Assignment 1 – Classification using Scikit-learn

Student Name: Ayodeji Ali **Student ID**: 20343733 **Programme**: 4BCT

Algorithm 1 – Decision Trees

Decision Trees are a non-parametric supervised learning method which is used to solve classification and regression problems. The idea is to create a model that makes predictions by follow a series of simple decision rules based on the dataset given.

Detailed Description of Decision Trees

A decision tree is a binary tree that recursively splits a dataset until it is left with pure leaf nodes. Pure leaf nodes are nodes with only one type of data classification in them. There are two types of nodes in decision trees: a decision node and a leaf node. Decision nodes contain the condition to split the data, while leaf nodes help decide the classification of the new data point. The model needs to learn which features to take and the corresponding value in which it can optimally split the data. It learns to optimally split the data by comparing all the root conditions that allow you to get pure leaf nodes. The model uses information theory—it chooses the split which maximizes information gain.

If the entropy is high, then we are uncertain of the randomly picked point, which means the model will use the split that provides the lowest amount of entropy. In this same sense, the model compares every possible split and uses the ones that maximize the information gain. The model traverses through every possible feature and feature value to determine the best possible feature and the best possible threshold. In each new level from the root on the decision tree, the impurity decreases.

The decision tree selects the current best split to maximize information gain; it does not backtrack or change after a decision has been made, which can stop it from guaranteeing optimization.

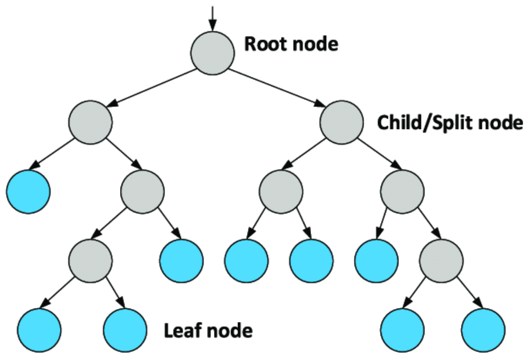


Figure 1: Graphic illustration of how decision trees works

Why I choose this algorithm.

The reason I chose decision trees is because to me they were easier to interpret as their structure resembles on of a flowchart, this makes it easy to visualise and understand how the model makes it's predictions. They can capture non-linear relationships in data allowing them to naturally model complex interactions between features that may not be linear. They are also known to perform well on classification tasks just like this one and regression tasks as they can handle numerical and categorical data directly. They are relatively fast to train and predict. The most appealing feature is they require minimal preprocessing of data. I think this is one the best model to choose as it’s quite accurate but not too complex allowing me to understand.

Hyperparameter Details for Tuning.

Hyperparamer 1: This would be *max_depth* which controls the depth on the tree.

Hyperparamer 2: This would be *min_samples_split* which is the minimum number of samples that are required to splits the node.

Algorithm 2 – Random Forest

Random Forest is a model that uses multiple decision trees using random subsets of the dataset. It combines the predictions of all the random subsets of the decision trees to improve accuracy and reduce the chance of overfitting. All the trees are trained using the best possible splits in the data (same for regular decision tree algorithm).

Detailed Description of Algorithm 2.

The random forest model uses a collection of decision trees, which makes it less sensitive to the training data and lowers the risk of overfitting. New datasets are built by randomly selecting rows from the original dataset, maintaining the same number of rows. This process is called bootstrapping, involving random sampling with replacement, so some rows might be picked more than once. Multiple random datasets are created, and a decision tree is trained on each subset.

When making predictions, the model runs the new data point through each tree and records the predictions. It then combines these predictions by majority voting—this part is called aggregation.

The random forest method combines bootstrapping and aggregation. It is termed "random" forest because it has two random features: bootstrapping, ensuring different data for every tree, and random feature selection, which reduces the correlation between the trees. These elements make the model less sensitive to the original training data and more robust overall, enhancing its accuracy.

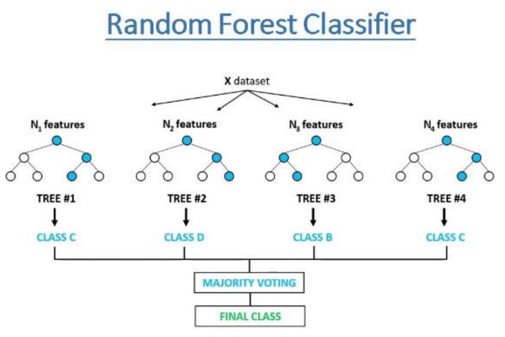


Figure 2: Graphic illustration of how Random Forest works

Why I choose this algorithm.

The reason I chose this algorithm is for multiple reasons. It involves the original algorithm that I chose (decision trees) so it gives a comparison to the original while also show how it could be possibly improved. It's a less sensitive model so it doesn't rely too much on the original dataset which makes it less risky I cases of overfitting. It's also easy to visualize as it's a collection of multiples decision trees. It's a powerful classification task and works well with many data types.

Hyperparameter Details for Tuning.

<u>Hyperparamer 1</u>: This would be *n_estimators* which is the number of trees.

<u>Hyperparamer 2</u>: This would be *max_depth* which is the depth on the tree.

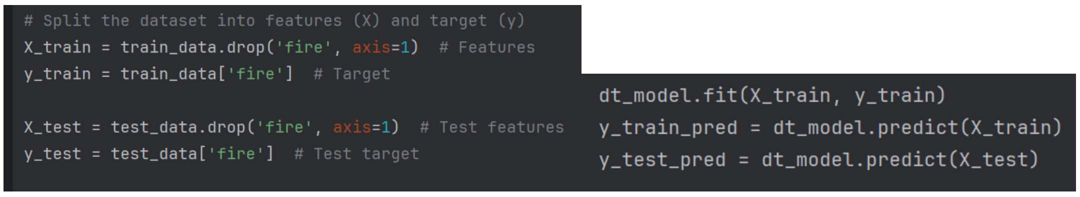
Algorithm 1 -Decision Trees- Model Training and Evaluation

Data Preprocessing and Visualisation

Data Preprocessing

I split the dataset into features (X_Train, X_Test) and target variable (Y_Train, Y_Test). This is so the model can learn patterns from the features and predicts the target, which would be if there's a fire or not.

The X_train and Y_Train Data is used to train for the DecisionTreeClassifier. The decision tree model split the data by using the feature values until the model fits the training dataset perfectly.



```
# Split the dataset into features (X) and target (y)
X_train = train_data.drop('fire', axis=1)  # Features
y_train = train_data['fire']  # Target

X_test = test_data.drop('fire', axis=1)  # Test features
y_test = test_data['fire']  # Test target
```

```
dt_model.fit(X_train, y_train)
y_train_pred = dt_model.predict(X_train)
y_test_pred = dt_model.predict(X_test)
```

Figure 3: Code example of how train data is pre-processed

Predictions by the model are made using the training data and the test data to make an evaluation on the model's performance.

Visualization

I used code to visualize the training if the decision tree structure after it's training. I used the plot_tree function from the sklearn library which graphically displayed the decision tree. It showed the decision nodes and leaf nodes, based on the feature splits.

The visualization helps understand how the model makes decisions makes decisions based on the features. The image shows the class the tree assigns to each decision and how the tree splits data based on the different features.

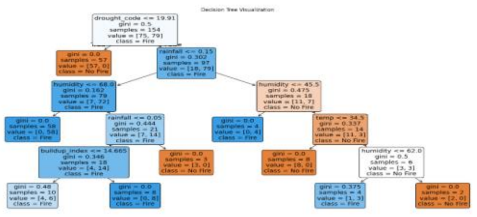

Decision Tree Visualization
drought_code <= 19.91 gini = 0.5 samples = 154 value = [75, 79] class = Fire
gini = 0.0 samples = 57 value = [57, 0] class = No Fire
rainfall <= 0.15 gini = 0.302 samples = 97 value = [18, 79] class = Fire
humidity <= 68.0 gini = 0.162 samples = 79 value = [7, 72] class = Fire
humidity <= 45.5 gini = 0.475 samples = 18 value = [11, 7] class = No Fire
gini = 0.0 samples = 58 value = [0, 58] class = Fire
rainfall <= 0.05 gini = 0.444 samples = 21 value = [7, 14] class = Fire
gini = 0.0 samples = 4 value = [0, 4] class = Fire
temp <= 34.5 gini = 0.337 samples = 14 value = [11, 3] class = No Fire
buildup_index <= 14.665 gini = 0.346 samples = 18 value = [4, 14] class = Fire
gini = 0.0 samples = 3 value = [3, 0] class = No Fire
gini = 0.0 samples = 8 value = [8, 0] class = No Fire
humidity <= 62.0 gini = 0.5 samples = 6 value = [3, 3] class = No Fire
gini = 0.48 samples = 10 value = [4, 6] class = Fire
gini = 0.0 samples = 8 value = [0, 8] class = Fire
gini = 0.375 samples = 4 value = [1, 3] class = Fire
gini = 0.0 samples = 2 value = [2, 0] class = No Fire

Figure 4: Illustration on how the decision tree worked with the data given

I plotted the impacts of the hyperperameters, max_depth and min_samples_split on the decision tree's performance by using sns.lineplot(). It shows how the test accuracy of the tree changes as the model is tuned.

The visualization helps to identify the best combination of the hyperparameters that give the highest accuracy on the test data.

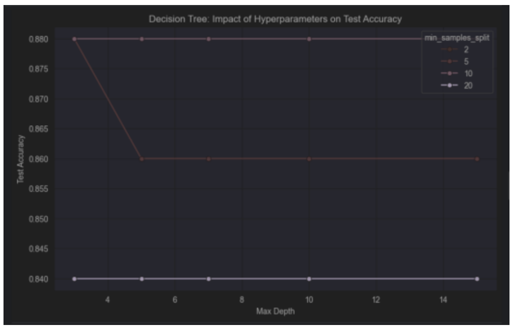


Figure 5: Visualisation of the correlation of test data when hyperparameters are tuned.

Training and Evaluation Details

Training:

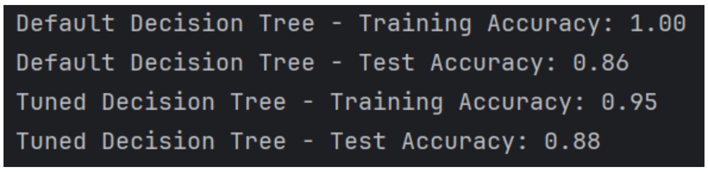
The default decision tree doesn't have any restriction on depth and node splits, so it grows fully. It mostly over-fits as a result. The model carves out the noise in the training data so it has very high accuracy.

The tuned decision tree has bounded max_depth to 5, and the minimum required samples to split a node is 10. This allows the model to prevent overfitting by focusing on general patterns for better generalization on the test data.

Evaluation:

The default decision tree has high training accuracy but lower test accuracy and so, it overfits. On the other hand, this tuned model has lower training accuracy but performs much better on the test set, which suggests that it generalizes better.

The tuned model has higher test accuracy, generalizing better compared to the default model.



```
Default Decision Tree - Training Accuracy: 1.00
Default Decision Tree - Test Accuracy: 0.86
Tuned Decision Tree - Training Accuracy: 0.95
Tuned Decision Tree - Test Accuracy: 0.88
```

Figure 6: Summary of Results Achieved from Training and Testing

Discussion of results

The default Decision Tree model achieved a perfect training accuracy of 1.00 but had a lower test accuracy of 0.86, showing clear signs of overfitting. It fit the training data too closely but struggled to generalize to unseen data.

After tuning with max_depth=5 and min_samples_split=10, the model's test accuracy improved to 0.88, while the training accuracy dropped slightly to 0.95. This balance shows that limiting the tree's depth and increasing the minimum samples per split helped prevent overfitting and improved generalization.

Conclusions

Tuning the Decision Tree reduced overfitting and led to better test accuracy. While the default model overfit the data, the tuned model generalized better, showing the value of proper hyperparameter selection.

Algorithm 2 – Random Forest - Model Training and Evaluation

Data Preprocessing and Visualisation

Data Preprocessing

I used the same steps covered in decision tree segment.

The RandomForestClassifier was trained with the training data. Random Forest creates multiple decision trees, where each tree is train using a random subset of features and samples. The helps to improve generalization and to avoid overfitting.

Like Decision Trees, the predictions are made on the training and test data sets, but the multiple trees allow for better generalizations.

Visualisation

For the Random Forest the hyperparameters that are tuned are the number of trees n_estimators and the depth of each individual tree max_depth. I plotted the impact of the parameters for the test accuracy are plotted using sns.lineplot().

The plot helps to visualize how increasing the number of trees or adjusting the max depth of the trees affects the model's performance. More trees improve performance but too many trees or trees that are too deep leads to overfitting.

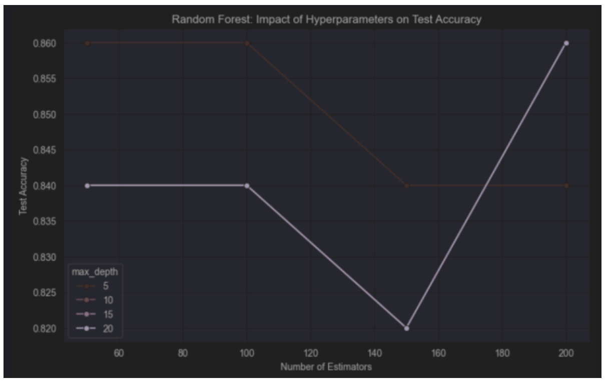


Figure 7: Visualisation of the correlation of test data when hyperparameters are tuned.

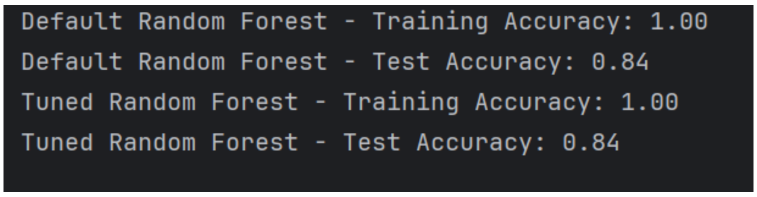
Training and Evaluation Details

Training

The forest that uses all the defaults of RandomForestClassifier is already an ensemble model that aggregates many decision trees, this prevents overfitting, returning a powerful performance without much tweaking. In contrast, the tuned random forest extends this baseline by increasing the number of trees to 100 (n_estimators = 100) and setting the maximum depth of each tree to 10 (max_depth = 10), modifying only two parameters from their defaults. This helps control the complexity of the individual trees, leading to improvement on generalization, especially on unseen data.

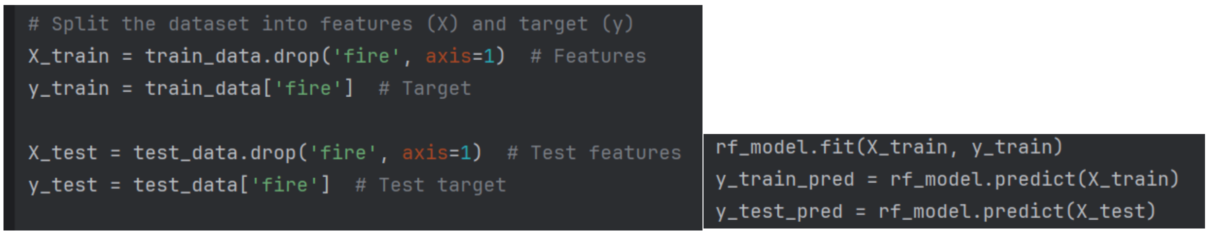
Evaluation:

The random forest by default does better with its high test accuracy, benefiting from the power of ensemble learning. Basically, it does better than a single decision tree. This tree provides an excellent balance between high training accuracy and good test accuracy, meaning there is a low risk of overfitting. The tuned model should enhance this by further improving test accuracy, in that its limiting over tree depth and increasing the trees provided ensure better generalization.



```
Default Random Forest - Training Accuracy: 1.00
Default Random Forest - Test Accuracy: 0.84
Tuned Random Forest - Training Accuracy: 1.00
Tuned Random Forest - Test Accuracy: 0.84
```

Figure 8: Summary of Results Achieved from Training and Testing



```
# Split the dataset into features (X) and target (y)
X_train = train_data.drop('fire', axis=1)  # Features
y_train = train_data['fire']  # Target

X_test = test_data.drop('fire', axis=1)  # Test features
y_test = test_data['fire']  # Test target
```

```
rf_model.fit(X_train, y_train)
y_train_pred = rf_model.predict(X_train)
y_test_pred = rf_model.predict(X_test)
```

Figure 9 :Code example of how train data is pre-processed

Discussion of results

The default Random Forest model achieved a perfect training accuracy of 1.00 but had a lower test accuracy of 0.84, indicating overfitting. The model performed well on the training data but failed to generalize as effectively on the test set.

After tuning the model with parameters such as n_estimators=100 and max_depth=10, there was no significant improvement in test accuracy, which remained at 0.84. Testing with different configurations of n_estimators and max_depth did not yield any major improvements either. The model maintained strong training accuracy but continued to face challenges with generalization.

Conclusions

While Random Forest showed strong training performance, tuning the model did not lead to noticeable gains in test accuracy. The model was resistant to overfitting but did not significantly improve generalization compared to the default version. This highlights the importance of further exploration into feature selection or tuning techniques for better test performance.

Comparative Analysis of Algorithm Performances

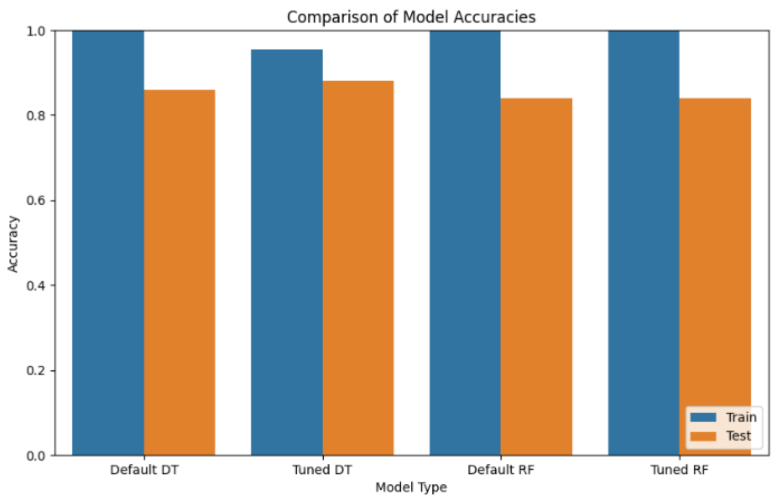


Figure 10: Graphed comparison of results from the two algorithms

Recommended Hyperparameter Valued based on Results

Decision Tree:

1. **max_depth=5**: This value effectively limits the tree's growth, reducing overfitting while enhancing generalization to unseen data.
2. **min_samples_split=10**: Setting a minimum number of samples required to split a node helps ensure that the tree is not too complex.

Random Forest:

1. **n_estimators=100**: This number of trees provides a good balance between model performance and computational efficiency.
2. **max_depth=5**: Similar to the Decision Tree, limiting depth helps mitigate overfitting and improves generalization.

Concluding Remarks

The Decision Tree model showed a high training accuracy but struggled with overfitting, achieving a test accuracy of 0.88 when tuned. The Random Forest model, while consistently performing well, demonstrated less improvement in test accuracy, remaining at 0.84 even after tuning.

The recommended hyperparameters for both models—max_depth=5 and appropriate values for min_samples_split in the Decision Tree, and n_estimators=100 for the Random Forest—offer a solid foundation for future iterations. This analysis highlights the importance of careful hyperparameter tuning and model evaluation in achieving optimal performance in machine learning tasks.

References

1. Scikit-learn developers, n.d. *Decision Trees — scikit-learn documentation*. [online] Available at: https://scikit-learn.org/stable/modules/tree.html [Accessed 1 October 2024].
2. Normalized Nerd, 2020. *Random Forest Algorithm Clearly Explained!* [video online] Available at: https://www.youtube.com/watch?v=v6VJ2RO66Ag [Accessed 4 October 2024].
3. Normalized Nerd, 2020. *Decision Tree Classification Clearly Explained!* [video online] Available at: https://www.youtube.com/watch?v=ZVR2Way4nwQ&t=438s [Accessed 3 October 2024].