Assignment Two.

1 Description of Algorithm

1.1 Choice of algorithm

For this assignment the random forest algorithm was selected, an ensemble learning method based on decision trees. For a random forest, multiple decision trees are created by bootstrapping the training data. When making a prediction, every tree in the forest trees tries to predict the class based on the input data. Each of these predictions is counted and the majority class is taken for the final prediction to output. This algorithm typically has better performance than a single decision tree.

We achieved our goal of fully implementing the random forest algorithm using no modules except for NumPy. This was completed in Python 3.8 using Jupyter notebooks. After the algorithm was completed, we used matplotlib for the demonstration of results. We based our implementation off two videos created by Josh Starmer, which detailed the steps preformed as part of the algorithm. These videos detailed the steps for creating a decision tree [1], and how to turn this decision tree into a random forest [2], but our code was created entirely from scratch.

1.2 Decision Tree Branching Rules

Due to all the inputs being continuous values we needed to calculate the value at which we should partition the data on, i.e. the threshold. This was done in a loop as each column needed to have a threshold value calculated. To calculate the threshold value of a column the input data column and output data was sorted so it was in order based on the input data column. The mid-value between each row of the input column was calculated and the Gini value of a partition based on this midway value was calculated. This was repeated for each row until the lowest Gini value was calculated and therefore the threshold to partition the data on was found.

Once this threshold was found for each column of data in the input array the root of the tree was found. This was done by finding the lowest Gini value of each column of the input data set partitioned around the threshold value. The column with the lowest Gini value and therefore the highest information gain was selected as the root node.

1.3 Recursive Algorithm for Creating Trees

A recursive algorithm was developed for the creation of the trees. This algorithm takes the current node (if the first call, establishes a root node), creates both a left and right child for this node, then recursively calls the algorithm again for each child. At each node, the attribute and corresponding threshold to split the data is determined according to the Gini impurity, as outlined in section 1.2, and the data is separated and passed to each child to repeat the process (the left child receives the data less than the threshold, the right receives data above the threshold). A list is also parsed through each recursion that keeps track of the attributes used in the current branch, as each attribute can only be used to split the data once per branch. The only exit conditions for are when a leaf node (i.e. prediction) should be created. These conditions are:

(a) All of the input attributes have been used exactly once.
(b) After splitting at each node in the current branch, all of the remaining data belongs to one class.
(c) The Gini impurity of the parent node is less than the minimum Gini from any of the remaining attributes.

In all other cases, two child nodes are created, and the algorithm is called again for both. The general structure for our implementation of this algorithm is as follows:

1. From the previous recursive call, pass in the input samples and their associated output classes, the Gini value of the parent node, and a list of the attributes used in previous nodes on this branch.
2. If there is no parent node, create the root node of the tree.
3. If all of the input attributes have been used in a node on the current branch, make a leaf node and return.
4. If the remaining data samples (split by the previous node) contain only one output class, flag the current node as a leaf and return.

5. Calculate the minimum Gini (and the threshold value that yields this minimum Gini), for each attribute that has not been used previously in this branch. From this list of Gini's for each attribute, again take the minimum.
6. If this minimum Gini is greater than the Gini of the parent node, flag the current node as a leaf and return.
7. Otherwise, create a new branch node with a left and right child. For both of these child nodes:
    7.1. Split the data based on the threshold calculated in step 5.
    7.2. Add the attribute associated with this threshold to the list of attributes used in the current branch.
    7.3. Finally, call the recursive algorithm again.

1.4 Tree Data Structure

In order to save a tree for navigation later a Node class was implemented. This Node class stores the Index value of the row and a data value (either the threshold value or the predicted value). There are methods to add and get a child node, to get the data stored in the Node and to check if a Node is a leaf. If the Node is a leaf, then the index pointer will equals to None and the value will be the classification value.
Once it was possible to create one tree a method with the input N will create N trees and add these trees to an array. In order to create different trees the data should be bootstrapped before training each tree. This means a random subset of data

2 Tests & Results

2.1 Decision Tree Tests

After implementing the decision tree it needed to be tested before we could implement the random forest. In order to test the decision tree we compared it against the decision tree classifier from scikit learn and plotted the results. The performance of both implementation was calculated ten times splitting the data into different testing and training datasets each time. The average of these 10 tests were tested and compared. The results of the ten cases is graphed below:

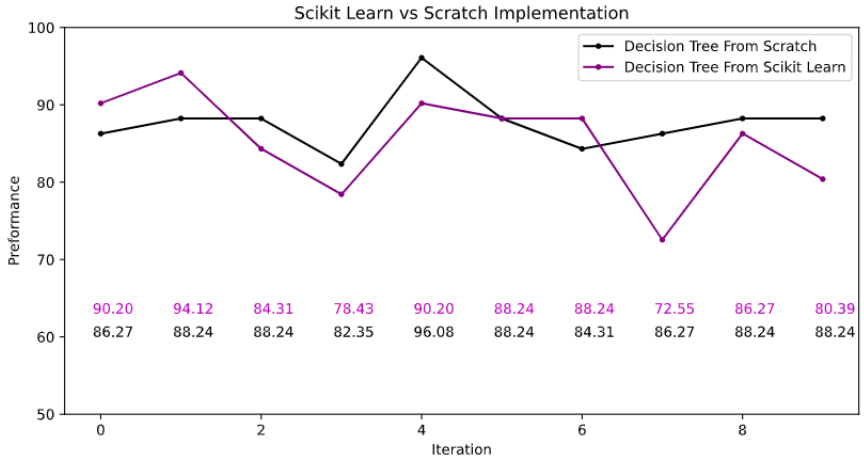

Figure 2.1.1: Average performance over ten iterations

As seen above in figure above both decision tree implementations have similar performance over the ten tests. These ten tests give a performance average of 85.29% for Scikit learns decision tree and 87.64% for the decision tree implemented from scratch. The implementations were also tested on the training data each time to ensure overfitting was not occurring. The implementations gave a performance of 100% and 96.50% for scikit learn and our implementation respectively.

2.2 Random Forest N Trees Tests

Next, we tested our random forest classifier accuracy for different numbers of trees in the forests. These were compared against scikit-learn random forests of the same size. We also set max_features=None for the scikit forests,

meaning at each node all remaining attributes are checked to be used for the next node, as this more closely matches our implementation (this applies for all subsequent tests where we compare to scikit-learn). Results are plotted in Figure 2.2 below.

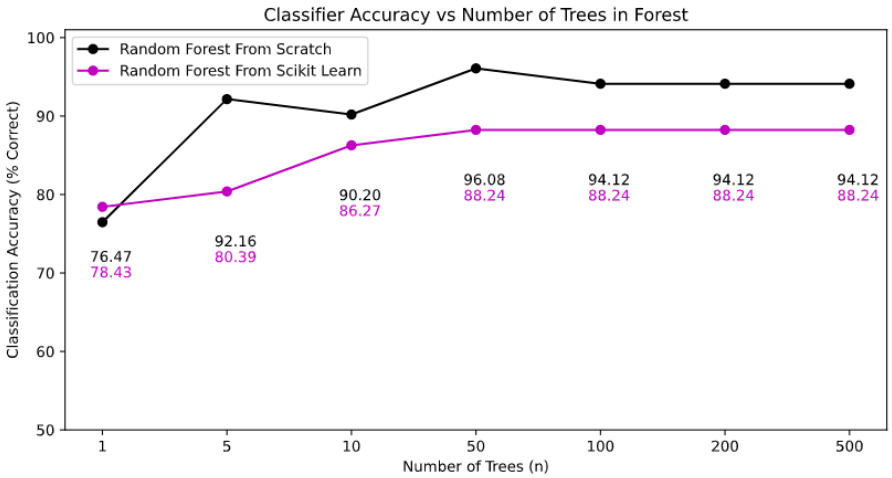

Figure 2.2.1

With this set of parameters, our algorithm tended to outperform the scikit implementation as the accuracies. Both implementations saw the accuracies plateau, ours from 100 trees, scikit's at 50 trees. Scikit's peak accuracy was 88.24% at this plateau point. However, our peak accuracy was 96.08 at 50 trees. The decrease in accuracy beyond this point indicates overfitting, so 50 trees was used for all forests in subsequent tests. (Not plotted, but testing shows that both algorithms maintained 100% accuracy on the training set beyond 50 trees, which supports the overfitting theory).

2.3 Normalization Tests

Moving forward with forests of 50 trees, tests were carried out to measure the effect of normalisation of the classifier accuracy. For each test, 3 random forests were created. One with no normalisation, one with the dataset range normalised (each attribute has min = 0 and max = 1) and one with the dataset Z normalised (each attribute has mean = 0 and standard deviation = 1). The accuracy for each was obtained, and the test repeated 10 times. The results are plotted in figure 2.2.1.

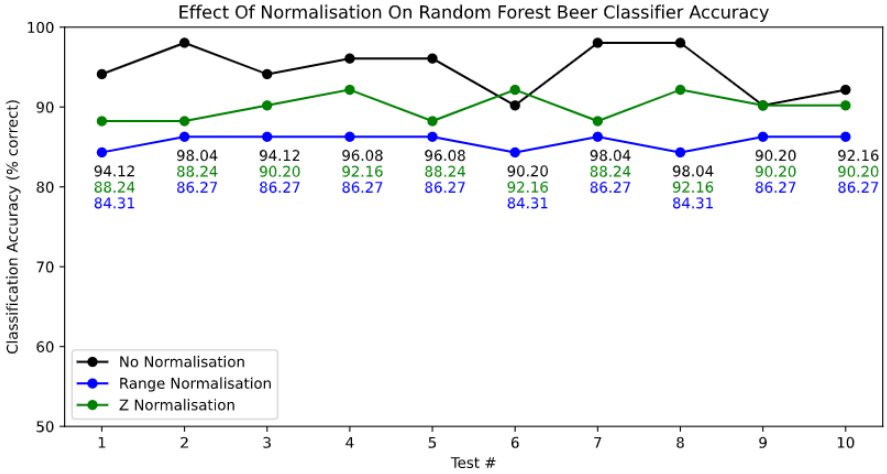

Figure 2.3.1: Classification accuracy tests with different normalisation types

The mean accuracy was calculated for each of these tests. The results are shown in table 2.3.1 below:

Normalisation type	Mean Random Forest Classification Accuracy (% correct)
None	94.706

Range	85.686
Z	90.000

Table 2.3.1: Mean classification accuracy using different normalisation types

Since normalisation did not increase the mean accuracy, further tests did not include this pre-processing.

2.4 Random Forest Results

Once the random forest algorithm was fully implemented and the results of normalization was found the final tests could be run on the algorithm. In order to test the random forest classifier our implementation was compared against Scikit learns algorithm. The random forest was trained and tested ten times with different random training and testing subsets of the whole data set. These performance values were plotted as seen below.

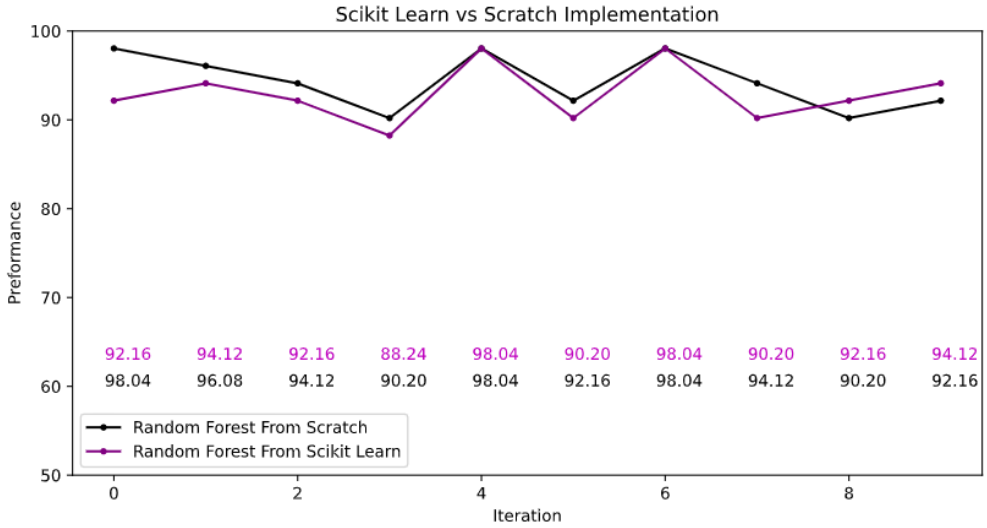

Figure 2.4.1: Average performance over ten iterations

As seen by the plot both implementations are returning similar performance values. Running these ten tests gave an average performance of 92.94% and 94.31% for Scikit Learn and our implementation respectively.

3 Conclusion and Observations

Implementing the machine learning algorithm was a difficult task. Once we implemented the decision tree it needed to be tested to ensure that the performance was similar to that of Scikit-Learn's decision tree implementation. The performance of both algorithms was extremely similar when run ten times with different configurations of the whole data set taken each time for testing and training data. The performance of both algorithms were 100% and 96.50% for Scikit Learn and the implementation from scratch respectively. This shows overfitting is not occurring meaning our implementation from scratch is successfully classifying data.

The results from our tests showed that random forests with >= 50 trees are quite capable in this classification task, with our implementation and the scikit-learn implementation achieving test accuracies of 94.31% and 92.94% respectively. It is important to note that both of these implementations had test accuracies of 100% for the same forest size. This variance may be caused by overfitting, which could potentially be reduced. One option for this would be to limit the number of attributes analysed for the best split when making the decision trees. The scikit implementation has a parameter "max_features" which achieves this. Given more time, we would also implement this into our algorithm and further test the performance to determine if the generalisation has improved.

4 Member Contributions

4.1 Maya McDevitt

During the assignment I actively participated in the process. During the initial stages of the project I took part in the research of various algorithms. My partner and I decided on doing a random forest classifier as the ensemble learning method introduced a level of complexity. Paul and I completed the methods to partition the data together as we could code together, due to us living together, this paired programming allowed us to agree on implementation methods and participate in all areas of the code. Finding the threshold values was also a function I initially had an input into but Paul made changes to it in order for it to work with his recursive implementation. I developed the method to branch the data to get the root node and the two data arrays that are a result of partitioning the data based off a threshold node. This method was then implemented in the recursion method developed by Paul. I implemented and tested the tree structure using the Node class. I completed the method used to train the forest and output an array of trees. This will be the method called to train the forest of trees. This method is called randomForestClassifierScratch(). I also implemented the method used to make predictions using the inputted data and a method to get the performance of the model. I developed and tested the predict function which would be used to classify the inputs to compare this implementation to the sklearn implementation one the trees were made. This method returns an array of predicted values. As part of the training I completed tests 2.1 and 2.3. I plotted the results for these sections and completed the report for the tests.

4.2 Paul Kielty

After Maya and I agreed an ensemble ML method would be a good direction to take for our assignment, I suggested one which involves bootstrapping of the data as this would allow us to make better use of our limited dataset. We agreed random forest would be sufficiently achievable and complex.

Regarding our python implementation, I created the code that reads in the beer.txt file, and organises it into an ndarray of floats and strings. I also created a method that allows the data to be range normalised or z normalised. Maya and I worked closely together (pair-programming) for the several functions. These include the methods to randomly split the data into training and testing sets, to separate these sets into input attributes and output classes, and the the methods for obtaining Gini impurity of given attributes.

I created the recursive algorithm for the construction of the decision trees, and the incorporated Maya's node data structure into it for later navigation and predictions. I also worked on the predict function such that it counts the classes at the leaf node of the tree and returns the majority class.

Maya and I both planned what tests we would carry out together then divided the work between us. I carried out and plotted tests for the classifier accuracy vs the number of trees in the forest, and accuracy of random forests using data with no normalisation, range normalisation, and z normalisation.

5 References

[1] J. Starmer, "Youtube statQuest: Decision Trees," 22 Jan 2018. [Online]. Available: https://www.youtube.com/watch?v=7VeUPuFGJHk&t=784s. [Accessed 21 Nov 2020].

[2] J. Starmer, "StatQuest: Random Forests Part 1 - Building, Using and Evaluating," 5 Feb 2018. [Online]. Available: https://www.youtube.com/watch?v=J4Wdy0Wc_xQ&t=196s. [Accessed 21 Nov 2020].

[3] Matplotlib Development Team, "Matplotlib.pyplot.plot Documentation," 8 April 2020. [Online]. Available: https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.plot.html. [Accessed 8 December 2020].

6 Appendix

6.1 Github repo link

https://github.com/mcDevittMaya5/RandomForestFromScratch

6.2 Code