Contents

[1. Precision , Recall & F-Score 2](#_Toc102892786)

[2. Logistic Regression 7](#_Toc102892787)

[Why Linear regression cannot be used for Classification and why Logistic Regression is needed instead ? 7](#_Toc102892788)

[Logistic Regression | Intro - 9](#_Toc102892789)

[Logit Function, Odds , log(odds) & Probability – 10](#_Toc102892790)

[Relationship Between Odds and Probability : 11](#_Toc102892791)

[Why do we need Log(Odds) and what is the practical significance ? 11](#_Toc102892792)

[How does Logistic Regression work – 13](#_Toc102892793)

[Advantages and Disadvantages 19](#_Toc102892794)

[Logistic Regression implementation in Python 19](#_Toc102892795)

[Loading and Preparing Voice data for Model training – 19](#_Toc102892796)

[Logistic Regression model training and Performance measurement – 21](#_Toc102892797)

[3. Decision Trees 23](#_Toc102892798)

[Pure/Impure datasets 25](#_Toc102892799)

[Gini Index 26](#_Toc102892800)

[Information Gain : 28](#_Toc102892801)

[Entropy 30](#_Toc102892802)

[Decision Tree Algorithm : 31](#_Toc102892803)

[Gini Index and Information Gain for Continuous features – 36](#_Toc102892804)

[Advantages and Drawbacks : 39](#_Toc102892805)

[Python Implementation of Decision Trees 40](#_Toc102892806)

[4. K- Nearest Neighbours ( KNN Algorithm ) 43](#_Toc102892807)

[Drawbacks and Advantages of KNN algorithm 47](#_Toc102892808)

[Python Implementation – 48](#_Toc102892809)

[5. MCQ 51](#_Toc102892810)

[6. Homework 53](#_Toc102892811)

# Precision , Recall & F-Score

Follow the content, it is exhaustive with concepts as well as examples

Do stress on the calculations part and if possible make the students calculate along the way. Manual calculations always provide good clarity about mathematical concepts.

If needed and possible, calculations can be done in Excel as well by both educator and students.

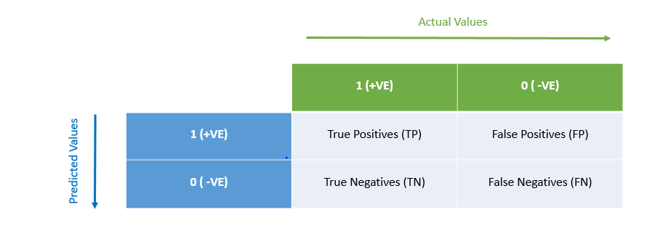
Before we dive into Classification algorithms, it is important that we discuss some more performance metrics for classification.

On Day 1, we had discussed few of the metrics like Accuracy, Type 1 error and Type 2 error. We also discussed how accuracy is considered a good metric for balanced dataset, but for unbalanced datasets there are some other metrics also that one can consider. These are namely –

1. Recall
2. Precision
3. F-Beta Score

It is also important to know about these because most of the Python “*scoring*” functions would return these as output, and it is always beneficial to understand thoroughly what we see as output.

We have already discussed the concept of Confusion Matrix earlier, but for sake of continuity, let’s look at the Confusion matrix for a binary classification below –



1. **Recall / True Positive Rate :**

For a binary classification, it is a score which tells us the ratio of **correct +ve class** predictions with respect to **all +ve class predictions**. Recall is also known as True Positive Rate. We’ll look at the mathematical formula for better understanding -

For example , for a confusion matrix as below –

|  |  |
| --- | --- |
| 5 | 4 |
| 3 | 1 |

TPR = 5/8

* 0.625

As a thumb rule, recall can be calculated by dividing the total correct +ve class predictions by the sum of entire column.

**Why Recall is used –**

Recall is used to control Type-II Error. Thus, a model with higher recall is expected to have lower Type-II error because the True Positives have been maximized while we also have False negatives in denominator. Hence a better Recall score would definitely reduce Type-II error.

For example –

In medical situations, controlling Type-2 errors is of more importance than Type-1 errors. Such as in a Cancer prediction model, it is okay to predict a healthy patient as a cancer patient ( Type – 1 error ), but it would be too dangerous to treat a Cancer patient as a healthy patient ( Type- 2 error). In latter case, the patient might lose his life due to this error, but in former case there is no such devastating danger of loss.

Improving Recall for such model would make sure that no cancer patient is predicted incorrectly to be a healthy patient, which is what is needed in such scenario.

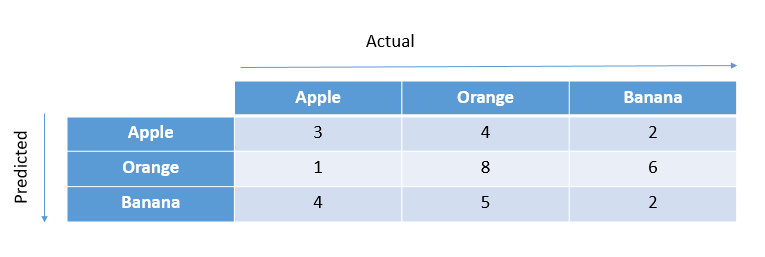
**Recall for multi-class prediction models –**

In multiclass models, there is no such concept as +ve class or –ve class.

For example, let’s assume a model which predicts a fruit name out of Orange, Apple and Banana. In such case, Recall will be calculated separately for each class.

An example of same has been discussed below.

Let’s assume the confusion matrix of above model as :



We need to go by definition of Recall which is -

*“Ratio of* ***correct +ve class*** *predictions with respect to* ***all +ve class predictions****.”*

The only difference is that here +ve class would be replaced by each class label iteratively.

Thus,

Recall for Apple = 3/(3+1+4) => 0.375

As a thumb rule, it is the correct predicted value divided by the sum of entire column for that class.

Recall for Orange = 4/(8+4+5) => 0.235

Recall for Banana = 2/(2+6+2) => 0.2

Recall is maximum for Apple, which means model is best for Apple class in terms of Type 2 error control.

1. **Precision :**

For a binary classification, It is the ratio of **correct +ve class predictions** with respect to **Total +ve class predictions**.

Mathematically,

For a confusion matrix as below –

|  |  |
| --- | --- |
| 5 | 4 |
| 3 | 1 |

Precision = 5/(5+4) => 5/9 => 0.56

As a thumb rule, precision is Correct Positive class predictions divided by the sum of entire row.

**Why precision is used -**

In contrast to TPR, precision is used to control Type 1 errors. Better the precision score, lower the Type 1 error in model.

A practical example –

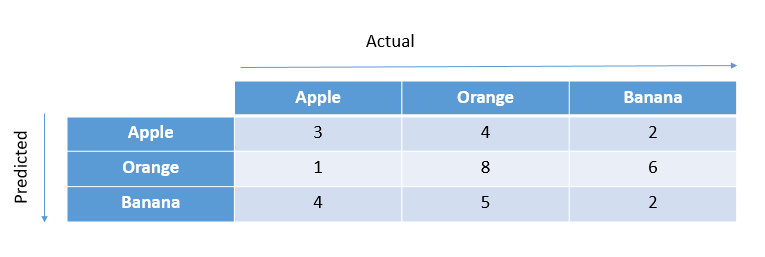
Spam detection model.

In such models, it is okay to miss marking some of the actual Spams as spam. The worst that will happen is that the user will read a spam mail.

But one can not afford missing an important mail because it was incorrectly tagged as spam. This can cause severe impacts. For example a person can miss the meeting invite to a very important meeting due to such error.

**Precision for multiclass models –**

Let’s take the same confusion matrix from Recall example –



Going by the Precision definition in this case we can calculate the same way as we did in case of Recall.

The definition says – *“ ratio of* ***correct +ve class predictions*** *with respect to* ***Total +ve class predictions****. “*

The only difference is that here +ve class will be replaced by Class labels as there is no concept of +ve /-ve class in multiclass models.

Precision for Apple : 3/(3+4+2) => 3/9 => 0.33

We can see that as a thumb rule, precision is Correct predictions divided by the sum of entire row.

Precision for Orange : 8/(1+8+6) = > 0.53

Precision for Banana : 2/(4+5+2) => 0.18

1. **F-Beta Score :**

Now that we have understood Precision and Recall, it is easy to get the idea behind F-Beta.

Generally, there might be cases where both Type-1 and Type-2 errors are of huge impact and it is important to control both. Or there may be cases where although both are important, but comparatively controlling Type-1 is of more importance than Type-2 and vice versa.

Precision and Recall only help us in controlling either Type-1 or Type-2 error, but not both at the same time.

This is achieved by the use of F-Beta Score.

The mathematical formula for F-Beta is given as :

There are 2 important things to note here –

1. F Beta uses both Precision and Recall in its formula, and thus it is able to take into account both
2. There is a variable which has following significance :

*At =1,*

the formula becomes :

This is nothing but the harmonic mean of 2 metrics. Which basically means that both are given equal significance. This formula at =1 is also known as **F1 Score**.

*At =0.5,*

The coefficient with Precision in denominator is squared, and is less than 1, thus Precision gets more significance overall.

Thus, at *<1,* Type-I errors are controlled better than Type-2 errors. Lower the value for, more prominent the role of Precision is*.*

On the other hand, at *>1,* Type-2 errors are controlled better than Type-1 errors. Higher the value for, more prominent the role of Recall becomes.

In Python, all of these performance metrics are available as scoring parameters in GridSearchCV function. We’ll be implementing the same later today.

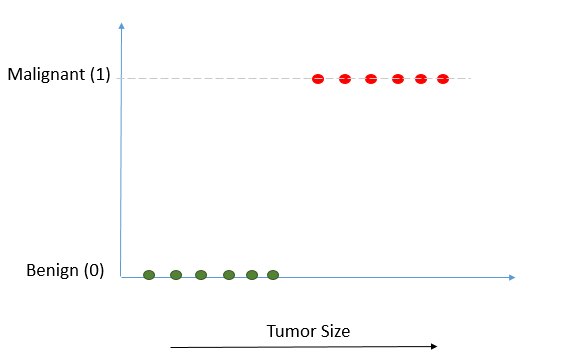
# Logistic Regression

Logistic Regression, contrary to its name, is a Classification algorithm. It is mainly used for Binary Classification, although it is also capable of multi class prediction as well.

It has “Regression” in its name because the backend mathematics is similar to Linear Regression. It also belongs to Regression class algorithms in ML, but the use case is classification prediction.

## Why Linear regression cannot be used for Classification and why Logistic Regression is needed instead ?

let’s assume a Binary Classification problem. Let’s take our Cancer classification example, where we need to Predict whether a Cancer tumor is Malignant or Benign ( target) , according to the tumor size ( Independent Variable).



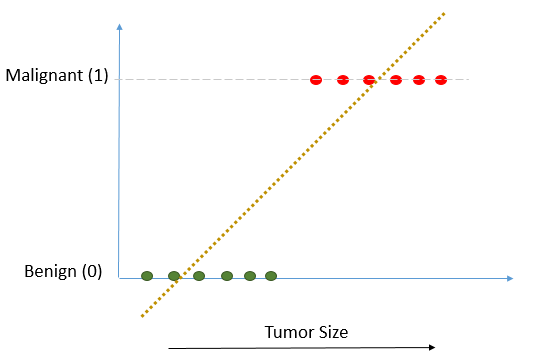
Since there are only 2 Target labels, these are represented by 0s and 1s on Y Axis. Thus, all the independent datapoints will take either of the 2 values 0 or 1 only. This is very different from a regression problem where datapoints are scattered across X and Y axis and a linear regression line is fitted.

**Problem :**

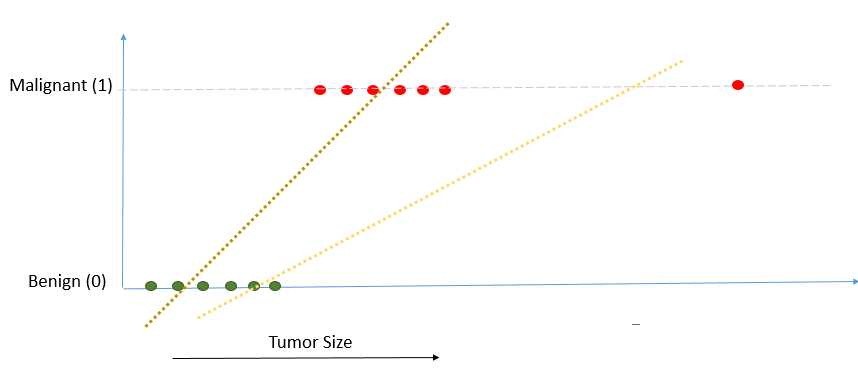
A linear regression line can also be fitted here, but it because the data points are organized in a really different manner, such a fitted line would be very unstable and sensitive to new data points and outliers. This is one of the reasons Linear Regression cannot be used for Binary Classification.

Let’s try to understand the same through visuals -

If a regression line is fitted to the same data as above, it may look like this –



Now let’s say a new data point which is also an outlier, is added, and the linear regression line is again plotted, it will look as below –



It can be seen that with just 1 outlier, the line has shifted so much. This new line will be predicting significantly different predictions than the first line, just because of addition of 1 outlier. Such sensitive and unstable models cannot be accepted.

Also, it can be observed, that Linear Regression line will have the prediction range from - to +, but our requirement is to predict in the range of 0 to 1 only. This is another reason Linear Regression is not suitable for such cases.

Just to summarise, Linear regression cannot be used for Classification because –

* It will be extremely sensitive to new data points and outliers if used
* The range is not between desirable bandwidth of 0 to 1.

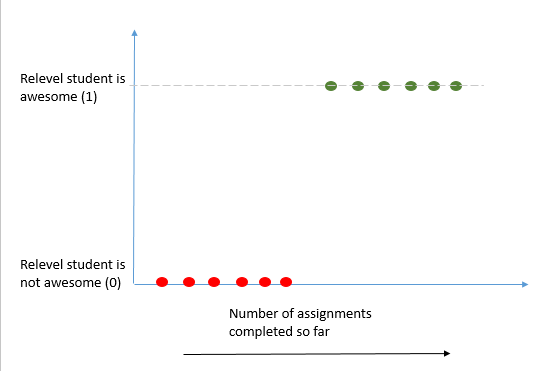
In order to overcome these problems, Logistic Regression is used.

## Logistic Regression | Intro -

Let’s get a high level understanding of Logistic regression with some pointers, and we will dive into technical part later –

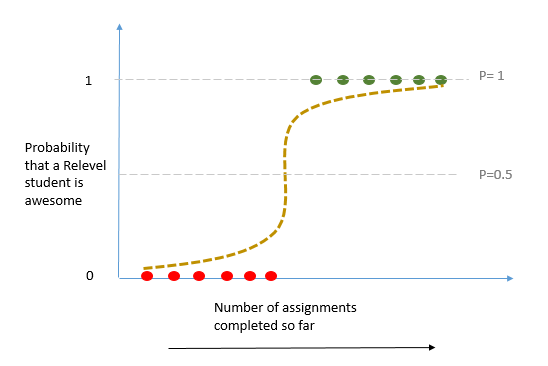
* **Used for Binary Classification -** Logistic Regression is used for Binary Classification (most of the times). For example, its job is usually to predict whether something is True or False, Big or Small, Dead or Alive etc. To make things easier we usually assign 0 and 1 labels to these 2 Target labels. Thus, in more simpler terms, Logistic regression predicts something to belong to either 0 or 1.

This is contrary to Linear Regression which can predict over a range of values.



* **Fits and S Shaped Curve -** Instead of fitting a straight line to the data, Logistic Regression fits and S shaped Curve. The range of this Curve is 0 to 1 on y Axis, and Y axis represents Probability. Probability 0 belongs to –ve class and 1 belongs to +ve class.

We’ll get into high level details of how Logistic Regression fits this S curve in a few minutes.



* Just like Linear Regression, Logistic Regression can also work with both Continuous and Categorical features.

**Differences between Linear Regression and Logistic Regression –**

* Linear Regression uses Residuals and Least Squared method. It also calculates R2 for comparison.

Logistic regression does not and cannot use Residuals or Least squared method. It uses a concept called *Maximum Likelihood*, which we’ll discuss in this session. An R2 value cannot be derived for Logistic regression.

( Although theoretically in advance mathematics , statisticians do calculate R2 for Logistic regression sometimes, but there are multiple methods and there is little consensus among statisticians on which method should be accepted as standard. )

* The range for Logistic Regression predictor is 0 to 1 , while for Linear regression is can be any real number.
* Logistic Regression is used for Classification and Linear regression is used for regression models.

## Logit Function, Odds , log(odds) & Probability –

Logistics regression uses the concept of Maximum Likelihood to fit an S Curve to the data. The process behind logistic regression also includes a log(odds) transformation using Logit function. In order to understand the what, why and how of that transformation and Maximum likelihood, it is important that we understand about odds and log(odds) and Logit function first.

Let’s take an example that India wins 2 out of 5 cricket matches. In this scenario the odds of India winning would be 2:3 or 2/3.

Let’s look at the formula for Odds , log(odds) and Probability ; and apply that to Cricket match example-

**Odds = Count of Something Happening / Count of something not happening**

* **Won Matches / Lost Matches** = > 2/3 => 0.67

**Probability = Count of Something happening/ Count of total things happening**

* **Won matches / Total Matches** => 2/5 => 0.4

**Log( Odds) of winning = log( 0.67 ) => -0.5778**

Thus, we can see that Odds and Probability are different. Log(Odds) is literally log of odds and nothing else.

### Relationship Between Odds and Probability :

Odds can be also be rewritten as :

(For proof, one can substitute the probability formula in above equation, the denominators will cancel out and we’ll be left with original Odds formula discussed above. )

This can be rearranged as –

* **General Notation => Odds =**

Taking log both sides ;

* **Log(Odds ) =**

**This is also known as the Logit function which forms the basis for Logistic Regression.** How, that we’ll discuss soon.

### Why do we need Log(Odds) and what is the practical significance ?

In order to understand the significance of log(odds), we need to look at the same Cricket example with multiple cases of Win-Lose ratio in parallel –

|  |  |
| --- | --- |
| **Scenario – Chances of India winning are low ( Lost Matches > Won Matches )** | |
| Cases | Odds Winning |
| Case 1 : 1 Win and 4 Losses | 1/4   * 0.25 |
| Case 2 : 1 Win and 8 Losses | 1/4   * 0.25 |
| Case 3 : 1 Win and 16 Losses | 1/16   * 0.063 |

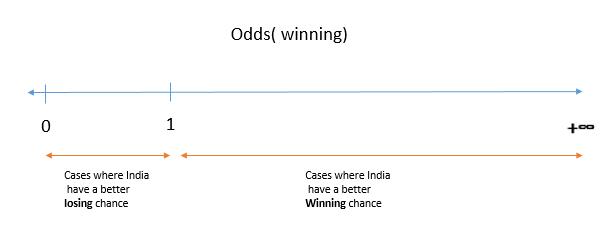
**Inference ->** Worse the Indian team Performs, Odds of winning tend to go to 0. The range of Odds in this scenario where India have more chance of losing lie between **0 to 1**

Now let’s look at the scenario where India have better chances of winning ( India have won more matches than lost matches)

|  |  |
| --- | --- |
| **Scenario – Chances of India winning are High ( Lost Matches < Won Matches )** | |
| Cases | Odds Winning |
| Case 1 : 4 Win and 3 Losses | 4/3   * 1.33 |
| Case 2 : 8 Win and 3 Losses | 8/3   * 2.7 |
| Case 3 : 32 Win and 3 Losses | 32/3   * 10.7 |

**Inference ->** Better the Indian team Performs, Odds of winning tend to go to +∞. The range of Odds in this scenario where India have more chance of winning lie between **1 to +∞**

If we plot these inferences on a number line, it will be as below –



Finally, the whole point of this exercise is to show that “Odds” is an asymmetrical metric. For negative cases it has range 0 to 1 while for Positive cases the range is 1 to +Infinity.

Due to this asymmetry, *Odds* as a metric is rarely used directly in Statistics or ML applications. Rather, *Log(Odds)* is used in maximum cases which transforms *Odds* to become Symmetrical in nature.

This can be shown with the help of following contrasting cases example –

|  |  |
| --- | --- |
| Case 1: 1 Win and 6 Losses | Case 2: 6 Win and 1 Losses |
| Odds = 1/6 => 0.167  Log(Odds) = log(0.167) => **-2.582** | Odds = 6/1 => 6  Log(Odds) = log(6) => **+2.582** |

Inference -> While the Odds are asymmetrical in nature for above opposite cases, the log(odds) are symmetrical with same magnitude but opposite sign.

**Conclusion** : Taking log(odds) makes everything symmetrical, and hence used in place of Odds.

## How does Logistic Regression work –

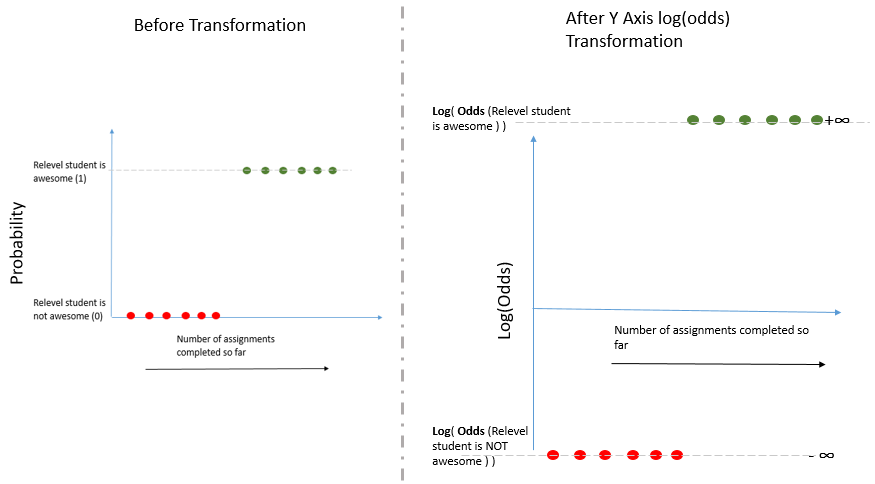
**Step 1 :**

Logistic Regression uses the log(odds) on y-Axis. This means that the Probability on Y axis is transformed to log(odds) using Logit function discussed above.

**Reason for Transformation :** The reason for using log(odds) is because we know it makes things symmetrical and maths easier. By this transformation we can fit a straight line which can be transformed back to S curve with the help of Logit function. We cannot fit an S curve directly, and this is why Y Axis is transformed to log(odds).

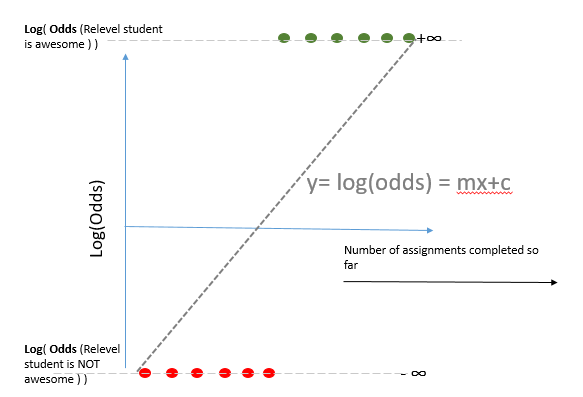
**Effect on datapoints :** Subsequently, this would mean that now data points which earlier had 1 and 0 probabilities in earlier chart would have values as +∞ an - ∞ respectively.

This transformation is shown in visuals below for better grasp :



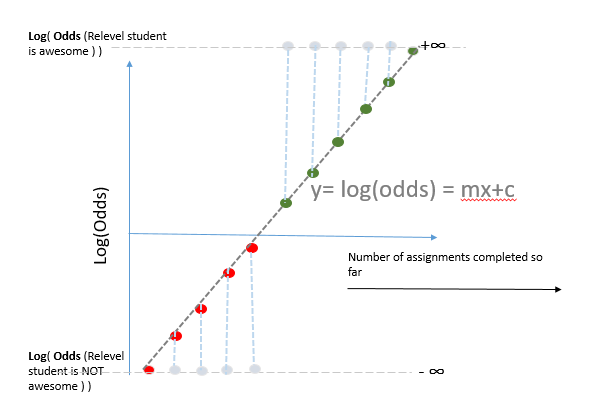
**Step 2 :**

An arbitrary line is drawn on the transformed axis, with the objective as to find the best fit line.

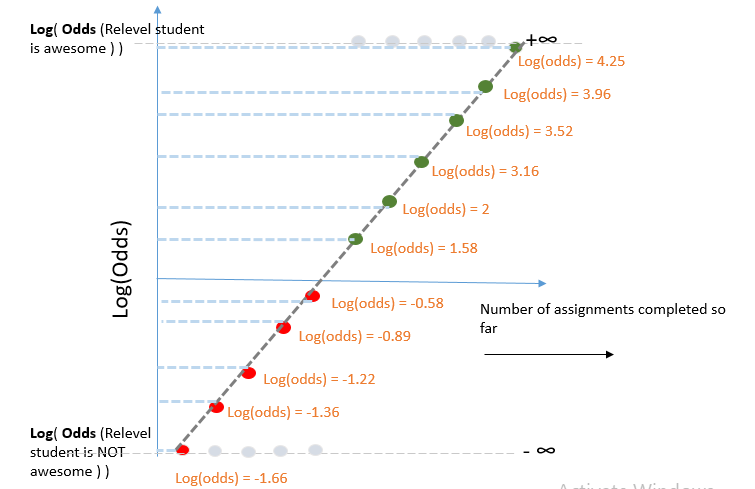


(Later on, the best fit is chosen as per Maximum Likelihood calculation ( Explained later) and Gradient Descent process.)

Since the datapoints are at extreme ends on this axis (+∞ an - ∞) , they are projected on to the line as shown in the visual below.



After projection, each data point will have a corresponding log(odds) value.



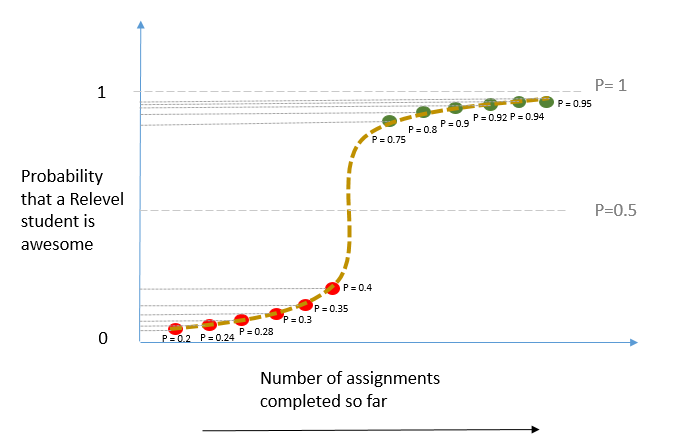
**Step 3 :**

For each datapoint, log(odds) can be converted in Probabilities (P) by using Sigmoid function –

( Sigmoid function is nothing but Logit function rearranged. It is the inverse of Logit function )

This formula is just a reordering of earlier Logit function that we discussed, let’s skip derivation to save time.

Geometrically, this function is transforming back out log(odds) axis to Probabilities. Our Straight line is convert into an S-Curve with the help of this function.



**Step 4 :**

Calculate Likelihood of this particular Log(odds) line or S Curve.

Likelihood of the line or S Curve is given by multiplication of all the Probabilities for all the datapoints as following –

Likelihood =

0.95 \* 0.94 \* 0.92 \* 0.9 \* 0.8 \* 0.75 \* (1-0.4) \* (1-0.35) \* (1-0.3) \* (1-0.28) \* (1-0.24) \* (1-0.2)

=> 0.0530119

Please observe that for negative class, (1-P) is used in likelihood formula as for those point we have to calculate Probability of those being 0.

Note:

Although for the sake of simplicity we have used Multiplication of Probabilities to calculate likelihood, but mostly statisticians prefer multiplication of log(P) instead. This is because use of log(P) makes the calculations simpler. A lot of computer based algorithm functions also use the same to save computation.

But, the result would be same in any case and the choice of log or not won’t impact the final outcome at all.

In case of log(P) based likelihood, the calculation would be as follows –

Likelihood =

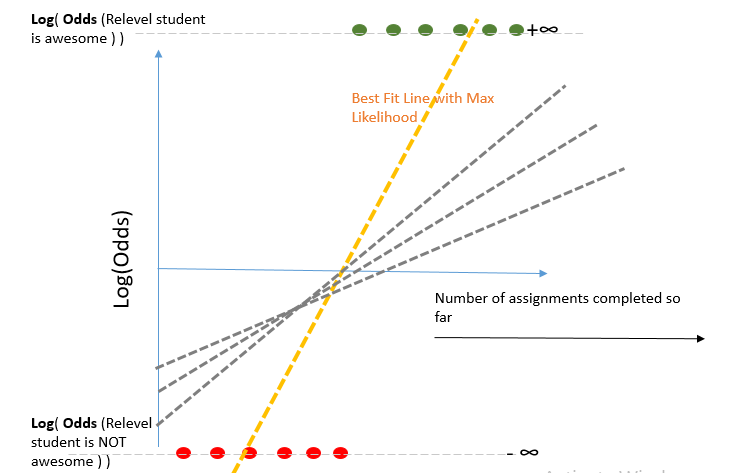
* log(0.95) \* log(0.94) \* log(0.92) \* log(0.9) \* log(0.8) \* log(0.75) \* log(1-0.4) \* log(1-0.35) \* log(1-0.3) \* log(1-0.28) \* log (1-0.24) \* log (1-0.2)
* Log(0.95 + 0.94 + 0.92 + 0.9 + 0.8 + 0.75 + (1-0.4) + (1-0.35) + (1-0.3) + (1-0.28) + (1-0.24) + (1-0.2))
* 3.2464

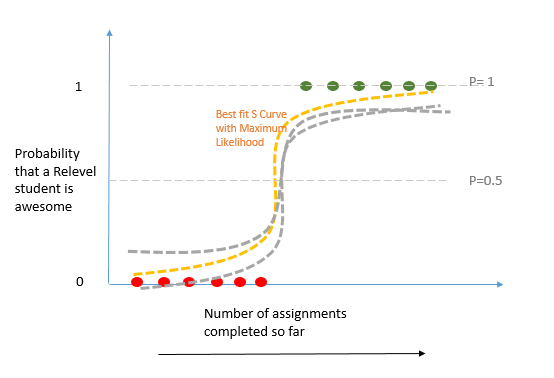
**Step 5 :**

Now the initial straight line with y axis as log(odds) ) will be rotated as per Gradient Descent process discussed earlier. A new straight line will be obtained, again projections would be taken and maximum likelihood would be calculated for the new S Curve obtained.

This iterative process will happen until the algorithm will reach minima on Gradient Descent, and that minima will be the configuration at which we’ll get maximum likelihood.

At that configuration, the model will have its best-fit line on log(odds) axis ; and corresponding S-Curve on original axis with y axis as Probabilities.





As can be seen geometrically in above charts, there will be multiple iterations as per Gradient Descent method, and the curve/line with Maximum Likelihood will be finally selected.

Note : By default the threshold P value is considered to be 0.5. It means that all the data points having P>0.5 will be classified as 1 and otherwise 0. But this can be adjusted as per the requirement to be slightly higher or lower as well.

## Advantages and Disadvantages

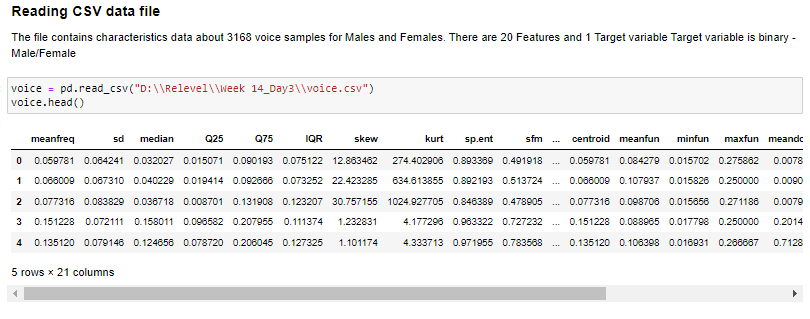
1. **Advantages:**
   1. Logistic Regression is very efficient in predicting target as compared to alternatives.
   2. Logistic regression is less prone to Overfitting unlike Decision trees and KNN ( alternatives)
   3. Logisitc Regression is scalable, ie. It works well with large datasets.
2. **Disadvantages :**
   1. A lot of pre-processing is required for Logistic Regression which requires considerable time and effort. This includes feature scaling, treating missing values and outliers etc.
   2. It assumes that there is a linear relation between a feature and target variable. In real world datasets, that is not usually the case, hence applicability of logistic regression goes for a toss.

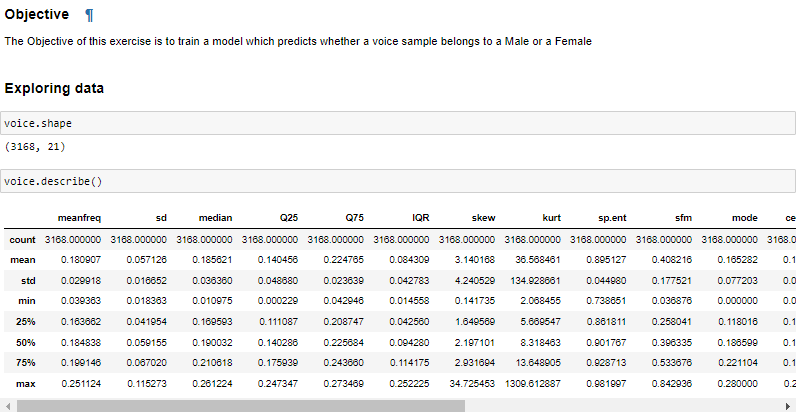
## Logistic Regression implementation in Python

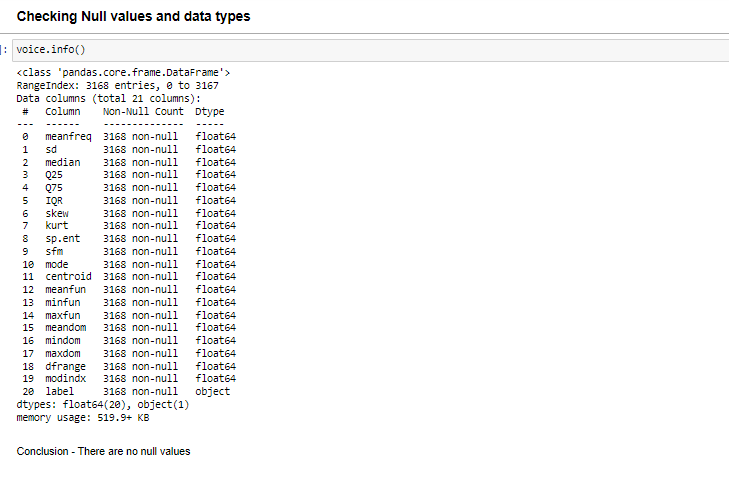
For Python demo of today’s models, we’ll be using Voice sample data of about 3000 males and females. The Target variable would be Male/Female , as the objective is to predict whether a person is Male or Female based on their voice characteristics.

Let’s first load and Process the data before we train our Logistic Regression model as below –

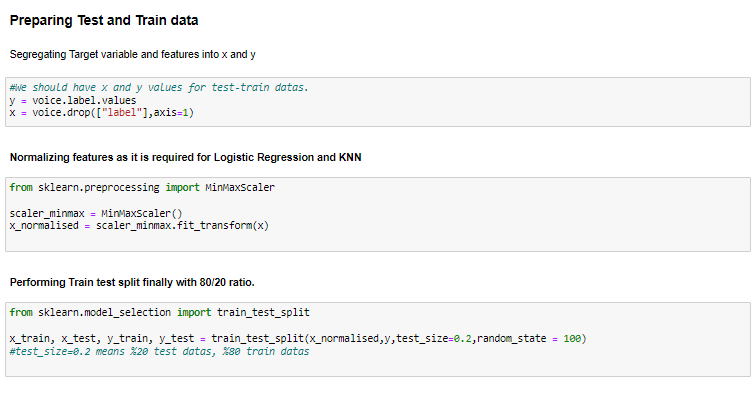
### Loading and Preparing Voice data for Model training –





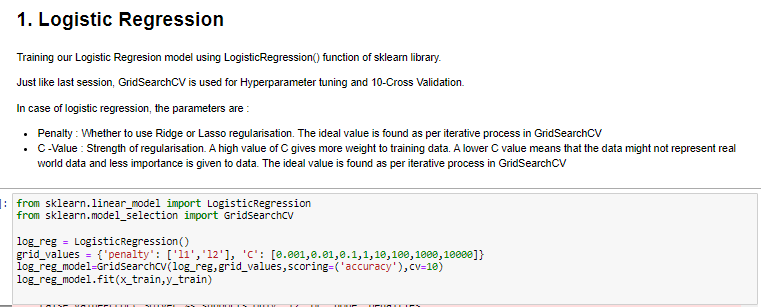


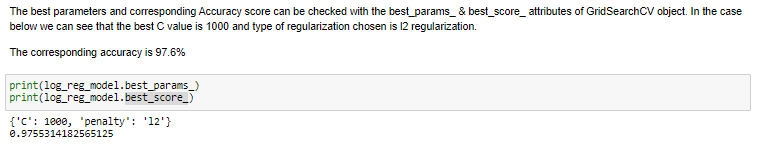


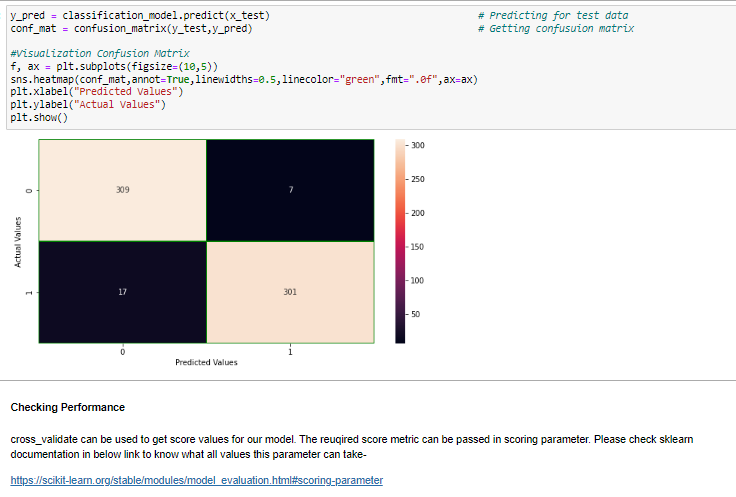


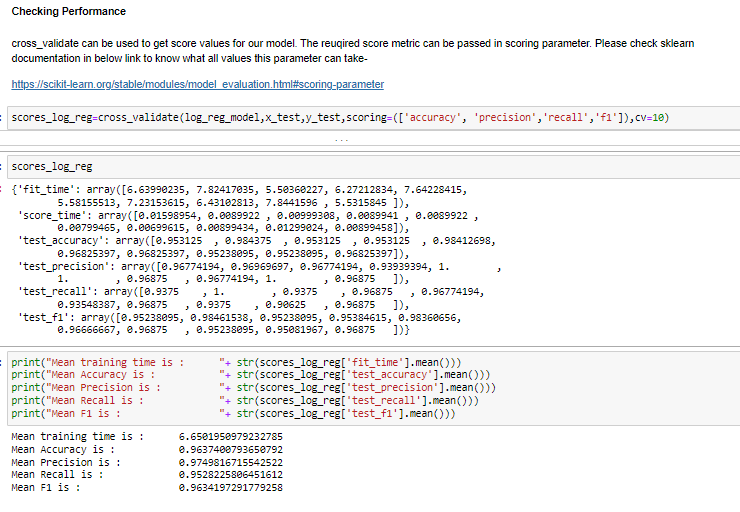
Our data is ready for training now.

### Logistic Regression model training and Performance measurement –





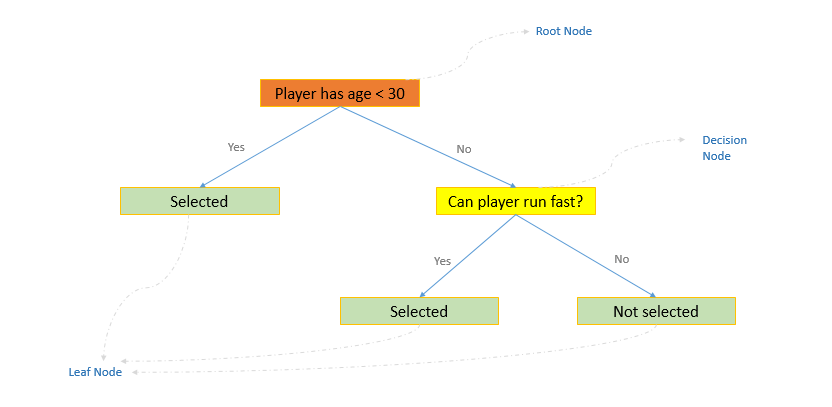




# Decision Trees

Decision trees are a class of algorithms which train a tree-like structure in order to predict target output. It can be used for both classification and regression problems, but in the interest of time, our scope of discussion would be limited to Classification use case only. The mechanism of decision trees is somewhat easier to understand than logistic regression because it can be visualized to a greater extent and an actual final tree can be drawn as we’ll see later.

Let’s look at a simple example of a decision tree below to expand on our understanding –



Context : This is decision tree which predicts whether a player should be selected for a particular team based on some player attributes and characteristics.

It can be observed that a decision tree essentially asks a series of questions to arrive at a prediction.

All of these questions are “Yes/No” questions. Based on the answer to that specific question, the data is split accordingly into Yes/No branches. These questions continue until the tree arrives at a decision. Thus, essentially the training of a decision tree is all about what questions to ask and when.

It can also be seen that blocks are labelled as Root Node, Decision Node or Leaf node in above image. Let’s try to understand these by definition –

is to be learnt that each split results into datasets or “nodes”, on which further splits are performed based on specific questions.

**Root Node :** The root node contains entire training data, and this is where the splitting starts. The first step in any decision tree would be to split the train data into 2 parts based on a specific question pertaining to a particular feature. We’ll talk soon about how that feature and particular question is selected. But to reiterate, root node is where the first split happens on entire data.

In above example, the question “Whether a player is under 30 “ is chosen at the root node.

**Leaf Node :** Leaf nodes are end point nodes in a decision tree which dictate the prediction ultimately. Leaf nodes are not divided further. In above figure, leaf nodes are represented by green color.

**Decision Nodes :** Splitting of root nodes results in subsequent splits of datasets which are further split unless a leaf node is achieved. Thus, all the nodes between top node ( root node) and bottom nodes ( leaf nodes) are called decision nodes. All decision nodes are split further based on specific feature and question.

Before we discuss the detailed step by step algorithm of decision trees, it is necessary to learn 3 concepts – Purity of a split, Gini Index, Entropy and Information Gain. Let’s discuss these one by one –

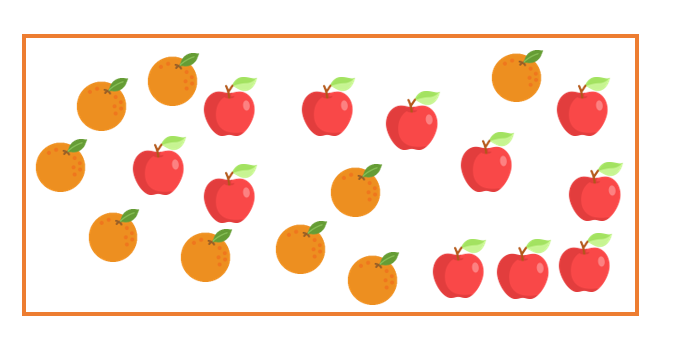
## Pure/Impure datasets

Here, purity or impurity of the data is related to how homogenous the data is in terms of Target variable.

Let’s look at it with an analogy of Oranges vs Apples data. ( Orange and Apple correspond to target variable in our analogy )

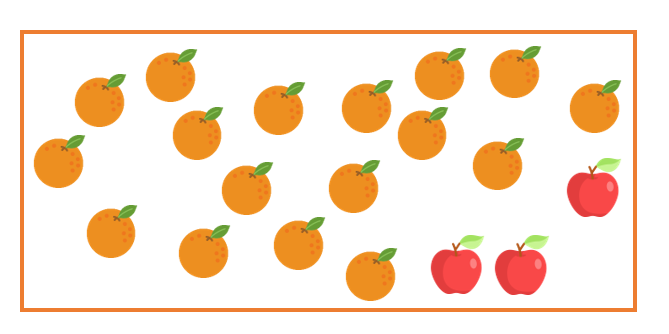
Let’s assume we have a box(dataset in analogy) of 20 fruits. Out of which ~11 are Apple and ~9 are oranges.

This can be visualized as –



In this case, we can see that there are about equal Oranges and Apples in our data. Such cases are called Impure datasets , because it is neither a pure dataset in terms of Apples nor in terms or Oranges. If a random fruit is picked from this box, it can well either be Orange or Apple.

But now, lets look at the below case where we have 17 Apple and 3 Oranges-



As we can see, in this case one can say that box is “Purely” an Orange box because it almost contains Oranges only. It is pure in terms of Oranges.

Remember that Oranges/Apples represent target variable in our analogy.

In decision trees, our goal is to split the data in such a way that resulting data contains “Purest” possible data. Because we would be able to predict a target variable out of a pure dataset only, since there will be almost 100% probability about a particular label of target class. These “Purest” nodes are eventually called “Leaf Nodes” in a decision tree.

In addition to making sure that the splits are “Pure”, it is also important that we reach these Pure nodes earliest as possible. For that, it is important to not only ask the right questions, but also asking the right questions at the right sequence. More on this later, for now let’s move to Gini Index.

## Gini Index

We’ve learnt that splits should be as Pure as possible in decision trees algorithm, but how do we measure that Purity ? Gini Index is the answer to this question.

Gini Index is a measure of Purity at a node. It sits at the core of Decision trees.

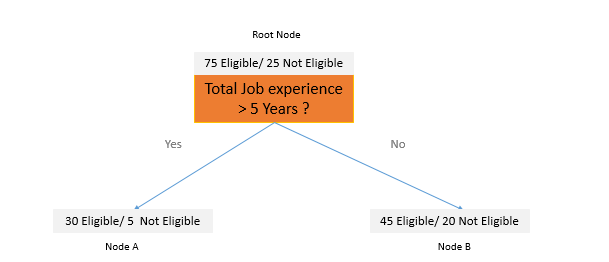
The mathematical formula for calculating GI is :

Here,

In case of multiclass problem, the formula will change accordingly by including P for all classes. The above formula represents GI for Binary classification case.

The value of GI ranges between 0 to 0.5. The closer GI is to 0 , purer a node is. Hence, model will always look for splits which results in lowest possible GI values.

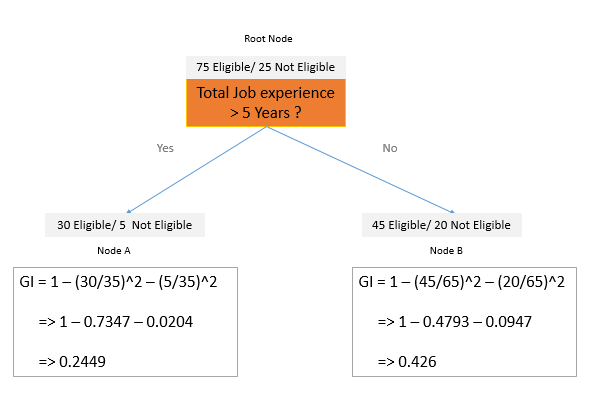
Let’s try to understand GI in action with an example, where we are splitting a data of 100 persons to determine whether they are eligible for a job application or not –



Here, based on first split on the root node, we get 2 nodes A and B. Initially in the Root node, the +ve/-ve Class split was 75/25.

After the split, the node A has class split as 30/5 and in node B it is 45/20.

Using the GI formula ; we can calculate the GI for Node A and Node B to be 0.2449 and 0.426 respectively as shown below :



This means that Node A is purer than Node B, since the GI is lower.

Also, please note that the GI is calculated for a particular node. While it does tell us about how Pure/Impure a node is, we still need some better measure to ascertain what kind of split would result in overall lowest GI for all split nodes. This is where Information Gain, which uses GI to ascertain which is the best possible split possible for best overall purity after the split.

## Information Gain :

Information Gain calculates how much Impurity is reduced ( or Purity is gained ) after a particular split. The higher the Information gain, the better a split is.

In decision trees, a split can be made in multiple ways at each point. If the training data has 10 features, then there are 10 features to chose from based on which the node will be split into further branches, even in a selected feature, there would be multiple possibilities to chose on how the split should be done based on that feature. As an analogy, all of this pertains to “selecting the right question to ask”. We will discuss this in detail later in this session. This process is briefly introduced here now to stress on to the fact that Information Gain is the metric which decides how a split should be performed. Out of multiple options, the option having highest Information gain would be chosen.

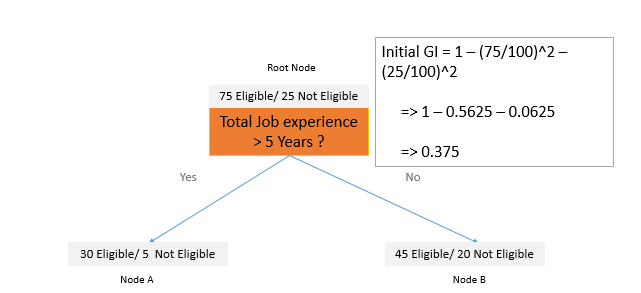
The formula for Information Gain is given as :

Information Gain = Initial Impurity of the Node before the Split – Weighted Average Impurity after the split

i.e, in case we are using GI for Information gain , formula becomes –

**Information Gain = GI before split – Weighted GI after the split**

Let’s understand this through the same examplea we discussed in GI :



In that case, the data before split had 75/25 ration between +ve/-ve class.

Thus, as per the calculation shown in above image, the GI was 0.375

We have already calculated the GIs for Node A and Node B.

The weighted Gini can be calculated as –

**+**

From above figure, we can see :

Node A datapoints = 35

Node B datapoints = 65

Node A + Node B datapoints = 35+65 => 100

GI Node A = 0.2449

GI Node B = 0.426

Substituting all of these in the formula above :

Weighted GI = +

* 0.0857+ 0.2769
* 0.3626

So now we have Initial GI = 0.375 , and also weighted GI after the split = 0.3626.

Substituting these in Information gain formula ;

Information Gain = 0.375 -0.3626

* 0.0124

This is how Information gain is calculated for a split.

In a decision tree, this Information gain is calculated iteratively for multiple combinations , and the combination with maximum Information gain is chosen as split condition. This process is repeated at each node until we reach Leaf node.

## Entropy

Just like GI, Entropy is also used to calculate Impurity of a node. It is an alternative to GI, and can be used in Information gain formula instead of GI.

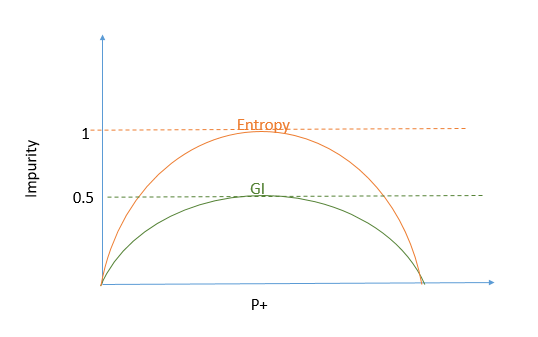
There is just formula difference between Entropy and GI, but the application and goal remains same.

The formula for Entropy is given as :

where,

One crucial difference between GI and Entropy is the range. While GI has range {0-0.5}, the range for Entropy is {0-1} instead. Similar to GI, the lower entropy value represents a better split and purer node.

Both Entropy and GI reaches peak value at , but the peak value differs for both by 0.5 as shown in the graph below –



Generally, by a lot of tree based algorithms, GI is preferred because it is efficient computationally. It doesn’t contain log term like Entropy and also has a lower range, which makes it more efficient.

## Decision Tree Algorithm :

Now that we have learnt about Gini Index, Information Gain and Entropy, we can finally discuss what are the steps involved in training a decision tree. In fact, we have already covered most of the procedure by now while learning about these concepts, and now it is just a matter of stitching it all together one last time for a clear picture on the process.

In order to see the practical implementation of decision tree, let’s take following data as example –

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature 1** | **Feature 2** | **Feature 3** | **Target Variable** |
| **Does Player Smoke?** | **Can player run 100 meters under 20 seconds ?** | **Goals Scored in last 5 matches** | **Eligible (1)/ Non Eligible ( 0)** |
| Yes | No | 4 | 1 |
| No | Yes | 3 | 1 |
| Yes | Yes | 7 | 0 |
| No | Ye | 2 | 0 |
| ….and so on ( Total 1000 Rows ) | | | |

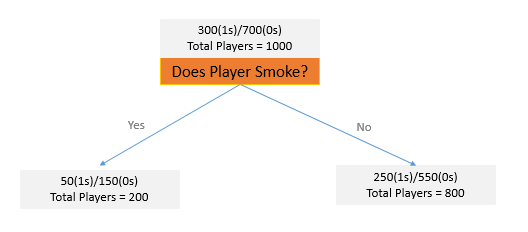
So we can see that we have data about 3 features of 1000 players, and a Target variable.

Now let’s start making a decision tree step by step :

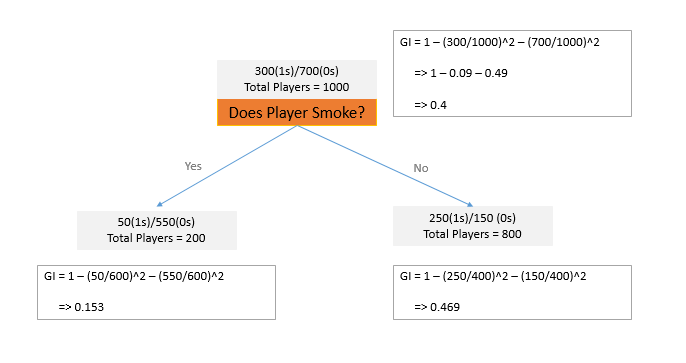
**Step 1: Choosing Feature to split the root node**

For the root node, calculate the Information Gain for each feature one by one iteratively –

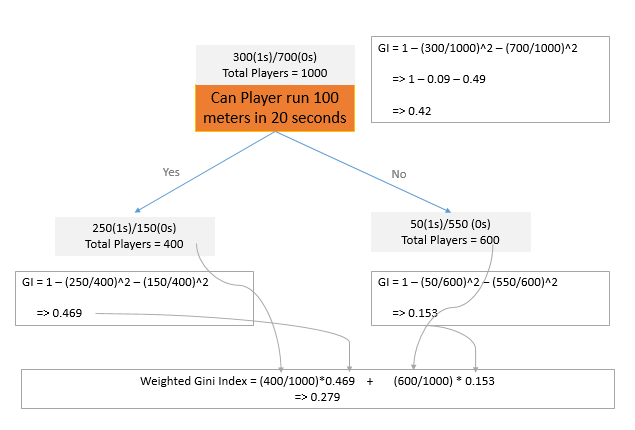
Case 1 : Choosing Feature 1 for split divides the root node as follows -



Lets’ calculate GIs for each individual node –

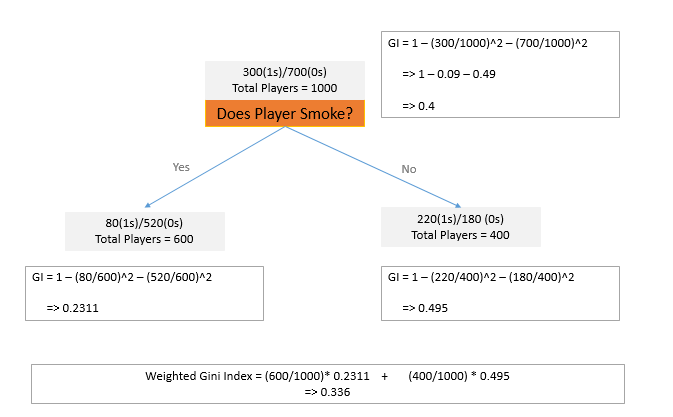


Now Weighted Gini Index can be calculated as –



Thus, **Information gain for Feature 1 would be =** 0.42-0.279 => **0.14**

Now we will repeat this process for Feature 2( Can player run 100 meters under 20 seconds ) :



As per above calculations , Information gain for this Feature 2 would be –

* 0.42-0.33
* 0.083

Similarly we will calculate Information Gain for our feature 3 as well. Let’s skip calculation and assume that Information Gain for our Feature 3 comes out to be 0.09

( Calculation of Gini Index and Entropy Gain is a bit different for continuous variables, and we shall look into that separately just in a few minutes )

Finally, we have our Information gain for all the 3 features as below –

|  |  |
| --- | --- |
| **Feature** | **Information Gain** |
| **Feature 1** | **0.14** |
| Feature 2 | 0.083 |
| Feature 3 | 0.09 |

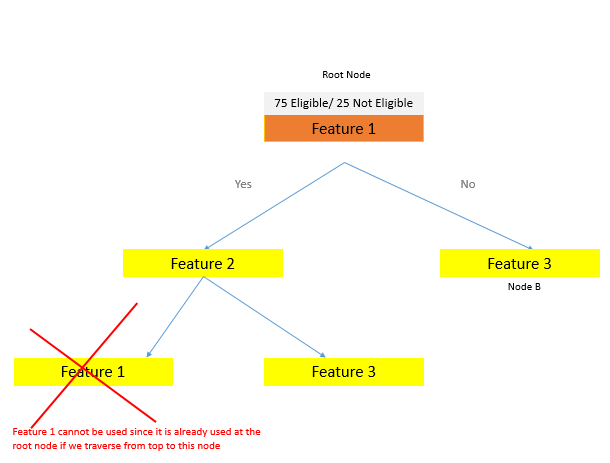
Since Information Gain is maximum for Feature 1, Feature 1 will be selected for splitting the root node. This concludes our step 1.

**Step 2 :**

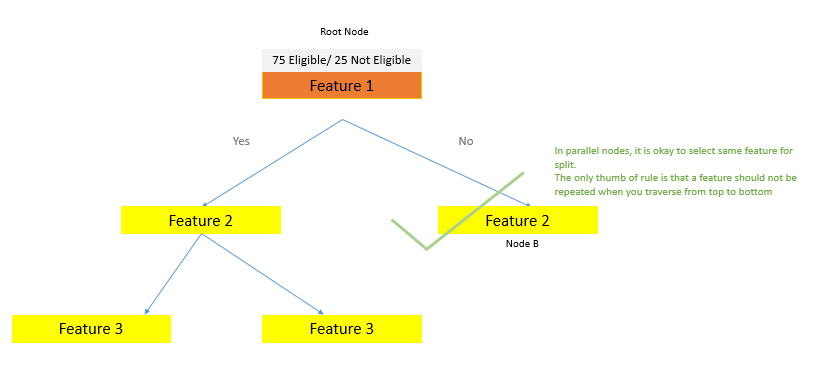
Once the root node is branched into child nodes, the same process will be repeated for each node on both sides. Once those child nodes are split, further splits will keep on happening with the help of same process until the tree reaches leaf node.

One important point to consider is that same feature can not be used again in the same branch while we move in the tree from top to bottom. But, at the same time it is okay and possible for the same feature to be used for splits in a parallel branch.

For example, below is an invalid case :



But it is okay to use Feature 2 in the right side of tree even though it is used at node A as below :



**Step 3 : Identifying a node as leaf node and end the tree**

This is the last step in training a decision tree and it’s a simple rule. There are multiple parameters which control the decision of reaching a leaf node as following –

* **Maximum Depth :** The maximum depth of the tree can be specified , and once any branch in the tree reaches that particular depth, all the nodes are marked as Leaf nodes. The prediction value would be the majority target value of the node.

A high maximum depth setting might cause overfitting ; which means it can give good results for train data but not so much for new test data.

A low maximum depth setting might cause under fitting, meaning the tree is not trained enough to predict accurately in any case. Thus, one has to find the right balance for best results.

* **Threshold minimum count in a node :** It dictates that if a node gets too small as specified by the threshold value the node can be made a leaf node and tree can be stopped there.

Again, a value too low might cause overfitting and too high will cause under fitting.

* **0 Information gain** – This is a default rule. If the Information gain for a split is 0 , then that split is not done and the node is marked as a Leaf node. In other words, there must be a +ve information gain for a split to happen.

The 0 Information gain practically means that the overall Purity of the node has no change before the split versus after the split. Hence it is better to not do the split instead.

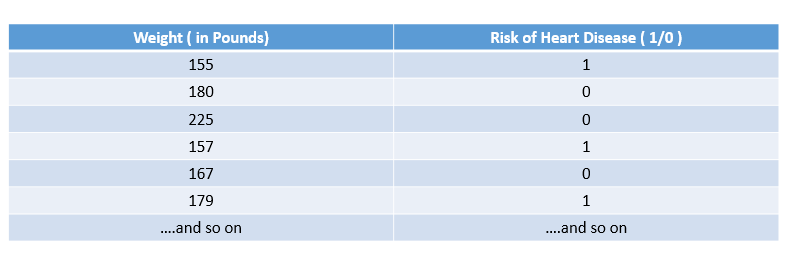
Tuning these parameters to their most optimum value is called Hyper-parameter tuning.

This concludes our Decision Tree algorithm.

Now as discussed earlier, as an additional discussion let’s also look at how Gini Index is calculated for Continuous features:

## Gini Index and Information Gain for Continuous features –

Let’s consider a continuous feature Age, and target variable to be Risk of diabetes ( 1/0) :

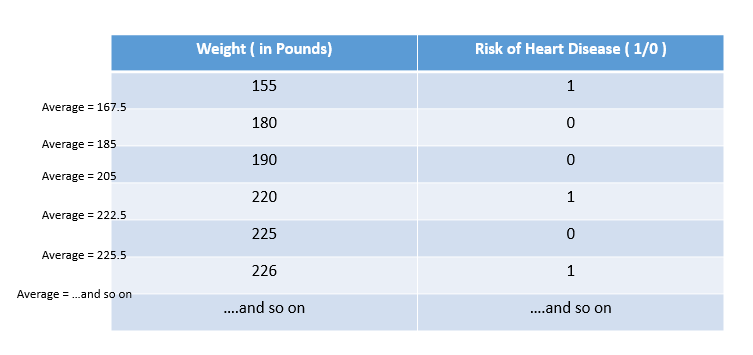


**Step 1 : Sort the continuous feature in ascending order**

After sorting, the feature will look like below –

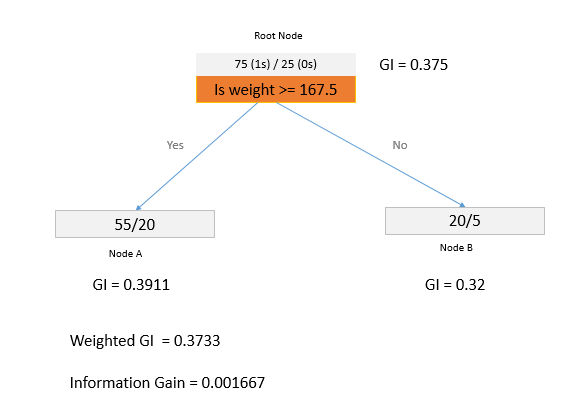


**Step 2: Calculate the averages of adjacent points in sorted feature :**



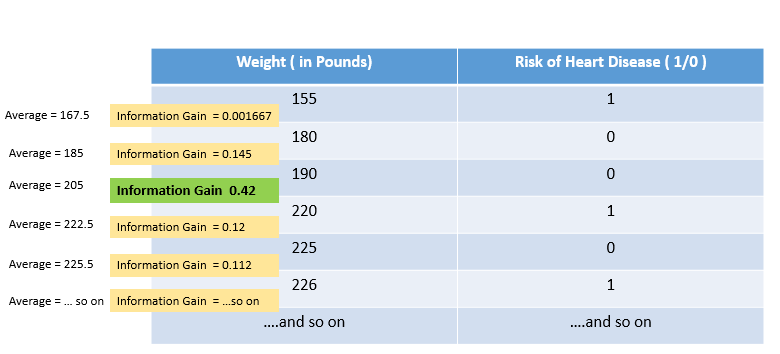
**Step 3 : Calculate Gini Indices and Information gain iteratively based on the splits considering these averages as threshold split condition**

For example, in iteration 1 the split condition would be – “ Is Weight >= 167.5 ?“



As per above calculation, the Information gain for this case came out to be 0.001667

This step will be repeated for all the adjacent averages and Information gain will be calculated as below:



**Step 4 : Select the Average value having Maximum Information gain as the final split condition for that feature**

In above example, Average >= 205 can be chosen as the split condition, assuming that for all other values, the information gain was lower than 0.42

This is how GI and Information gain is calculated for continuous variables. The next steps regarding choosing the best feature for split and identifying the leaf nodes remain same as we have already discussed in last section.

It is to be observed that this process for a continuous feature is too long and would take a lot of computational power. Assume a case where data would have millions of records, the model would need to calculate Information gain a million times just for 1 feature.

For this reason, decision trees’ performance gets really impacted if the data contains continuous features, specially in case of large datasets.

There are some other Feature types as well, for which the process of calculating GI and Information gain is slightly different. These are-

* Rank based features 9 example – (Example : Rank – 1/2/3/4)
* Multi-class categorical features – ( Example : Color – Red/Blue/Green )

( The categorical features discussed so far were only binary in nature having Yes/No values)

In the interest of time, students are advised to explore on these cases on their own. Our understanding so far is sufficient for us to proceed with training decision trees for now.

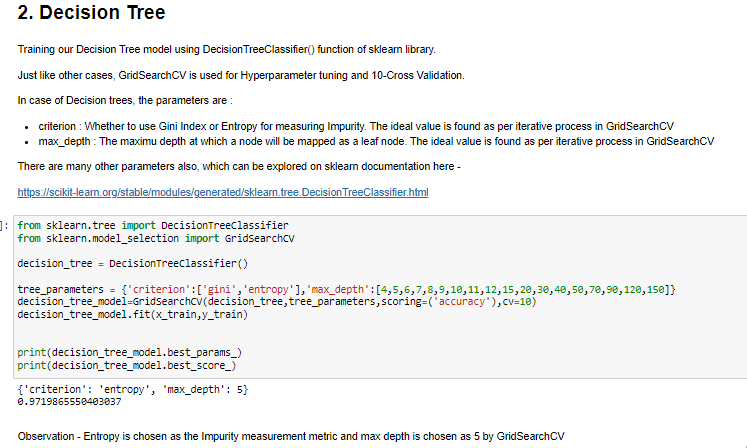
Finally, let’s summarize our learnings with respect to decision trees -

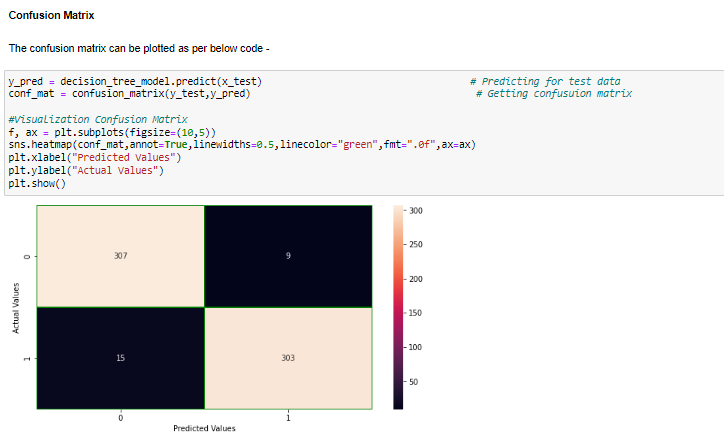
* Decision trees are Classification/Regression algorithm ( but used mainly for Classification)
* These are called Decision Trees because data is split in a tree-like manner based on specific decisions pertaining to specific questions
* Decision trees are all about identifying what best question to ask and when, so that model can split the data in most efficient manner to reach the leaf node quickly
* Decision trees can have both continuous and categorical features
* Gini Index and Information Gain are used to determine which split is best among many possible options.
* Leaf node is reached based on user specified parameters.
* Leaf node predicts output as the majority target in the node itself.

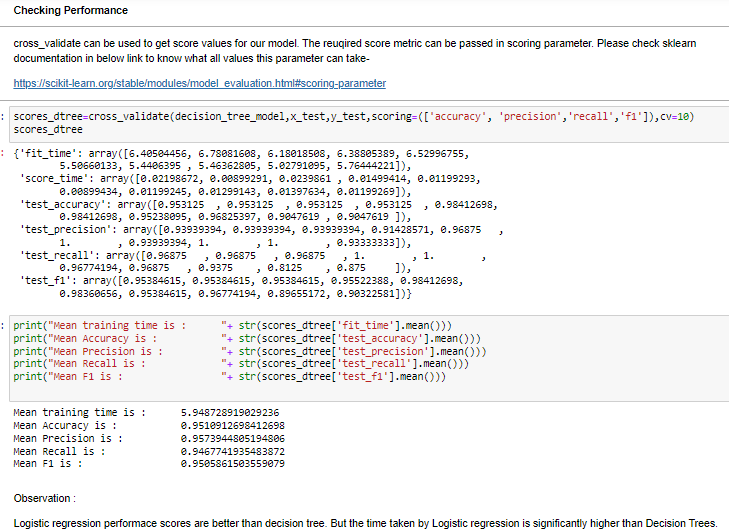
## Advantages and Drawbacks :

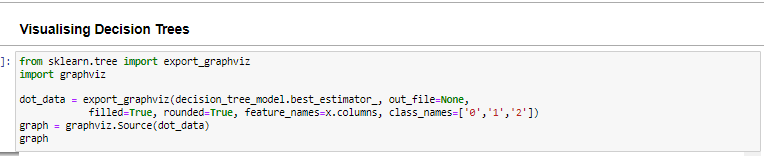
1. **Advantages :**
   1. Feature Scaling is not required. Decision tree works well with unscaled data.
   2. Missing value don’t impact the performance to a great extent
   3. From point 1 and 2, it can be concluded that Decision trees require less pre-processing effort on the data.
   4. A decision tree is easy to understand and implement. The visual tree is a great add on to explain the model to a business stakeholder as it is easy to interpret.
2. **Disadvantages:**
   1. Decision Trees are sensitive to the training data. A small change in training data might make the model change a lot.
   2. Requires more time to train the model, specially when a lot of continuous variables are involved.
   3. Although it can do regression prediction as well, but it is very inefficient for same.

## Python Implementation of Decision Trees

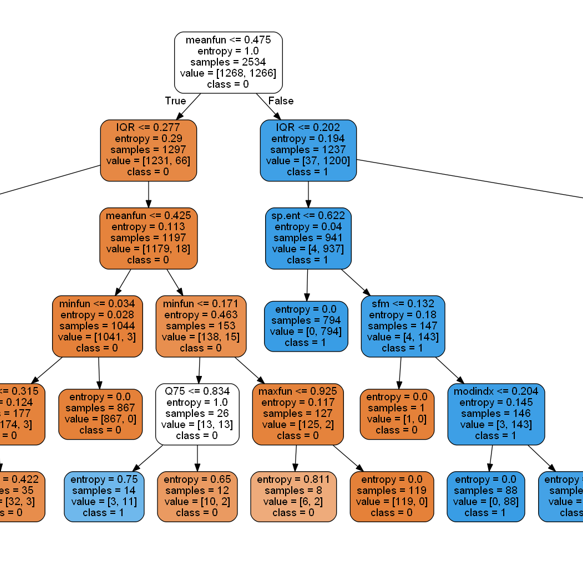








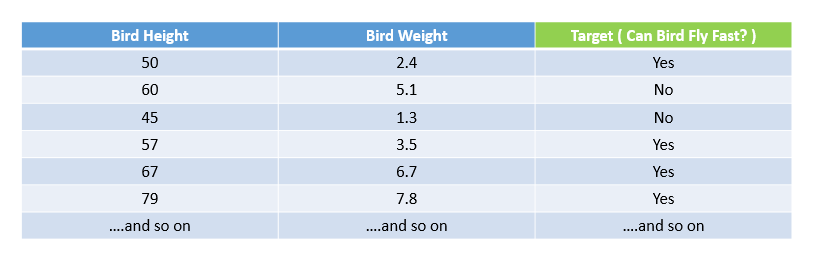
Since the image created is too large to fit here, a zoomed part of Decision tree is shown below. The entire image is provided as an attachment separately.



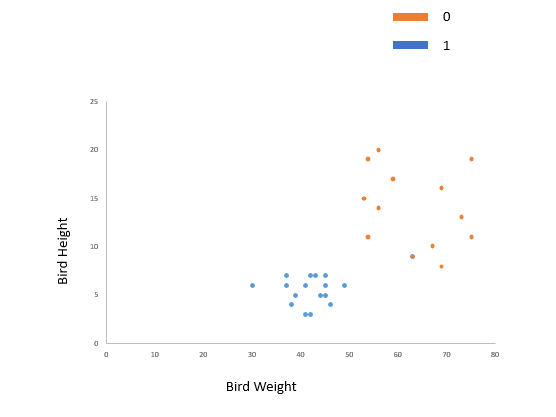
# 4. K- Nearest Neighbours ( KNN Algorithm )

K Nearest Neighbours is an extremely simple algorithm which can be used for both classification and regression use cases. Again, in the interest of time, our scope of discussion would remain limited to the classification use case only. The regression part is very similar and students can explore on their own.

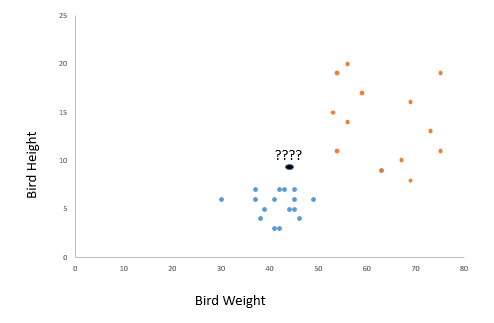
Let’s take a sample dataset in order to understand the underlying steps in KNN. Our sample dataset has 2 features – “Bird Height” and “Bird Weight” which predict whether bird can fly fast or not.



This can be plotted in a scatter plot as –



The goal of the model would be to classify a new point as either 0 or 1 here ( Orange or Blue color )



Following is the step by step process by which KNN will predict the class for this new point –

**1. Step 1 : Select a value of K**

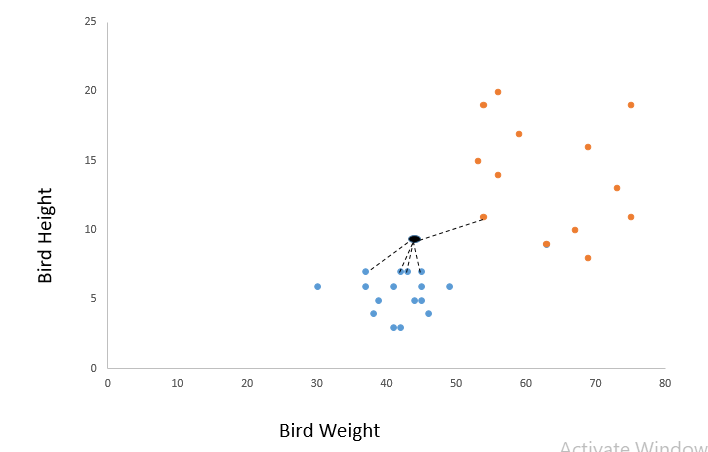
Value of K represents how much neighbouring points should be considered for prediction. Higher values will lead to overfitting and lower values would lead to under fitting.

There is no mathematical way or rule to ascertain the ideal K value. It is found out using iteratively training the model for multiple values of K and then selecting an appropriate value which is lower enough not to cause overfitting and provides good accuracy as well. We’ll soon see how it is done in Python.

For now let’s assume the value of K=5 to continue with the further steps.

**2. Step 2 : Identify K Nearest Neighbours:**

Calculate the distance of new point from all the points in the data and identify nearest K points. It must be evident from this step that why this algorithm is called KNN.

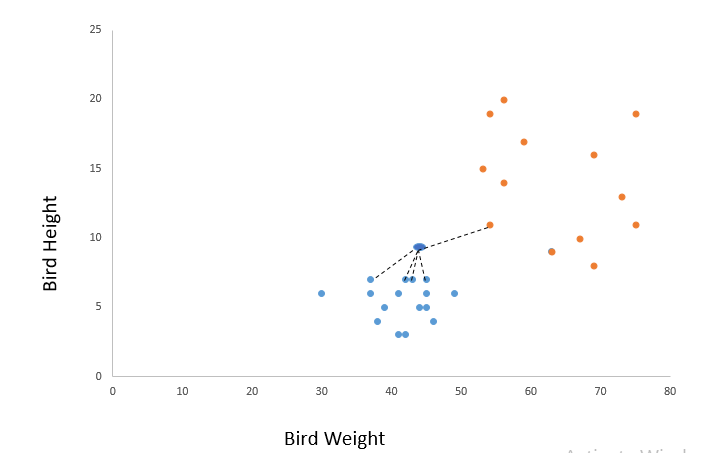


There are multiple distance metrics available to choose from, for example Euclidean distance and Manhattan distance to name a few. Euclidean distance is the most common distance metric. We’ll discuss about these metrics as well soon. For now let’s proceed with Euclidean distance.

**3 Step 3 : Make the prediction**

That’s about it for KNN algorithm. The model can make prediction as the “mode” value of corresponding target values for K nearest neighbours.

From the last image, it is evident that out of 5 nearest neighbours, 4 are blue ( +ve class), hence this new datapoint would also be classified as +Ve class as shown below –



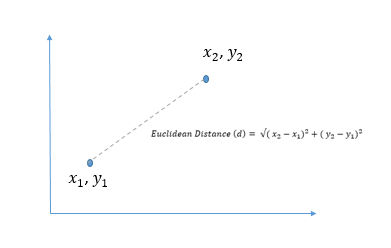
This concludes the KNN algorithm steps.

For regression problems, everything would remain same except that the output prediction would be mean or median of nearest neighbours instead of mode.

Since calculation of distances is at the core of KNN, lets also discuss 2 of the most used distance metrics :

1. **Euclidean Distance**

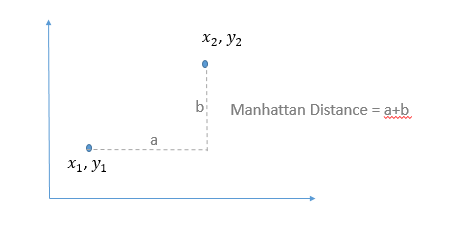
For 2 points (x1,y1) and (x2,y2) , the Euclidean distance is given as –



1. **Manhattan Distance :**

Manhattan distance is sum of the absolute vertical and horizontal distances.

Manhattan Distance =



Finally, let’s discuss some of the drawbacks and advantages of KNN.

## Drawbacks and Advantages of KNN algorithm

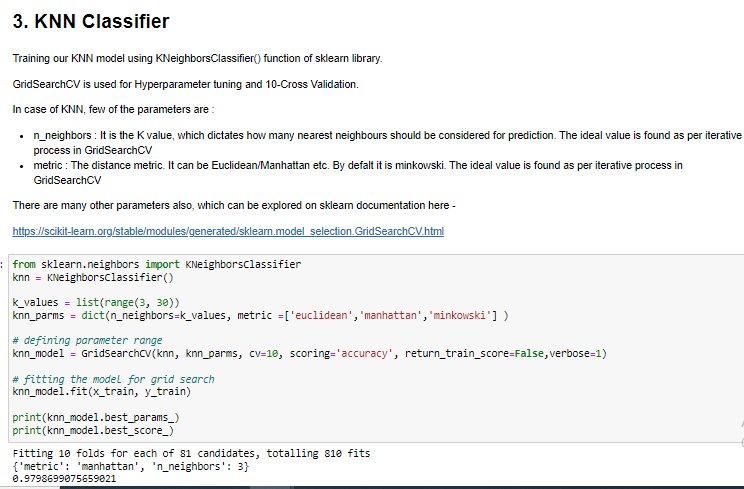
**Drawbacks :**

* Since it is distance based algorithm, KNN is highly sensitive to Outliers and Imbalanced datasets. Hence, both issues must be treated well before using KNN
* Different values of K would result in different results, and there is no standard way to find the best K value.
* As the data increase, KNN becomes computationally inefficient. This is not a good algorithm for really large datasets.

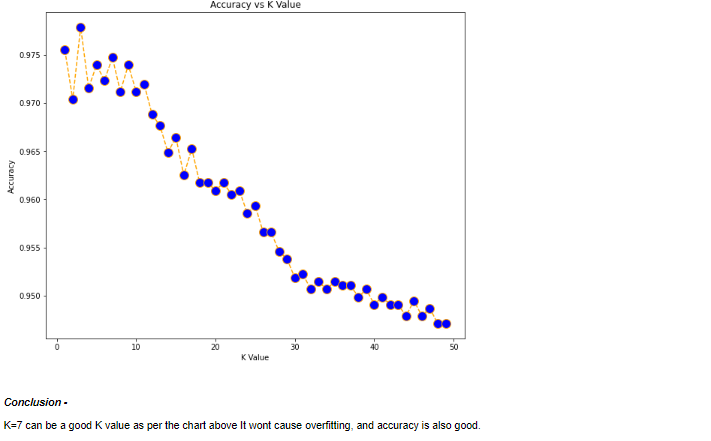
**Advantages :**

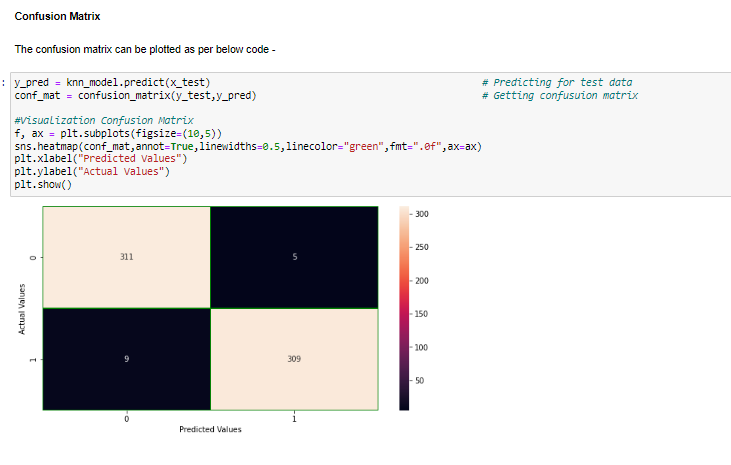
* Flexibility of choosing from a range of distance metrics
* Simple and easy to understand and implement.
* For smaller datasets, KNN is quicker than alternatives.

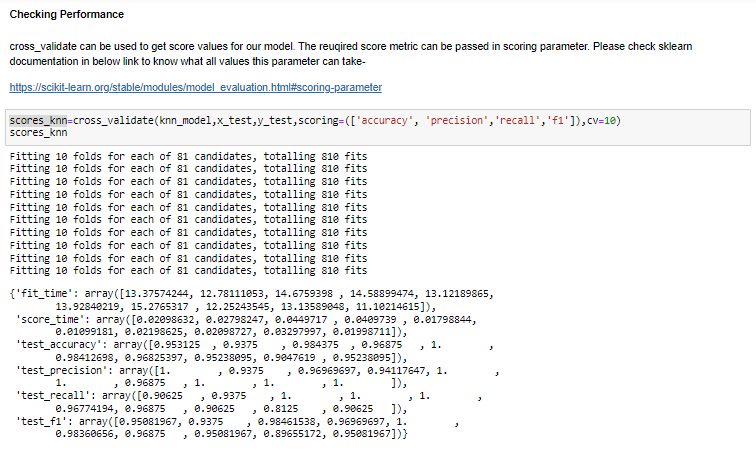
## Python Implementation –

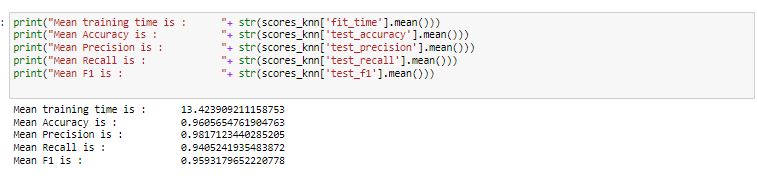




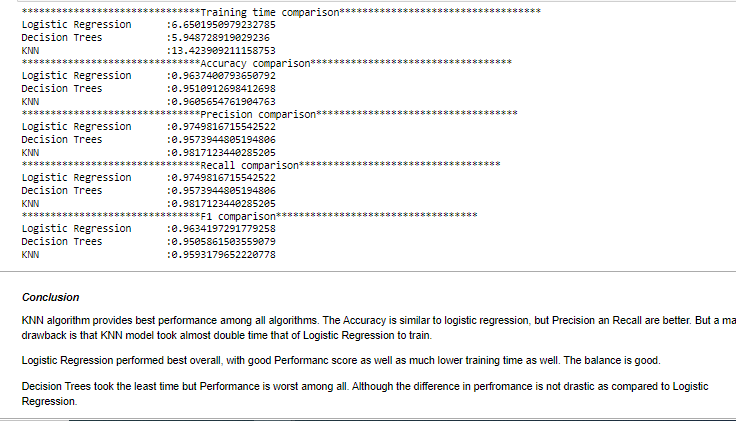












# 5. MCQ

Q1. Which of the following is not True about KNN :

1. For smaller values of K, model may be overfitted
2. For larger values of K, points too far away can be included in neighbourhood
3. KNN can be used only for classification algorithms
4. KNN requires feature scaling

Q2. Which of the below is true about Decision trees :

1. Decision trees can only be used for classification problems
2. Features have to be categorical for training a decision tree algorithm
3. Decision tree stops splitting the nodes when Information gain is negative
4. Same feature can not be used multiple times in the same branch for splitting

Q3. Which is false about Information gain :

1. Information gain decides which feature is selected for splitting a node
2. Information gain can use either Gini Index or Entropy
3. Information gain ranges from -1 to 1
4. Information gain must be +ve for a split, otherwise the node is classified as Leaf node

Q4. Which of the following controls only Type-2 error :

1. Accuracy
2. Precision
3. Recall
4. F1 Score

Q5. Why log(odds) is used instead of odds?

1. It is computationally better than odds
2. It transforms odds to a symmetrical entity
3. It has less error
4. Log is preferred by statisticians always

Q6. A dataset with 1s/0s ratio as 600:500 is purer than a dataset having ratio of 1s/0s as 900:100

1. True
2. False

Q7. Which function(s) is/are responsible for transforming S-Curve in logistic regression to a straight line and vice versa ?

1. Logit Function
2. Sigmoid Function
3. Both of these, as these are 2 are inverse functions of each other
4. Modulus function

Q8. How Information gain is calculated for a continuous numerical variable ?

1. Weighted Gini Indices are calculated for each of the value, and divided from the original Gini Index before split
2. Weighted Gini Indices are calculated for averages of the adjacent sorted values, and divided from the original Gini Index before split
3. Weighted Gini Indices are calculated for averages of the adjacent values, and divided from the original Gini Index before split
4. Information gain cannot be calculated for such features

Q9. Which of the following are used to identify the best fit S Curve in Logistic Regression?

1. Ordinary Least Squared
2. Maximum Likelihood
3. Entropy Gain
4. All of above

Q10. Performance of Logistic Regression is usually measured in terms of R2 Adjusted-

1. True
2. False

# 6. Homework

* Explore additional performance metrics for binary classification called ROC and AUC.

Measure ROC AUC as an additional scoring metric for the case discussed in class and compare it among models.

* Learn how Information Gain can be calculated for Ranked features and Multi Class features
* Explore how Logistic Regression can be used to predict Multi Class prediction problems.