Christopher Sullivan

Professor O’Neill

CS-390-02

2/27/23

An Analysis of Supervised Learning Techniques

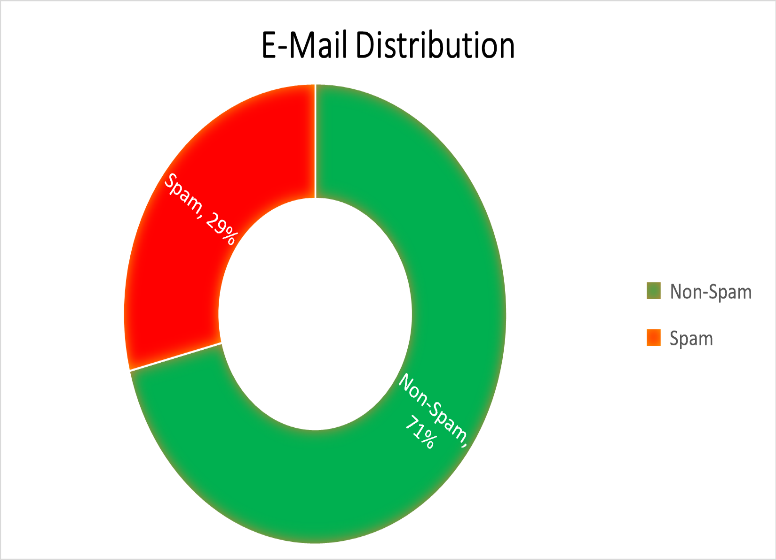
**Introduction**

***Techniques & Data:***

Within this study, machine learning analysis was conducted on two unique datasets representing binary classification problems. Specifically, performance tests were conducted on the *sci-kit learn* library’s decision tree, neural network, k-nearest neighbor, and random forest learners given these datasets. The classification variable for the first dataset ‘emails.csv’ (Biswas) is whether or not a given e-mail is labelled as “spam”. Within this first dataset, there exist 5,172 instances of e-mails, each with 3,000 recorded features. These features are the number of occurrences per instance of the 3,000 most common “words” (letter sequences), collected from the set of e-mails itself. Thus, since this data is collected from itself, we can consider it to be mostly free of missing values. However, in this case, there is a possibility that the input is not strongly correlated to the output, which may affect the results. This dataset provides a classification problem that is interesting for multiple reasons. To begin, there is the simple practicality of e-mail filtering; this is something that must be conducted by any industry e-mail provider, and thus it presents a real-world application for machine learning in this context. Additionally, since the dataset has thousands of instances and attributes, it can showcase the different results that these learning techniques produce when faced with large amounts of data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Email No.** | **‘the’** | **‘to’** | **‘ect’** | **‘and’** |
| Email 1 | 0 | 0 | 1 | 0 |
| Email 2 | 8 | 13 | 24 | 6 |
| Email 3 | 0 | 0 | 1 | 0 |
| Email 4 | 0 | 5 | 22 | 0 |
| Email 5 | 7 | 6 | 17 | 1 |

|  |
| --- |
| **class\_val (y)** |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |



**|** **Figure 1.2**: Small Sample of the E-Mails Dataset

**Figure 1.1**: Distribution of the E-Mail Dataset by Output (%)

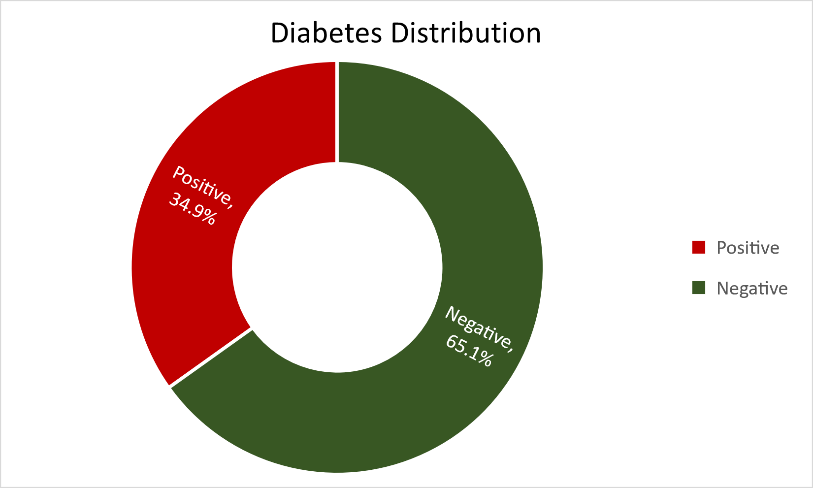
Additionally, the classification variable of the second dataset diabetes.csv (Akturk) is whether or not a given patient tested positive for diabetes in a study of Pima Indian women. This set of data is comprised of 768 instances of these women, with 8 features related to their blood, age, and overall health recorded over each instance. In this case, the input and output have a stronger scientific correlation, which may aid the learners in coming to more correct conclusions during the test phase. However, something to note about the dataset is its high concentration of missing values. Across all of the features, there are approximately 652 missing data points out of 6,144 total. This represents about ~10.5% of the data pool which is completely absent from the set.

**Graphical user interface, text, application, chat or text message

Description automatically generated**

**Figure 1.3**: Missing Value Totals (Akturk)

The diabetes dataset creates an interesting classification problem for machine learners; beyond its obvious real-world application, this specific dataset can showcase how different techniques are able to overcome missing data and generalize beyond the training data.



**Figure 1.4**: Patient Outcome Distribution by %

For the purpose of testing the aforementioned learning techniques’ robustness and adaptability, neither set of data was altered from the state in which it was obtained apart from the output column labels. Subsequently, each learning technique was subjected to cross-validation over a set of either 5 or 10 folds from the data, so as to hopefully circumvent overfitting. Furthermore, both sets contain only discrete input and outputs, which is something to note in the analysis of technique efficiency.

**Neural Networks:**

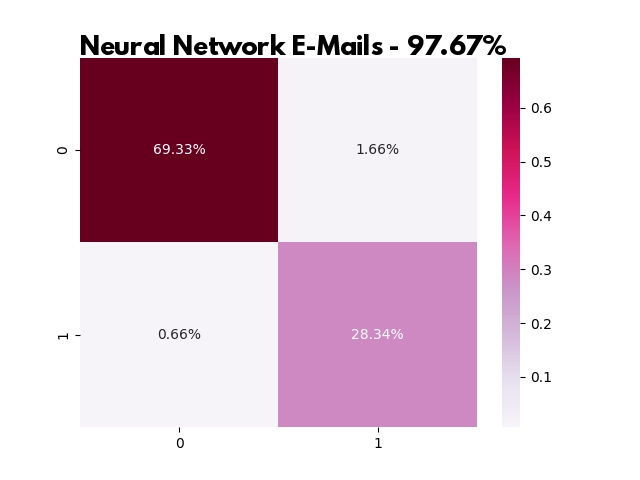
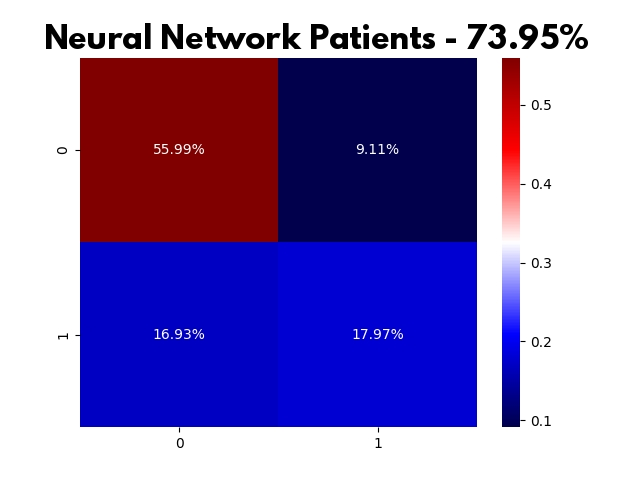
Overall, the results produced by neural networks varied across the two datasets. While a neural network was responsible for the most accurate e-mail classifier at 97.68% average, the technique conversely produced the least accurate test results when classifying the diabetes patients. The neural network which was used to classify the patients reported only 73.95% average accuracy after hyper-parameter tuning, which demonstrates weak learning, but is far from optimal. Due to the non-linear, numerical representation format of the e-mail words dataset as well as its large cardinality, it is not entirely surprising that a neural network could classify the e-mails so well. Neural networks tend to efficiently classify larger, complex data which can be represented numerically so it is plausible given the domain. Additionally, the *MLPClassifier* (Multi-Level Perceptron)from *sci-kit learn* used to represent the e-mail neural network gave its optimal prediction performance without any parameter tuning. The only noticeable performance difference was observed when the max iteration count was changed from its default of 200, to lower values. As the graph below shows, the training time increases further as the number of iterations does. However, at only 50 iterations, the classifier reports that the stochastic gradient function used to determine the weights of the perceptrons repeatedly finishes without converging on a result, and this issue only becomes more prominent as iterations decrease.

Chart, line chart

Description automatically generated

**Figure 2.1**: E-mail Neural Net Training Time as a Function of maximum iterations allowed. Appears to be a logarithmic curve.

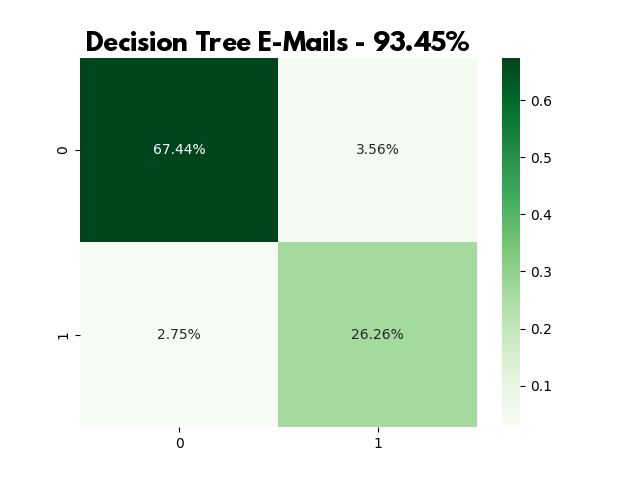
Due to this, iteration count was set to 75 in order to optimize training time to accuracy ratio, with an accuracy of 97.68% and a training time of 31.1 seconds. The average training accuracy of this neural network across all runs was 99.98%. This appears to be overfitting, but does not reflect in the test results. Also note the *random\_state* attribute, which is set for many of the learners. The purpose of this is simply to make the randomization of numbers in the calculations static so that results do not change across function calls. Contrary to the high accuracy of the e-mail classifier, the diabetes neural network reported only ~74% accuracy when tuned heavily. This classifier’s accuracy benefitted from a boost in the initial learning rate, as increasing it from .001 to .04 gave a percentage increase of about 3.5%: from 68.2% to 71.7%. I believe this improvement occurs because as the learning rate increases, so does the weight adjustment factor. My theory is that this allows the neural net to reach outputs earlier than it would naturally on the instances with missing values, and thus it classifies a portion of these correctly where it typically would not. Going further, I found 67 to be the optimal number of internal node layers by simply tweaking the value from the default in ranges at a time until I closed in on the most efficient learner I could: 73.95% average accuracy with a training accuracy of 74.5% and training time of 0.13 seconds. In addition to accuracy, this gave a small step up in speed from the basic *MLPClassifier* for patients, which originally trained at an average of 0.21 seconds. As a final note, the neural net which classified patients was the only patients learner to benefit from 10-fold cross-validation rather than 5. (As for the e-mails set, each learner achieved a higher accuracy rate using 10-fold validation.) In sum, neural networks appeared to be extremely efficient when passed the emails dataset, but not as accurate when it came to the diabetes patients. However, this may well be explained by the properties of the datasets themselves, and not the learning technique.



**Figures 2.2**, **2.3**: Neural Net Confusion Matrices where (1,1) and (0,0) represent correct classifications.

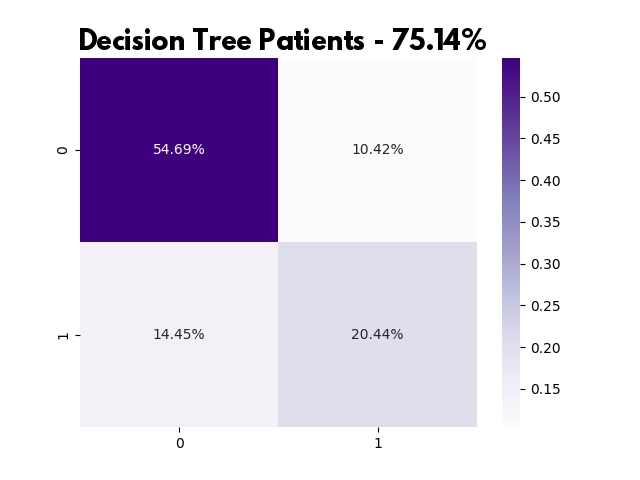
*\*Note that the algorithms used to calculate the scores and confusion matrices slightly differ, so percents may vary by decimal margins.*

**Decision Trees:**

 Following neural networks, the datasets were passed into decision tree classifiers for further analysis. To begin, a base decision tree was passed the e-mails dataset. After fitting and scoring, the un-tuned decision tree returned an average test accuracy score of 92.3% derived from 10 cross-validation runs. In addition, this tree trained at an average of 0.95 seconds with 100% training accuracy. Again, as with 5 other learners in this study, training results are presented as, on average, perfect over cross-validation. While this could certainly be a sign of overfitting in these models, I believe that, at least to some degree, something deeper is at play. Although it is evident that the models are no doubt reliant on the training sets, the high test accuracies lead me to believe that these numbers are due in part to the massive amount of features present as well as the strength of these learning functions provided by *sci-kit learn*. Nevertheless, the perfect training scores are of note, and to be considered in the overall comparison of learners. Following exhaustive parameter tuning attempts, the most efficient decision tree e-mail classifier reported an average test accuracy of 93.45%, with a training time of 1.3 seconds (+ 0.35). While the average training time increased fractionally, it is worth it in order to gain a percentage of accuracy. To commence tuning, I employed many methods of tree pruning such as minimizing leaf nodes, the number of features, and reducing tree depth. However, despite the theoretical gain from such methods, no significant gain was reported in accuracy testing. For the most part, the base tree was the most efficient. I attempted to weight the emails.csv file by creating a python dictionary which contained {key : value} pairs {0 → 2999: 3000 → 1}, working under the assumption that the words were in descending order of frequency throughout the set. However, this produced negligible results with zero or negative accuracy gain depending on the run. These results led me to believe that the dataset was not ordered as I had thought. Thus, I toggled the tree parameter ‘*class\_weight*’ to the ‘*balanced*’ setting in order to account for this. In turn, the e-mail tree’s average raised to its’ final optimization of 93.45%.

**Figure 3.1**: E-Mail Decision Tree - Confusion Matrix

I theorize that this was effective due to the nature of the data. By default, *class\_weight* for each column of data is a constant of 1. However, the ‘*balanced’* option sets each class’ weight inversely proportionate to its occurrence distribution throughout the set. I believe that this accounted for the frequency of word occurrences more accurately than my method due to the data format, which resulted in the ~1.25% accuracy gain when classifying e-mails.

 Beyond the decision tree responsible for classifying e-mails, the patients tree also provided interesting results. The final version of this classifier reported an average test score of 75.14%, with 83.46% training accuracy and an average training time of 0.00 seconds, before and after pruning.

**Figure 3.2**: Diabetes Decision Tree - Confusion Matrix

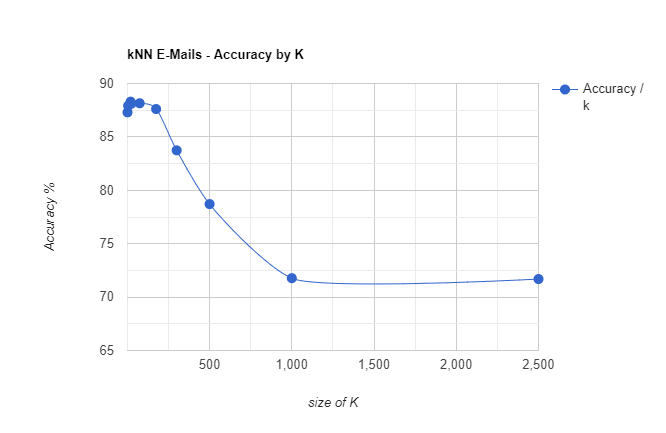
At first, my thoughts were that the e-mail tree would respond better to conventional pruning, as it contained so many instances and features that this seemed logical in terms of increasing accuracy. However, in practice, this was not the case as traditional pruning did not affect the e-mail tree, but rather gave the patient classifier an accuracy boost of about 2.5% from a baseline of 72.7% to a final 75.14%. For this tree, I limited the maximum depth to a value of 5, and set the total amount of leaf nodes to 21. While finding both of these values certainly consisted of much trial and error, my starting idea was to limit the tree to ~log2(*features*).

As the classifier again reports a somewhat lackluster accuracy score for the diabetes dataset, it appears that there is a pattern between the two learners across our data. For both sets of classifiers, despite tuning, the average e-mail accuracy is in the 90-100 percent range, and the average patient score lies in the mid-70 percentage range. This leads me to believe that the data has its own quirks and traits which limit or aid in the classification of the subjects. However, although the accuracy ranges are similar across the datasets, the separate learning methods continue to showcase how they differ through their varying times and scores. Between decision trees and neural networks, it is apparent that decision trees fare slightly better with the smaller dataset containing more ‘holes’, while neural networks excel at classifying the larger dataset. Given the nature of these learners’ biases and typical use-cases, this was to be expected, and helps validate the claim that decision trees can be used to handle missing datapoints.

**k-Nearest Neighbor:**

Of the set of four classifiers, k-Nearest Neighbor was the least efficient at classifying e-mails, with an average accuracy score of 88.3%. This e-mail classifier also reported a training accuracy of 100% with an average time of 0.18 seconds, both prior to and following tuning. Using this algorithm, the highest chance of overfitting was recognized. Since the k-Nearest Neighbors training accuracy for not only e-mails but diabetes were both 100%, considering the test scores, it is likely that these two models suffered from overfitting during training.

However, something of note was that by tuning the algorithm behind the e-mail learner, a +4.62% gain was achieved from the base learner (83.68%). Apart from making the random state constant, this was achieved in three ways: Selecting the optimal value of k, weighting the inputs, and adjusting the distance metric. The base value of k is given as 5 by *sci-kit learn,* and the optimal value for each classifier was determined by raising (and lowering) this value incrementally until the highest test accuracy was achieved.



**Figure 4.1**: Accuracy of E-Mail Classification as a function of the value k. Classifier accuracy drops off at ~71% around k = 1,000. Optimal k = 20.

As for weights, I adjusted the weight parameter to ‘*distance’,* which weights each point by its’ closeness to the given instance rather than each ki point being equal to one another. Lastly, the distance measurement: by adjusting the power parameter in the underlying Minkowski formula to (p = 1), thus using Manhattan distance over basic Euclidean distance, the accuracy increased by about a whole percentage point. This method was also employed while training the k-NN patients classifier with oddly similar results, yielding a 4.04% gain in test accuracy (72.4% → 76.44%). The patients classifier also had an average training time of zero seconds, which we can attribute to the relatively small size of the dataset.

Chart, line chart

Description automatically generated

**Figure 4.2**: Accuracy of Diabetes Classification as a function of the value k. Includes base and tuned models, drops off around k = 200 (65.10%, 67.84%)

Of all the adjustments made to *sci-kit learn*’s k-NN given the domain, the boost from the Manhattan distance switch is the one which I can not account for, and it likely would not make such a difference in a scaled environment. Nonetheless, it is evident that some deep trait of these datasets, or an error in calculation, cause Manhattan distance to produce superior test results with little known drawback. The final important metric about k-NN in this context was its run-time. Although both k-NN learners boasted relatively insignificant training times under a second, the k-NN e-mail classifier took exceptionally long to run each time the code was executed. While this is unavoidable to an extent due to the large dataset itself, it took far longer to run than any other learner, even the e-mail neural nets which took 30+ seconds each to train. This is the reason as to why base accuracy as a function of k was not recorded in this study, as it exceeded the time frame. This speaks volumes to the nature of the k-NN algorithm as a ‘lazy learning’ algorithm, which does not do any calculations until explicitly asked to.

Chart, treemap chart

Description automatically generatedChart, treemap chart

Description automatically generated

**Figures 4.3, 4.4**: k-Nearest Neighbor Classification - Confusion Matrices

**Random Forest:**

The final classification method of the four to be analyzed is the Random Forest Bagging technique. The bagging method (bootstrap aggregation) is unique from the rest in the sense that it does not possess its own inherent learning model, but rather is comprised of many individual learners, in this case decision trees. Within this study, the random forest produced the highest overall results following only slight tuning, classifying the e-mails correctly 97% of the time, and 77.87% of the patients correctly. Additionally, these models reported training times of only 2.57 seconds and 0.15 seconds respectively following tuning.

Chart, treemap chart

Description automatically generated

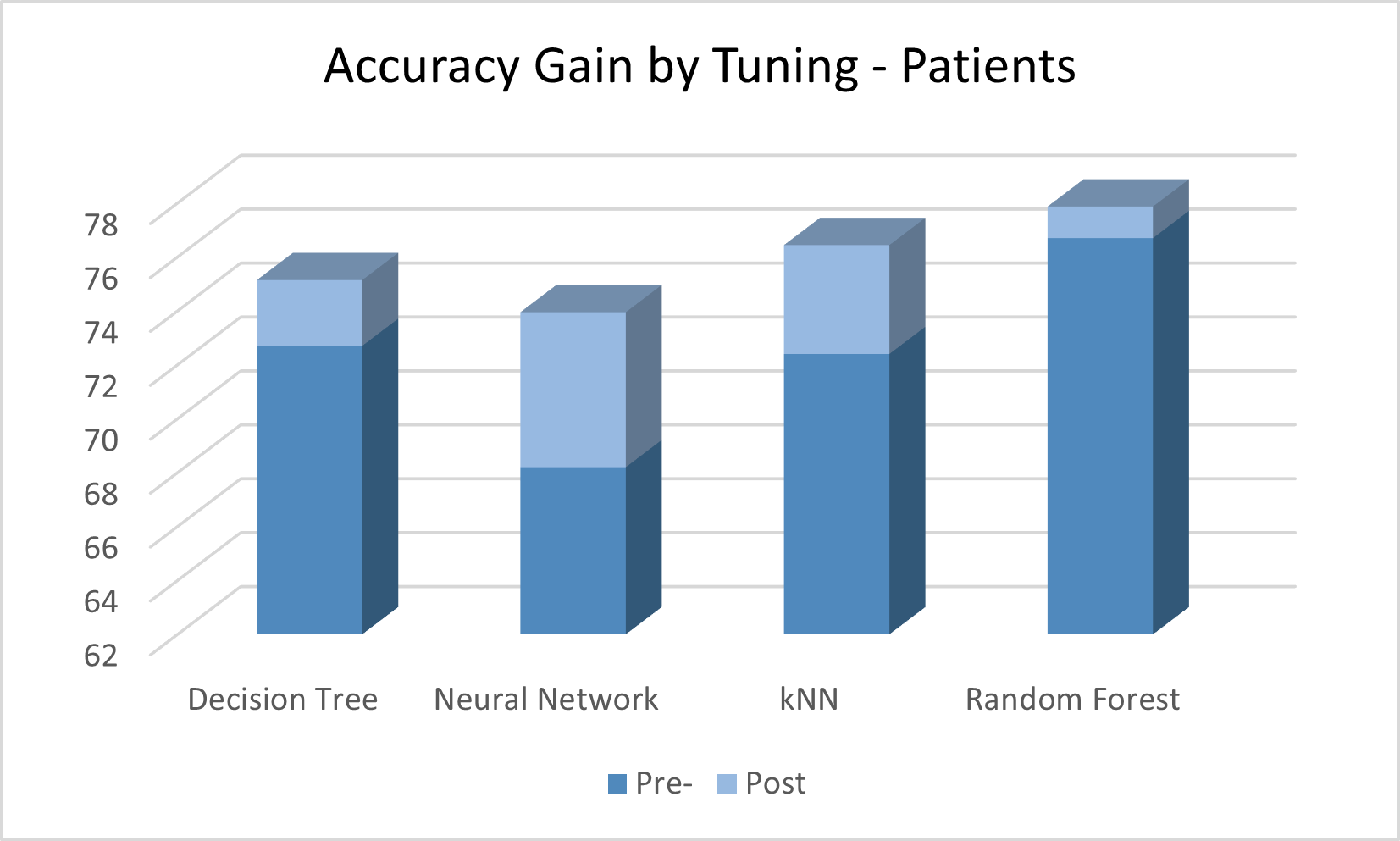
**Figure 5.1**: Random Forest E-Mails – Confusion Matrix

Chart, treemap chart

Description automatically generated As for the tuning itself, the general idea of bagging using a random forest is to cut down the amount of features considered in each tree to only a small, random subset of the features which are available; hence the name. However, in the coding portion for these problems, this method did not produce better results than full-feature consideration. In the case of the e-mail classifier, the parameters given followed exactly that of the individual e-mail decision tree. Upon attempting to further reduce the feature pool and sample size, the classifier’s test results only went down each time. This is explainable though, due to the already high test accuracy of 97.00%. As for the patients, the same was true. However, even the parameters used on the diabetes.csv decision tree did not improve this random forest, but rather harmed its accuracy. This was the only learner of the 8 total to report its best results in its base state, only requiring the random generation to be made constant for consistent results. My theory behind this is that since diabetes.csv is missing so much data and is already comparably small, removing features and tree branches would only serve to widen the data gap, thus reducing random forest accuracy. Lastly, each random forest had 100% training scores, yet still proved relatively effective.

**Figure 5.1**: Random Forest Patients – Confusion Matrix

**Conclusions:**

In brief, this study showcased the usefulness of having large datasets for machine learning, and demonstrated the differences in technique and results between the four supervised learning techniques. Below are the marked improvements of each learner following tuning.

Chart, bar chart

Description automatically generated