Module : 5

Supervised Learning-Linear regression

Regression

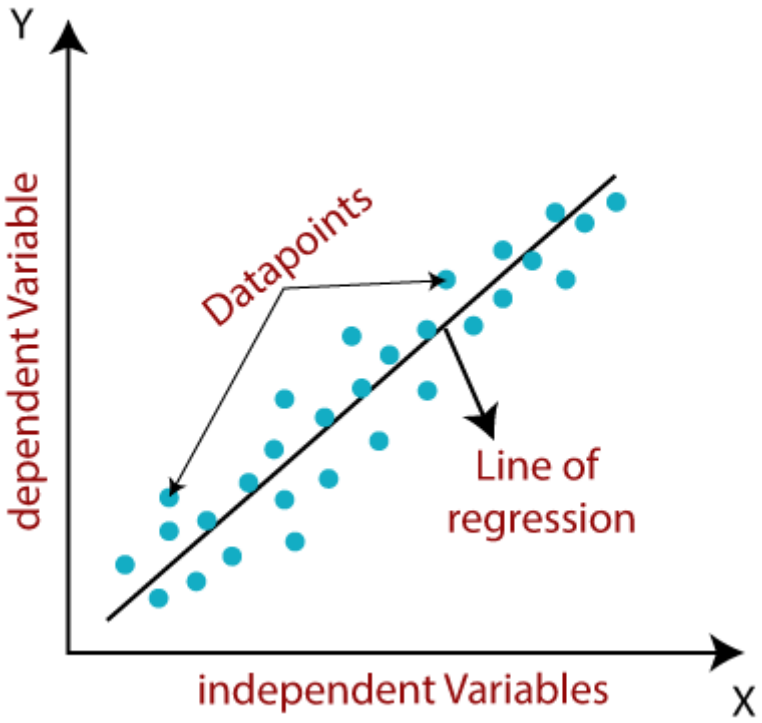
The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as:

$$y= a_0+a_1x+ \varepsilon$$

Here,

Y= Dependent Variable (Target Variable)
X= Independent Variable (predictor Variable)
a0= intercept of the line (Gives an additional degree of freedom)

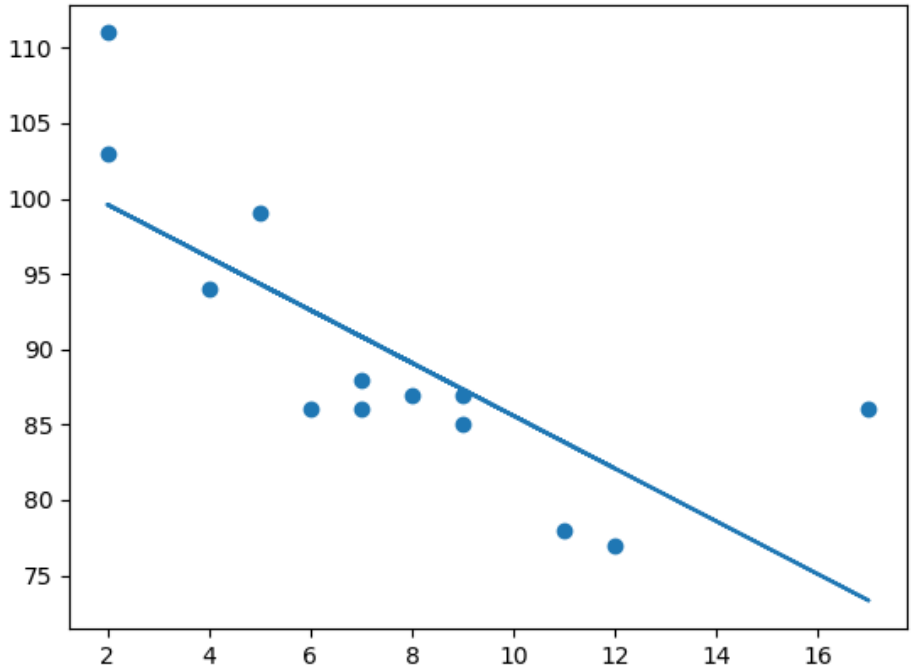
a1 = Linear regression coefficient (scale factor to each input value).
ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.



In Machine Learning, predicting the future is very important.

Types of Linear Regression

Linear regression can be further divided into two types of the algorithm:

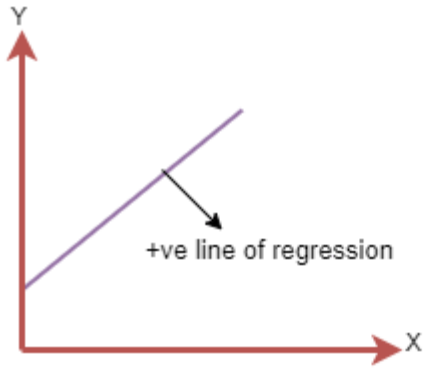
- **Simple Linear Regression:**
  If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.

- **Multiple Linear regression:**
  If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

Linear Regression Line

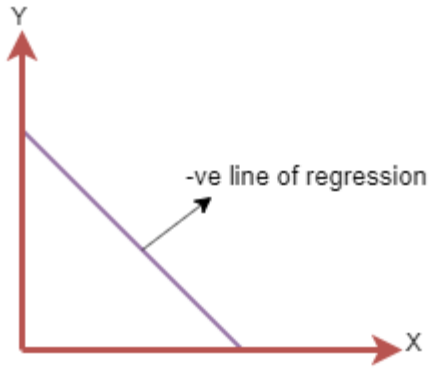
A linear line showing the relationship between the dependent and independent variables is called a **regression line**. A regression line can show two types of relationship:

- **Positive Linear Relationship:**
  If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

- **Negative Linear Relationship:**
  If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

Finding the best fit line:

When working with linear regression, our main goal is to find the best fit line that means the error between predicted values and actual values should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a_0, a_1) gives a different line of regression, so we need to calculate the best values for a_0 and a_1 to find the best fit line, so to calculate this we use cost function.

Cost function-

- The different values for weights or coefficient of lines (a_0, a_1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the **mapping function**, which maps the input variable to the output variable. This mapping function is also known as **Hypothesis function**.

For Linear Regression, we use the **Mean Squared Error (MSE)** cost function, which is the average of squared error occurred between the predicted values and actual values. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where,

N=Total number of observation
Yi = Actual value
$(a1x_i+a_0)$= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Model Performance:

The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called **optimization**. It can be achieved by below method:

1. R-squared method:

- R-squared is a statistical method that determines the goodness of fit.
- It measures the strength of the relationship between the dependent and independent variables on a scale of 0-100%.
- The high value of R-square determines the less difference between the predicted values and actual values and hence represents a good model.
- It is also called a **coefficient of determination,** or **coefficient of multiple determination** for multiple regression.
- It can be calculated from the below formula:

$$\text{R-squared} = \frac{\text{Explained variation}}{\text{Total Variation}}$$

Assumptions of Linear Regression

Below are some important assumptions of Linear Regression. These are some formal checks while building a Linear Regression model, which ensures to get the best possible result from the given dataset.

- **Linear relationship between the features and target:**
  Linear regression assumes the linear relationship between the dependent and independent variables.
- **Small or no multicollinearity between the features:**
  Multicollinearity means high-correlation between the independent variables. Due to multicollinearity, it may difficult to find the true relationship between the predictors and target variables. Or we can say, it is difficult to determine which predictor variable is affecting the target variable and which is not. So, the model assumes either little or no multicollinearity between the features or independent variables.
- **Homoscedasticity Assumption:**
  Homoscedasticity is a situation when the error term is the same for all the values of independent variables. With homoscedasticity, there should be no clear pattern distribution of data in the scatter plot.
- **Normal distribution of error terms:**
  Linear regression assumes that the error term should follow the normal distribution pattern. If error terms are not normally distributed, then

confidence intervals will become either too wide or too narrow, which may cause difficulties in finding coefficients.
It can be checked using the **q-q plot**. If the plot shows a straight line without any deviation, which means the error is normally distributed.

- **No autocorrelations:**
  The linear regression model assumes no autocorrelation in error terms. If there will be any correlation in the error term, then it will drastically reduce the accuracy of the model. Autocorrelation usually occurs if there is a dependency between residual errors.[Autocorrelation refers to **the degree of correlation of the same variables between two successive time intervals**.]

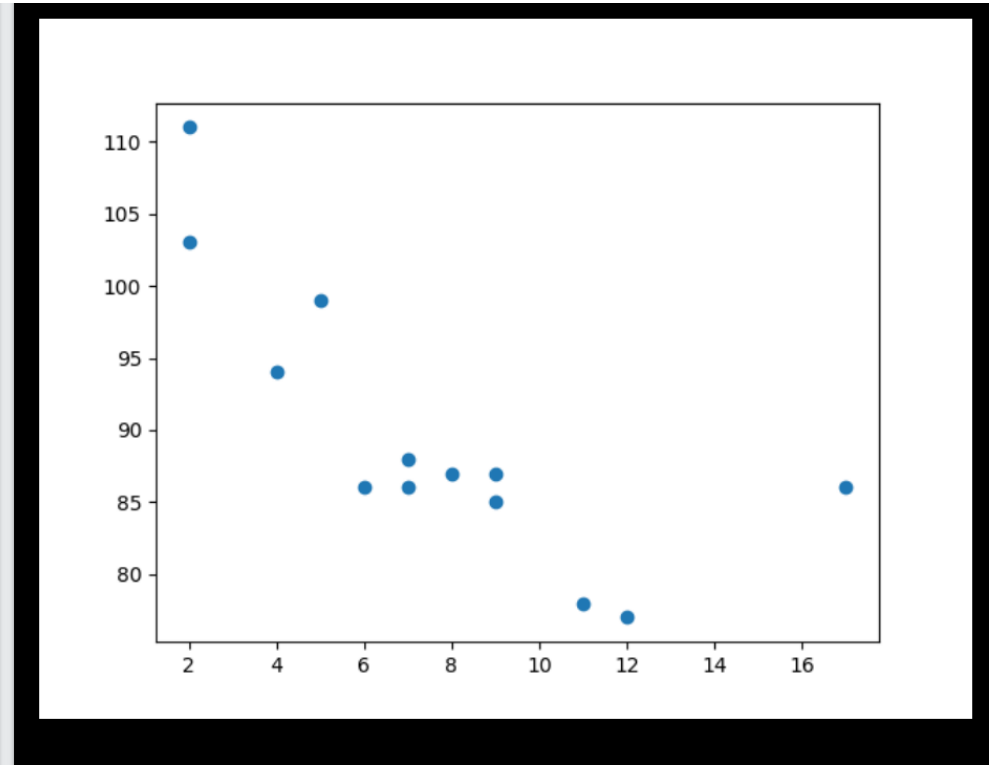
How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.

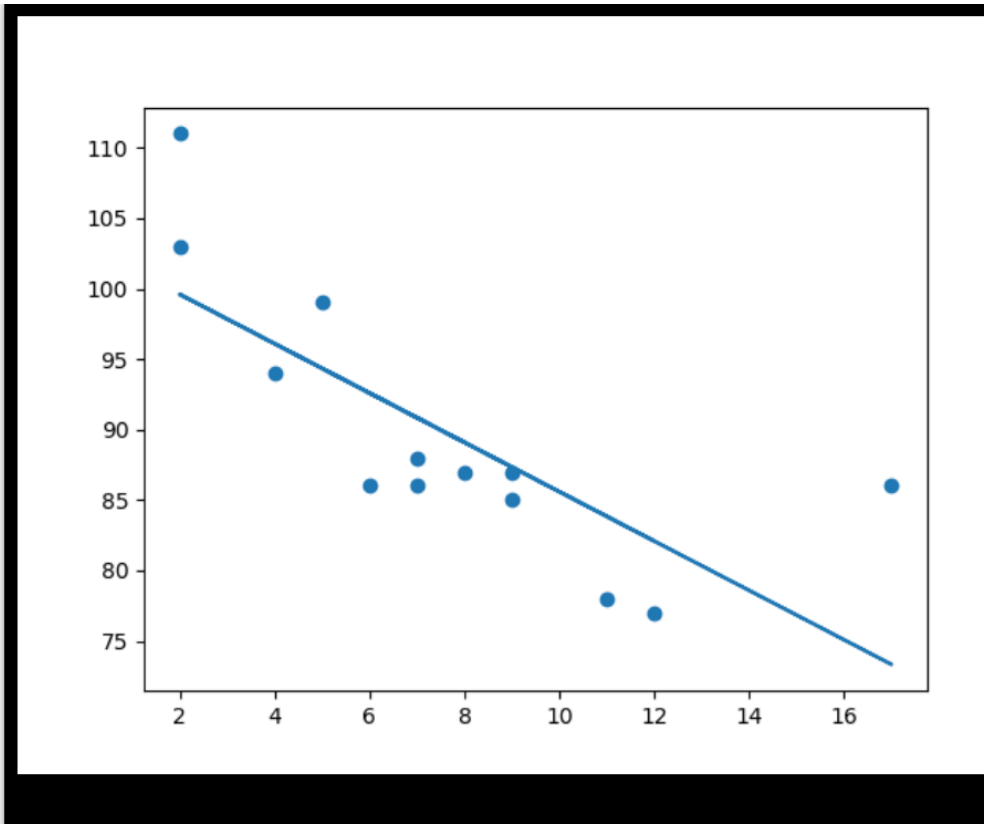
In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```



```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```




```
import matplotlib.pyplot as plt
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Create a function that uses the `slope` and `intercept` values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
  return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list(map(myfunc, x))
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```

R for Relationship

It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called `r`.

The `r` value ranges from -1 to 1, where 0 means no relationship, and 1 (and -1) means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

```
-0.758591524376155
```

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a 10 years old car.

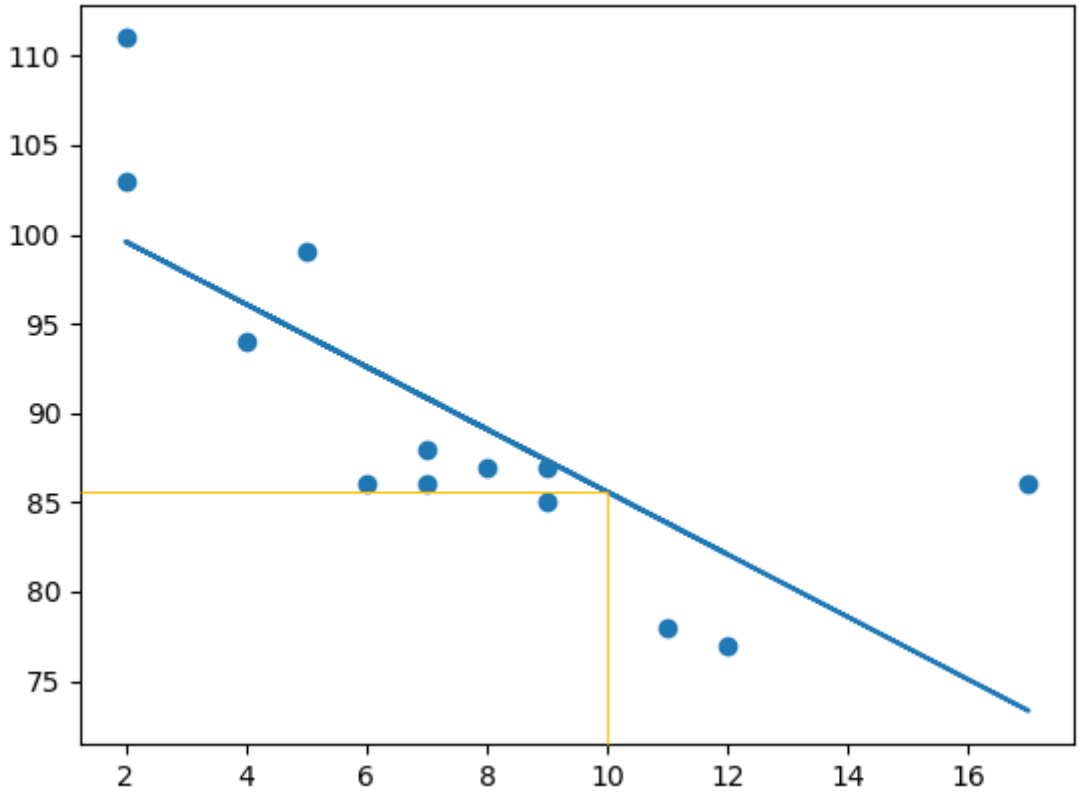
To do so, we need the same `myfunc()` function from the example above:

```
def myfunc(x):
  return slope * x + intercept
```

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

speed = myfunc(10)

print(speed)
```

```
85.59308314937454
```

The example predicted a speed at 85.6, which we also could read from the diagram:


110
105
100
95
90
85
80
75
2
4
6
8
10
12
14
16

Multiple Regression

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on **two or more** variables.

Take a look at the data set below, it contains some information about cars.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94

Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

How Does it Work?

In Python we have modules that will do the work for us. Start by importing the Pandas module.

```
import pandas
```

The Pandas module allows us to read csv files and return a DataFrame object.

The file is meant for testing purposes only, data.csv

```
df = pandas.read_csv("data.csv")
```

Then make a list of the independent values and call this variable X.

Put the dependent values in a variable called y.

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

Car	Model	Volume	Weight	CO2
Toyoty	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104

Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	216	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

We will use some methods from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LinearRegression() method to create a linear regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Now we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm^3:
predictedCO2 = regr.predict([[2300, 1300]])

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)


#predict the CO2 emission of a car where the weight is 2300g, and the
volume is 1300ccm:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

```
[107.2087328]
```

```
Fiat,500,900,865,90
Mini,Cooper,1500,1140,105
VW,Up!,1000,929,105
Skoda,Fabia,1400,1109,90
Mercedes,A-Class,1500,1365,92
Ford,Fiesta,1500,1112,98
Audi,A1,1600,1150,99
Hyundai,I20,1100,980,99
Suzuki,Swift,1300,990,101
Ford,Fiesta,1000,1112,99
Honda,Civic,1600,1252,94
Hundai,I30,1600,1326,97
Opel,Astra,1600,1330,97
BMW,1,1600,1365,99
Mazda,3,2200,1280,104
Skoda,Rapid,1600,1119,104
Ford,Focus,2000,1328,105
Ford,Mondeo,1600,1584,94
Opel,Insignia,2000,1428,99
Mercedes,C-Class,2100,1365,99
Skoda,Octavia,1600,1415,99
Volvo,S60,2000,1415,99
Mercedes,CLA,1500,1465,102
Audi,A4,2000,1490,104
Audi,A6,2000,1725,114
Volvo,V70,1600,1523,109
BMW,5,2000,1705,114
Mercedes,E-Class,2100,1605,115
Volvo,XC70,2000,1746,117
```

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometer it drives.

Coefficient

The coefficient is a factor that describes the relationship with an unknown variable.

Example: if `x` is a variable, then `2x` is `x` two times. `x` is the unknown variable, and the number `2` is the coefficient.

In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2. The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

Example

Print the coefficient values of the regression object:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

```
[0.00755095 0.00780526]
```

Result Explained

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095
Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm^3, the CO2 emission increases by 0.00780526 g.

We have already predicted that if a car with a 1300cm^3 engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Example

Copy the example from before, but change the weight from 2300 to 3300:

```python
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

```
[114.75968007]
```

```
Car,Model,Volume,Weight,CO2
Toyoty,Aygo,1000,790,99
Mitsubishi,Space Star,1200,1160,95
Skoda,Citigo,1000,929,95
Fiat,500,900,865,90
Mini,Cooper,1500,1140,105
VW,Up!,1000,929,105
Skoda,Fabia,1400,1109,90
Mercedes,A-Class,1500,1365,92
Ford,Fiesta,1500,1112,98
Audi,A1,1600,1150,99
Hyundai,I20,1100,980,99
Suzuki,Swift,1300,990,101
Ford,Fiesta,1000,1112,99
Honda,Civic,1600,1252,94
Hundai,I30,1600,1326,97
Opel,Astra,1600,1330,97
BMW,1,1600,1365,99
Mazda,3,2200,1280,104
Skoda,Rapid,1600,1119,104
Ford,Focus,2000,1328,105
Ford,Mondeo,1600,1584,94
Opel,Insignia,2000,1428,99
Mercedes,C-Class,2100,1365,99
Skoda,Octavia,1600,1415,99
Volvo,S60,2000,1415,99
Mercedes,CLA,1500,1465,102
```

We have predicted that a car with 1.3 liter engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometer it drives.

Which shows that the coefficient of 0.00755095 is correct:

107.2087328 + (1000 * 0.00755095) = 114.75968

Overfitting and Underfitting in Machine Learning

Overfitting and Underfitting are the two main problems that occur in machine learning and degrade the performance of the machine learning models.

The main goal of each machine learning model is **to generalize well**. Here **generalization** defines the ability of an ML model to provide a suitable output by adapting the given set of unknown input. It means after providing training on the dataset, it can produce reliable and accurate output. Hence, the underfitting and overfitting are the two terms that need to be checked for the performance of the model and whether the model is generalizing well or not.

Before understanding the overfitting and underfitting, let's understand some basic term that will help to understand this topic well:

- **Signal:** It refers to the true underlying pattern of the data that helps the machine learning model to learn from the data.
- **Noise:** Noise is unnecessary and irrelevant data that reduces the performance of the model.
- **Bias:** Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.
- **Variance:** If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

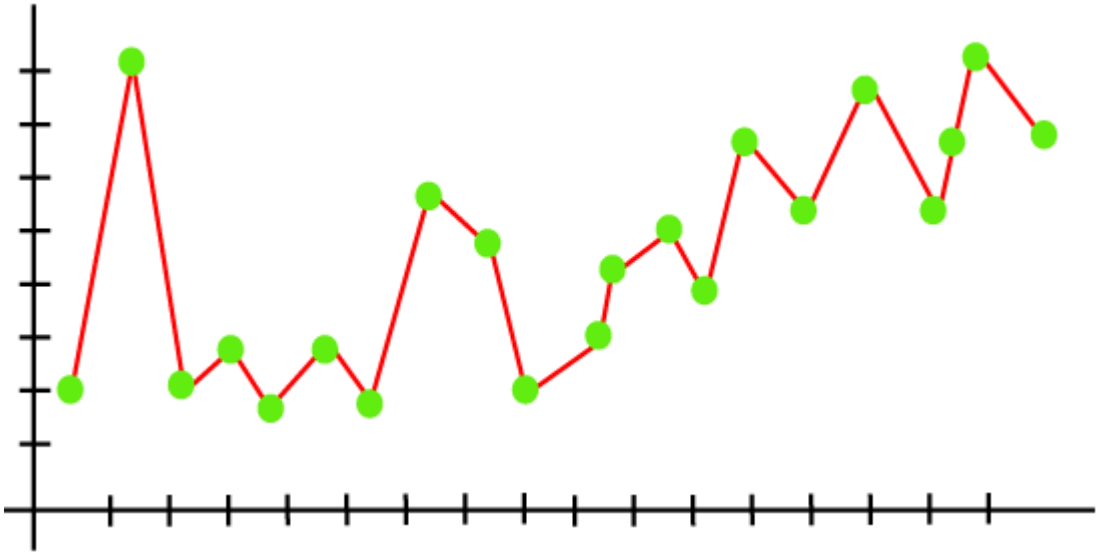
Overfitting

Overfitting occurs when our machine learning model tries to cover all the data points or more than the required data points present in the given dataset. Because of this, the model starts caching noise and inaccurate values present in the dataset, and all these factors reduce the efficiency and accuracy of the model. The overfitted model has **low bias** and **high variance.**

The chances of occurrence of overfitting increase as much we provide training to our model. It means the more we train our model, the more chances of occurring the overfitted model.

Overfitting is the main problem that occurs in supervised learning.

Example: The concept of the overfitting can be understood by the below graph of the linear regression output:



As we can see from the above graph, the model tries to cover all the data points present in the scatter plot. It may look efficient, but in reality, it is not so. Because the goal of the regression model to find the best fit line, but here we have not got any best fit, so, it will generate the prediction errors.

How to avoid the Overfitting in Model

Both overfitting and underfitting cause the degraded performance of the machine learning model. But the main cause is overfitting, so there are some ways by which we can reduce the occurrence of overfitting in our model.

- **Cross-Validation**
- **Training with more data**
- **Removing features**
- **Early stopping the training**
- **Regularization**
- **Ensembling**

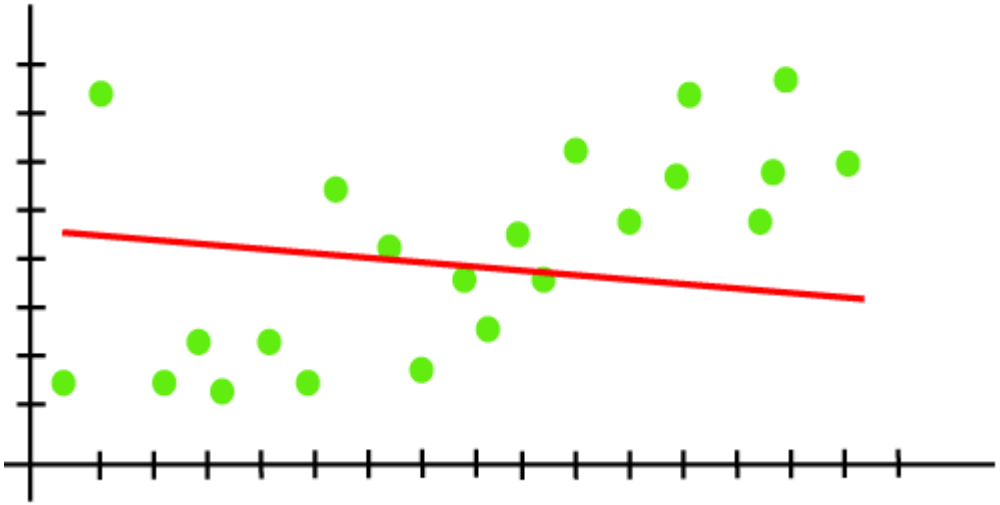
Underfitting

Underfitting occurs when our machine learning model is not able to capture the underlying trend of the data. To avoid the overfitting in the model, the fed of training data can be stopped at an early stage, due to which the model may not learn enough from the training data. As a result, it may fail to find the best fit of the dominant trend in the data.

In the case of underfitting, the model is not able to learn enough from the training data, and hence it reduces the accuracy and produces unreliable predictions.

An underfitted model has high bias and low variance.

Example: We can understand the underfitting using below output of the linear regression model:



As we can see from the above diagram, the model is unable to capture the data points present in the plot.

How to avoid underfitting:

- By increasing the training time of the model.
- By increasing the number of features.

Goodness of Fit

The "Goodness of fit" term is taken from the statistics, and the goal of the machine learning models to achieve the goodness of fit. In statistics modeling, *it defines how closely the result or predicted values match the true values of the dataset.*

The model with a good fit is between the underfitted and overfitted model, and ideally, it makes predictions with 0 errors, but in practice, it is difficult to achieve it.

As when we train our model for a time, the errors in the training data go down, and the same happens with test data. But if we train the model for a long duration, then the performance of the model may decrease due to the overfitting, as the model also learn the noise present in the dataset. The errors in the test dataset start increasing, *so the point, just before the raising of errors, is the good point, and we can stop here for achieving a good model.*

There are two other methods by which we can get a good point for our model, which are the **resampling method** to estimate model accuracy and **validation dataset**.

Bias and Variance in Machine Learning

- **Bias**: Bias refers to the error due to overly simplistic assumptions in the learning algorithm. These assumptions make the model easier to comprehend and learn but might not capture the underlying complexities of the data. It is the error due to the model's inability to represent the true relationship between input and output accurately. When a model has poor performance both on the training and testing data means high bias because of the simple model, indicating underfitting.

- **Variance**: Variance, on the other hand, is the error due to the model's sensitivity to fluctuations in the training data. It's the variability of the model's predictions for different instances of training data. High variance occurs when a model learns the training data's noise and random fluctuations rather than the underlying pattern. As a result, the model performs well on the training data but poorly on the testing data, indicating overfitting.

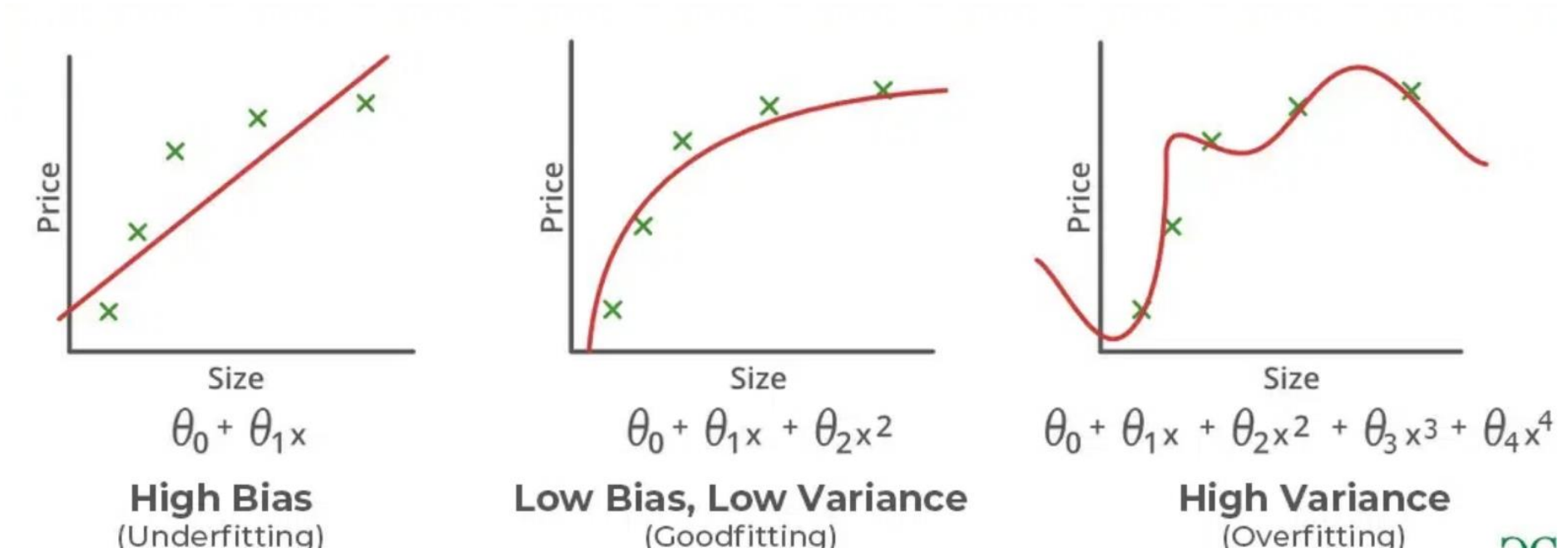
Bias and Variance

Underfitting in Machine Learning

A statistical model or a machine learning algorithm is said to have underfitting when a model is too simple to capture data complexities. It represents the inability of the model to learn the training data effectively result in poor performance both on the training and testing data. In simple terms, an underfit model's are inaccurate, especially when applied to new, unseen examples. It

mainly happens when we uses very simple model with overly simplified assumptions. To address underfitting problem of the model, we need to use more complex models, with enhanced feature representation, and less regularization.

Note: The underfitting model has High bias and low variance.



Reasons for Underfitting

1. The model is too simple, So it may be not capable to represent the complexities in the data.

2. The input features which is used to train the model is not the adequate representations of underlying factors influencing the target variable.

3. The size of the training dataset used is not enough.

4. Excessive regularization are used to prevent the overfitting, which constraint the model to capture the data well.

5. Features are not scaled.

Techniques to Reduce Underfitting

1. Increase model complexity.

2. Increase the number of features, performing feature engineering.

3. Remove noise from the data.

4. Increase the number of epochs or increase the duration of training to get better results.

Overfitting in Machine Learning

A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. And when testing with test data results in High variance. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

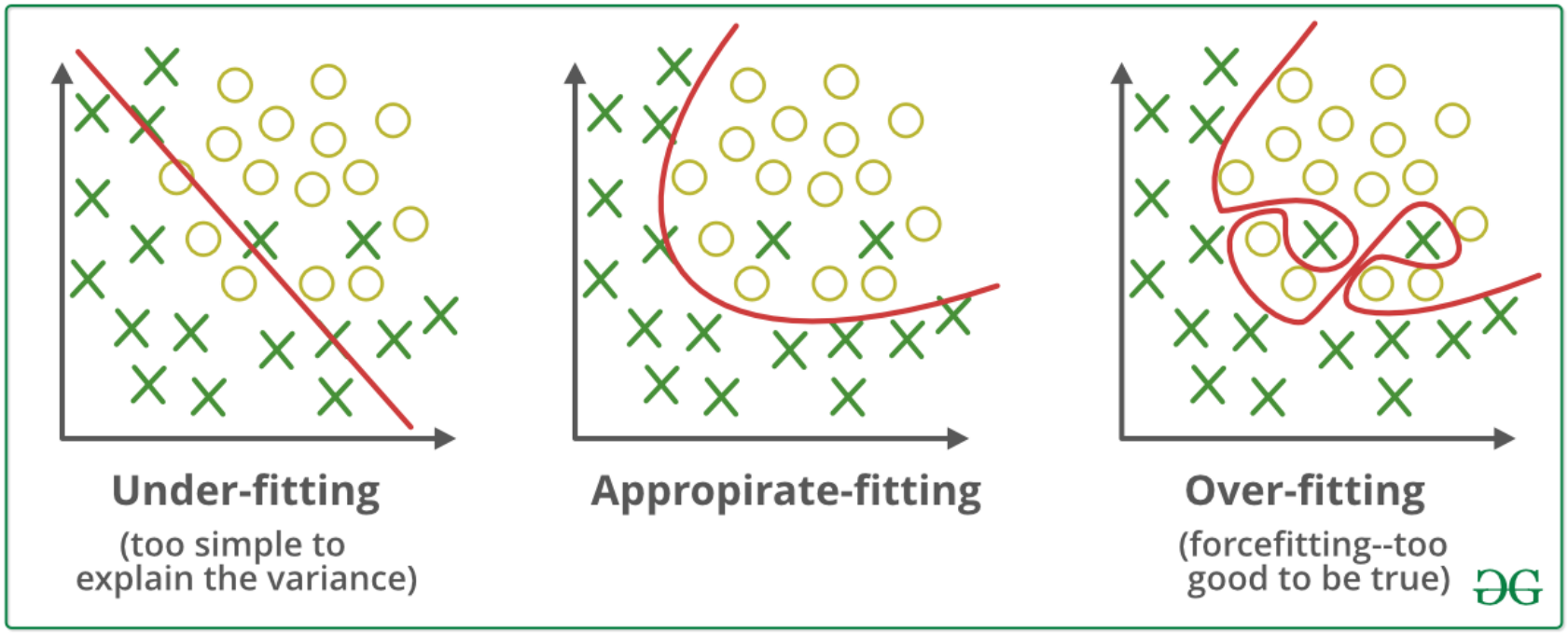
In a nutshell, Overfitting is a problem where the evaluation of machine learning algorithms on training data is different from unseen data.

Reasons for Overfitting:

1. High variance and low bias.
2. The model is too complex.
3. The size of the training data.

Techniques to Reduce Overfitting

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization and Lasso Regularization.
5. Use dropout for neural networks to tackle overfitting.



Underfitting and Overfitting

Good Fit in a Statistical Model

Ideally, the case when the model makes the predictions with 0 error, is said to have a good fit on the data. This situation is achievable at a spot between overfitting and underfitting. In order to understand it, we will have to look at the performance of our model with the passage of time, while it is learning from the training dataset.

With the passage of time, our model will keep on learning, and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point, the model is said to have good skills in training datasets as well as our unseen testing dataset.

MODULE-5

Supervised Learning, Regression, Linear regression, Multiple linear regression, A multiple regression analysis, The analysis of variance for multiple regression, Examples for multiple regression, Overfitting, Detecting overfit models: Cross validation, Cross validation: The ideal procedure, Parameter estimation, Logistic regression, Decision trees: Background, Decision trees, Decision trees for credit card promotion, An algorithm for building decision trees, Attribute selection measure: Information gain, Entropy, Decision Tree: Weekend example, Occam's Razor, Converting a tree to rules, Unsupervised learning, Semi-Supervised learning, Clustering, K – means clustering, Automated discovery, Reinforcement learning, Multi-Armed Bandit algorithms, Influence diagrams, Risk modeling, Sensitivity analysis, Casual learning.

CROSS VALIDATION

Cross-validation is **a technique for evaluating ML models by training several ML models on subsets of the available input data and evaluating them on the complementary subset of the data**. Use cross-validation to detect overfitting, ie, failing to generalize a pattern.

To ensure that the model is correctly trained on the data provided without much noise, you need to use cross-validation techniques. These are statistical methods used to estimate the performance of machine learning models.

Types of cross-validation

1. **K-fold cross-validation**
2. **Hold-out cross-validation**
3. **Stratified k-fold cross-validation**
4. **Leave-p-out cross-validation**
5. **Leave-one-out cross-validation**

K-Fold Cross Validation Technique and its Essentials

K-fold cross-validation is a technique for evaluating predictive models. The dataset is divided into k subsets or folds. The model is trained and evaluated k times, using a different fold as the validation set each time. Performance metrics from each fold are averaged to estimate the model's generalization performance. This method aids in model assessment, selection, and hyperparameter tuning, providing a more reliable measure of a model's effectiveness.

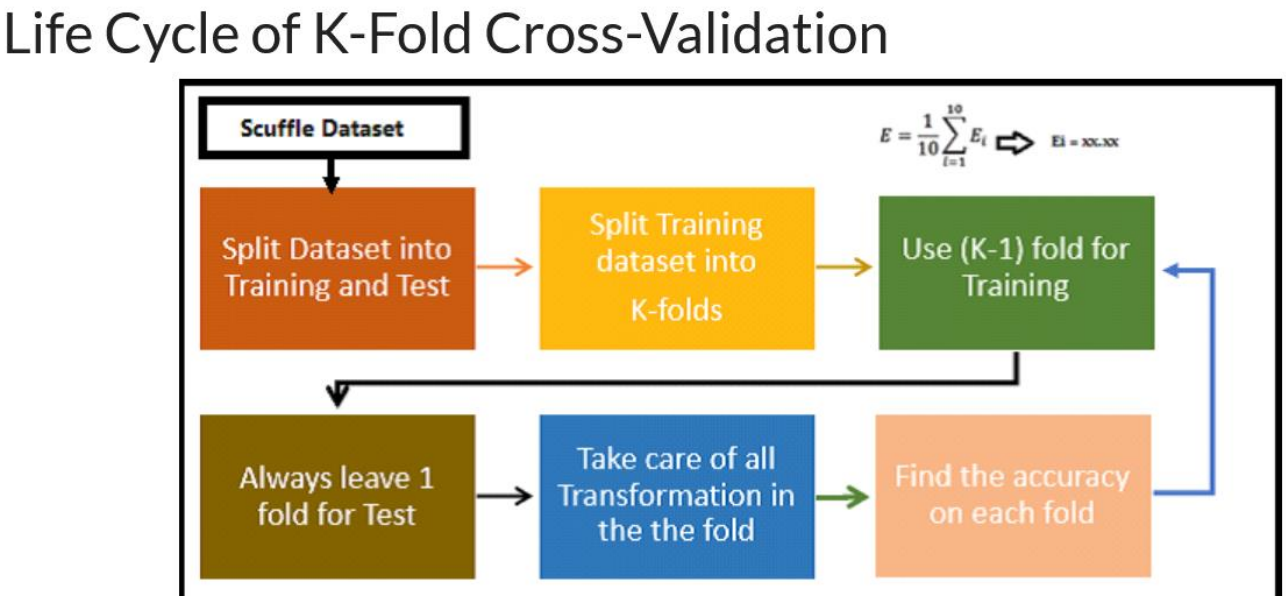
In each set (fold) training and the test would be performed precisely once during this entire process. It helps us to avoid overfitting. As we know when a model is trained using all of the data in a single short and give the best performance accuracy. To resist this k-fold cross-validation helps us to build the model is a generalized one.

To achieve this K-Fold Cross Validation, we have to split the data set into three sets, Training, Testing, and Validation, with the challenge of the volume of the data.

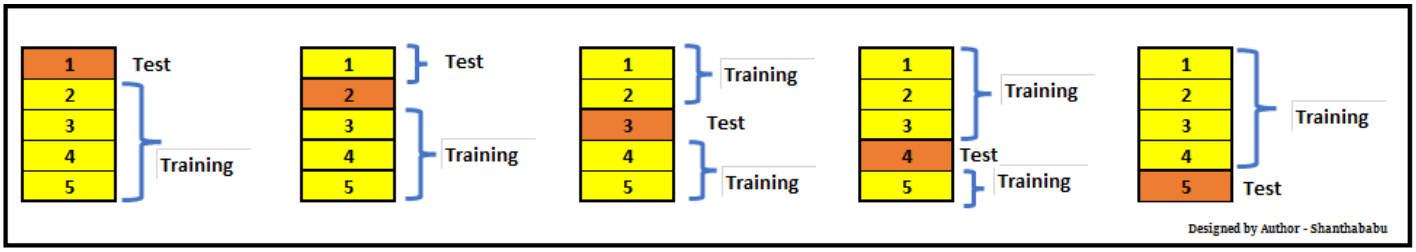
Here Test and Train data set will support building model and hyperparameter assessments.

In which the model has been validated multiple times based on the value assigned as a parameter and which is called K and it should be an INTEGER.

Make it simple, based on the K value, the data set would be divided, and train/testing will be conducted in a sequence way equal to K time.



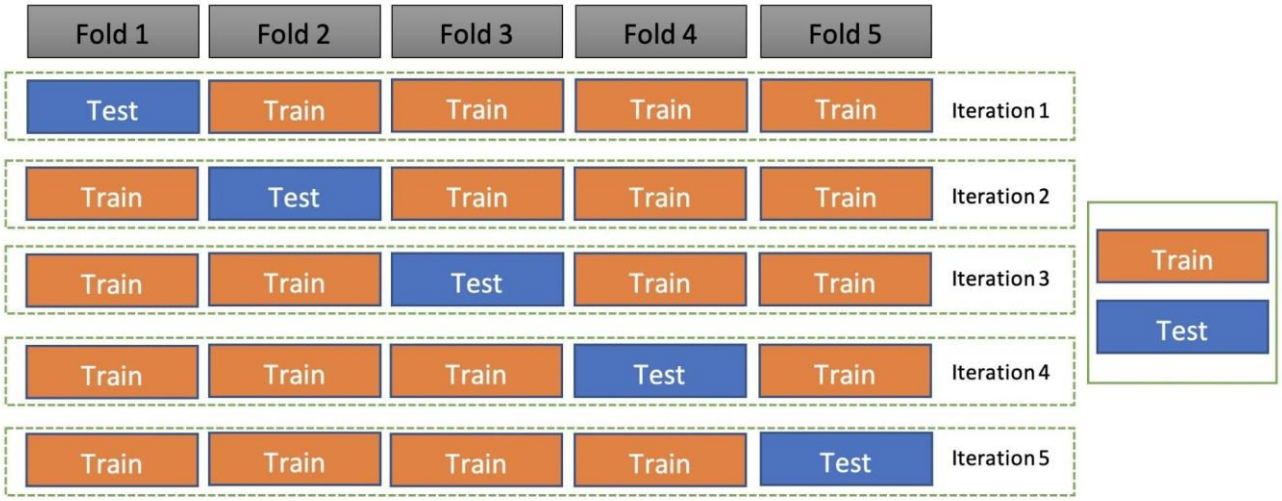
Let's have a generalised K value. If K=5, it means, in the given dataset and we are splitting into 5 folds and running the Train and Test. During each run, one fold is considered for testing and the rest will be for training and moving on with iterations, the below pictorial representation would give you an idea of the flow of the fold-defined size.



K-fold cross-validation

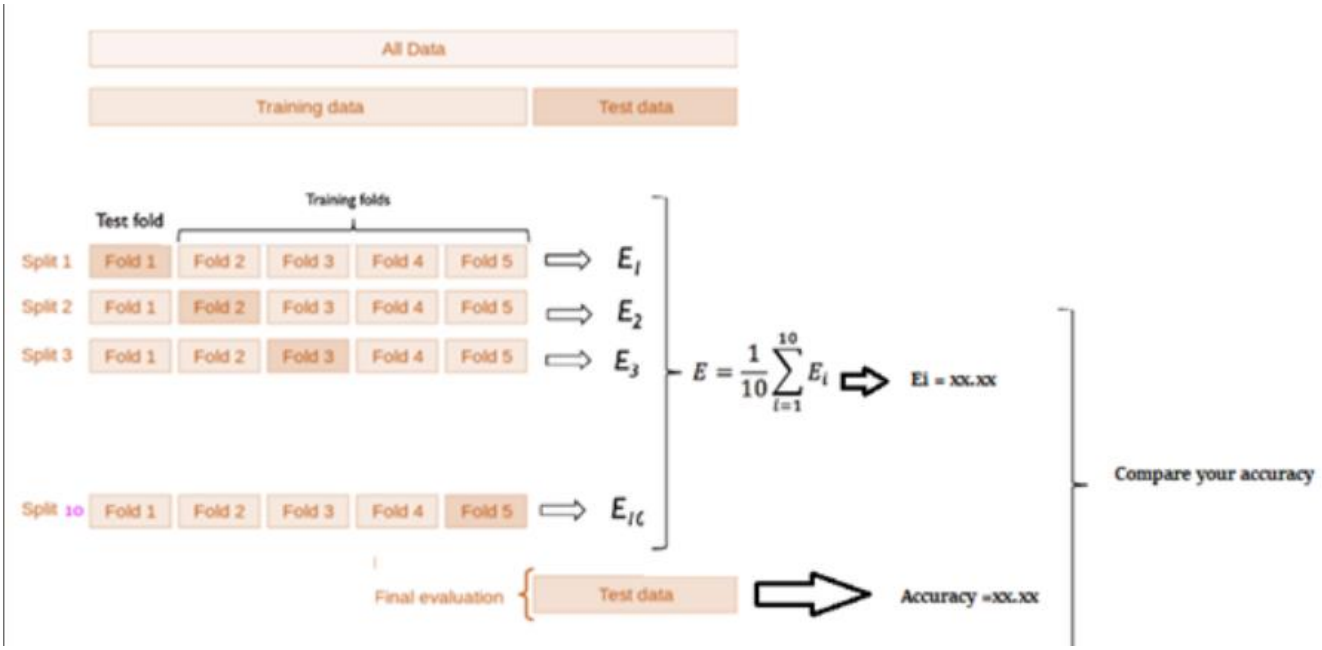
In this technique, the whole dataset is partitioned in k parts of equal size and each partition is called a fold. It's known as k-fold since there are k parts where k can be any integer - 3,4,5, etc.

One fold is used for validation and other K-1 folds are used for training the model. To use every fold as a validation set and other left-outs as a training set, this technique is repeated k times until each fold is used once.



The image above shows 5 folds and hence, 5 iterations. In each iteration, one fold is the test set/validation set and the other k-1 sets (4 sets) are the train set. To get the final accuracy, you need to take the accuracy of the k-models validation data.

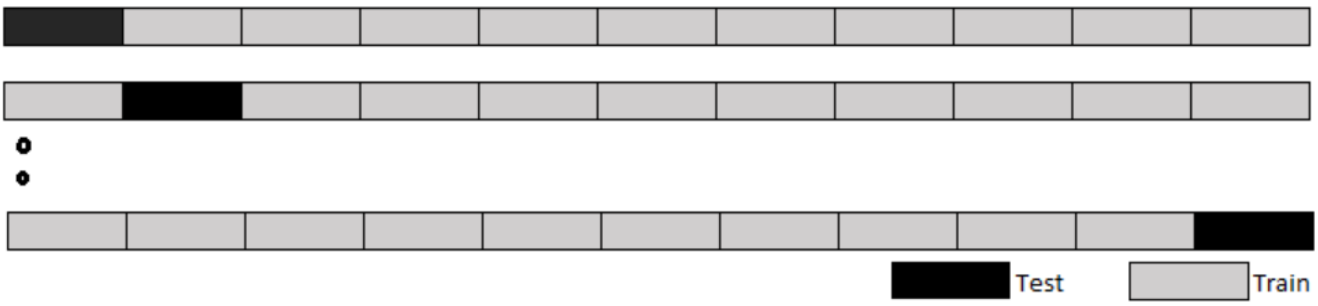
This validation technique is not considered suitable for imbalanced datasets as the model will not get trained properly owing to the proper ratio of each class's data.



Rules Associated with K Fold

Now, we will discuss a few thumb rules while playing with K – fold

- K should be always >= 2 and = to number of records, (LOOCV)
    - If 2 then just 2 iterations
    - If K=No of records in the dataset, then 1 for testing and n- for training
- The optimized value for the K is 10 and used with the data of good size. (Commonly used)
- If the K value is too large, then this will lead to less variance across the training set and limit the model currency difference across the iterations.
- The number of folds is indirectly proportional to the size of the data set, which means, if the dataset size is too small, the number of folds can increase.
- Larger values of K eventually increase the running time of the cross-validation process.

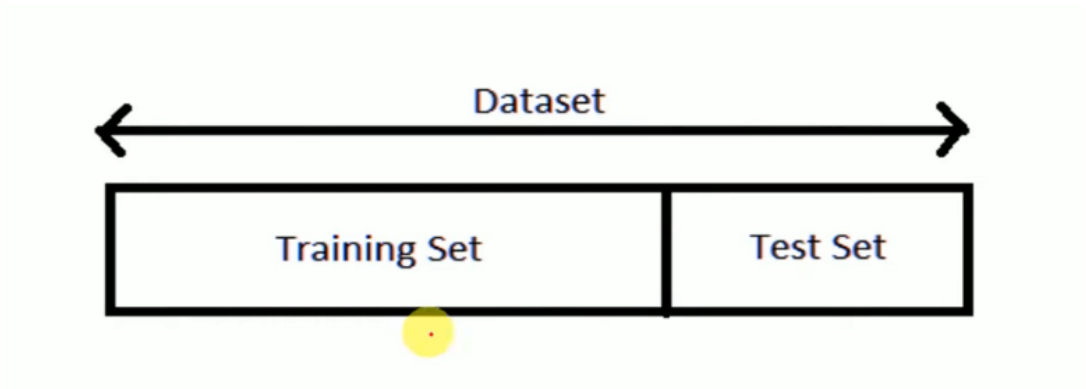


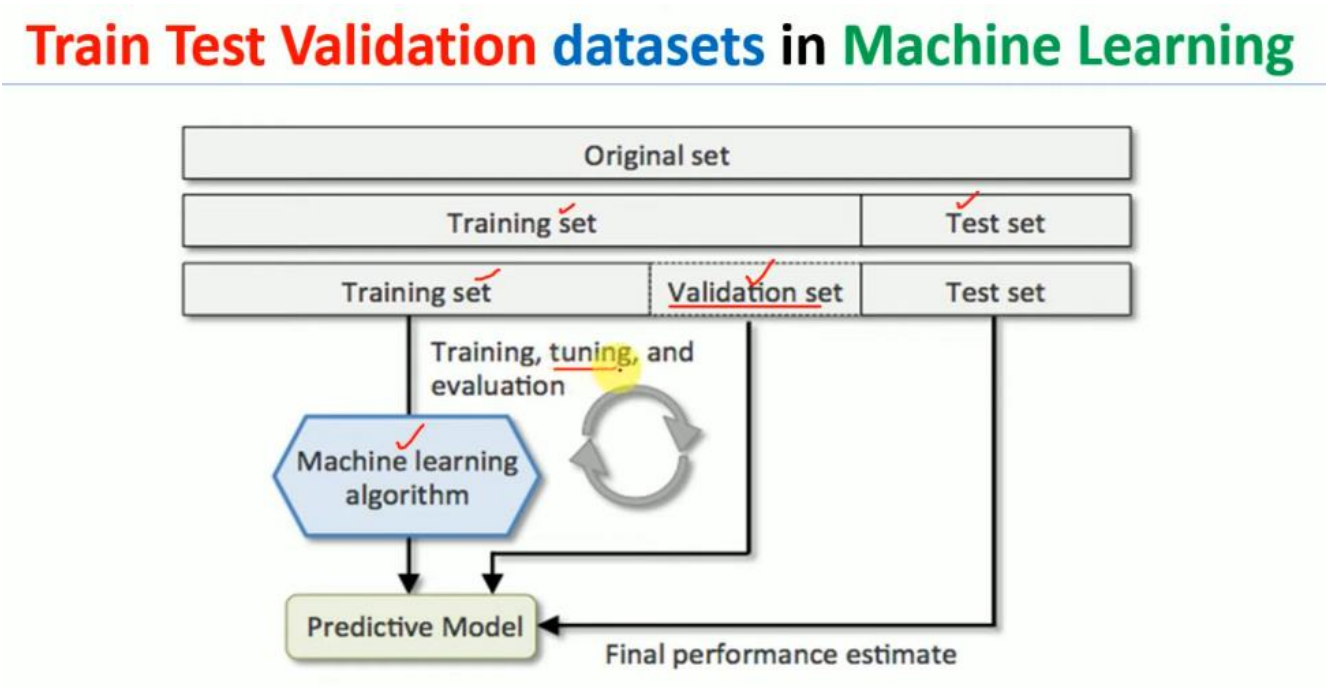
Basic Example

I am creating a simple array, defining the K size as 5 and splitting my array. Using the simple loop and printing the Train and Test portions. Here we could see clearly that the data points in TT buckets and Test data are unique in each cycle.

```python
from sklearn.model_selection import KFold

import numpy as np

data=np.array([5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80
,85,90,95,100])
kfold=KFold(5,shuffle = True)
for train,test in kfold.split(data):
  print("Train: %s,Test: %s"%(data[train],data[test]))
```

```
Train: [  5  10  15  20  25  40  45  50  60  65  70  75  85  90  95 100],Test: [30 35 55 80
]
Train: [  5  15  20  25  30  35  50  55  65  70  75  80  85  90  95 100],Test: [10 40 45 60
]
Train: [ 10  15  25  30  35  40  45  50  55  60  65  70  80  85  90 100],Test: [ 5 20 75 95
]
Train: [ 5 10 20 25 30 35 40 45 50 55 60 70 75 80 90 95],Test: [ 15  65  85 100]
Train: [  5  10  15  20  30  35  40  45  55  60  65  75  80  85  95 100],Test: [25 50 70 90
```





To avoid overfitting we are using cross validation

Cross Validation in Machine Learning

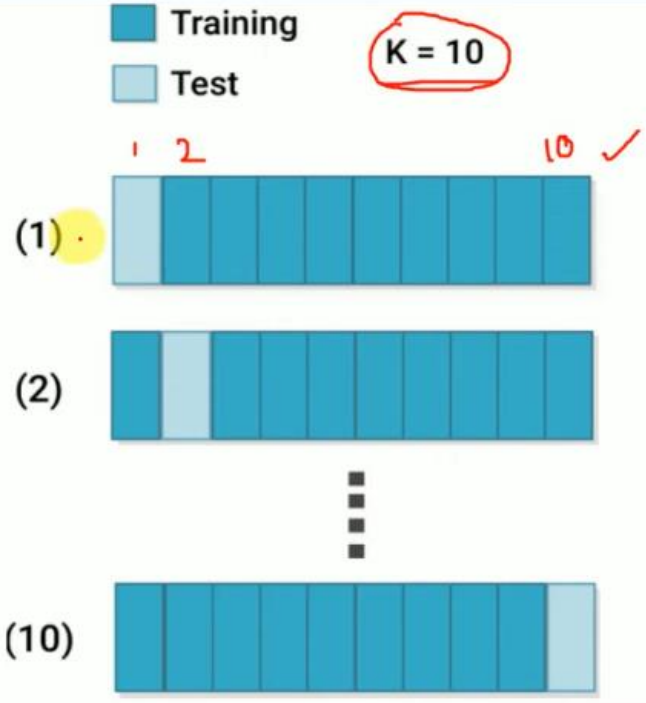
- The idea of cross-validation arises because of the *problems* with train test model or train test validation model.
- It basically wants to guarantee that the score of our model does not depend on the way we picked the train and test set.

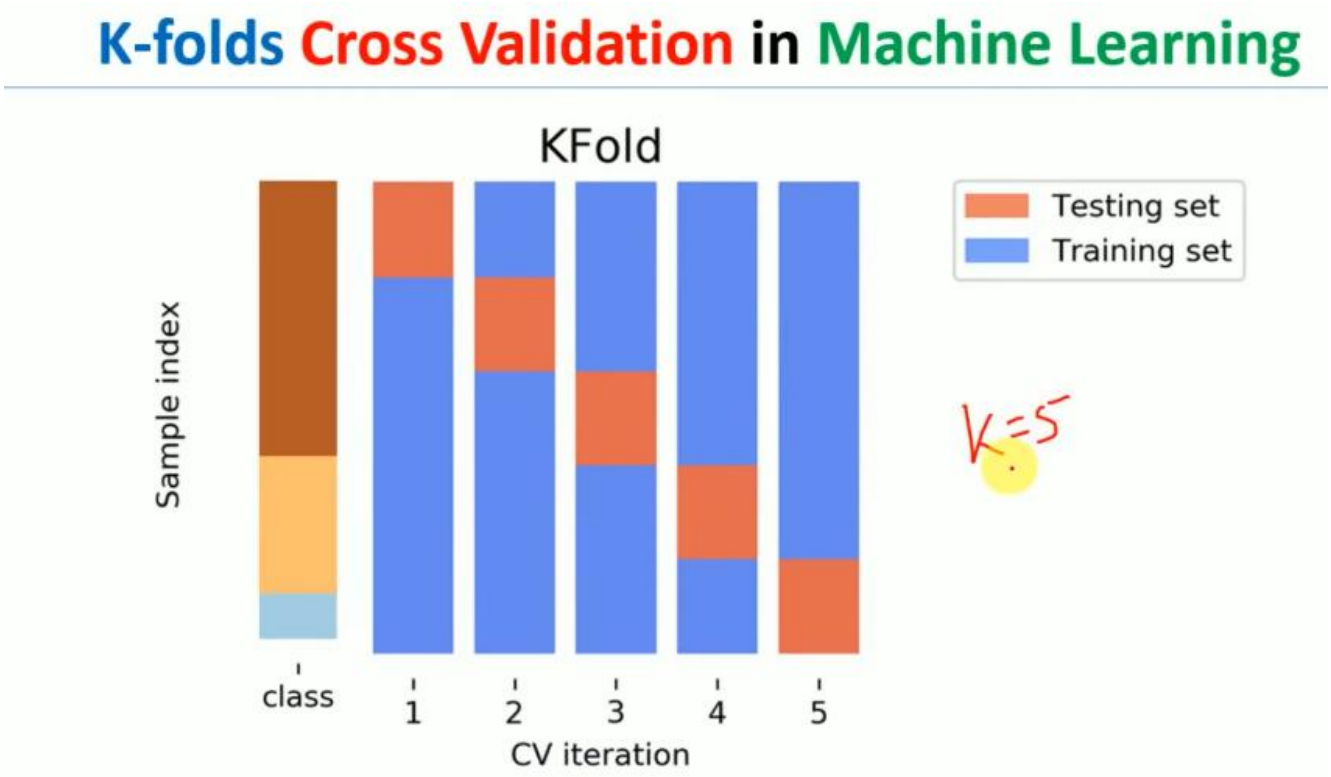
Types Cross Validation in Machine Learning

Following are the type of Cross Validation Techniques

- K-folds
- Stratified K-folds
- Leave-one-out
- Leave-p-out

K-Fold Cross Validation in Machine Learning





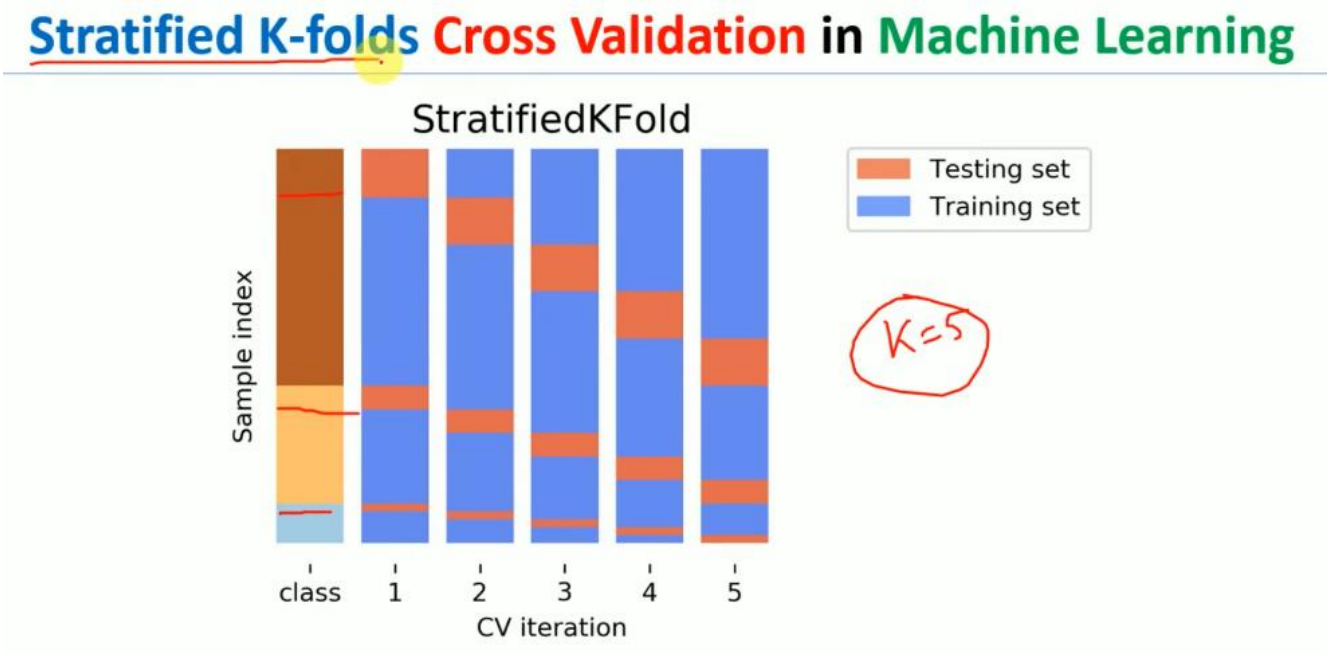
To avoid Overfitting we are using Stratified KFold

Stratified k-fold cross-validation

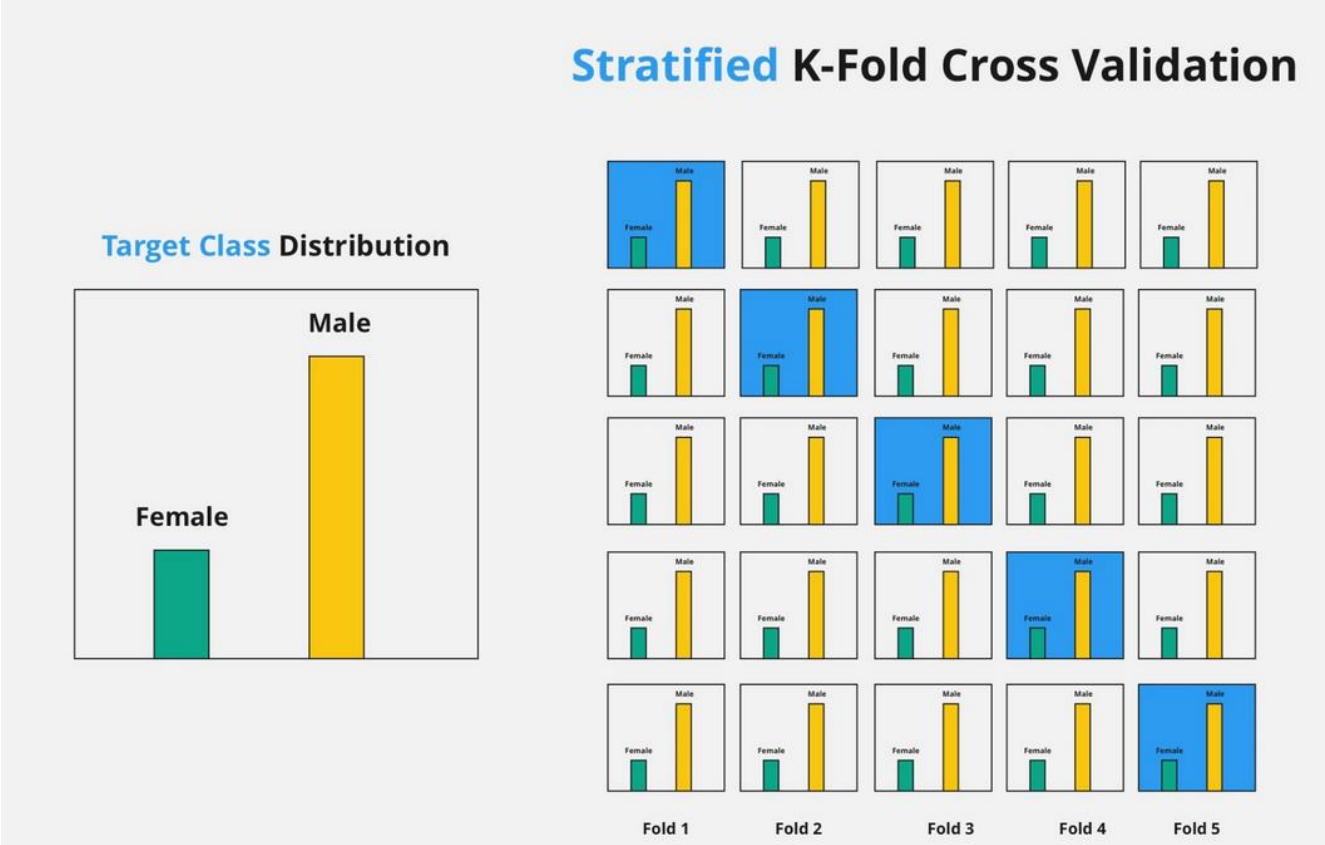
As seen above, k-fold validation can't be used for imbalanced datasets because data is split into k-folds with a uniform probability distribution. Not so with stratified k-fold, which is an enhanced version of the k-fold cross-validation technique.

Although it too splits the dataset into k equal folds, each fold has the same ratio of instances of target variables that are in the complete dataset.

This enables it to work perfectly for imbalanced datasets, but not for time-series data.



Equal representation values



In the example above, the original dataset contains females that are a lot less than males, so this target variable distribution is imbalanced.

In the stratified k-fold cross-validation technique, this ratio of instances of the target variable is maintained in all the folds.

Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score,StratifiedKFold
from sklearn.linear_model import LogisticRegression
iris=load_iris()
X=iris.data
Y=iris.target
lr=LogisticRegression()
st_kf=StratifiedKFold(n_splits=3)
score=cross_val_score(lr,X,Y,cv=st_kf)
print("Cross Validation Scores:{}".format(score))
print("Average Cross Validation score:{}".format(score.mean()))
```

Output:

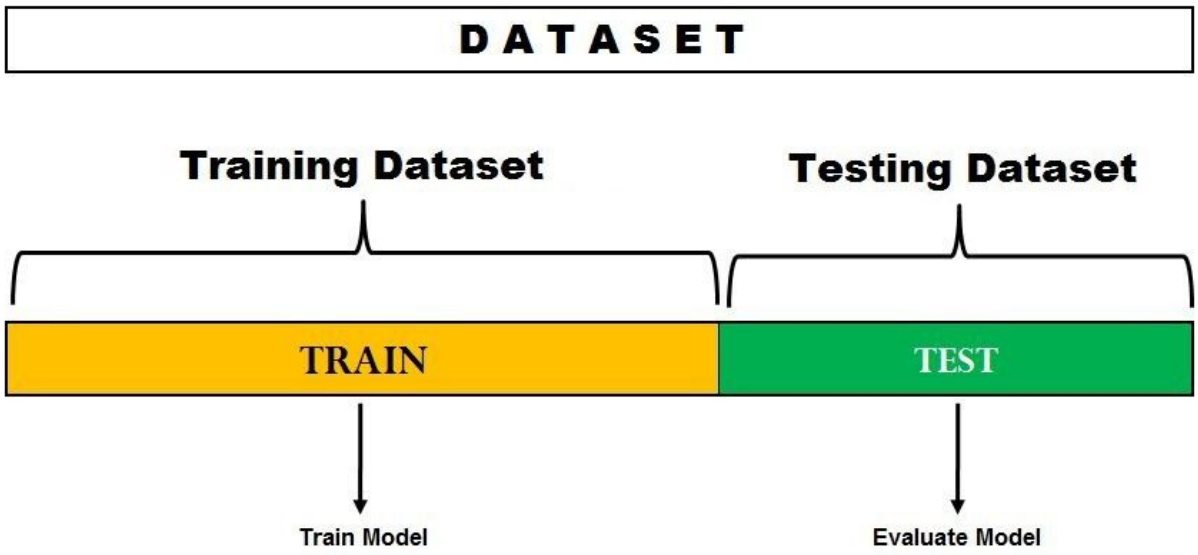
```
Cross Validation Scores:[0.98 0.96 0.98]
Average Cross Validation score:0.9733333333333333
```

Holdout cross-validation

Also called a train-test split, holdout cross-validation has the entire dataset partitioned randomly into a training set and a validation set.

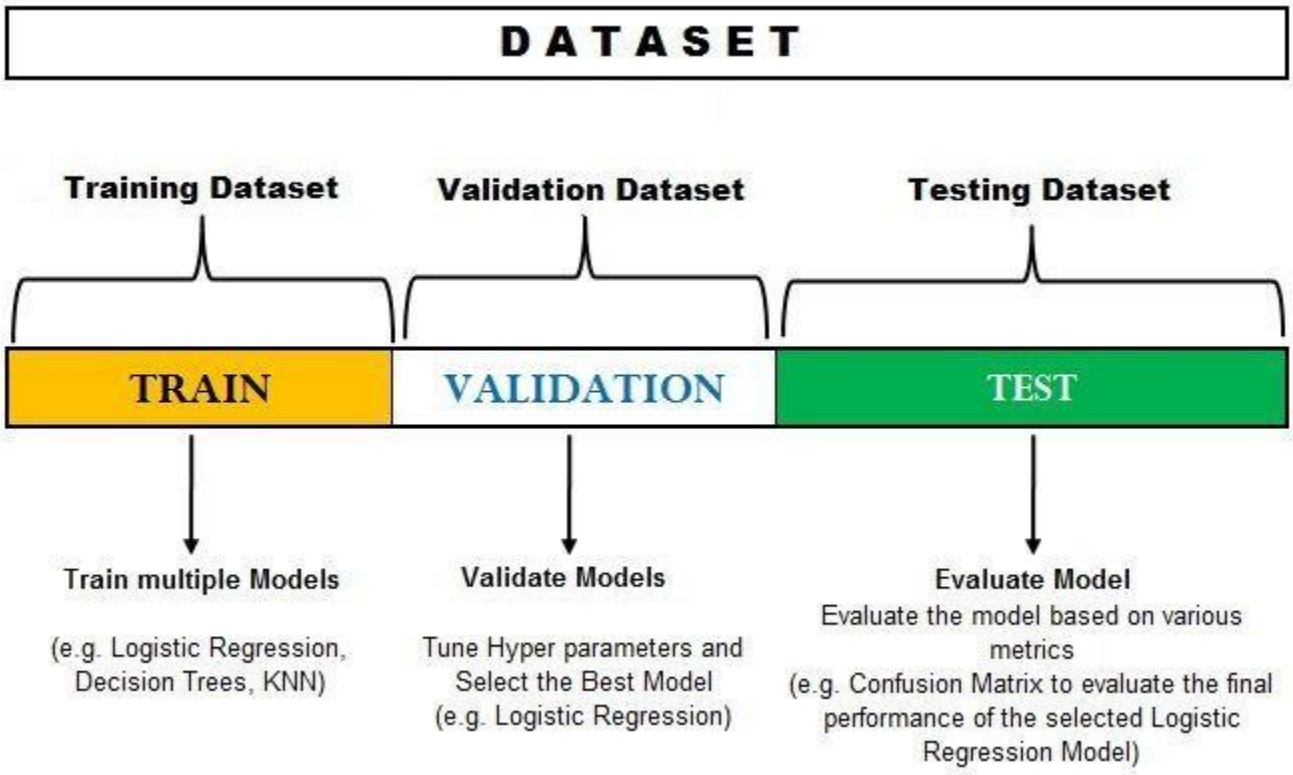
A rule of thumb to partition data is that nearly 70% of the whole dataset will be used as a training set and the remaining 30% will be used as a validation set.

Since the dataset is split into only two sets, the model is built just one time on the training set and executed faster.



In the image above, the dataset is split into a training set and a test set. You can train the model on the training set and test it on the testing dataset.

However, if you want to hyper-tune your parameters or want to select the best model, you can make a validation set like the one below.

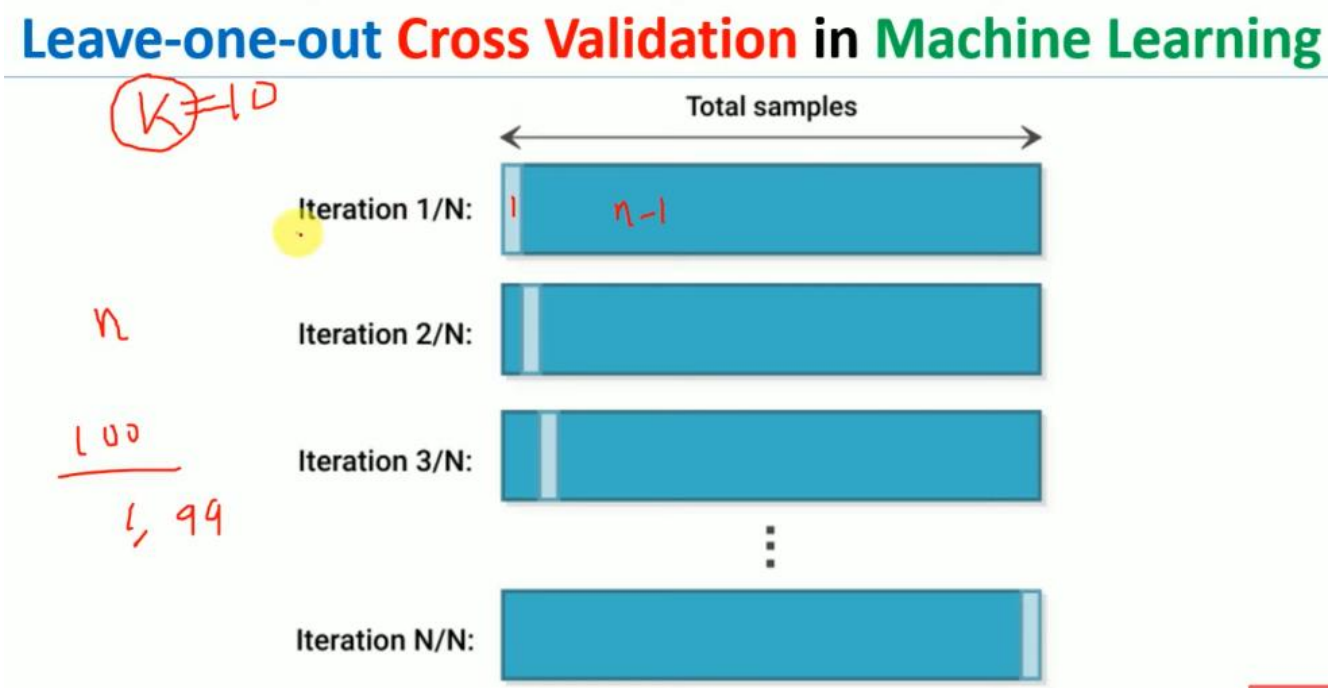


Code:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
iris=load_iris()
X=iris.data
Y=iris.target
lr=LogisticRegression()
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,ran
dom_state=42)
lr.fit(x_train,y_train)
results=lr.predict(x_test)
print("Training
accuracy:{}".format(accuracy_score(lr.predict(x_train),y_train)))
print("Testing accuracy:{}".format(accuracy_score(results,y_test)))
```

Output:

```
Training accuracy:0.9642857142857143
Testing accuracy:1.0
```



It use more time.

Leave-one-out cross-validation

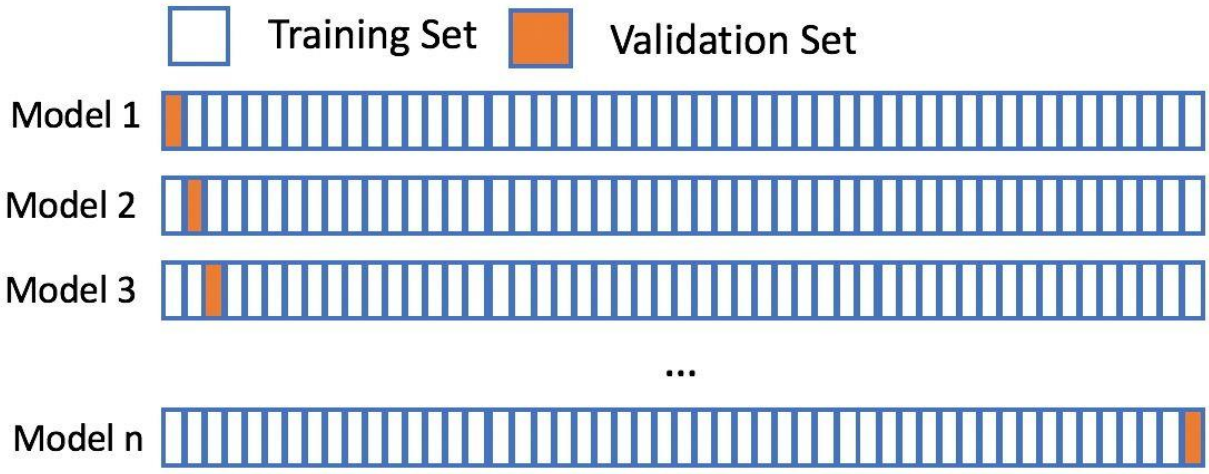
In this technique, only 1 sample point is used as a validation set and the remaining n-1 samples are used in the training set.

Think of it as a more specific case of the leave-p-out cross-validation technique with P=1.

To understand this better, consider this example:

There are 1000 instances in your dataset. In each iteration, 1 instance will be used for the validation set and the remaining 999 instances will be used as the training set.

The process repeats itself until every instance from the dataset is used as a validation sample.



The leave-one-out cross-validation method is computationally expensive to perform and shouldn't be used with very large datasets. This technique is very simple and requires no configuration to specify. It also provides a reliable and unbiased estimate for your model performance.

Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import LeaveOneOut,cross_val_score
iris=load_iris()
X=iris.data
Y=iris.target
leave_one_out=LeaveOneOut()
rfc=RandomForestClassifier(n_estimators=7,max_depth=4,n_jobs=-1)
score=cross_val_score(rfc,X,Y,cv=leave_one_out)
print("Cross Validation Scores:{}".format(score))
print("Average Cross Validation score:{}".format(score.mean()))
```

Output:

```
Cross Validation Scores:[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1.
1.
 1. 1. 1. 1. 1. 1.]
Average Cross Validation score:0.9466666666666667
```

Leave-p-out cross-validation

An exhaustive cross-validation technique, p samples are used as the validation set and n-p samples are used as the training set if a dataset has n samples. The process is repeated until the entire dataset containing n samples gets divided on the validation set of p samples and the training set of n-p samples. This continues till all samples are used as a validation set.

The technique, which has a high computation time, produces good results.

However, it's not considered ideal for an imbalanced dataset and is deemed to be a computationally unfeasible method. This is because if the training set has all samples of one class, the model will not be able to properly generalize and will become biased to either of the classes.

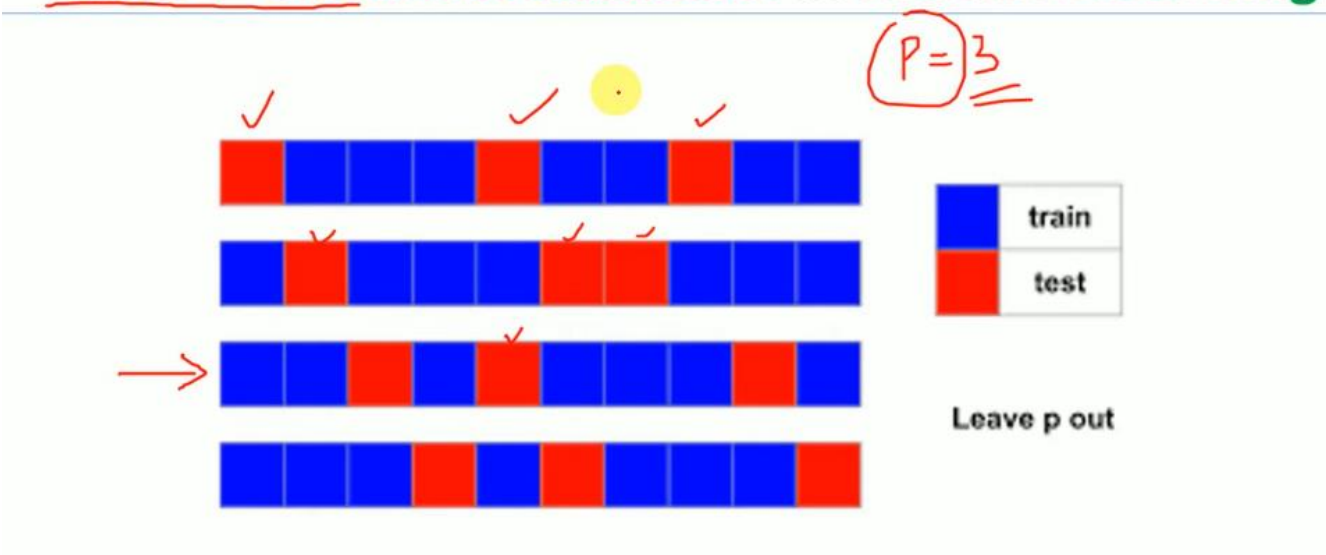
Code:

```
from sklearn.model_selection import LeavePOut,cross_val_score
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
iris=load_iris()
X=iris.data
Y=iris.target
lpo=LeavePOut(p=1)
lpo.get_n_splits(X)
tree=RandomForestClassifier(n_estimators=5,max_depth=3,n_jobs=-1)
score=cross_val_score(tree,X,Y,cv=lpo)
print("Cross Validation Scores-{}".format(score))
print("Average Cross Validation score :{}".format(score.mean()))
```

Output:

```
Cross Validation Scores-[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
Average Cross Validation score :0.9533333333333334
```

Leave-P-out Cross Validation in Machine Learning



MODULE-5

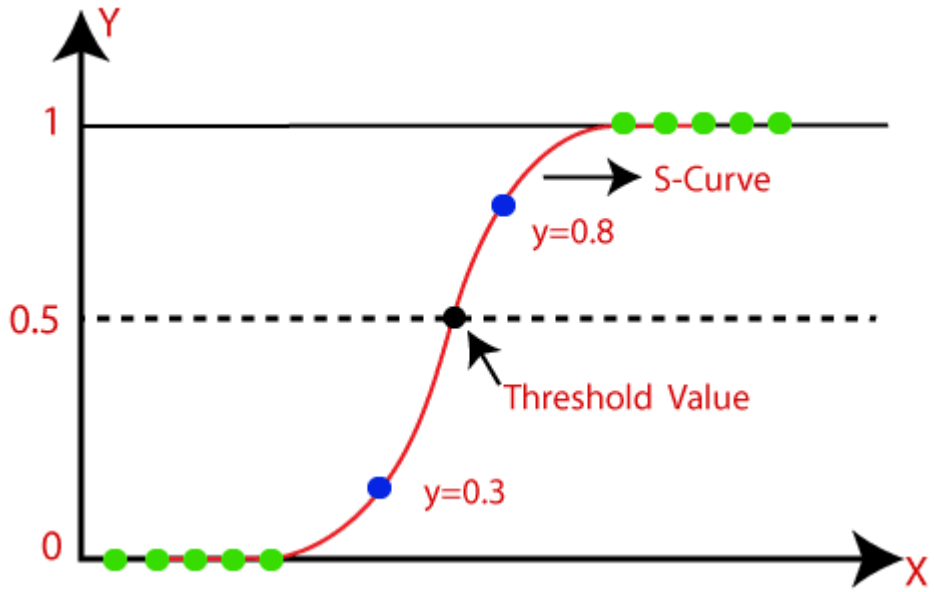
Supervised Learning, Regression, Linear regression, Multiple linear regression, A multiple regression analysis, The analysis of variance for multiple regression, Examples for multiple regression, Overfitting, Detecting overfit models: Cross validation, Cross validation: The ideal procedure, Parameter estimation, Logistic regression, Decision trees: Background, Decision trees, Decision trees for credit card promotion, An algorithm for building decision trees, Attribute selection measure: Information gain, Entropy, Decision Tree: Weekend example, Occam's Razor, Converting a tree to rules, Unsupervised learning, Semi-Supervised learning, Clustering, K – means clustering, Automated discovery, Reinforcement learning, Multi-Armed Bandit algorithms, Influence diagrams, Risk modeling, Sensitivity analysis, Casual learning.

LOGISTIC REGRESSION

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.
- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:

- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).



Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression:

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Logistic regression aims to solve classification problems. It does this by predicting categorical outcomes, unlike linear regression that predicts a continuous outcome.

In the simplest case there are two outcomes, which is called binomial, an example of which is predicting if a tumor is malignant or benign. Other cases have more than two outcomes to classify, in this case it is called multinomial. A common example for multinomial logistic regression would be predicting the class of an iris flower between 3 different species.

How does it work?

In Python we have modules that will do the work for us. Start by importing the NumPy module.

```
import numpy
```

Store the independent variables in X.

Store the dependent variable in y.

Below is a sample dataset:

```
#X represents the size of a tumor in centimeters.
X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96, 4.52, 3.69,
5.88]).reshape(-1,1)

#Note: X has to be reshaped into a column from a row for the LogisticRegression() function to
work.
#y represents whether or not the tumor is cancerous (0 for "No", 1 for "Yes").
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

We will use a method from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the LogisticRegression() method to create a logistic regression object.

This object has a method called `fit()` that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
logr = linear_model.LogisticRegression()
logr.fit(X,y)
```

Now we have a logistic regression object that is ready to whether a tumor is cancerous based on the tumor size:

```
#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))
```

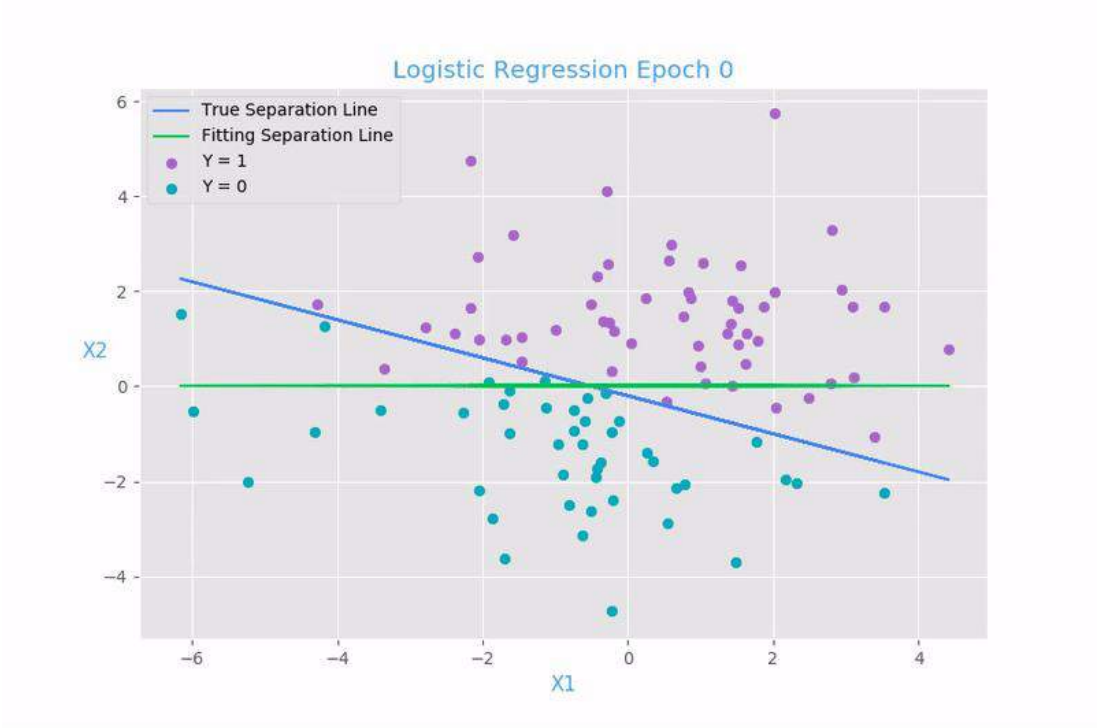
Example

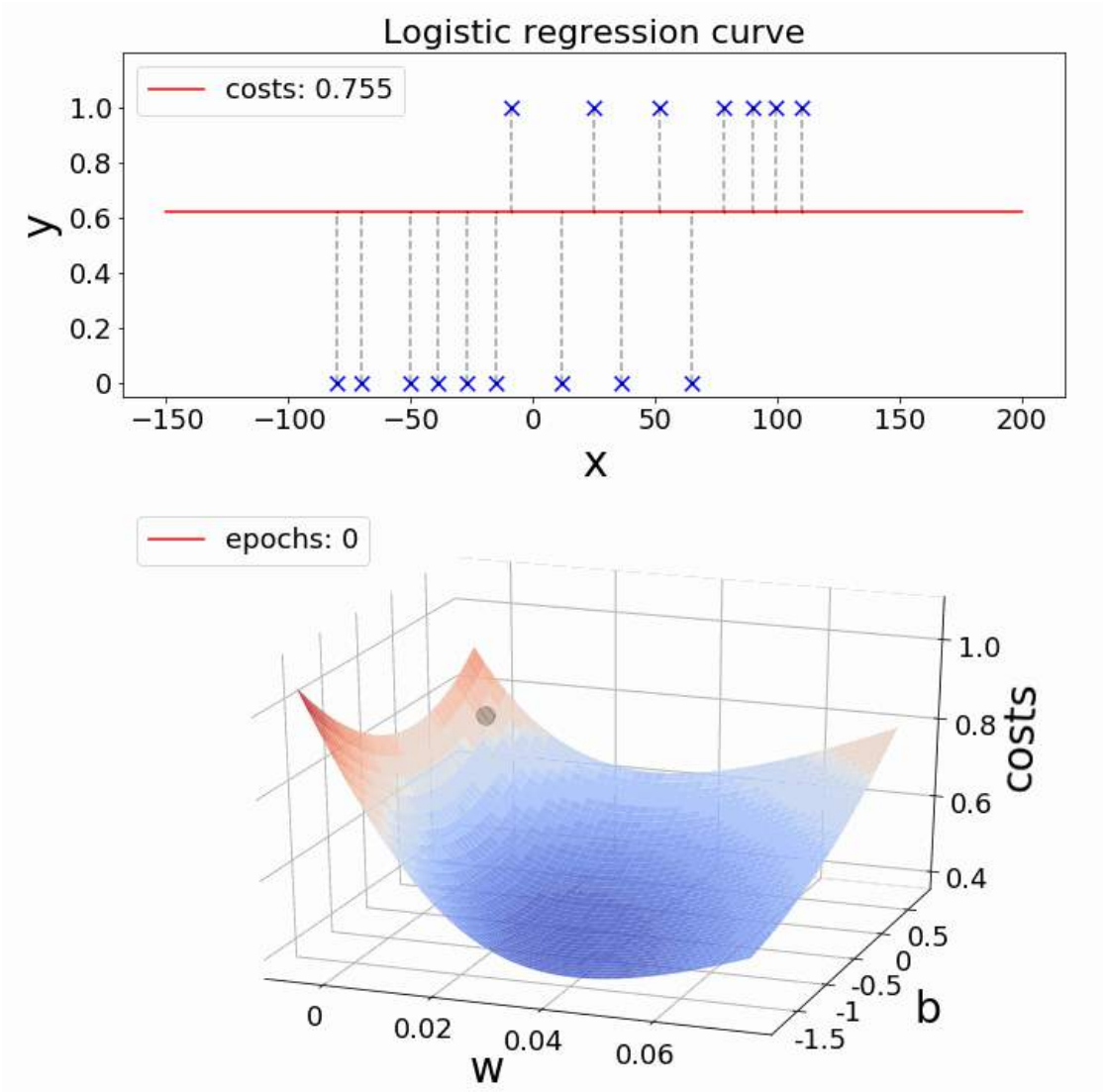
```
import numpy
from sklearn import linear_model

X = numpy.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96,
4.52, 3.69, 5.88]).reshape(-1,1)
y = numpy.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

logr = linear_model.LogisticRegression()
logr.fit(X,y)

#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(numpy.array([3.46]).reshape(-1,1))

print(predicted)
```

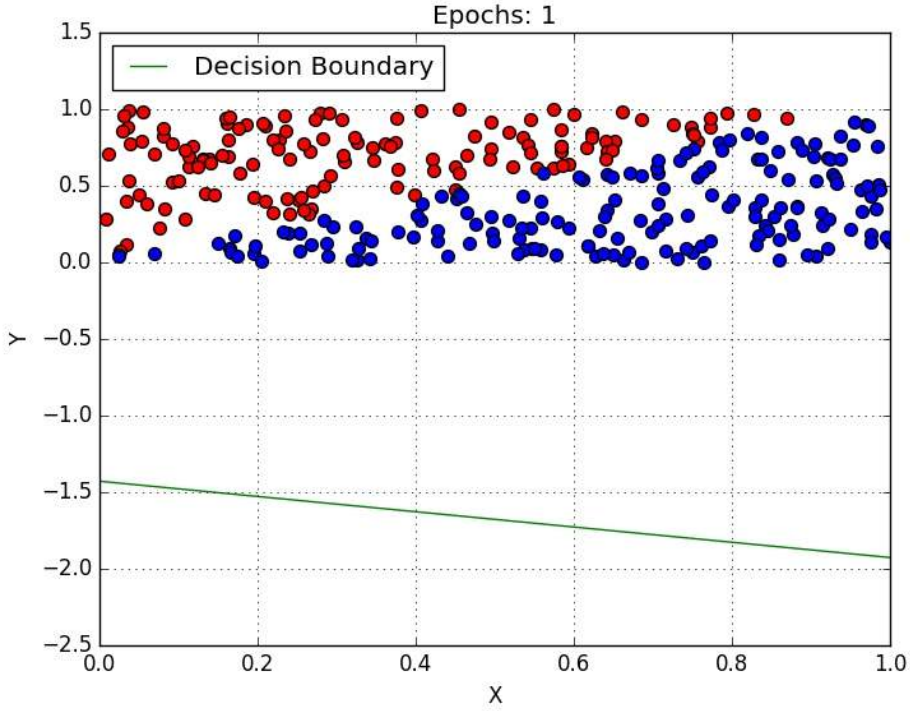
Result

```
[[4.03541657]]
```

We have predicted that a tumor with a size of 3.46mm will not be cancerous.




Logistic regression curve
costs: 0.755
y
x
epochs: 0
costs
w
b


Epochs: 1
Decision Boundary
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
Y
0.0
0.2
0.4
0.6
0.8
1.0
X

UNIT-5

NAIVE BAYES CLASSIFIER

Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly used in *text classification* that includes a high-dimensional training dataset.
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object**.
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles**.

Why is it called Naïve Bayes?

The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

- **Naïve**: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.
- **Bayes**: It is called Bayes because it depends on the principle of Bayes' Theorem.

Bayes' Theorem:

- Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
- The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier:

Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions. So to solve this problem, we need to follow the below steps:

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability.

Problem: If the weather is sunny, then the Player should play or not?

Solution: To solve this, first consider the below dataset:

	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes

Frequency table for the Weather Conditions:

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = **0.41**

So as we can see from the above calculation that **P(Yes|Sunny)>P(No|Sunny)**

Hence on a Sunny day, Player can play the game.

Advantages of Naïve Bayes Classifier:

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier:

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier:

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

- **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
- **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.
  The classifier uses the frequency of words for the predictors.
- **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

Python Implementation of the Naïve Bayes algorithm:

Now we will implement a Naive Bayes Algorithm using Python. So for this, we will use the "**user_data**" **dataset**, which we have used in our other classification model. Therefore we can easily compare the Naive Bayes model with the other models.

Steps to implement:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

1) Data Pre-processing step:

In this step, we will pre-process/prepare the data so that we can use it efficiently in our code. It is similar as we did in data-pre-processing. The code for this is given below:

```
Importing the libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('user_data.csv')
x = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In the above code, we have loaded the dataset into our program using "**dataset = pd.read_csv('user_data.csv')**. The loaded dataset is divided into training and test set, and then we have scaled the feature variable.

The output for the dataset is given as:

dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1
19	15621083	Female	48	29000	1

2) Fitting Naive Bayes to the Training Set:

After the pre-processing step, now we will fit the Naive Bayes model to the Training set. Below is the code for it:

```
1. # Fitting Naive Bayes to the Training set
2. from sklearn.naive_bayes import GaussianNB
3. classifier = GaussianNB()
4. classifier.fit(x_train, y_train)
```

In the above code, we have used the **GaussianNB classifier** to fit it to the training dataset. We can also use other classifiers as per our requirement.

Output:

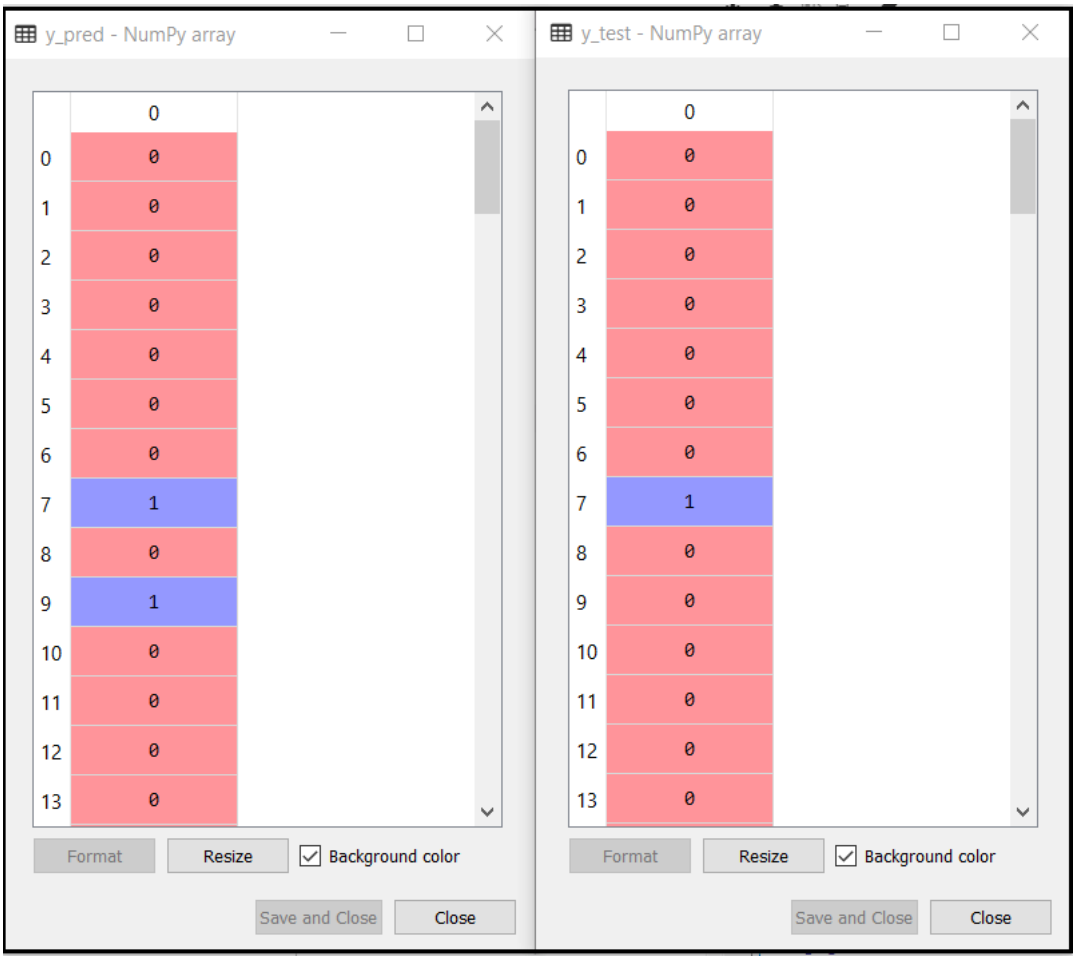
```
Out[6]: GaussianNB(priors=None, var_smoothing=1e-09)
```

3) Prediction of the test set result:

Now we will predict the test set result. For this, we will create a new predictor variable **y_pred**, and will use the predict function to make the predictions.

```
1. # Predicting the Test set results
2. y_pred = classifier.predict(x_test)
```

Output:



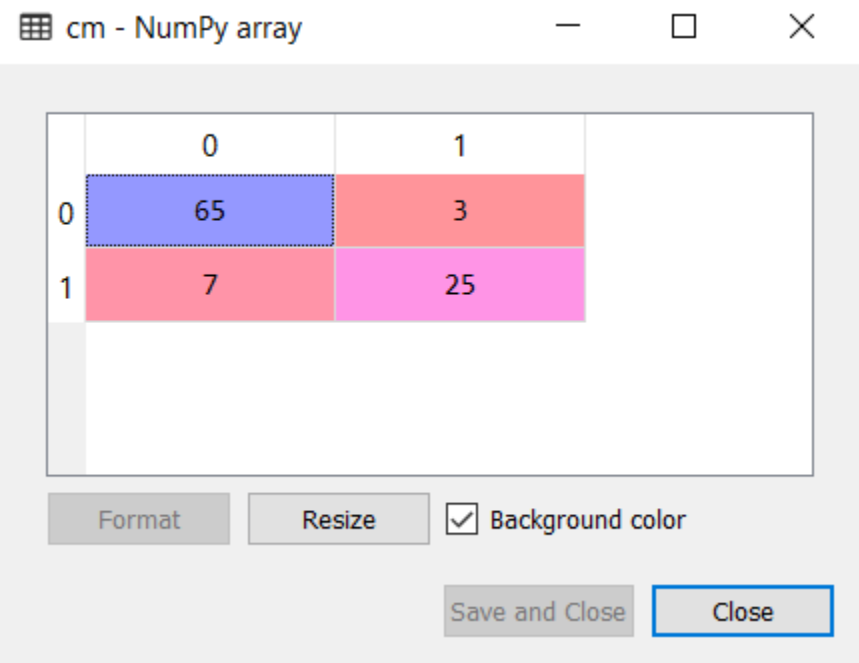
The above output shows the result for prediction vector **y_pred** and real vector y_test. We can see that some predications are different from the real values, which are the incorrect predictions.

4) Creating Confusion Matrix:

Now we will check the accuracy of the Naive Bayes classifier using the Confusion matrix. Below is the code for it:

```
1. # Making the Confusion Matrix
2. from sklearn.metrics import confusion_matrix
3. cm = confusion_matrix(y_test, y_pred)
```

Output:



As we can see in the above confusion matrix output, there are 7+3= 10 incorrect predictions, and 65+25=90 correct predictions.

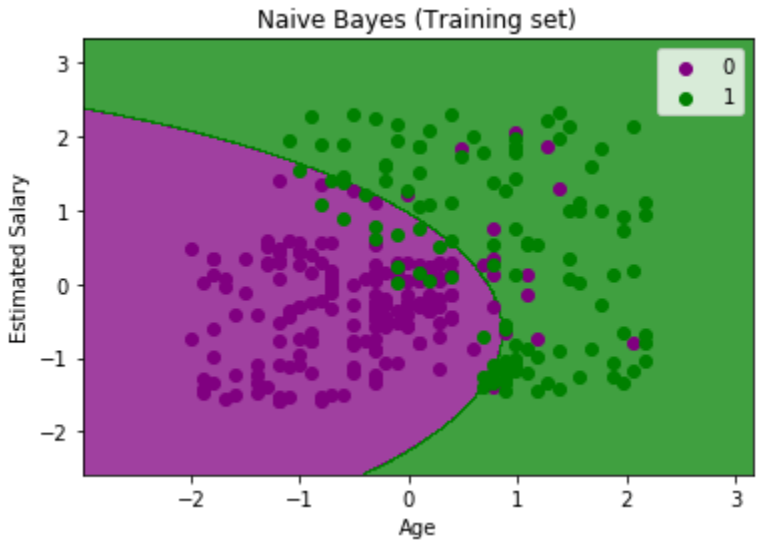
5) Visualizing the training set result:

Next we will visualize the training set result using Naïve Bayes Classifier. Below is the code for it:

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01)
,
                     nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
```

```
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
            c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

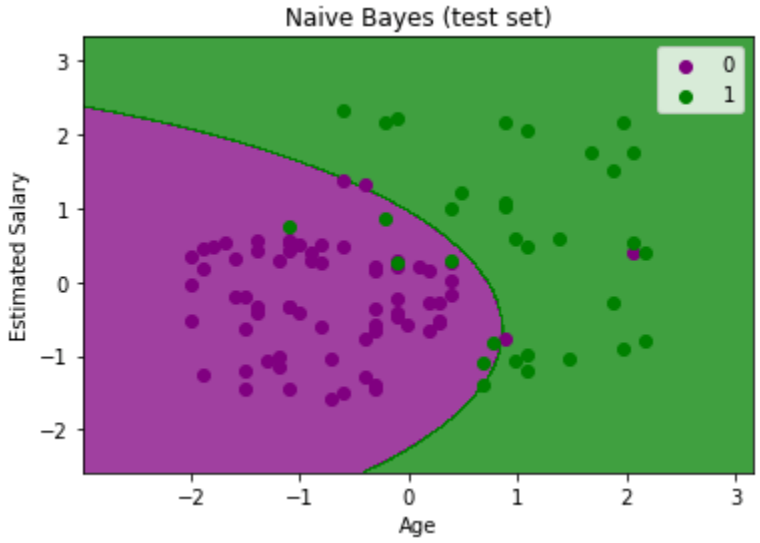


In the above output we can see that the Naïve Bayes classifier has segregated the data points with the fine boundary. It is Gaussian curve as we have used **GaussianNB** classifier in our code.

6) Visualizing the Test set result:

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01)
,
                     nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))

mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
        alpha = 0.75, cmap = ListedColormap(('purple', 'green')))
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
            c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



The above output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables. There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier.

REINFORCEMENT LEARNING

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation. Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

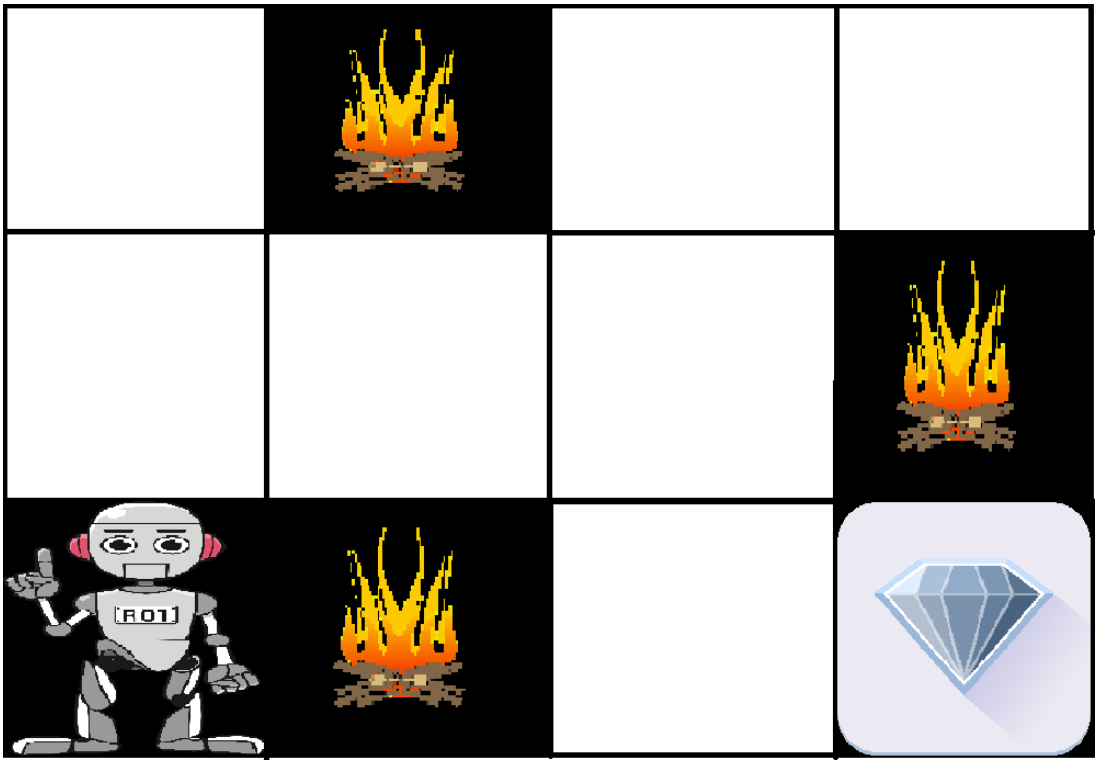
Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behavior in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning.

Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.

Reinforcement learning is an autonomous, self-teaching system that essentially learns by trial and error. It performs actions with the aim of maximizing rewards, or in other words, it is learning by doing in order to achieve the best outcomes.

Example:

The problem is as follows: We have an agent and a reward, with many hurdles in between. The agent is supposed to find the best possible path to reach the reward. The following problem explains the problem more easily.



The above image shows the robot, diamond, and fire. The goal of the robot is to get the reward that is the diamond and avoid the hurdles that are fired. The robot learns by trying all the possible paths and then choosing the path which gives him the reward with the least hurdles. Each right step will give the robot a reward and each wrong step will subtract the reward of the robot. The total reward will be calculated when it reaches the final reward that is the diamond.

Main points in Reinforcement learning –

- Input: The input should be an initial state from which the model will start
- Output: There are many possible outputs as there are a variety of solutions to a particular problem
- Training: The training is based upon the input, The model will return a state and the user will decide to reward or punish the model based on its output.
- The model keeps continues to learn.
- The best solution is decided based on the maximum reward.

Difference between Reinforcement learning and Supervised learning:

Reinforcement learning	Supervised learning
Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input	In Supervised learning, the decision is made on the initial input or the input given at the start
In Reinforcement learning decision is dependent, So we give labels to sequences of dependent decisions	In supervised learning the decisions are independent of each other so labels are given to each decision.
Example: Chess game,text summarization	Example: Object recognition,spam detetction

Types of Reinforcement:

There are two types of Reinforcement:

1. **Positive:** Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.

   Advantages of reinforcement learning are:

   - Maximizes Performance
   - Sustain Change for a long period of time
   - Too much Reinforcement can lead to an overload of states which can diminish the results
2. **Negative:** Negative Reinforcement is defined as strengthening of behavior because a negative condition is stopped or avoided.
   Advantages of reinforcement learning:
   - Increases Behavior
   - Provide defiance to a minimum standard of performance
   - It Only provides enough to meet up the minimum behavior

Elements of Reinforcement Learning

Reinforcement learning elements are as follows:

1. Policy
2. Reward function
3. Value function
4. Model of the environment

Policy: Policy defines the learning agent behavior for given time period. It is a mapping from perceived states of the environment to actions to be taken when in those states.

Reward function: Reward function is used to define a goal in a reinforcement learning problem.A reward function is a function that provides a numerical score based on the state of the environment

Value function: Value functions specify what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

Model of the environment: Models are used for planning.

Credit assignment problem: Reinforcement learning algorithms learn to generate an internal value for the intermediate states as to how good they are in leading to the goal. The learning decision maker is called the agent. The agent interacts with the environment that includes everything outside the agent.

The agent has sensors to decide on its state in the environment and takes action that modifies its state.

The reinforcement learning problem model is an agent continuously interacting with an environment. The agent and the environment interact in a sequence of time steps. At each time step t, the agent receives the state of the environment and a scalar numerical reward for the previous action, and then the agent then selects an action.

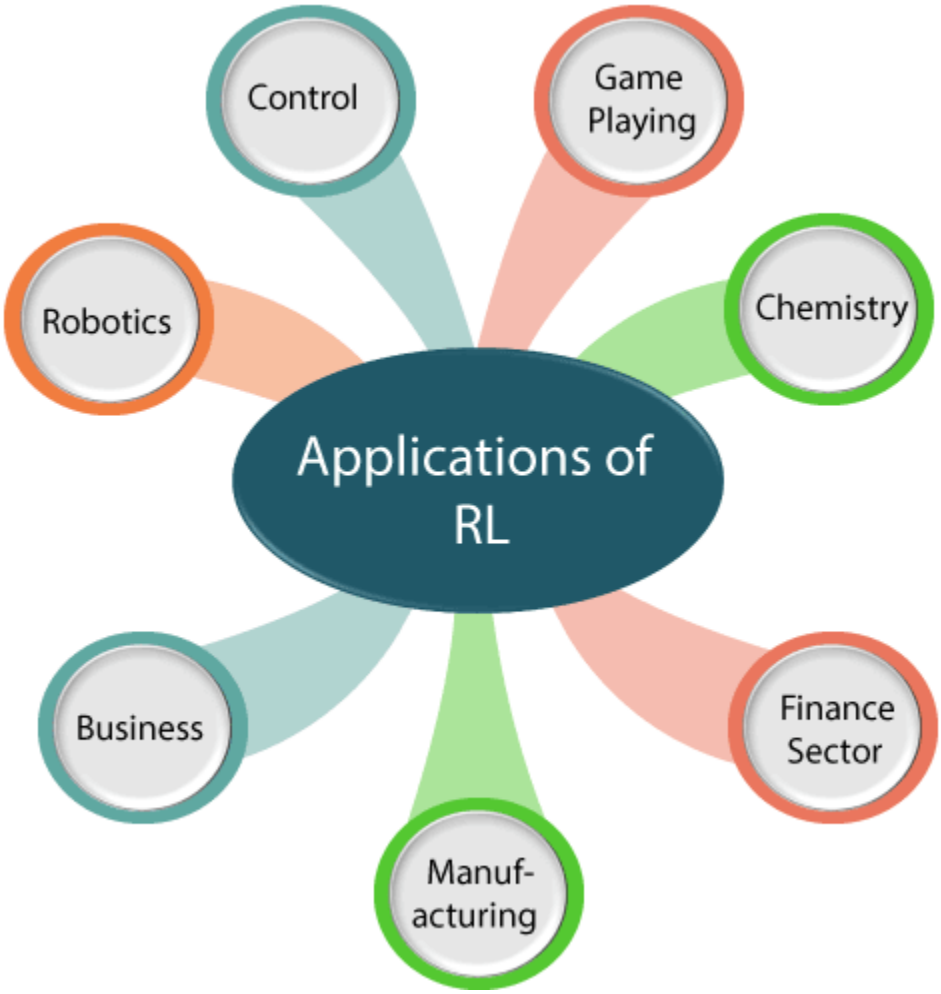
Reinforcement learning is a technique for solving Markov decision problems.

Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem.

Various Practical Applications of Reinforcement Learning –

- RL can be used in robotics for industrial automation.
- RL can be used in machine learning and data processing
- RL can be used to create training systems that provide custom instruction and materials according to the requirement of students.

Reinforcement Learning Applications



1. **Robotics:**

   RL is used in **Robot navigation, Robo-soccer, walking, juggling**, etc.

2. **Control:**

   RL can be used for **adaptive control** such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.

3. **Game Playing:**

   RL can be used in **Game playing** such as tic-tac-toe, chess, etc.

4. **Chemistry:**

   RL can be used for optimizing the chemical reactions.

5. **Business:**

   RL is now used for business strategy planning.

6. **Manufacturing:**

In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.

7. **Finance Sector:**

The RL is currently used in the finance sector for evaluating trading strategies.

Advantages and Disadvantages of Reinforcement Learning

Advantages of Reinforcement learning

1. Reinforcement learning can be used to solve very complex problems that cannot be solved by conventional techniques.

2. The model can correct the errors that occurred during the training process.

3. In RL, training data is obtained via the direct interaction of the agent with the environment

4. Reinforcement learning can handle environments that are non-deterministic, meaning that the outcomes of actions are not always predictable. This is useful in real-world applications where the environment may change over time or is uncertain.

5. Reinforcement learning can be used to solve a wide range of problems, including those that involve decision making, control, and optimization.

6. Reinforcement learning is a flexible approach that can be combined with other machine learning techniques, such as deep learning, to improve performance.

Disadvantages of Reinforcement learning

1. Reinforcement learning is not preferable to use for solving simple problems.

2. Reinforcement learning needs a lot of data and a lot of computation

3. Reinforcement learning is highly dependent on the quality of the reward function. If the reward function is poorly designed, the agent may not learn the desired behavior.

4. Reinforcement learning can be difficult to debug and interpret. It is not always clear why the agent is behaving in a certain way, which can make it difficult to diagnose and fix problems.

SEMI SUPERVISED LEARNING

Semi-Supervised learning is a type of Machine Learning algorithm that represents the intermediate ground between Supervised and Unsupervised learning algorithms. It uses the combination of labeled and unlabeled datasets during the training period.

Semi-supervised learning is an important category that lies between the Supervised and Unsupervised machine learning. Although Semi-supervised learning is the middle ground between supervised and unsupervised learning and operates on the data that consists of a few labels, it mostly consists of unlabeled data. As labels are costly, but for the corporate purpose, it may have few labels.

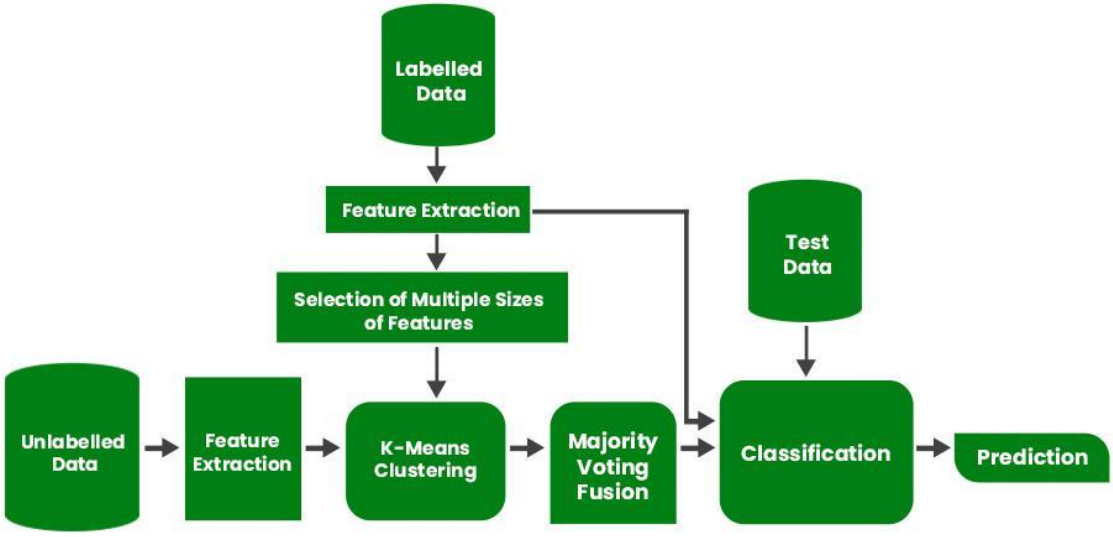
The basic disadvantage of supervised learning is that it requires hand-labeling by ML specialists or data scientists, and it also requires a high cost to process. Further unsupervised learning also has a limited spectrum for its applications. **To overcome these drawbacks of supervised learning and unsupervised learning algorithms, the concept of Semi-supervised learning is introduced**. In this algorithm, training data is a combination of both labeled and unlabeled data. However, labeled data exists with a very small amount while it consists of a huge amount of unlabeled data. Initially, similar data is clustered along with an unsupervised learning algorithm, and further, it helps to label the unlabeled data into labeled data. It is why label data is a comparatively, more expensive acquisition than unlabeled data.

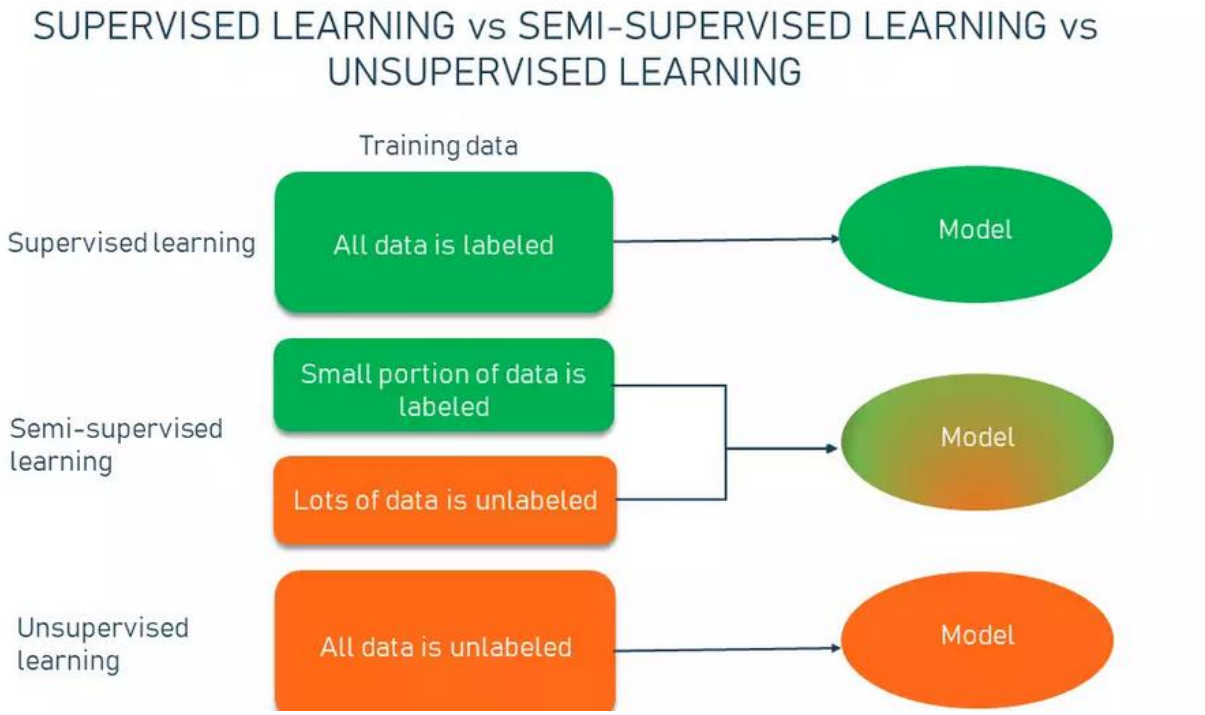
We can imagine these algorithms with an example. Supervised learning is where a student is under the supervision of an instructor at home and college. Further, if that student is self-analyzing the same concept without any help from the instructor, it comes under unsupervised learning. Under semi-supervised learning, the student has to revise itself after analyzing the same concept under the guidance of an instructor at college.

Assumptions followed by Semi-Supervised Learning

To work with the unlabeled dataset, there must be a relationship between the objects. To understand this, semi-supervised learning uses any of the following assumptions:

- **Continuity Assumption:**
  As per the continuity assumption, the objects near each other tend to share the same group or label. This assumption is also used in supervised learning, and the datasets are separated by the decision boundaries. But in semi-supervised, the decision boundaries are added with the smoothness assumption in low-density boundaries.
- **Cluster assumptions-** In this assumption, data are divided into different discrete clusters. Further, the points in the same cluster share the output label.
- **Manifold assumptions-** This assumption helps to use distances and densities, and this data lie on a manifold of fewer dimensions than input space.
- The dimensional data are created by a process that has less degree of freedom and may be hard to model directly. **(This assumption becomes practical if high).**





Working of Semi-Supervised Learning

Semi-supervised learning uses pseudo labeling to train the model with less labeled training data than supervised learning. The process can combine various neural network models and training ways. The whole working of semi-supervised learning is explained in the below points:

- Firstly, it trains the model with less amount of training data similar to the supervised learning models. The training continues until the model gives accurate results.
- The algorithms use the unlabeled dataset with pseudo labels in the next step, and now the result may not be accurate.
- Now, the labels from labeled training data and pseudo labels data are linked together.
- The input data in labeled training data and unlabeled training data are also linked.

- In the end, again train the model with the new combined input as did in the first step. It will reduce errors and improve the accuracy of the model.

Difference between Semi-supervised and Reinforcement Learning.

Reinforcement learning is different from semi-supervised learning, as it works with rewards and feedback. ***Reinforcement learning aims to maximize the rewards by their hit and trial actions, whereas in semi-supervised learning, we train the model with a less labeled dataset.***

Real-world applications of Semi-supervised Learning-

Semi-supervised learning models are becoming more popular in the industries. Some of the main applications are as follows.

- **Speech Analysis-** It is the most classic example of semi-supervised learning applications. Since, labeling the audio data is the most impassable task that requires many human resources, this problem can be naturally overcome with the help of applying SSL in a Semi-supervised learning model.
- **Web content classification-** However, this is very critical and impossible to label each page on the internet because it needs mode human intervention. Still, this problem can be reduced through Semi-Supervised learning algorithms.
  Further, Google also uses semi-supervised learning algorithms to rank a webpage for a given query.
- **Protein sequence classification-** DNA strands are larger, they require active human intervention. So, the rise of the Semi-supervised model has been proximate in this field.
- **Text document classifier-** As we know, it would be very unfeasible to find a large amount of labeled text data, so semi-supervised learning is an ideal model to overcome this.

How semi-supervised learning works

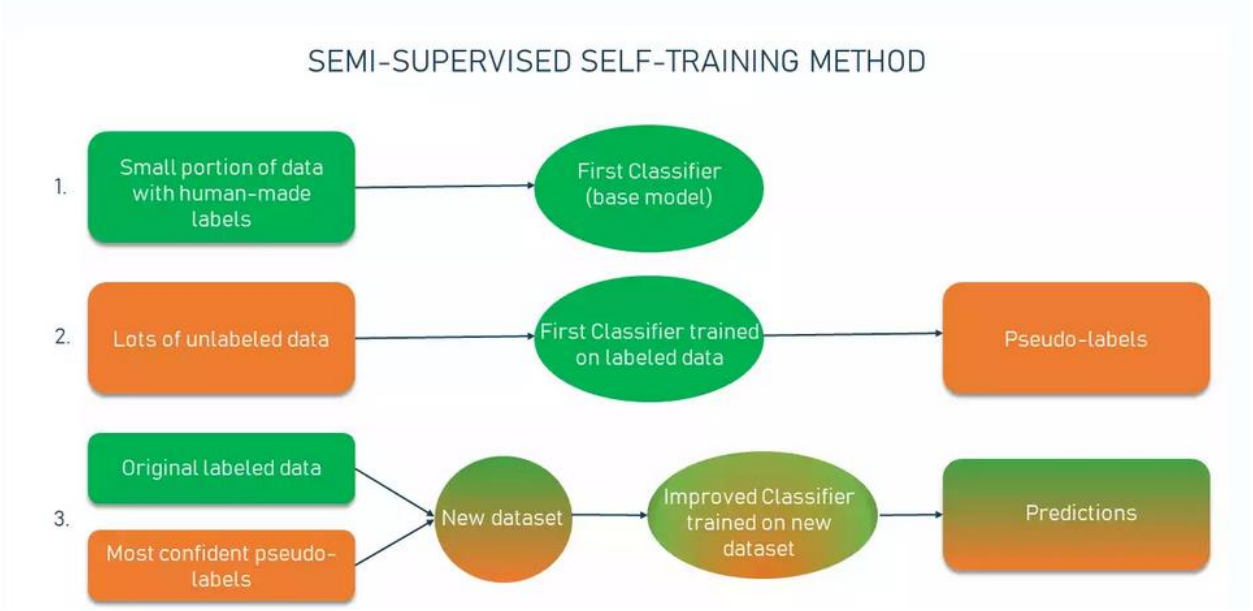
Imagine, you have collected a large set of unlabeled data that you want to train a model on. Manual labeling of all this information will probably cost you a fortune, besides taking months to complete the annotations. That's when the semi-supervised machine learning method comes to the rescue.

The working principle is quite simple. Instead of adding tags to the entire dataset, you go through and hand-label just a small part of the data and use it to train a model, which then is applied to the ocean of unlabeled data.

Self-training

One of the simplest examples of semi-supervised learning, in general, is self-training.

Self-training is the procedure in which you can take any supervised method for classification or regression and modify it to work in a semi-supervised manner, taking advantage of labeled and unlabeled data. The standard workflow is as follows.

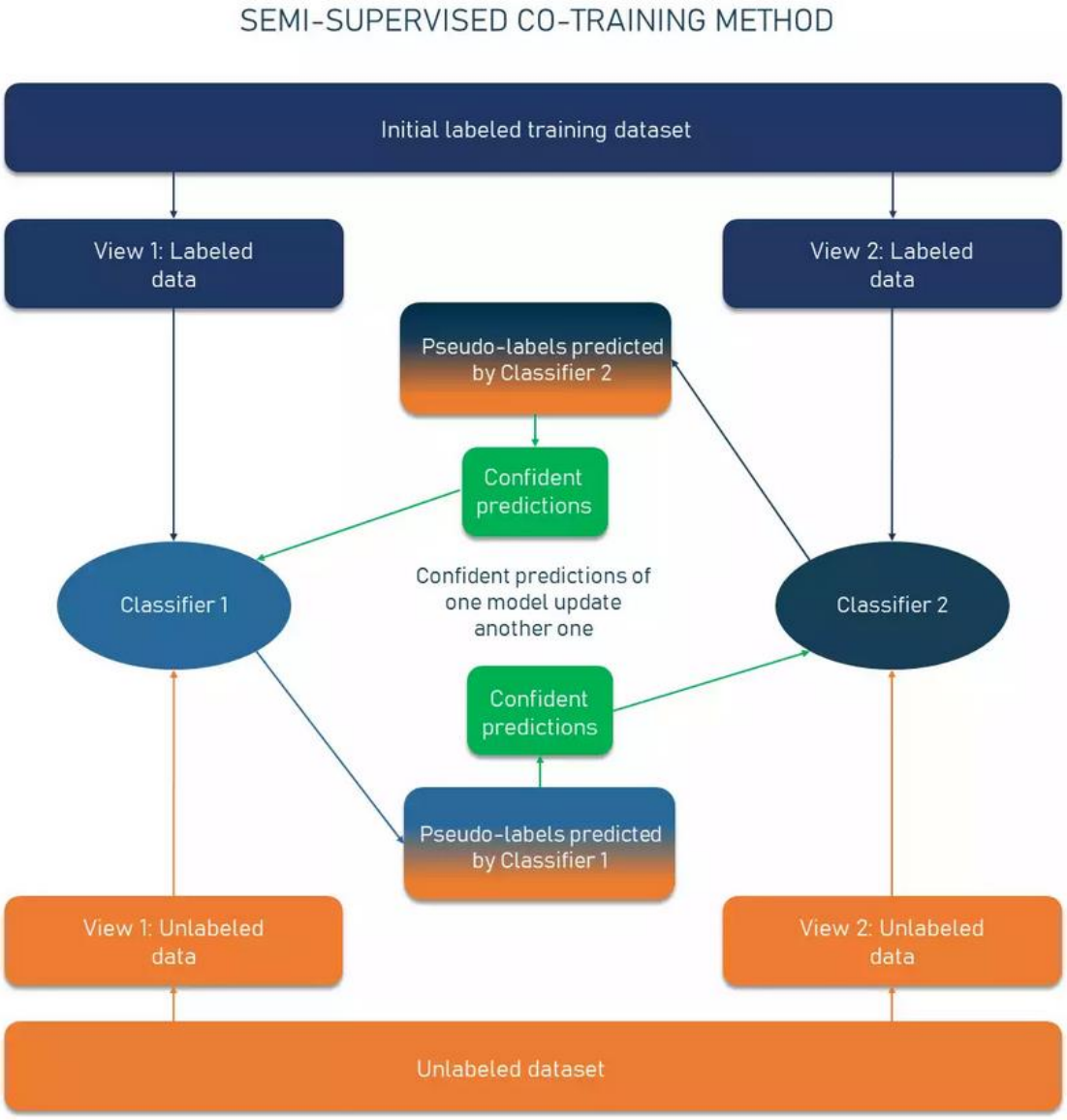


Co-training

Derived from the self-training approach and being its improved version, **co-training** is another semi-supervised learning technique used when only a small portion of labeled data is available. Unlike the typical process, co-training trains two individual classifiers based on two *views* of data.

The views are basically different sets of features that provide additional information about each instance, meaning they are independent given the class. Also, each view is sufficient — the class of sample data can be accurately predicted from each set of features alone.

The original co-training research paper claims that the approach can be successfully used, for example, for web content classification tasks. The description of each web page can be divided into two views: one with words occurring on that page and the other with anchor words in the link leading to it.



K-MEANS CLUSTERING

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

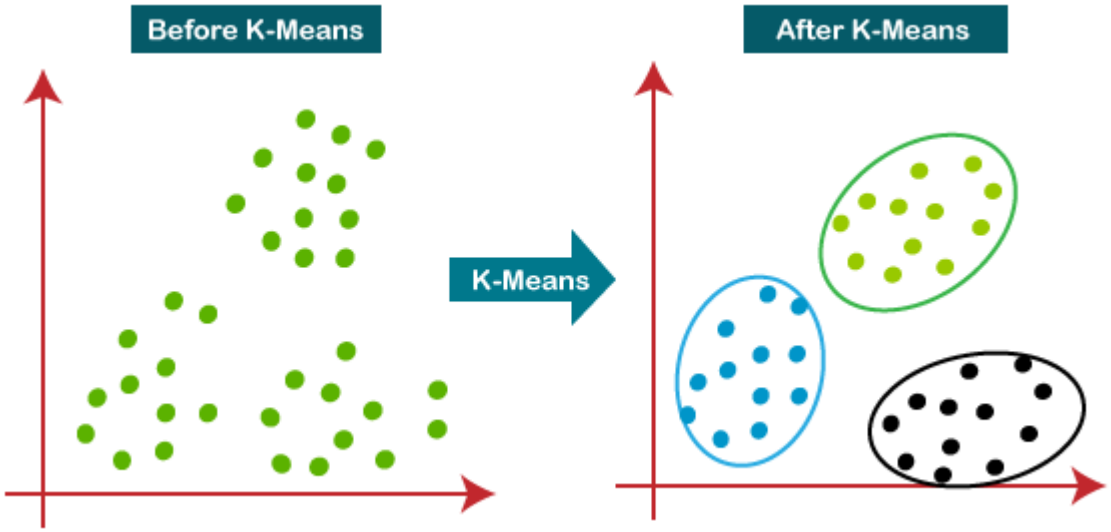
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

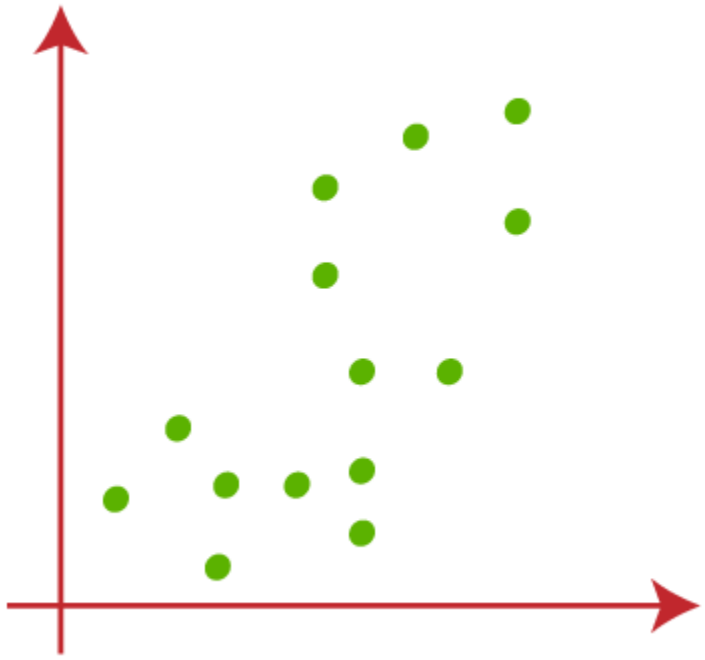
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

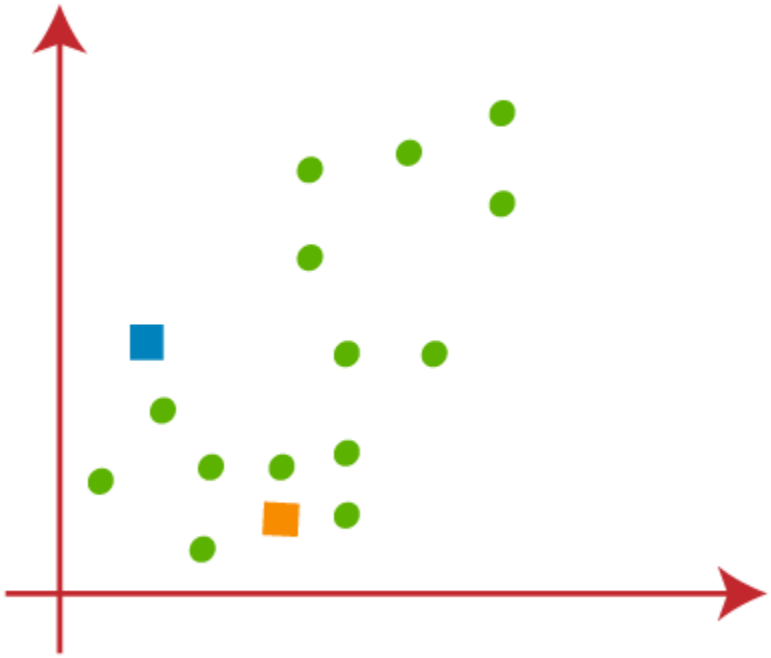
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

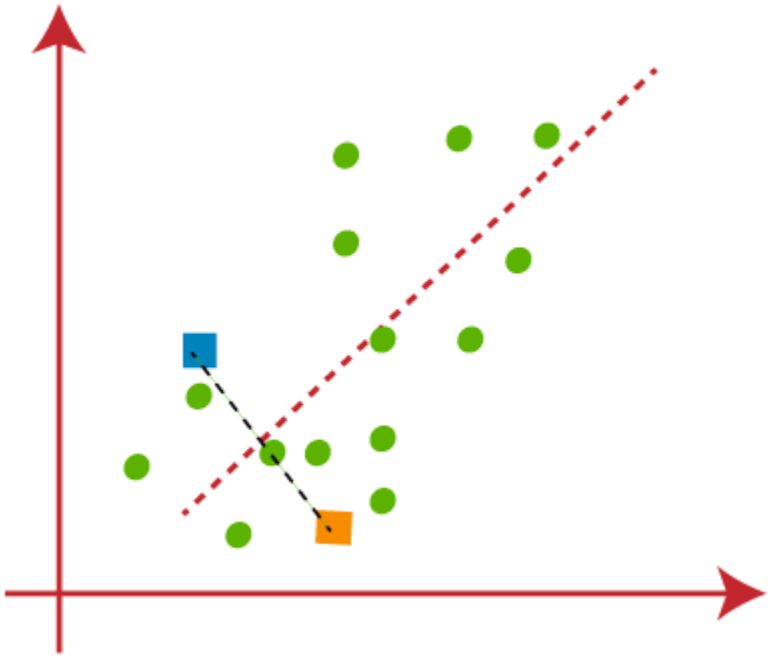


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two

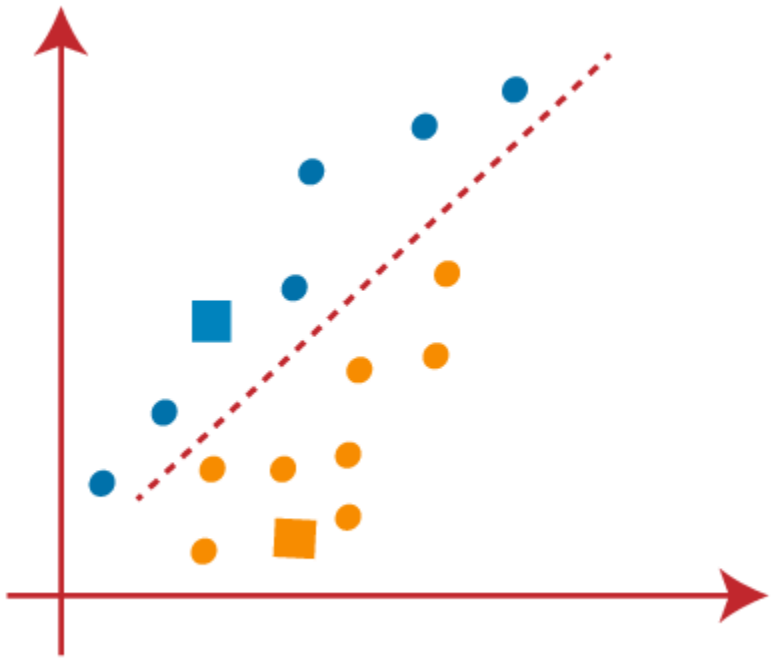
points as k points, which are not the part of our dataset. Consider the below image:



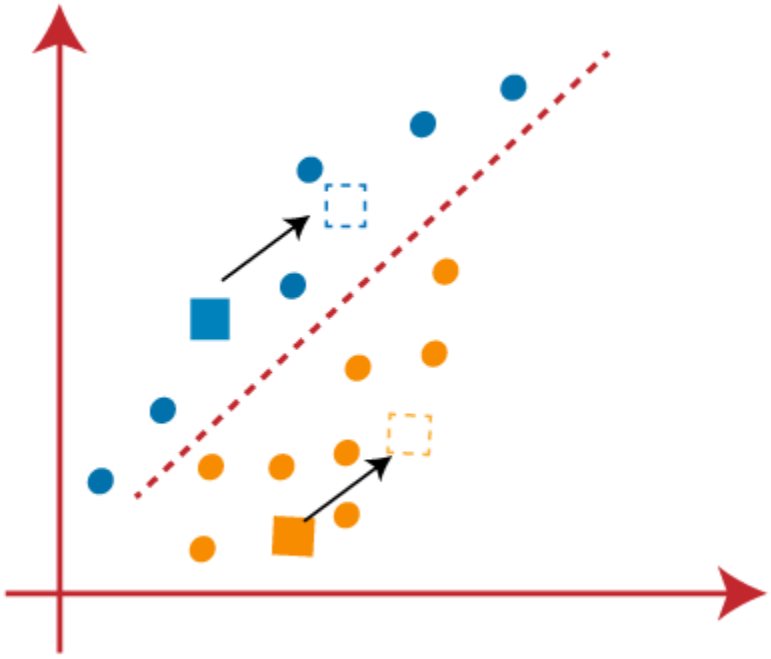
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



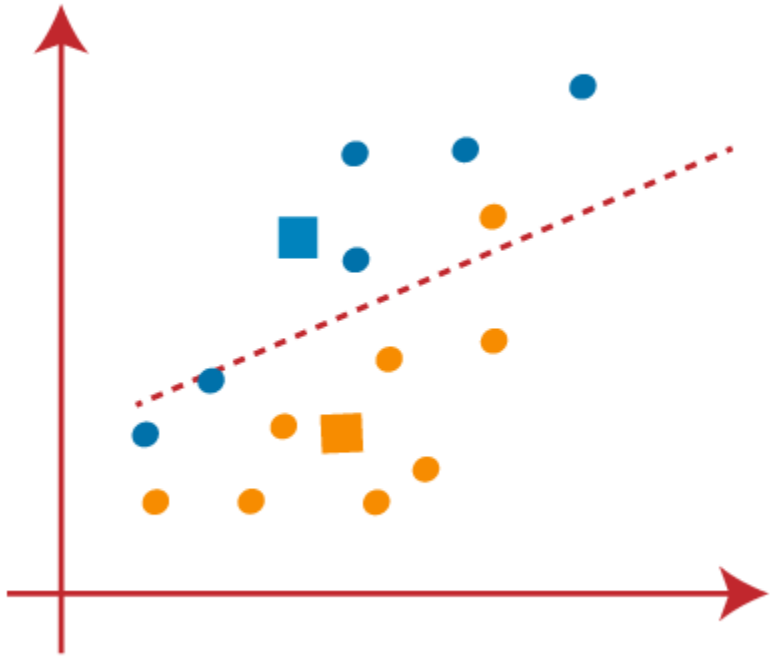
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



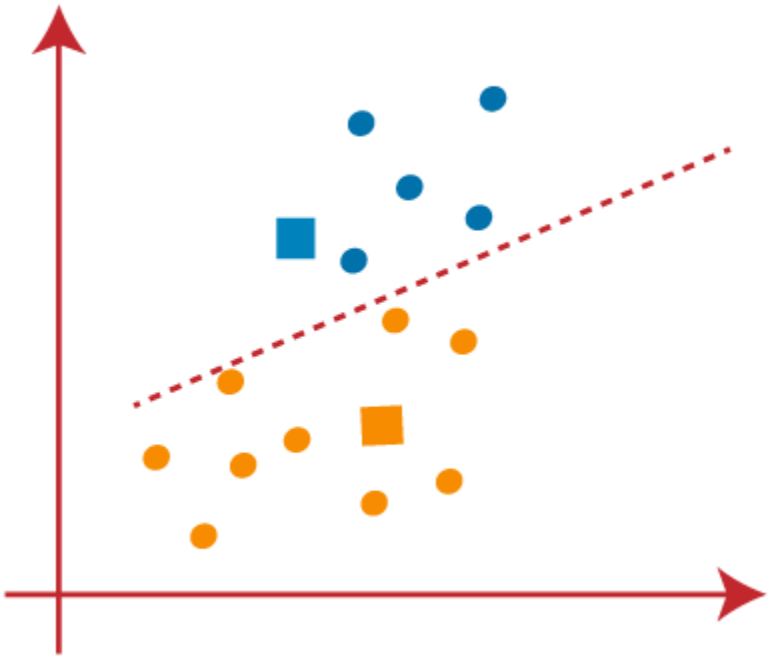
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:



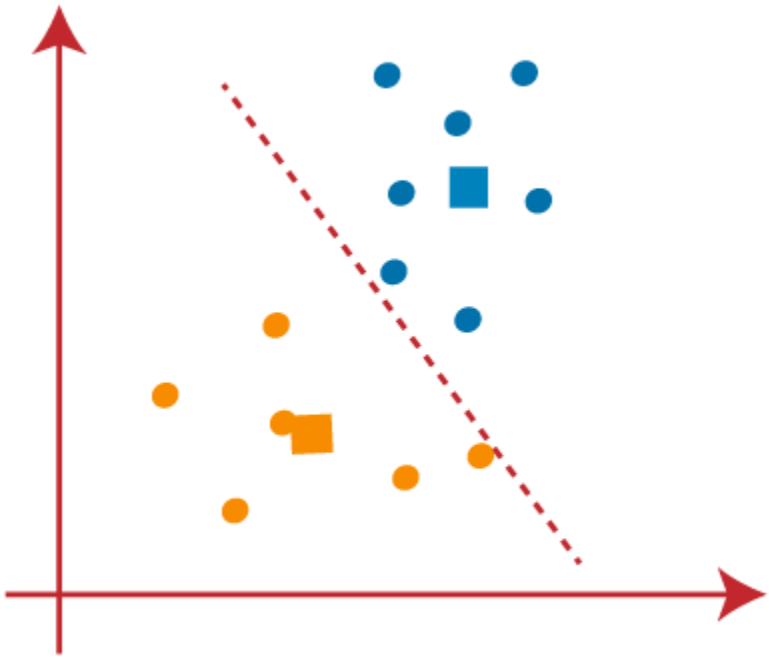
From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



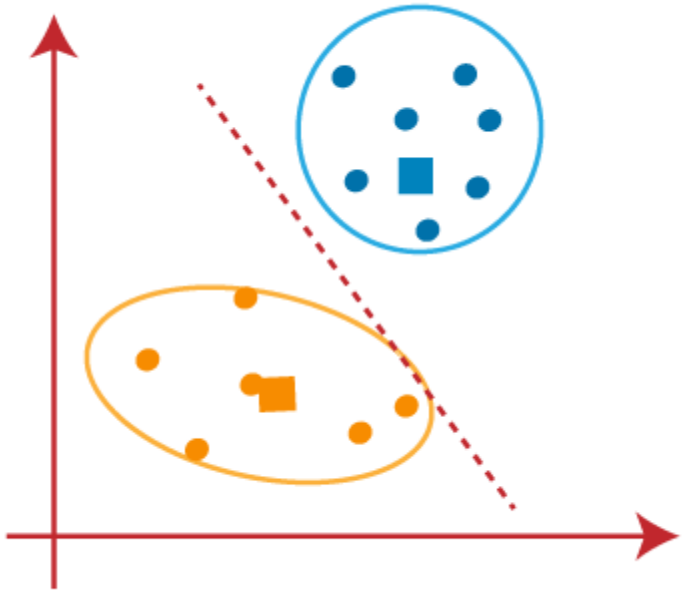
As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

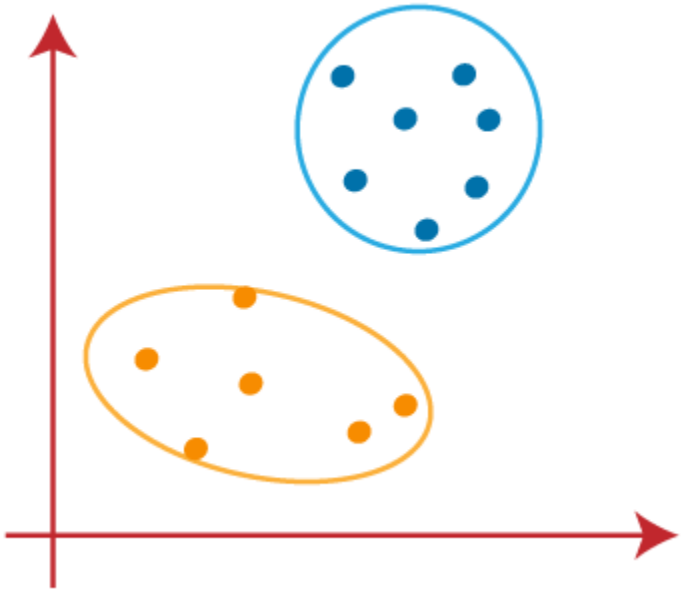
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{Pi \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{Pi \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

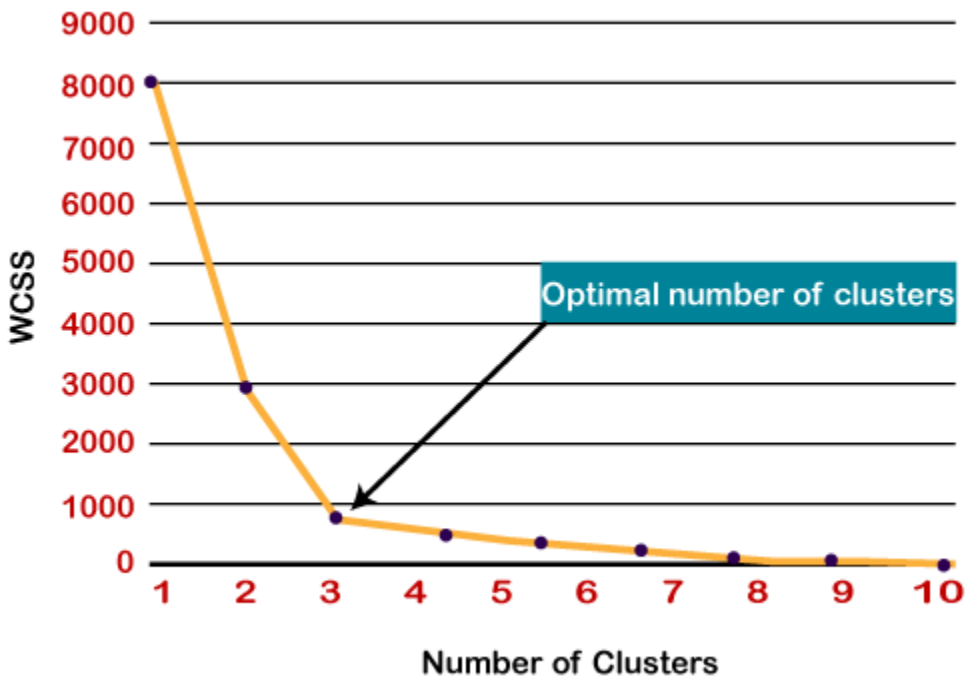
$\sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Python Implementation of K-means Clustering Algorithm

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id, Gender, Age, Annual Income ($), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Step-1: Data pre-processing Step

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the clustering problem, it will be different from other models. Let's discuss it:

- **Importing Libraries**
  As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

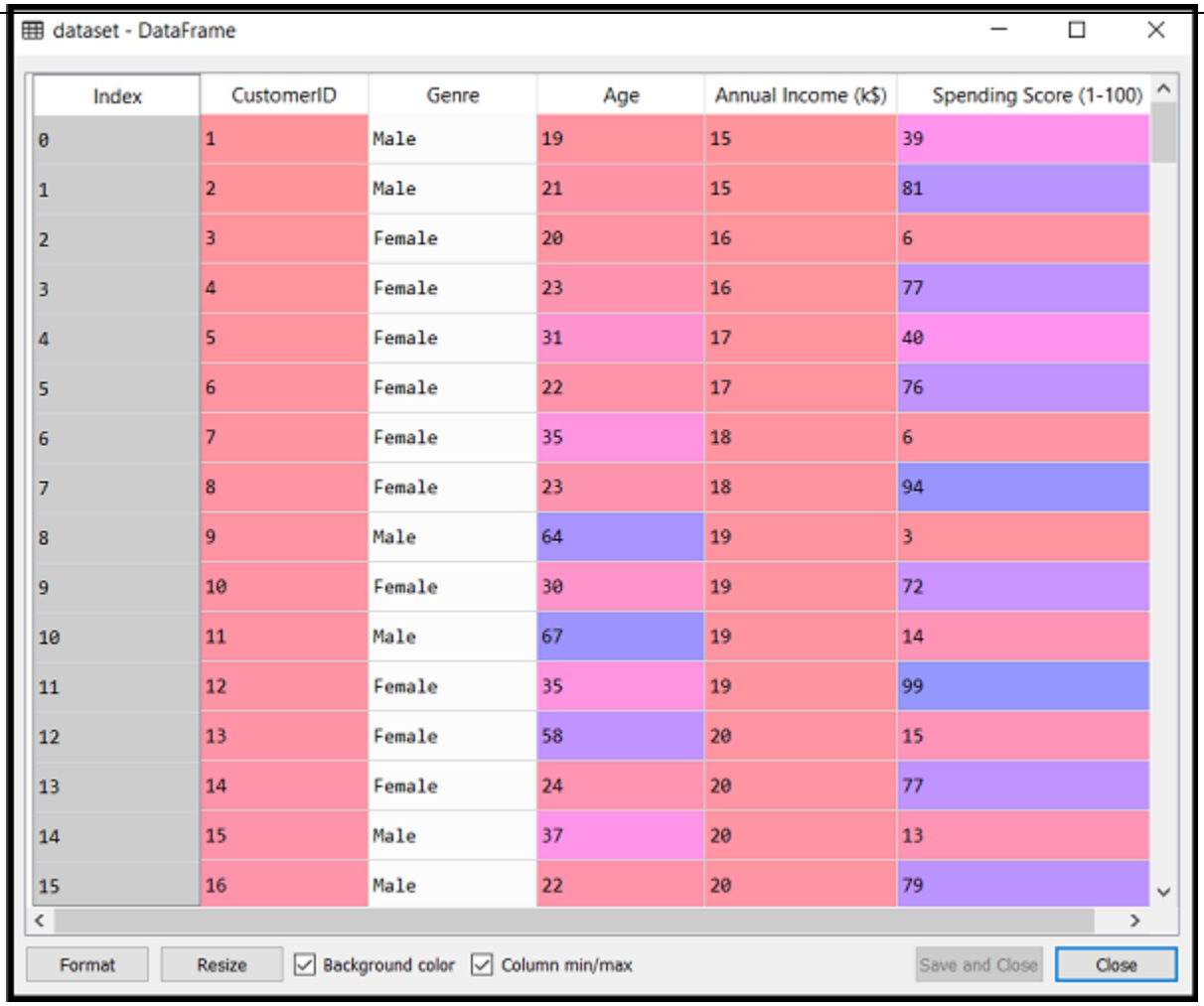
```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
```

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

- **Importing the Dataset:**
  Next, we will import the dataset that we need to use. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

```
1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers_data.csv')
```

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:



Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
1. x = dataset.iloc[:, [3, 4]].values
```

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
```

```
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
   kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

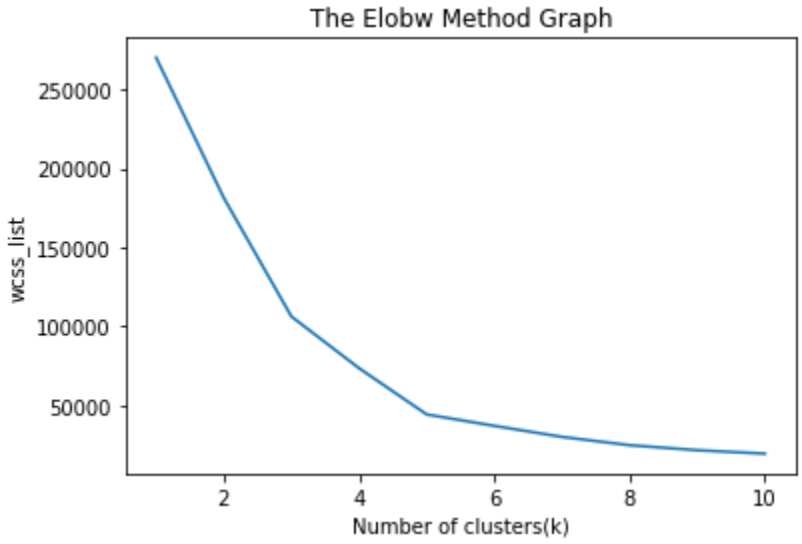
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

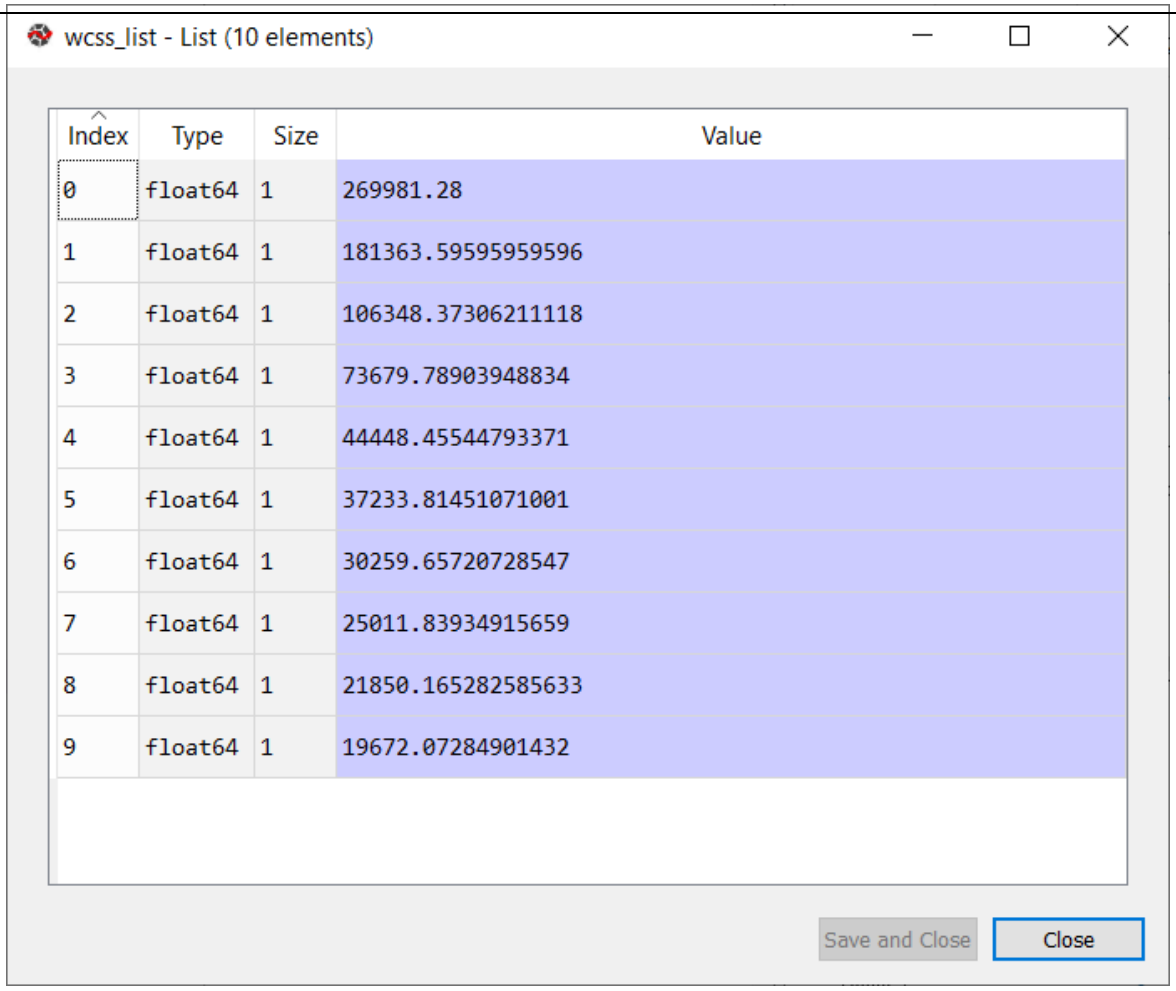
After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10^{th} value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5. So the number of clusters here will be 5.**



Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

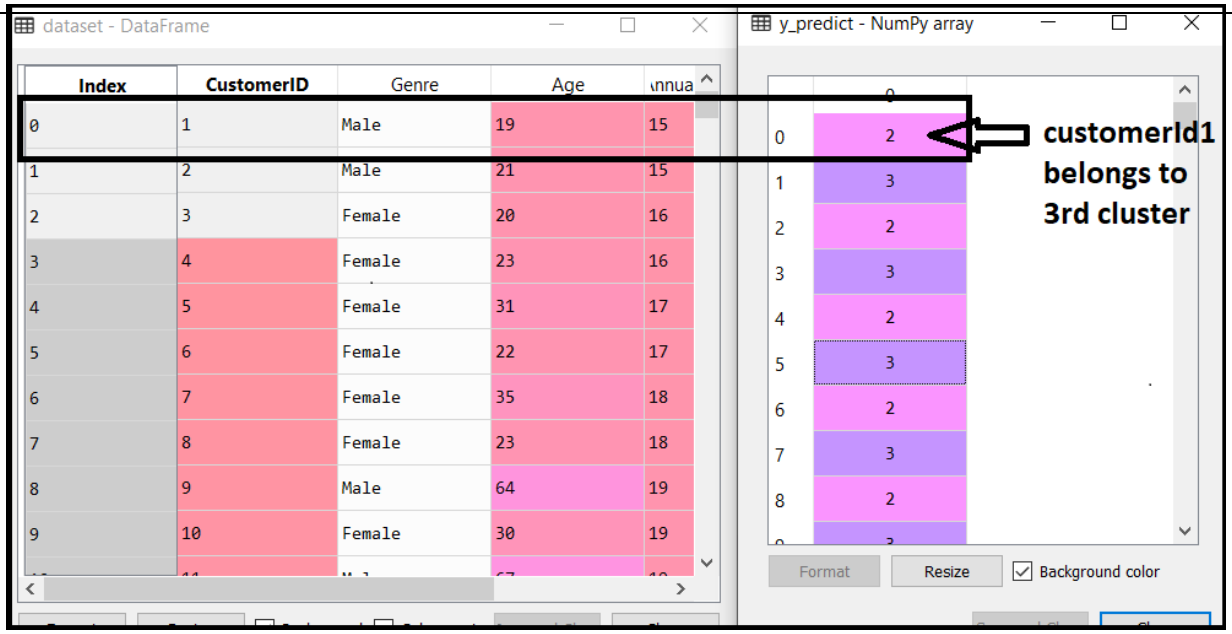
To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:

```
1. #training the K-means model on a dataset
2. kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
3. y_predict= kmeans.fit_predict(x)
```

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y_predict** to train the model.

By executing the above lines of code, we will get the y_predict variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of y_predict with our original dataset. Consider the below image:



From the above image, we can now relate that the CustomerID 1 belongs to a cluster

3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

Step-4: Visualizing the Clusters

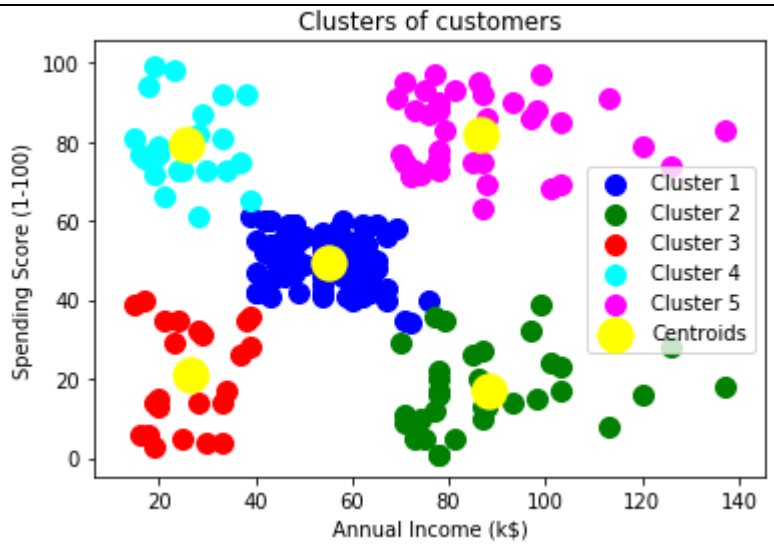
The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cl
uster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
 #for second cluster
mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cl
uster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
#for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fi
fth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow'
, label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the mtp.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.

Output:



The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as sensible.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as target, and these customers can be the most profitable customers for the mall owner.

POLYNOMIAL REGRESSION

ML Polynomial Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

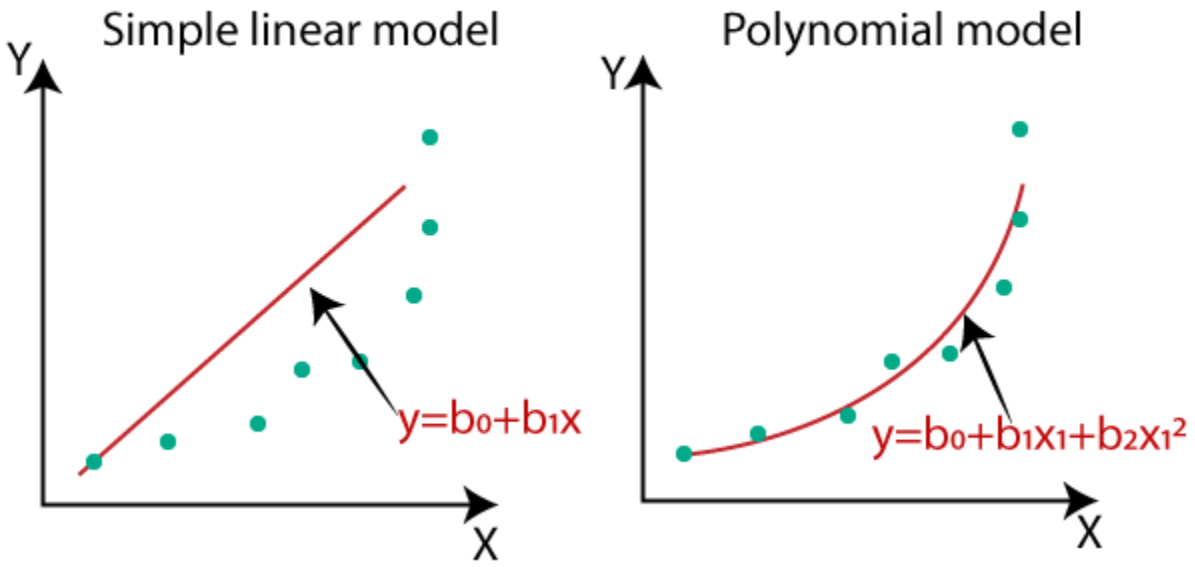
$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+..... b_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence,** ***"In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."***

Need for Polynomial Regression:

The need of Polynomial Regression in ML can be understood in the below points:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



- In the above image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.
- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

Equation of the Polynomial Regression Model:

Simple Linear Regression equation: $y = b_0+b_1x$ **.....(a)**

Multiple Linear Regression equation: $y= b_0+b_1x+ b_2x_2+ b_3x_3+....+ b_nx_n$ **.....(b)**

Polynomial Regression equation: $y= b_0+b_1x + b_2x^2+ b_3x^3+....+ b_nx^n$ **.....(c)**

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

Implementation of Polynomial Regression using Python:

Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

Problem Description: There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff. So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a **non-linear relationship between the Position levels and the salaries**. Our goal is to build a **Bluffing detector regression** model, so HR can hire an honest candidate. Below are the steps to build such a model.

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Steps for Polynomial Regression:

The main steps involved in Polynomial Regression are given below:

- Data Pre-processing
- Build a Linear Regression model and fit it to the dataset
- Build a Polynomial Regression model and fit it to the dataset
- Visualize the result for Linear Regression and Polynomial Regression model.
- Predicting the output.

Data Pre-processing Step:

The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:

- The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.

- In this model, we want very accurate predictions for salary, so the model should have enough information.

The code for pre-processing step is given below:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Position_Salaries.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

Explanation:

- In the above lines of code, we have imported the important Python libraries to import dataset and operate on it.
- Next, we have imported the dataset '**Position_Salaries.csv**', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).
- After that, we have extracted the dependent(Y) and independent variable(X) from the dataset. For x-variable, we have taken parameters as [:,1:2], because we want 1 index(levels), and included :2 to make it as a matrix.

Output:

By executing the above code, we can read our dataset as:

Index	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

As we can see in the above output, there are three columns present (Positions, Levels, and Salaries). But we are only considering two columns because Positions are equivalent to the levels or may be seen as the encoded form of Positions.

Here we will predict the output for level **6.5** because the candidate has 4+ years' experience as a regional manager, so he must be somewhere between levels 7 and 6.

Building the Linear regression model:

Now, we will build and fit the Linear regression model to the dataset. In building polynomial regression, we will take the Linear regression model as reference and compare both the results. The code is given below:

```
1. #Fitting the Linear Regression to the dataset
2. from sklearn.linear_model import LinearRegression
3. lin_regs= LinearRegression()
4. lin_regs.fit(x,y)
```

In the above code, we have created the Simple Linear model using **lin_regs** object of **LinearRegression** class and fitted it to the dataset variables (x and y).

Output:

```
Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

Building the Polynomial regression model:

Now we will build the Polynomial Regression model, but it will be a little different from the Simple Linear model. Because here we will use **PolynomialFeatures** class of **preprocessing** library. We are using this class to add some extra features to our dataset.

```
1.  #Fitting the Polynomial regression to the dataset
2.  from sklearn.preprocessing import PolynomialFeatures
3.  poly_regs= PolynomialFeatures(degree= 2)
4.  x_poly= poly_regs.fit_transform(x)
5.  lin_reg_2 =LinearRegression()
6.  lin_reg_2.fit(x_poly, y)
```

In the above lines of code, we have used **poly_regs.fit_transform(x)**, because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model. The parameter value(degree= 2) depends on our choice. We can choose it according to our Polynomial features.

After executing the code, we will get another matrix **x_poly**, which can be seen under the variable explorer option:

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25
5	1	6	36
6	1	7	49
7	1	8	64
8	1	9	81
9	1	10	100

Next, we have used another LinearRegression object, namely **lin_reg_2**, to fit our **x_poly** vector to the linear model.

Output:

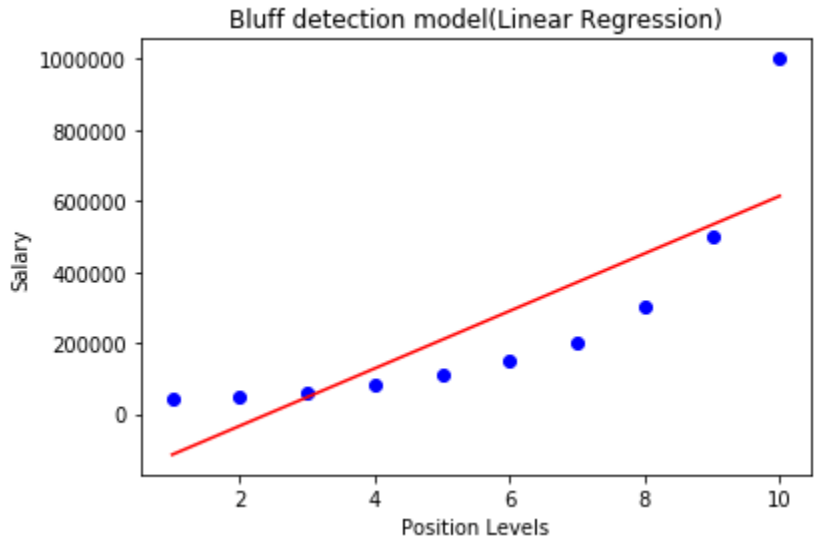
```
Out[11]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

Visualizing the result for Linear regression:

Now we will visualize the result for Linear regression model as we did in Simple Linear Regression. Below is the code for it:

```
1. #Visulaizing the result for Linear Regression model
2. mtp.scatter(x,y,color="blue")
3. mtp.plot(x,lin_regs.predict(x), color="red")
4. mtp.title("Bluff detection model(Linear Regression)")
5. mtp.xlabel("Position Levels")
6. mtp.ylabel("Salary")
7. mtp.show()
```

Output:



In the above output image, we can clearly see that the regression line is so far from the datasets. Predictions are in a red straight line, and blue points are actual values. If we consider this output to predict the value of CEO, it will give a salary of approx. 600000$, which is far away from the real value.

So we need a curved model to fit the dataset other than a straight line.

Visualizing the result for Polynomial Regression

Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.

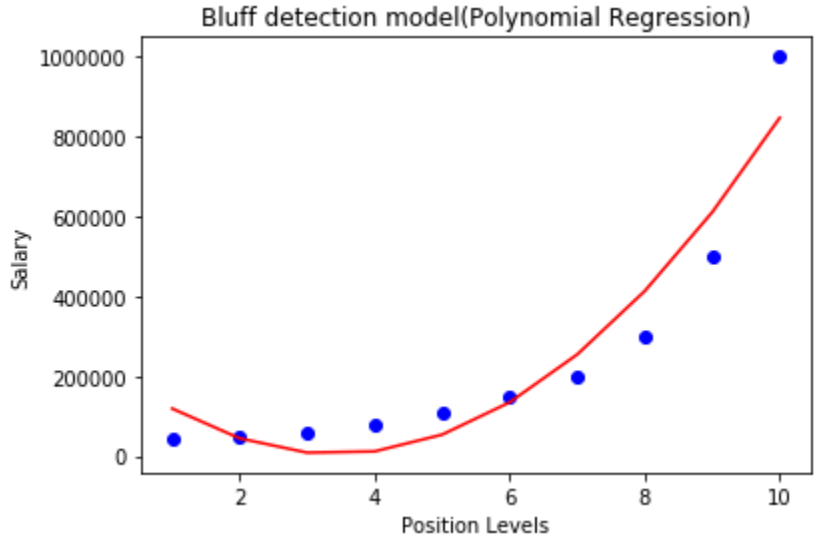
Code for this is given below:

```
1. #Visulaizing the result for Polynomial Regression
2. mtp.scatter(x,y,color="blue")
3. mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
4. mtp.title("Bluff detection model(Polynomial Regression)")
5. mtp.xlabel("Position Levels")
```

6. mtp.ylabel("Salary")
7. mtp.show()

In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x), instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.

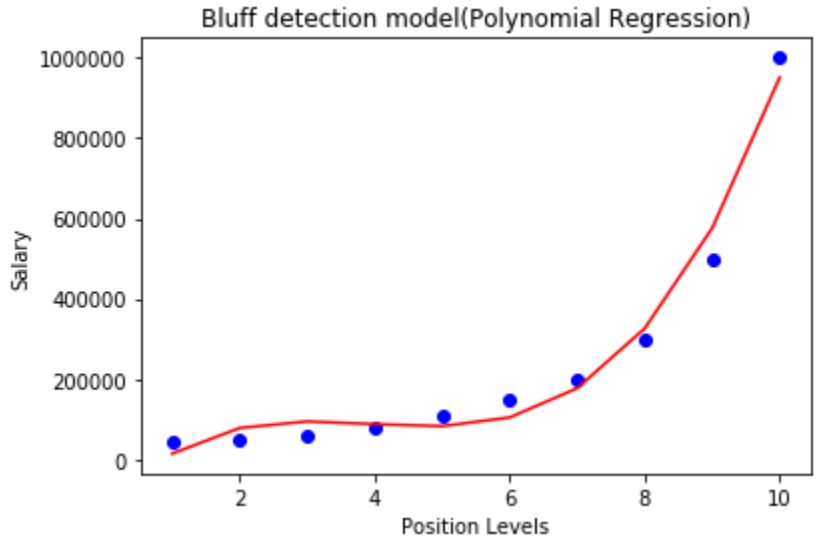
Output:



As we can see in the above output image, the predictions are close to the real values. The above plot will vary as we will change the degree.

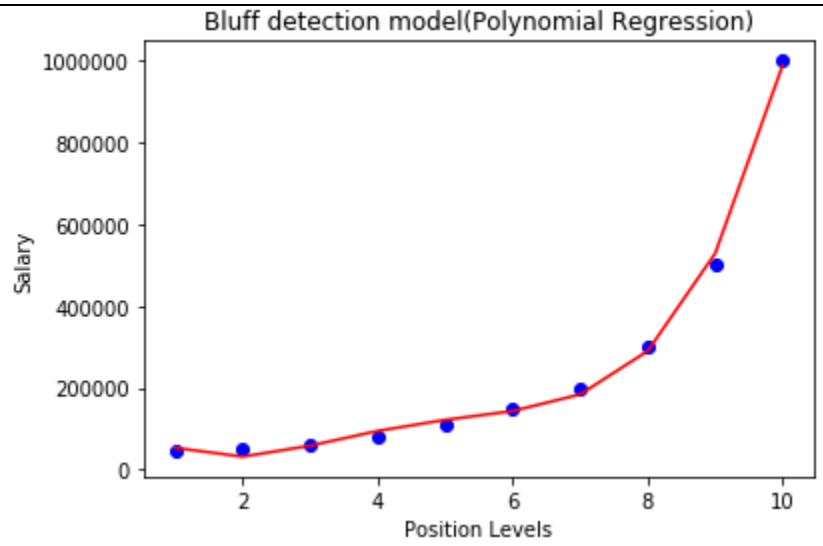
For degree= 3:

If we change the degree=3, then we will give a more accurate plot, as shown in the below image.



SO as we can see here in the above output image, the predicted salary for level 6.5 is near to 170K$-190k$, which seems that future employee is saying the truth about his salary.

Degree= 4: Let's again change the degree to 4, and now will get the most accurate plot. Hence we can get more accurate results by increasing the degree of Polynomial.



Predicting the final result with the Linear Regression model:

Now, we will predict the final output using the Linear regression model to see whether an employee is saying truth or bluff. So, for this, we will use the **predict()** method and will pass the value 6.5. Below is the code for it:

```
1. lin_pred = lin_regs.predict([[6.5]])
2. print(lin_pred)
```

Output:

```
[330378.78787879]
```

Predicting the final result with the Polynomial Regression model:

Now, we will predict the final output using the Polynomial Regression model to compare with Linear model. Below is the code for it:

```
1. poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
2. print(poly_pred)
```

Output:

```
[158862.45265153]
```

As we can see, the predicted output for the Polynomial Regression is [158862.45265153], which is much closer to real value hence, we can say that future employee is saying true.

MULTIPLE LINEAR REGRESSION

Multiple Linear Regression attempts to model the relationship between two or more features and a response by fitting a linear equation to observed data. The steps to perform multiple linear Regression are almost similar to that of simple linear Regression. The Difference Lies in the evaluation. We can use it to find out which factor has the highest impact on the predicted output and how different variables relate to each other.

Here : **Y = b0 + b1 * x1 + b2 * x2 + b3 * x3 + …… bn * xn**
Y = Dependent variable and x1, x2, x3, …… xn = multiple independent variables

Assumption of Regression Model :

- **Linearity:** The relationship between dependent and independent variables should be linear.
- **Homoscedasticity:** Constant variance of the errors should be maintained.
- **Multivariate normality:** Multiple Regression assumes that the residuals are normally distributed.
- **Lack of Multicollinearity:** It is assumed that there is little or no multicollinearity in the data.

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on **two or more** variables.

Take a look at the data set below, it contains some information about cars.

Car Model		Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92
Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99

Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114
Mercedes	E-Class	2100	1605	115
Volvo	XC70	2000	1746	117
Ford	B-Max	1600	1235	104
BMW	2	1600	1390	108
Opel	Zafira	1600	1405	109
Mercedes	SLK	2500	1395	120

We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

How Does it Work?

In Python we have modules that will do the work for us. Start by importing the Pandas module.

```
import pandas
```

Learn about the Pandas module in our Pandas Tutorial.

The Pandas module allows us to read csv files and return a DataFrame object.

The file is meant for testing purposes only, you can download it here: data.csv

```
df = pandas.read_csv("data.csv")
```

Then make a list of the independent values and call this variable `X`.

Put the dependent values in a variable called `y`.

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

Tip: It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

We will use some methods from the sklearn module, so we will have to import that module as well:

```
from sklearn import linear_model
```

From the sklearn module we will use the `LinearRegression()` method to create a linear regression object.

This object has a method called `fit()` that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Now we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume
is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])
```

EXAMPLE

See the whole example in action:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Result:

```
[107.2087328]
```

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometer it drives.

Coefficient

The coefficient is a factor that describes the relationship with an unknown variable.

Example: if `x` is a variable, then `2x` is `x` two times. `x` is the unknown variable, and the number `2` is the coefficient.

In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2. The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

Example

Print the coefficient values of the regression object:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Result:

```
[0.00755095 0.00780526]
```

Result Explained

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095
Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm^3, the CO2 emission increases by 0.00780526 g.

I think that is a fair guess, but let test it!

We have already predicted that if a car with a 1300cm^3 engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Example

Copy the example from before, but change the weight from 2300 to 3300:

```
import pandas
from sklearn import linear_model
```

```
df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Result:

```
[114.75968007]
```

We have predicted that a car with 1.3 liter engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometer it drives.

Which shows that the coefficient of 0.00755095 is correct:

107.2087328 + (1000 * 0.00755095) = 114.75968

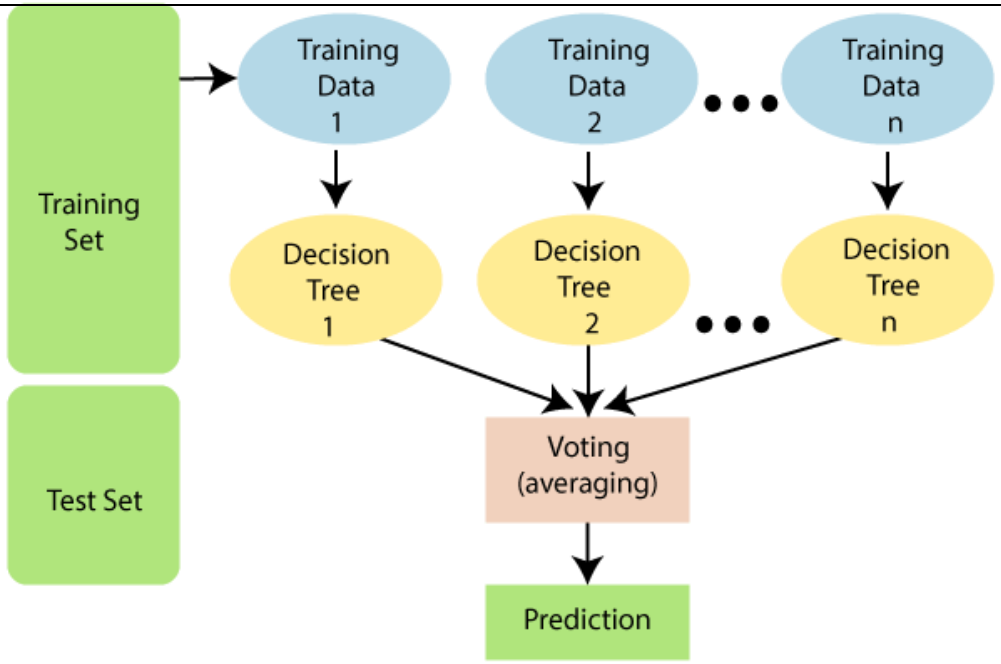
RANDOM FOREST ALGORITHM

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Note: To better understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

<=”” li=””>

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

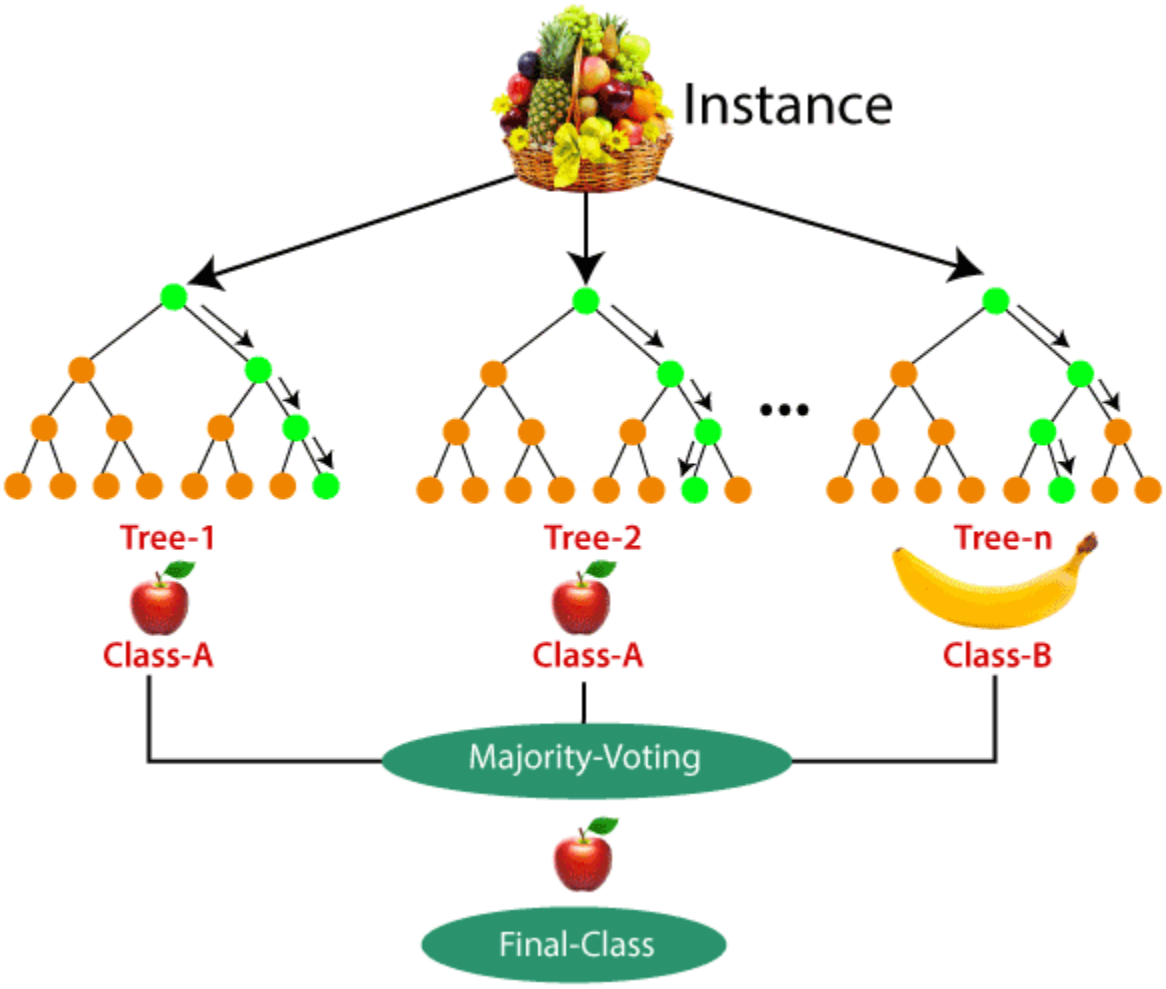
Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as Decision tree Classifier, KNN, SVM, Logistic Regression, etc.

Implementation Steps are given below:

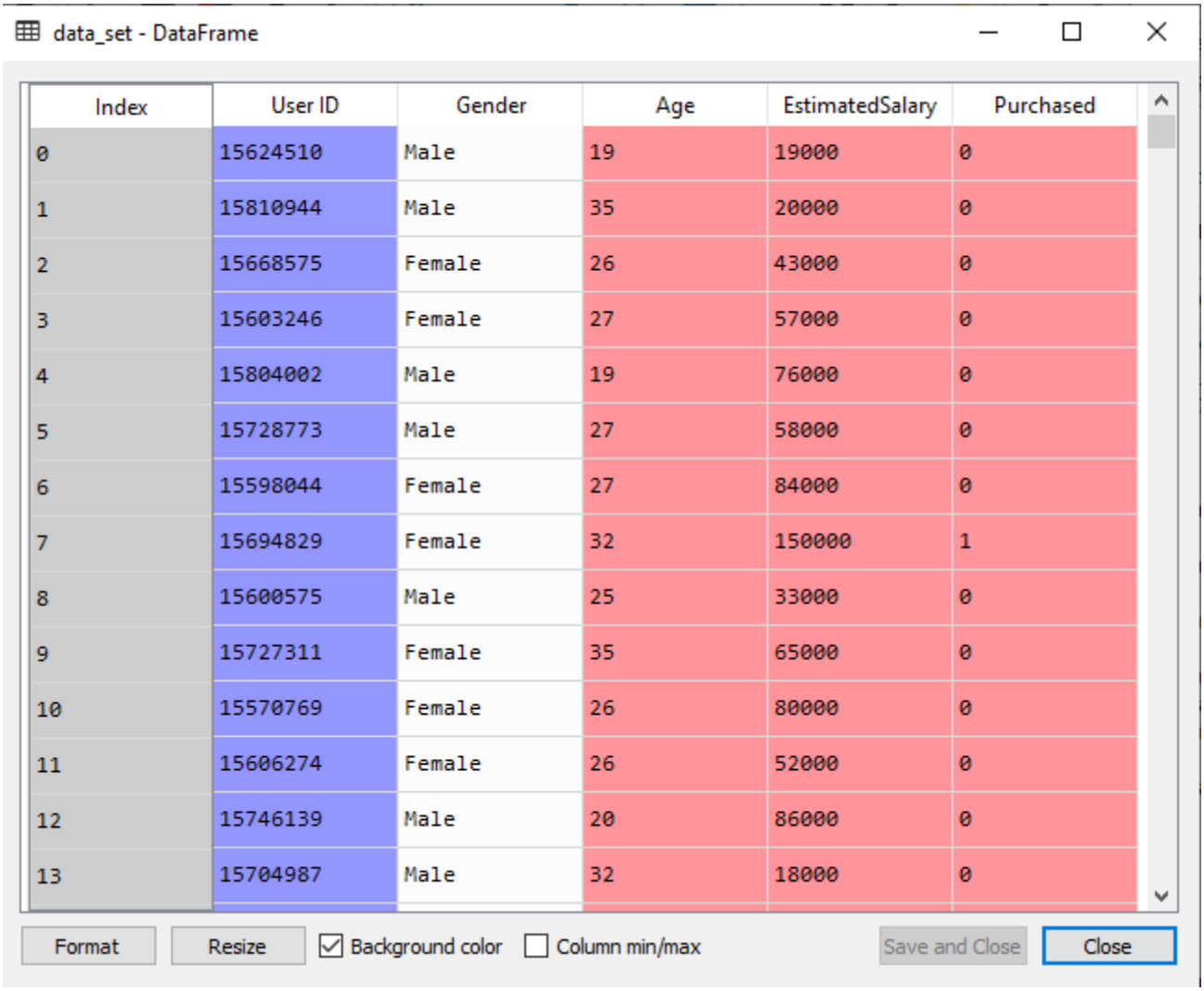
- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0

2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

```
1. #Fitting Decision Tree classifier to the training set
2. from sklearn.ensemble import RandomForestClassifier
3. classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
4. classifier.fit(x_train, y_train)
```

In the above code, the classifier object takes below parameters:

- **n_estimators=** The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion=** It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

Output:

```
RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='entropy',
```

```
                        max_depth=None, max_features='auto',
max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

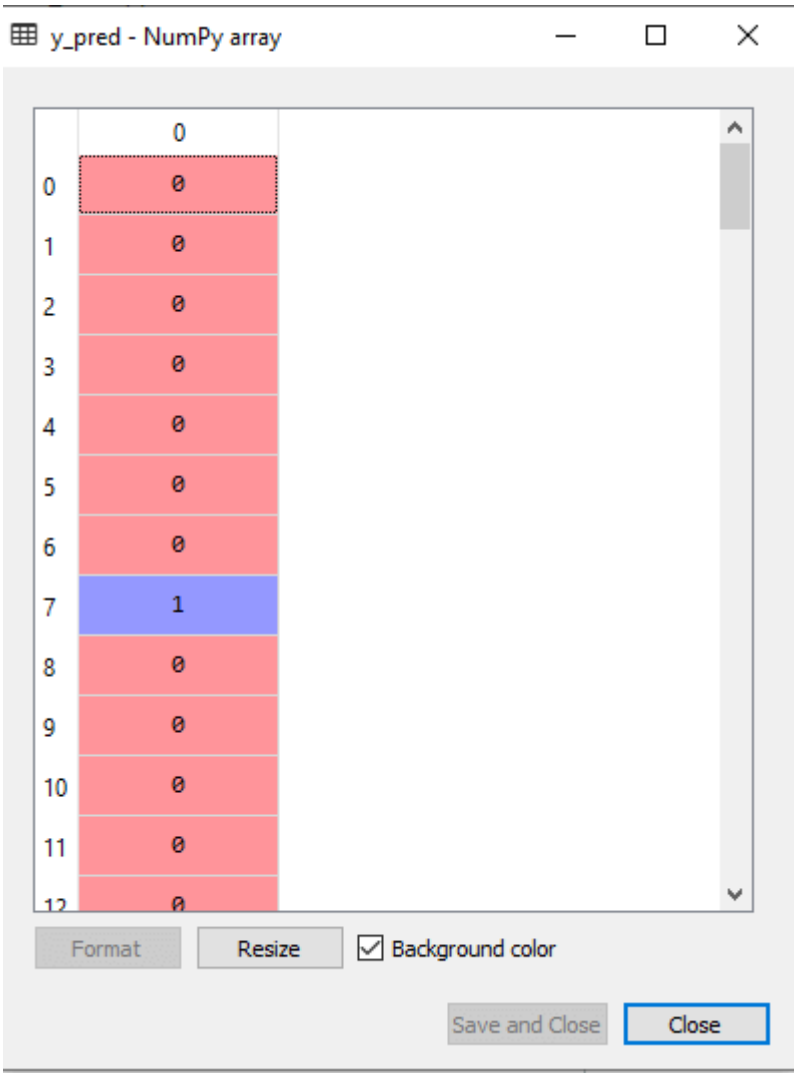
3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector y_pred. Below is the code for it:

```
1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)
```

Output:

The prediction vector is given as:



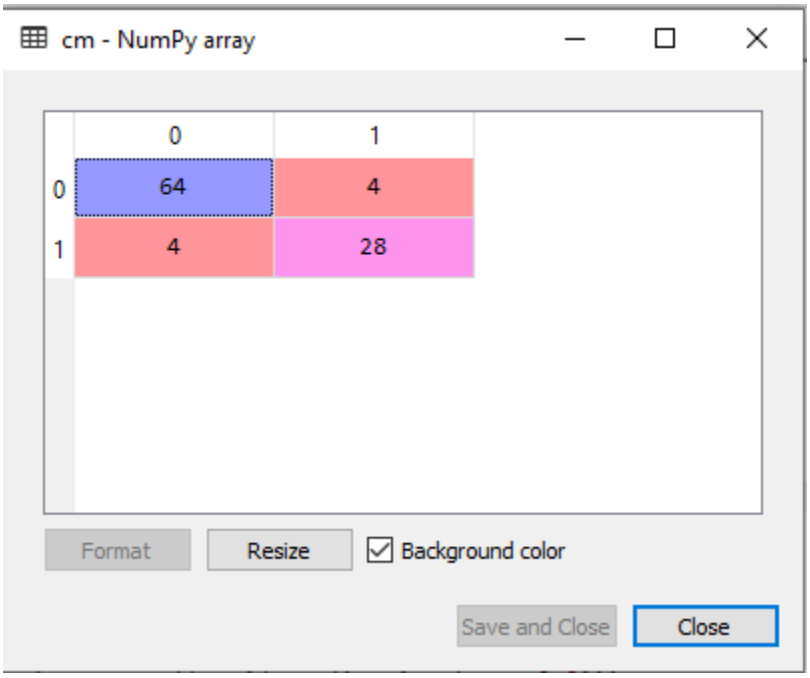
By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

```
1. #Creating the Confusion matrix
2. from sklearn.metrics import confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)
```

Output:



As we can see in the above matrix, there are **4+4= 8 incorrect predictions** and **64+28= 92 correct predictions.**

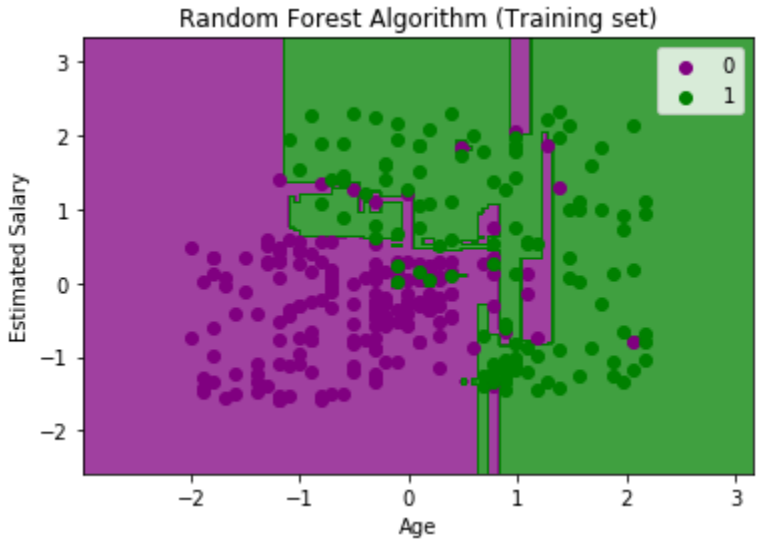
5. Visualizing the training Set result

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the Random forest classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Logistic Regression. Below is the code for it:

```
1. from matplotlib.colors import ListedColormap
2. x_set, y_set = x_train, y_train
3. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1
   , step  =0.01),
4. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
5. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.sh
   ape),
6. alpha = 0.75, cmap = ListedColormap(('purple','green' )))
7. mtp.xlim(x1.min(), x1.max())
8. mtp.ylim(x2.min(), x2.max())
9. for i, j in enumerate(nm.unique(y_set)):
10.    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
11.        c = ListedColormap(('purple', 'green'))(i), label = j)
```

```
12. mtp.title('Random Forest Algorithm (Training set)')
13. mtp.xlabel('Age')
14. mtp.ylabel('Estimated Salary')
15. mtp.legend()
16. mtp.show()
```

Output:



The above image is the visualization result for the Random Forest classifier working with the training set result. It is very much similar to the Decision tree classifier. Each data point corresponds to each user of the user_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV.

So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.

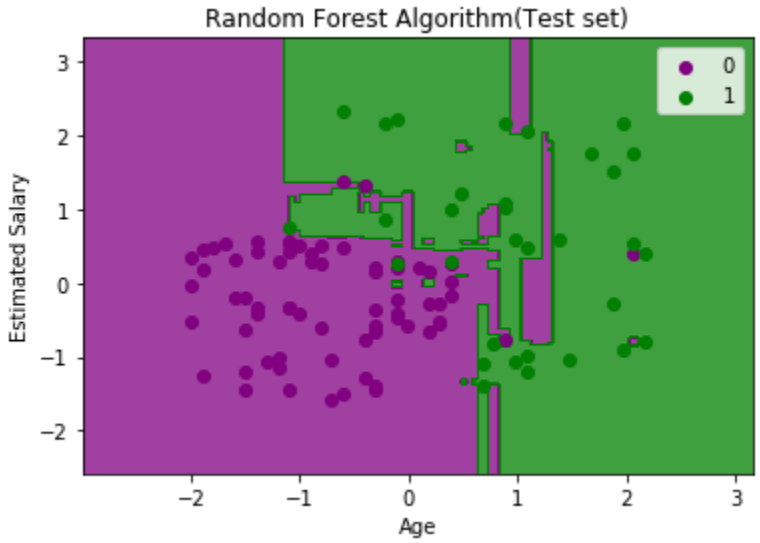
6. Visualizing the test set result

Now we will visualize the test set result. Below is the code for it:

```
1. #Visulaizing the test set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0
   .01),
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('purple','green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.         c = ListedColormap(('purple', 'green'))(i), label = j)
13. mtp.title('Random Forest Algorithm(Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
```

16. mtp.legend()
17. mtp.show()

Output:

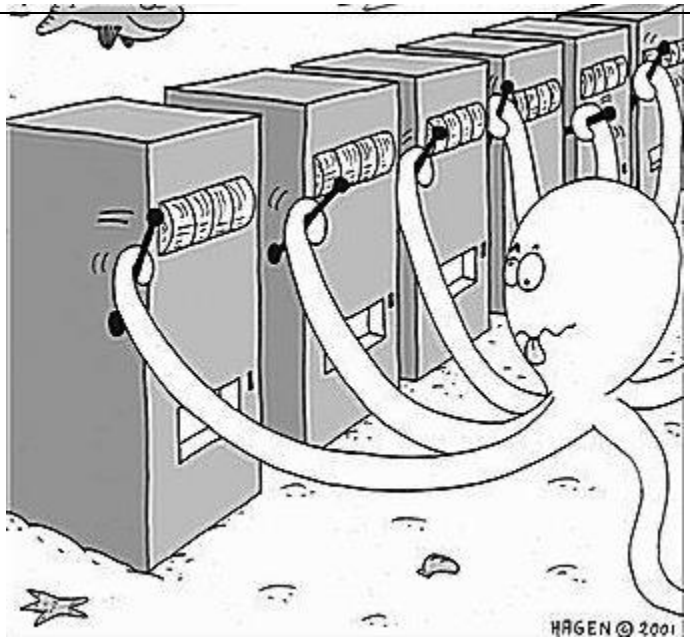


The above image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

MULTIARMED BANDIT ALGORITHM

A bandit is defined as someone who steals your money. A one-armed bandit is a simple slot machine wherein you insert a coin into the machine, pull a lever, and get an immediate reward. But why is it called a bandit? It turns out all casinos configure these slot machines in such a way that all gamblers end up losing money!

A multi-armed bandit is a complicated slot machine wherein instead of 1, there are several levers which a gambler can pull, with each lever giving a different return. The probability distribution for the reward corresponding to each lever is different and is unknown to the gambler.



The task is to identify which lever to pull in order to get maximum reward after a given set of trials. This problem statement is like a single step Markov decision process,Each arm chosen is equivalent to an action, which then leads to an immediate reward.

Exploration Exploitation in the context of Bernoulli MABP

The below table shows the sample results for a 5-armed Bernoulli bandit with arms labelled as 1, 2, 3, 4 and 5:

Arm	Reward
1	0
2	0
3	1
4	1
5	0
3	1
3	1
2	0
1	1
4	0
2	0

This is called Bernoulli, as the reward returned is either 1 or 0. In this example, it looks like the arm number 3 gives the maximum return and hence one idea is to keep playing this arm in order to obtain the maximum reward (pure exploitation).

Just based on the knowledge from the given sample, 5 might look like a bad arm to play, but we need to keep in mind that we have played this arm only once and maybe we should play it a few

more times (exploration) to be more confident. Only then should we decide which arm to play (exploitation).

Use Cases

Bandit algorithms are being used in a lot of research projects in the industry. I have listed some of their use cases in this section.

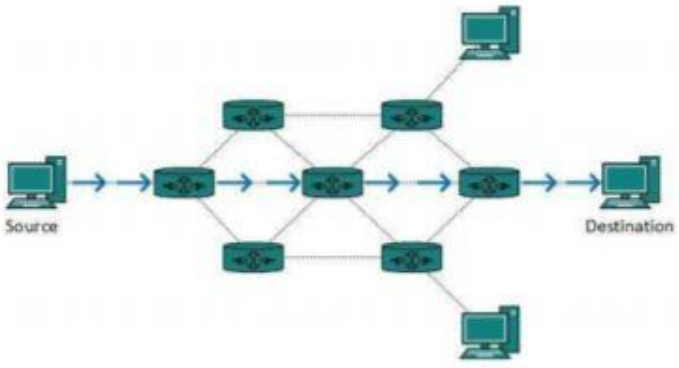
Clinical Trials

The well being of patients during clinical trials is as important as the actual results of the study. Here, exploration is equivalent to identifying the best treatment, and exploitation is treating patients as effectively as possible during the trial.

Clinical Trials

Network Routing

Routing is the process of selecting a path for traffic in a network, such as telephone networks or computer networks (internet). Allocation of channels to the right users, such that the overall throughput is maximised, can be formulated as a MABP.



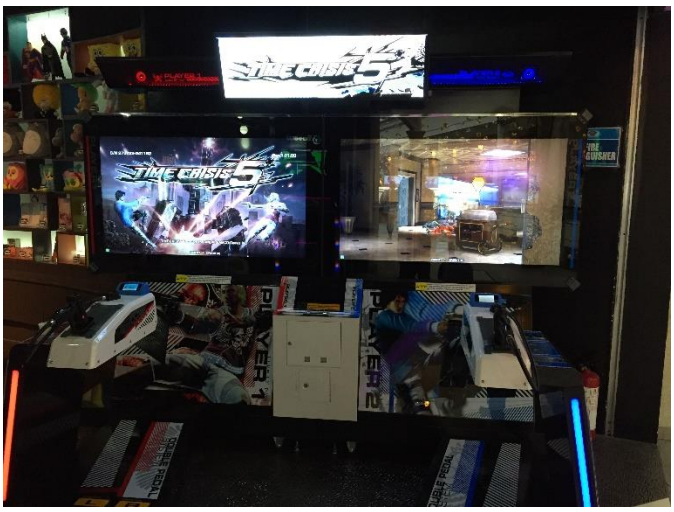
Online Advertising

The goal of an advertising campaign is to maximise revenue from displaying ads. The advertiser makes revenue every time an offer is clicked by a web user. Similar to MABP, there is a trade-off between exploration, where the goal is to collect information on an ad's performance using click-through rates, and exploitation, where we stick with the ad that has performed the best so far.

Online Ads

Game Designing

Building a hit game is challenging. MABP can be used to test experimental changes in game play/interface and exploit the changes which show positive experiences for players.



Game Designing

Solution Strategies

In this section, we will discuss some strategies to solve a multi-armed bandit problem. But before that, let's get familiar with a few terms we'll be using from here on.

Action-Value Function

The expected payoff or expected reward can also be called an action-value function. It is represented by q(a) and defines the average reward for each action at a time t.

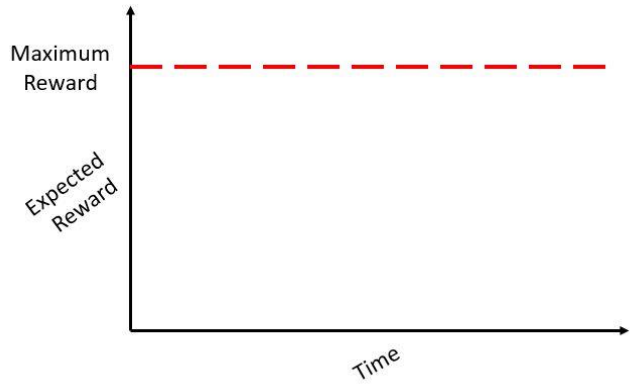
$$Q(a) = \mathbb{E}[r|a]$$

Suppose the reward probabilities for a K-armed bandit are given by {P1, P2, P3 …… Pk}. If the *ith* arm is selected at time t, then Qt(a) = Pi.

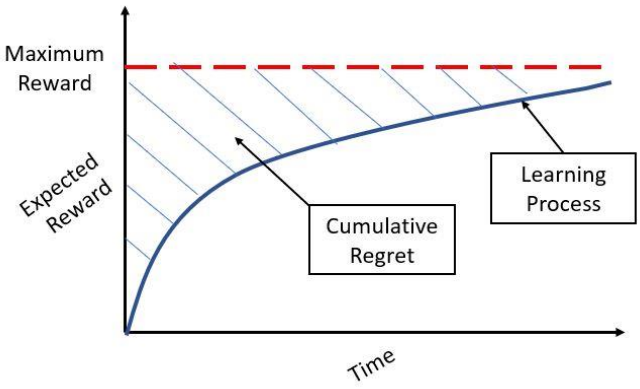
The question is, how do we decide whether a given strategy is better than the rest? One direct way is to compare the total or average reward which we get for each strategy after *n* trials. If we already know the best action for the given bandit problem, then an interesting way to look at this is the concept of regret.

Regret

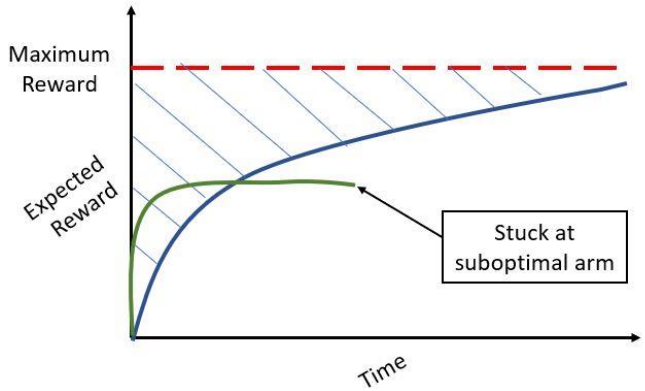
Let's say that we are already aware of the best arm to pull for the given bandit problem. If we keep pulling this arm repeatedly, we will get a maximum expected reward which can be represented as a horizontal line (as shown in the figure below):



But in a real problem statement, we need to make repeated trials by pulling different arms till we am approximately sure of the arm to pull for maximum average return at a time *t*. **The loss that we incur due to time/rounds spent due to the learning is called regret.** In other words, we want to maximise my reward even during the learning phase. Regret is very aptly named, as it quantifies exactly how much you regret not picking the optimal arm.



Now, one might be curious as to how does the regret change if we are following an approach that does not do enough exploration and ends exploiting a suboptimal arm. Initially there might be low regret but overall we are far lower than the maximum achievable reward for the given problem as shown by the green curve in the following figure.



Based on how exploration is done, there are several ways to solve the MABP. Next, we will discuss some possible solution strategies.

No Exploration (Greedy Approach)

A naïve approach could be to calculate the q, or action value function, for all arms at each timestep. From that point onwards, select an action which gives the maximum q. The action values for each action will be stored at each timestep by the following function:

$$Q_t(a) = \frac{\sum_{i=1}^{t} 1_{(a_t=a)} . R_i}{\sum_{i=1}^{t} 1_{(a_t=a)}}$$

It then chooses the action at each timestep that maximises the above expression, given by:

$$argmax_a \, Q_t(a)$$

However, for evaluating this expression at each time t, we will need to do calculations over the whole history of rewards. We can avoid this by doing a running sum. So, at each time t, the q-value for each action can be calculated using the reward:

$$Q_t(a) = \frac{Q_{t-1}(a)N_t(a_t) + R_t.1_{(a_t=a)}}{N_t(a_t)}$$

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{N_t(a_t)}(R_t - Q_{t-1}(a))$$

The problem here is this approach only exploits, as it always picks the same action without worrying about exploring other actions that might return a better reward. Some exploration is necessary to actually find an optimal arm, otherwise we might end up pulling a suboptimal arm forever.

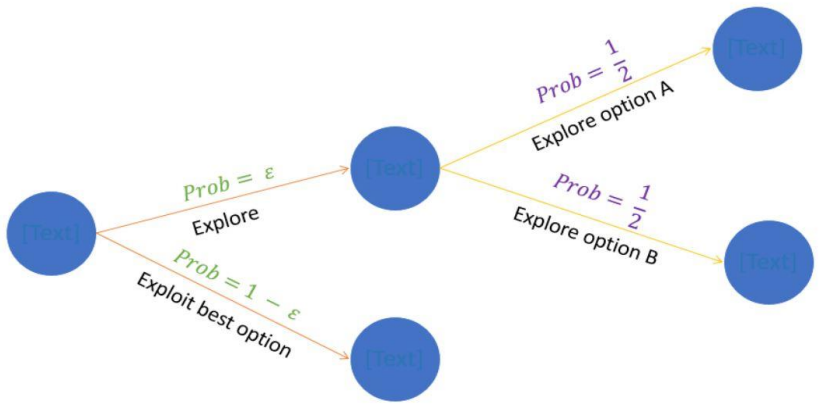
Epsilon Greedy Approach

One potential solution could be to now, and we can then explore new actions so that we ensure we are not missing out on a better choice of arm. With epsilon probability, we will choose a random action (exploration) and choose an action with maximum $q_t(a)$ with probability 1-epsilon.

With probability 1- epsilon – we choose action with maximum value (argmaxa Qt(a))

With probability epsilon – we randomly choose an action from a set of all actions A

For example, if we have a problem with two actions – A and B, the epsilon greedy algorithm works as shown below:



This is much better than the greedy approach as we have an element of exploration here. However, if two actions have a very minute difference between their q values, then even this algorithm will choose only that action which has a probability higher than the others.

Softmax Exploration

The solution is to make the probability of choosing an action proportional to q. This can be done using the softmax function, where the probability of choosing action *a* at each step is given by the following expression:

$$P(a_t = a) = \frac{e^{Q_t(a)}}{\sum_1^n e^{Q_t(b)}}$$

Decayed Epsilon Greedy

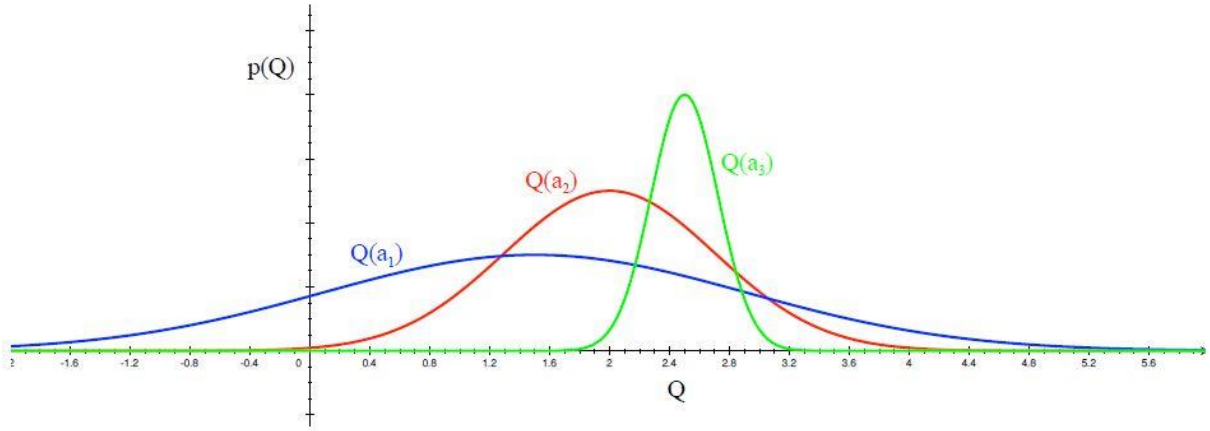
The value of epsilon is very important in deciding how well the epsilon greedy works for a given problem. We can avoid setting this value by keeping epsilon dependent on time. For example, epsilon can be kept equal to *1/log(t+0.00001)*. It will keep reducing as time passes, to the point where we starting exploring less and less as we become more confident of the optimal action or arm.

The problem with random selection of actions is that after sufficient timesteps even if we know that some arm is bad, this algorithm will keep choosing that with probability *epsilon/n*. Essentially, we are exploring a bad action which does not sound very efficient. The approach to get around this could be to favour exploration of arms with a strong potential in order to get an optimal value.

Upper Confidence Bound

Upper Confidence Bound (UCB) is the most widely used solution method for multi-armed bandit problems. This algorithm is based on the principle of optimism in the face of uncertainty.

In other words, the more uncertain we are about an arm, the more important it becomes to explore that arm.



- Distribution of action-value functions for 3 different arms a1, a2 and a3 after several trials is shown in the figure above. This distribution shows that the action value for a1 has the highest variance and hence maximum uncertainty.
- UCB says that we should choose the arm a_1 and receive a reward making us less uncertain about its action-value. For the next trial/timestep, if we still are very uncertain about a_1, we will choose it again until the uncertainty is reduced below a threshold.

The intuitive reason this works is that when acting optimistically in this way, one of two things happen:

- optimism is justified and we get a positive reward which is the objective ultimately
- the optimism was not justified. In this case, we play an arm that we believed might give a large reward when in fact it does not. If this happens sufficiently often, then we will learn what is the true payoff of this action and not choose it in the future.

UCB is actually a family of algorithms. Here, we will discuss UCB1.

Steps involved in UCB1:

- *Play each of the K actions once, giving initial values for mean rewards corresponding to each action at*
- *For each round t = K:*
- *Let $N_t(a)$ represent the number of times action a was played so far*
- *Play the action at maximising the following expression:*

$$Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

- *Observe the reward and update the mean reward or expected payoff for the chosen action*

We will not go into the mathematical proof for UCB. However, it is important to understand the expression that corresponds to our selected action. Remember, in the random exploration we just had Q(a) to maximise, while here we have two terms. First is the action value function, while the second is the confidence term.

- Each time *a* is selected, the uncertainty is presumably reduced: $N_t(a)$ increments, and, as it appears in the denominator, the uncertainty term decreases.

$$N_t(a) \uparrow \quad \sqrt{\frac{2 \log t}{N_t(a)}} \downarrow$$

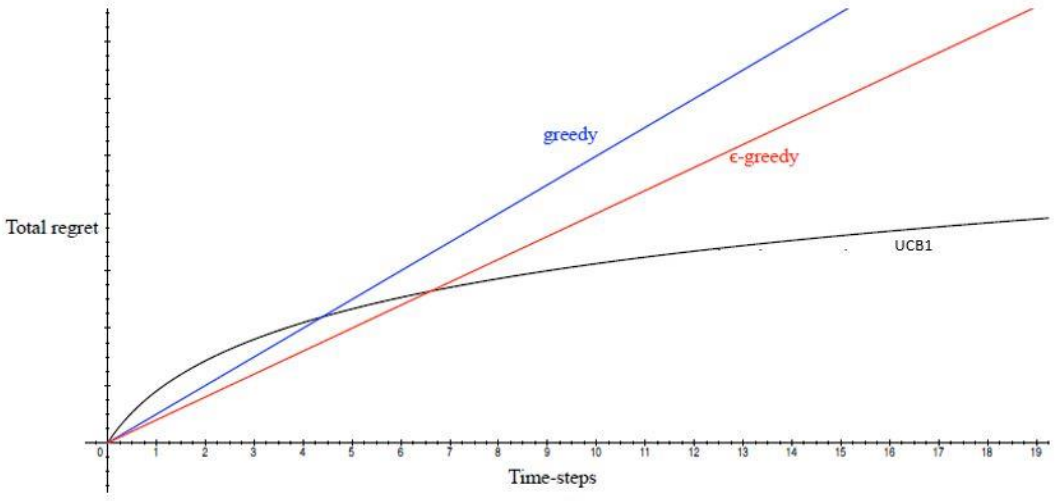
- On the other hand, each time an action other than *a* is selected, t increases, but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.

$$t \uparrow \quad \text{With } N_t(a) \text{ constant} \quad \sqrt{\frac{2 \log t}{N_t(a)}} \uparrow$$

- The use of the natural logarithm means that the increases get smaller over time; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.
- This will ultimately lead to the optimal action being selected repeatedly in the end.

Regret Comparison

Among all the algorithms given in this article, only the UCB algorithm provides a strategy where the regret increases as log(t), while in the other algorithms we get linear regret with different slopes.



Non-Stationary Bandit problems

An important assumption we are making here is that we are working with the same bandit and distributions from which rewards are being sampled at each timestep stays the same. This is called a stationary problem. To explain it with another example, say you get a reward of 1 every time a coin is tossed, and the result is head. Say after 1000 coin tosses due to wear and tear the coin becomes biased then this will become a non-stationary problem.

To solve a non-stationary problem, more recent samples will be important and hence we could use a constant discounting factor alpha and we can rewrite the update equation like this:

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{\alpha}(R_t - Q_{t-1}(a))$$

Note that we have replaced Nt(at) here with a constant alpha, which ensures that the recent samples are given higher weights, and increments are decided more by such recent samples. There are other techniques which provide different solutions to bandits with non-stationary rewards.

Python Implementation from scratch for Ad CTR Optimization

As mentioned in the use cases section, MABP has a lot of applications in the online advertising domain.

Suppose an advertising company is running 10 different ads targeted towards a similar set of population on a webpage. We have results for which ads were clicked by a user here. Each column index represents a different ad. We have a 1 if the ad was clicked by a user, and 0 if it was not. A sample from the original dataset is shown below:

Ad 1	Ad 2	Ad 3	Ad 4	Ad 5	Ad 6	Ad 7	Ad 8	Ad 9	Ad 10
1	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0

This is a simulated dataset and it has Ad #5 as the one which gives the maximum reward.

First, we will try a random selection technique, where we randomly select any ad and show it to the user. If the user clicks the ad, we get paid and if not, there is no profit.

```
# Random Selection

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Ads_Optimisation.csv')

# Implementing Random Selection
import random
N = 10000
d = 10
ads_selected = []
total_reward = 0
for n in range(0, N):
    ad = random.randrange(d)
    ads_selected.append(ad)
```

```
    reward = dataset.values[n, ad]
    total_reward = total_reward + reward
```

Total reward for the random selection algorithm comes out to be 1170. As this algorithm is not learning anything, it will not smartly select any ad which is giving the maximum return. And hence even if we look at the last 1000 trials, it is not able to find the optimal ad.

```
pd.Series(ads_selected).tail(1000).value_counts(normalize = True)
```

```
0    0.121
7    0.117
2    0.109
3    0.101
4    0.099
8    0.096
6    0.095
9    0.090
1    0.090
5    0.082
```

Now, let's try the Upper Confidence Bound algorithm to do the same:

```
# Implementing UCB
import math
N = 10000
d = 10
ads_selected = []
numbers_of_selections = [0] * d
sums_of_reward = [0] * d
total_reward = 0

for n in range(0, N):
    ad = 0
    max_upper_bound = 0
    for i in range(0, d):
        if (numbers_of_selections[i] > 0):
            average_reward = sums_of_reward[i] / numbers_of_selections[i]
            delta_i = math.sqrt(2 * math.log(n+1) / numbers_of_selections[i])
            upper_bound = average_reward + delta_i
        else:
            upper_bound = 1e400
        if upper_bound > max_upper_bound:
            max_upper_bound = upper_bound
            ad = i
    ads_selected.append(ad)
    numbers_of_selections[ad] += 1
    reward = dataset.values[n, ad]
    sums_of_reward[ad] += reward
    total_reward += reward
```

The *total_reward* for UCB comes out to be 2125. Clearly, this is much better than random selection and indeed a smart exploration technique that can significantly improve our strategy to

solve a MABP.

```
pd.Series(ads_selected).head(1500).value_counts(normalize = True)
```

```
4    0.250000
7    0.170667
0    0.106000
3    0.091333
8    0.075333
1    0.073333
6    0.066000
2    0.066000
9    0.053333
5    0.048000
dtype: float64
```

After just 1500 trials, UCB is already favouring Ad #5 (index 4) which happens to be the optimal ad, and gets the maximum return for the given problem.

MODULE-5

LEARNING DETERMINISTIC MODELS

Supervised Learning, Regression, Linear regression, Multiple linear regression, A multiple regression analysis, The analysis of variance for multiple regression, Examples for multiple regression, Overfitting, Detecting overfit models: Cross validation, Cross validation: The ideal procedure, Parameter estimation, Logistic regression, Decision trees: Background, Decision trees, Decision trees for credit card promotion, An algorithm for building decision trees, Attribute selection measure: Information gain, Entropy, Decision Tree: Weekend example, Occam's Razor, Converting a tree to rules, Unsupervised learning, Semi-Supervised learning, Clustering, K – means clustering, Automated discovery, Reinforcement learning, Multi-Armed Bandit algorithms, Influence diagrams, Risk modeling, Sensitivity analysis, Casual learning.

MULTIPLE LINEAR REGRESSION

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Example:

Prediction of CO_2 emission based on engine size and number of cylinders in a car.

One of the most common types of predictive analysis is multiple linear regression. This type of analysis allows you to understand the relationship between a continuous dependent variable and two or more independent variables.

The independent variables can be either continuous (like age and height) or categorical (like gender and occupation). It's important to note that if your dependent variable is categorical, you should dummy code it before running the analysis.

Formula and Calculation of Multiple Linear Regression

Several circumstances that influence the dependent variable simultaneously can be controlled through multiple regression analysis. Regression analysis is a method of analyzing the relationship between independent variables and dependent variables.

Let k represent the number of variables denoted by x1, x2, x3, ……, xk.

For this method, we assume that we have k independent variables x1, . . . , xk that we can set, then they probabilistically determine an outcome Y.

Furthermore, we assume that Y is linearly dependent on the factors according to

$Y = \beta 0 + \beta 1x1 + \beta 2x2 + \cdots + \beta kxk + \varepsilon$

- The variable yi is dependent or predicted
- The slope of y depends on the y-intercept, that is, when xi and x2 are both zero, y will be β0.
- The regression coefficients β1 and β2 represent the change in y as a result of one-unit changes in xi1 and xi2.
- βp refers to the slope coefficient of all independent variables
- ε term describes the random error (residual) in the model.

Where ε is a standard error, this is just like we had for simple linear regression, except k doesn't have to be 1.

We have n observations, n typically being much more than k.

For i th observation, we set the independent variables to the values xi1, xi2 . . . , xik and measure a value yi for the random variable Yi.

Thus, the model can be described by the equations.

Yi = β0 + β1xi1 + β2xi2 + · · · + βkxik + i for i = 1, 2, . . . , n,

Where the errors i are independent standard variables, each with mean 0 and the same unknown variance σ2.

Altogether the model for multiple linear regression has k + 2 unknown parameters:

β0, β1, . . . , βk, and σ 2.

When k was equal to 1, we found the least squares line y = βˆ 0 +βˆ 1x.

It was a line in the plane R 2.

Now, with k ≥ 1, we'll have a least squares hyperplane.

y = βˆ 0 + βˆ 1x1 + βˆ 2x2 + · · · + βˆ kxk in Rk+1.

The way to find the estimators βˆ 0, βˆ 1, . . ., and βˆ k is the same.

Take the partial derivatives of the squared error.

Q = Xn i=1 (yi − (β0 + β1xi1 + β2xi2 + · · · + βkxik))2

When that system is solved we have fitted values

yˆi = βˆ 0 + βˆ 1xi1 + βˆ 2xi2 + · · · + βˆ kxik for i = 1, . . . , n that should be close to the actual values yi.

Example of How to Use Multiple Linear Regression

```
from sklearn.datasets import load_boston
import pandas as pd
from sklearn.model_selection import train_test_split
def sklearn_to_df(data_loader):
    X_data = data_loader.data
    X_columns = data_loader.feature_names
    X = pd.DataFrame(X_data, columns=X_columns)
    y_data = data_loader.target
    y = pd.Series(y_data, name='target')
    return x, y
x, y = sklearn_to_df(load_boston())
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42)
from load_dataset import x_train, x_test, y_train, y_test
from multiple_linear_regression import MultipleLinearRegression
from sklearn.linear_model import LinearRegression
mulreg = MultipleLinearRegression()
# fit our LR to our data
mulreg.fit(x_train, y_train)
# make predictions and score
pred = mulreg.predict(x_test)
# calculate r2_score
score = mulreg.r2_score(y_test, pred)
```

```
print(f'Our Final R^2 score: {score}')
```

DETECTING OVER FIT MODEL:CROSS VALIDATION

Approximate a Target Function in Machine Learning

Supervised machine learning is best understood as approximating a target function (f) that maps input variables (X) to an output variable (Y).

Y = f(X)

This characterization describes the range of classification and prediction problems and the machine algorithms that can be used to address them.

An important consideration in learning the target function from the training data is how well the model generalizes to new data. Generalization is important because the data we collect is only a sample, it is incomplete and noisy.

Generalization in Machine Learning

In machine learning we describe the learning of the target function from training data as inductive learning.

Induction refers to learning general concepts from specific examples which is exactly the problem that supervised machine learning problems aim to solve. This is different from deduction that is the other way around and seeks to learn specific concepts from general rules.

Generalization refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning.

The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen.

There is a terminology used in machine learning when we talk about how well a machine learning model learns and generalizes to new data, namely overfitting and underfitting.

Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms.

Statistical Fit

In statistics, a fit refers to how well you approximate a target function.

This is good terminology to use in machine learning, because supervised machine learning algorithms seek to approximate the unknown underlying mapping function for the output variables given the input variables.

Statistics often describe the goodness of fit which refers to measures used to estimate how well the approximation of the function matches the target function.

Some of these methods are useful in machine learning (e.g. calculating the residual errors), but some of these techniques assume we know the form of the target function we are approximating, which is not the case in machine learning.

If we knew the form of the target function, we would use it directly to make predictions, rather than trying to learn an approximation from samples of noisy training data.

Overfitting in Machine Learning

Overfitting refers to a model that models the training data too well.

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.

Underfitting in Machine Learning

Underfitting refers to a model that can neither model the training data nor generalize to new data.

An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

A Good Fit in Machine Learning

Ideally, you want to select a model at the sweet spot between underfitting and overfitting.

This is the goal, but is very difficult to do in practice.

To understand this goal, we can look at the performance of a machine learning algorithm over time as it is learning a training data. We can plot both the skill on the training data and the skill on a test dataset we have held back from the training process.

Over time, as the algorithm learns, the error for the model on the training data goes down and so does the error on the test dataset. If we train for too long, the performance on the training dataset may continue to decrease because the model is overfitting and learning the irrelevant detail and noise in the training dataset. At the same time the error for the test set starts to rise again as the model's ability to generalize decreases.

The sweet spot is the point just before the error on the test dataset starts to increase where the model has good skill on both the training dataset and the unseen test dataset.

You can perform this experiment with your favorite machine learning algorithms. This is often not useful technique in practice, because by choosing the stopping point for training using the skill on the test dataset it means that the testset is no longer "unseen" or a standalone objective measure. Some knowledge (a lot of useful knowledge) about that data has leaked into the training procedure.

There are two additional techniques you can use to help find the sweet spot in practice: resampling methods and a validation dataset.

How To Limit Overfitting

Both overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is overfitting.

Overfitting is such a problem because the evaluation of machine learning algorithms on training data is different from the evaluation we actually care the most about, namely how well the algorithm performs on unseen data.

There are two important techniques that you can use when evaluating machine learning algorithms to limit overfitting:

1. Use a resampling technique to estimate model accuracy.
2. Hold back a validation dataset.

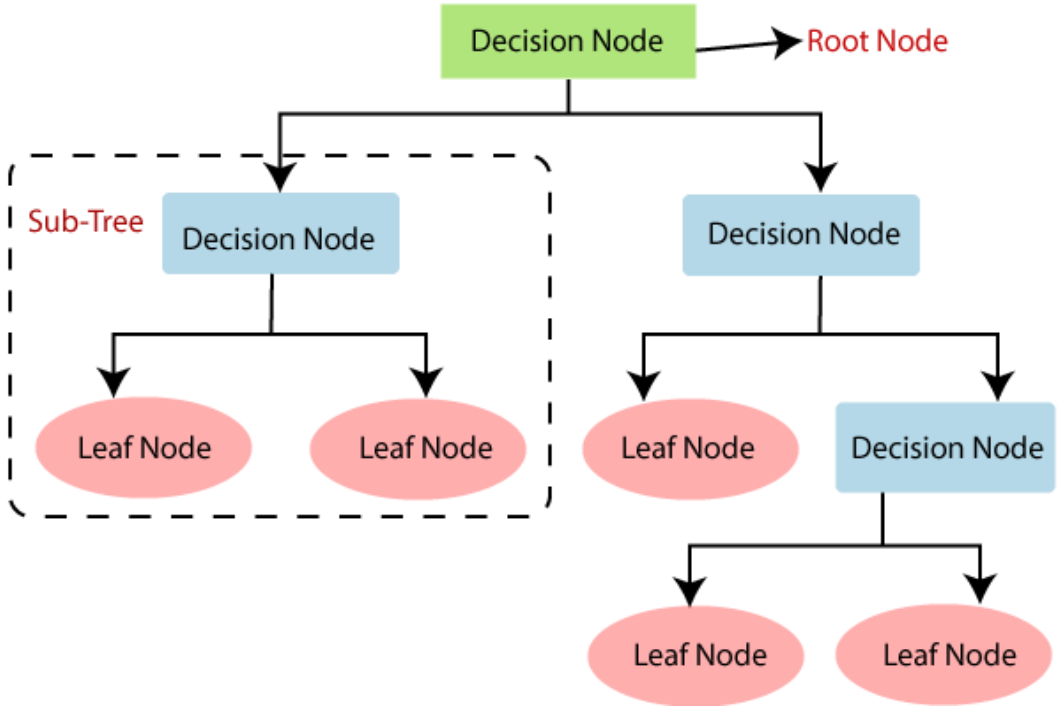
The most popular resampling technique is k-fold cross validation. It allows you to train and test your model k-times on different subsets of training data and build up an estimate of the performance of a machine learning model on unseen data.

A validation dataset is simply a subset of your training data that you hold back from your machine learning algorithms until the very end of your project. After you have selected and tuned your machine learning algorithms on your training dataset you can evaluate the learned models on the validation dataset to get a final objective idea of how the models might perform on unseen data.

Using cross validation is a gold standard in applied machine learning for estimating model accuracy on unseen data. If you have the data, using a validation dataset is also an excellent practice.

Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.

Pruning: Pruning is the process of removing the unwanted branches from the tree.

Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

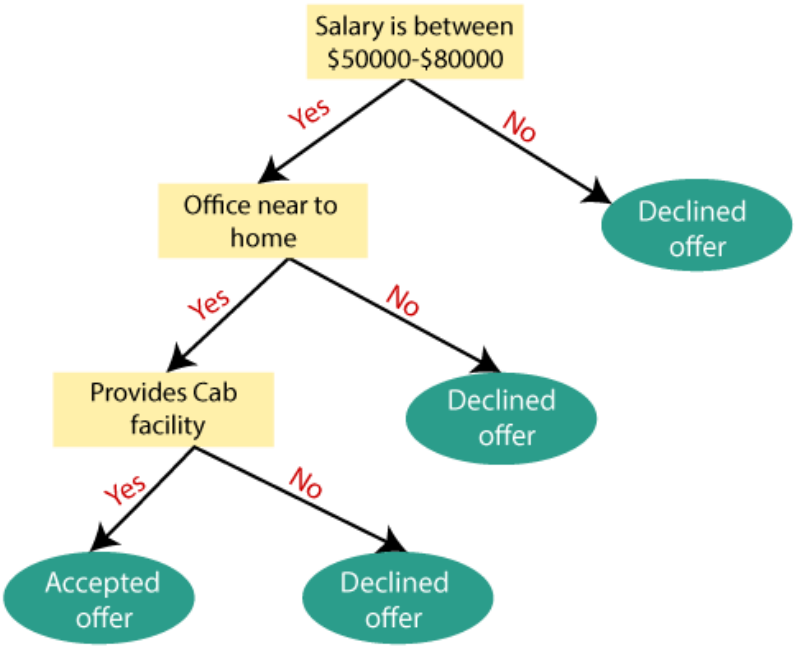
How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

  Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

Gini Index= $1- \sum_j P_j^2$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as KNN SVM, LogisticRegression, etc.

Steps will also remain the same, which are given below:

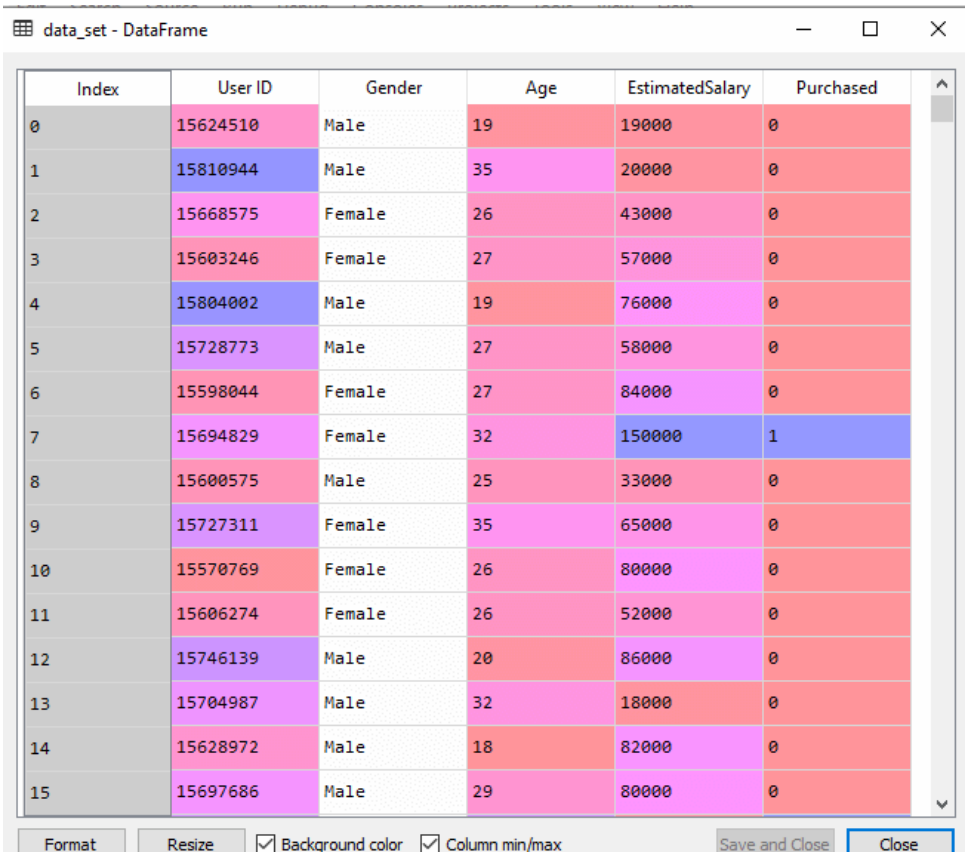
- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy':** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random_state=0":** For generating the random states.

Below is the output for this:

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

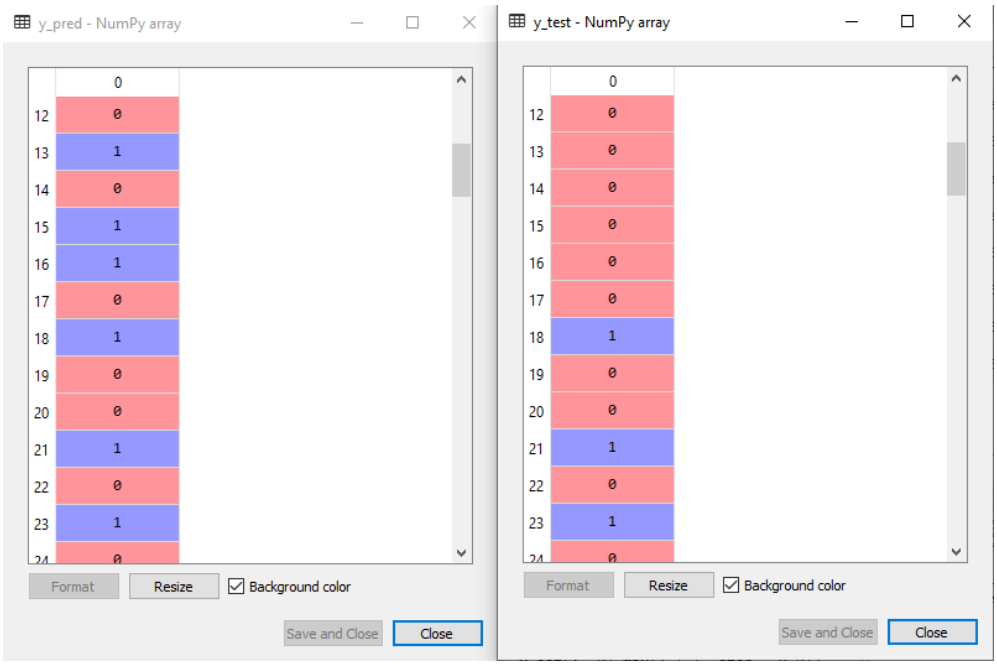
3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector **y_pred.** Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

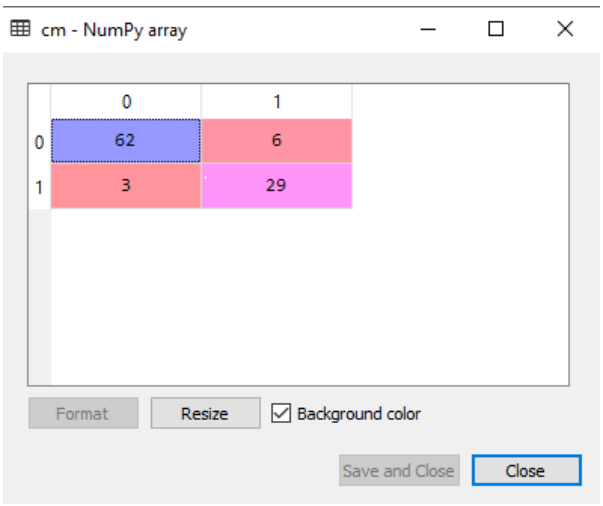


4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

```
1. #Creating the Confusion matrix
2. from sklearn.metrics import confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)
```

Output:



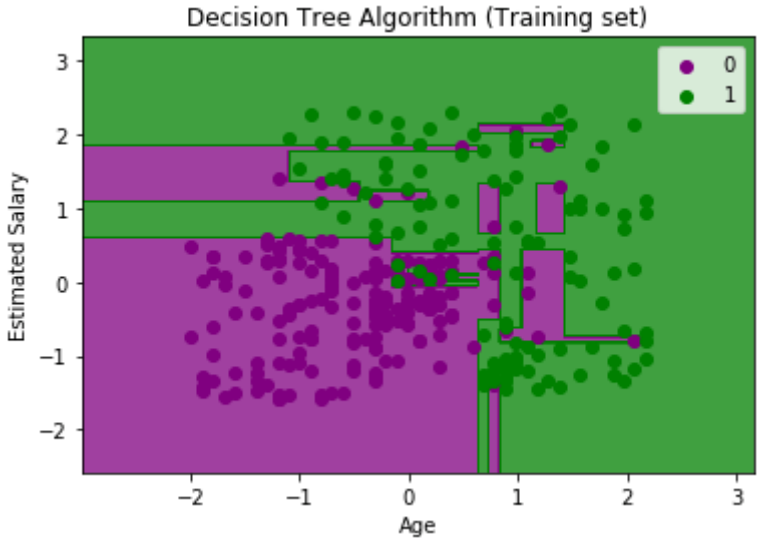
In the above output image, we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and**62+29=91 correct predictions. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.**

5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Logistic Regression. Below is the code for it:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

K-MEANS CLUSTERING ALGORITHM

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

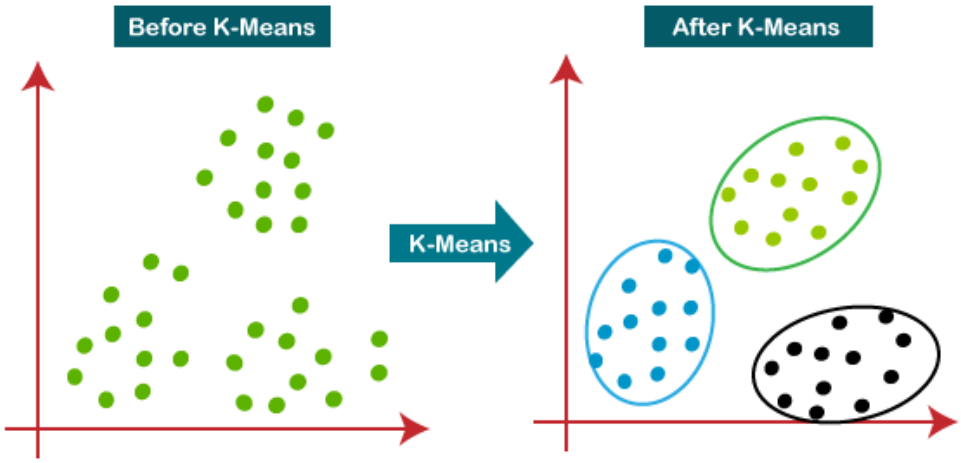
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

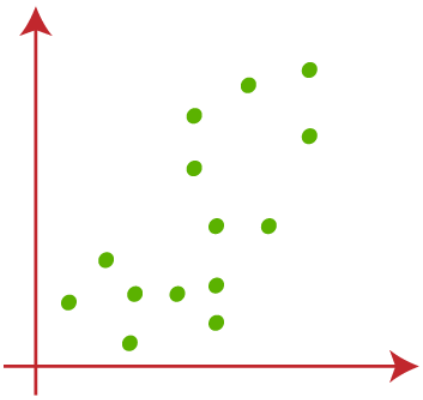
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

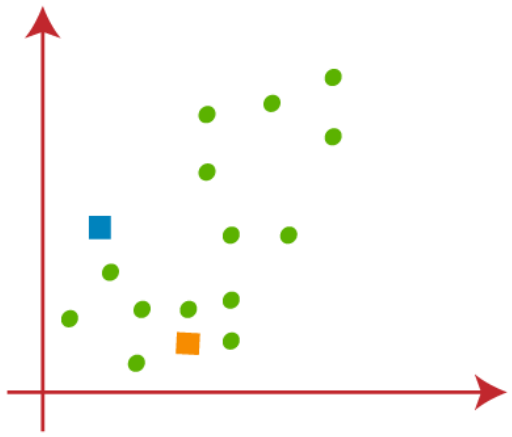
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

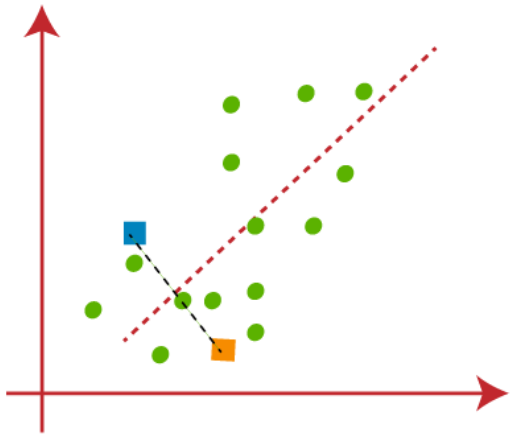


- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the

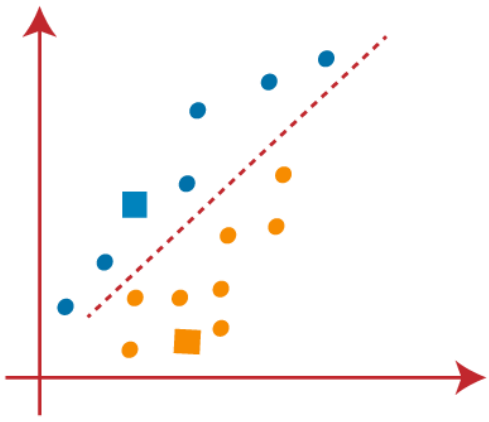
part of our dataset. Consider the below image:



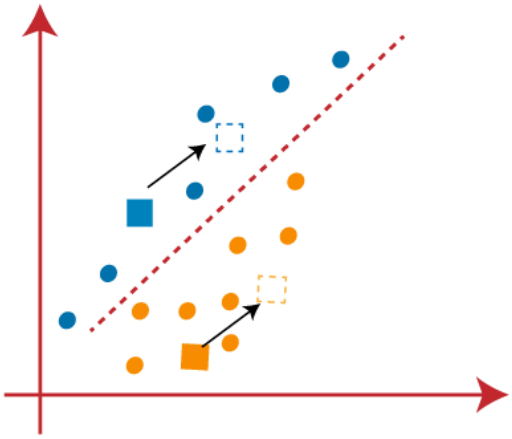
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



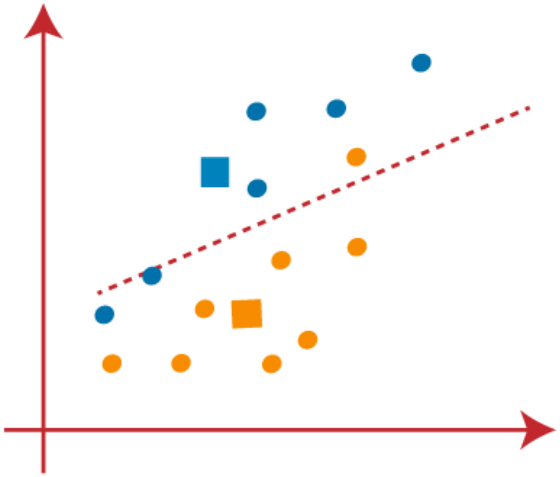
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



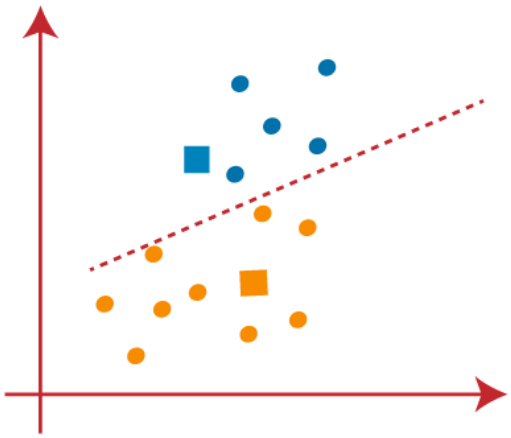
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:



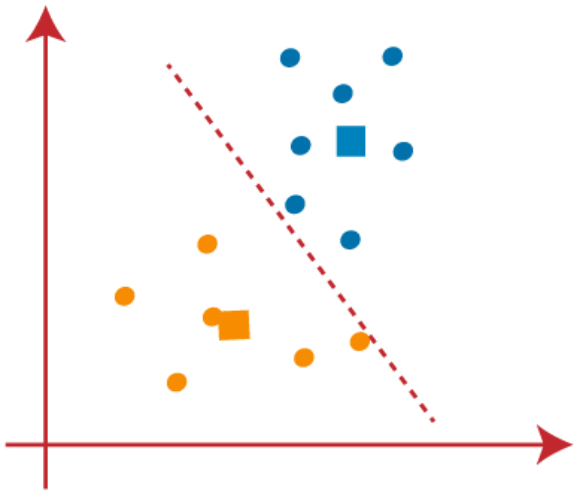
From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.



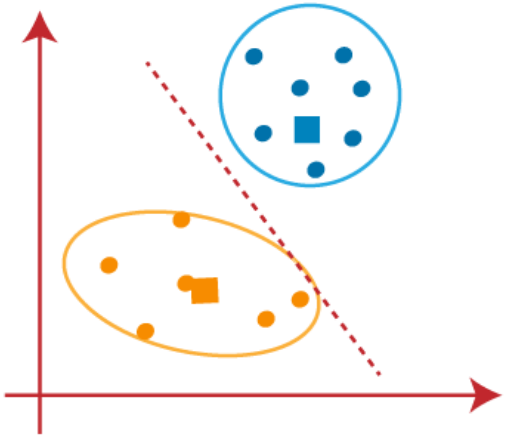
As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:

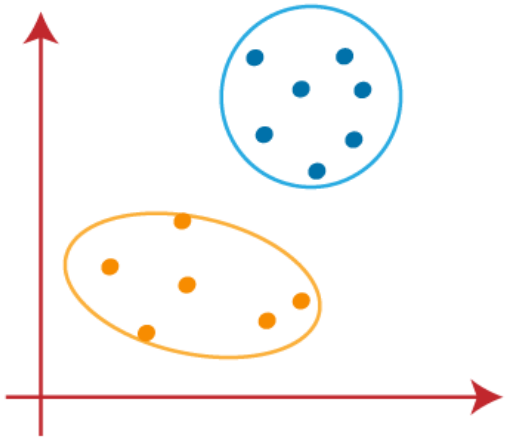
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2 + \sum_{Pi \text{ in Cluster2}} distance(P_i\ C_2)^2 + \sum_{Pi \text{ in CLuster3}} distance(P_i\ C_3)^2$$

In the above formula of WCSS,

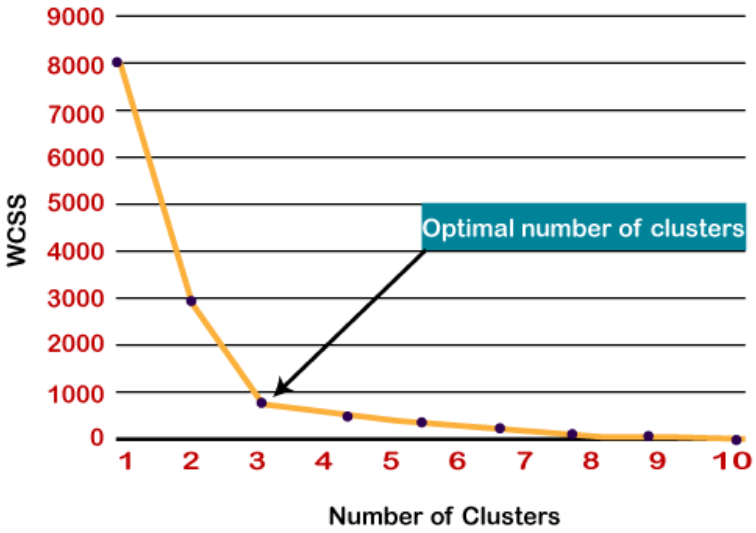
$\sum_{Pi \text{ in Cluster1}} distance(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using Python.

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id, Gender, Age, Annual Income ($), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Step-1: Data pre-processing Step

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the clustering problem, it will be different from other models. Let's discuss it:

- **Importing Libraries**
  As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

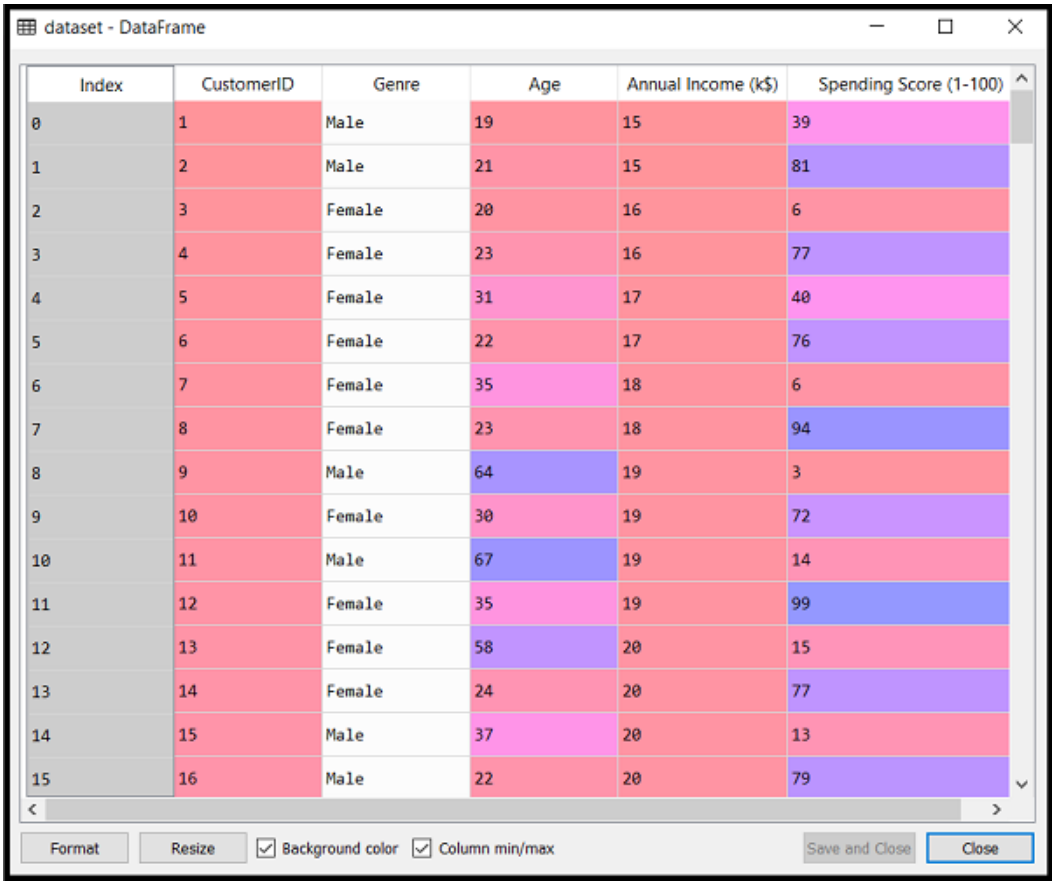
```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
```

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

- **Importing the Dataset:**
  Next, we will import the dataset that we need to use. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')
```

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:



Index	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
x = dataset.iloc[:, [3, 4]].values
```

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
```

```
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

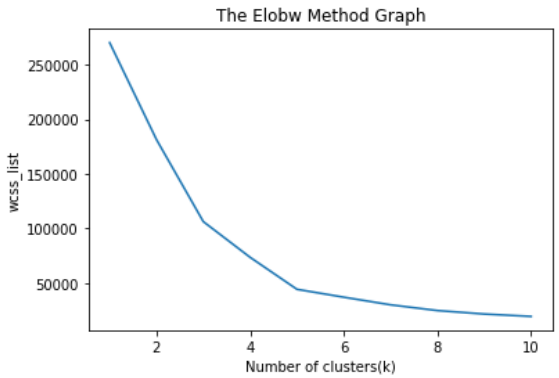
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

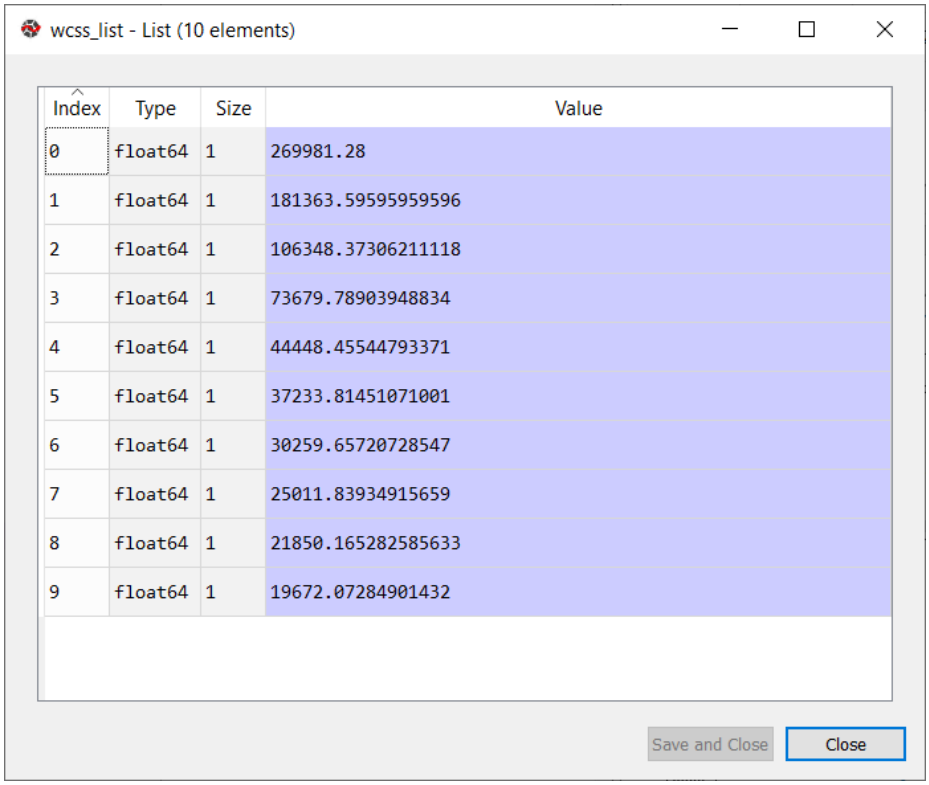
After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10^{th} value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5. So the number of clusters here will be 5.**



Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

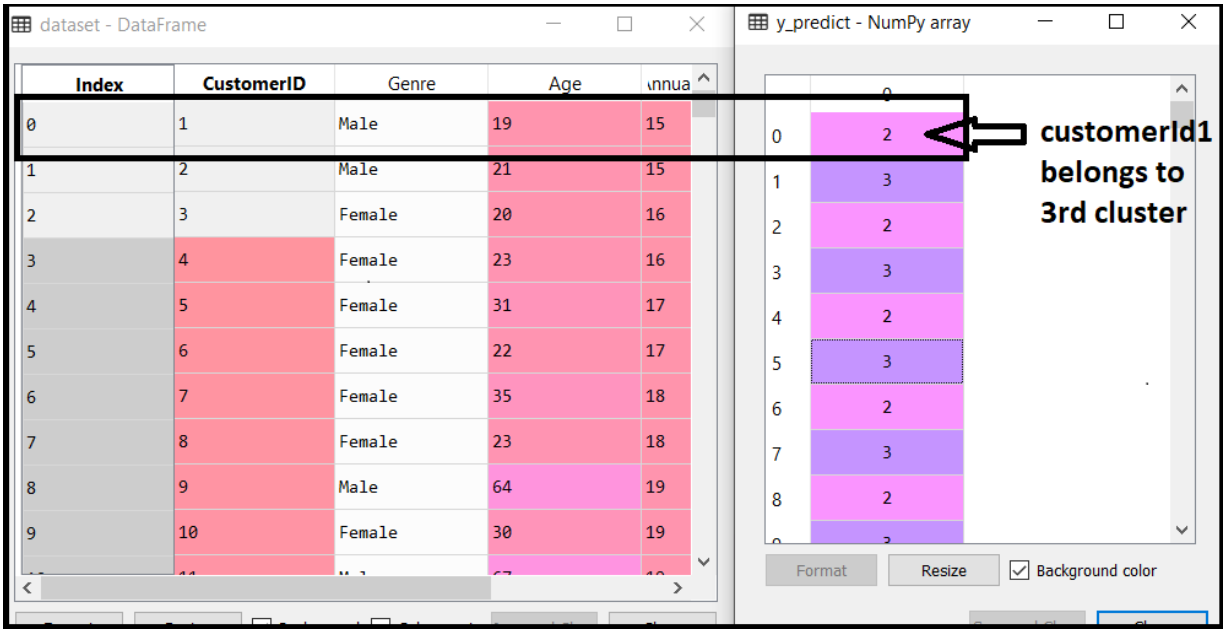
To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:

```
1. #training the K-means model on a dataset
2. kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
3. y_predict= kmeans.fit_predict(x)
```

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y_predict** to train the model.

By executing the above lines of code, we will get the y_predict variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of y_predict with our original dataset. Consider the below image:



From the above image, we can now relate that the CustomerID 1 belongs to a cluster

3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

Step-4: Visualizing the Clusters

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

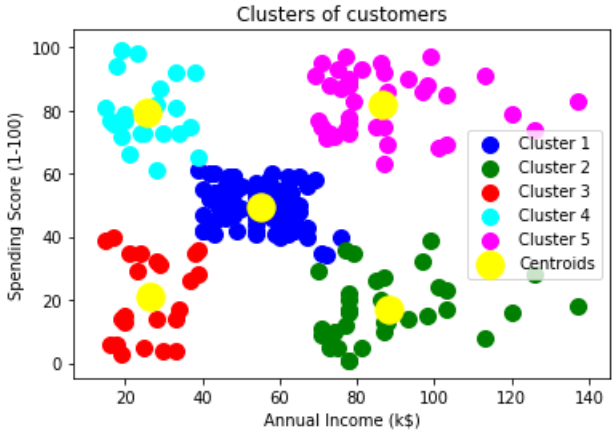
To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.

```
1. #visulaizing the clusters
2. mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
3. mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second
   cluster
4. mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
5. mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth clu
   ster
```

```
6. mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
7. mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Ce
   ntroid')
8. mtp.title('Clusters of customers')
9. mtp.xlabel('Annual Income (k$)')
10. mtp.ylabel('Spending Score (1-100)')
11. mtp.legend()
12. mtp.show()
```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the mtp.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.

Output:



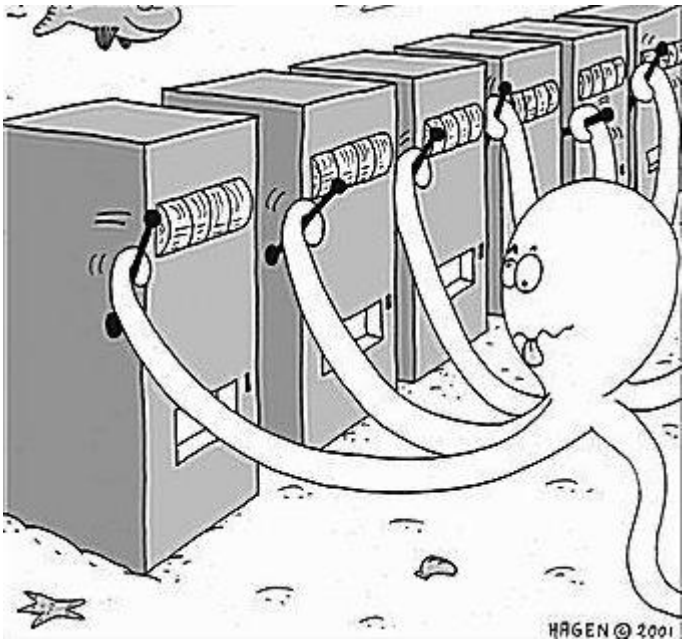
The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as sensible.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as target, and these customers can be the most profitable customers for the mall owner.

MULTI-ARMED BANDIT ALGORITHMS

A bandit is defined as someone who steals your money. A one-armed bandit is a simple slot machine wherein you insert a coin into the machine, pull a lever, and get an immediate reward. But why is it called a bandit? It turns out all casinos configure these slot machines in such a way that all gamblers end up losing money!

A multi-armed bandit is a complicated slot machine wherein instead of 1, there are several levers which a gambler can pull, with each lever giving a different return. The probability distribution for the reward corresponding to each lever is different and is unknown to the gambler.



The task is to identify which lever to pull in order to get maximum reward after a given set of trials. This problem statement is like a single step Markov decision process, which I discussed in this article. Each arm chosen is equivalent to an action, which then leads to an immediate reward.

Exploration Exploitation in the context of Bernoulli MABP

The below table shows the sample results for a 5-armed Bernoulli bandit with arms labelled as 1, 2, 3, 4 and 5:

Arm	Reward
1	0
2	0
3	1
4	1
5	0
3	1
3	1
2	0
1	1
4	0
2	0

This is called Bernoulli, as the reward returned is either 1 or 0. In this example, it looks like the arm number 3 gives the maximum return and hence one idea is to keep playing this arm in order to obtain the maximum reward (pure exploitation).

Just based on the knowledge from the given sample, 5 might look like a bad arm to play, but we need to keep in mind that we have played this arm only once and maybe we should play it a few more times (exploration) to be more confident. Only then should we decide which arm to play (exploitation).

Use Cases

Bandit algorithms are being used in a lot of research projects in the industry. I have listed some of their use cases in this section.

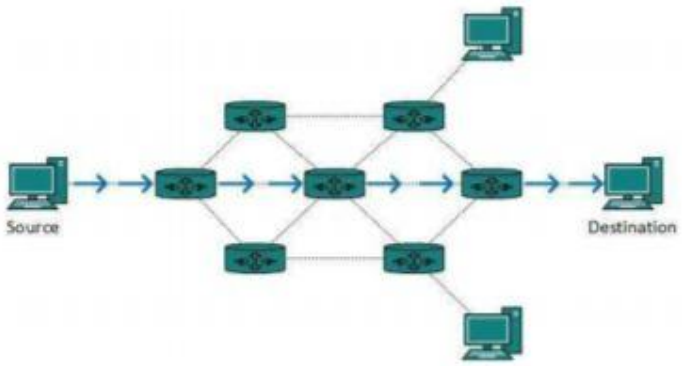
Clinical Trials

The well being of patients during clinical trials is as important as the actual results of the study. Here, exploration is equivalent to identifying the best treatment, and exploitation is treating patients as effectively as possible during the trial.

Clinical Trials

Network Routing

Routing is the process of selecting a path for traffic in a network, such as telephone networks or computer networks (internet). Allocation of channels to the right users, such that the overall throughput is maximised, can be formulated as a MABP.



Network Routing

Online Advertising

The goal of an advertising campaign is to maximise revenue from displaying ads. The advertiser makes revenue every time an offer is clicked by a web user. Similar to MABP, there is a trade-off between exploration, where the goal is to collect information on an ad's performance using click-through rates, and exploitation, where we stick with the ad that has performed the best so far.

Online Ads

Game Designing

Building a hit game is challenging. MABP can be used to test experimental changes in game play/interface and exploit the changes which show positive experiences for players.

Game Designing

Solution Strategies

In this section, we will discuss some strategies to solve a multi-armed bandit problem. But before that, let's get familiar with a few terms we'll be using from here on.

Action-Value Function

The expected payoff or expected reward can also be called an action-value function. It is represented by q(a) and defines the average reward for each action at a time t.

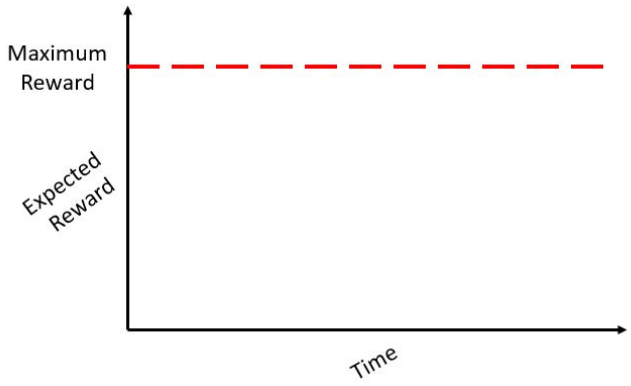
$$Q(a) = \mathbb{E}[r|a]$$

Suppose the reward probabilities for a K-armed bandit are given by {P1, P2, P3 …… Pk}. If the *ith* arm is selected at time t, then Qt(a) = Pi.

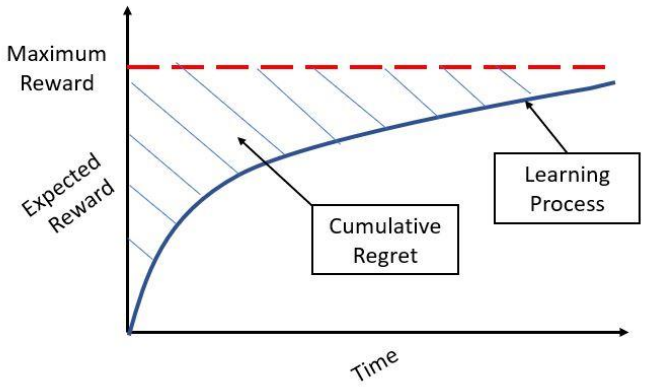
The question is, how do we decide whether a given strategy is better than the rest? One direct way is to compare the total or average reward which we get for each strategy after *n* trials. If we already know the best action for the given bandit problem, then an interesting way to look at this is the concept of regret.

Regret

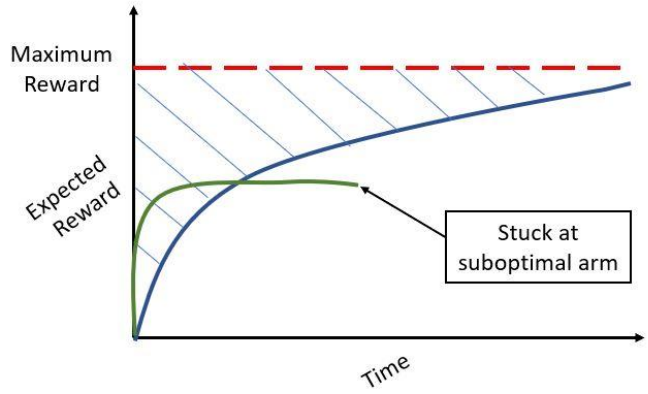
Let's say that we are already aware of the best arm to pull for the given bandit problem. If we keep pulling this arm repeatedly, we will get a maximum expected reward which can be represented as a horizontal line (as shown in the figure below):



But in a real problem statement, we need to make repeated trials by pulling different arms till we am approximately sure of the arm to pull for maximum average return at a time *t*. **The loss that we incur due to time/rounds spent due to the learning is called regret.** In other words, we want to maximise my reward even during the learning phase. Regret is very aptly named, as it quantifies exactly how much you regret not picking the optimal arm.



Now, one might be curious as to how does the regret change if we are following an approach that does not do enough exploration and ends exploiting a suboptimal arm. Initially there might be low regret but overall we are far lower than the maximum achievable reward for the given problem as shown by the green curve in the following figure.



Based on how exploration is done, there are several ways to solve the MABP. Next, we will discuss some possible solution strategies.

No Exploration (Greedy Approach)

A naïve approach could be to calculate the q, or action value function, for all arms at each timestep. From that point onwards, select an action which gives the maximum q. The action values for each action will be stored at each timestep by the following function:

$$Q_t(a) = \frac{\sum_{i=1}^{t} 1_{(a_t=a)} . R_i}{\sum_{i=1}^{t} 1_{(a_t=a)}}$$

It then chooses the action at each timestep that maximises the above expression, given by:

$$argmax_a \, Q_t(a)$$

However, for evaluating this expression at each time t, we will need to do calculations over the whole history of rewards. We can avoid this by doing a running sum. So, at each time t, the q-value for each action can be calculated using the reward:

$$Q_t(a) = \frac{Q_{t-1}(a)N_t(a_t) + R_t.1_{(a_t=a)}}{N_t(a_t)}$$

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{N_t(a_t)}(R_t - Q_{t-1}(a))$$

The problem here is this approach only exploits, as it always picks the same action without worrying about exploring other actions that might return a better reward. Some exploration is necessary to actually find an optimal arm, otherwise we might end up pulling a suboptimal arm forever.

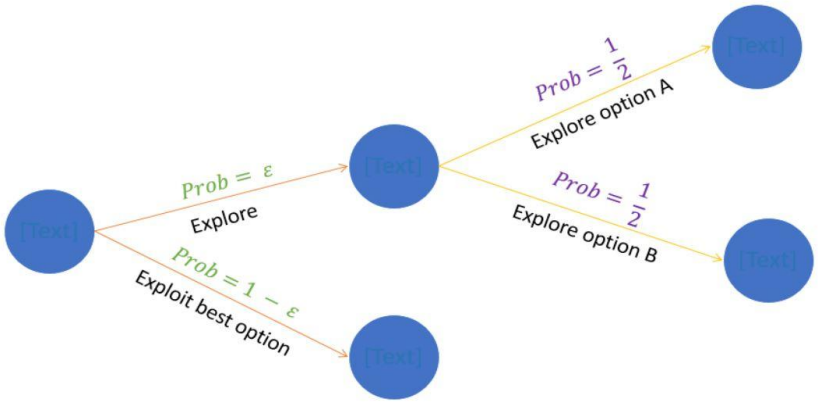
Epsilon Greedy Approach

One potential solution could be to now, and we can then explore new actions so that we ensure we are not missing out on a better choice of arm. With epsilon probability, we will choose a random action (exploration) and choose an action with maximum $q_t(a)$ with probability 1-epsilon.

With probability 1- epsilon – we choose action with maximum value (argmaxa Qt(a))

With probability epsilon – we randomly choose an action from a set of all actions A

For example, if we have a problem with two actions – A and B, the epsilon greedy algorithm works as shown below:



This is much better than the greedy approach as we have an element of exploration here. However, if two actions have a very minute difference between their q values, then even this algorithm will choose only that action which has a probability higher than the others.

Softmax Exploration

The solution is to make the probability of choosing an action proportional to q. This can be done using the softmax function, where the probability of choosing action *a* at each step is given by the following expression:

$$P(a_t = a) = \frac{e^{Q_t(a)}}{\sum_1^n e^{Q_t(b)}}$$

Decayed Epsilon Greedy

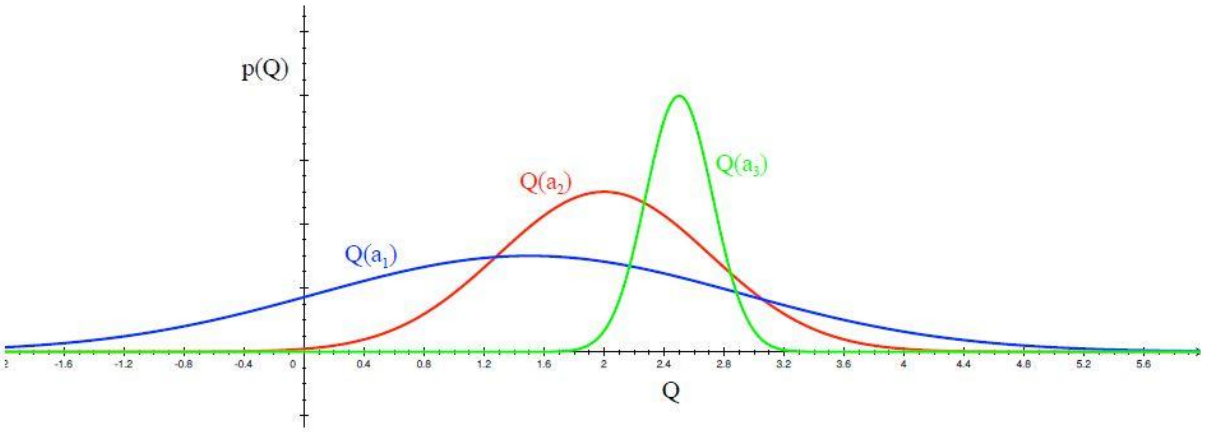
The value of epsilon is very important in deciding how well the epsilon greedy works for a given problem. We can avoid setting this value by keeping epsilon dependent on time. For example, epsilon can be kept equal to *1/log(t+0.00001)*. It will keep reducing as time passes, to the point where we starting exploring less and less as we become more confident of the optimal action or arm.

The problem with random selection of actions is that after sufficient timesteps even if we know that some arm is bad, this algorithm will keep choosing that with probability *epsilon/n*. Essentially, we are exploring a bad action which does not sound very efficient. The approach to get around this could be to favour exploration of arms with a strong potential in order to get an optimal value.

Upper Confidence Bound

Upper Confidence Bound (UCB) is the most widely used solution method for multi-armed bandit problems. This algorithm is based on the principle of optimism in the face of uncertainty.

In other words, the more uncertain we are about an arm, the more important it becomes to explore that arm.



- Distribution of action-value functions for 3 different arms a1, a2 and a3 after several trials is shown in the figure above. This distribution shows that the action value for a1 has the highest variance and hence maximum uncertainty.
- UCB says that we should choose the arm a_1 and receive a reward making us less uncertain about its action-value. For the next trial/timestep, if we still are very uncertain about a_1, we will choose it again until the uncertainty is reduced below a threshold.

The intuitive reason this works is that when acting optimistically in this way, one of two things happen:

- optimism is justified and we get a positive reward which is the objective ultimately
- the optimism was not justified. In this case, we play an arm that we believed might give a large reward when in fact it does not. If this happens sufficiently often, then we will learn what is the true payoff of this action and not choose it in the future.

UCB is actually a family of algorithms. Here, we will discuss UCB1.

Steps involved in UCB1:

- *Play each of the K actions once, giving initial values for mean rewards corresponding to each action at*
- *For each round t = K:*
- *Let $N_t(a)$ represent the number of times action a was played so far*
- *Play the action at maximising the following expression:*

$$Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$$

- *Observe the reward and update the mean reward or expected payoff for the chosen action*

We will not go into the mathematical proof for UCB. However, it is important to understand the expression that corresponds to our selected action. Remember, in the random exploration we just had Q(a) to maximise, while here we have two terms. First is the action value function, while the second is the confidence term.

- Each time *a* is selected, the uncertainty is presumably reduced: $N_t(a)$ increments, and, as it appears in the denominator, the uncertainty term decreases.

$$N_t(a) \uparrow \quad \sqrt{\frac{2 \log t}{N_t(a)}} \downarrow$$

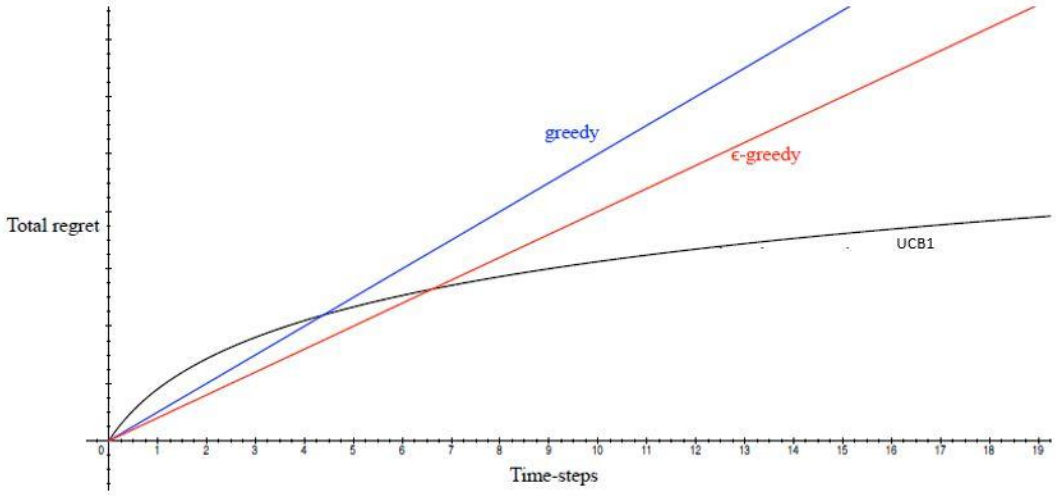
- On the other hand, each time an action other than *a* is selected, t increases, but $N_t(a)$ does not; because t appears in the numerator, the uncertainty estimate increases.

$$t \uparrow \quad \text{With } N_t(a) \text{ constant} \quad \sqrt{\frac{2 \log t}{N_t(a)}} \uparrow$$

- The use of the natural logarithm means that the increases get smaller over time; all actions will eventually be selected, but actions with lower value estimates, or that have already been selected frequently, will be selected with decreasing frequency over time.
- This will ultimately lead to the optimal action being selected repeatedly in the end.

Regret Comparison

Among all the algorithms given in this article, only the UCB algorithm provides a strategy where the regret increases as log(t), while in the other algorithms we get linear regret with different slopes.



An Algorithm for Building Decision Trees

- C4.5 is a computer program for inducing classification rules in the form of decision trees from a set of given instances
- C4.5 is a software extension of the basic ID3 algorithm designed by Quinlan

Algorithm Description

- Select one attribute from a set of training instances
- Select an initial subset of the training instances
- Use the attribute and the subset of instances to build a decision tree
- Use the rest of the training instances (those not in the subset used for construction) to test the accuracy of the constructed tree
- If all instances are correctly classified – stop
- If an instances is incorrectly classified, add it to the initial subset and construct a new tree
- Iterate until
  - A tree is built that classifies all instance correctly
  - OR
  - A tree is built from the entire training set

Simplified Algorithm

- Let T be the set of training instances
- Choose an attribute that best differentiates the instances contained in T (C4.5 uses the Gain Ratio to determine)
- Create a tree node whose value is the chosen attribute
  - Create child links from this node where each link represents a unique value for the chosen attribute
  - Use the child link values to further subdivide the instances into subclasses

Example

Credit Card Promotion Data from Chapter 2

Example – Credit Card Promotion Data Descriptions

Attribute Name	Value Description	Numeric Values	Definition
Income Range	20-30K, 30-40K, 40-50K, 50-60K	20000, 30000, 40000, 50000	Salary range for an individual credit card holder
Magazine Promotion	Yes, No	1, 0	Did card holder participate in magazine promotion offered before?
Watch Promotion	Yes, No	1, 0	Did card holder participate in watch promotion offered before?
Life Ins Promotion	Yes, No	1, 0	Did card holder participate in life insurance promotion offered before?
Credit Card Insurance	Yes, No	1, 0	Does card holder have credit card insurance?
Sex	Male, Female	1, 0	Card holder's gender
Age	Numeric	Numeric	Card holder's age in whole years

Problem to be Solved from Data

- Acme Credit Card Company is going to do a life insurance promotion – sending the promo materials with billing statements. They have done a similar promotion in the past, with results as represented by the data set. They want to target the new promo materials to credit card holders similar to those who took advantage of the prior life insurance promotion.
- Use supervised learning with output attribute = life insurance promotion to develop a profile for credit card holders likely to accept the new promotion.

Sample of Credit Card Promotion Data (from Table 2.3)

Income Range	Magazine Promo	Watch Promo	Life Ins Promo	CC Ins	Sex	Age
40-50K	Yes	No	No	No	Male	45
30-40K	Yes	Yes	Yes	No	Female	40
40-50K	No	No	No	No	Male	42
30-40K	Yes	Yes	Yes	Yes	Male	43
50-60K	Yes	No	Yes	No	Female	38
20-30K	No	No	No	No	Female	55
30-40K	Yes	No	Yes	Yes	Male	35
20-30K	No	Yes	No	No	Male	27
30-40K	Yes	No	No	No	Male	43
30-40K	Yes	Yes	Yes	No	Female	41

Problem Characteristics

- Life insurance promotion is the output attribute
- Input attributes are income range, credit card insurance, sex, and age
  - Attributes related to the instance's response to other promotions is not useful for prediction because new credit card holders will not have had a chance to take advantage of these prior offers (except for credit card insurance which is always offered immediately to new card holders)
  - Therefore, magazine promo and watch promo are not relevant for solving the problem at hand – disregard – do not include this data in data mining

Apply the Simplified C4.5 Algorithm to the Credit Card Promotion Data

Income Range	Magazine Promo	Watch Promo	Life Ins Promo	CC Ins	Sex	Age
40-50K	Yes	No	No	No	Male	45
30-40K	Yes	Yes	Yes	No	Female	40
40-50K	No	No	No	No	Male	42
30-40K	Yes	Yes	Yes	Yes	Male	43
50-60K	Yes	No	Yes	No	Female	38
20-30K	No	No	No	No	Female	55
30-40K	Yes	No	Yes	Yes	Male	35
20-30K	No	Yes	No	No	Male	27
30-40K	Yes	No	No	No	Male	43
30-40K	Yes	Yes	Yes	No	Female	41

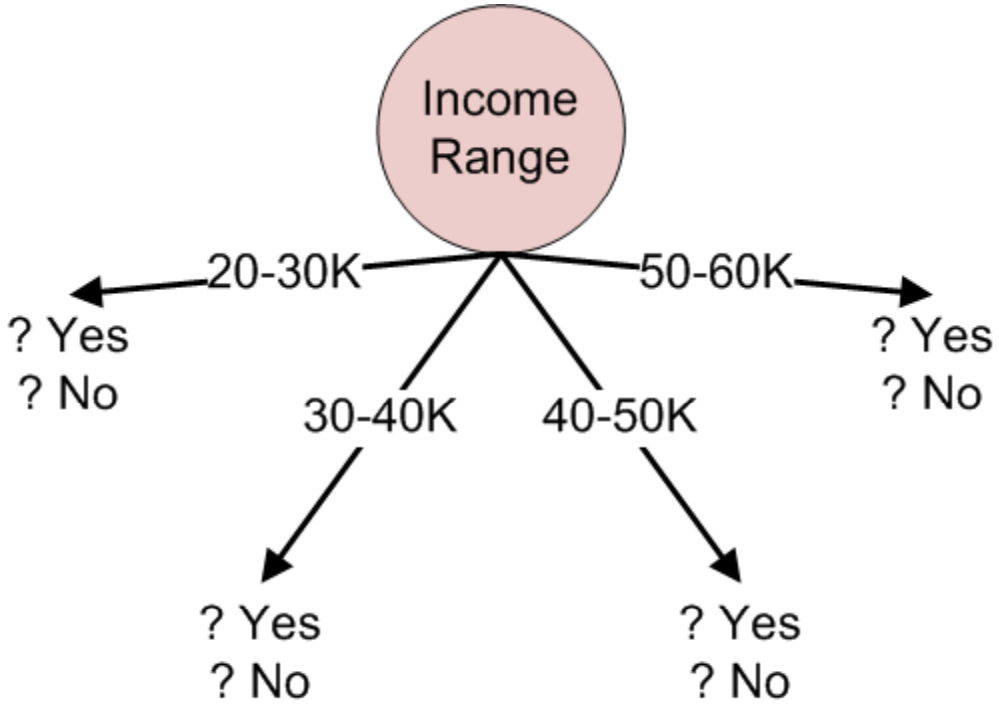
Training set = 15 instances (see handout)

Apply the Simplified C4.5 Algorithm to the Credit Card Promotion Data

Income Range	Magazine Promo	Watch Promo	Life Ins Promo	CC Ins	Sex	Age
40-50K	Yes	No	No	No	Male	45
30-40K	Yes	Yes	Yes	No	Female	40
40-50K	No	No	No	No	Male	42
30-40K	Yes	Yes	Yes	Yes	Male	43
50-60K	Yes	No	Yes	No	Female	38
20-30K	No	No	No	No	Female	55
30-40K	Yes	No	Yes	Yes	Male	35
20-30K	No	Yes	No	No	Male	27
30-40K	Yes	No	No	No	Male	43
30-40K	Yes	Yes	Yes	No	Female	41

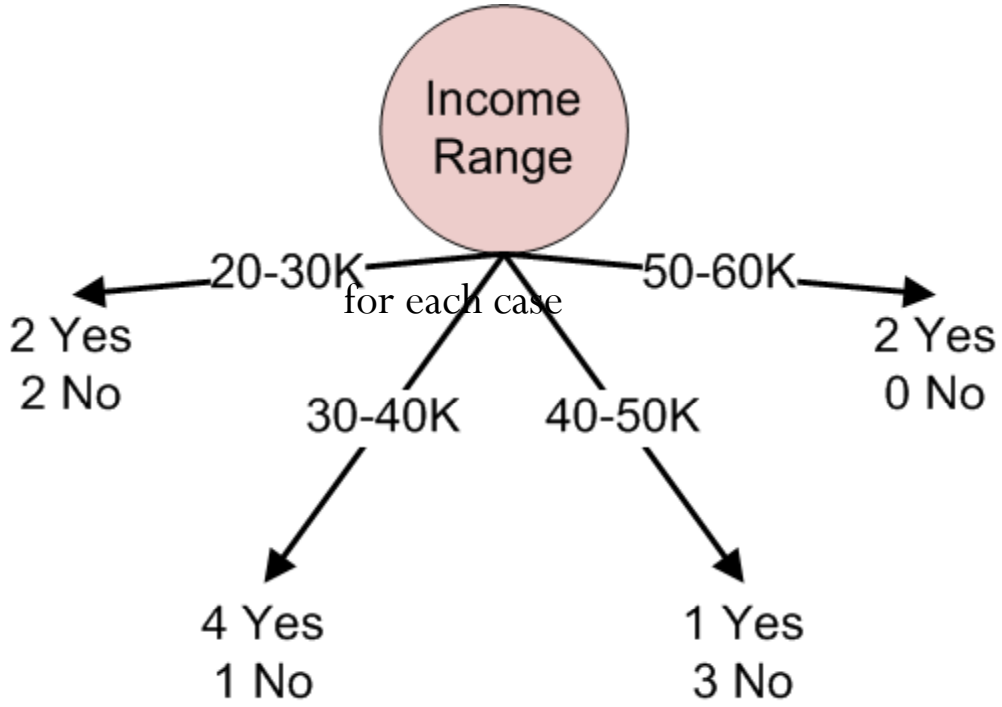
Step 2: Which input attribute best differentiates the instances?

Apply Simplified C4.5



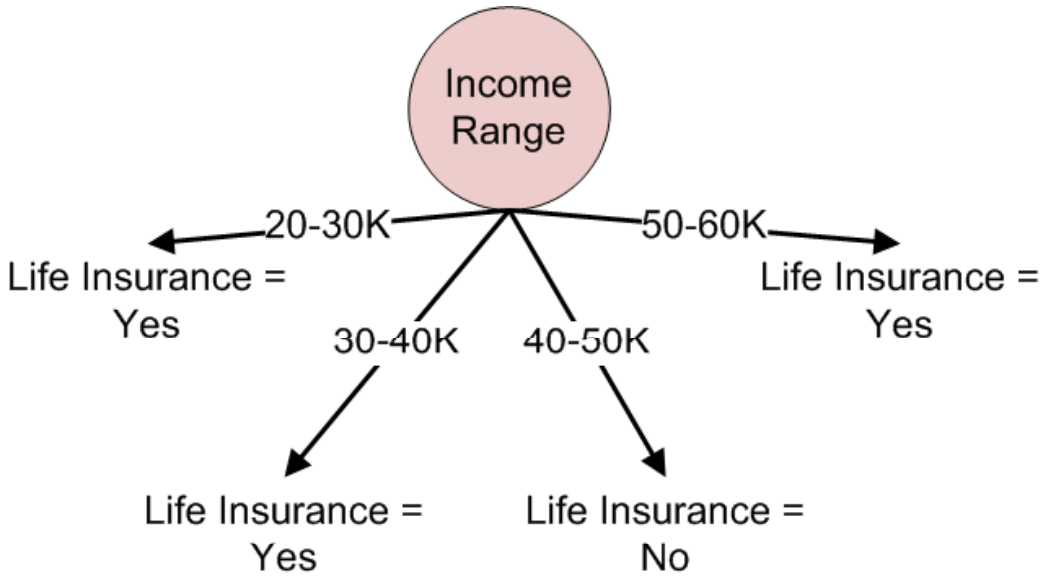
For each case (attribute value), how many instances of Life Insurance Promo = Yes and Life Insurance Promo = No?

Apply Simplified C4.5



For each branch, choose the most frequently occurring decision. If there is a tie, then chooseYes, since there are more overall Yes instances (9) than No instances (6) with respect to Life Insurance Promo

Apply Simplified C4.5

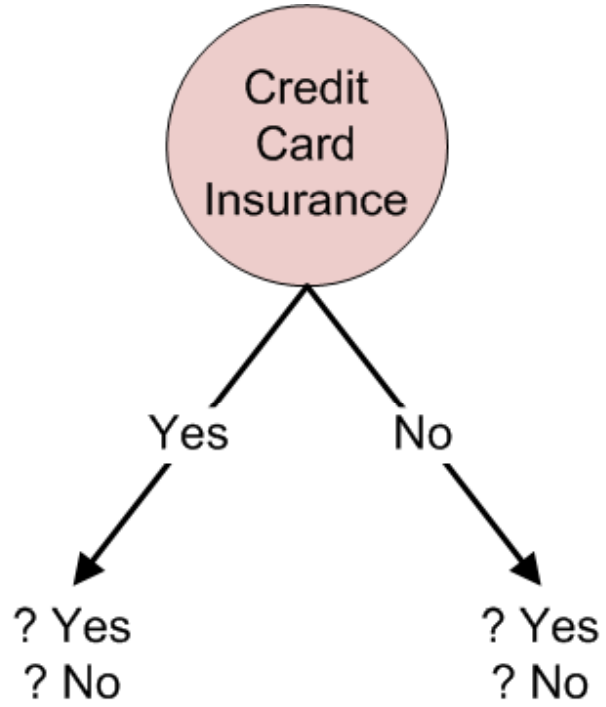


Evaluate the classification model (the tree) on the basis of accuaracy. How many of the 15 training instances are classified correctly by this tree?

Apply Simplified C4.5

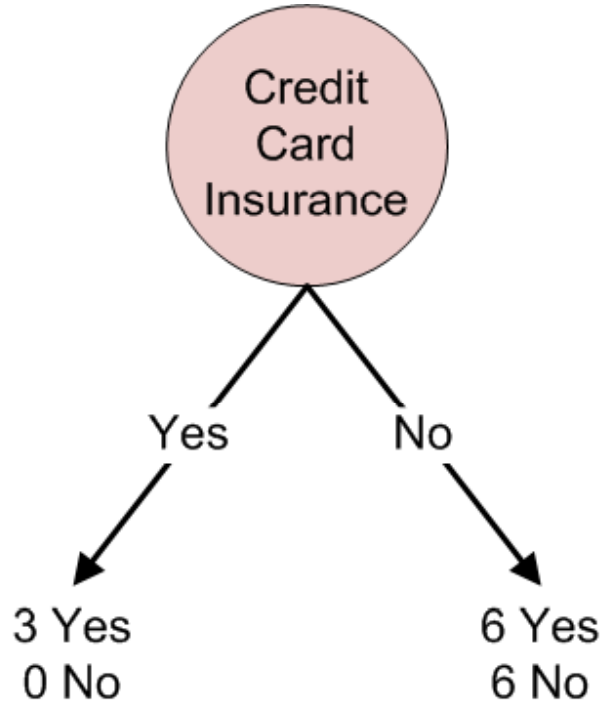
- Tree accuracy = 11/15 = 73.3%
- Tree cost = 4 branches for the computer program to use
- Goodness score for Income Range attribute is 11/15/4 = 0.183
- Including Tree "cost" to assess goodness lets us compare trees

Apply Simplified C4.5
Consider a Different Top-Level Node



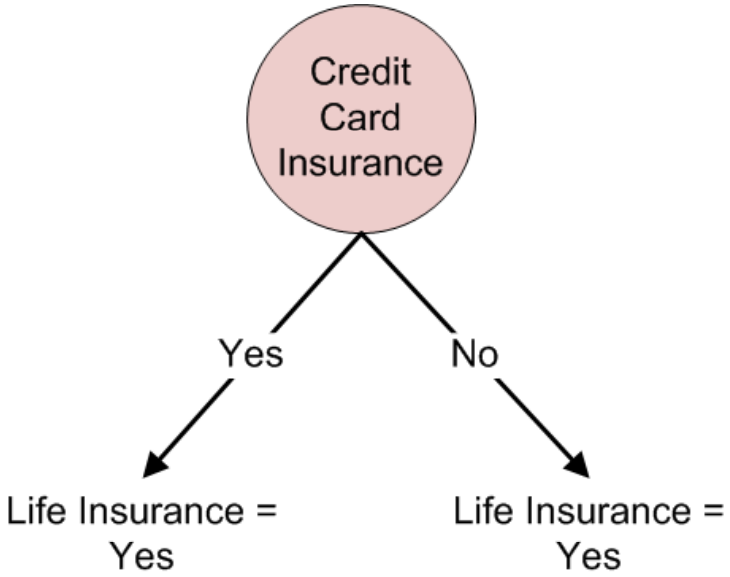
For each case (attribute value), how many instances of Life Insurance Promo = Yes and Life Insurance Promo = No?

Apply Simplified C4.5



For each branch, choose the most frequently occurring decision. If there is a tie, then chooseYes, since there are more total Yes instances (9) than No instances (6).

Apply Simplified C4.5

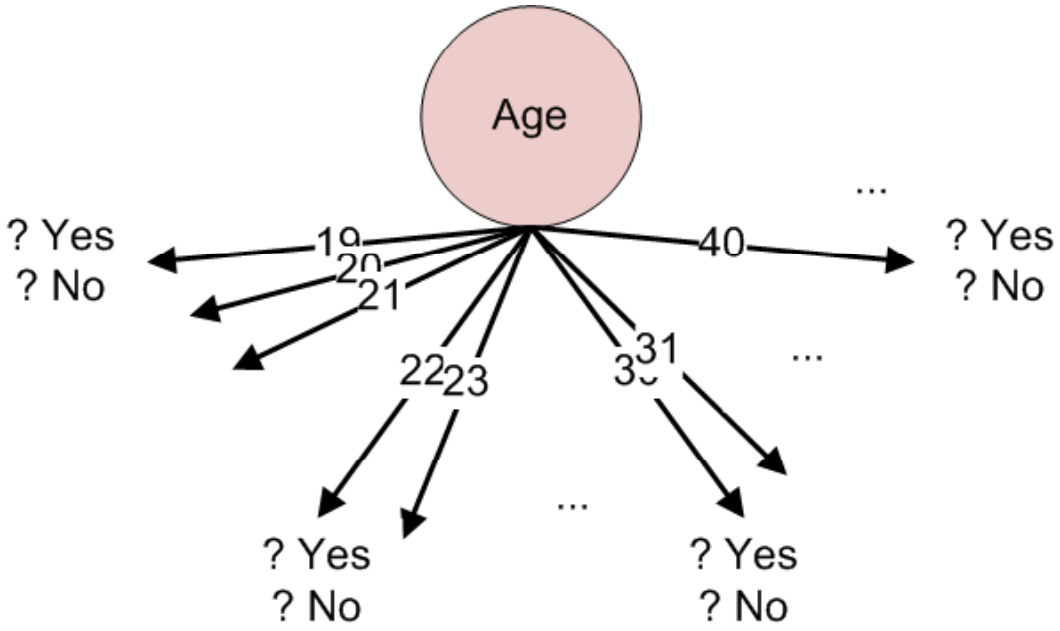


Evaluate the classification model (the tree). How many of the 15 training instances are classified correctly by this tree?

Apply Simplified C4.5

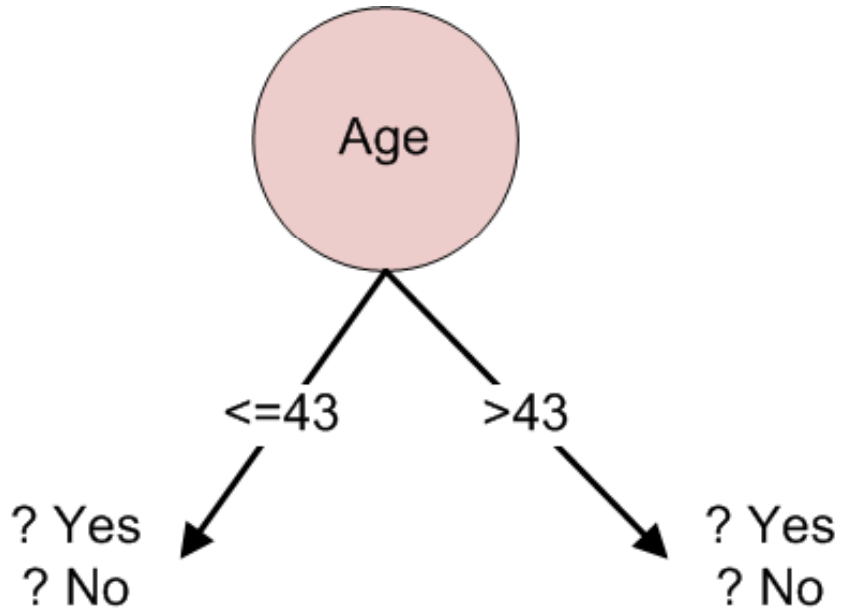
- Tree accuracy = 9/15 = 60.0%
- Tree cost = 2 branches for the computer program to use
- Goodness score for Income Range attribute is 9/15/2 = 0.300
- Including Tree "cost" to assess goodness lets us compare trees

Apply Simplified C4.5



What's problematic about this?

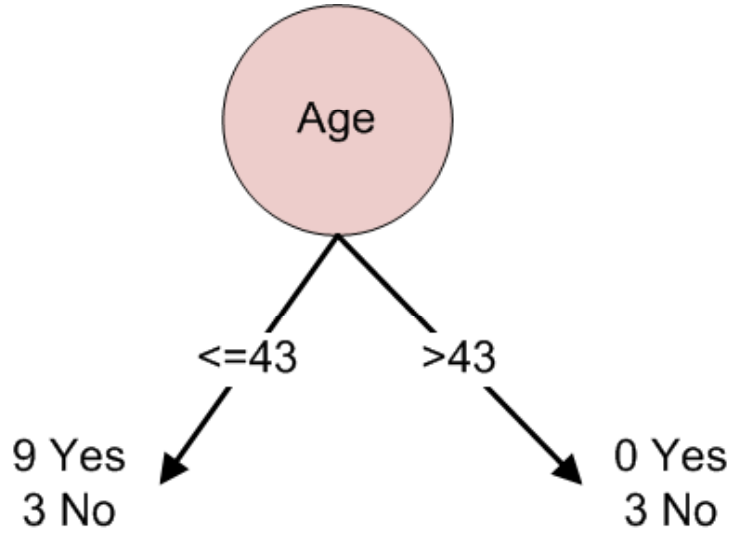
Apply Simplified C4.5



How many instances for each case?

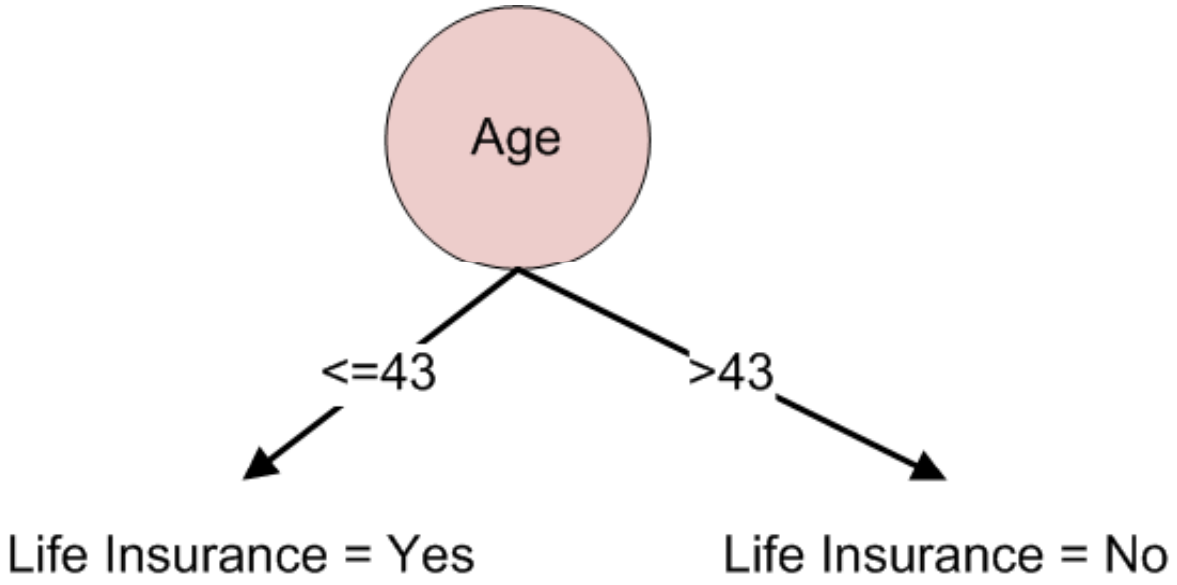
A binary split requires the addition of only two branches. Why 43?

Apply Simplified C4.5



For each branch, choose the most frequently occurring decision. If there is a tie, then choose Yes, since there are more total Yes instances (9) than No instances (6).

Apply Simplified C4.5

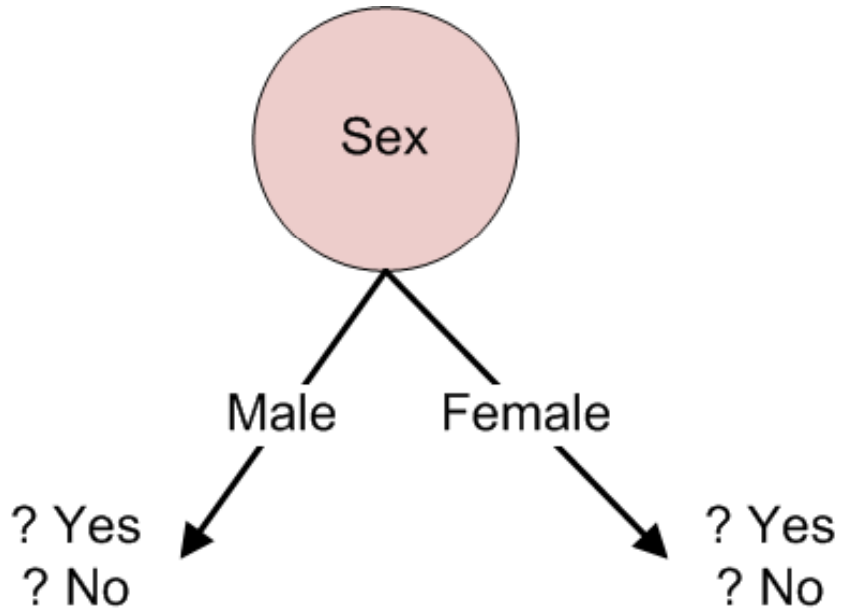


For this data, a binary split at 43 results in the best "score".

Apply Simplified C4.5

- Tree accuracy = 12/15 = 80.0%
- Tree cost = 2 branches for the computer program to use
- Goodness score for Income Range attribute is 12/15/2 = 0.400
- Including Tree "cost" to assess goodness lets us compare trees

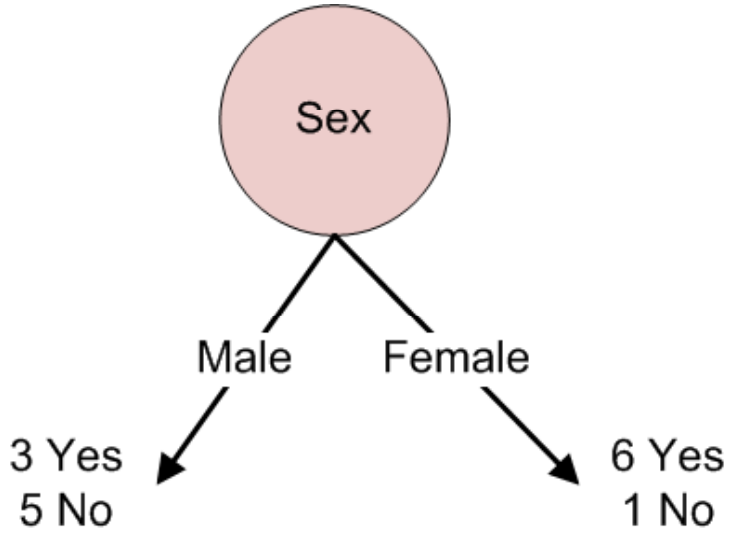
Apply Simplified C4.5



How many instances for each case?

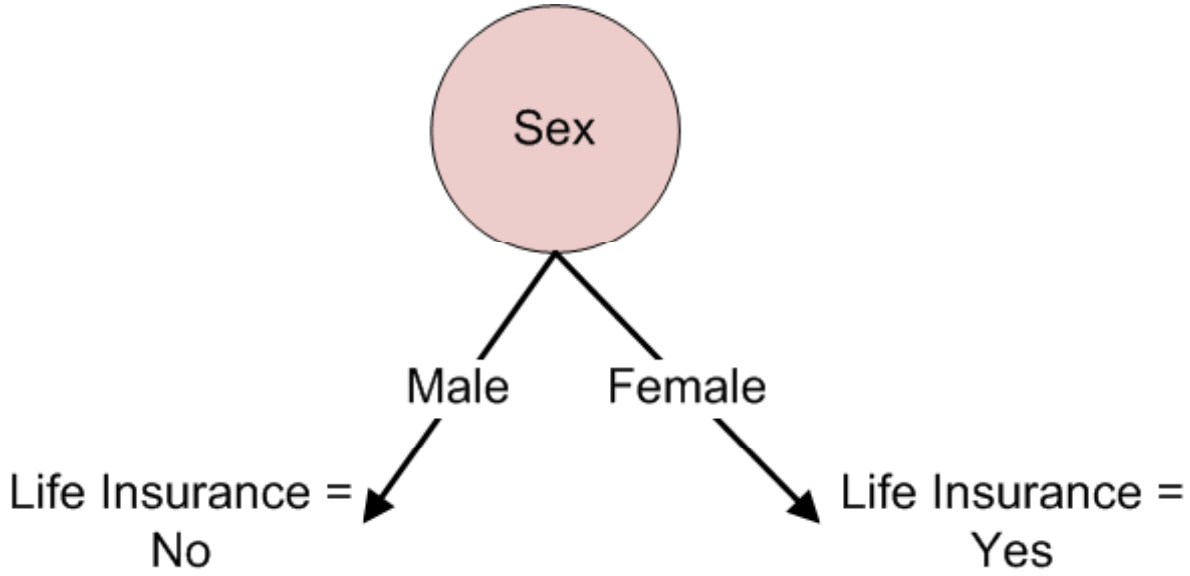
A binary split requires the addition of only two branches. Why 43?

Apply Simplified C4.5



For each branch, choose the most frequently occurring decision. If there is a tie, then choose Yes, since there are more total Yes instances (9) than No instances (6).

Apply Simplified C4.5



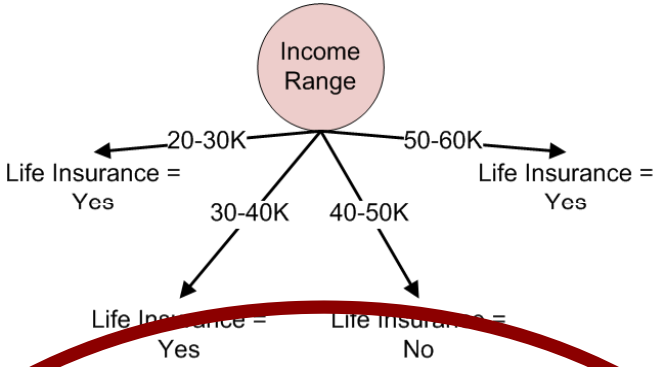
Evaluate the classification model (the tree). How many of the 15 training instances are classified correctly by this tree?

Apply Simplified C4.5

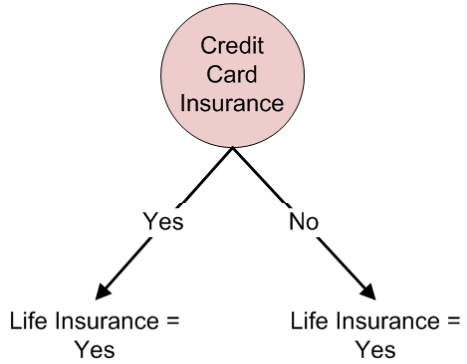
- Tree accuracy = 11/15 = 73.3%
- Tree cost = 2 branches for the computer program to use
- Goodness score for Income Range attribute is 11/15/2 = 0.367
- Including Tree "cost" to assess goodness lets us compare trees

Apply Simplified C4.5

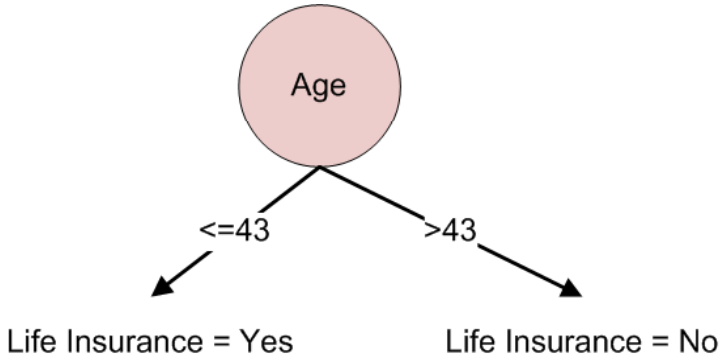
Model "goodness" = 0.183



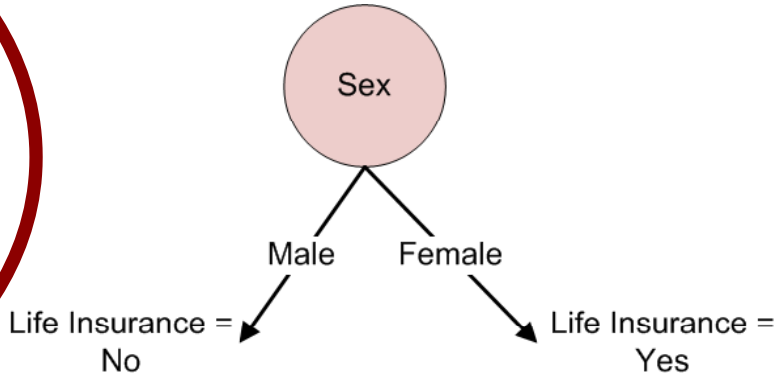
Model "goodness" = 0.30



Model "goodness" = 0.40



Model "goodness" = 0.367

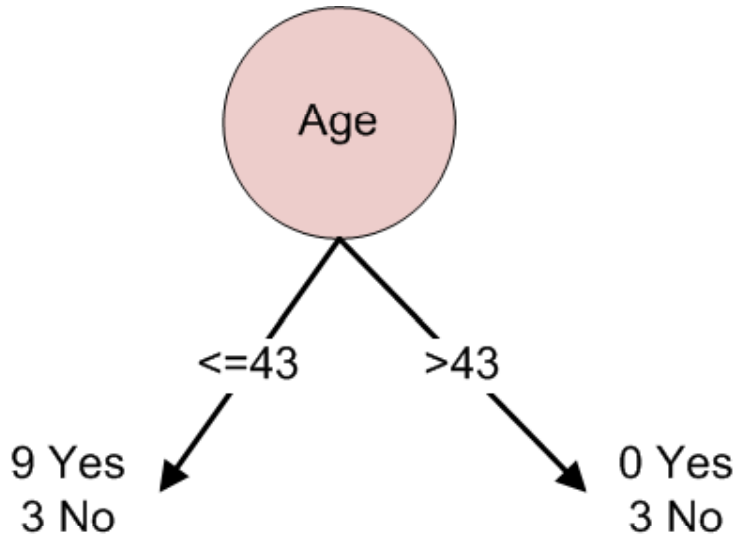


Apply Simplified C4.5

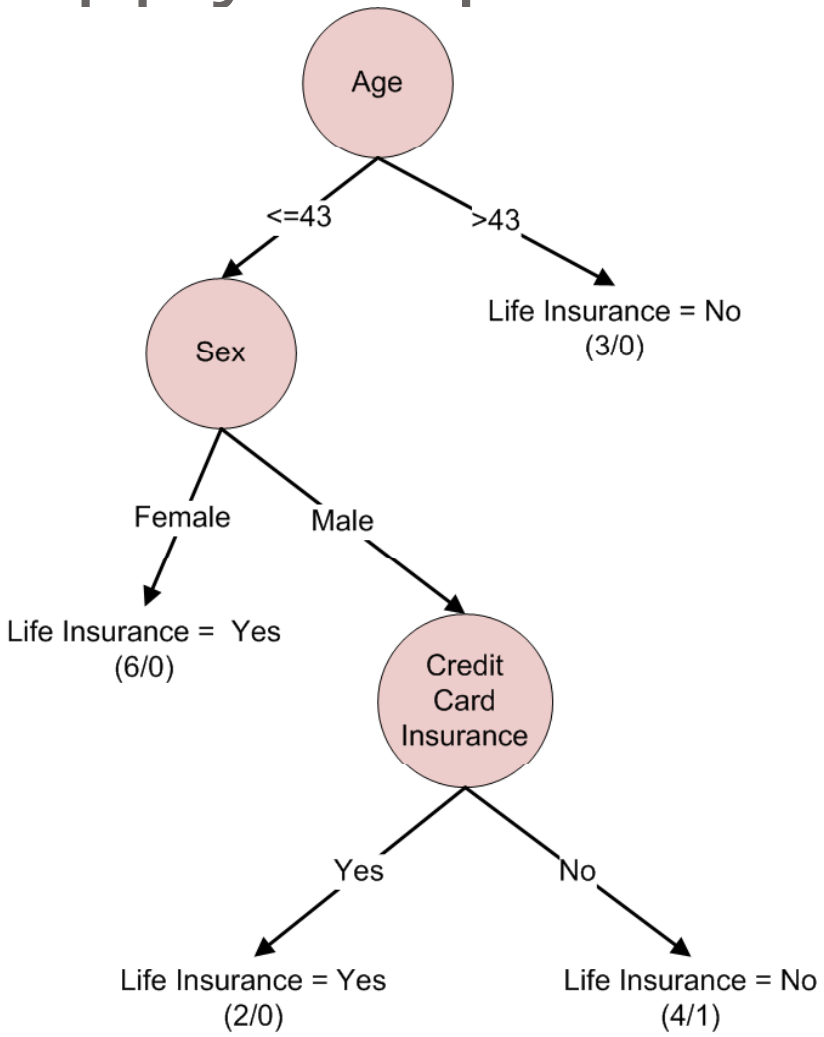
- Consider each branch and decide whether to terminate or add an attribute for further classification
- Different termination criteria make sense
  - If the instances following a branch satisfy a predetermined criterion, such as a certain level of accuracy, then the branch becomes a terminal path
  - No other attribute adds information

Apply Simplified C4.5

- 100% accuracy for >43 branch

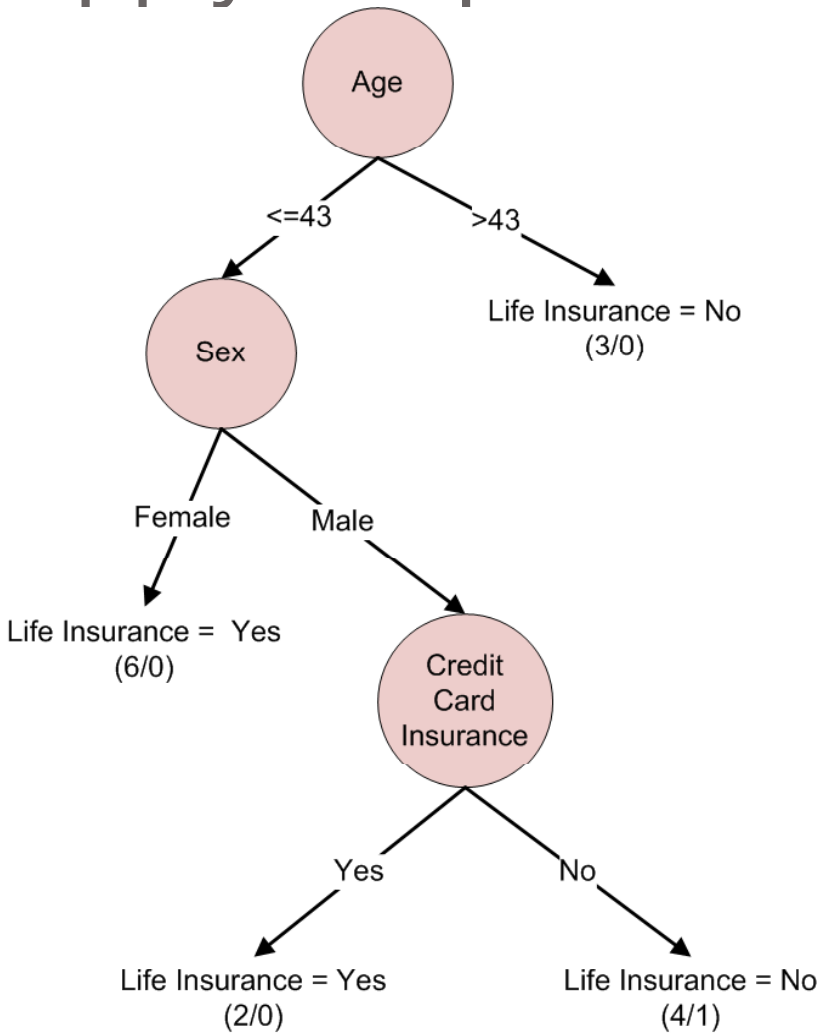


Apply Simplified C4.5



- Production rules are generated by following to each terminal branch

Apply Simplified C4.5



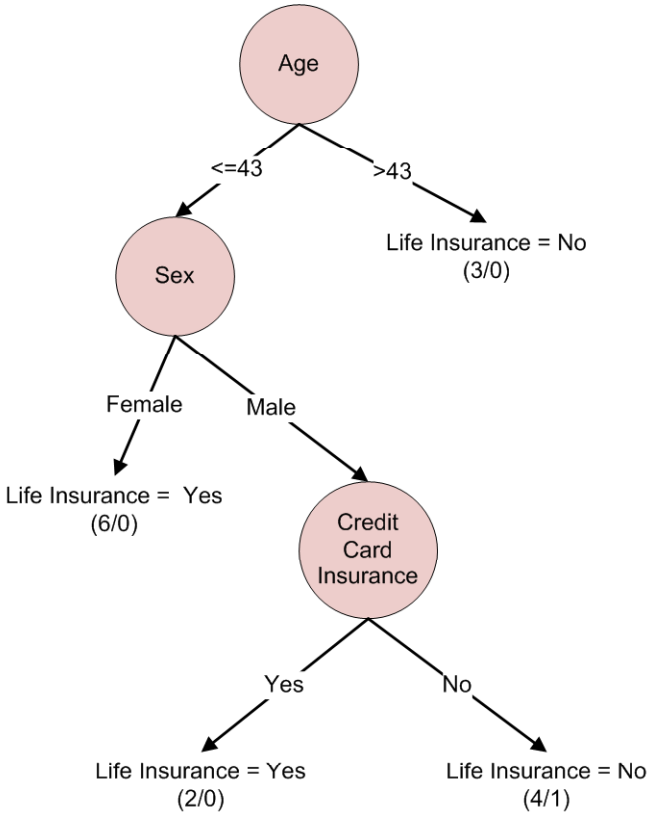
If Age <=43 AND Sex = Male AND CCIns = No

Then Life Insurance Promo = No

Accuracy = 75%

Coverage = 26.7%

Apply Simplified C4.5



Simplify the Rule

If Sex = Male AND CCIns = No

Then Life Insurance Promo = No

Accuracy = 83.3%

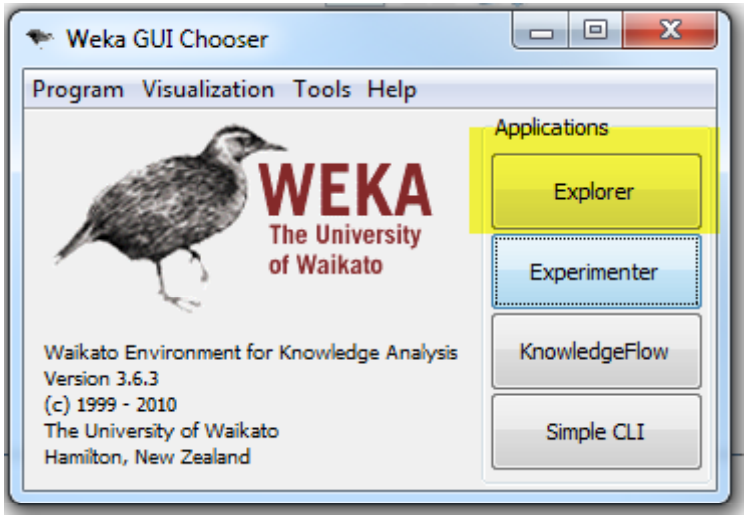
Coverage = 40.0%

This rule is more general, more accurate

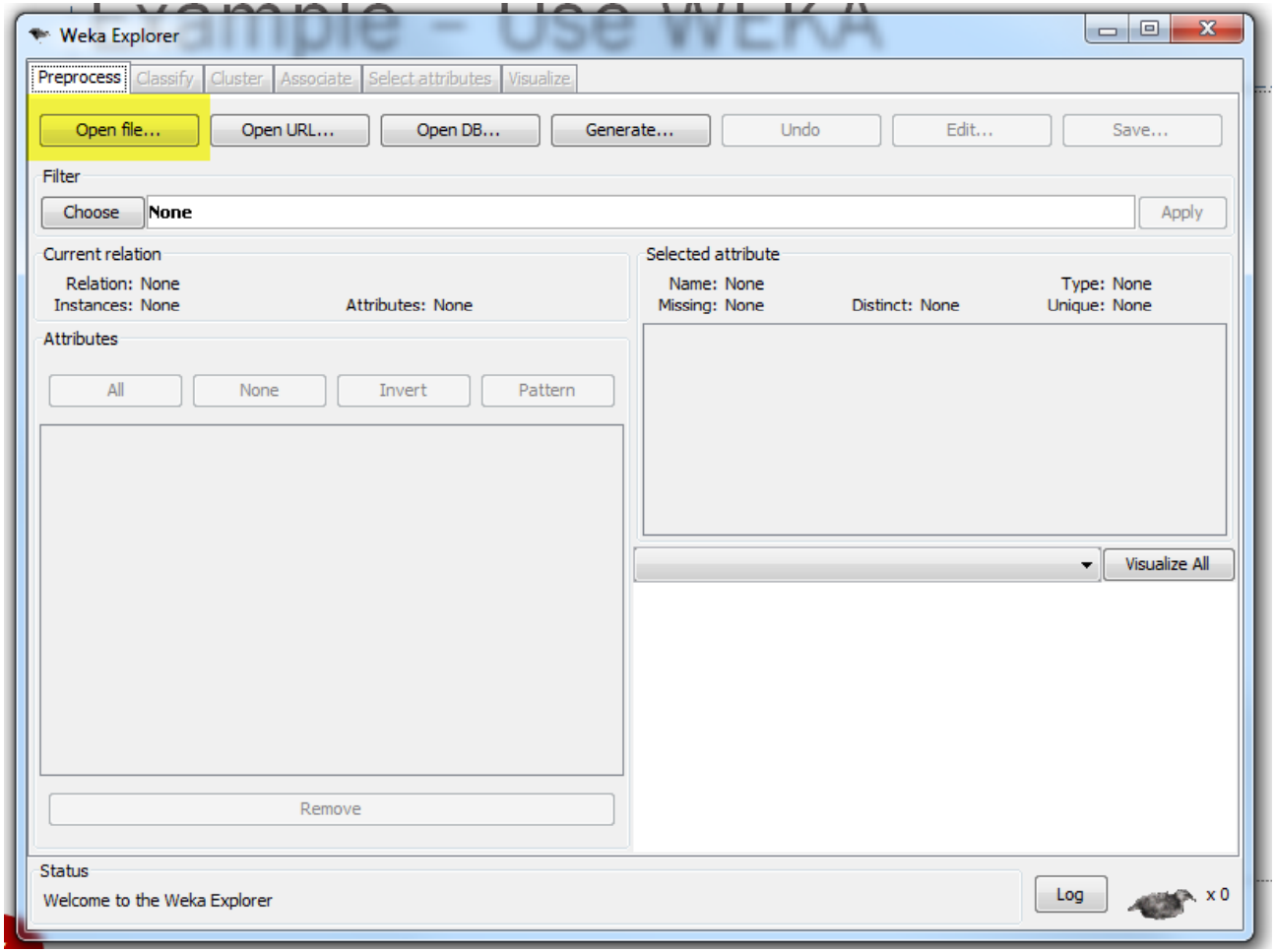
Decision Tree Algorithm Implementations

- Automate the process of rule creation
- Automate the process of rule simplification
- Choose a default rule – the one that states the classification of an instance that does not meet the preconditions of any listed rule

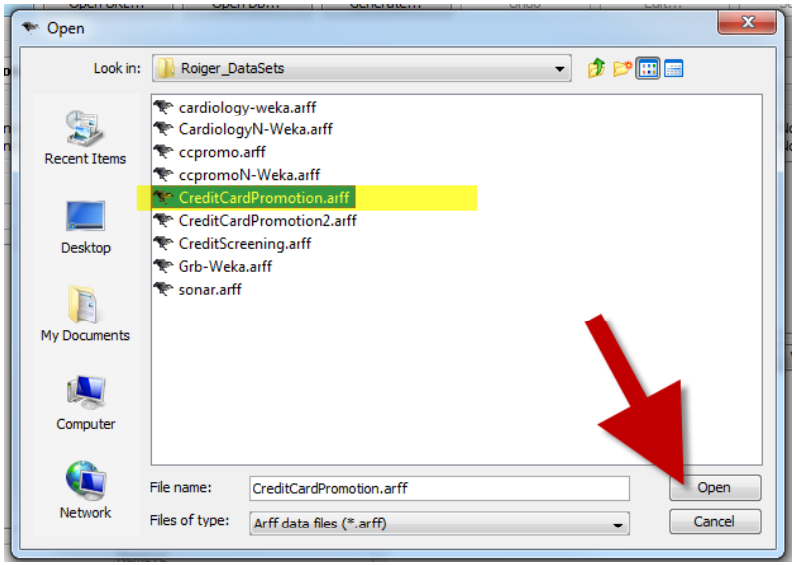
Example – Use WEKA



Example – Use WEKA

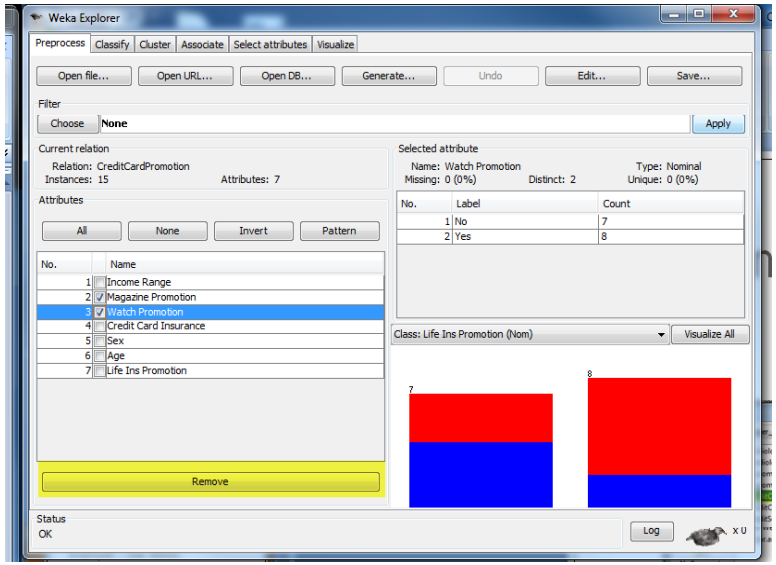


Example – Use WEKA



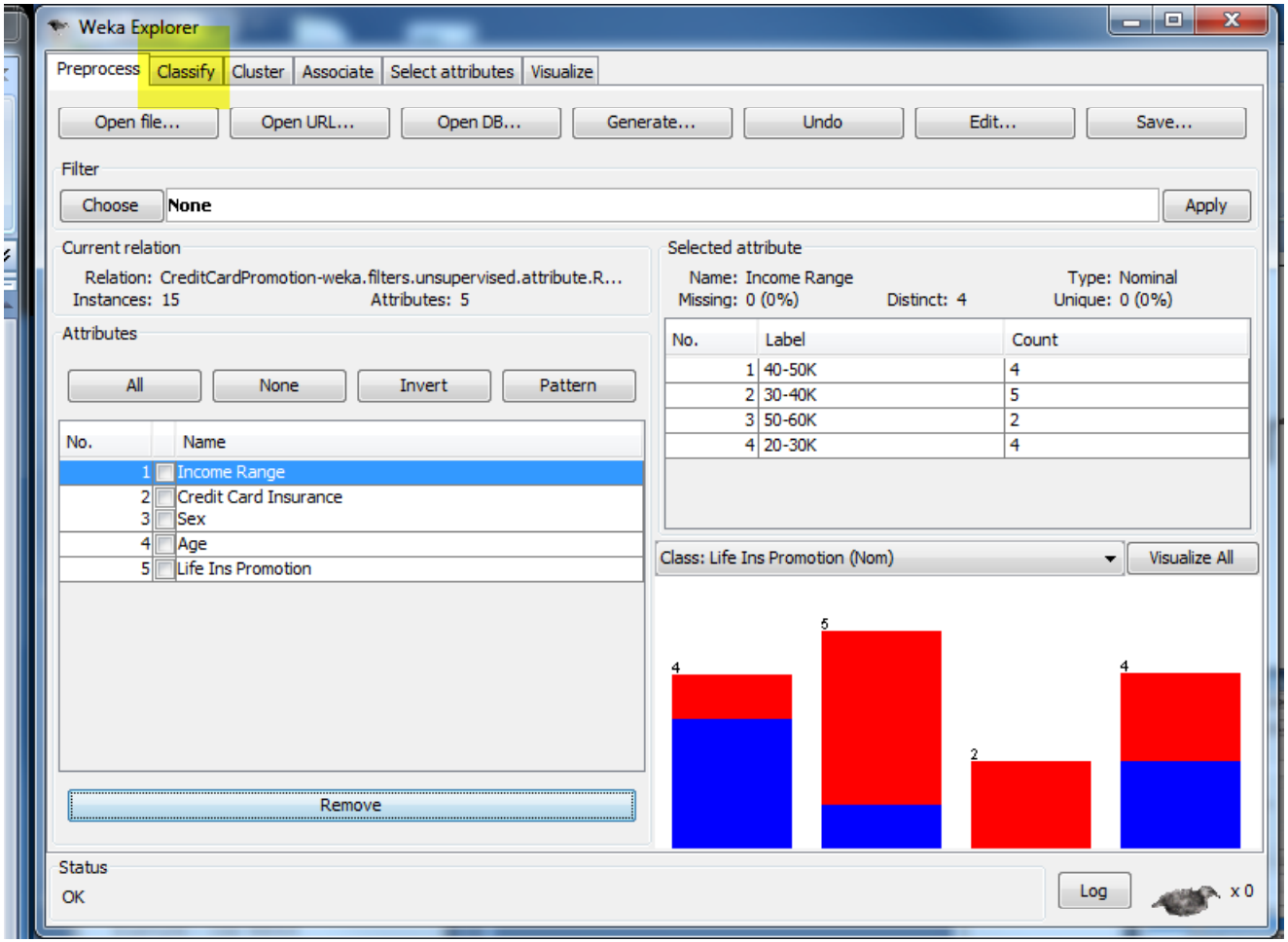
- Download CreditCardPromotion.zip from Blackboard and extract CreditCardPromotion.arff

Example – Use WEKA



- Why remove magazine promotion and watch promotion from the analysis?

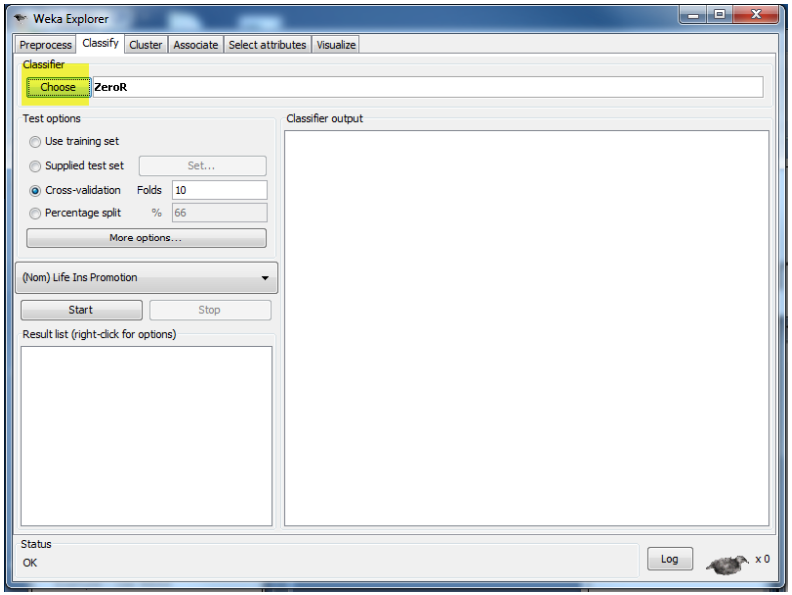
Example – Use WEKA

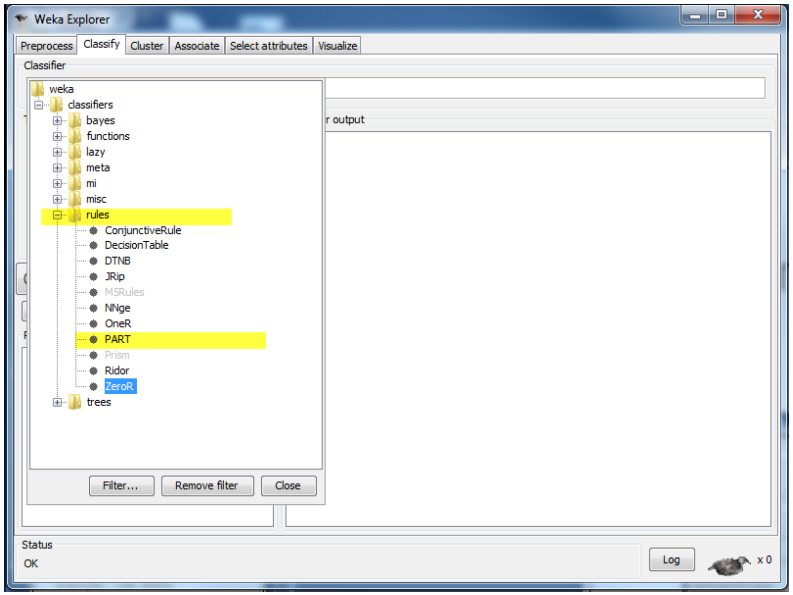


Example – Use WEKA

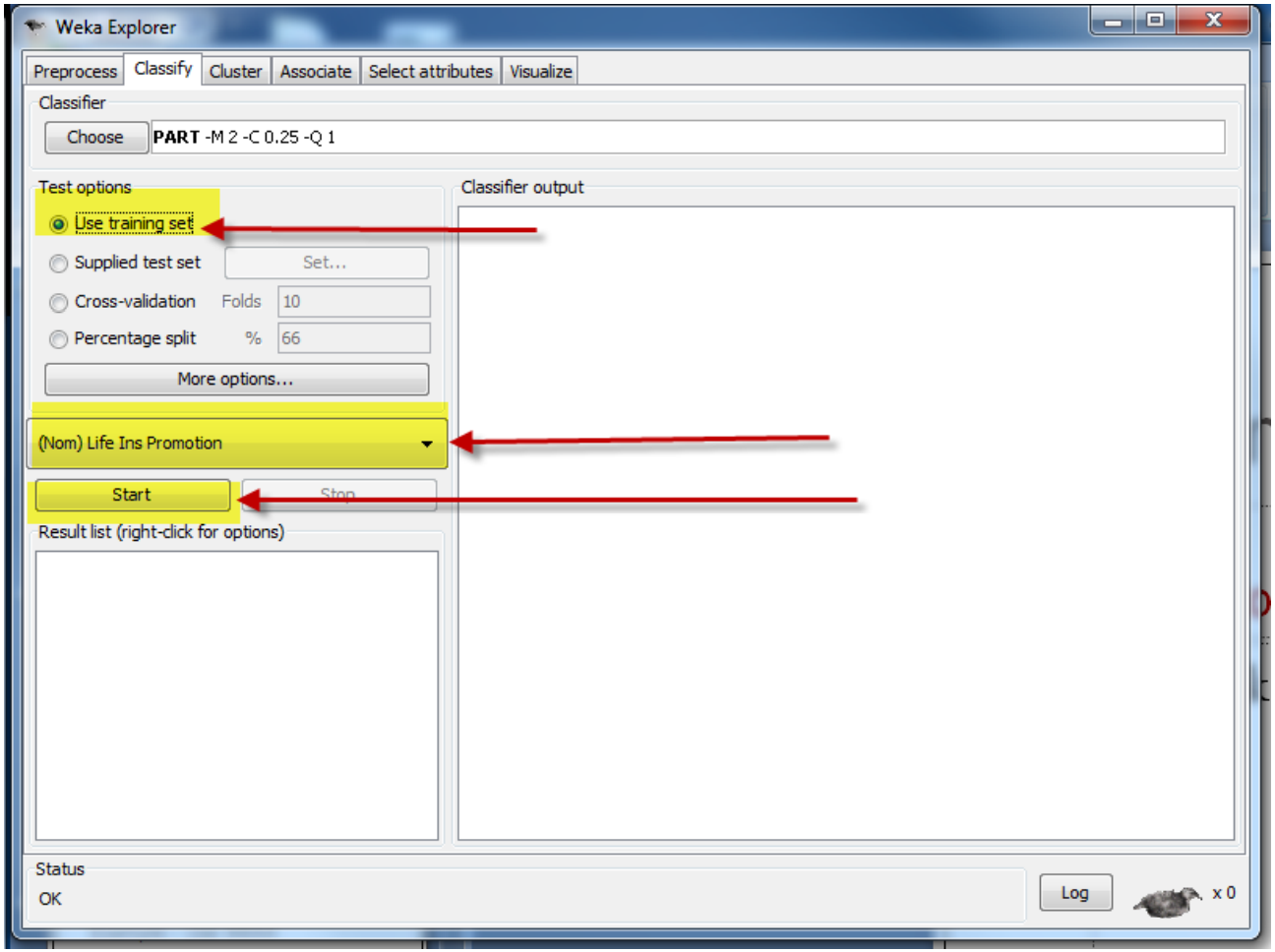
See algorithm options through Choose

Choose PART under rules

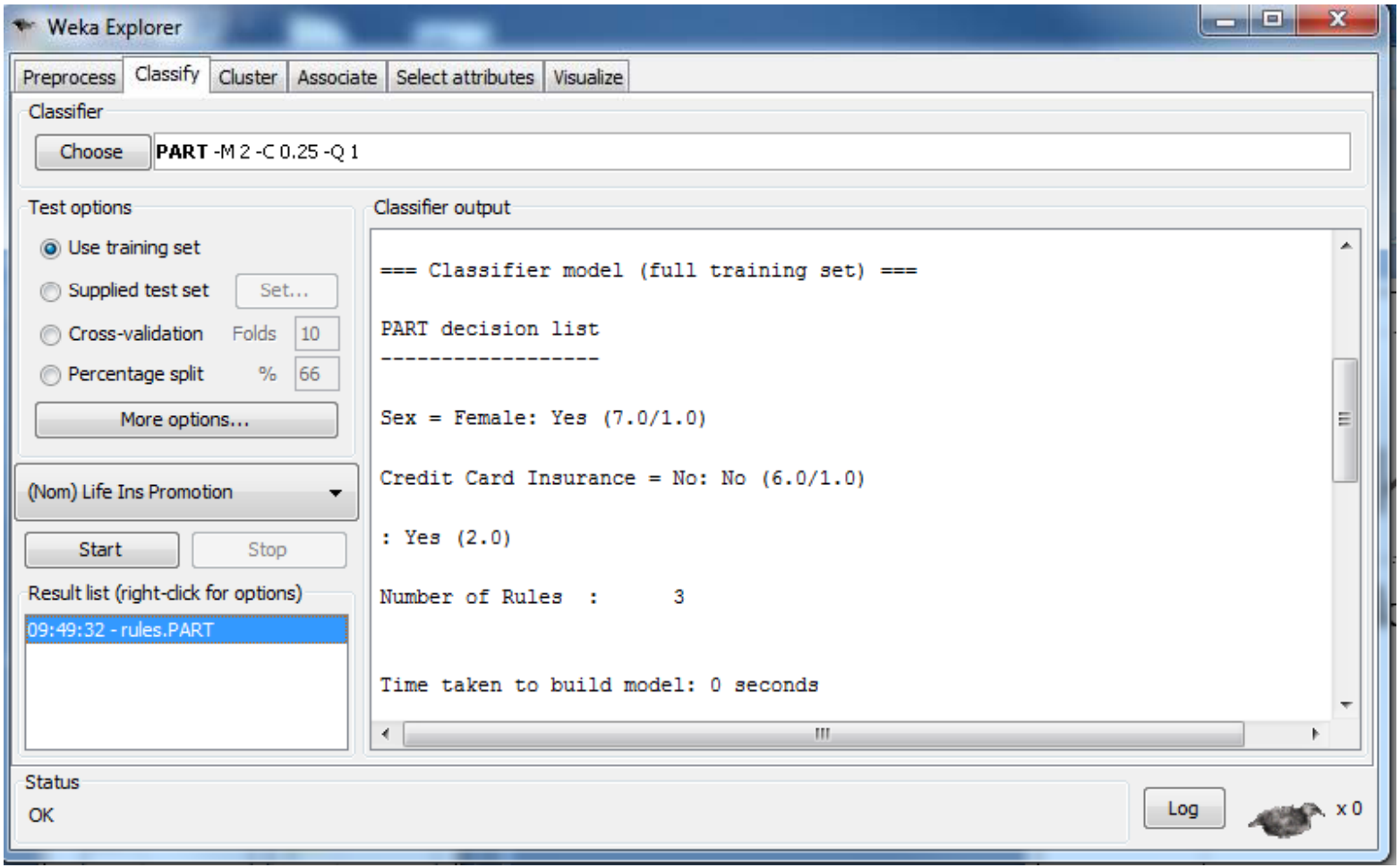




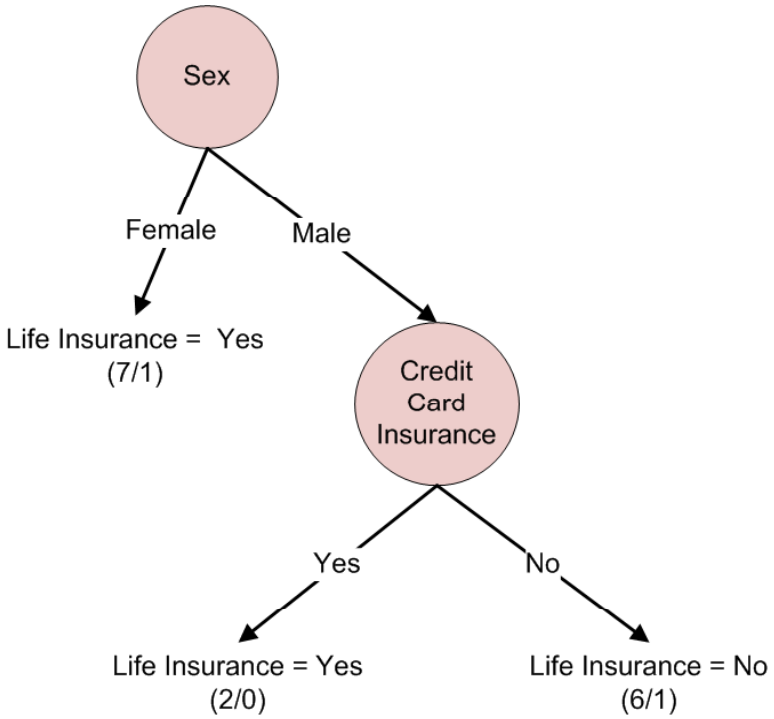
Example – Use WEKA



Example – Use WEKA

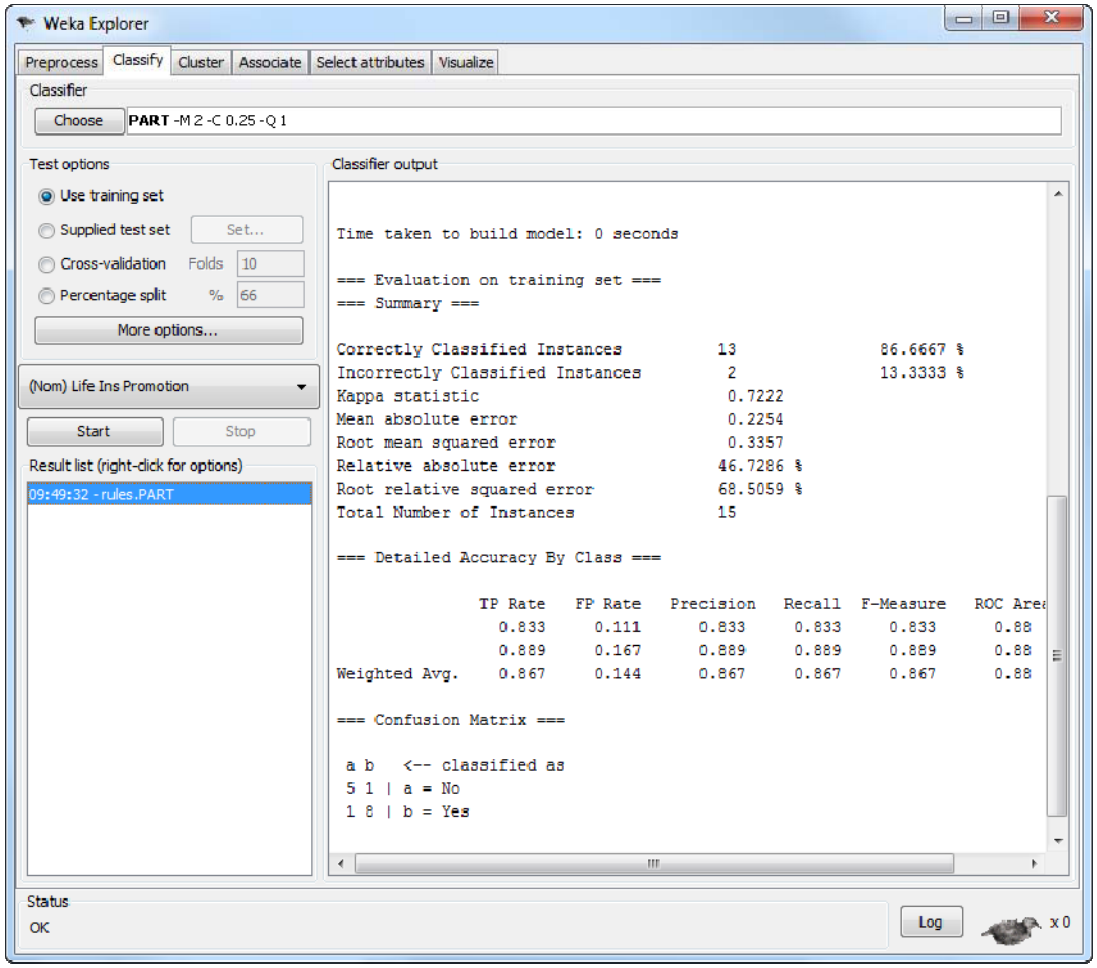


Example – Use WEKA



- Decision tree equivalent of rules generated by PART

Example – Use WEKA



Decision Trees – Advantages

Pluses

- Easy to understand
- Map readily to production rules
- No prior assumptions about the nature of the data needed
  - e.g., no assumption of normally distributed data needed
- Apply to categorical data, but numerical data can be binned for application

Issues

- Output attribute must be categorical
- Only one output attribute
- Sufficiently robust?
  - Change in one training set data item can change outcome
- Numerical attributes can create complex decision trees (due to split algorithms)