# Assignment 1 – Classification using Scikit-learn

**Student Name**: <Hameed Adagun> **Student ID**: <20329901> **Programme**: <4BCT>

## Algorithm 1 - <Decision Trees>

<Introduction> Decision trees are like a flowchart for making decisions. In the context of machine learning, they are a way to make predictions by splitting data into smaller and smaller groups based on features. You ask true-or-false questions to narrow down your guess and predict outcomes. They help make complex choices into simple steps and can be used for both classification and regression problems.

### Detailed Description of Algorithm 1.

At the top of the decision tree, you have what's called the "**root node.**" This represents the initial question or condition that you use to split your data. It's often the most important feature or attribute. To determine the most important feature you use various splitting criterions. Splitting criterions are hyperparameters and I will discuss them further on in this document.

From the root node, you have branches that lead to other nodes. Each branch represents a possible outcome or answer to the question posed by the root node. These branches connect to child nodes.

**Child nodes** are nodes that follow the root node. They represent further questions or conditions based on the answer to the root node's question. These nodes can be thought of as subsequent decisions. You use splitting criterions to determine the conditions similarly to the root node.

The end points of the branches are called "**leaf nodes.**" These nodes represent the final outcomes or classifications. In a classification problem, each leaf node corresponds to a specific class or category. In a regression problem, leaf nodes might contain predicted numerical values.

The process of splitting nodes into child nodes is repeated **recursively**. At each node, the algorithm selects the best feature and threshold to split the data based on the chosen criteria. This process continues until a stopping condition is met. This condition could be a predefined tree depth, a minimum number of samples in a node, or other criteria.

After the decision tree is constructed, it may be pruned to remove branches that don't contribute much to the accuracy of the model. This is done by comparing the error rate with and without a given subtree and only keeping the tree if it improves the error rate. This is called **post-pruning**. **Pre-pruning** is done before the decision tree is constructed. This is done by limiting the depth of the tree or only creating a new node the splitting criterion is above a certain threshold. Pruning is completely optional and helps prevent overfitting, where the model fits the training data too closely and performs poorly on new, unseen data.

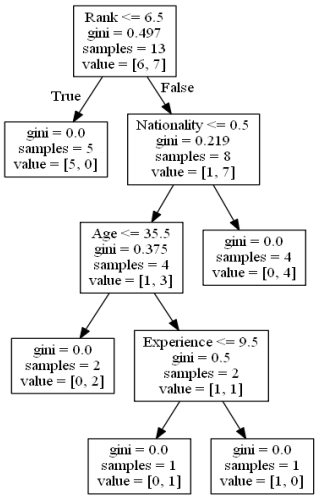



*Figure 1: Graphic illustration [2] of how the algorithm works*

In the illustration above the rank, nationality, age and experience are the features. In this example the **gini index** is used. ***See Appendix [2]***. The value next to gini is the quality of the split. 0.0 means no there is no split in the sample and 0.5 means the split is exactly in the middle. The value refers to which of the samples are represent by which category in the **target variable**. In the root node the value is [6,7]. In this instance this means that 6 of the samples "No" and 7 will get a "Yes".

Based on the decision tree Rank is the most important feature. This is the feature that had the highest gini gain when splitting the data. Meaning that this feature is least likely to be misclassified.

### Why I choose this algorithm.

Decision trees are highly **interpretable**. This means that you can easily visualize and understand how the model makes decisions. Each branch of the tree corresponds to a feature and its threshold, making it intuitive to follow the decision-making process.

As well as providing a measure of **feature importance**. You can analyse which features contribute the most to the model's decisions. This can be valuable in understanding the dataset given.

### Hyperparameter Details for Tuning.

*Criterion:* You can choose how the tree measures the quality of a split for each of the different features. "**Gini**" and "**Entropy**" are two options. ***See Appendix [1]***. Different criteria give you different results.

*Max depth:* Setting a limit on how deep your tree can grow. It's important for prevent **overfitting**, where your tree gets too specific to your training data and doesn't generalize well to new data (the test data in this instance)

## Algorithm 2 - <Support Vector Machines (SVM)>

<Introduction> SVMs finds the best way to draw a line (or a hyperplane in higher dimensions) between data points and groups them into classes. This is done using the "**margin**". The **margin** in the context of a SVM is the distance between the decision boundary (hyperplane) and the nearest data points from each class, which are known as support vectors. It represents the level of confidence or separation between the classes.

### Detailed Description of Algorithm 2.

Since we are using the "**C**" hyperparameter I am using the soft margin SVM which allows some degree of misclassification making it less prone to overfitting. The margin is controlled by the hyperparameter "**C**". The goal is to is to find a hyperplane that separates the data into two classes while maximizing the margin between them, just like in any SVM.

The type of SVM I will be describing in detail is the **linear SVM**. SVMs with different kernels work differently. With a linear SVM you should be using only two features. A straight line should separate the data points into its respective classes. This straight line should maximise the distances from the support vectors. Below in the illustration you can distances from the support vector and the line. This is known as the margin.

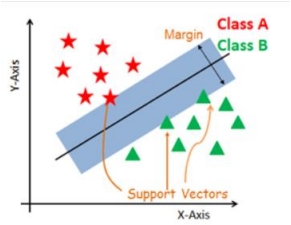



*Figure 2: Graphic illustration [6] of how the algorithm works*

### Why I choose this algorithm.

SVMs are a well-established and widely used machine learning algorithm. You can find a wealth of resources, libraries, and tools for SVM implementation and optimization.

### Hyperparameter Details for Tuning.

*C (Regularization Strength):* A smaller C value adds more regularization, making the model simpler and potentially less prone to overfitting but allows **more misclassifications**. A larger C relaxes the regularization, allowing the model to fit the training data more closely, which could lead to **overfitting** if not careful.

*Kernel:* SVMs can transform your data into a higher-dimensional space to make it easier to separate. Common kernel choices include "**linear**" (for straight lines) and "**poly**" (polynomial). The choice depends on your data's shape.

## Algorithm 1 - < Decision Trees > - Model Training and Evaluation

<Introduction> Before I could even start coding a decision tree, I needed to split the training and testing datasets into its **independent and target variables**. For both datasets the target variable was 'fire' and was named y_train and y_test respectively. The rest of the variables were the independent variables. These were inputted into its own separate dataframe and are used to determine whether there was going to be fire or not, the target variable outputting 'yes' for a fire and 'no' for no fire. These were named X_train and X_test respectively. The rows for the training dataset were 154 and the rows for the testing dataset were 50 meaning that split for the total dataset was approximately 68% for the training dataset and 32% for the testing dataset. The default split for sklearn is 75% training and 25% testing which shows that the split is close to a common starting point. ***See Appendix [3].***

## Visualisation

Below is a visualisation of the dataset using both the **gini and entropy criterion**. I left as depth as the default which is None. This means that all the nodes are expanded until all the leaves are pure or until it reaches the minimum samples required for a leaf node. The default or the minimum sample is 2. You can see for both the gini and entropy decision tree that the **root node is drought_code**. This means that that gini gain and information gain for that feature was the highest out of all the other features making it the features least likely to get misclassified. The accuracy of for both models for the **training dataset** was 1.0. For the **entropy model the accuracy** for the **testing dataset** was **0.82** and for the **gini model the accuracy was 0.86**.

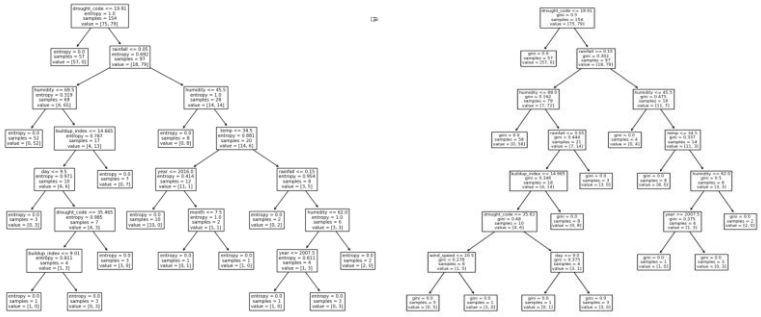


*Figure 3: Visualisation of the dataset before hyperparameter tuning*

## Training and Evaluation Details

The two hyperparameters that I used for the decision tree was **max depth** and **criterion**. To tune these two hyperparameters and discover which combination created the best result for the decision tree I iterated through both using nested loop. The first loop iterated through the criterion and the second loop iterated through the depth. I decided the depth was **1-10**. Below you can see the results represented visually.

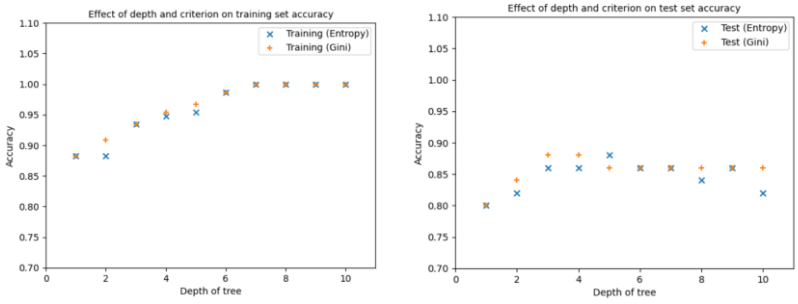
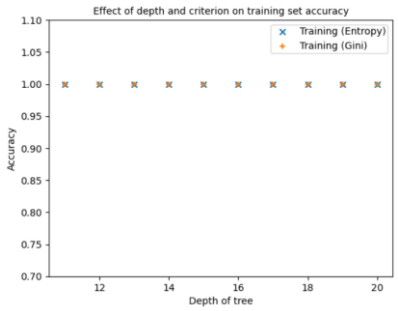



*Figure 4: Summary of Results Achieved from Training and Testing*

## Discussion of results

For the **training dataset** for both the **gini and entropy criterion** the best depth is in the range of **7-10**. Since the accuracy for all these values are the same regardless of the criterion, I thought maybe that from 7 onwards the accuracy will always be the same as this is the maximum depth for the decision tree where the accuracy doesn't increase or decrease anymore. To test this, I created another graph for the training data where the depth of the starts at 11 and ends at 20. The results below shows that my intuition was right.




For the **testing dataset** the best depth for the **gini criterion** was **3 and 4**. And for the **entropy criterion** the best depth was **5**. The results for both were the same so there is no absolute best. You can see at depth 8 and 10 for the entropy criterion the accuracy starts to decrease. It is possible that as the tree begins to get deeper it get too specific to the training data causing the tree not to generalise well to new data, in this case the testing data.

## Algorithm 2 - < Support Vector Machines (SVM) > - Model Training and Evaluation

<Introduction> Similar to decision tree.

### Visualisation

To be able more easily visualize the dataset I created a subset of the training dataset which included only two features. The two features I chose were **drought_code** and **wind_speed.** You can see the line that separates the 'yes' category from the 'no' category. Some of the datapoints are misclassified which shows the soft margin. I used the default hyperparameters for the kernel and C, which was rbf and 1. The **result** for this model was **0.84 for the training dataset** and **0.74 for the testing dataset**.

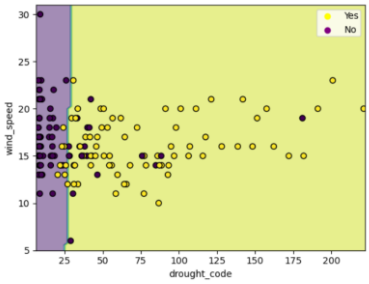



*Figure 5: Visualisation of the dataset*

### Training and Evaluation Details

The two hyperparameters I used for the SVM was **C and the kernel**. Similarly, to the decision tree I was nested loop for hyperparameters to discovers which combination yielded the best results. The three kernels used were linear, poly and rbf. The values that were used for C were in the range of 1-10.

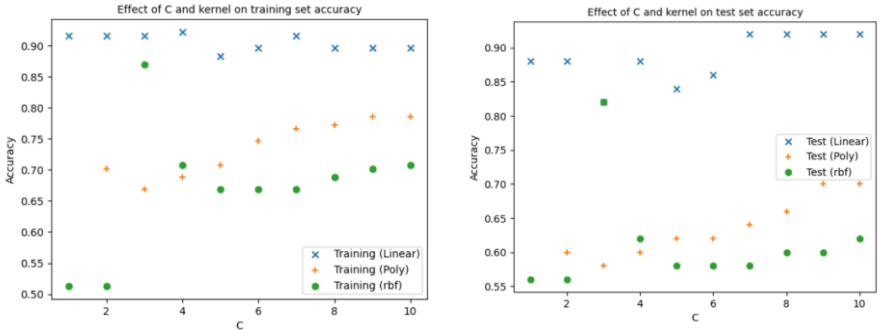



*Figure 6: Summary of Results Achieved from Training and Testing*

### Discussion of results

For the **training dataset** the best result was **linear kernel** with the value of **C being 4**. For the **testing dataset** the best result was also the **linear kernel** with the value of **C being in the range of 7-10**. It is interesting to note that the results for the rbf kernel SVM spikes for both the training and testing dataset when the value of C is 3. This shows that this specific model for the rbf kernel is generalizing well to the dataset.

## Conclusions

Based on the results found on the **decision tree** I believe the best combination of hyperparameters is the **criterion being gini** and the **depth being 7**. The reason I believe that this is the best combination is because it gives us 1.00 for the training dataset. While the result for the testing dataset may not be the highest choosing the higher results is not worth the drop in the training dataset. The reason chose the depth being 7 instead of 8-10 is because I wanted to pick lowest possible depth. This is because lower depth decision trees are more interpretable and run faster with new data.

For the **SVM** I believe the best combination of hyperparameters would be the **kernel being linear** and the **C being 7**. The results for the training dataset are the highest here and the results for the testing dataset is only slightly lower than the highest.

### Comparative Analysis of Algorithm Performances

The metric of accuracy used for both algorithms were the same. It allows us to directly compare the accuracy to determine which algorithm is the best for the dataset.

```
Accuracy on training data with depth 7 and criterion gini: 1.0
Accuracy on testing data with depth 7 and criterion gini: 0.86

Accuracy on training data with C 7 and kernel linear: 0.9155844155844156
Accuracy on testing data with C 7 and kernel linear: 0.92
```

Looking at the results for the best hyperparameters for both algorithms I believe that the **decision tree** algorithm is the **best for the dataset**. It gives us an accuracy of 1 for the training dataset compared to an accuracy of 0.92 for the SVM. The accuracy for the testing data may not be as high but I believe

the accuracy for the training dataset is more important as the testing dataset. It is not guaranteed that new data introduced to the model is going be similar the testing dataset.