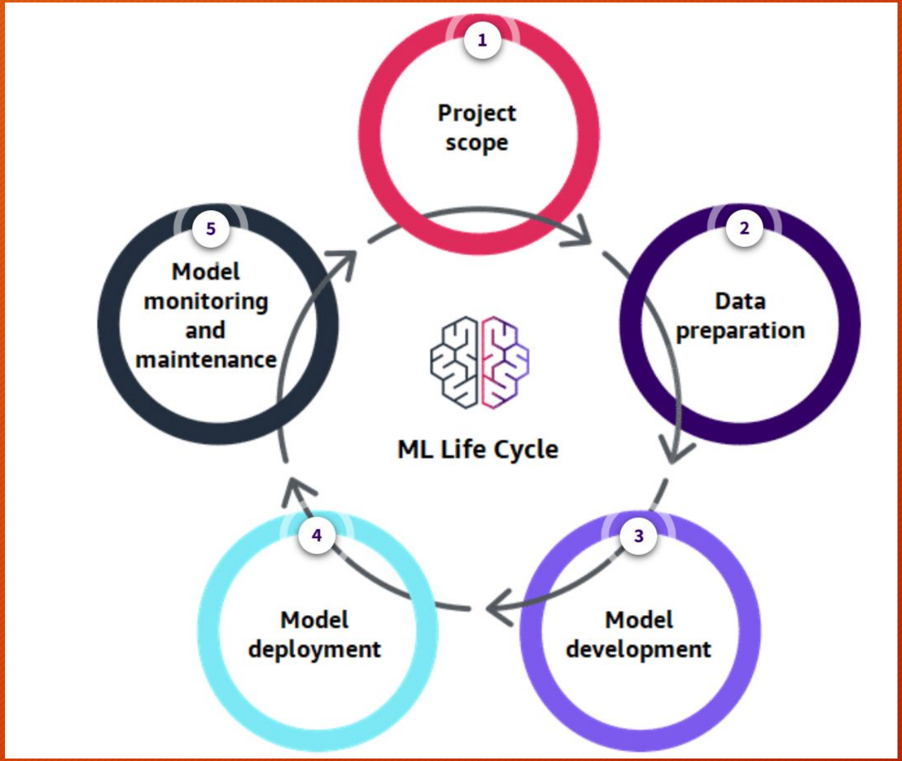
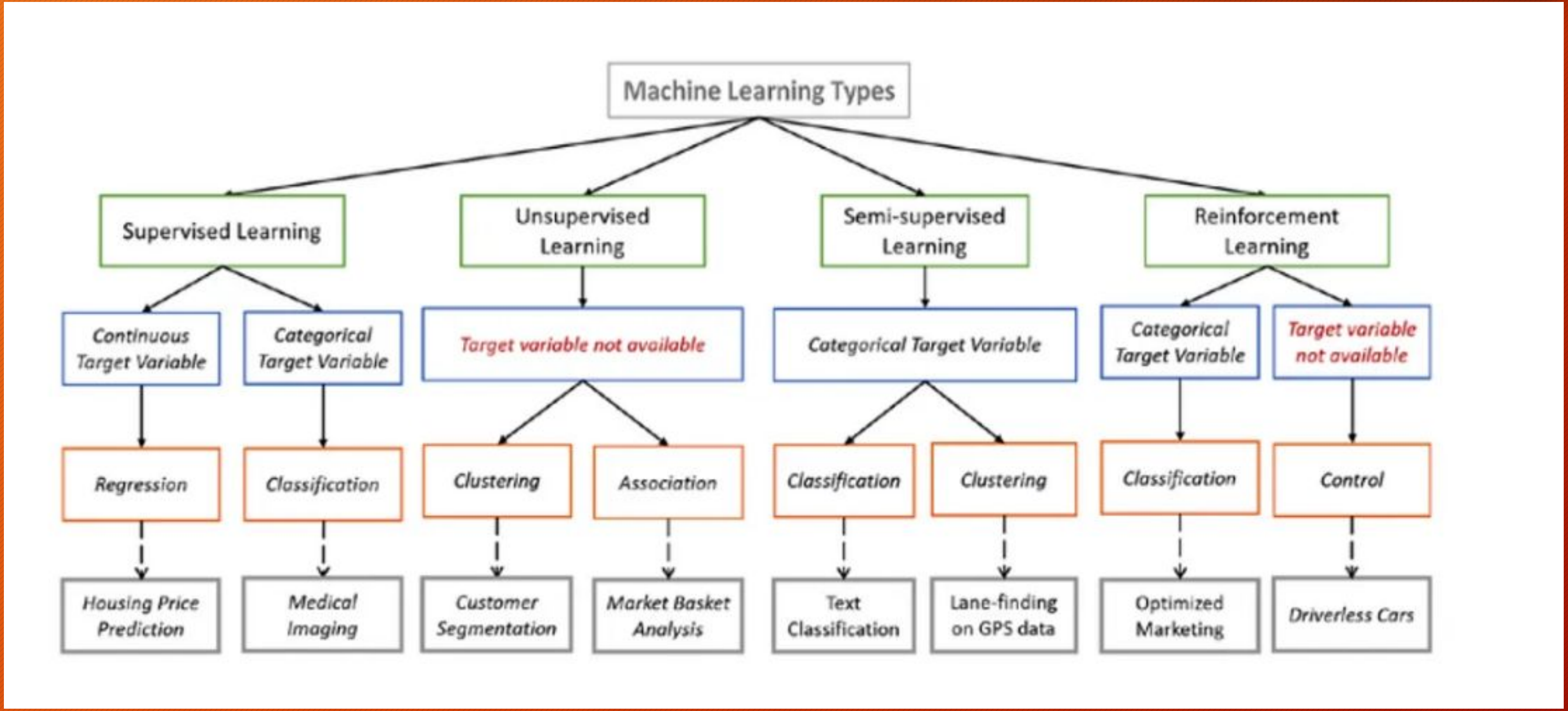
Machine Learning Algorithms

ML LIFE CYCLE



ML algorithms



Supervised ML Algorithms

Supervised learning algorithms try to *model relationships and dependencies between the target prediction output and the input features* such that we can predict the output values for new data based on those relationships which it learned from the previous data sets.

Supervised Algorithm

1. Linear Regression
2. Logistic Regression
3. Decision Tree
4. SVM
5. Naive Bayes
6. kNN
7. K-Means
8. Random Forest
9. Gradient Boosting algorithms
    1. GBM
    2. XGBoost
    3. LightGBM
    4. CatBoost

Linear regression

The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing the order of weight, without asking them their weights! What do you think the child will do? He/she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to weight by a relationship, which looks like the equation above.

In this equation:

Y – Dependent Variable

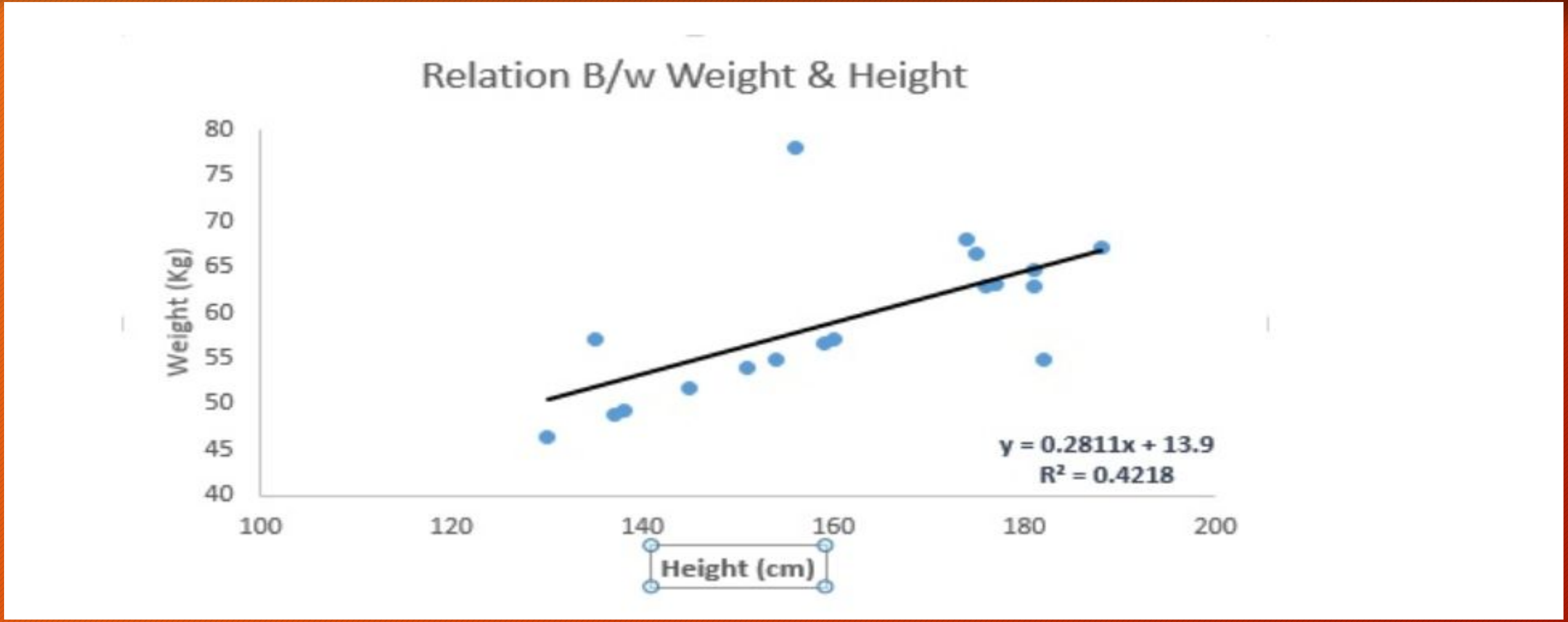
a – Slope

X – Independent variable

b – Intercept

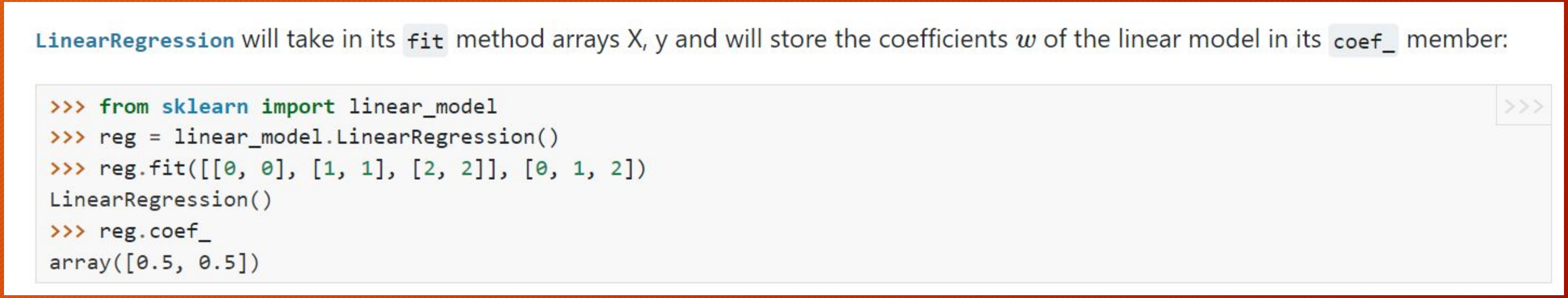
These coefficients a and b are derived based on minimizing the sum of the squared difference of distance between data points and the regression line.

Linear Regression



Linear Regression

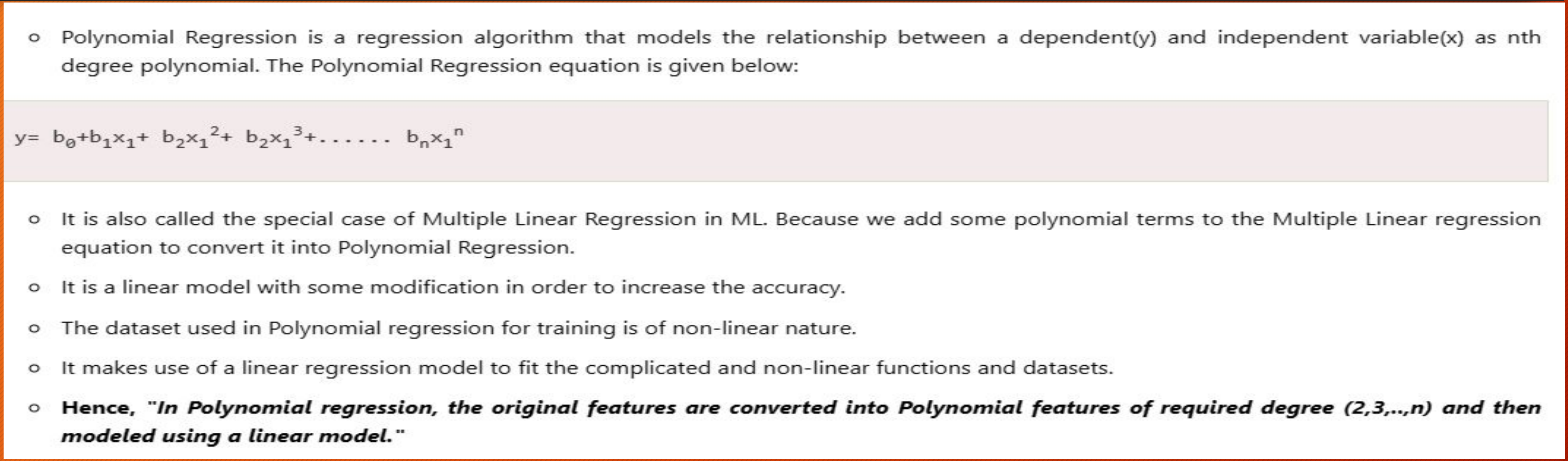
Linear Regression is mainly of two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent variables. While finding the best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression.



`LinearRegression` will take in its `fit` method arrays X, y and will store the coefficients w of the linear model in its `coef_` member:

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

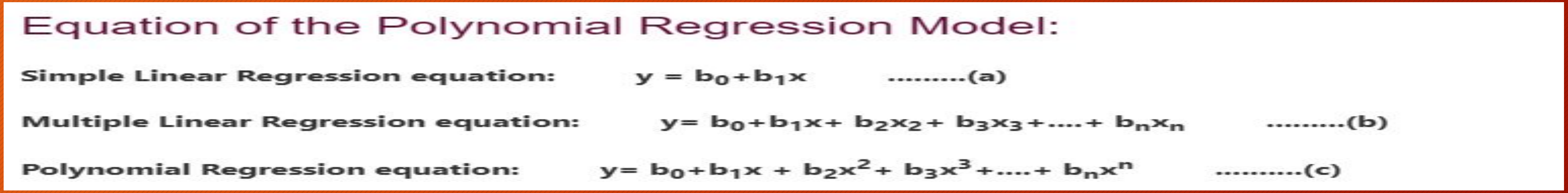
ML Polynomial Regression



- 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:

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

- 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."***



Equation of the Polynomial Regression Model:

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

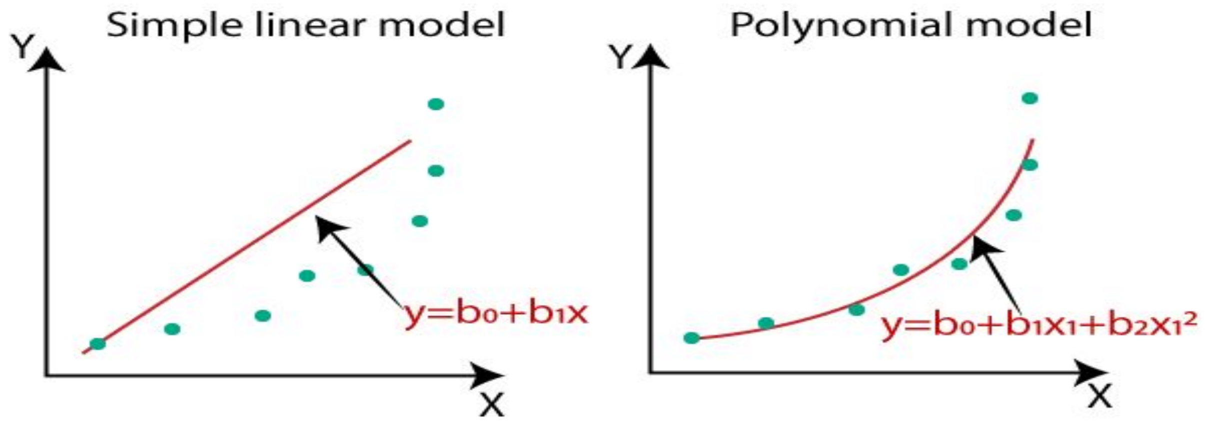
Multiple Linear Regression equation: $y= b_0+b_1x+ b_2x_2+ b_3x_3+\ldots.+ b_nx_n$ ....(b)

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

Need for Polynomial Regression:

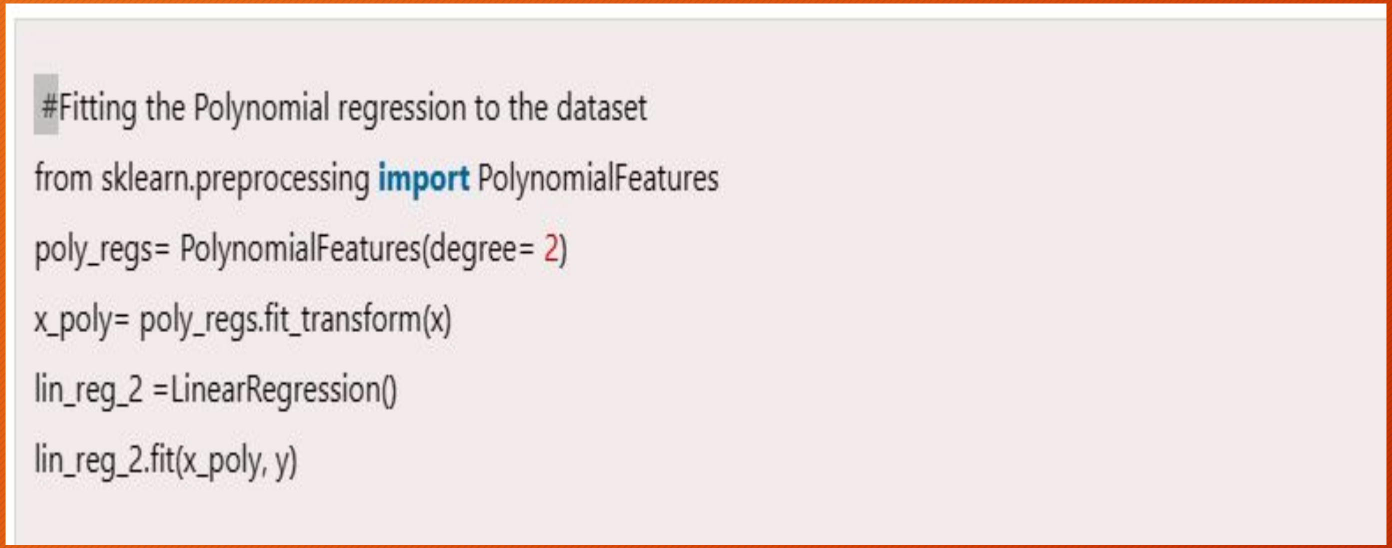
The need of Polynomial Regression in ML can be understood in the below points:

- 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.*

Syntax



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

Flow of Machine learning Project.

- Import necessary libraries
- Connect to data
- Perform EDA(missing values, outliers, standardisation, Label encoding)
- Separate X data and Y data
- Split the data in Train and Test data.
- Perform PCA
- Fit the Data to a ML algorithm.
- Predict on Test data.
- Calculate error ,display confusion metrics.
- Display the model accuracy

Logistic regression

- Logistic regression estimates the probability of an event occurring, such as voted or didn't vote, based on a given dataset of independent variables. Since the outcome is a probability, the dependent variable is bounded between 0 and 1. In logistic regression, a logit transformation is applied on the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:
- Logit(pi) = 1/(1+ exp(-pi))
- ln(pi/(1-pi)) = Beta_0 + Beta_1*X_1 + … + B_k*K_k

Logistic Regression

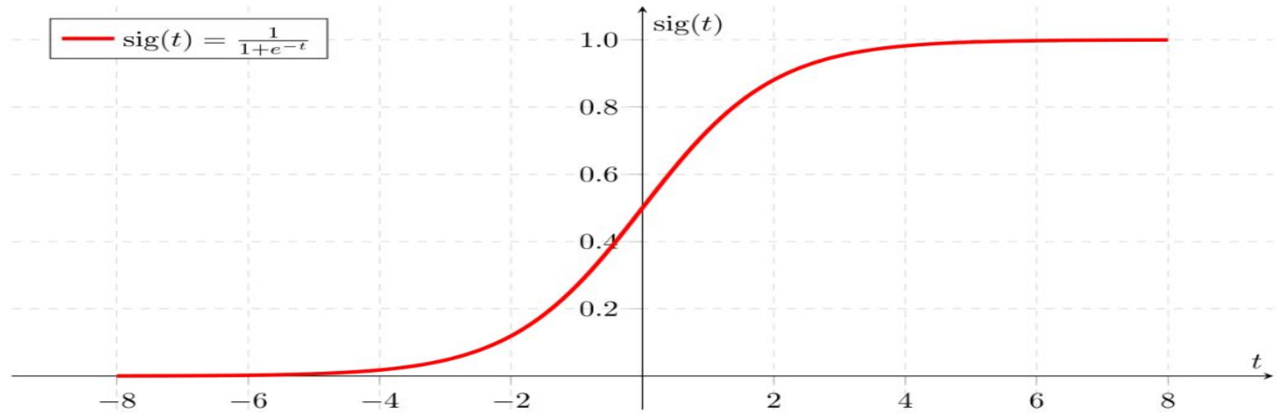
- In this logistic regression equation, logit(pi) is the dependent or response variable and x is the independent variable. The beta parameter, or coefficient, in this model is commonly estimated via maximum likelihood estimation (MLE). This method tests different values of beta through multiple iterations to optimize for the best fit of log odds. All of these iterations produce the log likelihood function, and logistic regression seeks to maximize this function to find the best parameter estimate. Once the optimal coefficient (or coefficients if there is more than one independent variable) is found, the conditional probabilities for each observation can be calculated, logged, and summed together to yield a predicted probability. For binary classification, a probability less than .5 will predict 0 while a probability greater than 0 will predict 1

Sigmoid Function

Hypothesis => Z = WX + B

hΘ(x) = sigmoid (Z)

Sigmoid Function



Decision Boundary

To predict which class a data belongs, a threshold can be set. Based upon this threshold, the obtained estimated probability is classified into classes.

Say, if predicted_value ≥ 0.5, then classify email as spam else as not spam.

Types of logistic regression

There are three types of logistic regression models, which are defined based on categorical response.

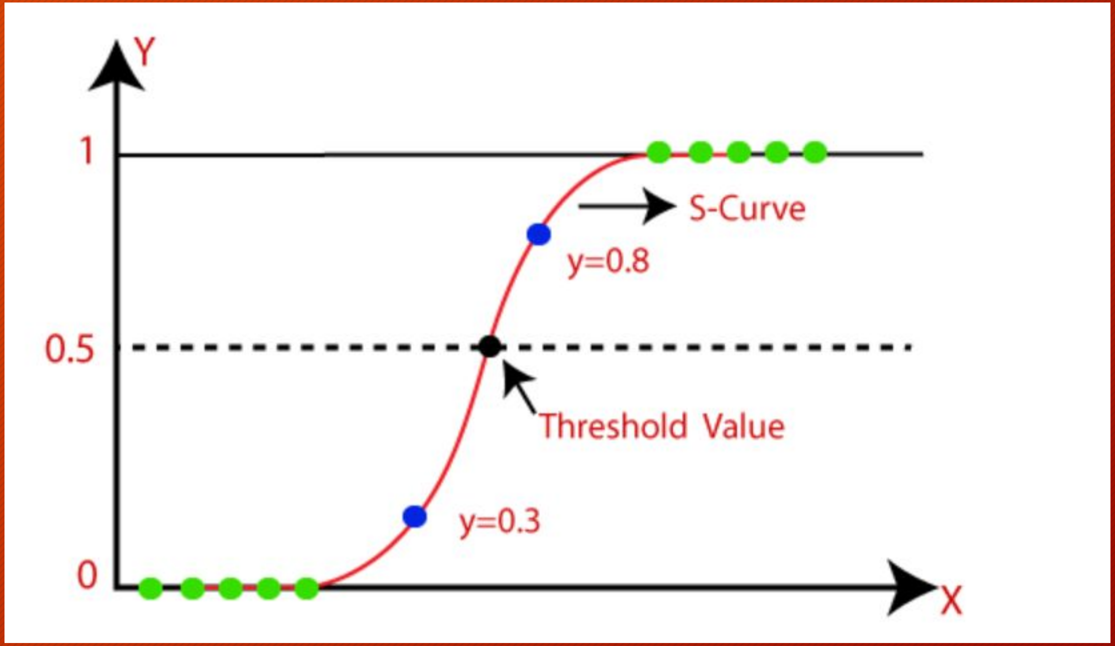
- **Binary logistic regression:** In this approach, the response or dependent variable is dichotomous in nature—i.e. it has only two possible outcomes (e.g. 0 or 1). Some popular examples of its use include predicting if an e-mail is spam or not spam or if a tumor is malignant or not malignant. Within logistic regression, this is the most commonly used approach, and more generally, it is one of the most common classifiers for binary classification.
- **Multinomial logistic regression:** In this type of logistic regression model, the dependent variable has three or more possible outcomes; however, these values have no specified order. For example, movie studios want to predict what genre of film a moviegoer is likely to see to market films more effectively. A multinomial logistic regression model can help the studio to determine the strength of influence a person's age, gender, and dating status may have on the type of film that they prefer. The studio can then orient an advertising campaign of a specific movie toward a group of people likely to go see it.
- **Ordinal logistic regression:** This type of logistic regression model is leveraged when the response variable has three or more possible outcome, but in this case, these values do have a defined order. Examples of ordinal responses include grading scales from A to F or

Use cases of logistic regression

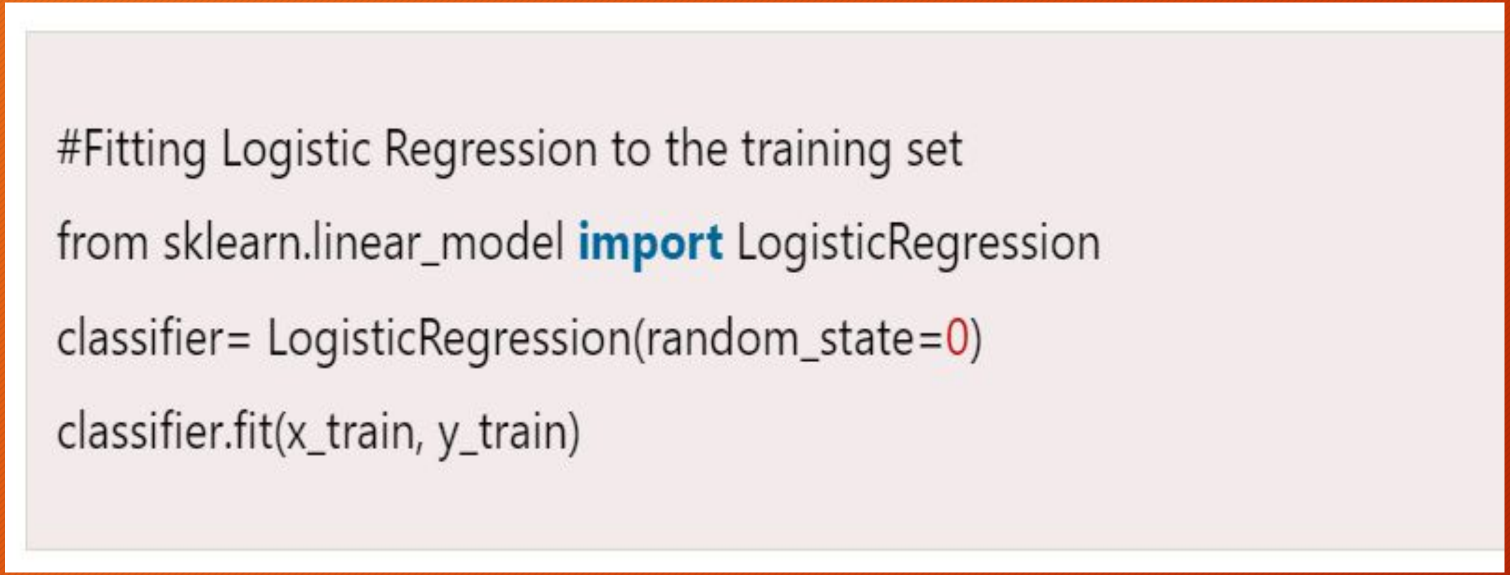
- Logistic regression is commonly used for prediction and classification problems. Some of these use cases include:
- **Fraud detection:** Logistic regression models can help teams identify data anomalies, which are predictive of fraud. Certain behaviors or characteristics may have a higher association with fraudulent activities, which is particularly helpful to banking and other financial institutions in protecting their clients. SaaS-based companies have also started to adopt these practices to eliminate fake user accounts from their datasets when conducting data analysis around business performance.
- **Disease prediction:** In medicine, this analytics approach can be used to predict the likelihood of disease or illness for a given population. Healthcare organizations can set up preventative care for individuals that show higher propensity for specific illnesses.
- **Churn prediction**: Specific behaviors may be indicative of churn in different functions of an organization. For example, human resources and management teams may want to know if there are high performers within the company who are at risk of leaving the organization; this type of insight can prompt conversations to understand problem areas within the company, such as culture or compensation. Alternatively, the sales organization may want to learn which of their clients are at risk of taking their business elsewhere. This can prompt teams to set up a retention strategy to avoid lost revenue.

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



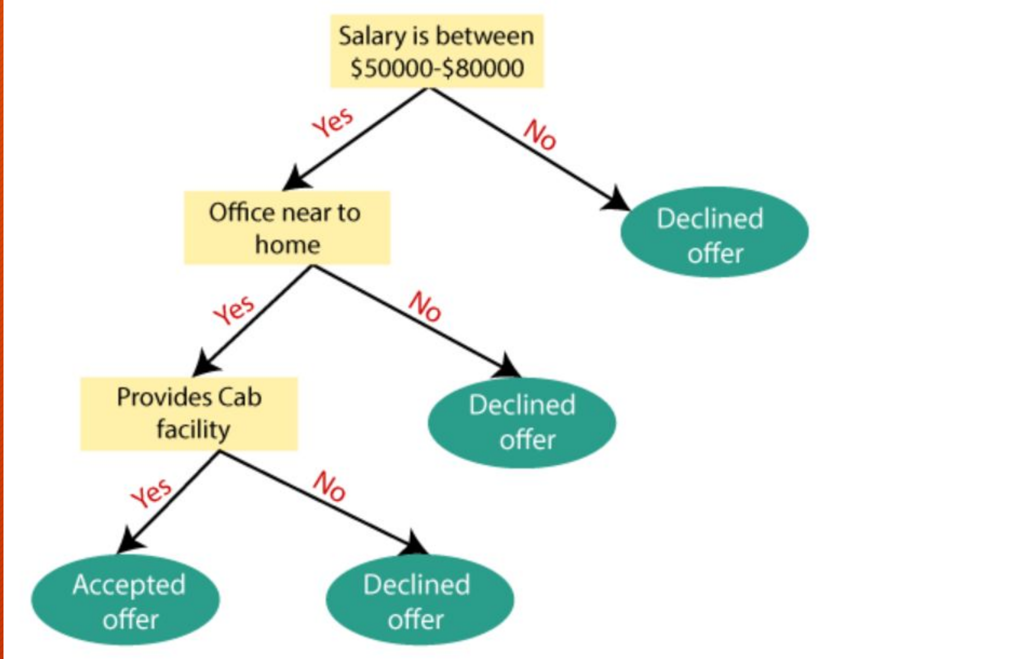
SYNTAX



```
#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

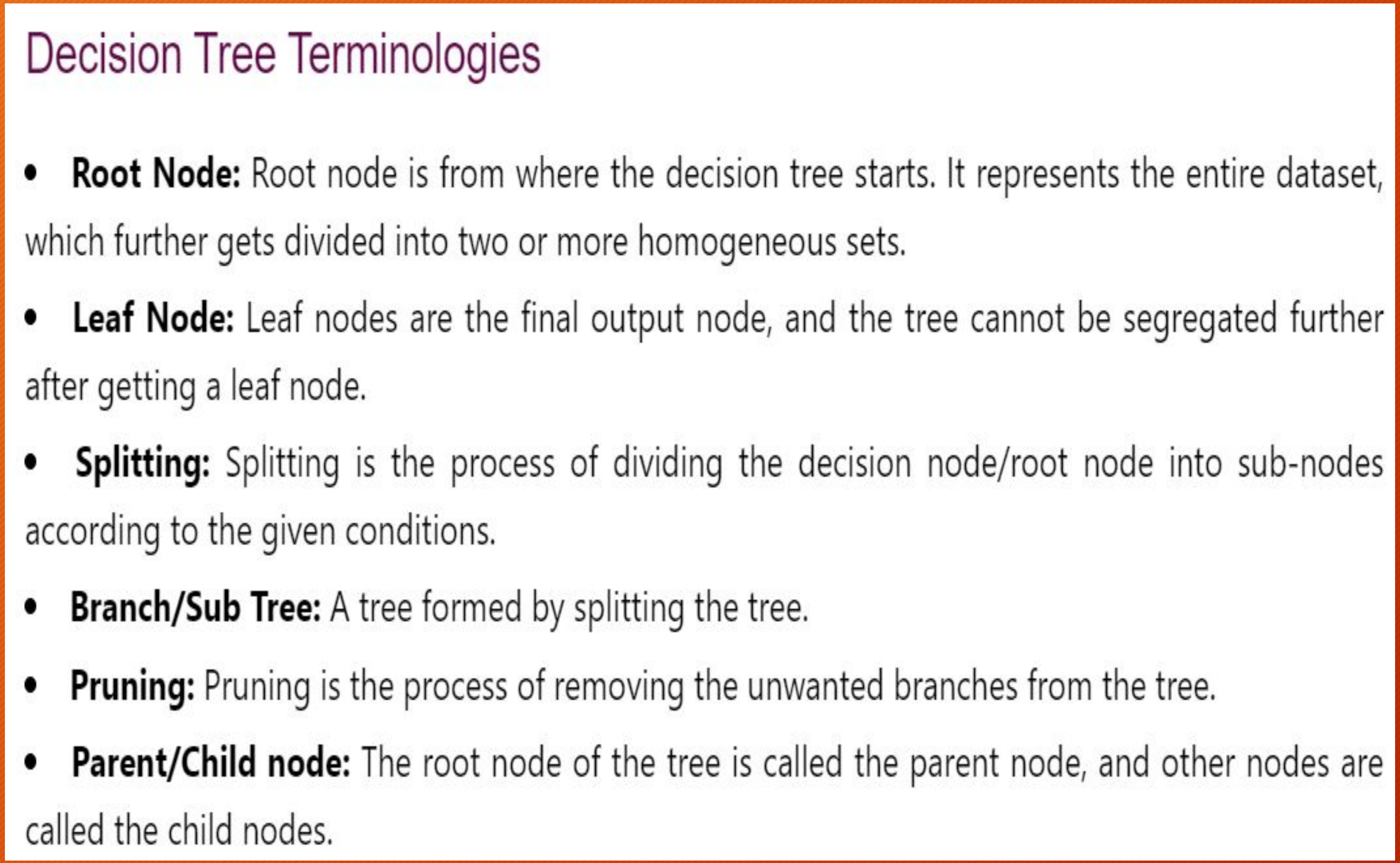
Decision Tree

- 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.



Attribute Selection Measures

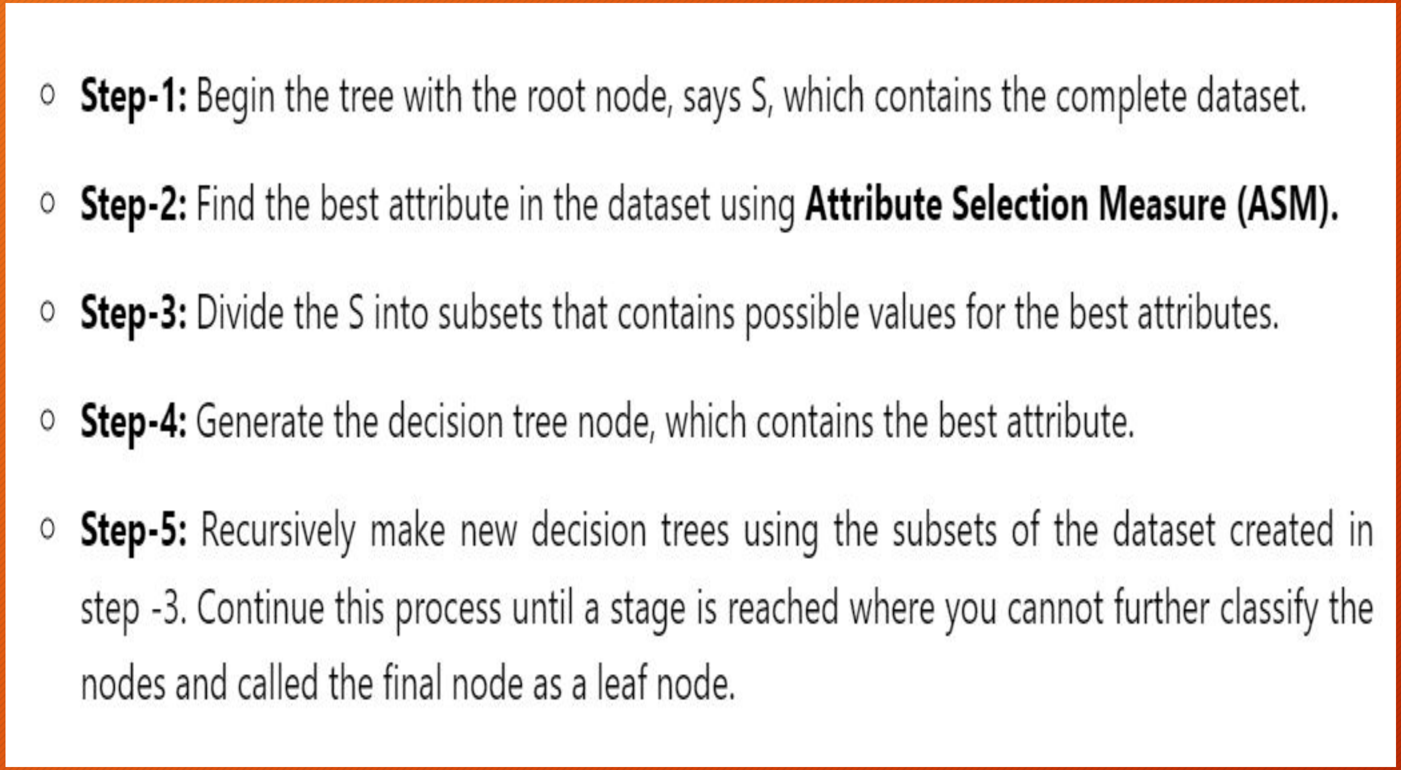
Terminologies



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.

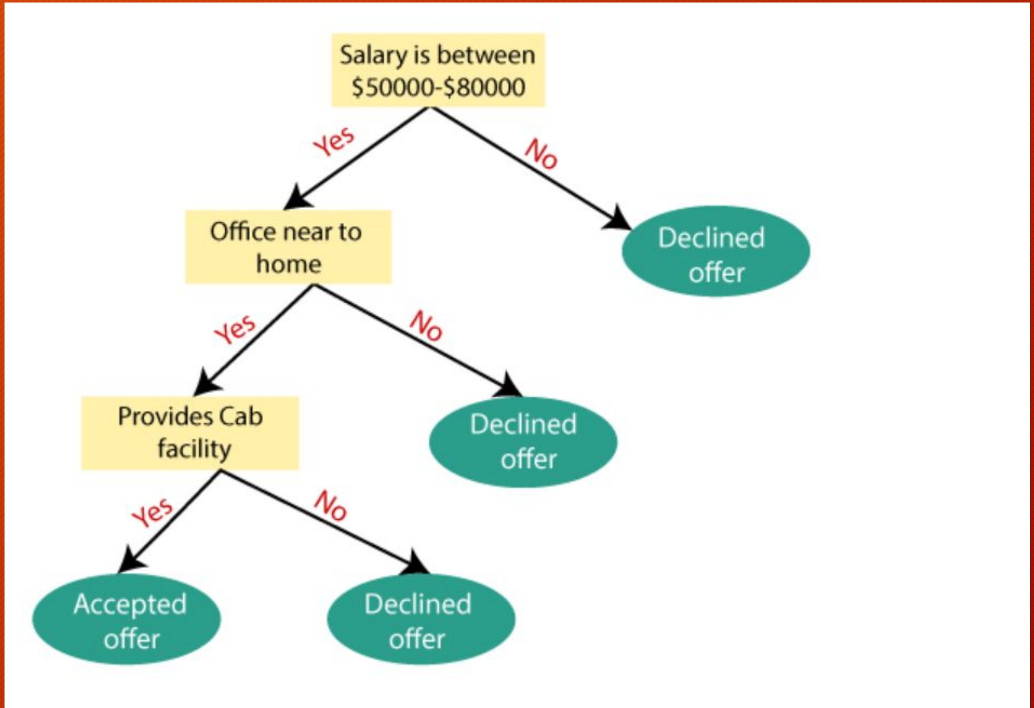
Algorithm



- **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.

An 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).



Attribute Selection Measures

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:

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

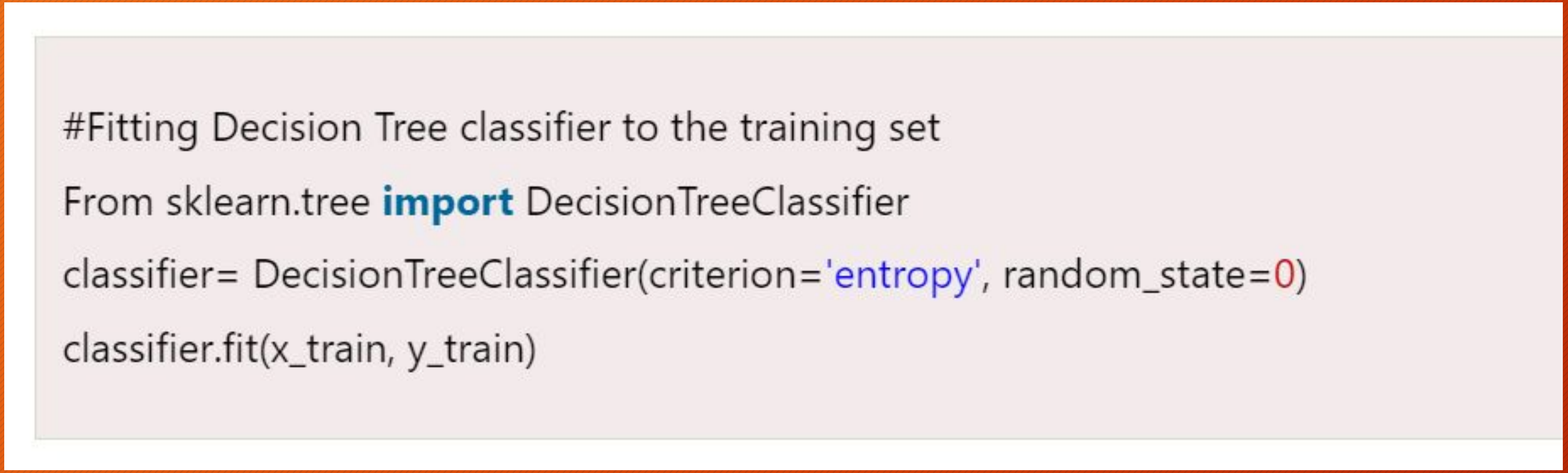
2. Gini Index:

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.

SYNTAX

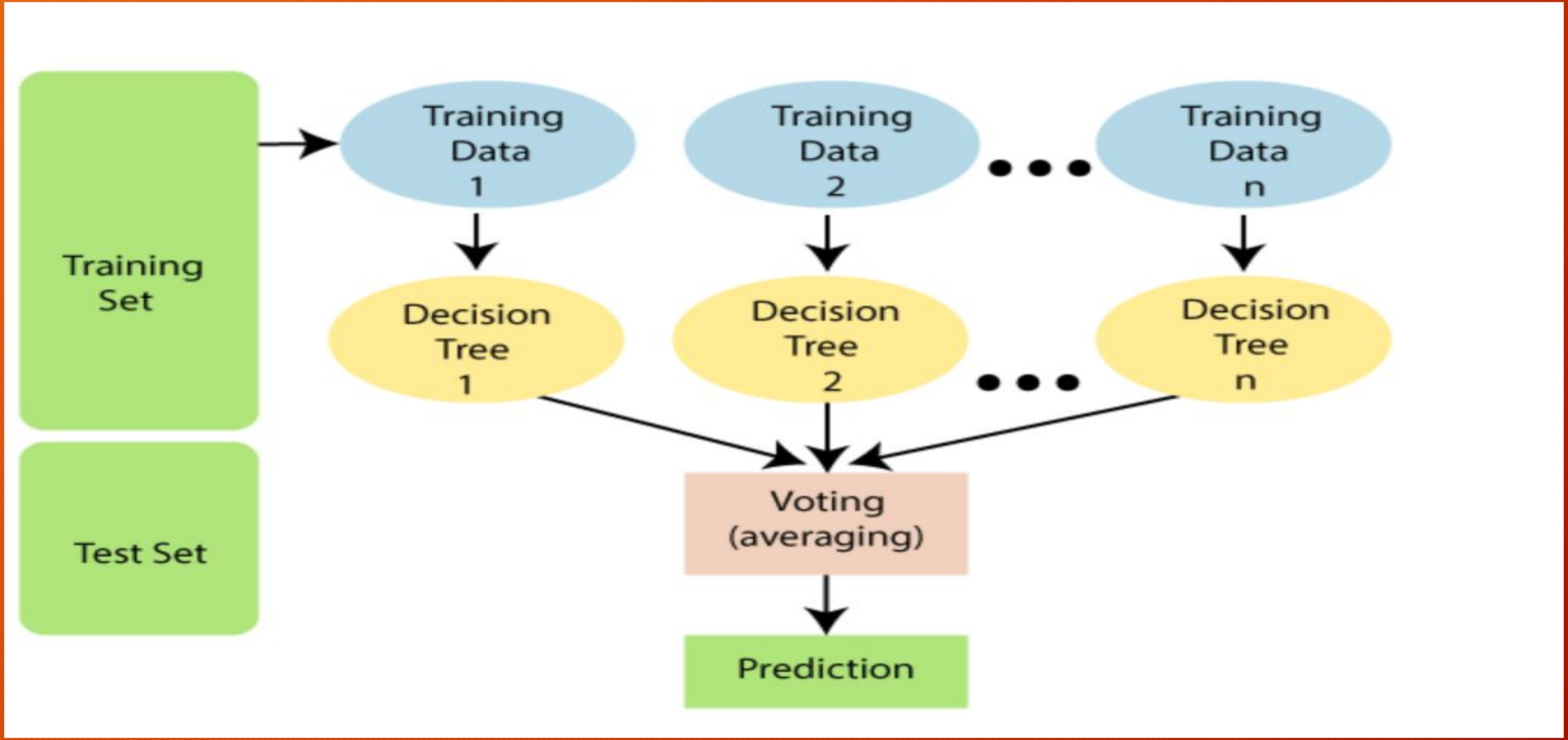


```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

Random Forest

- 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.

Decision making



Pros & Cons

Advantages of Random Forest

Random Forest is capable of performing both Classification and Regression tasks.

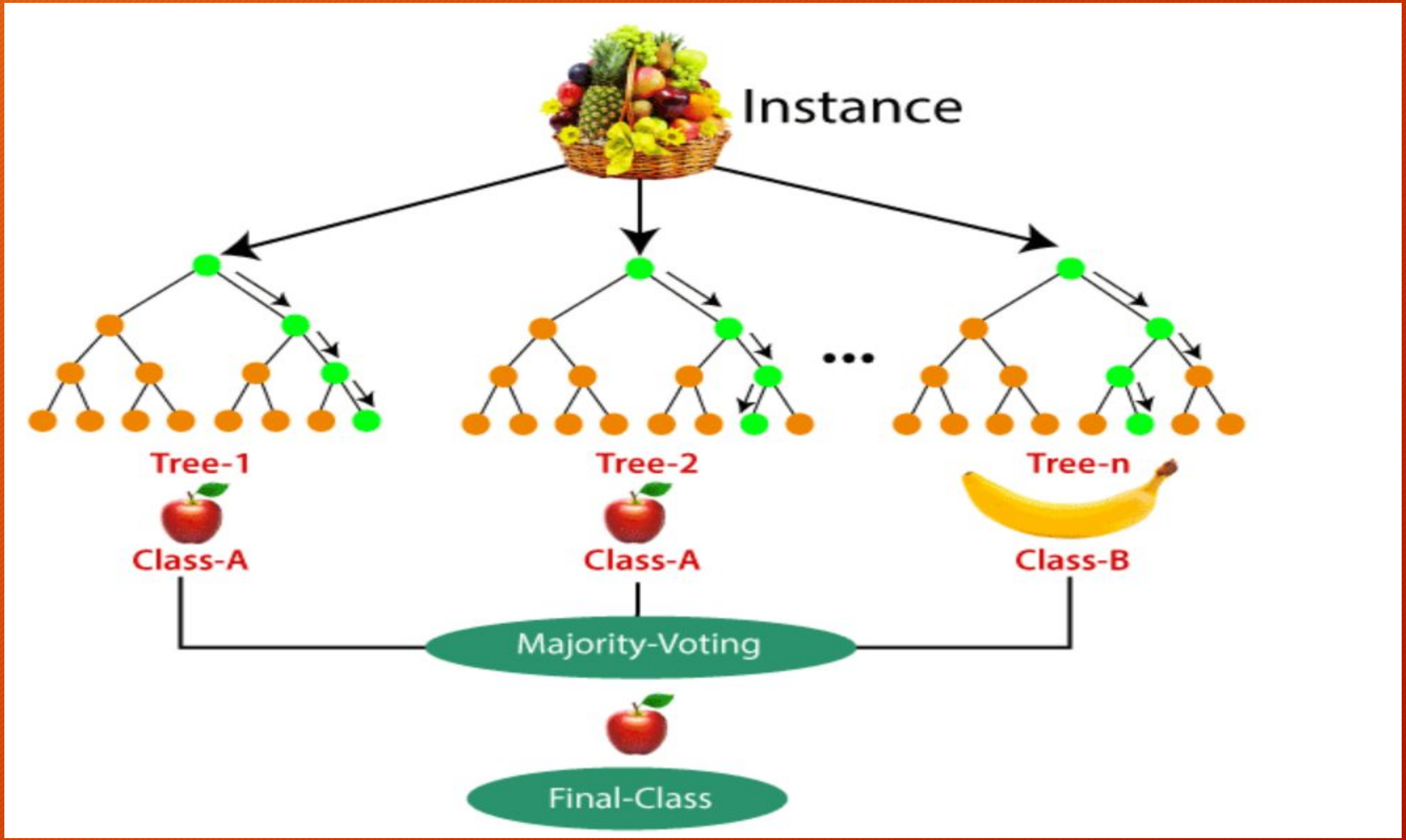
It is capable of handling large datasets with high dimensionality.

It enhances the accuracy of the model and prevents the overfitting issue.

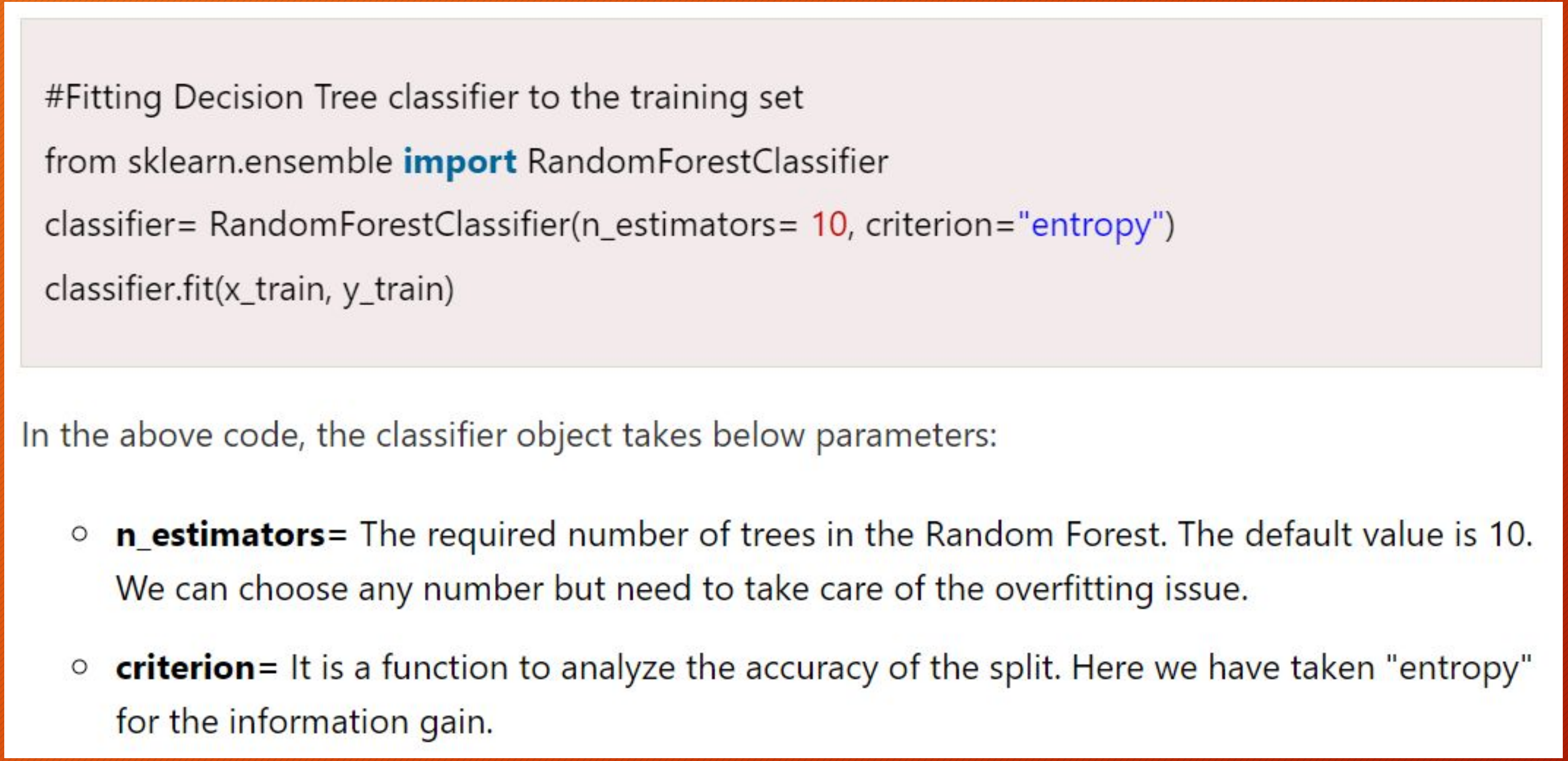
Disadvantages of Random Forest

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

An example


Instance
Tree-1
Tree-2
Tree-n
Class-A
Class-A
Class-B
Majority-Voting
Final-Class

Syntax



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

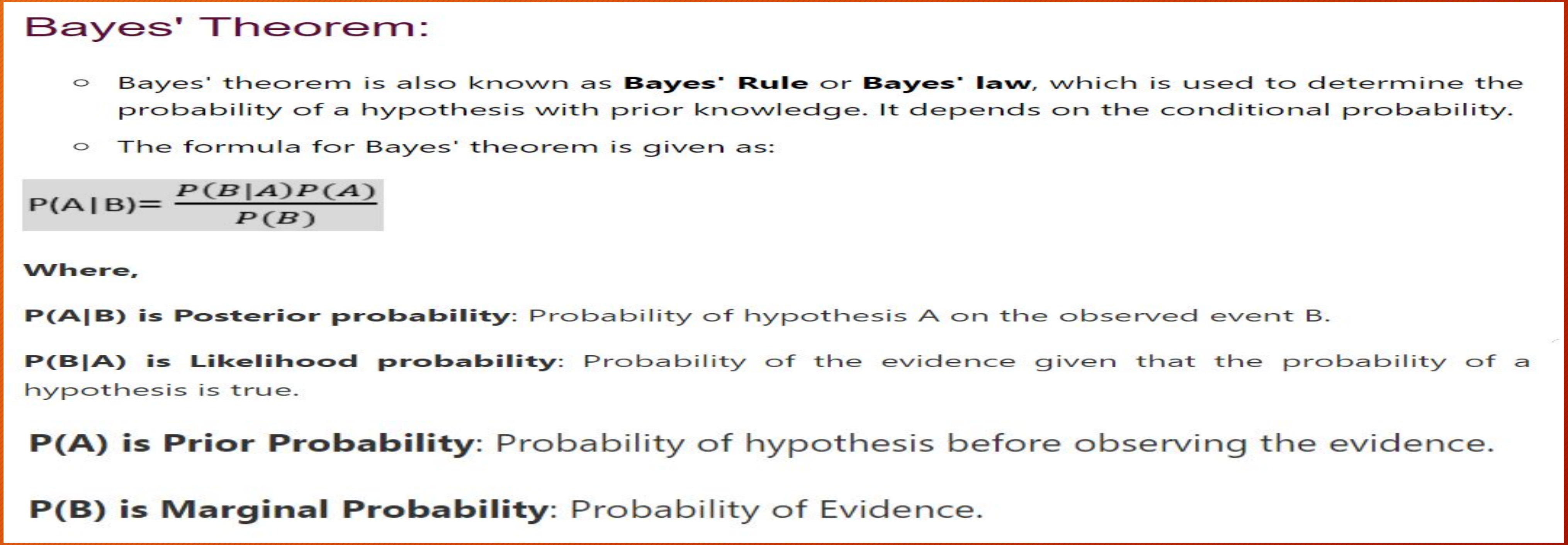
In the above code, the classifier object takes below parameters:

- **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.

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**.

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:

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

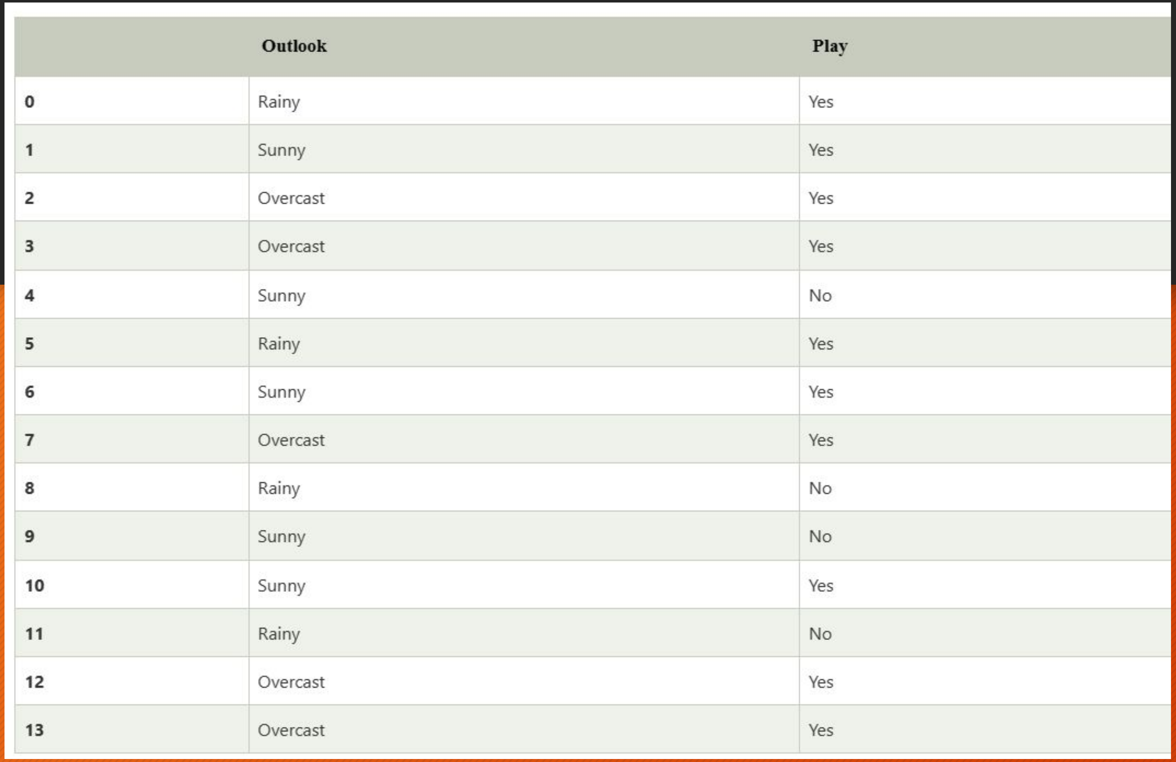
Where,

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

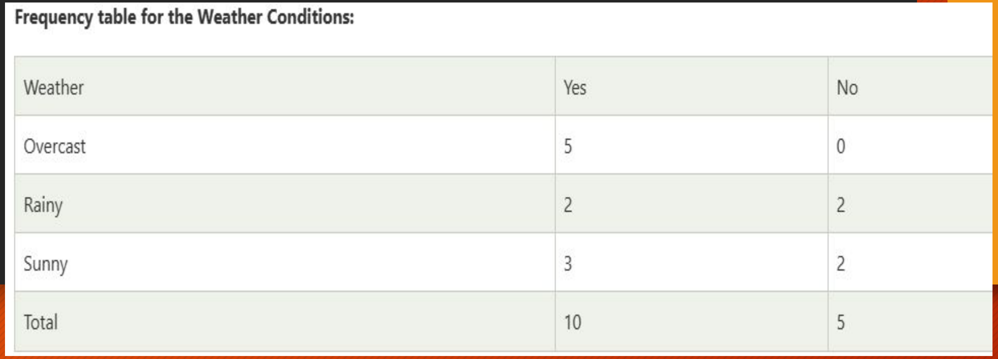
P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

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

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

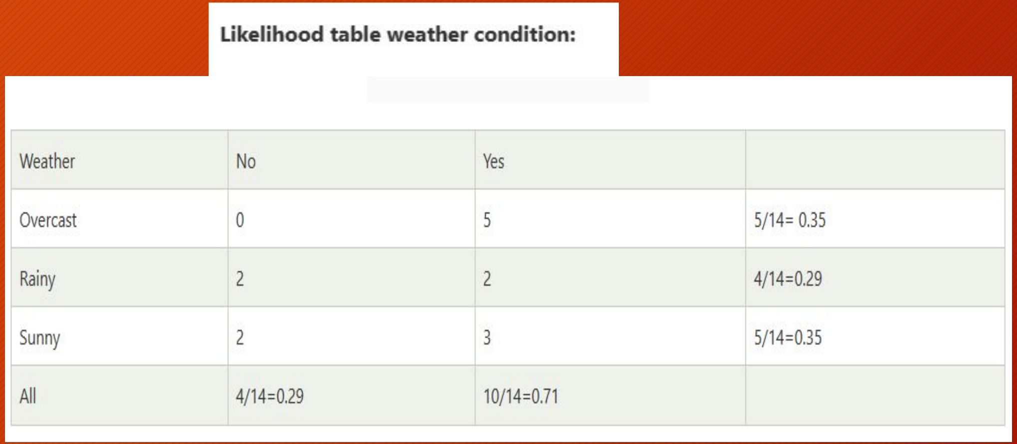


	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes



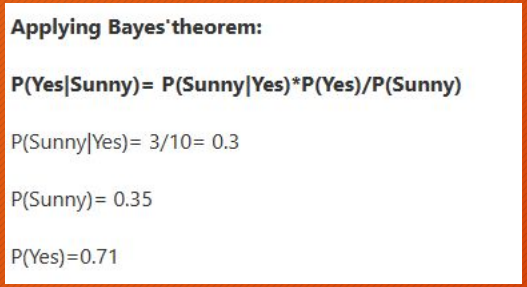
Frequency table for the Weather Conditions:

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	



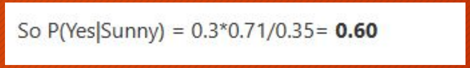
Applying Bayes'theorem:

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

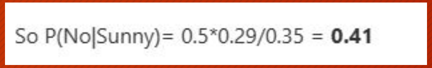
P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

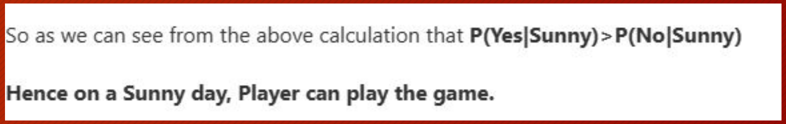
P(Yes)=0.71



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



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.

Pros & Cons

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

Types of Naïve Bayes Model:

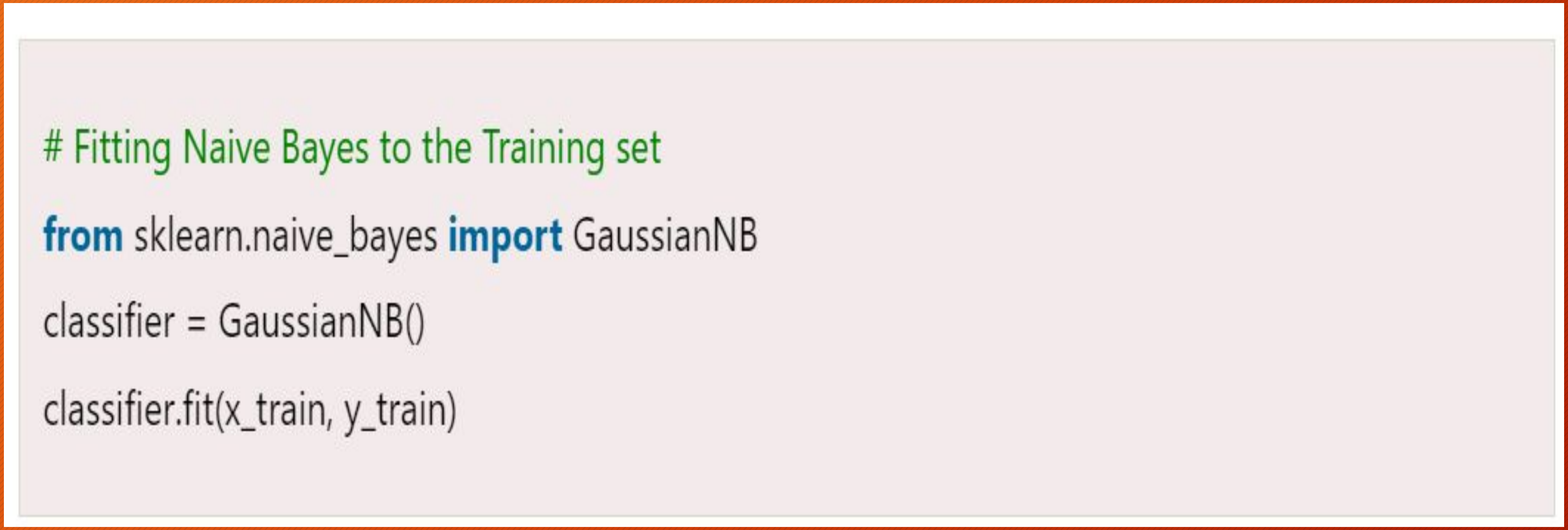
There are three types of Naive Bayes Model, which are given below:

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.
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.

Syntax

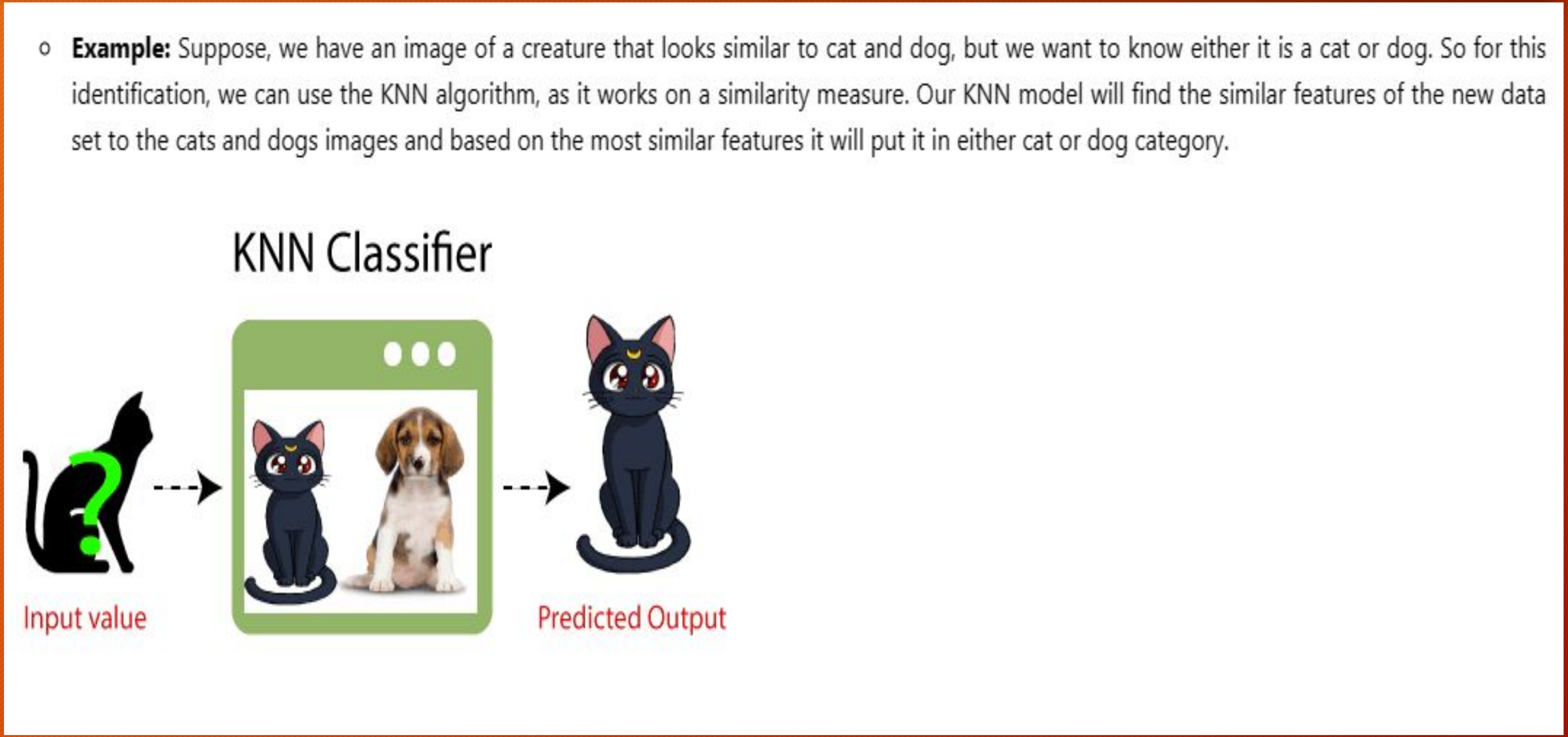


```
# Fitting Naive Bayes to the Training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train, y_train)
```

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

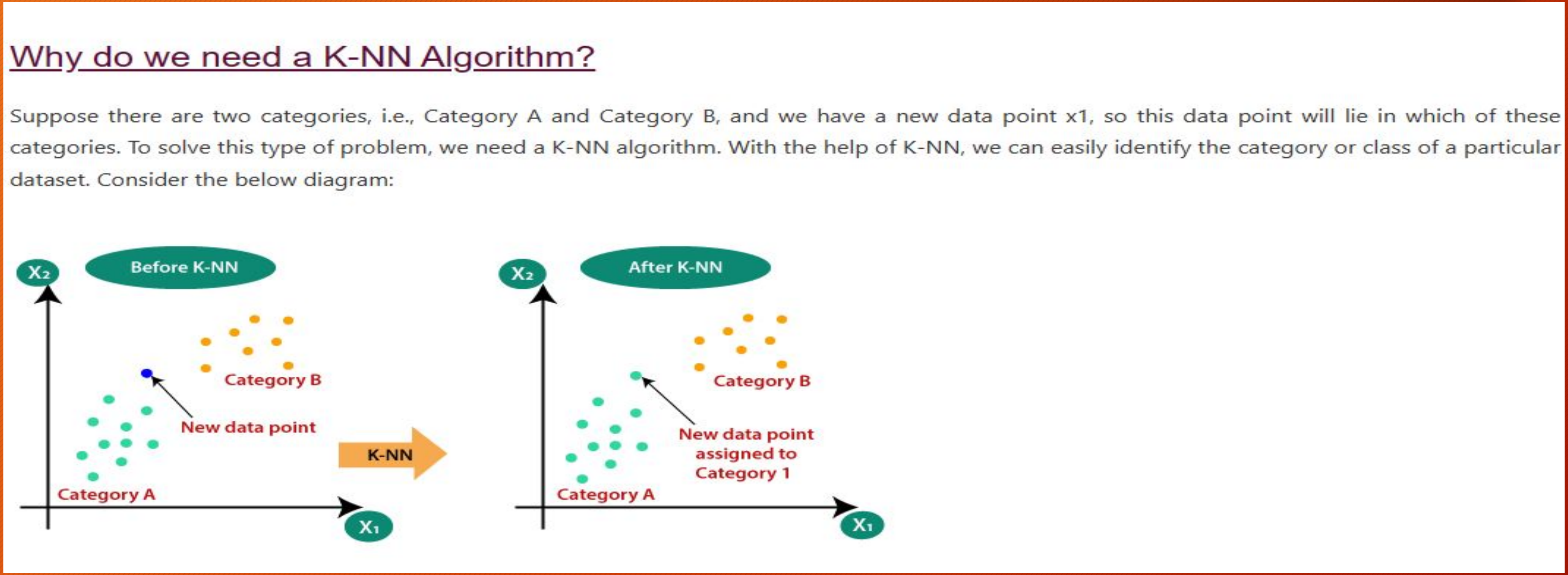
- K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

Decision making



- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

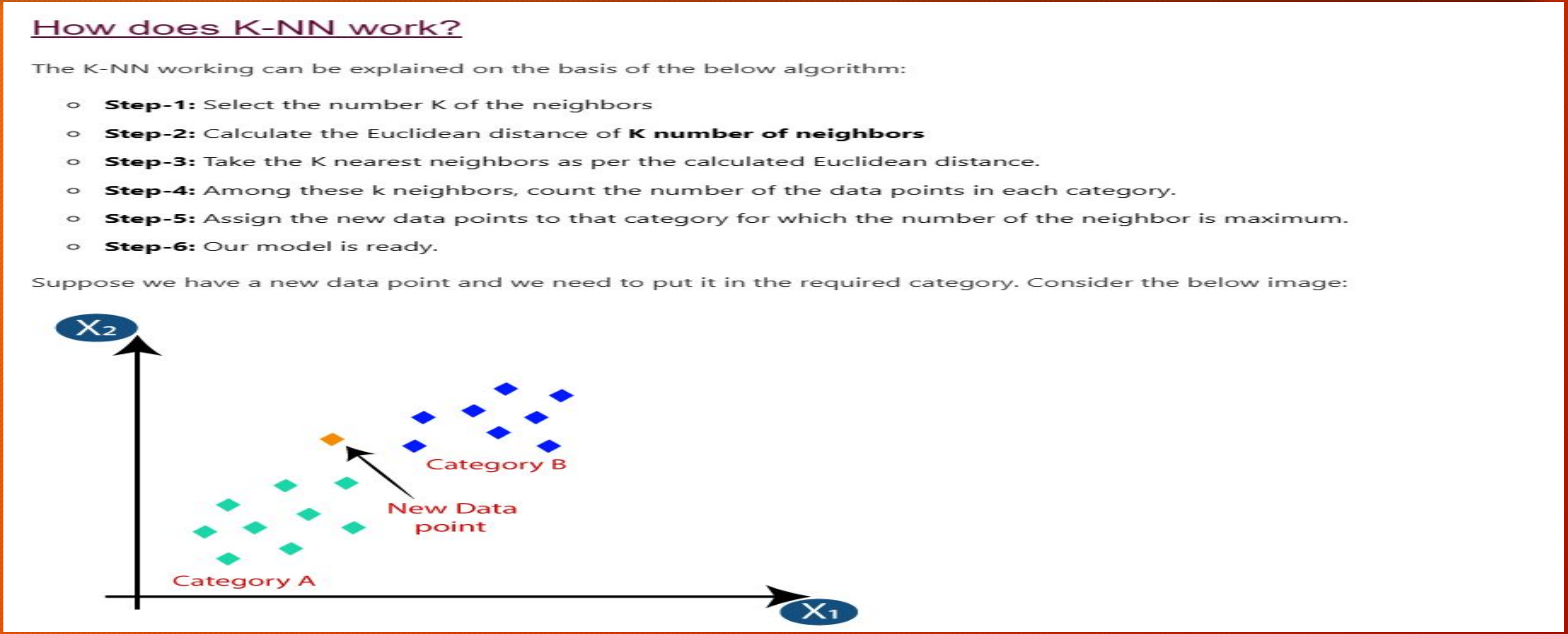
Categorization



Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:

Algorithm



How does K-NN work?

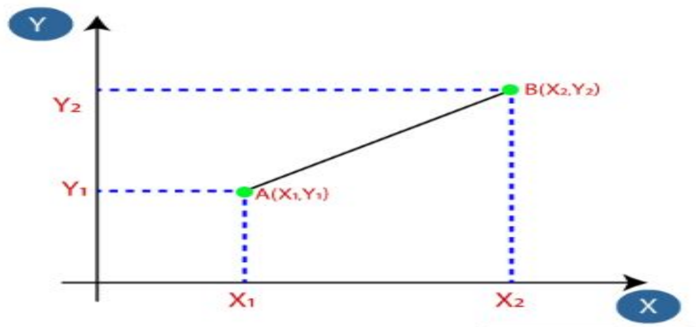
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

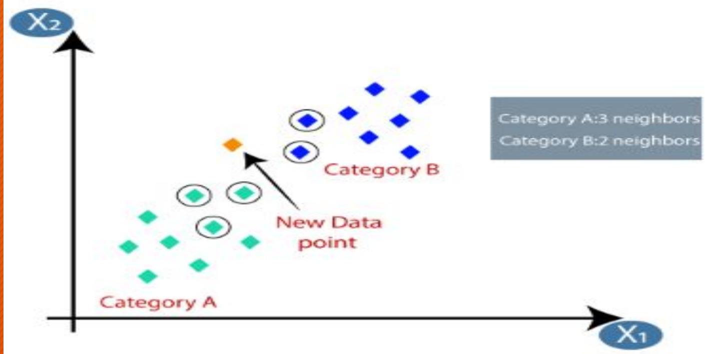
Example

- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



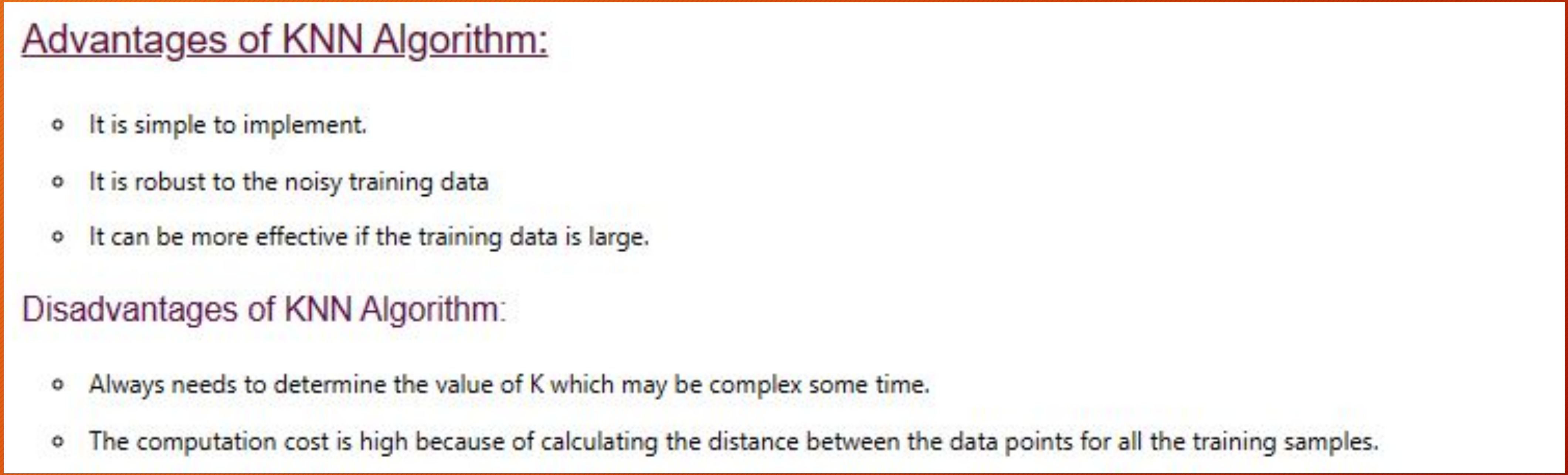
Euclidean Distance between A_1 and B_2 = $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

Pros & Cons



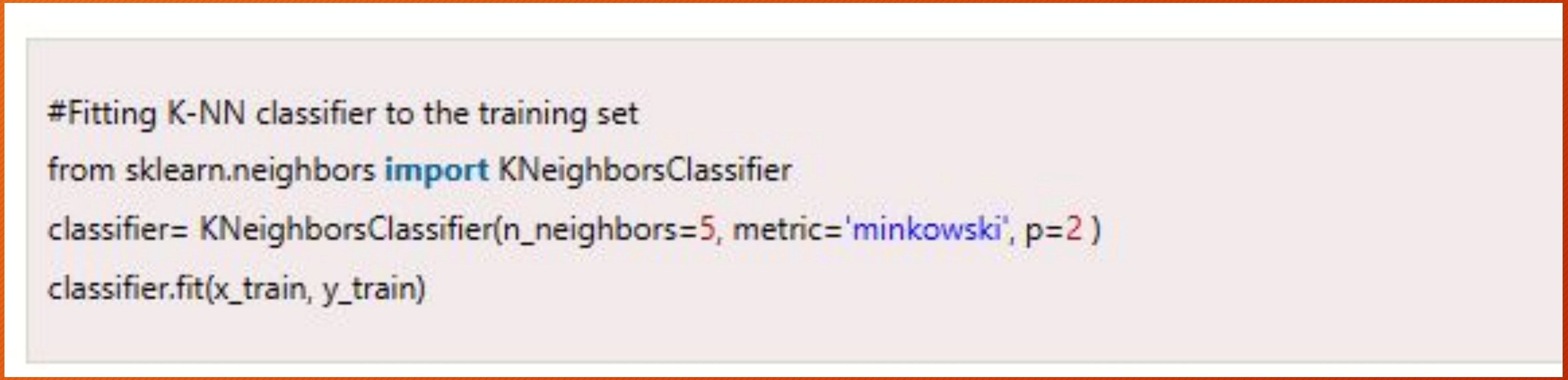
Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Syntax

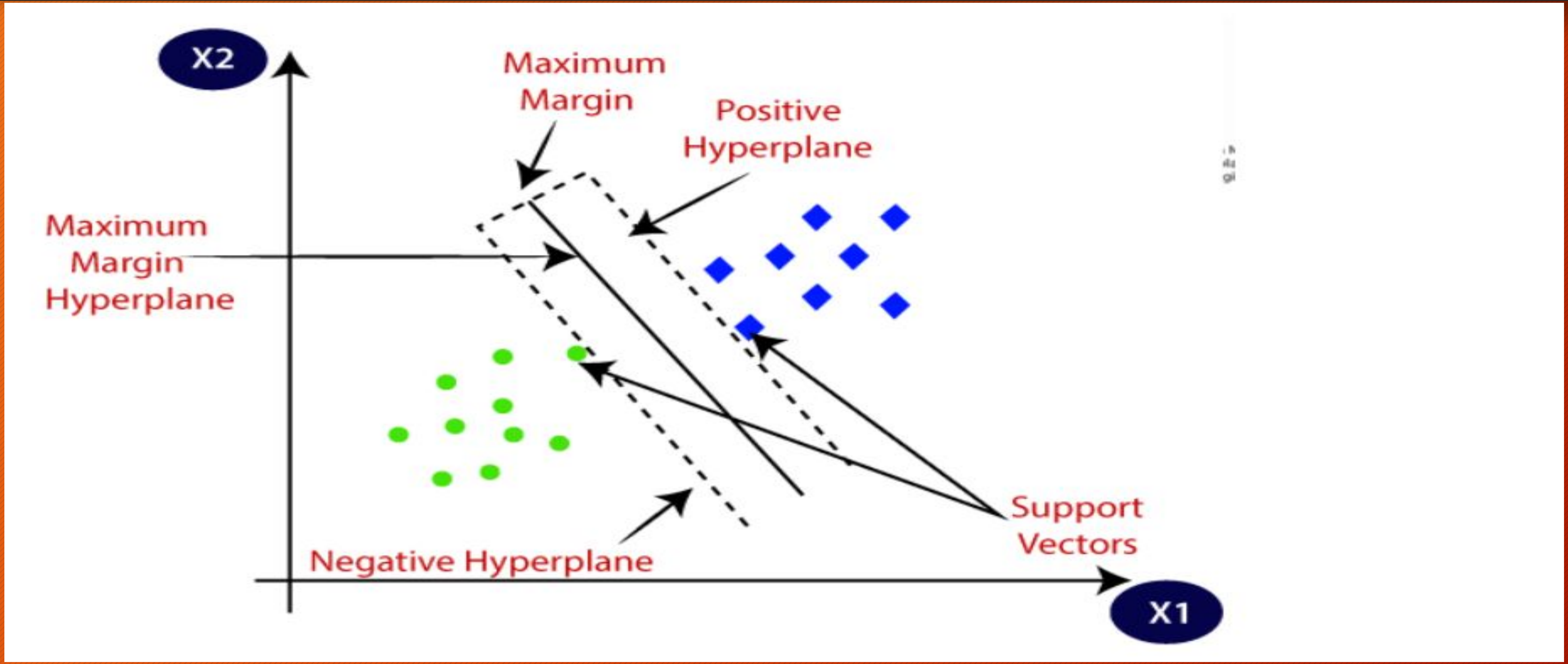


```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

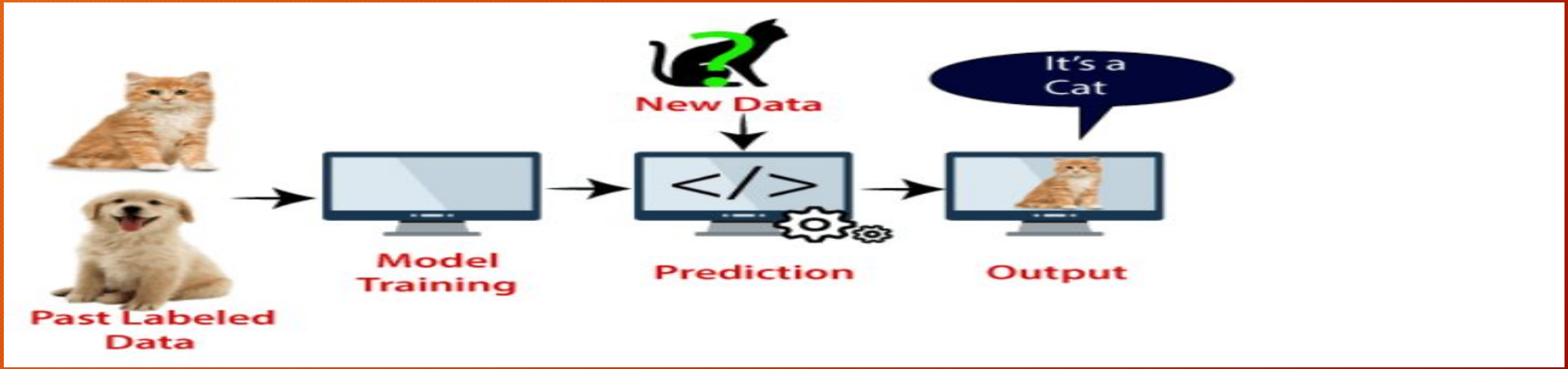
Support Vector Machine Algorithm

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
- SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane
- SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Calculating Support Vectors



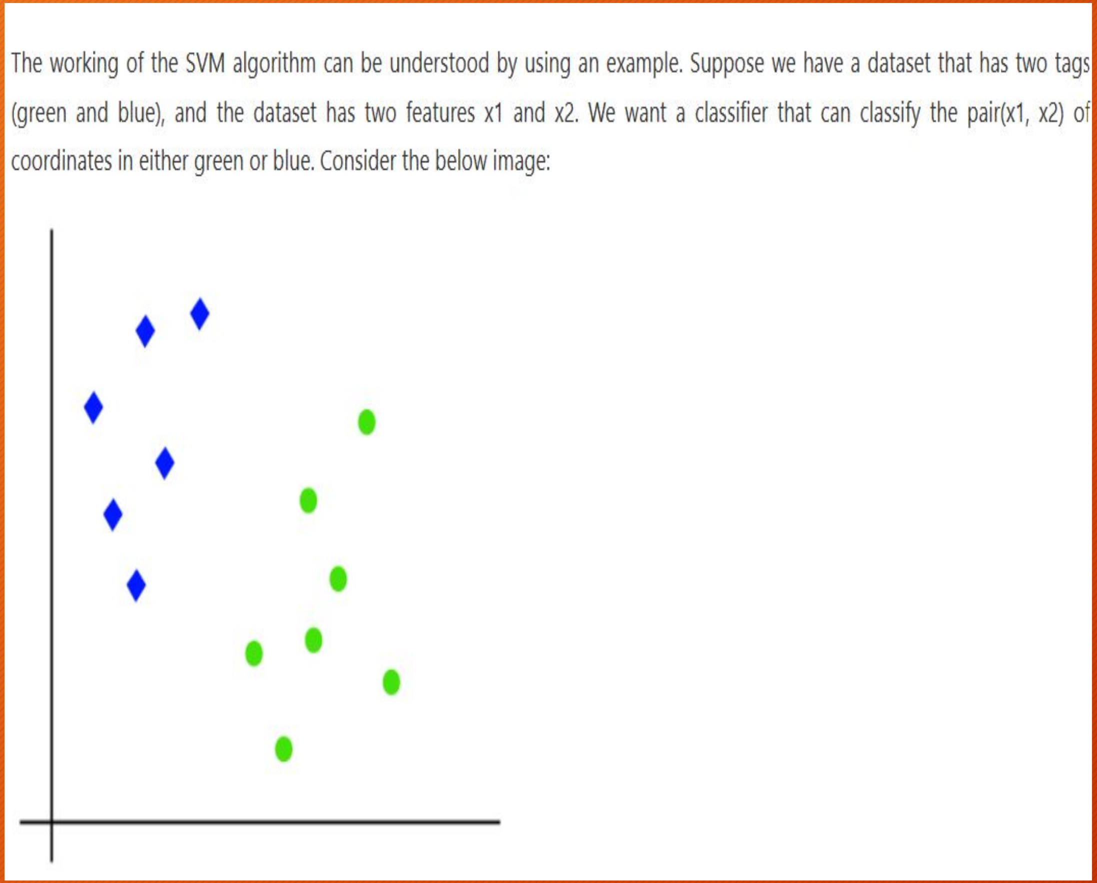
Example: SVM can be understood with the example that we have used in the KNN classifier. Suppose we see a strange cat that also has some features of dogs, so if we want a model that can accurately identify whether it is a cat or dog, so such a model can be created by using the SVM algorithm. We will first train our model with lots of images of cats and dogs so that it can learn about different features of cats and dogs, and then we test it with this strange creature. So as support vector creates a decision boundary between these two data (cat and dog) and choose extreme cases (support vectors), it will see the extreme case of cat and dog. On the basis of the support vectors, it will classify it as a cat. Consider the below diagram:



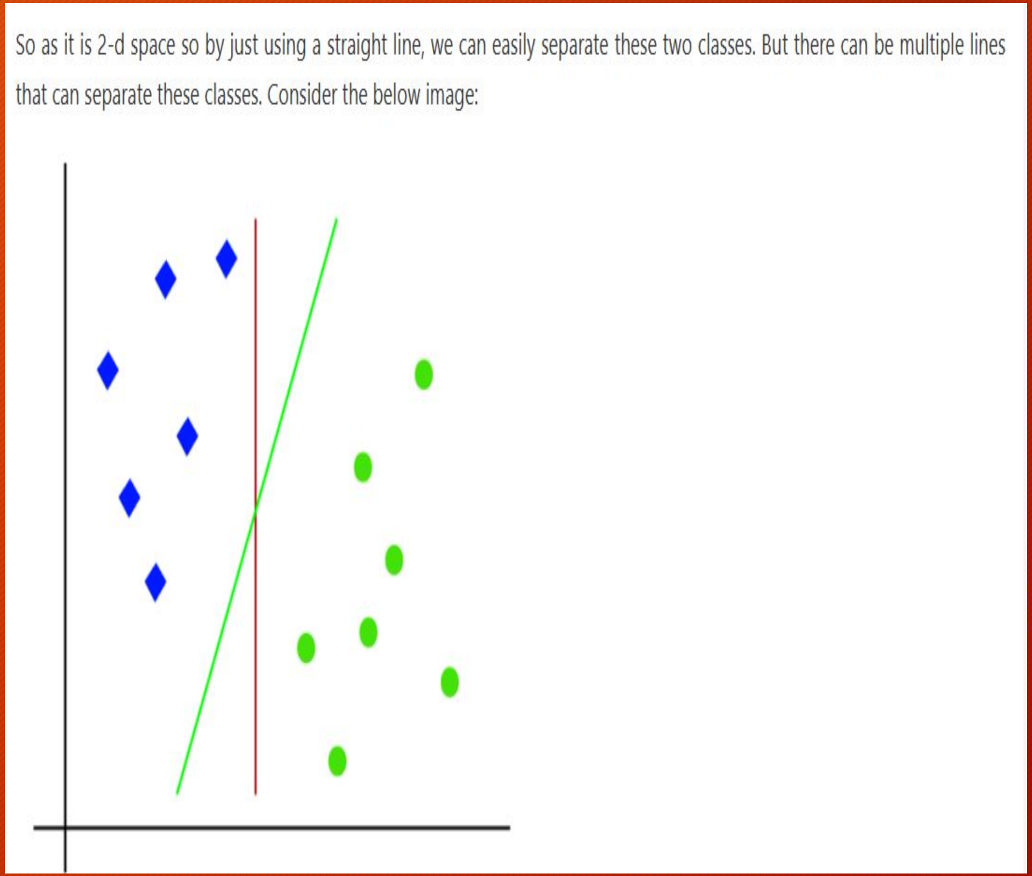
Types of SVM

- **SVM can be of two types:**
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Working

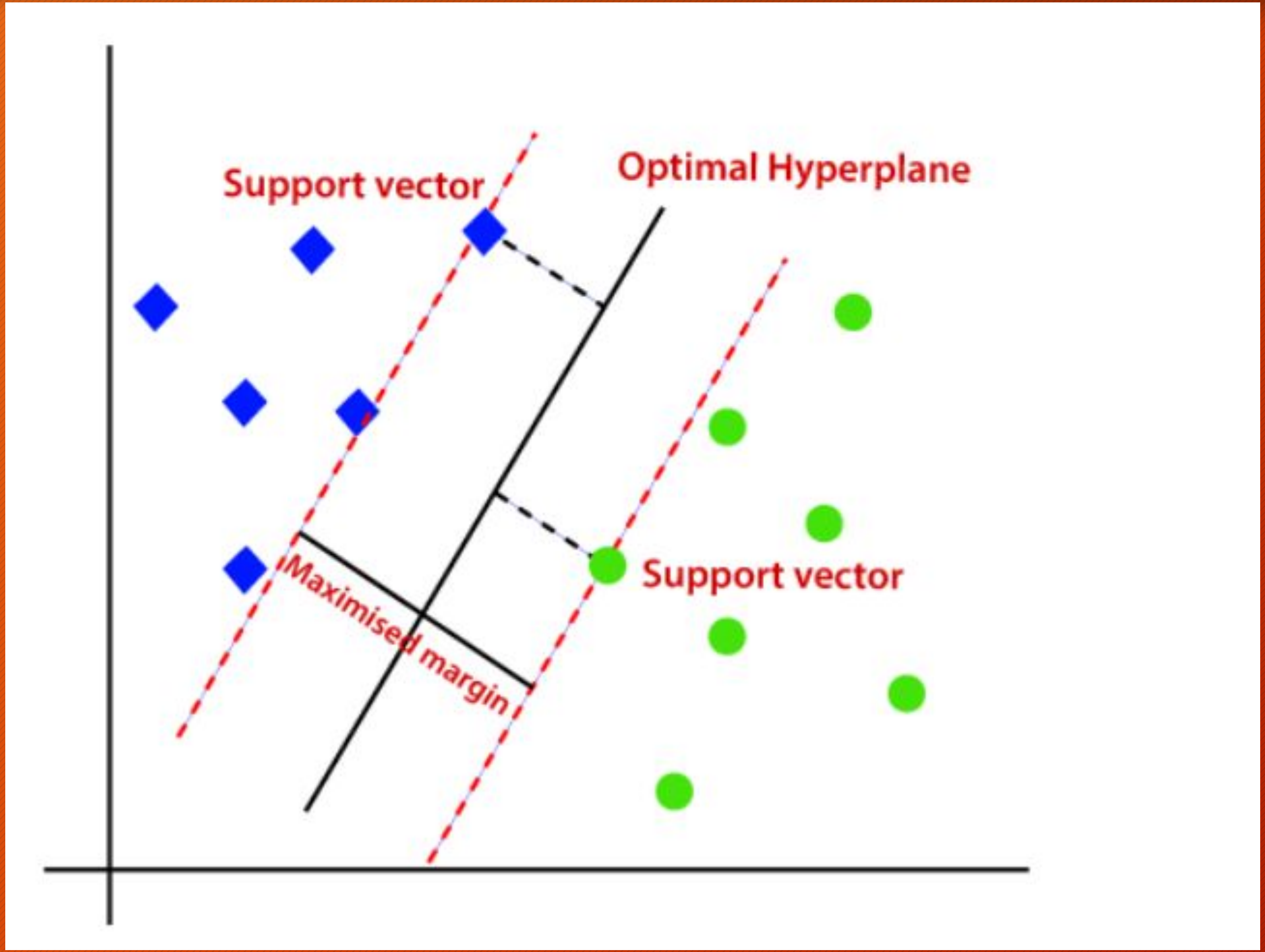


The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



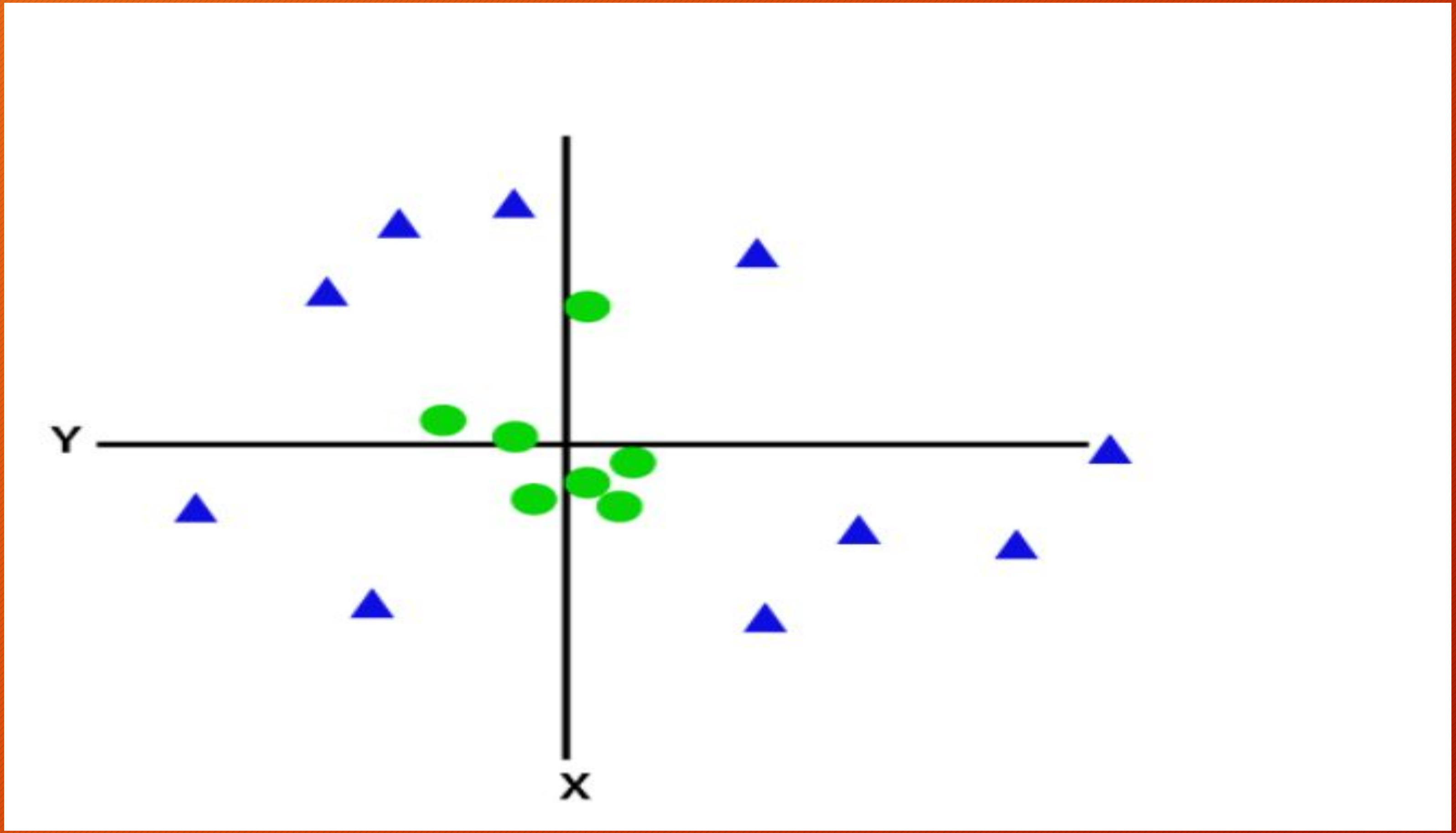
So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.



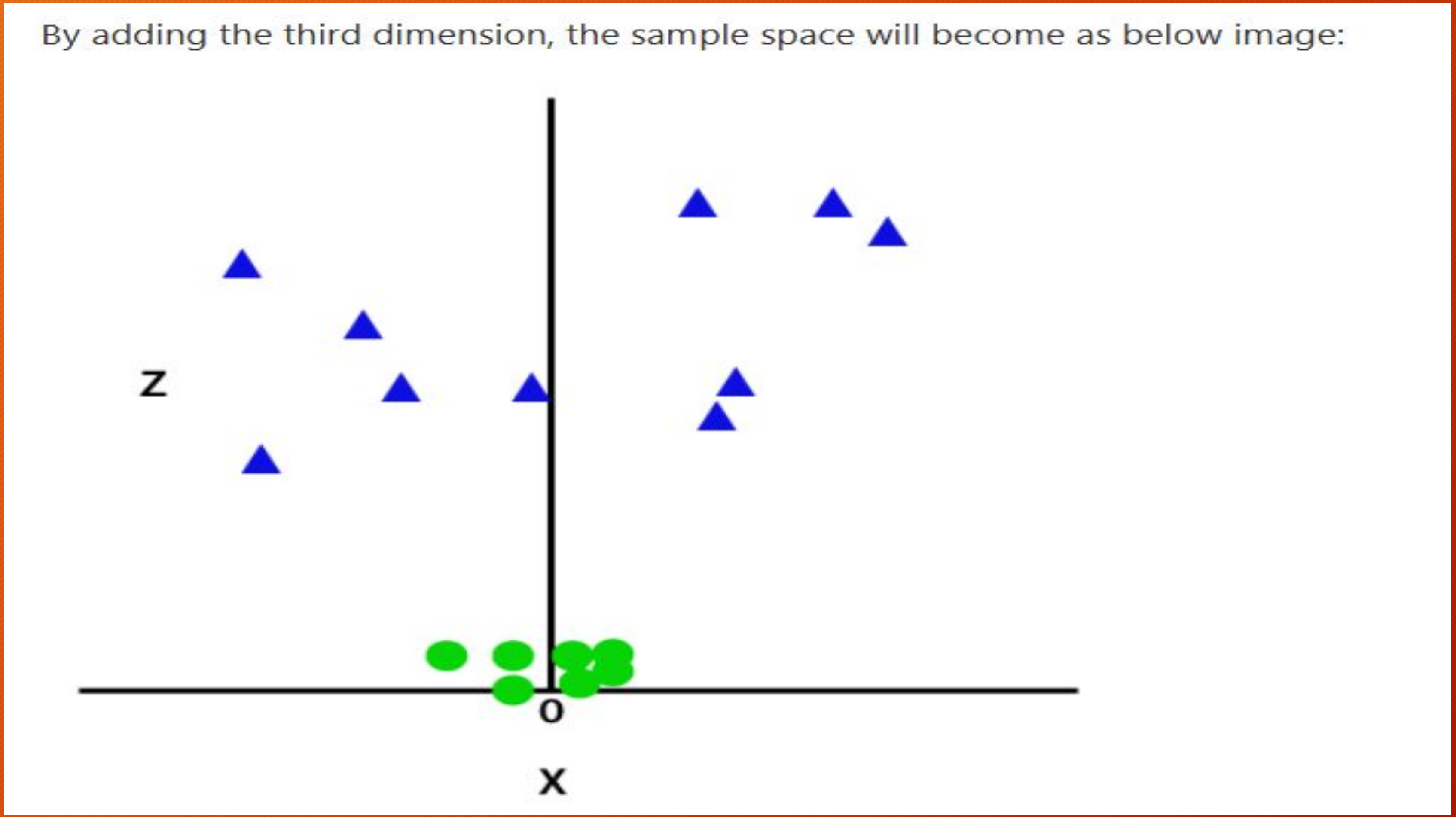
Non-Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



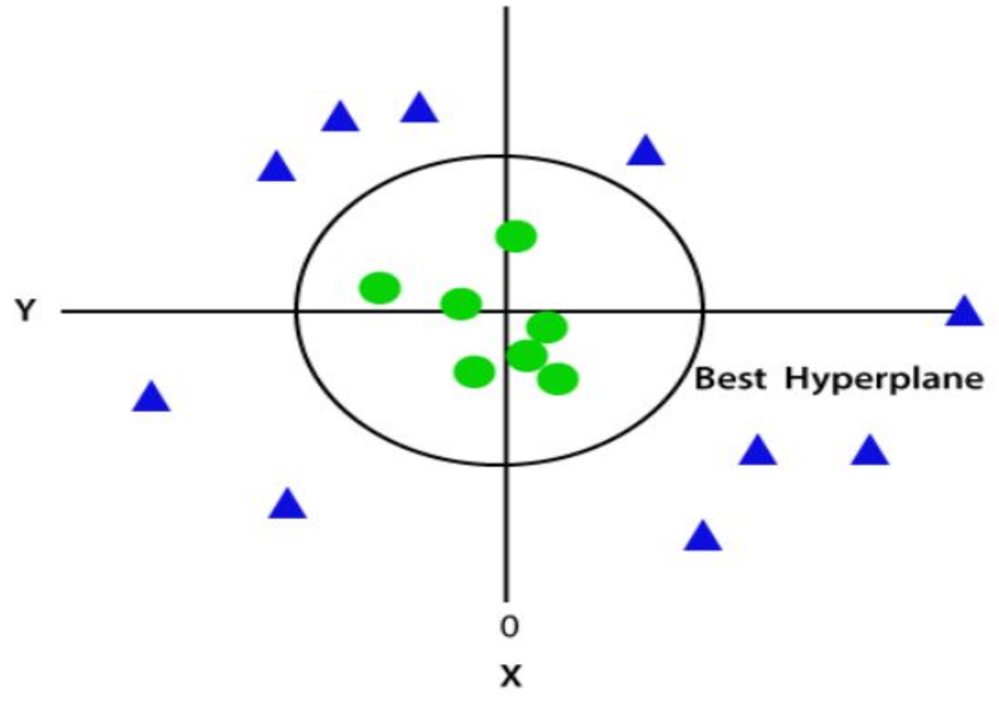
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2 +y^2$

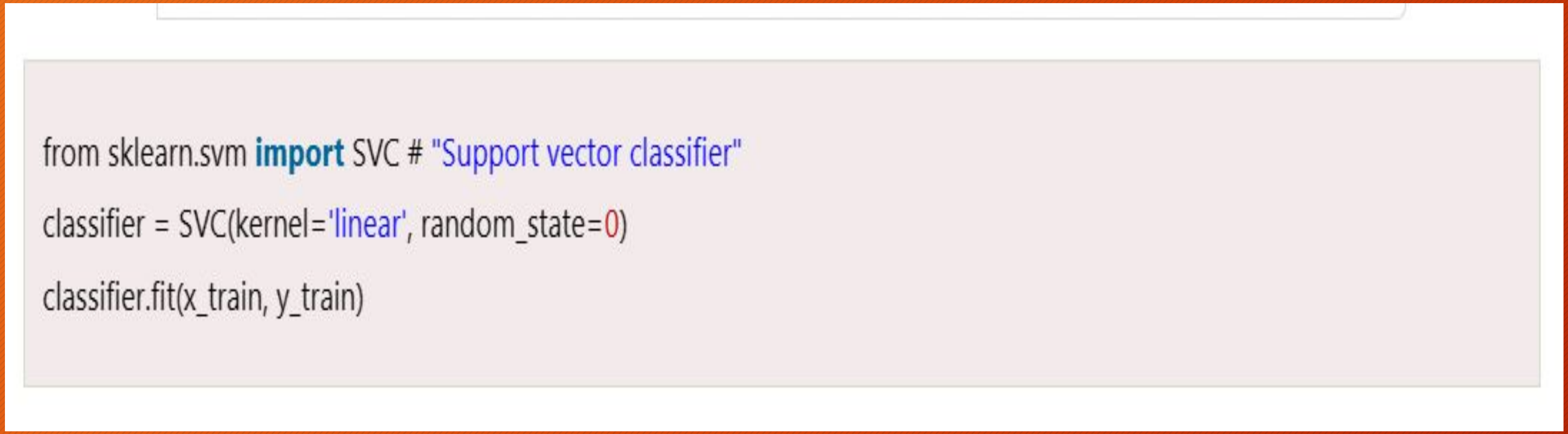

By adding the third dimension, the sample space will become as below image:

Converting 3D to 2D

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



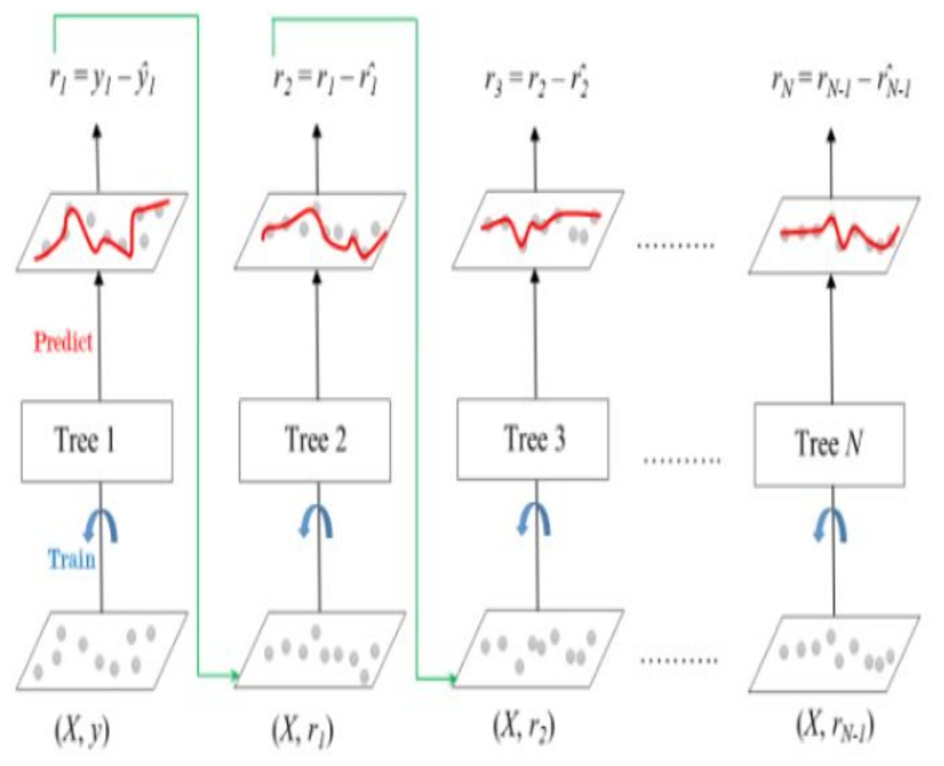
Syntax



```
from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

ML – Gradient Boosting

- **Gradient Boosting** is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor’s error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.
- There is a technique called the **Gradient Boosted Trees** whose base learner is CART (Classification and Regression Trees).
- The diagram explains how gradient boosted trees are trained for regression



Gradient Boosted Trees for Regression

Gradient Boosting

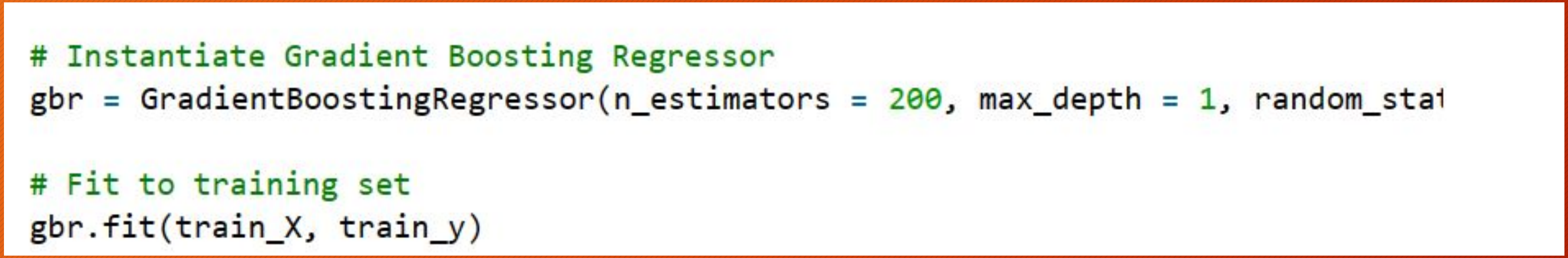
- The ensemble consists of *N* trees. Tree1 is trained using the feature matrix *X* and the labels *y*. The predictions labelled *y1(hat)* are used to determine the training set residual errors *r1*. Tree2 is then trained using the feature matrix *X* and the residual errors *r1* of Tree1 as labels. The predicted results *r1(hat)* are then used to determine the residual *r2*. The process is repeated until all the *N* trees forming the ensemble are trained.
- **Shrinkage** refers to the fact that the prediction of each tree in the ensemble is shrunk after it is multiplied by the learning rate (eta) which ranges between 0 to 1. There is a trade-off between eta and number of estimators, decreasing learning rate needs to be compensated with increasing estimators in order to reach certain model performance. Since all trees are trained now, predictions can be made.

Each tree predicts a label and final prediction is given by the formula,

```
y(pred) = y1 + (eta * r1) + (eta * r2) + ....... + (eta * rN)
```

The class of the gradient boosting regression in scikit-learn is **GradientBoostingRegressor**. A similar algorithm is used for classification known as **GradientBoostingClassifier**.

Syntax



```
# Instantiate Gradient Boosting Regressor
gbr = GradientBoostingRegressor(n_estimators = 200, max_depth = 1, random_sta

# Fit to training set
gbr.fit(train_X, train_y)
```

Unsupervised Algorithm

- Below is the list of some popular unsupervised learning algorithms:
- **K-means clustering**
- **Hierarchal clustering**
- **DBSCAN algorithm**

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 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.

K means clusturing

- 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.

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).

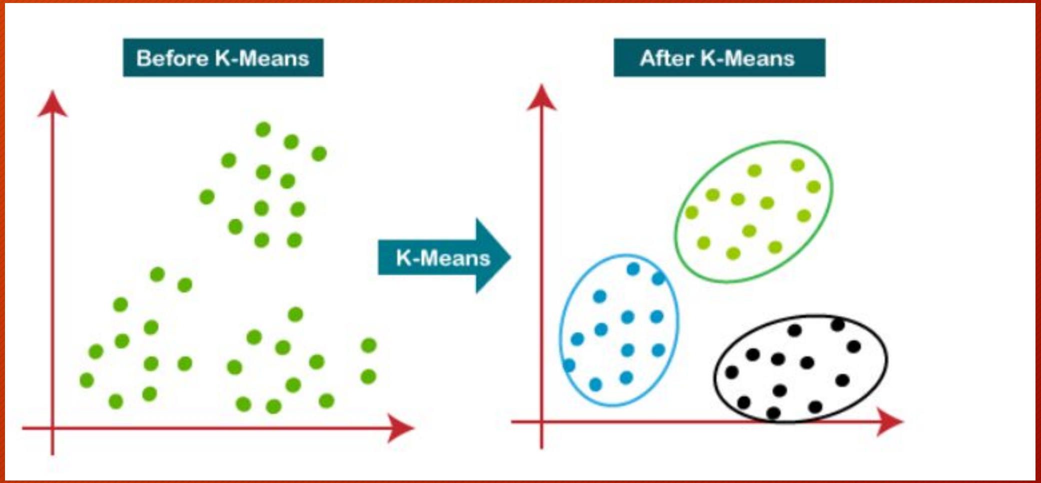
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

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.

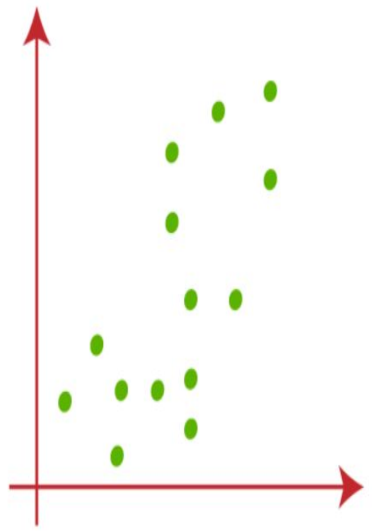
Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

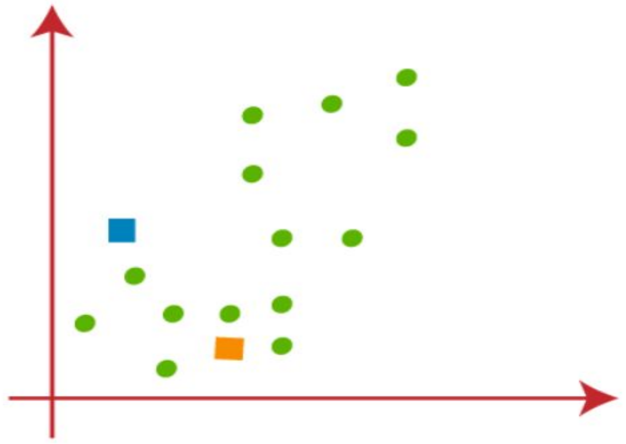


Example

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

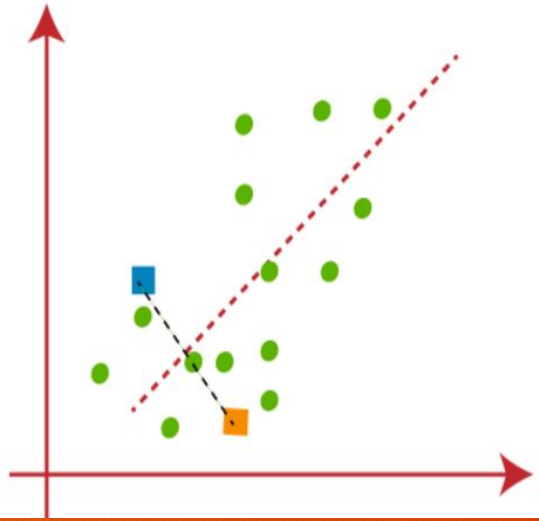


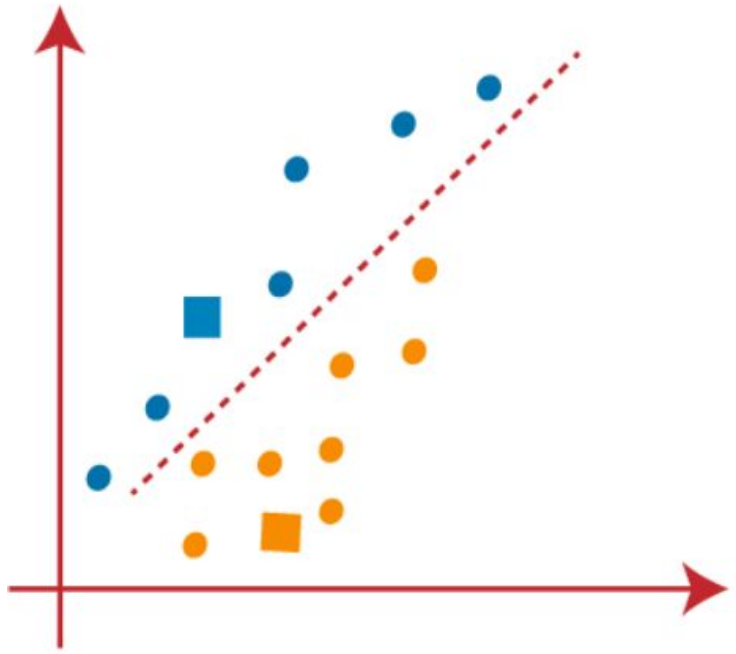
- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points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



Calculating best centroid

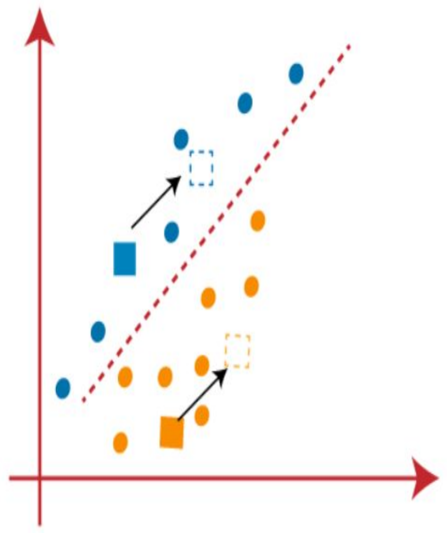
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



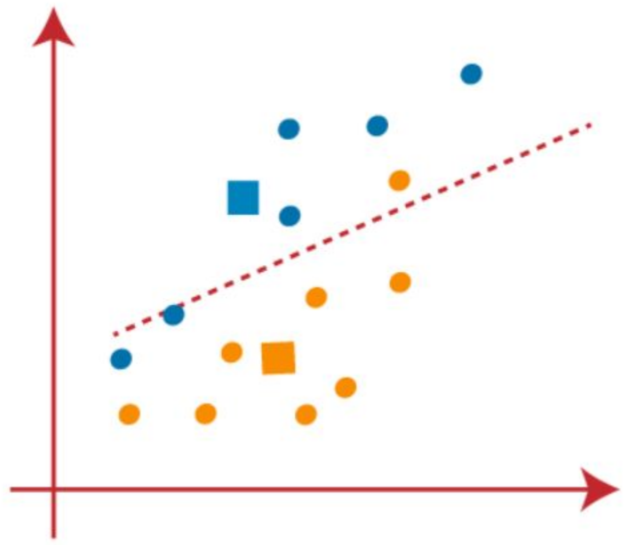


Calculating new Centroid

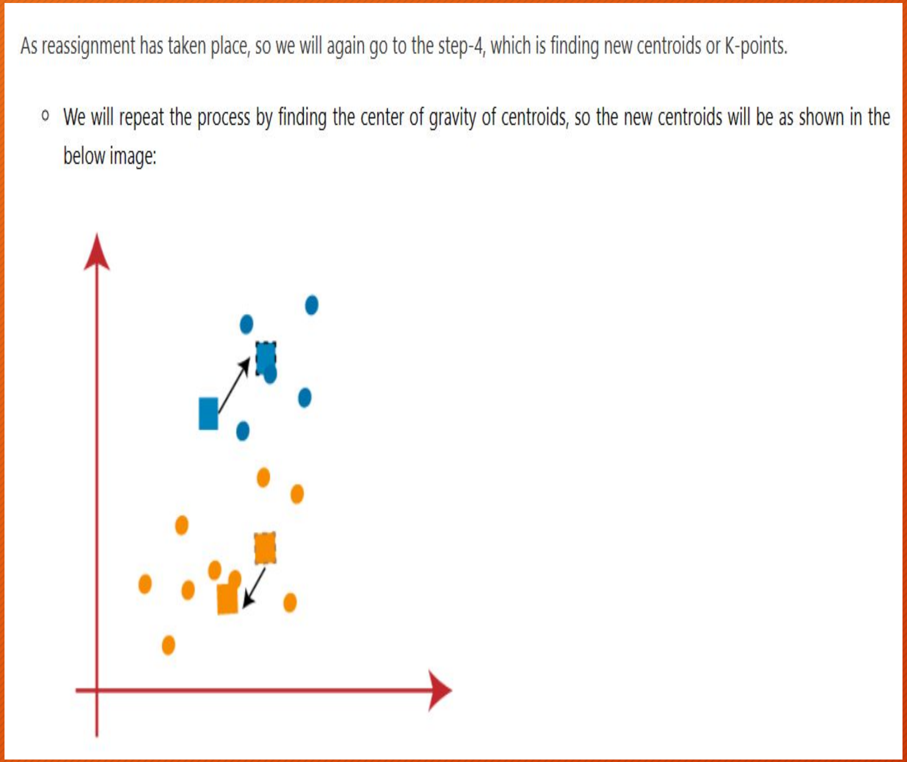
As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



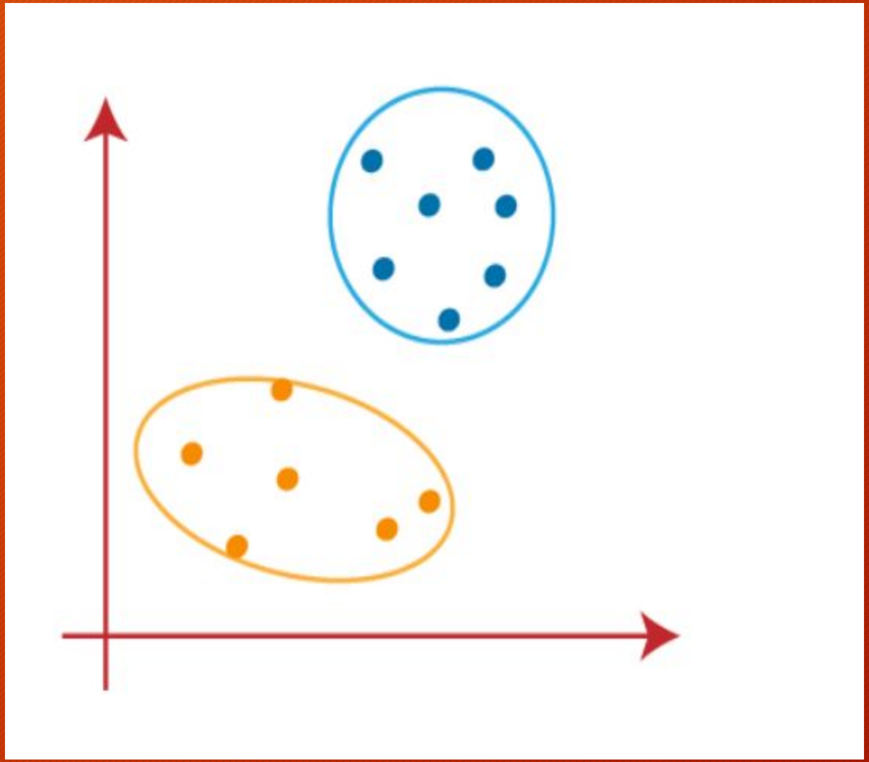
- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:



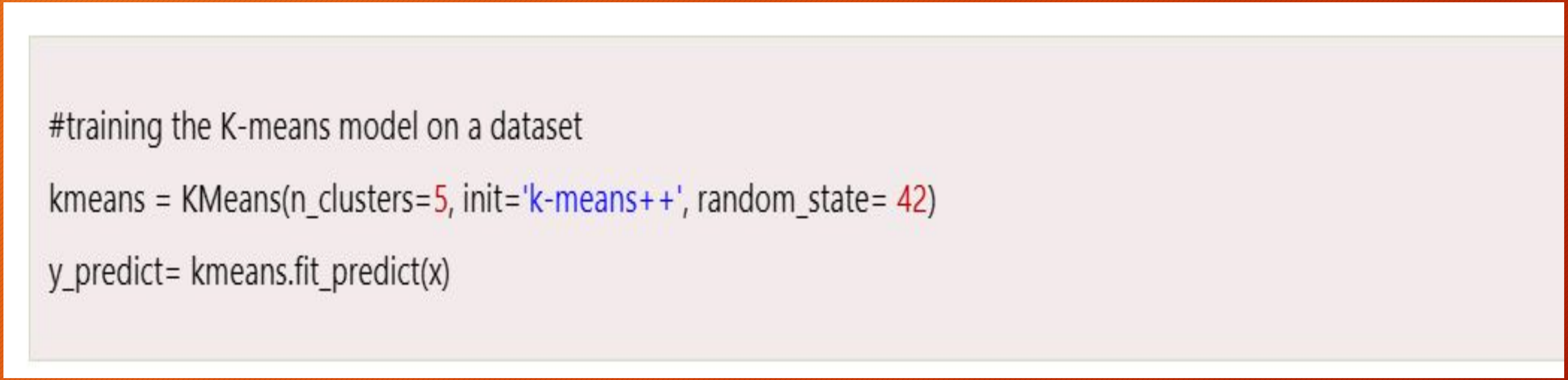
convergence


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:



Syntax

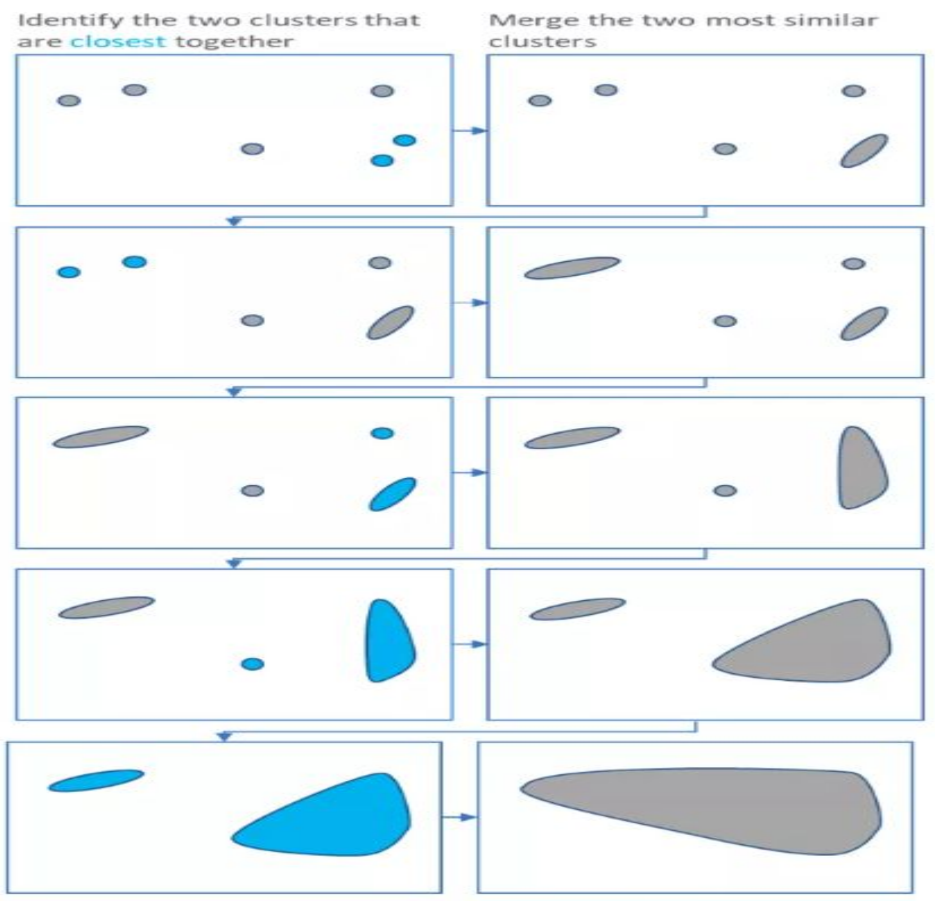


```
#training the K-means model on a dataset

kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)

y_predict= kmeans.fit_predict(x)
```

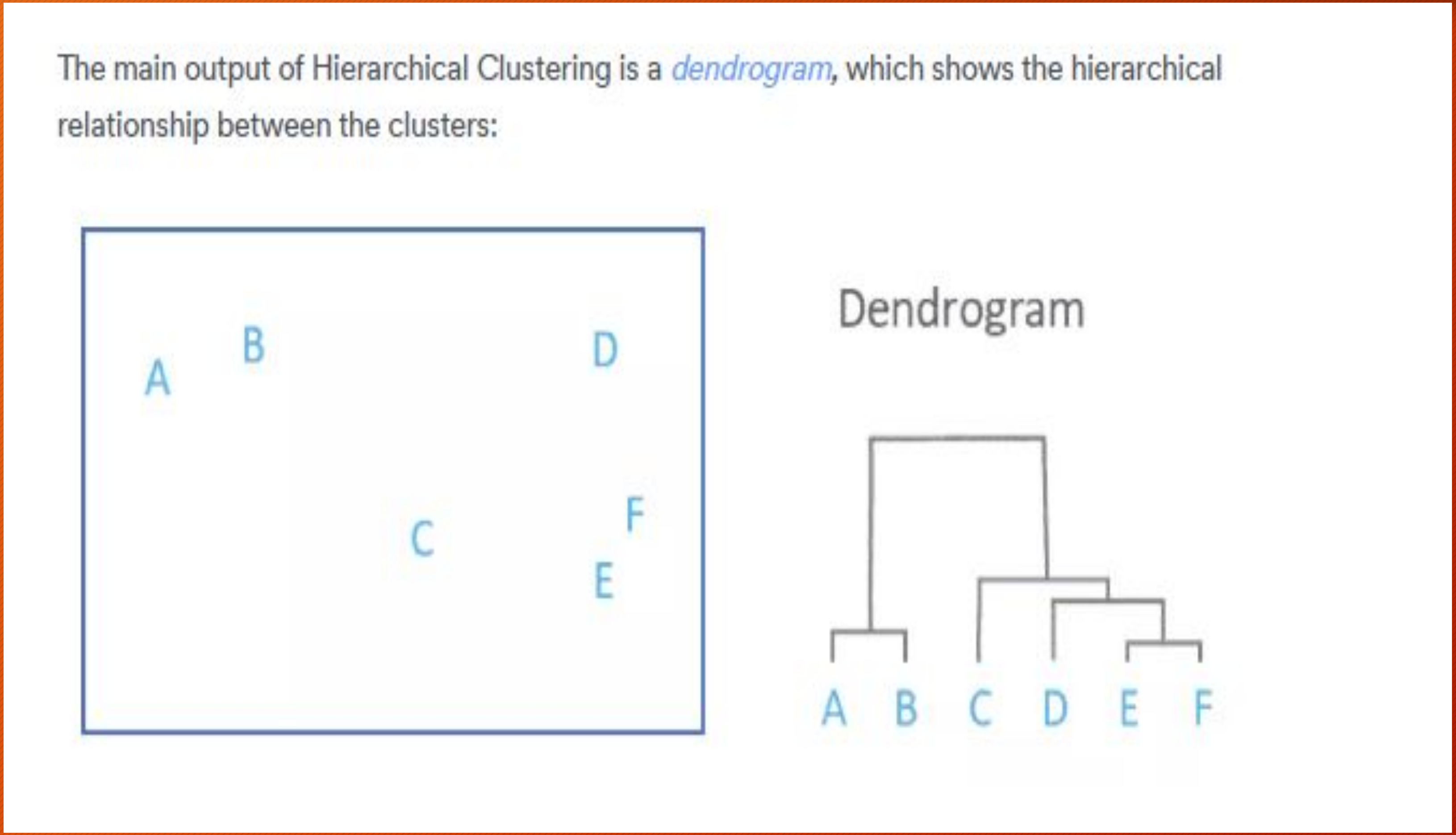
Hierarchical clustering

- ***Hierarchical clustering***, also known as *hierarchical cluster analysis,* is an algorithm that groups similar objects into groups called *clusters*. The endpoint is a set of clusters*,* where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

Hierarchical clustering


Identify the two clusters that are closest together
Merge the two most similar clusters

Dendogram



The main output of Hierarchical Clustering is a *dendrogram*, which shows the hierarchical relationship between the clusters:

Measures of distance (similarity)

- In the example above, the *distance* between two clusters has been computed based on the length of the straight line drawn from one cluster to another. This is commonly referred to as the *Euclidean distance.* Many other *distance metrics* have been developed.
- The choice of distance metric should be made based on theoretical concerns from the domain of study. That is, a distance metric needs to define similarity in a way that is sensible for the field of study. For example, if clustering crime sites in a city, city block distance may be appropriate. Or, better yet, the time taken to travel between each location. Where there is no theoretical justification for an alternative, the Euclidean should generally be preferred, as it is usually the appropriate measure of distance in the physical world.
-

Linkage Criteria

- After selecting a distance metric, it is necessary to determine from where distance is computed. For example, it can be computed between the two most similar parts of a cluster (*single-linkage*), the two least similar bits of a cluster (*complete-linkage*), the center of the clusters (*mean* or *average-linkage*), or some other criterion. Many linkage criteria have been developed.
- As with *distance metrics,* the choice of linkage criteria should be made based on theoretical considerations from the domain of application. A key theoretical issue is what causes variation. For example, in archeology, we expect variation to occur through innovation and natural resources, so working out if two groups of artifacts are similar may make sense based on identifying the most similar members of the cluster.
- Where there are no clear theoretical justifications for the choice of linkage criteria, *Ward's method* is the sensible default. This method works out which observations to group based on reducing the sum of squared distances of each observation from the average observation in a cluster. This is often appropriate as this concept of distance matches the standard assumptions of how to compute differences between groups in statistics (e.g., *ANOVA*, *MANOVA*).

TYPES

This clustering technique is divided into two types:

1. Agglomerative Hierarchical Clustering
2. Divisive Hierarchical Clustering

Agglomerative Hierarchical Clustering

The Agglomerative Hierarchical Clustering is the most common type of hierarchical clustering used to group objects in clusters based on their similarity. It’s also known as AGNES (Agglomerative Nesting). It's a “bottom-up” approach: **each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.**

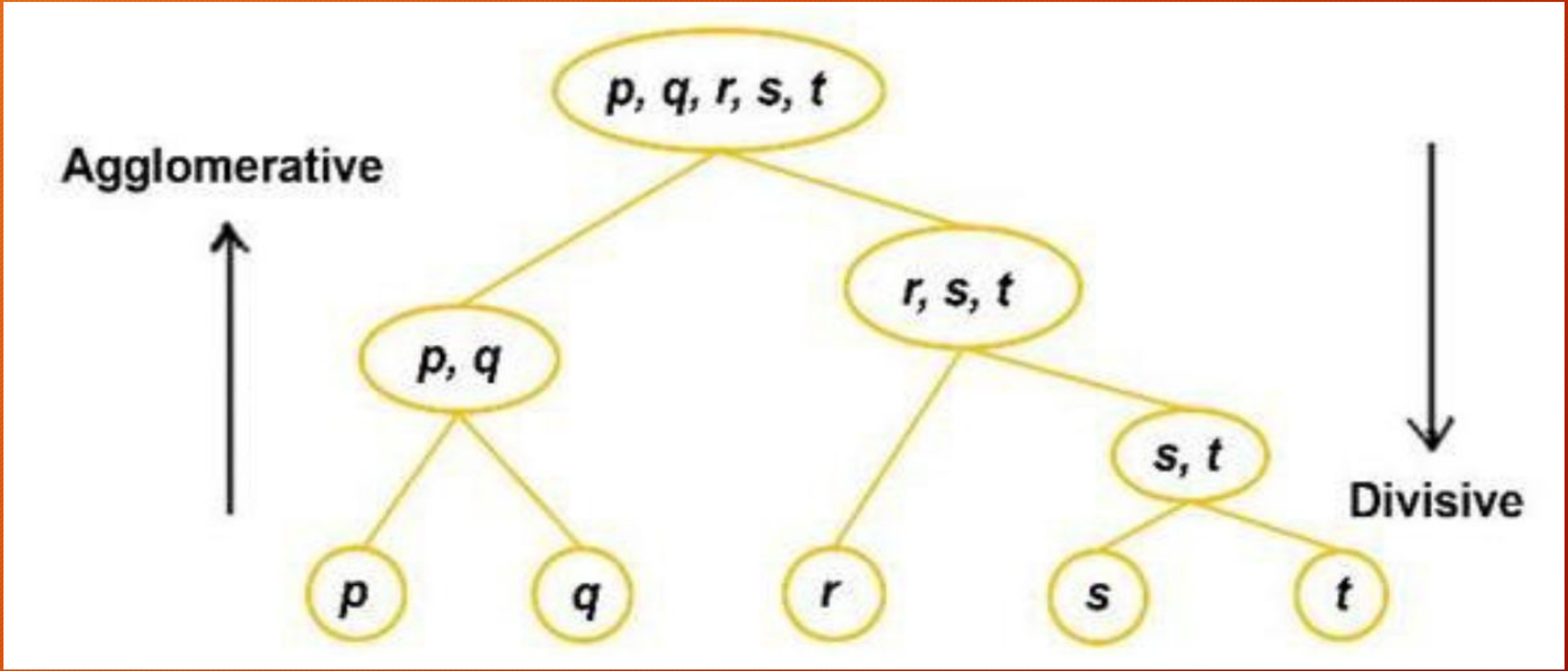
- **How does it work?**

1. Make each data point a single-point cluster → forms N clusters
2. Take the two closest data points and make them one cluster → forms N-1 clusters
3. Take the two closest clusters and make them one cluster → Forms N-2 clusters.
4. Repeat step-3 until you are left with only one cluster.

Divisive Hierarchical Clustering

Divisive or DIANA(Divisive Analysis Clustering) is a top-down clustering method where we assign all of the observations to a single cluster and then partition the cluster to two least similar clusters. Finally, we proceed recursively on each cluster until there is one cluster for each observation. So this clustering approach is exactly opposite to Agglomerative clustering.

Hierarchical clustering



DBSCAN clustering

- **Density**-Based **Clustering** refers to unsupervised learning methods that identify distinctive groups/clusters in the data, based on the idea that a cluster in data space is a contiguous region of high point density, separated from other such clusters by contiguous regions of low point density.
- Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

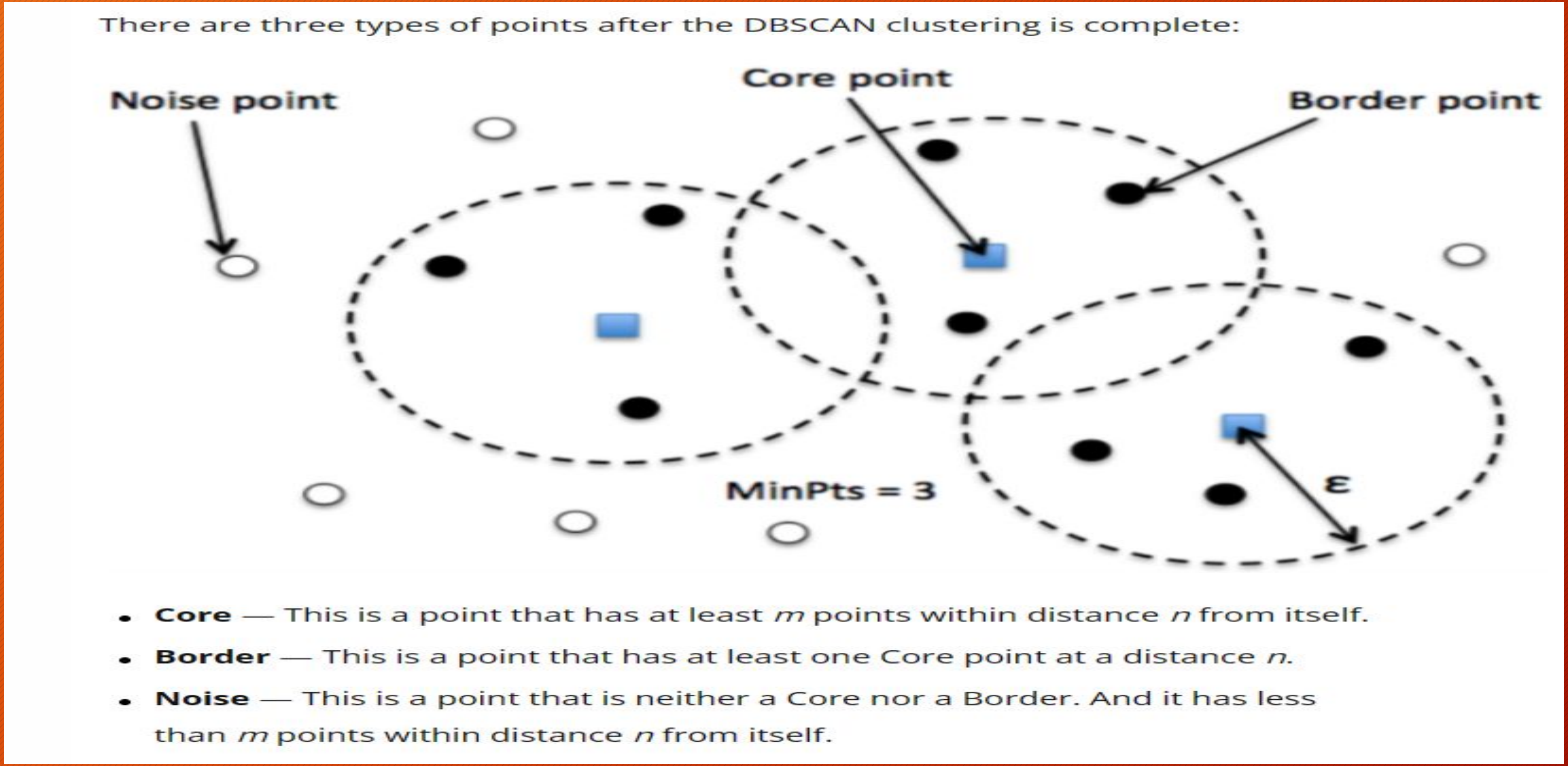
The DBSCAN algorithm uses two parameters

- **minPts:** The minimum number of points (a threshold) clustered together for a region to be considered dense.
- **eps (ε):** A distance measure that will be used to locate the points in the neighborhood of any point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity.

- **Reachability** in terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it.
- **Connectivity**, on the other hand, involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighborhood of a.

Algorithm


There are three types of points after the DBSCAN clustering is complete:

- **Core** — This is a point that has at least *m* points within distance *n* from itself.
- **Border** — This is a point that has at least one Core point at a distance *n*.
- **Noise** — This is a point that is neither a Core nor a Border. And it has less than *m* points within distance *n* from itself.

Algorithmic steps for DBSCAN clustering

- The algorithm proceeds by arbitrarily picking up a point in the dataset (until all points have been visited).
- If there are at least 'minPoint' points within a radius of 'ε' to the point then we consider all these points to be part of the same cluster.
- The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point

Parameter Estimation

- **minPts**: As a rule of thumb, a minimum minPts can be derived from the number of dimensions D in the data set, as **minPts ≥ D + 1**. The low value **minPts = 1** does not make sense, as then every point on its own will already be a cluster. With **minPts ≤ 2**, the result will be the same as of hierarchical clustering with the single link metric, with the dendrogram cut at height ε. Therefore, minPts must be chosen at least 3. However, larger values are usually better for data sets with noise and will yield more significant clusters. As a rule of thumb, **minPts = 2·dim** can be used, but it may be necessary to choose larger values for very large data, for noisy data or for data that contains many duplicates.

- ε: The value for ε can then be chosen by using a k-distance graph, plotting the distance to the **k = minPts-1** nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot shows an "elbow": if ε is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster. In general, small values of ε are preferable, and as a rule of thumb, only a small fraction of points should be within this distance of each other.

- **Distance function**: The choice of distance function is tightly linked to the choice of ε, and has a major impact on the outcomes. In general, it will be necessary to first identify a reasonable measure of similarity for the data set, before the parameter ε can be chosen. There is no estimation for this parameter, but the distance functions need to be chosen appropriately for the data set.

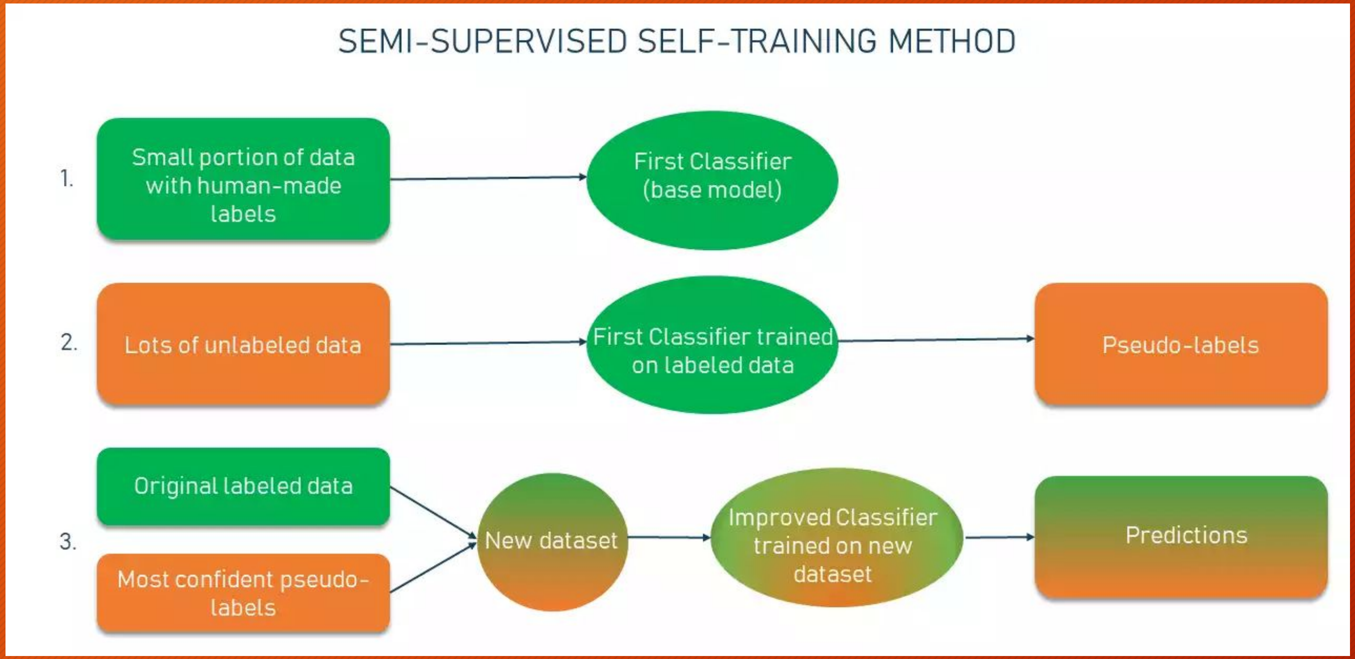
Semi-Supervised Learning

It refers to a learning problem (and algorithms designed for the learning problem) that involves a small portion of labeled examples and a large number of unlabeled examples from which a model must learn and make predictions on new examples .There are two scenario:

Inductive Learning: inductive learning refers to a learning algorithm that learns from labeled training data and generalizes to new data, such as a test dataset.

Transductive learning refers to learning from labeled training data and generalizing to available unlabeled (training) data

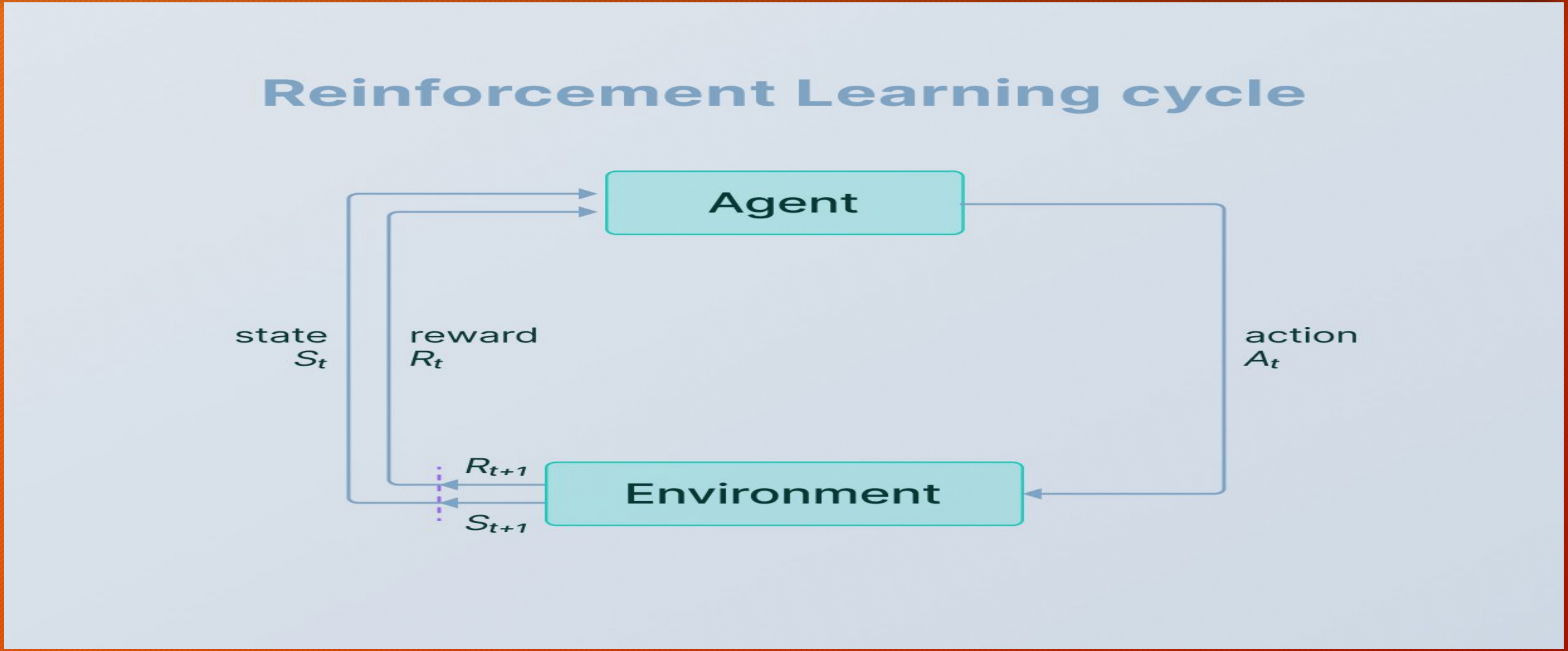
Method


SEMI-SUPERVISED SELF-TRAINING METHOD
1.
Small portion of data with human-made labels
First Classifier (base model)
2.
Lots of unlabeled data
First Classifier trained on labeled data
Pseudo-labels
3.
Original labeled data
Most confident pseudo-labels
New dataset
Improved Classifier trained on new dataset
Predictions

Reinforcement Learning

Reinforcement learning (RL) is a subset of machine learning that allows an AI-driven system (sometimes referred to as an agent) to learn through trial and error using feedback from its actions.

RL Cycle


Reinforcement Learning cycle
Agent
Environment
state S_t
reward R_t
action A_t
R_{t+1}
S_{t+1}

RL Labels

- **Agent** - Agent (A) takes actions that affect the environment. Citing an example, the machine learning to play chess is the *agent*.
- **Action** - It is the set of all possible operations/moves the agent can make. The agent makes a decision on which action to take from a set of discrete actions (a).
- **Environment** - All actions that the reinforcement learning agent makes directly affect the environment. Here, the board of chess is the environment. The environment takes the agent's present state and action as information and returns the reward to the agent with a new state.
- For example, the move made by the bot will either have a negative/positive effect on the whole game and the arrangement of the board. This will decide the next action and state of the board.
- **State** - A state (S) is a particular situation in which the agent finds itself.
- **Reward** (R) - The environment gives feedback by which we determine the validity of the agent’s actions in each state. It is crucial in the scenario of Reinforcement Learning where we want the machine to learn all by itself and the only critic that would help it in learning is the feedback/reward it receives.
- For example, in a chess game scenario it happens when the bot takes the place of an opponent's piece and later captures it.