3. Data pre-processing

3.1. Ratio of spam to non-spam records

We can check the balance of spam vs non-spam emails by checking totalling the last column in the dataset. Since each spam record will be labelled with a 1, if we total the column, we get the total number of spam email records.

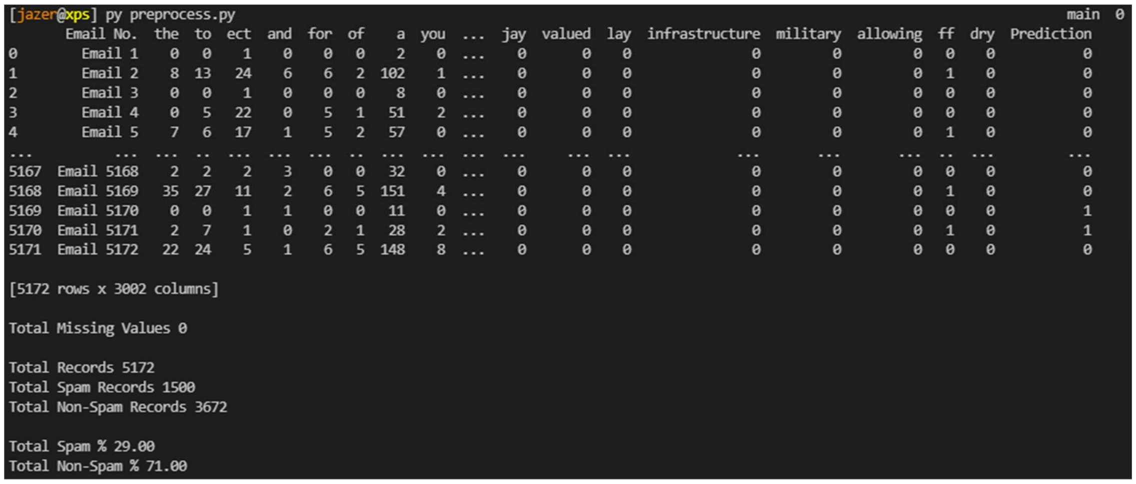


```
[jazer@xps] py preprocess.py                                                 main  0
        Email No.  the  to  ect  and  ...  military  allowing  ff  dry  Prediction
0         Email 1    0   0    1    0  ...         0         0   0    0           0
1         Email 2    8  13   24    6  ...         0         0   1    0           0
2         Email 3    0   0    1    0  ...         0         0   0    0           0
3         Email 4    0   5   22    0  ...         0         0   0    0           0
4         Email 5    7   6   17    1  ...         0         0   1    0           0
...           ...  ...  ..  ...  ...  ...       ...       ...  ..  ...         ...
5167   Email 5168    2   2    2    3  ...         0         0   0    0           0
5168   Email 5169   35  27   11    2  ...         0         0   1    0           0
5169   Email 5170    0   0    1    1  ...         0         0   0    0           1
5170   Email 5171    2   7    1    0  ...         0         0   1    0           1
5171   Email 5172   22  24    5    1  ...         0         0   0    0           0

[5172 rows x 3002 columns]
Total Records 5172
Total Spam Records 1500
Total Non-Spam Records 3672
```

3.2. Missing values

To work out if we have any missing values, we can use the isnull() function which is a part of the pandas library. This will return the total number of null values in each feature of our dataset. By summing the totals of each feature, we get the total count of missing values of our entire dataset. The results can be seen as "Total Missing Values" in the following image.

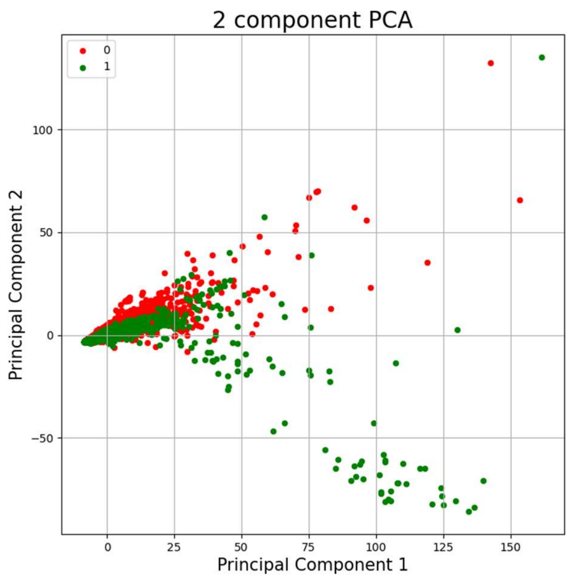
Total Missing Values and Record Split Output



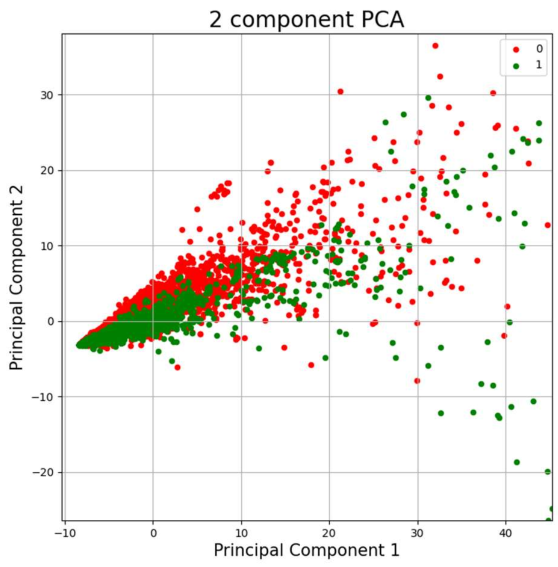
```
[jazer@xps] py preprocess.py                                                                                          main  0
        Email No.  the  to  ect  and  for  of    a  you  ...  jay  valued  lay  infrastructure  military  allowing  ff  dry  Prediction
0         Email 1    0   0    1    0    0   0    2    0  ...    0       0    0               0         0         0   0    0           0
1         Email 2    8  13   24    6    6   2  102    1  ...    0       0    0               0         0         0   1    0           0
2         Email 3    0   0    1    0    0   0    8    0  ...    0       0    0               0         0         0   0    0           0
3         Email 4    0   5   22    0    5   1   51    2  ...    0       0    0               0         0         0   0    0           0
4         Email 5    7   6   17    1    5   2   57    0  ...    0       0    0               0         0         0   1    0           0
...           ...  ...  ..  ...  ...  ...  ..  ...  ...  ...  ...     ...  ...             ...       ...       ...  ..  ...         ...
5167   Email 5168    2   2    2    3    0   0   32    0  ...    0       0    0               0         0         0   0    0           0
5168   Email 5169   35  27   11    2    6   5  151    4  ...    0       0    0               0         0         0   1    0           0
5169   Email 5170    0   0    1    1    0   0   11    0  ...    0       0    0               0         0         0   0    0           1
5170   Email 5171    2   7    1    0    2   1   28    2  ...    0       0    0               0         0         0   1    0           1
5171   Email 5172   22  24    5    1    6   5  148    8  ...    0       0    0               0         0         0   0    0           0

[5172 rows x 3002 columns]

Total Missing Values 0

Total Records 5172
Total Spam Records 1500
Total Non-Spam Records 3672

Total Spam % 29.00
Total Non-Spam % 71.00
```

3.3. Visualisation using Principal Component Analysis

The data can be visualised in a single 2 dimensional graph using a method called Principal Component Analysis. By first normalising the data and using the PCA pre-processing library from sklearn, we can create a copy of the dataset that has 2 features and the label at the end. Using this we can graph the data as a simple graph using the matplotlib library. This was the following graph after having done so with my dataset.



The chart indicates that the spam and non-spam have a close clustering together which then disperse in mostly different directions. A closer look at the cluster reveals that this divergence occurs at the smaller scale too.

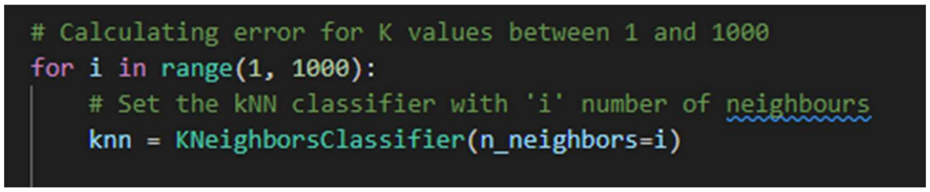


4. K-nearest neighbour

I started with the k-nearest neighbour model. K-nearest neighbour, or 'kNN' for short, is a supervised learning algorithm that determines an unlabelled datapoint's classification based on the features found in the datapoint. It works by using known datapoints that are close in features to determine its label. On two or three dimensional graphs, we would see if the new datapoint is relative to a group that we could say is of the same type. With 3000 features, it would be difficult to graph; however, the concept is the same.

To pick the best k value, I ran a test against every k value from 1 to 1000 and recorded each value's returned error rate. Then, I graphed this using the 'matplotlib' library and got the following result.

Iterated 'k' value as n_neighbors using the range 'i' from 1 to 100



```
# Calculating error for K values between 1 and 1000
for i in range(1, 1000):
    # Set the kNN classifier with 'i' number of neighbours
    knn = KNeighborsClassifier(n_neighbors=i)
```

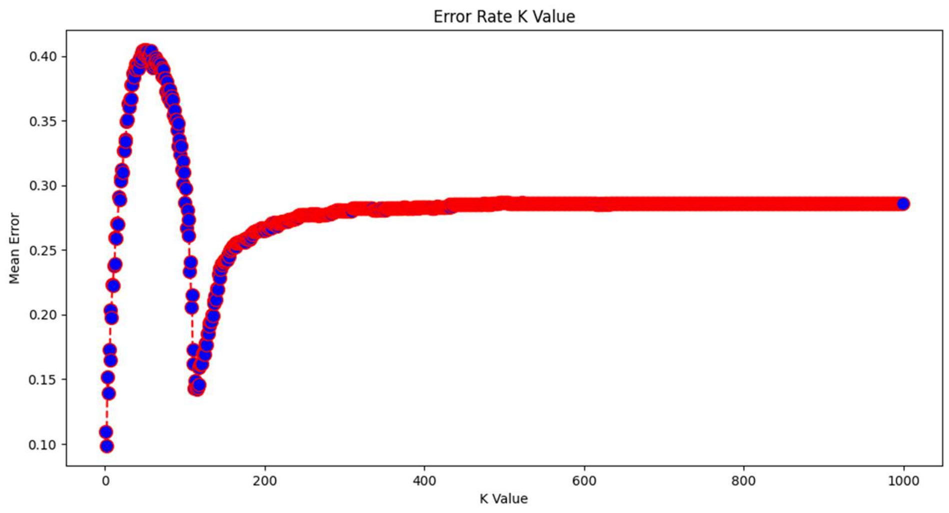


Figure 1.1 – Test results for k values 1 to 1000 vs the mean error rate graphed using matplotlib (lower is better)

The results were conclusive that the higher values would yield more errors than numbers lower than 10. After running a test against values 1 to 10 for a more detailed view, I selected a 'k' value of 3 for the final result.

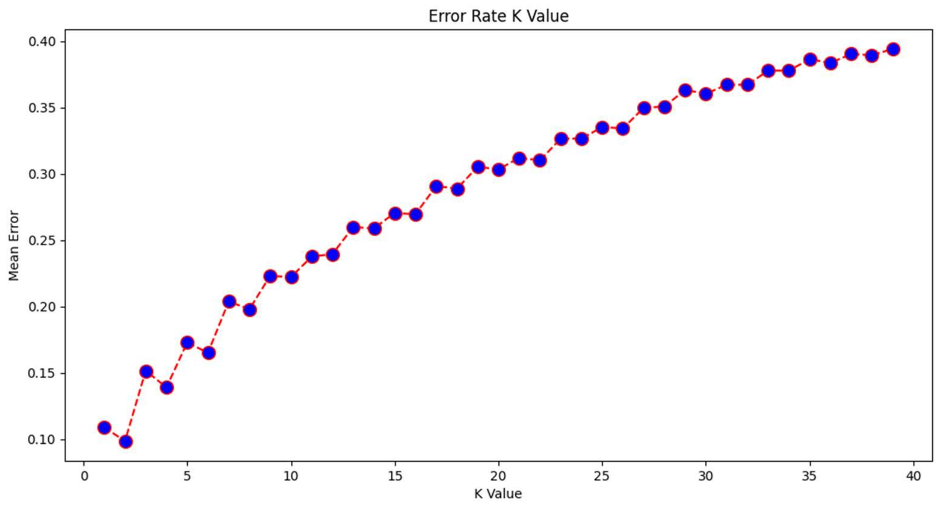
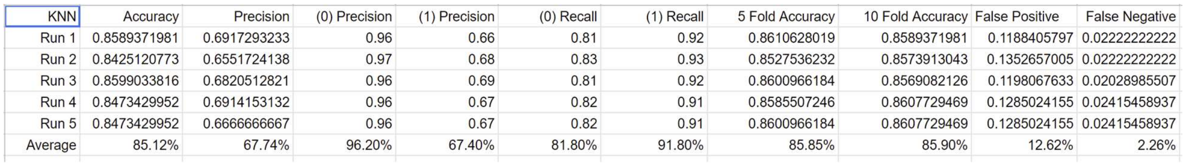


Figure 1.2 - Test results for k values from 1 to 40 using same results (lower is better)

This model was then run five times using 5 and 10 fold cross-validation. Finally, the results were recorded into an excel spreadsheet for average accuracy and precision across the multiple tests. With data for our first model, we can move on to other models to test which performs better. A snapshot of the results is shown below.

KNN Test Results (5 tests with 5 and 10 fold shuffle split)



KNN	Accuracy	Precision	(0) Precision	(1) Precision	(0) Recall	(1) Recall	5 Fold Accuracy	10 Fold Accuracy	False Positive	False Negative
Run 1	0.8589371981	0.6917293233	0.96	0.66	0.81	0.92	0.8610628019	0.8589371981	0.1188405797	0.02222222222
Run 2	0.8425120773	0.6551724138	0.97	0.68	0.83	0.93	0.8527536232	0.8573913043	0.1352657005	0.02222222222
Run 3	0.8599033816	0.6820512821	0.96	0.69	0.81	0.92	0.8600966184	0.8569082126	0.1198067633	0.02028985507
Run 4	0.8473429952	0.6914153132	0.96	0.67	0.82	0.91	0.8585507246	0.8607729469	0.1285024155	0.02415458937
Run 5	0.8473429952	0.6666666667	0.96	0.67	0.82	0.91	0.8600966184	0.8607729469	0.1285024155	0.02415458937
Average	85.12%	67.74%	96.20%	67.40%	81.80%	91.80%	85.85%	85.90%	12.62%	2.26%

Figure 1.3 – Recorded data from running k-nearest neighbour with the most optimal value of 3

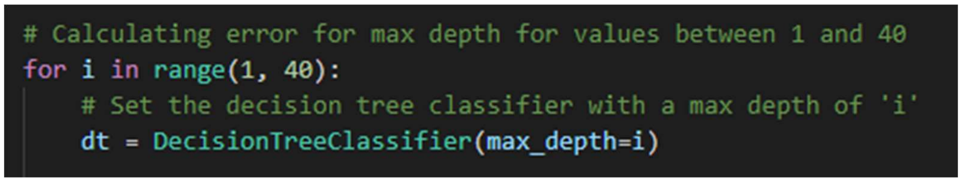
Once we have more data from other models, we will be able to compare these results and graph the difference between their averages.

5. Decision tree

Next, a decision tree is used. A decision tree works by splitting data based on a calculated entropy of a datapoint's features. This splitting of the dataset results in decisions that channel the new datapoint to a classification.

The critical variable here is the maximum depth the decision tree uses. By testing depths from 1 to 40, we can see the following graph chart the error rate as the depth increases.

Setting the max depth using ‘i’ in range 1 to 40



```
# Calculating error for max depth for values between 1 and 40
for i in range(1, 40):
    # Set the decision tree classifier with a max depth of 'i'
    dt = DecisionTreeClassifier(max_depth=i)
```

Mean Error vs Max Depth (lower is better)

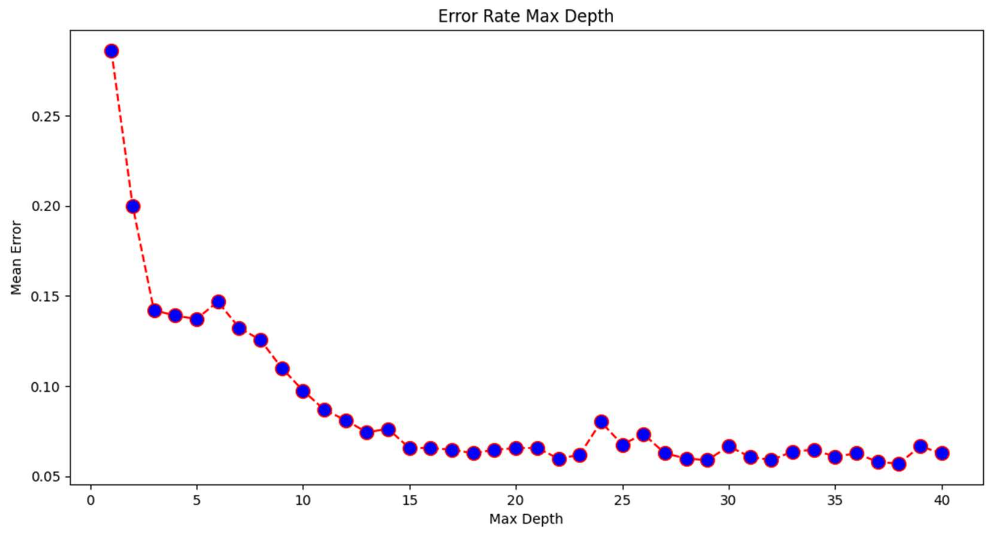


Figure 2.1 – Decision tree test results with depths from 1 to 40 vs the mean error rate graphed (lower is better)

It is clear that the deeper the tree, the more accurate our results; however, the duration also increases as depth increases. In light of this, the depth variable was left unset to allow the tree to go as deep as required to gain the best accuracy possible. The time taken to classify is not as important as the accuracy of a classification and, provided it doesn't take more than 1 minute, is within the upper bounds.

These results compared to the k-nearest neighbour is shocking. Although running this model against the dataset took far longer, the accuracy and precision are much higher than the k-nearest neighbour model. This difference can be seen more visually in the following graphs.

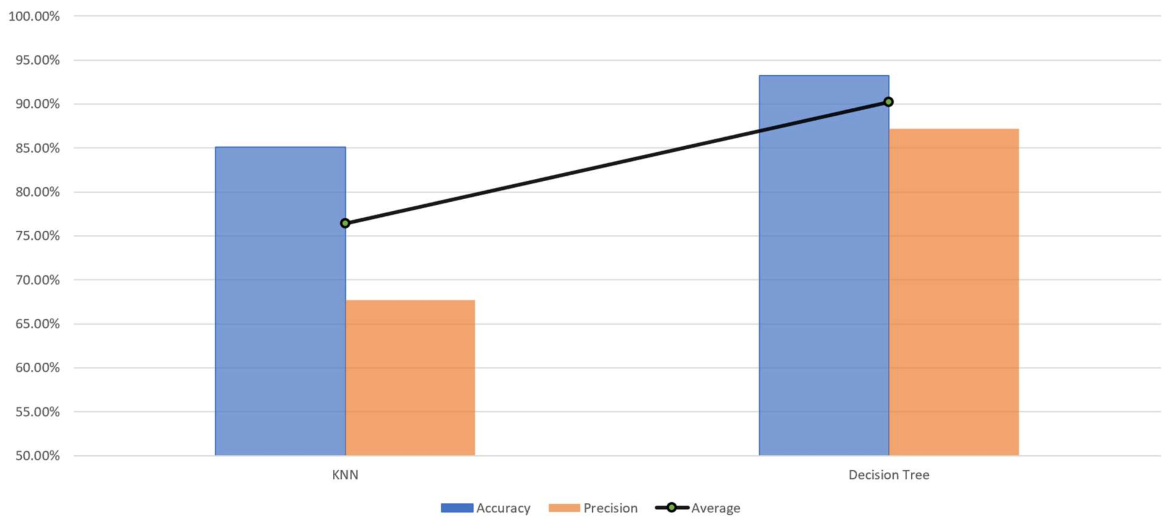


Figure 2.2 – Graph comparing k-nearest neighbour vs decision tree for accuracy and precision (higher is better)

We can also conclude that a decision tree yields far greater precision for positive and negative results, meaning fewer false positives where legitimate mail would be classified as spam. This is far more desirable for our given business use case.

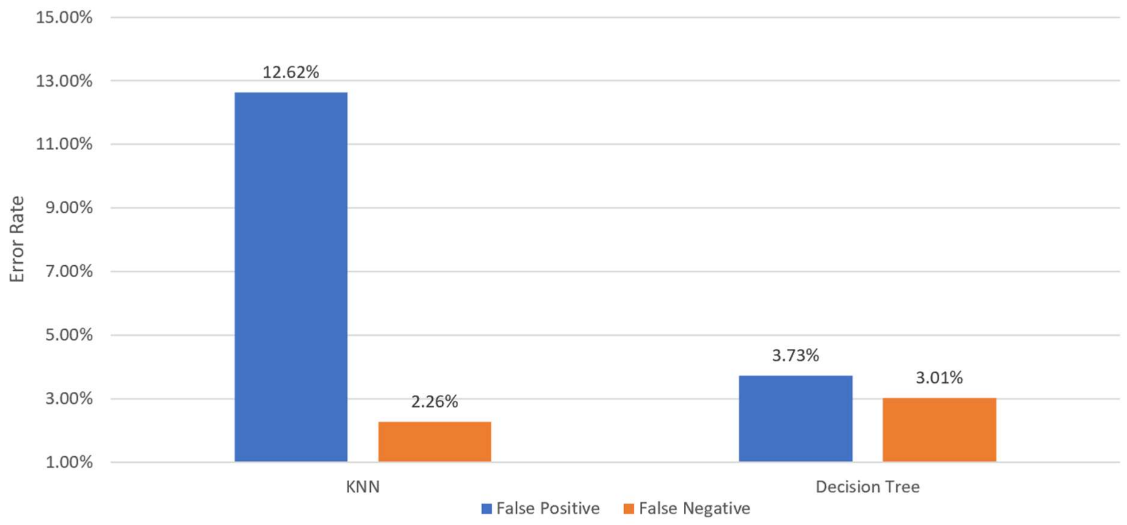


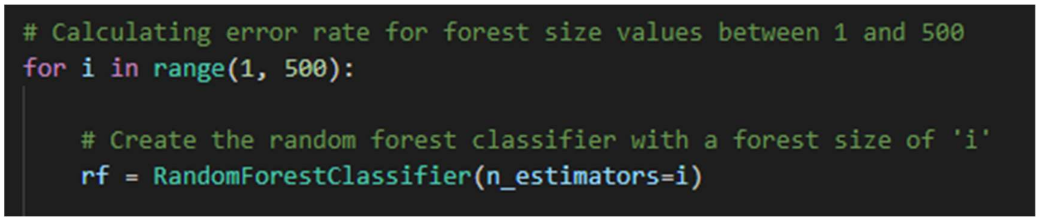
Figure 2.3 – K-nearest neighbour vs decision tree graph of false positives and false negatives (lower is better)

6. Random forest

Expanding on decision trees, we take a look at using a random forest model. Random forests use a collection of decision trees that work on subsets of the data that combine to help classify a given datapoint.

Much like the depth of the decision tree can impact performance, so too does the forest size of a random forest. Therefore, a test was run using the random forest classifier with a range of forest sizes from 1 to 500 to see the forest size's impact against the error rate. This test yielded the following graph showing the error rate vs the forest size.

Setting forest size with n_estimators using 'i' with values 1 to 500



```
# Calculating error rate for forest size values between 1 and 500
for i in range(1, 500):

    # Create the random forest classifier with a forest size of 'i'
    rf = RandomForestClassifier(n_estimators=i)
```

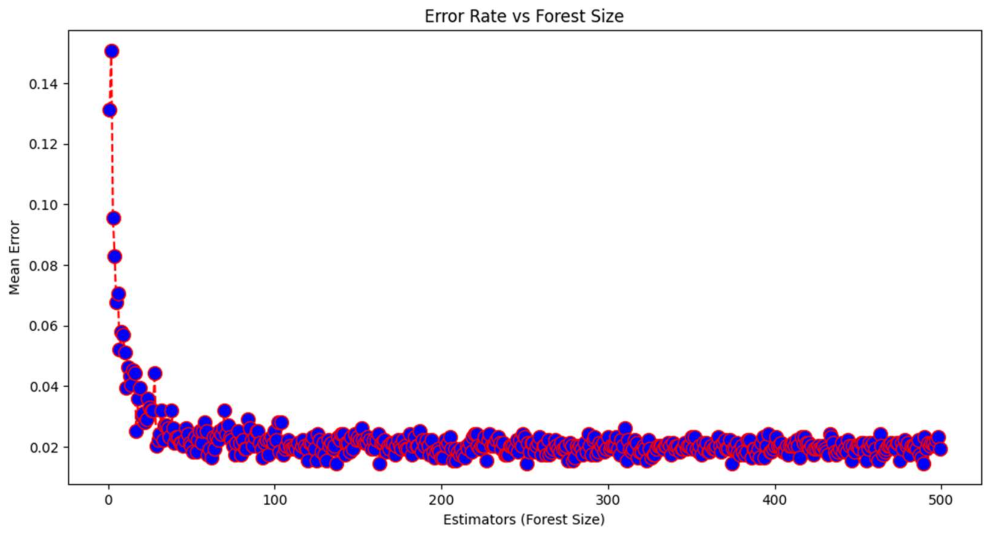


Figure 3.1 - Random forest error rate using forest sizes from 1 to 500 graphed (Lower Is Better)

This graph shows that the error rate begins very high and reduces as the forest size increases. However, it also shows that forest sizes greater than 100 negatively impact the time taken and do little to reduce the error rate. By zooming in on the first 100 values, we can see the most optimal values at a smaller forest size.

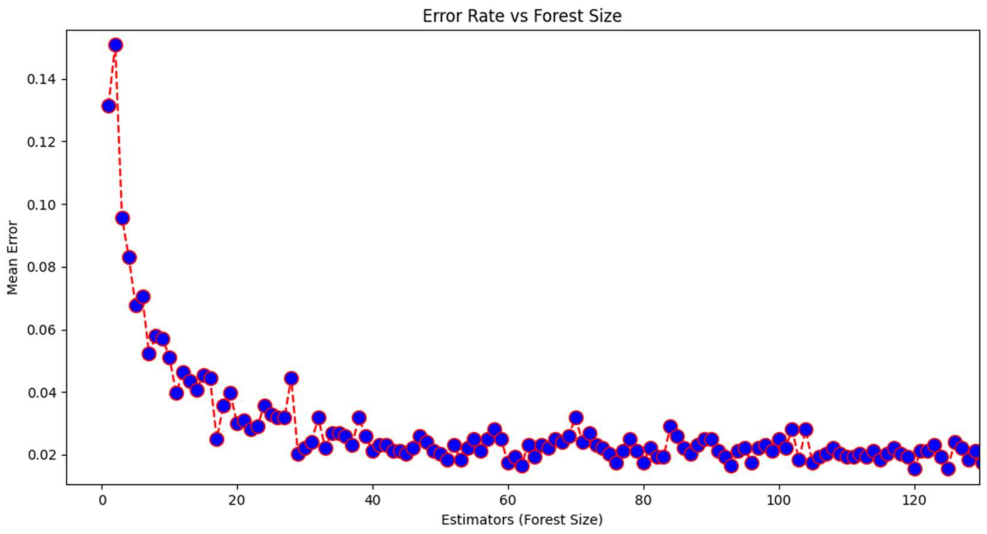


Figure 3.2 – Random forest error rate using forest sizes from 1 to 100 graphed (lower is better)

By using the value 30 we optimise for the best accuracy and precision at a reasonable speed. The results for accuracy and precision compared to both prior models is shown in the following graph.

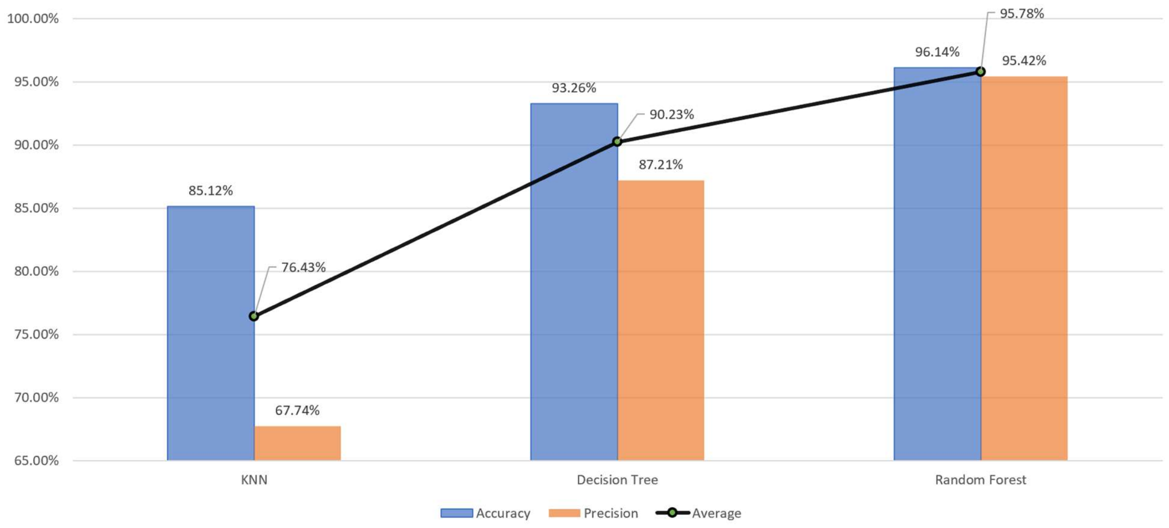


Figure 3.2 – Comparison of all classification models against their accuracy and precision averages (higher is better)

Not only is the accuracy and precision higher using a random forest, we also have a stark difference in false positives and false negatives. In the next chart we can see a clear distinction between the previous two models and the random forest with higher, however mediocre, false negatives and far lower false positives. This is far more desirable in a system where false positives will result in a loss of income however false negatives will impact the staff members little in comparison.

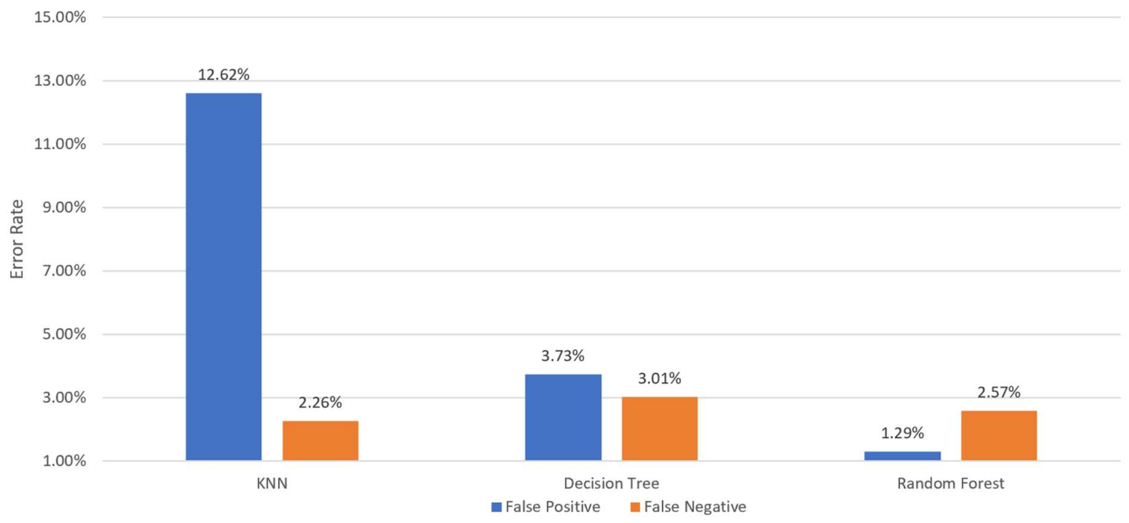


Figure 3.3 – False positives vs false negatives on all tested models (lower is better)

7. Conclusion

My findings show that k-nearest neighbour works good for a first pass of the given datapoint as it is fast to classify and uses very little compute power to perform however has a lacking accuracy and precision compared to other classification algorithms.

Decision trees also perform well with a good balance between speed and accuracy however does result in a high false positive error rate which has a greater negative impact on the business in comparison with false negatives.

Random forests work best for our use case as time is not of grave concern since its accuracy and precision is greater than a decision tree with the added benefit of a low 1.29% false positives. This is a 34% improvement over the 3.73% false positive rate of the decision tree and a 15% increase in false negatives from 3.01% to 2.57%.

To improve on this further, a snapshot of real data from within the company in question would give far greater insights where weights could be added to features most commonly found in spam and non-spam related emails which could be used in conjunction with a stacking regressor. Furthermore, adding more data to this existing dataset would greatly enhance its potential when deployed.

References

Kaggle.com. (n.d.). Email Spam Classification Dataset CSV. [online] Available at: https://www.kaggle.com/balaka18/email-spam-classification-dataset-csv/version/1 [Accessed 11 Nov. 2021].

Scikit-learn.org. (2019). Scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. [online] Available at: https://scikit-learn.org/.

Matplotlib (2012). Matplotlib: Python plotting — Matplotlib 3.1.1 documentation. [online] Matplotlib.org. Available at: https://matplotlib.org/.

Russell, P. (2010). ARTIFICIAL INTELLIGENCE : a modern approach, global edition. S.L.: Pearson Education Limited.