

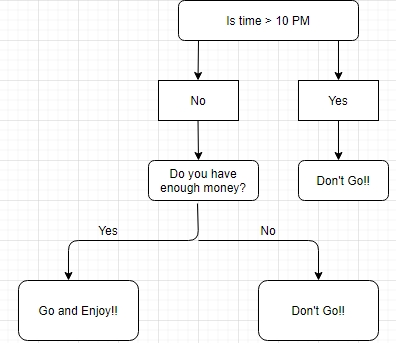
**Application Flow**

Decision Tree is one of the most fundamental algorithms for classification and regression in the Machine Learning world.

But before proceeding with the algorithm, let’s first discuss the lifecycle of any machine learning model. This diagram explains the creation of a Machine Learning model from scratch and then taking the same model further with hyper parameter tuning to increase its accuracy, deciding the deployment strategies for that model and once deployed setting up the logging and monitoring frameworks to generate reports and dashboards based on the client requirements. A typical lifecycle diagram for a machine learning model looks like:

**Decision Tree**

Decision tree algorithm is one of the most versatile algorithms in machine learning which can perform both classification and regression analysis. It is very powerful and works great with complex datasets. Apart from that, it is very easy to understand and read. That makes it more popular to use. When coupled with ensemble techniques – which we will learn very soon- it performs even better. As the name suggests, this algorithm works by dividing the whole dataset into a tree-like structure based on some rules and conditions and then gives prediction based on those conditions. Let’s understand the approach to decision tree with a basic scenario. Suppose it’s Friday night and you are not able to decide if you should go out or stay at home. Let the decision tree decide it for you.



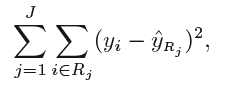
Although we may or may not use the decision tree for such decisions, this was a basic example to help you understand how a decision tree makes a decision. So how did it work?

* It selects a root node based on a given condition, e.g. our root node was chosen as time >10 pm.
* Then, the root node was split into child notes based on the given condition. The right child node in the above figure fulfilled the condition, so no more questions were asked.
* The left child node didn’t fulfil the condition, so again it was split based on a new condition.
* This process continues till all the conditions are met or if you have predefined the depth of your tree, e.g. the depth of our tree is 3, and it reached there when all the conditions were exhausted.

Let’s see how the parent nodes and condition is chosen for the splitting to work.

**Decision Tree for Regression**

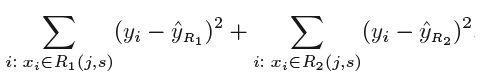
When performing regression with a decision tree, we try to divide the given values of X into distinct and non-overlapping regions, e.g. for a set of possible values X1, X2,..., Xp; we will try to divide them into J distinct and non-overlapping regions R1, R2, . . . , RJ. For a given observation falling into the region Rj, the prediction is equal to the mean of the response(y) values for each training observations(x) in the region Rj. The regions R1,R2, . . . , RJ are selected in a way to reduce the following sum of squares of residuals :



Where, yrj (second term) is the mean of all the response variables in the region ‘j’.

**Recursive binary splitting(Greedy approach)**

As mentioned above, we try to divide the X values into j regions, but it is very expensive in terms of computational time to try to fit every set of X values into j regions. Thus, decision tree opts for a top-down greedy approach in which nodes are divided into two regions based on the given condition, i.e. not every node will be split but the ones which satisfy the condition are split into two branches. It is called greedy because it does the best split at a given step at that point of time rather than looking for splitting a step for a better tree in upcoming steps. It decides a threshold value(say s) to divide the observations into different regions(j) such that the RSS for Xj>= s and Xj <s is minimum.

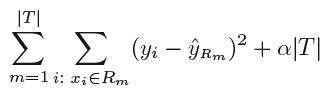


Here for the above equation, j and s are found such that this equation has the minimum value. The regions R1, R2 are selected based on that value of s and j such that the equation above has the minimum value. Similarly, more regions are split out of the regions created above based on some condition with the same logic. This continues until a stopping criterion (predefined) is achieved. Once all the regions are split, the prediction is made based on the mean of observations in that region.

The process mentioned above has a high chance of overfitting the training data as it will be very complex.

**Tree Pruning**

Tree pruning is the method of trimming down a full tree (obtained through the above process) to reduce the complexity and variance in the data. Just as we regularised linear regression, we can also regularise the decision tree model by adding a new term.



Where, T is the sub tree which is a subset of the full tree T0 And α is the non-negative tuning parameter which penalises the MSE with an increase in tree length. By using cross-validation, such values of α and T are selected for which our model gives the lowest test error rate. This is how the decision tree regression model works. Let’s now see the working algorithm of doing classification using a decision tree. Greedy Algorithm As per Hands-on machine learning book “greedy algorithm greedily searches for an optimum split at the top level, then repeats the process at each level. It does not check whether or not the split will lead to the lowest possible impurity several levels down. A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution.”

**Post-pruning**

Post-pruning, also known as backward pruning, is the process where the decision tree is generated first and then the non-significant branches are removed. Cross-validation set of data is used to check the effect of pruning and tests whether expanding a node will make an improvement or not. If any improvement is there then we continue by expanding that node else if there is reduction in accuracy then the node not be expanded and should be converted in a leaf node.

**Pre-pruning**

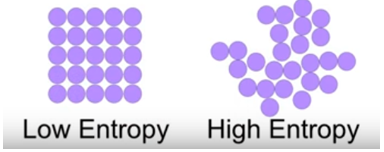
Pre-pruning, also known as forward pruning, stops the non-significant branches from generating. It uses a condition to decide when should it terminate splitting of some of the branches prematurely as the tree is generated.

**Classification Trees**

Regression trees are used for quantitative data. In the case of qualitative data or categorical data, we use classification trees. In regression trees, we split the nodes based on RSS criteria, but in classification, it is done using classification error rate, Gini impurity and entropy. Let’s understand these terms in detail.

**Entropy**

Entropy is the measure of randomness in the data. In other words, it gives the impurity present in the dataset.



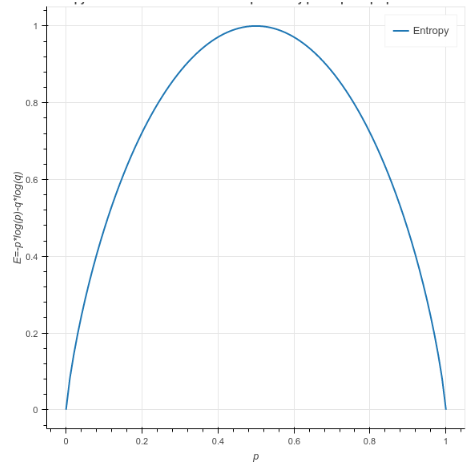
When we split our nodes into two regions and put different observations in both the regions, the main goal is to reduce the entropy i.e. reduce the randomness in the region and divide our data cleanly than it was in the previous node. If splitting the node doesn’t lead into entropy reduction, we try to split based on a different condition, or we stop. A region is clean (low entropy) when it contains data with the same labels and random if there is a mixture of labels present (high entropy). Let’s suppose there are ‘m’ observations and we need to classify them into categories 1 and 2. Let’s say that category 1 has ‘n’ observations and category 2 has ‘m-n’ observations.

p= n/m and q = m-n/m = 1-p

then, entropy for the given set is:

E = -p\*log2(p) – q\*log2(q)

When all the observations belong to category 1, then p = 1 and all observations belong to category 2, then p =0, in the both cases E =0, as there is no randomness in the categories. If half of the observations are in category 1 and another half in category 2, then p =1/2 and q =1/2, and the entropy is maximum, E =1.



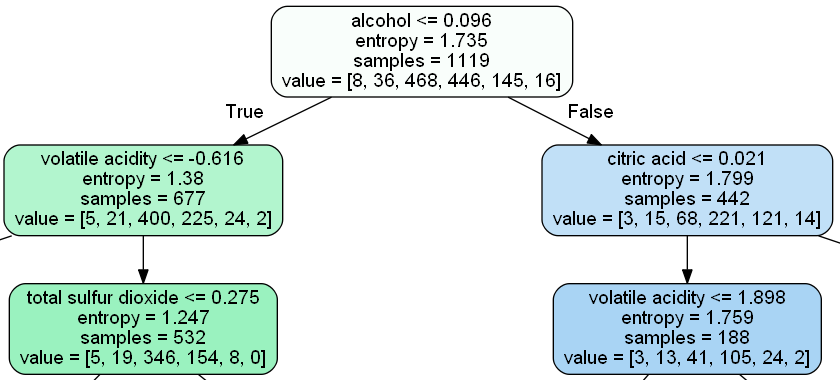
**Information Gain**

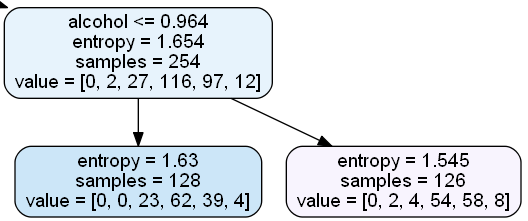
Information gain calculates the decrease in entropy after splitting a node. It is the difference between entropies before and after the split. The more the information gain, the more entropy is removed.

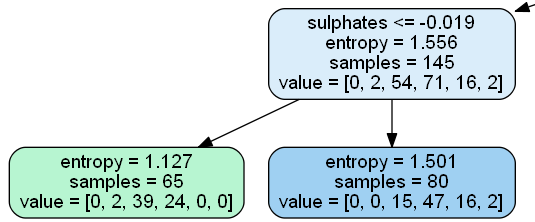


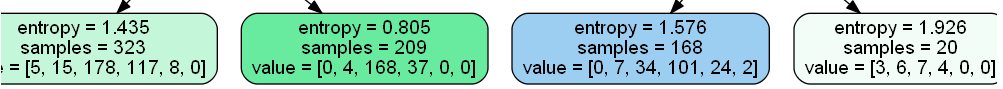
Where, T is the parent node before split and X is the split node from T.

**A tree which is splitted on basis of entropy and information gain value looks like:**



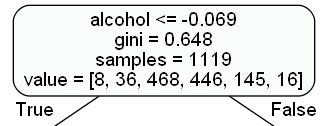


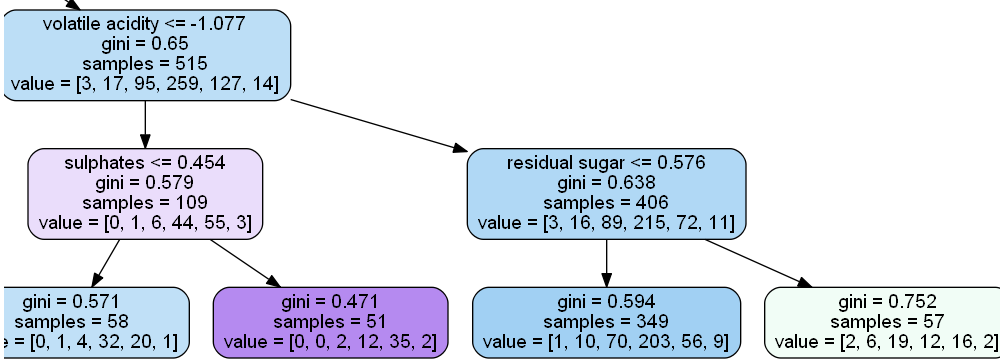
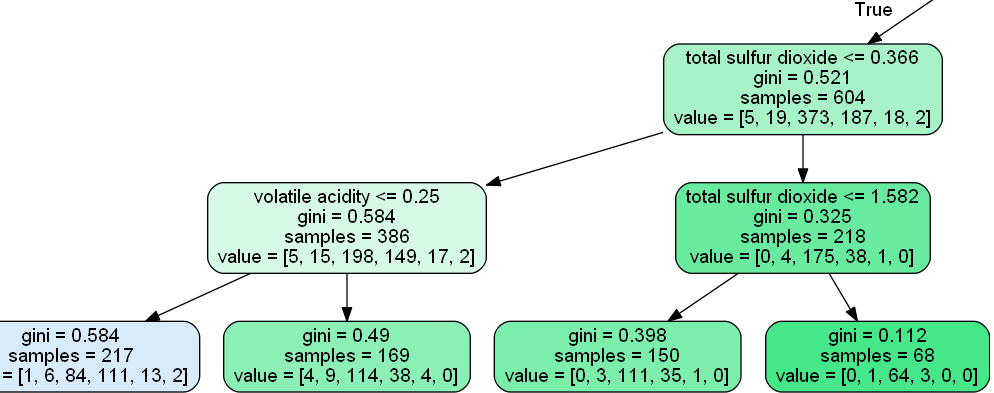




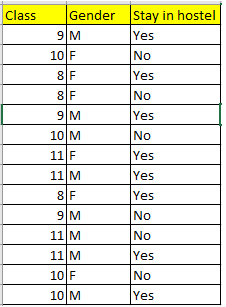
**Gini Impurity**

According to wikipedia, ‘Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset.’ It is calculated by multiplying the probability that a given observation is classified into the correct class and sum of all the probabilities when that particular observation is classified into the wrong class. Let’s suppose there are k number of classes and an observation belongs to the class ‘i’, then Gini impurity is given as:





**Maths behind Decision Tree Classifier**

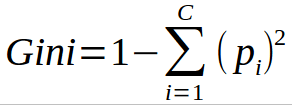
Before we see the python implementation of decision tree. let's first understand the math behind the decision tree classification. We will see how all the above mentioned terms are used for splitting.

We will use a simple dataset which contains information about students of different classes and gender and see whether they stay in school's hostel or not. This is how our data set looks like :

Let's try and understand how the root node is selected by calculating gini impurity. We will use the above mentioned data.

We have two features which we can use for nodes: "Class" and "Gender". We will calculate gini impurity for each of the features and then select that feature which has least gini impurity.

Let's review the formula for calculating gini impurity:

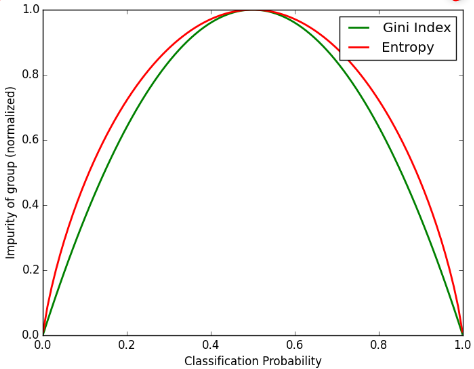


**Different Algorithms for Decision Tree**

* ID3 (Iterative Dichotomiser) : It is one of the algorithms used to construct decision tree for classification. It uses Information gain as the criteria for finding the root nodes and splitting them. It only accepts categorical attributes.
* C4.5 : It is an extension of ID3 algorithm, and better than ID3 as it deals both continuous and discreet values. It is also used for classification purposes.
* Classification and Regression Algorithm(CART) : It is the most popular algorithm used for constructing decision trees. It uses ginni impurity as the default calculation for selecting root nodes, however one can use "entropy" for criteria as well. This algorithm works on both regression as well as classification problems. We will use this algorithm in our python implementation.

Entropy and Ginni impurity can be used reversibly. It doesn't affects the result much. Although, ginni is easier to compute than entropy, since entropy has a log term calculation. That's why CART algorithm uses ginni as the default algorithm.

If we plot ginni vs entropy graph, we can see there is not much difference between them:



**Advantages of Decision Tree:**

* It can be used for both Regression and Classification problems.
* Decision Trees are very easy to grasp as the rules of splitting is clearly mentioned.
* Complex decision tree models are very simple when visualized. It can be understood just by visualising.
* Scaling and normalization are not needed.

**Disadvantages of Decision Tree:**

* A small change in data can cause instability in the model because of the greedy approach.
* Probability of overfitting is very high for Decision Trees.
* It takes more time to train a decision tree model than other classification algorithms.

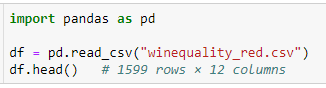
**Implementation in Python**

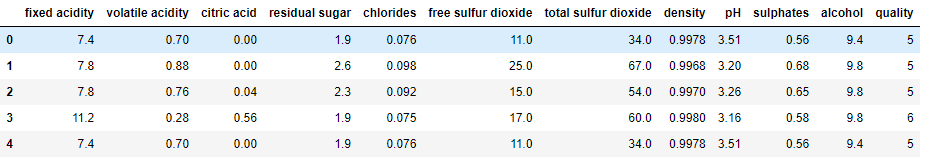
we will use Sklearn module to implement decision tree algorithm. Sklearn uses CART (classification and Regression trees) algorithm and by default it uses Gini impurity as a criteria to split the nodes.

There are other algorithms like ID3, C4.5, Chi-square etc.

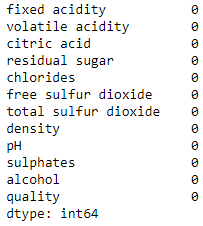
We will see the use of CART in following implementation.

**1. import the data**

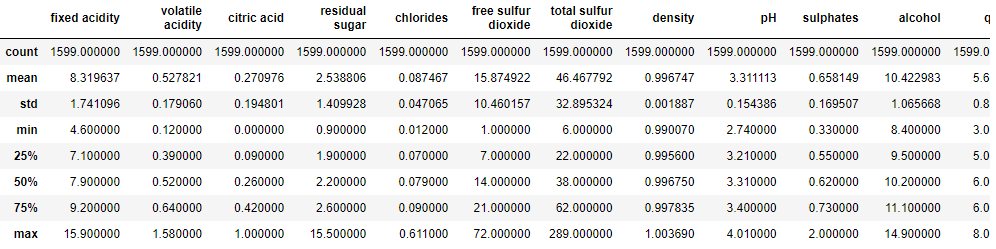












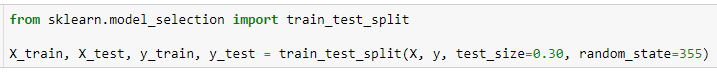
**2. preprocessing the data**

We can see there is no missing data in the columns.Hence it is balanced dataset no need of preprocessing

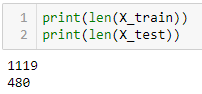
**3. Identify the X(independent) and y(dependent variables)**



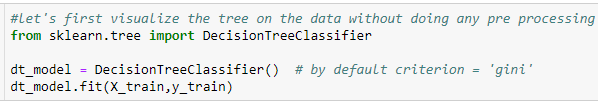
**4. Split the data into Train and Test**





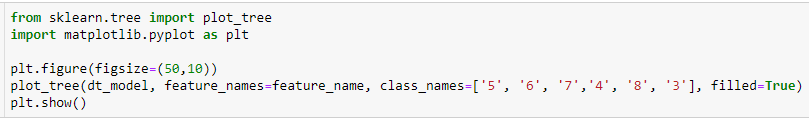


**5. Fit / Train the model using train data**



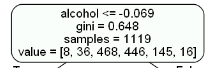
DecisionTreeClassifier()







**Let's understand the above tree:**



* the first value indicates the column and the condition on which the root node was selected and further will be splitted
* the second value gives the gini impurity of the selected node
* samples gives the number of observations at that point of time present in the node
* value within the square brackets represents number of observations present in each class(output) i.e. in the above given figure, 8 observations are in class 1, 38 in class 2 , 468 in class 3 and so on.

Then the split was made on the basis of given condition.

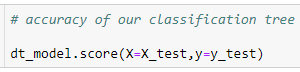


1.0

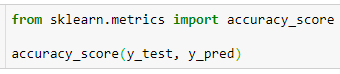
**6. predict on test data**



**7. Accuracy**

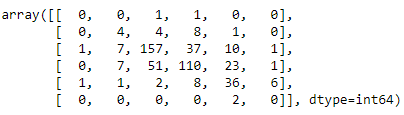


0.6375



0.6375

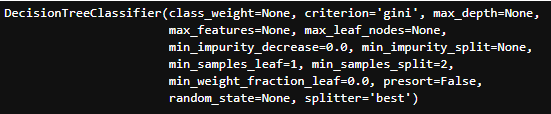




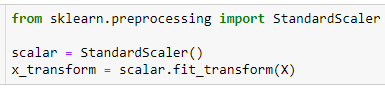
Now we haven't done any preprocessing with our data and neither done any hyper parameter tunings. Let's do all those and see how our score improves.

**What are hyper parameters?**

**8. if we are not happy with accuracy then do hyper parameter tunning and rebuild the model**



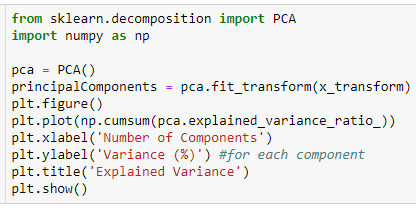
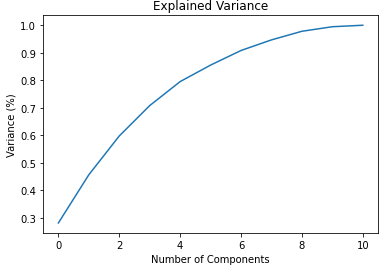
**Preprocessing**



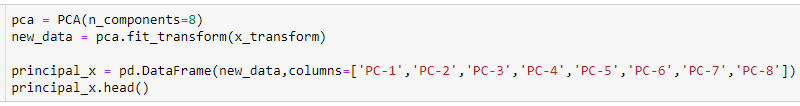


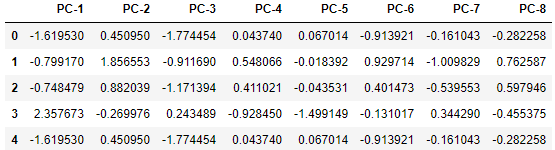
Although our dataset is realtively small, let's use PCA for feature selection and see if it improves our accuracy.

from above dataset we are using 11 features as X by using PCA we can consider contributing features remaining all we can drop off

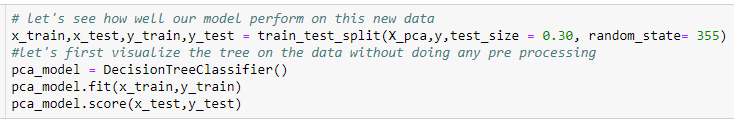


We can see that around 95% of the variance is being explained by 8 components. So instead of giving all 11 columns as input in our algorithm let's use these 8 principal components instead.

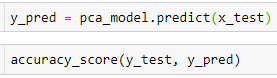






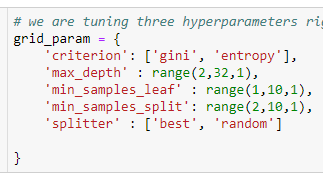


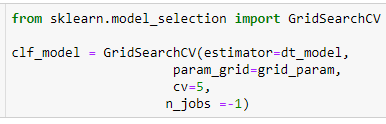
0.5895833333333333

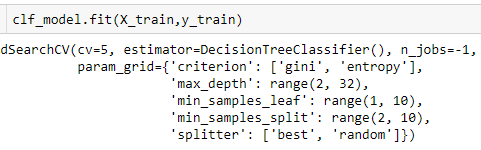


0.5895833333333333

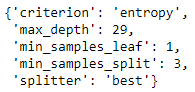
GridSearchCV is a method used to tune our hyperparameters. We can pass different values of hyperparameters as parameters for grid search. It does a exhaustive generation of combination of different parameters passed. Using cross validation score, Grid Search returns the combination of hyperparameters for which the model is performing the best.







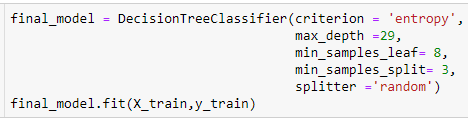






0.6094851057014734

**9. final accuracy score from final model**



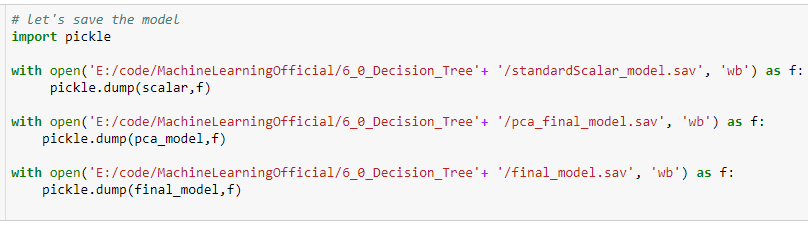
DecisionTreeClassifier(criterion='entropy', max\_depth=29, min\_samples\_leaf=8,

min\_samples\_split=3, splitter='random')



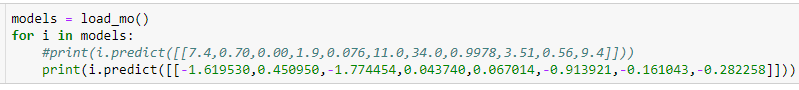


0.5895833333333333



**load model**





[5]