**Decision Tree**

DT mimic the human decision-making process and are efficient in dealing with categorical data.

DTs do not help in finding a linear relationship between the independent variable and the target variable. However, they can be **used to model highly non-linear data**.

**Random forests** are collections of multiple trees.

**Why do we need DT and RF, because these models are also used for classification and regression.**

Linear models like regression cannot handle collinearity and non linear relationships in the data well. DTs leverages these properties.

There are certain cases where you cannot directly apply linear regression to solve a regression problem. Linear regression fits only one model to the entire data set; however, you may want to divide the data set into multiple subsets and apply decision tree algorithm in such cases to handle non-linearity.

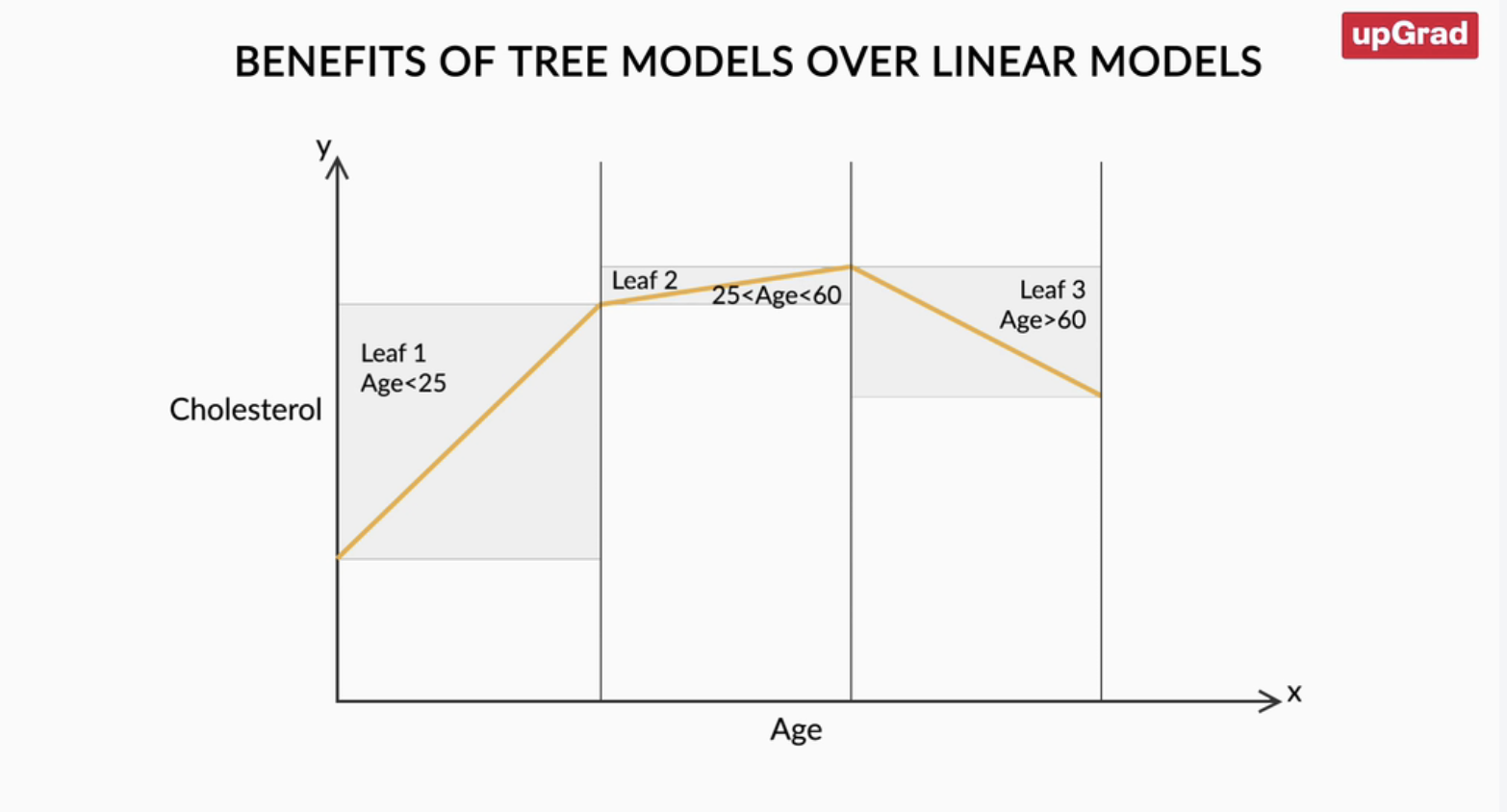
Advantages of DT over LR and other linear models

* Interpretable – very easy to interpret tree models. Simple in English language. But LR and other Linear models are difficult to interpret because there we have coefficients, p-values etc.
* Versatile – DT can handle both classification and regression. seamlessly handle all kinds of data such as numeric, categorical, strings, Boolean, etc
* Fast
* handles multicollinearity better: it is scale invariant means does not require normalization of values at different scales. Because in DT it only has to compare the values within an attribute – it compares values within one column.

In DTs, you do not have to treat missing values, outliers and multicollinearity before proceeding with model building.

* They can identify complex relationships and work well in certain cases where you cannot fit a single linear relationship between the target and feature variables. This is where regression with decision trees comes into the picture.

So basically, if we have a data set and in X dataset we observed different groups within datset. So relationship between X and Y variable is not linear and different groups are performing differently against target Y var. and in that case it would be appropriate to use DT along with regression. Below age is not liner with cholesterol. Age has different groups.



The difference between DT classification and DT regression is that in regression,

* each leaf represents the average of all the values as the prediction in regression
* as opposed to a class label in classification trees.
  + For classification problems, the prediction is assigned to a leaf node using majority voting but for regression, it is done by taking the average value.

DT –

* DT uses a tree-like model to make predictions.
* It resembles an upside-down tree.
* splits data into multiple sets of data.
* Each of these sets is then further split into subsets to arrive at a decision.
* The first and top node of a decision tree is called the **root node**. The arrows in a decision tree always point away from this node.
* The node that cannot be further classified or split is called the **leaf node**. The arrows in a decision tree always point towards this node.
* Any node that contains descendant nodes and is not a leaf node is called the **internal node**.

A diagram of a root node

Description automatically generated

* DT uses a natural decision-making process, it asks a series of questions in a nested if-then-else structure. On each node, you ask a question to further split the data that is held by the node. If the test passes, you move to the left; otherwise, you move to the right.

A diagram of decision trees

Description automatically generated

The first question we asked was, "Is the age less than 54.5?". Depending on the answer, we moved to the next step where we asked whether the person is a male or a female, and so on. At the end of this line of question, lies an answer - whether the person has heart disease or not.

So from the above tree we can conclude that

* Young females with cholesterol < 300 have lowest chance of heart disease.
* Old males with cholesterol > 300 have highest chance of heart disease

**In DTs, you can traverse the attributes backwards and identify the factors that lead to a particular decision.**

DT building process is a **top-down** approach. The top-down approach refers to the process of starting from the top with the whole data and gradually splitting the data into smaller subsets.

We call DT process **greedy** because - it does not take into account what will happen in the next two or three steps.

The entire structure of the tree changes with small variations in the input data.

Small changes in data, changes the way you split and the final decisions altogether.

Process **aims to** **gain an immediate result** that is derived after splitting the data at a particular node based on a certain rule of the attribute.

constructing a decision tree involves the following steps:

1. Recursive binary splitting/partitioning the data into smaller subsets
2. Selecting the best rule from a variable/ attribute for the split
3. Applying the split based on the rules obtained from the attributes
4. Repeating the process for the subsets obtained
5. Continuing the process until the stopping criterion is reached
6. Assigning the majority class/average value as the prediction

**Two important things are-**

1. How to identify the variable or attribute for the split at the node
2. What is the stopping criterion

In python code built a DT model using default hyperparameters of decision tree except depth. Accuracy of test data was very low. So will have to perform hyperparameter tuning.

**What are hyperparameters?**

* H. are simply the parameters that we pass on to the learning algorithm to control the training of the model.
* H. are choices that the algorithm designer makes to ‘tune’ the behaviour of the learning algorithm.
* So basically anything that is passed on to the algorithm before it begins its training or learning process is a H, i.e., these are the parameters that the user provides and not something that the algorithm learns on its own during the training process.
* Here, one of the H. was "max\_depth" which essentially determines how many levels of nodes will you have from root to leaf.
* Now, obviously, since hyperparameters can take many values, it is essential for us to determine the optimal values where the model will perform the best. This process of optimizing hyperparameters is called hyperparameter tuning.

In DT how should we split the tree –

What is split in DT trying to achieve – homogeneity or pure subsets. For example if we set a threshold like suppose if out of total 80% of people have cholesterol > 300 then that set is homogeneous and we define a rule that people with cholesterol >300 have heart disease. So if we split the dataset using condition cholesterol >300 and we observed that more than 80% people are with heart disease that means we breached threshold of 80% and we do not have to split further.

So splitting continuous until the homogeneity value exceeds the threshold value.

A diagram of a diagram of a number of dots

Description automatically generated

We measure homogeneity using different methods –

* Gini index,
* entropy,
* classification error (for classification)
* MSE (for regression)

If an attribute is ordinal categorical or continuous in nature with n different values -> n - 1 different possible splits for it

If an attribute is nominal categorical, ->  2k-1−1 possible splits for this attribute, where k is the number of classes.

Classification error

Let’s say we have two classes 0 and 1. 20 people belongs to 0 and 80 people belongs to 1. **CE = 1 – max(pi)** basically probability of 1 is 0.8 and 0 is 0.2. so error will be 1-0.8 = 0.2. it is like most of the people belongs to group 1. So if any observation gets classifies as 0 that means classification error will be 0.2. If I assign everything to the majority class what is the error rate I have.

A diagram of a number of equations

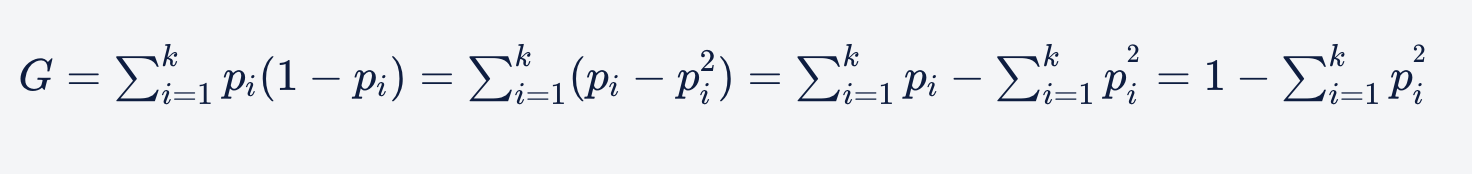
Description automatically generated with medium confidence

Gini Impurity or Gini Index –

G=∑ki=1pi(1−pi)

A math equations and numbers

Description automatically generated with medium confidenceSuppose you have a data set with two class labels. If the data set is completely homogeneous, i.e., all the data points belong to label 1, then the probability of finding a data point corresponding to label 2 will be 0 and that of label 1 will be 1. So, p1 = 1 and p2 = 0. The Gini index 1(1-1), which is equal to 0, will be the lowest in such a case. Hence, **the higher the homogeneity, the lower the Gini index.**



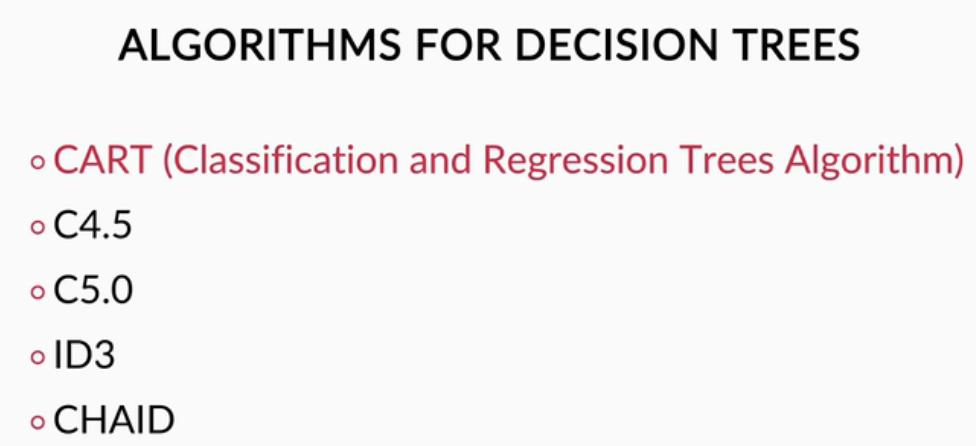
Entropy -

A math equations and formulas

Description automatically generated with medium confidence

* D=−∑ki=1pi.log2(pi),

Entropy quantifies the degree of disorder in the given data, its value varies from 0 to 1. Entropy and the Gini index are similar numerically. If a data set is **completely homogenous**, then the entropy of such a data set **will be 0,** i.e., there is no disorder in the data. If a data set contains an **equal distribution of both the classes**, then the **entropy of that data set will be 1**, i.e., there is complete disorder in the data. Hence, like the Gini index,**the higher the homogeneity, the lower the entropy.**



To summarise, **the information gain** is calculated by:

* Gain=D−DA

where D is the entropy of the parent set (data before splitting),DA is the entropy of the partitions obtained after splitting on attribute A. Note that **reduction in entropy** implies **information gain**.

Example –

A math equations on a white background

Description automatically generated A diagram of a diagram

Description automatically generated

**Gini Impurity using Example –**

A screenshot of a paper

Description automatically generated

A screenshot of a table

Description automatically generated

For Male -

A screenshot of a math problem

Description automatically generated

For Female –

A screenshot of a math problem

Description automatically generated

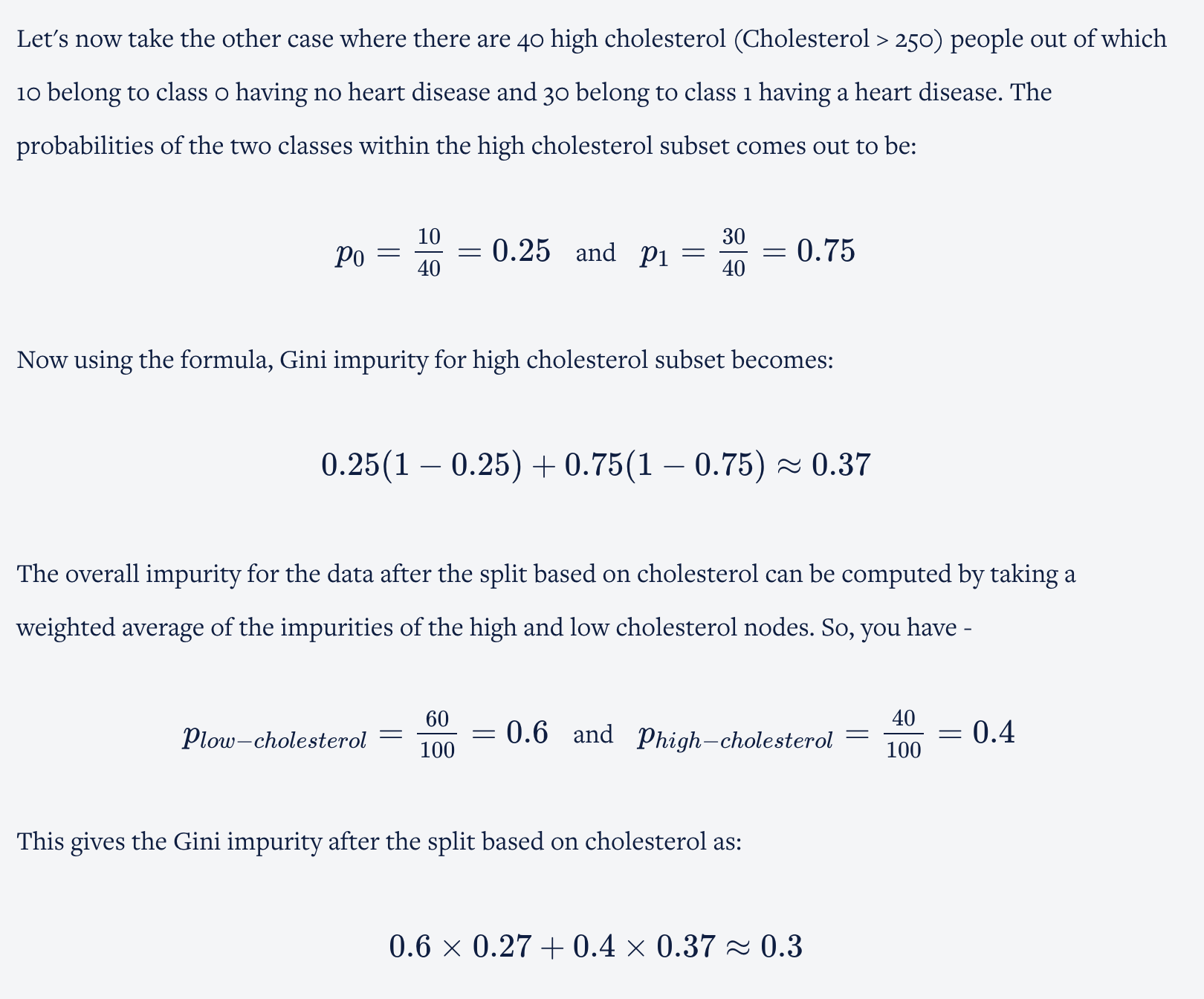
A diagram of a tree

Description automatically generated

For cholesterol –

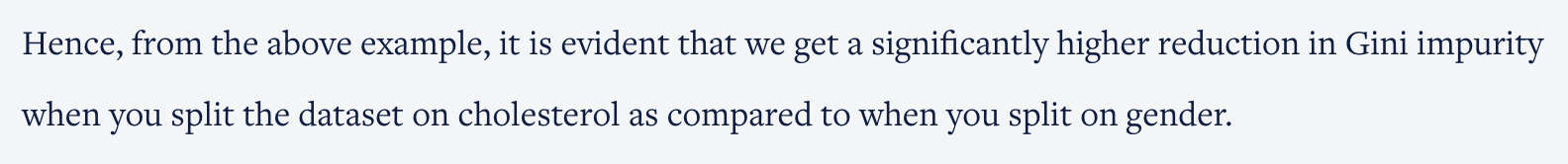
A screenshot of a computer

Description automatically generated



A diagram of a number of numbers

Description automatically generated



The feature that results in a **significant reduction in the impurity is the important variable**, and the one that results in **less impurity reduction is the less important variable**.

DT summary

* A decision tree first decides on an attribute to split on.
* To select this attribute, it measures the homogeneity of the nodes before and after the split.
* We can measure homogeneity in various ways with metrics like Gini index and entropy.
* Attribute that results in the increase of homogeneity the most is selected for splitting
* Then, this whole cycle is repeated until we obtain a sufficiently homogeneous data set.

**Hyperparameter Tuning in Decision Trees**

disadvantages of decision trees:

* **Overfitting**- They tend to overfit the data. If allowed to grow with no check on its complexity, a decision tree will keep splitting until it has correctly classified (or rather, mugged up) all the data points in the training set.
* **Unstable** - They tend to be quite unstable, which is an implication of overfitting. A few changes in the data can considerably change a tree.

There are two broad strategies to control overfitting in DT: **truncation & pruning**.

1. **Truncation** - Stop the tree while it is still growing so that it may not end up with leaves containing very few data points. Note that truncation is also known as **pre-pruning**.
2. **Pruning** - Let the tree grow to any complexity. Then, cut the branches of the tree in a bottom-up fashion, starting from the leaves. It is more common to use pruning strategies to avoid overfitting in practical implementations.

A diagram of a tree pruning strategies

Description automatically generated

Methods of truncation – (hyperparameters in bold – these are present in decisiontreeclassifier)

1. Limit the minimum size of the partition after the split. **min\_samples\_ split**
2. Minimize change in the measure of homogeneity
3. Limit the depth of the tree. **max\_depth**
4. Set a minimum threshold on number of samples that appear in a leaf. **min\_samples\_leaf**
5. Set a limit on maximum number of leaves present in a tree
6. **max\_features:** It defines the no. of features to consider when looking for the best split.

So using default parameters of DecisionTreeClassifier accuracy was very low. Therefore, I have performed hyperparameter tuning. Considered different hyperparameters and calculated accuracy of the model again and again. This is a trial and error approach.

**In DecisionTreeClassifier it uses gini to measure the quality of a split by default. We can use entropy also by adding criterion=”entropy”.**

you cannot always choose the best set of hyperparameters for the model manually. Instead, you can use **gridsearchcv()** in Python, which uses the **cross-validation** technique.

how to tune the hyperparameters to find their optimal values using **k-fold cross-validation**.

**A screenshot of a computer

Description automatically generated**

So basically in cross validation we split the training dataset into different folds. We will consider the folds one by one. If we consider fold 1 the fold 1 will be the validation set and other fold 2,3,4,5 will be training sets.

Specifically, you can apply the **k-fold cross-validation** technique, where you can divide the training data into k-folds/groups of samples. If k = 5, you can use k-1 folds to build the model and test it on the kth fold.

* The green and orange boxes constitute the training data.
* green ones are the actual training data and orange ones are the test (or validation) data points selected within the training dataset.
* training data is divided into 5 blocks or folds, and each time 4 blocks are being used as training data and the remaining one back is being used as the validation data.
* Once the training process is complete, you jump to model evaluation on the test data depicted by the blue box.

No we will try different combinations of hyperparameters and run data in each fold and calculate accuracy. Will consider the average of the accuracies of all folds.

GridsearchCV() function - helps you try out different combinations of hyperparameters which ultimately eased the process of figuring out these best values.

# Instantiate the grid search model

grid\_search = GridSearchCV(estimator=dt,

param\_grid=params,

cv=4, n\_jobs=-1, verbose=1, scoring = "accuracy")

here **cv** is the number of folds our training data set will get split into. 3 fols will be for training and 1 fold will be for validation.

The **n\_jobs** parameter in GridSearchCV controls the number of jobs (parallel threads) that are run simultaneously during the grid search process. This can significantly speed up the hyperparameter tuning process, especially when dealing with large datasets or complex models.

### Roles of n\_jobs

1. **Parallel Processing**:
   * By setting n\_jobs to a positive integer, you specify the number of CPU cores to use for parallel processing. For example, n\_jobs=4 will use 4 cores.
   * If you set n\_jobs=-1, it will use all available CPU cores, which maximizes parallelism.

The verbose parameter in GridSearchCV controls the level of verbosity, i.e., the amount of information printed to the console during the execution of the grid search process.

* **verbose=0 (default)**:
  + No output is printed. This is the default behavior.
  + Use this setting if you prefer not to see any progress messages.
* **verbose=1**:
  + Prints the overall progress of the grid search, including the number of parameter combinations processed and the estimated time remaining.
  + Use this setting for a basic level of progress monitoring.
* **verbose=2 or higher**:
  + Prints detailed information, including messages for each fold and each parameter combination.
  + Higher values result in more detailed output. For example, verbose=3 would print even more detailed information than verbose=2.

You can change or add more parameters and increase number of folds and do trial and error and see which combination is giving you the best result.

If we have one target variable value then it is easy to predict but if there are multiple continuous target values present in a node. We will take the average of those continuous values.

In decision tree regression, each leaf represents the average of all the values as the prediction as opposed to taking an majority vote in classification trees. This **average** is calculated using the following formula:

A mathematical equation with numbers and symbols

Description automatically generated

A black and white text with black text

Description automatically generated with medium confidence

And we calculate variance to calculate the impurity/purity of the node. If the node is completely pure there would be no variance. This is nothing but the variance of all data points.

The regression tree building process can be summarised as follows:

1. Calculate the MSE of the target variable.
2. Split the data set based on different rules obtained from the attributes and calculate the MSE for each of these nodes.
3. The resulting MSE is subtracted from the MSE before the split. This result is called the **MSE reduction**.
4. The attribute with the largest MSE reduction is chosen for the decision node.
5. The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until you get significantly low MSE and the node becomes as homogeneous as possible.
6. Finally, when no further splitting is required, assign this as the leaf node and calculate the average as the final prediction when the number of instances is more than one at a leaf node.

So, you need to split the data such that the weighted MSE of the partitions obtained after splitting is lower than that obtained with the original or parent data set. In other words, the fit of the model should be as ‘good’ as possible after splitting.