Christopher Sullivan

Professor O’Neill

CS-390-02

2/27/23

An Analysis of Supervised Learning Techniques

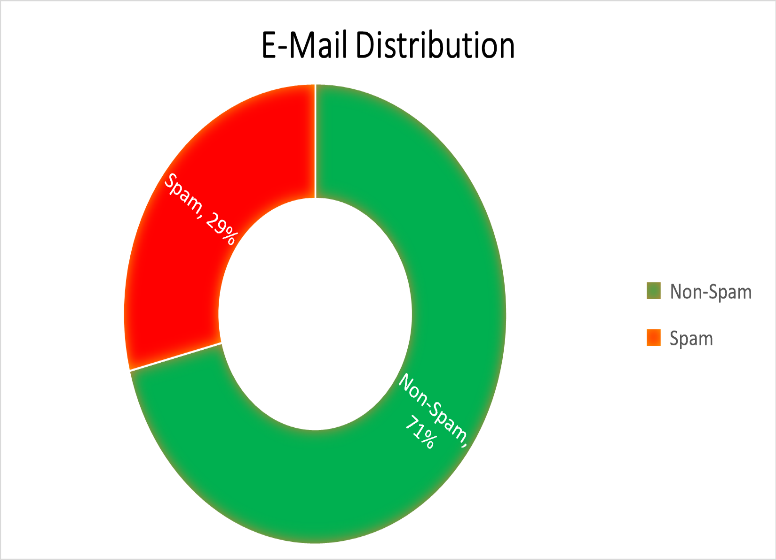
**Introduction**

***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 trees, neural networks, k-nearest neighbor algorithm, and random forests which were given these datasets. The classification variable for the first dataset(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 (%)

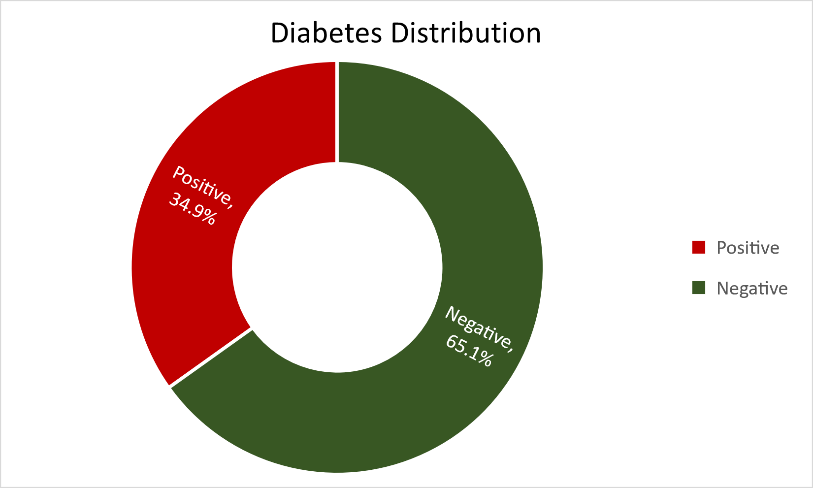
Additionally, the classification variable of the second dataset is whether or not a given patient tested positive for diabetes in a study of Pima Indian women. This set of data is comprised of 768 instances of these women, with 8 features related to their blood, age, and overall health recorded over 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:**

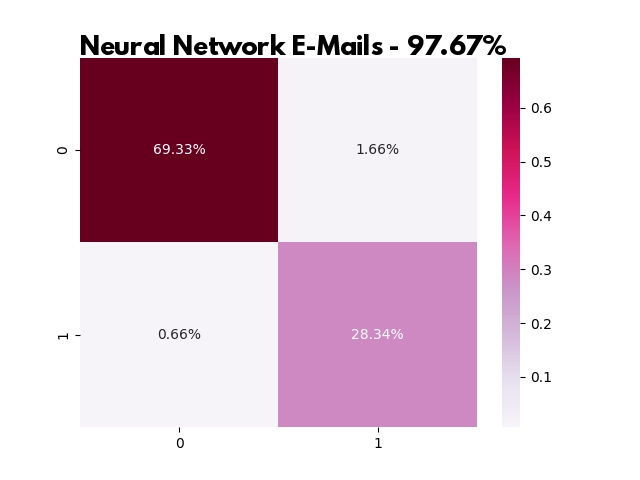
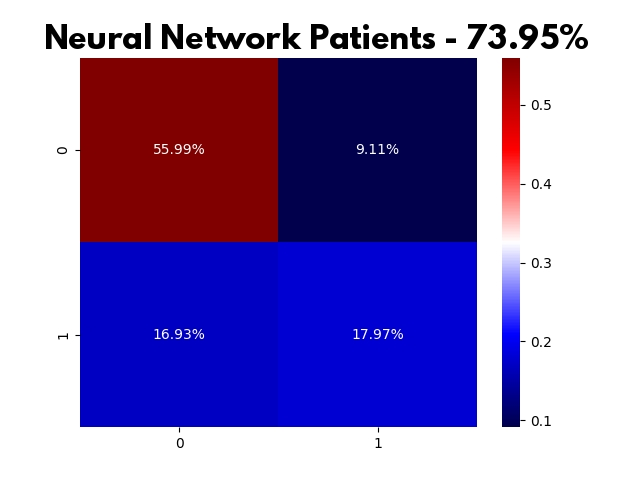
Overall, the results produced by neural networks varied across the two datasets. While a neural network was responsible for the most accurate e-mail classifier at 97.68% average, the technique conversely produced the least accurate test results when classifying the diabetes patients. The neural network 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.

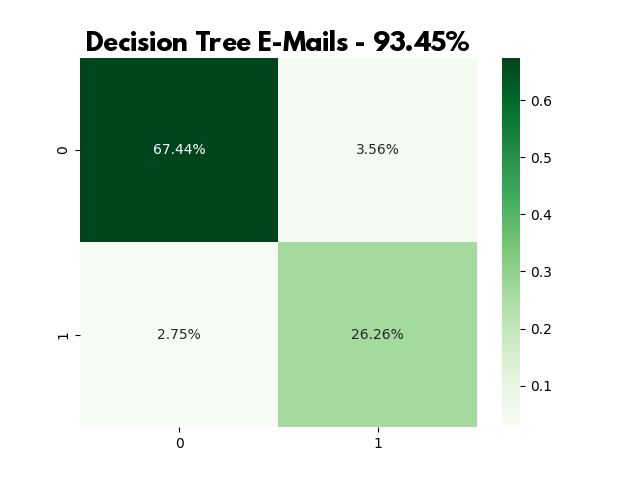
Due to this, iteration count was set to 75 in order to optimize training time to accuracy ratio, with an accuracy of 97.68% and a training time of 31.1 seconds. The average training accuracy of this neural network across all runs was 99.98%. This appears to be overfitting, but does not reflect in the 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 it results do not change across function calls. Contrary to the high accuracy of the e-mail classifier, the 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 to be close to log2(*features*). 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. Thus, I toggled the tree parameter ‘*class\_weight*’ to the ‘*balanced*’ setting in order to account for this. # GITHUB

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

