Unit 02

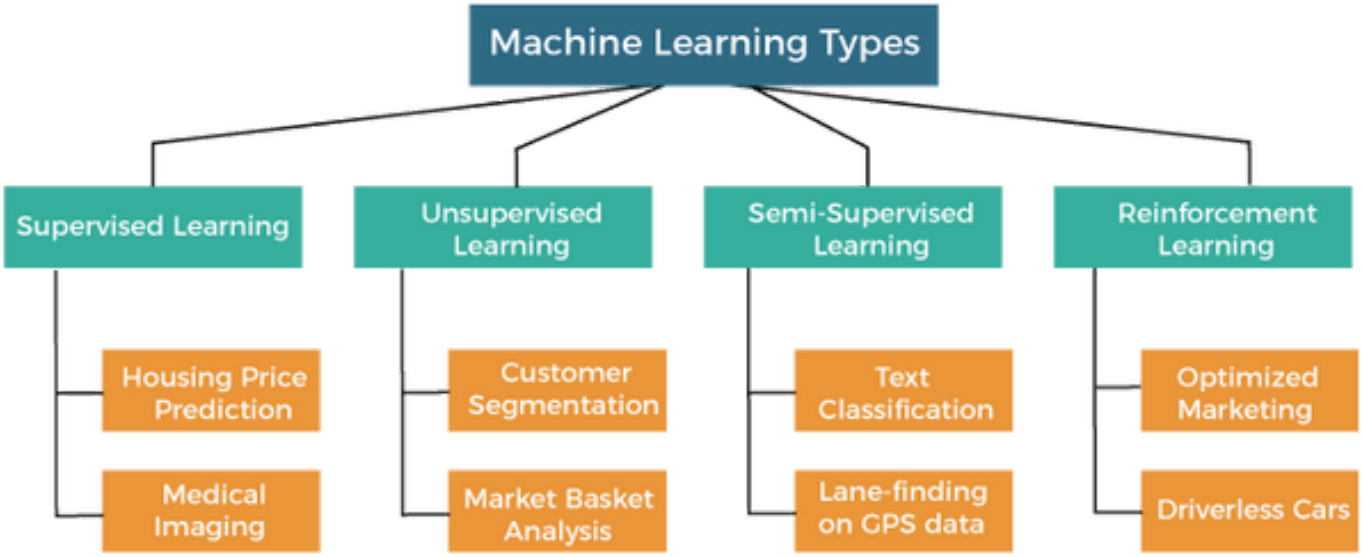
Types of Machine Learning

Machine learning is a subset of AI, which enables the machine to automatically learn from data, improve performance from past experiences, and make predictions. Machine learning contains a set of algorithms that work on a huge amount of data. Data is fed to these algorithms to train them, and on the basis of training, they build the model & perform a specific task.

These ML algorithms help to solve different business problems like Regression, Classification, Forecasting, Clustering, and Associations, etc.

Based on the methods and way of learning, machine learning is divided into mainly four types, which are:

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-Supervised Machine Learning
4. Reinforcement Learning



In this topic, we will provide a detailed description of the types of Machine Learning along with their respective algorithms:

1. Supervised Machine Learning

As its name suggests, Supervised machine learning is based on supervision. It means in the supervised learning technique, we train the machines using the "labelled" dataset, and based on the training, the machine predicts the output. Here, the labelled data specifies that some of the inputs are already mapped to the output. More preciously, we can say; first, we train the machine with the input and corresponding output, and then we ask the machine to predict the output using the test dataset.

Let's understand supervised learning with an example. Suppose we have an input dataset of cats and dog images. So, first, we will provide the training to the machine to understand the images, such as the shape & size of the tail of cat and dog, Shape of eyes, colour, height (dogs are taller, cats are smaller), etc. After completion of training, we input the picture of a cat and ask the machine to identify the object and predict the output. Now, the machine is well trained, so it will check all the features of the object, such as height, shape, colour, eyes, ears, tail, etc., and find that it's a cat. So, it will put it in the Cat category. This is the process of how the machine identifies the objects in Supervised Learning.

The main goal of the supervised learning technique is to map the input variable(x) with the output variable(y). Some real-world applications of supervised learning are Risk Assessment, Fraud Detection, Spam filtering, etc.

Categories of Supervised Machine Learning

Supervised machine learning can be classified into two types of problems, which are given below:

- Classification
- Regression

a) Classification

Classification algorithms are used to solve the classification problems in which the output variable is categorical, such as "Yes" or No, Male or Female, Red or Blue, etc. The classification algorithms predict the categories present in the dataset. Some real-world examples of classification algorithms are Spam Detection, Email filtering, etc.

Some popular classification algorithms are given below:

- Random Forest Algorithm
- Decision Tree Algorithm
- Logistic Regression Algorithm
- Support Vector Machine Algorithm

b) Regression

Regression algorithms are used to solve regression problems in which there is a linear relationship between input and output variables. These are used to predict continuous output variables, such as market trends, weather prediction, etc.

Some popular Regression algorithms are given below:

- Simple Linear Regression Algorithm
- Multivariate Regression Algorithm
- Decision Tree Algorithm
- Lasso Regression

Advantages and Disadvantages of Supervised Learning

Advantages:

- Since supervised learning work with the labelled dataset so we can have an exact idea about the classes of objects.
- These algorithms are helpful in predicting the output on the basis of prior experience.

Disadvantages:

- These algorithms are not able to solve complex tasks.
- It may predict the wrong output if the test data is different from the training data.
- It requires lots of computational time to train the algorithm.

Applications of Supervised Learning

Some common applications of Supervised Learning are given below:

- Image Segmentation:Supervised Learning algorithms are used in image segmentation. In this process, image classification is performed on different image data with pre-defined labels.

- Medical Diagnosis:Supervised algorithms are also used in the medical field for diagnosis purposes. It is done by using medical images and past labelled data with labels for disease conditions. With such a process, the machine can identify a disease for the new patients.

- Fraud Detection - Supervised Learning classification algorithms are used for identifying fraud transactions, fraud customers, etc. It is done by using historic data to identify the patterns that can lead to possible fraud.
- Spam detection - In spam detection & filtering, classification algorithms are used. These algorithms classify an email as spam or not spam. The spam emails are sent to the spam folder.
- Speech Recognition - Supervised learning algorithms are also used in speech recognition. The algorithm is trained with voice data, and various identifications can be done using the same, such as voice-activated passwords, voice commands, etc.

2. Unsupervised Machine Learning

Unsupervised learning is different from the Supervised learning technique; as its name suggests, there is no need for supervision. It means, in unsupervised machine learning, the machine is trained using the unlabeled dataset, and the machine predicts the output without any supervision.

In unsupervised learning, the models are trained with the data that is neither classified nor labelled, and the model acts on that data without any supervision.

The main aim of the unsupervised learning algorithm is to group or categories the unsorted dataset according to the similarities, patterns, and differences. Machines are instructed to find the hidden patterns from the input dataset.

Let's take an example to understand it more preciously; suppose there is a basket of fruit images, and we input it into the machine learning model. The images are totally unknown to the model, and the task of the machine is to find the patterns and categories of the objects.

So, now the machine will discover its patterns and differences, such as colour difference, shape difference, and predict the output when it is tested with the test dataset.

Categories of Unsupervised Machine Learning

Unsupervised Learning can be further classified into two types, which are given below:

- Clustering
- Association

1) Clustering

The clustering technique is used when we want to find the inherent groups from the data. It is a way to group the objects into a cluster such that the objects with the most similarities remain in one group and have fewer or no similarities with the objects of other groups. An example of the clustering algorithm is grouping the customers by their purchasing behaviour.

Some of the popular clustering algorithms are given below:

- K-Means Clustering algorithm
- Mean-shift algorithm
- DBSCAN Algorithm
- Principal Component Analysis
- Independent Component Analysis

2) Association

Association rule learning is an unsupervised learning technique, which finds interesting relations among variables within a large dataset. The main aim of this learning algorithm is to find the dependency of one data item on another data item and map those variables accordingly so that it can generate maximum profit. This algorithm is mainly applied in Market Basket analysis, Web usage mining, continuous production, etc.

Some popular algorithms of Association rule learning are Apriori Algorithm, Eclat, FP-growth algorithm.

Advantages and Disadvantages of Unsupervised Learning Algorithm

Advantages:

- These algorithms can be used for complicated tasks compared to the supervised ones because these algorithms work on the unlabeled dataset.
- Unsupervised algorithms are preferable for various tasks as getting the unlabeled dataset is easier as compared to the labelled dataset.

Disadvantages:

- The output of an unsupervised algorithm can be less accurate as the dataset is not labelled, and algorithms are not trained with the exact output in prior.
- Working with Unsupervised learning is more difficult as it works with the unlabelled dataset that does not map with the output.

Applications of Unsupervised Learning

- Network Analysis: Unsupervised learning is used for identifying plagiarism and copyright in document network analysis of text data for scholarly articles.
- Recommendation Systems: Recommendation systems widely use unsupervised learning techniques for building recommendation applications for different web applications and e-commerce websites.
- Anomaly Detection: Anomaly detection is a popular application of unsupervised learning, which can identify unusual data points within the dataset. It is used to discover fraudulent transactions.
- Singular Value Decomposition: Singular Value Decomposition or SVD is used to extract particular information from the database. For example, extracting information of each user located at a particular location.

3. Semi-Supervised Learning

Semi-Supervised learning is a type of Machine Learning algorithm that lies between Supervised and Unsupervised machine learning. It represents the intermediate ground between Supervised (With Labelled training data) and Unsupervised learning (with no labelled training data) algorithms and uses the combination of labelled and unlabeled datasets during the training period.

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 corporate purposes, they may have few labels. It is completely different from supervised and unsupervised learning as they are based on the presence & absence of labels.

To overcome the drawbacks of supervised learning and unsupervised learning algorithms, the concept of Semi-supervised learning is introduced. The main aim of semi-supervised learning is to effectively use all the available data, rather than only labelled data like in supervised learning. Initially, similar data is clustered along with an unsupervised learning algorithm, and further, it helps to label

the unlabeled data into labelled data. It is because labelled 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-analysing the same concept without any help from the instructor, it comes under unsupervised learning. Under semi-supervised learning, the student has to revise himself after analyzing the same concept under the guidance of an instructor at college.

Advantages and disadvantages of Semi-supervised Learning

Advantages:

- It is simple and easy to understand the algorithm.
- It is highly efficient.
- It is used to solve drawbacks of Supervised and Unsupervised Learning algorithms.

Disadvantages:

- Iterations results may not be stable.
- We cannot apply these algorithms to network-level data.
- Accuracy is low.

4. Reinforcement Learning

Reinforcement learning works on a feedback-based process, in which an AI agent (A software component) automatically explore its surrounding by hitting & trail, taking action, learning from experiences, and improving its performance. Agent gets rewarded for each good action and get punished for each bad action; hence the goal of reinforcement learning agent is to maximize the rewards.

In reinforcement learning, there is no labelled data like supervised learning, and agents learn from their experiences only.

The reinforcement learning process is similar to a human being; for example, a child learns various things by experiences in his day-to-day life. An example of reinforcement learning is to play a game, where the Game is the environment, moves of an agent at each step define states, and the goal of the agent is to get a high score. Agent receives feedback in terms of punishment and rewards.

Due to its way of working, reinforcement learning is employed in different fields such as Game theory, Operation Research, Information theory, multi-agent systems.

A reinforcement learning problem can be formalized using Markov Decision Process(MDP). In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.

Categories of Reinforcement Learning

Reinforcement learning is categorized mainly into two types of methods/algorithms:

- Positive Reinforcement Learning: Positive reinforcement learning specifies increasing the tendency that the required behaviour would occur again by adding something. It enhances the strength of the behaviour of the agent and positively impacts it.
- Negative Reinforcement Learning: Negative reinforcement learning works exactly opposite to the positive RL. It increases the tendency that the specific behaviour would occur again by avoiding the negative

condition.

Real-world Use cases of Reinforcement Learning

- Video Games:RL algorithms are much popular in gaming applications. It is used to gain super-human performance. Some popular games that use RL algorithms are
  AlphaGO and AlphaGO Zero.
- Resource Management:The "Resource Management with Deep Reinforcement Learning" paper showed that how to use RL in computer to automatically learn and schedule resources to wait for different jobs in order to minimize average job slowdown.

- Robotics:RL is widely being used in Robotics applications. Robots are used in the industrial and manufacturing area, and these robots are made more powerful with reinforcement learning. There are different industries that have their vision of building intelligent robots using AI and Machine learning technology.

- Text MiningText-mining, one of the great applications of NLP, is now being implemented with the help of Reinforcement Learning by
  Salesforce company.

Advantages and Disadvantages of Reinforcement Learning

Advantages

- It helps in solving complex real-world problems which are difficult to be solved by general techniques.
- The learning model of RL is similar to the learning of human beings; hence most accurate results can be found.
- Helps in achieving long term results.

Disadvantage

- RL algorithms are not preferred for simple problems.
- RL algorithms require huge data and computations.
- Too much reinforcement learning can lead to an overload of states which can weaken the results.

The curse of dimensionality limits reinforcement learning for real physical systems.

Supervised Machine Learning

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

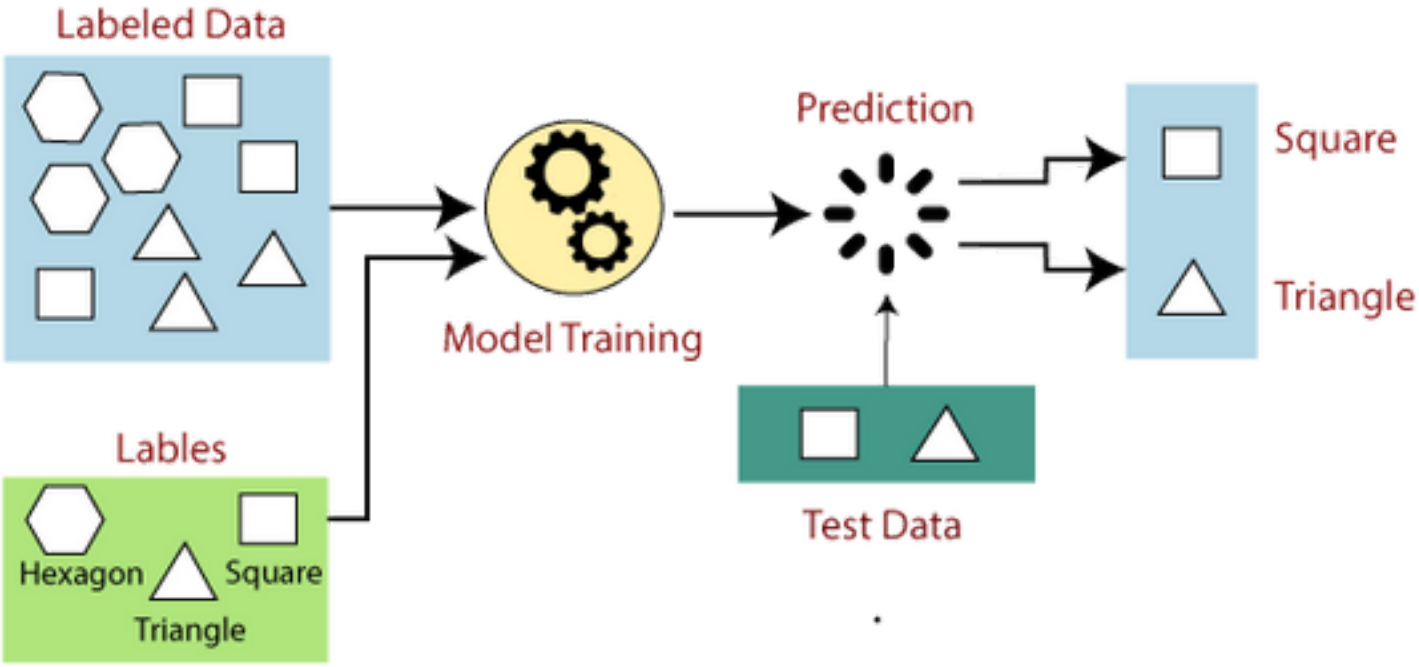
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

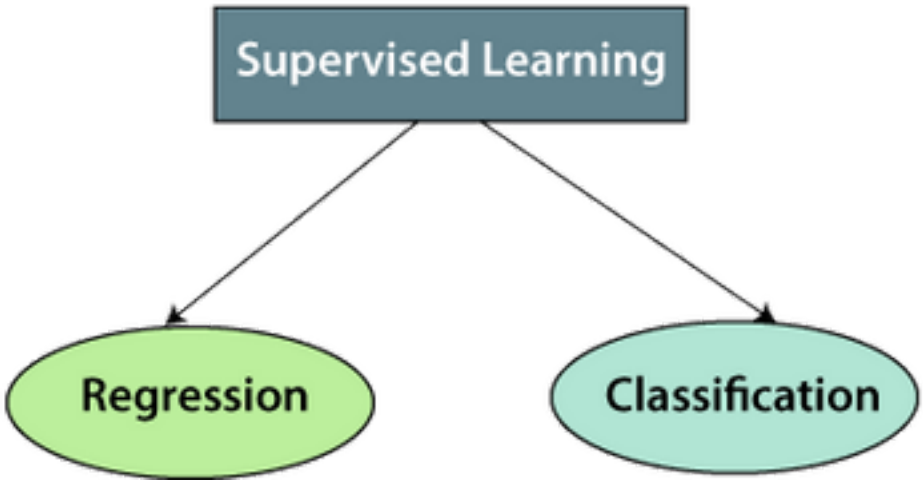
Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.
- Split the training dataset into training **dataset, test dataset, and validation dataset**.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.

- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees
- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Note: We will discuss these algorithms in detail in later chapters.

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

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:

Working of Naïve Bayes' Classifier can be understood with the help of the below example:

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
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

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.

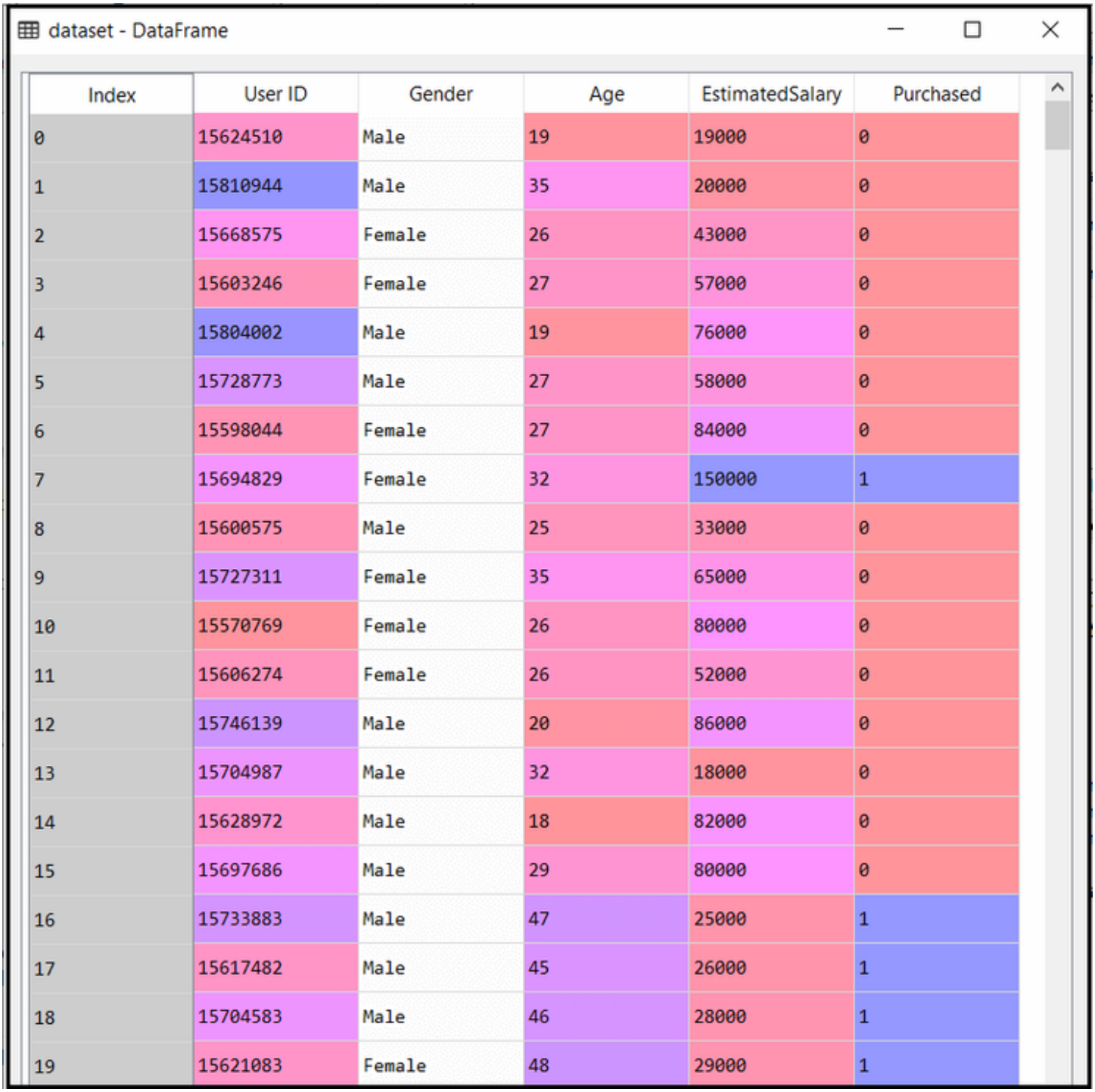
1) Data Pre-processing step:

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:

```
1. Importing the libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. # Importing the dataset
7. dataset = pd.read_csv('user_data.csv')
8. x = dataset.iloc[:, [2, 3]].values
9. y = dataset.iloc[:, 4].values
10.
11. # Splitting the dataset into the Training set and Test set
12. from sklearn.model_selection import train_test_split
13. x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)
14.
15. # Feature Scaling
16. from sklearn.preprocessing import StandardScaler
17. sc = StandardScaler()
18. x_train = sc.fit_transform(x_train)
19. 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:

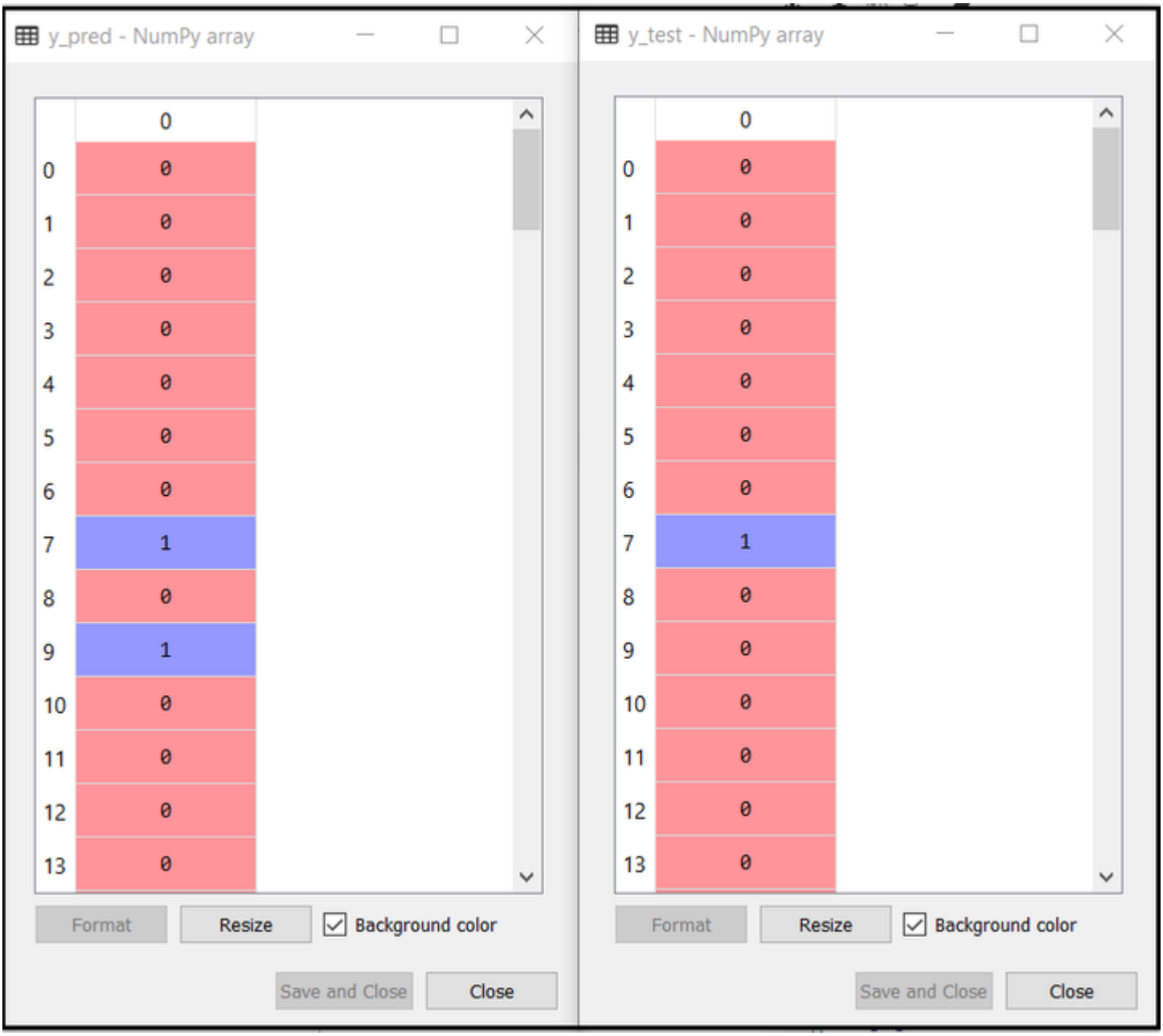
Out[6]: GaussianNB(priors=None, var_smoothing=1e-09)

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:



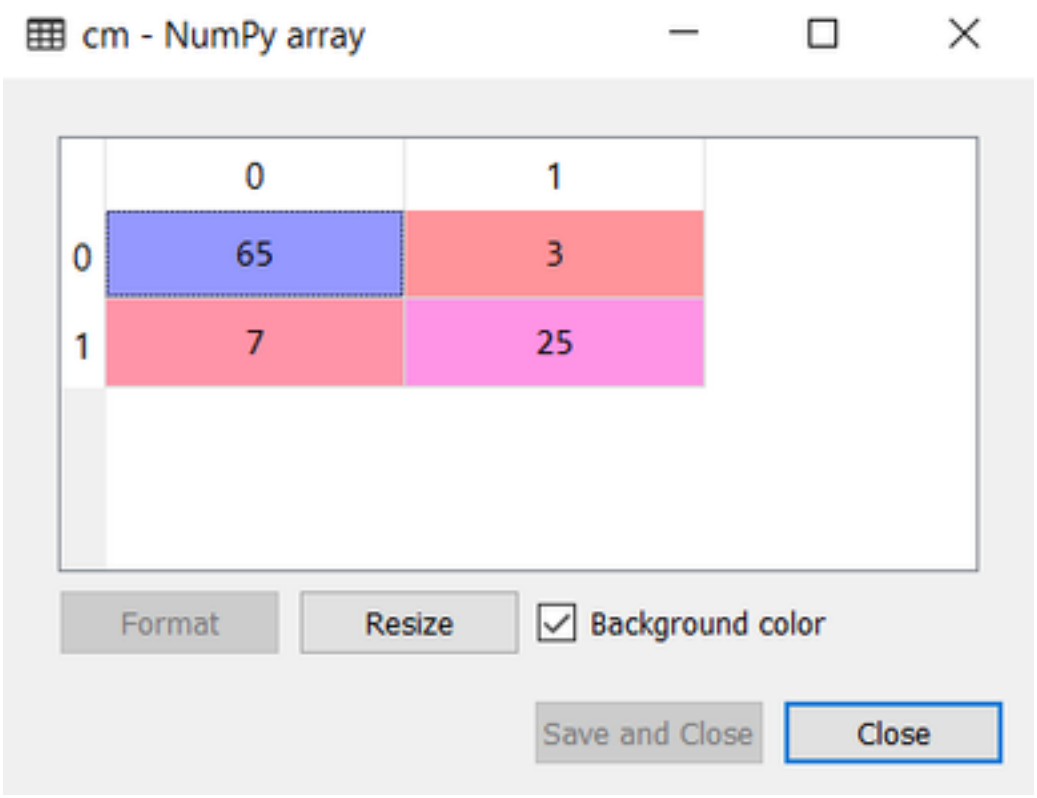
The above output shows the result for prediction vector y_pred and real vector y_test. We can see that some predications are different from the real values, which are the incorrect predictions.

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.

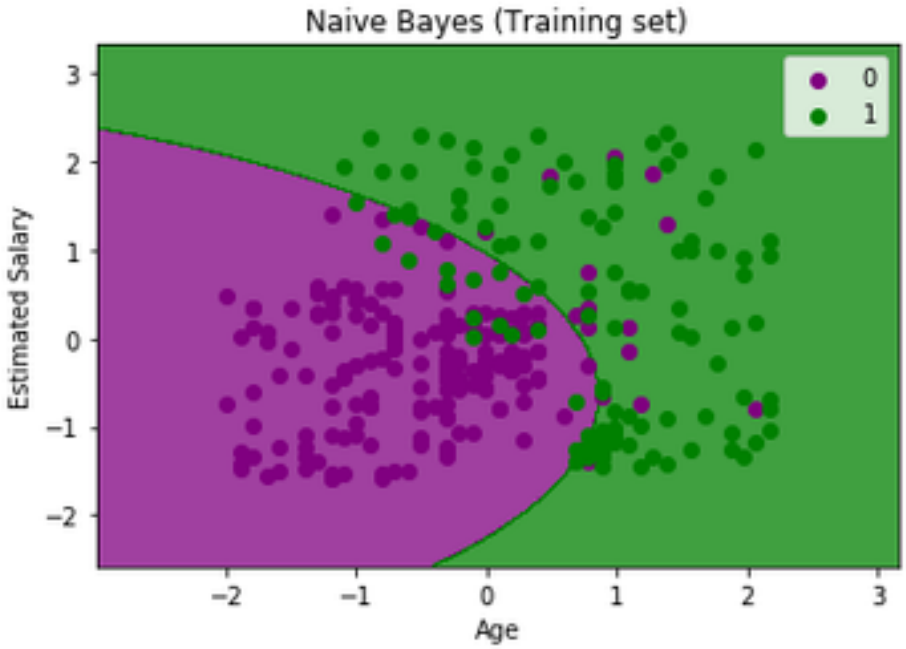
5) Visualizing the training set result:

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

```
1. # Visualising the Training set results
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_train, y_train
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('Naive Bayes (Training set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()
```

Output:



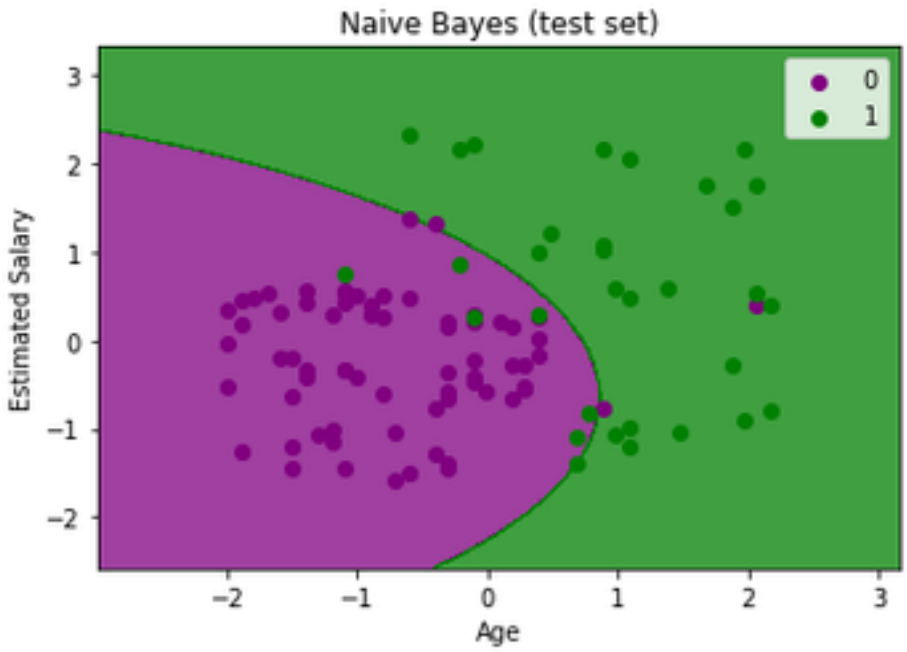
In the above output we can see that the Naïve Bayes classifier has segregated the data points with the fine boundary. It is Gaussian curve as we have used GaussianNB classifier in our code.

6) Visualizing the Test set result:

```
1.  # Visualising the Test set results
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('Naive Bayes (test set)')
```

```
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()
```

Output:



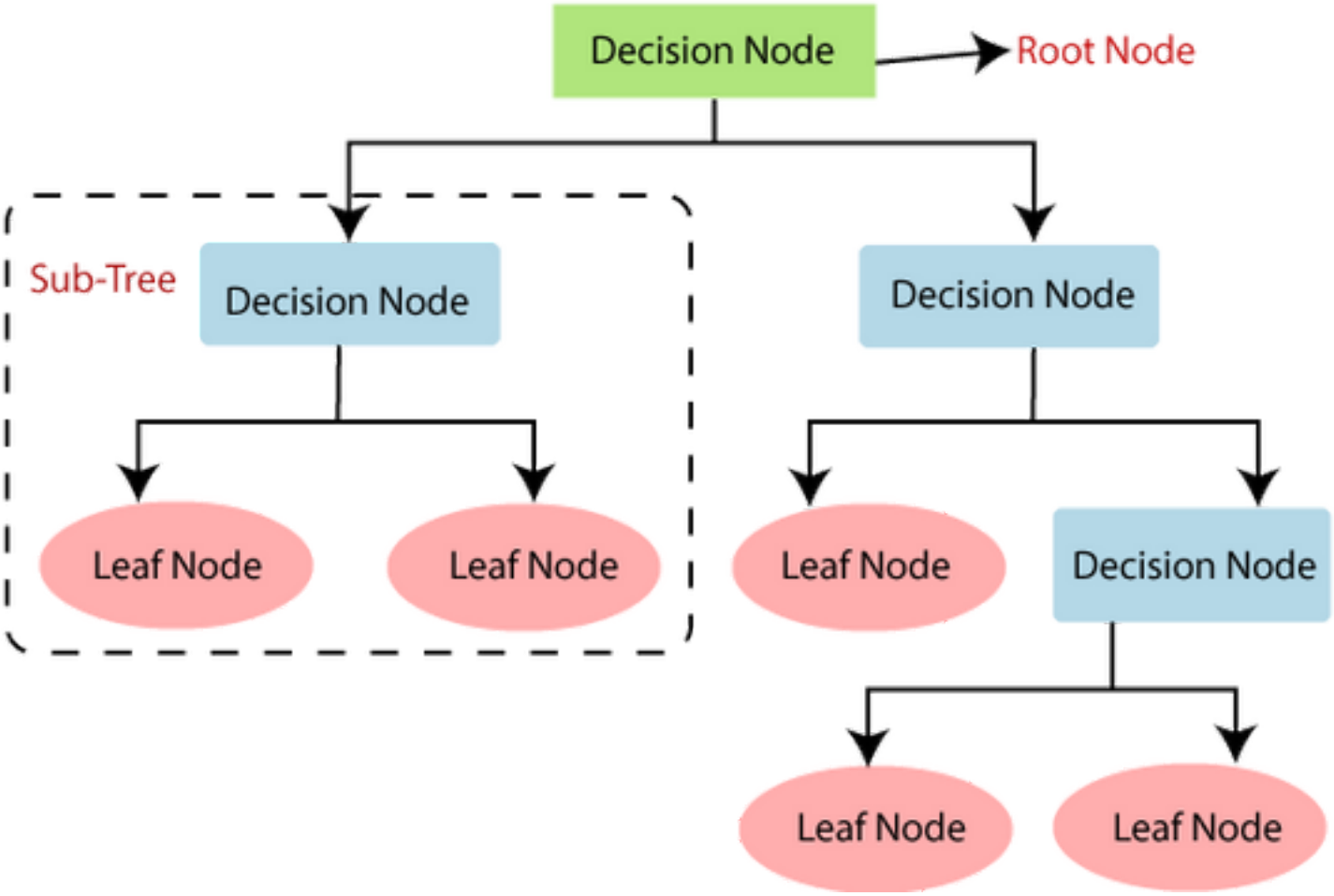
The above output is final output for test set data. As we can see the classifier has created a Gaussian curve to divide the "purchased" and "not purchased" variables. There are some wrong predictions which we have calculated in Confusion matrix. But still it is pretty good classifier.

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:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.



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.

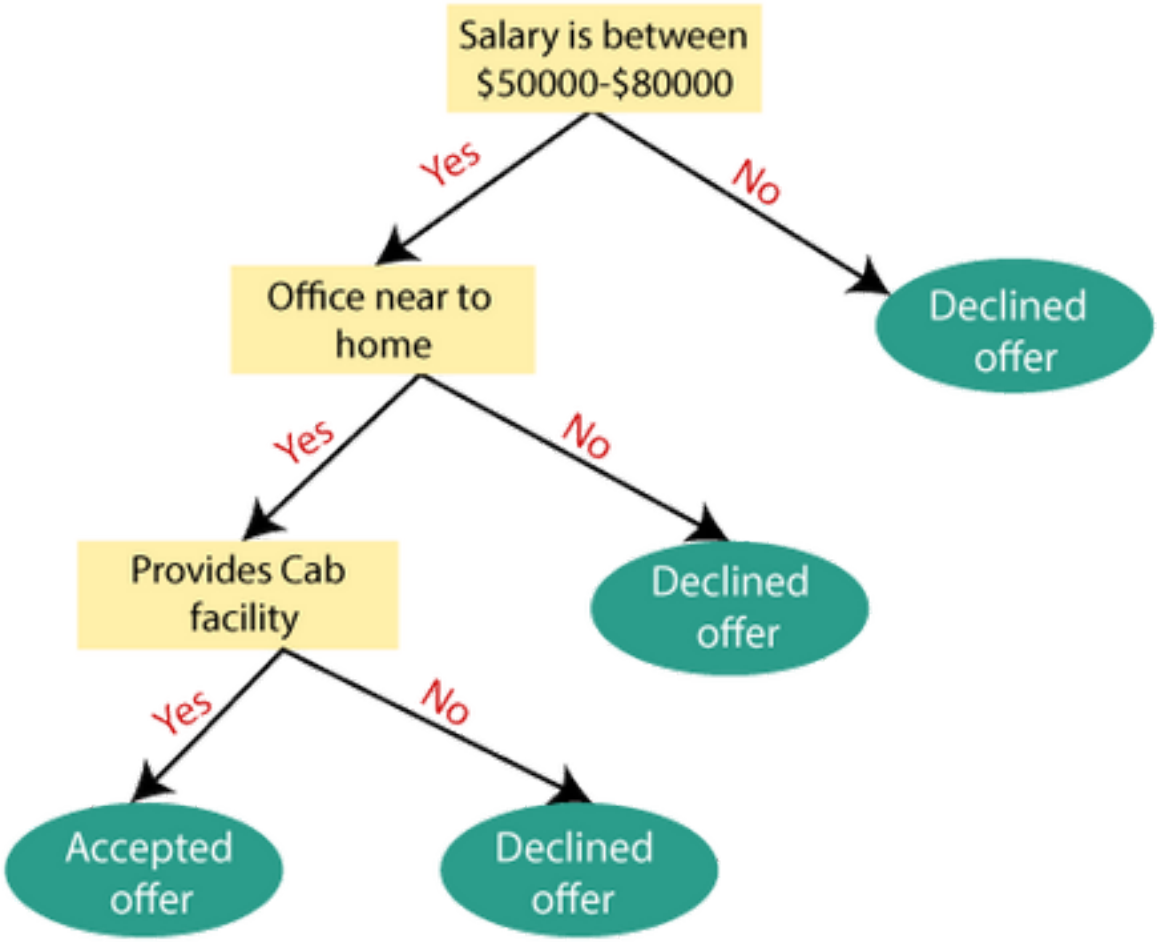
How does the Decision Tree algorithm Work?

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:

1. 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− ∑jPj2

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

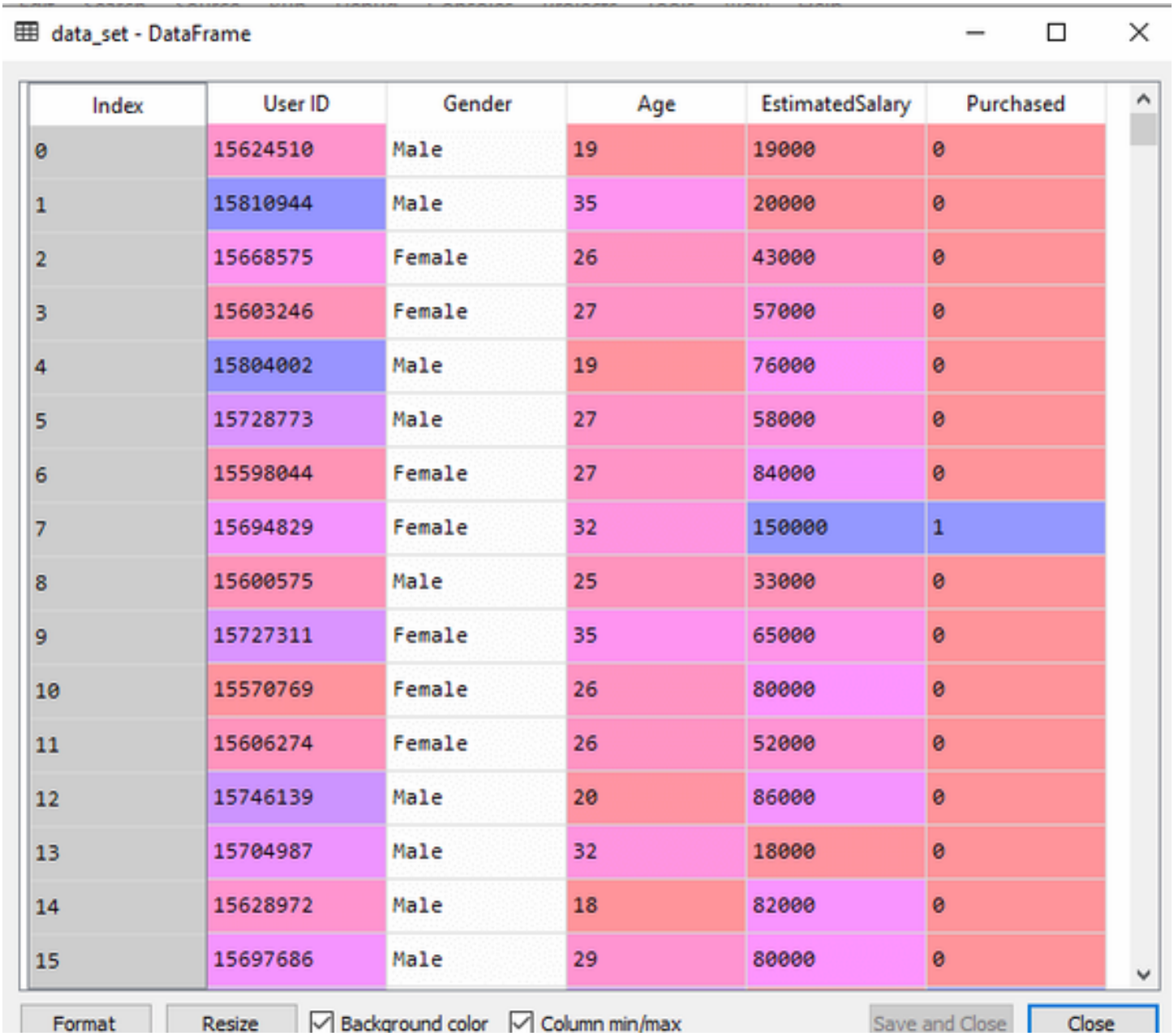
- Fitting a Decision-Tree algorithm to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, [2,3]].values
11. y= data_set.iloc[:, 4].values
12.
13. # Splitting the dataset into training and test set.
14. from sklearn.model_selection import train_test_split
15. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
16.
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
21. 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:

```
1. #Fitting Decision Tree classifier to the training set
2. From sklearn.tree import DecisionTreeClassifier
3. classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
4. 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')
```

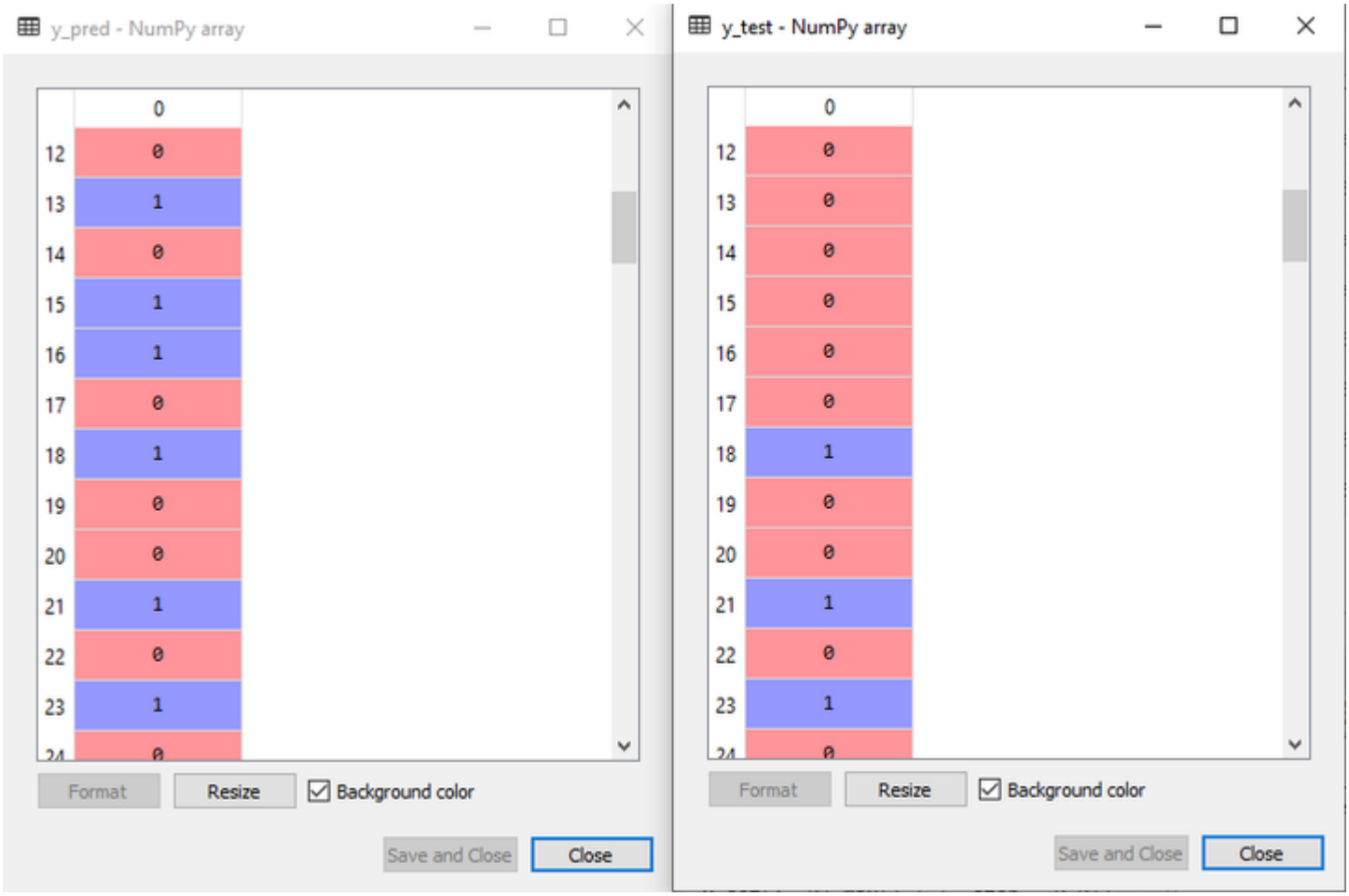
3. Predicting the test result

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

```
1. #Predicting the test set result
2. 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.

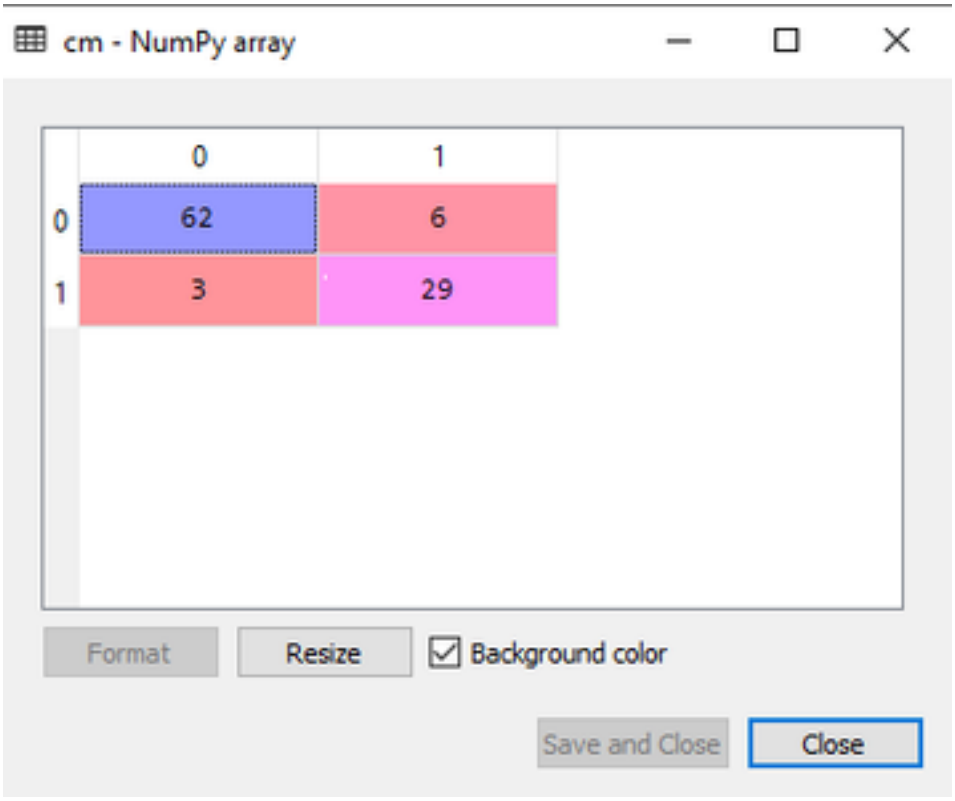


4. Test accuracy of the result (Creation of Confusion matrix)

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 and62+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:

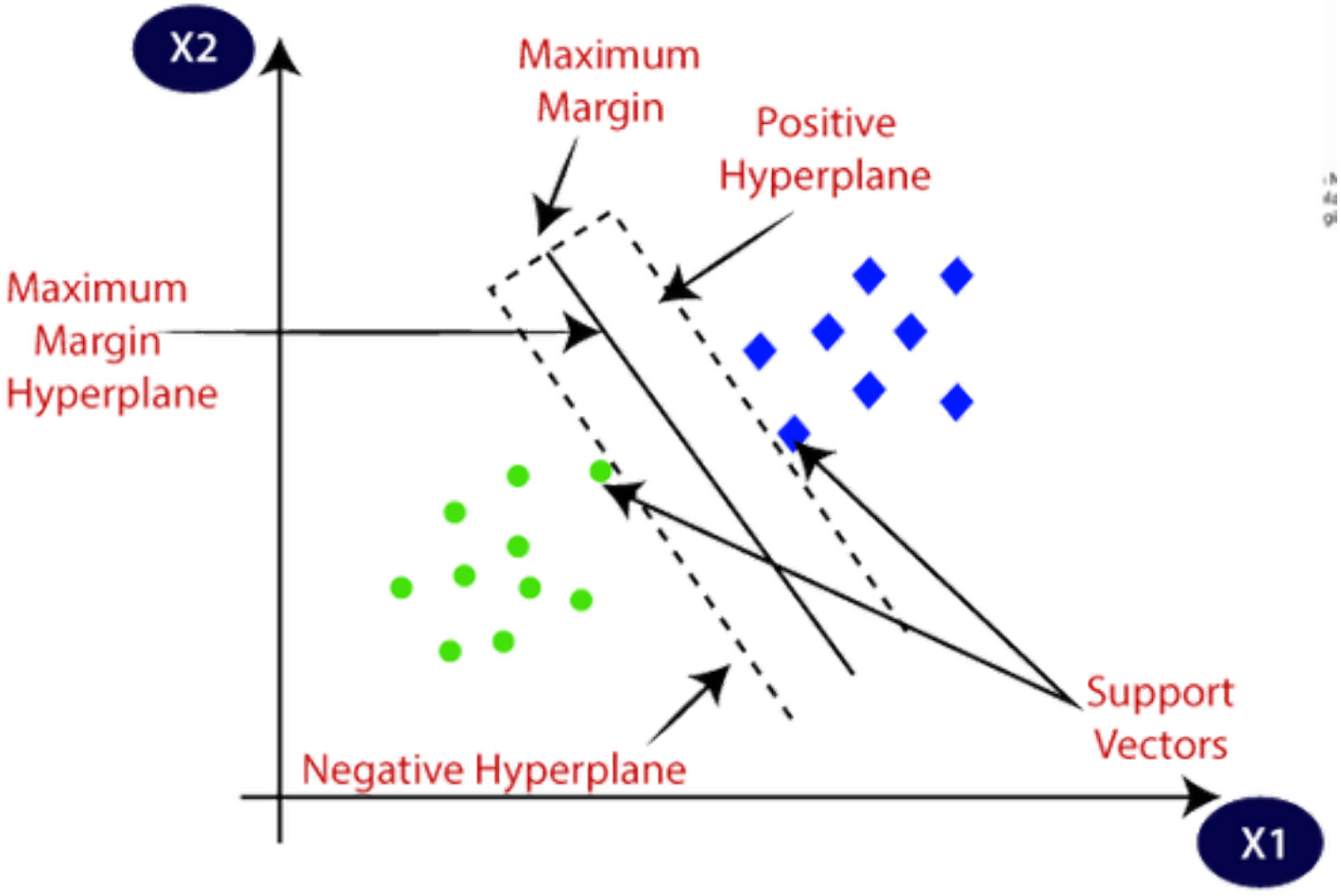
As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

Support Vector Machine Algorithm

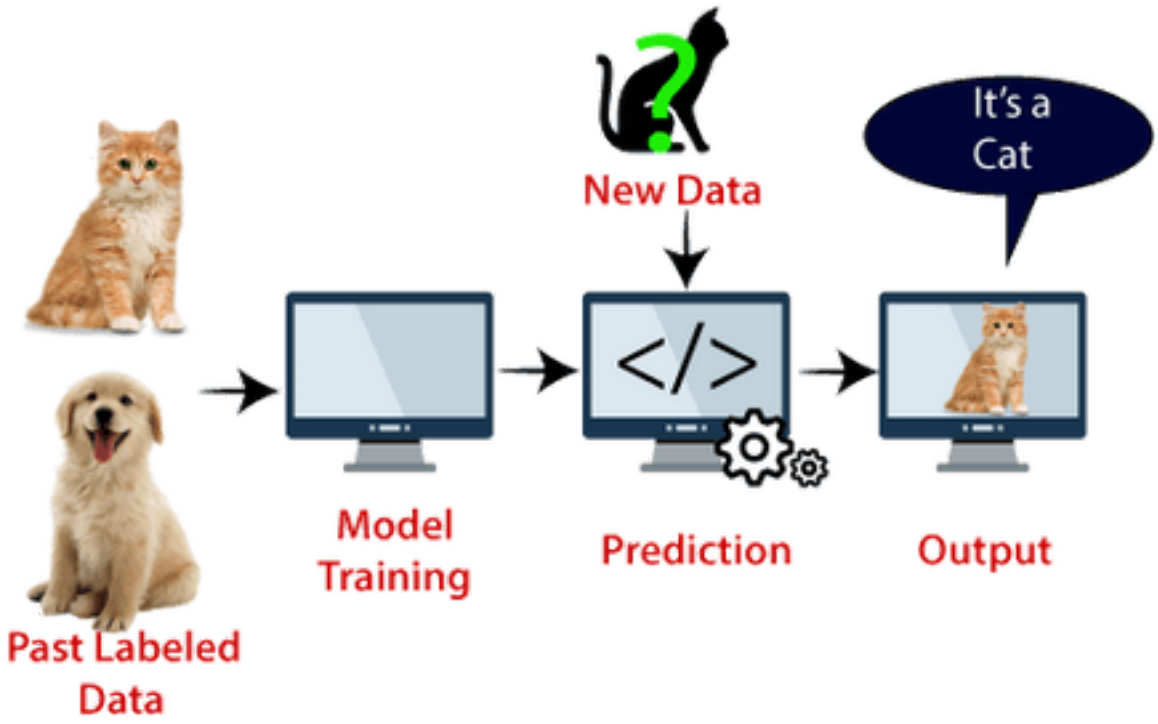
Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

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:



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:



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

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.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

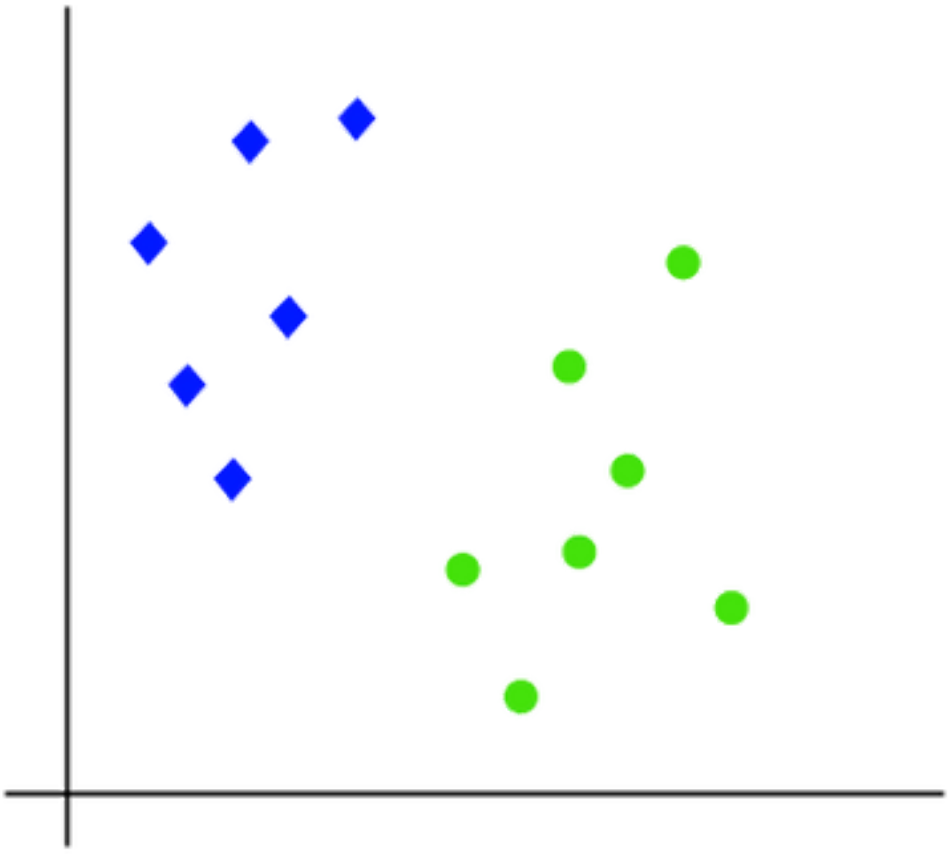
Support Vectors:

The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

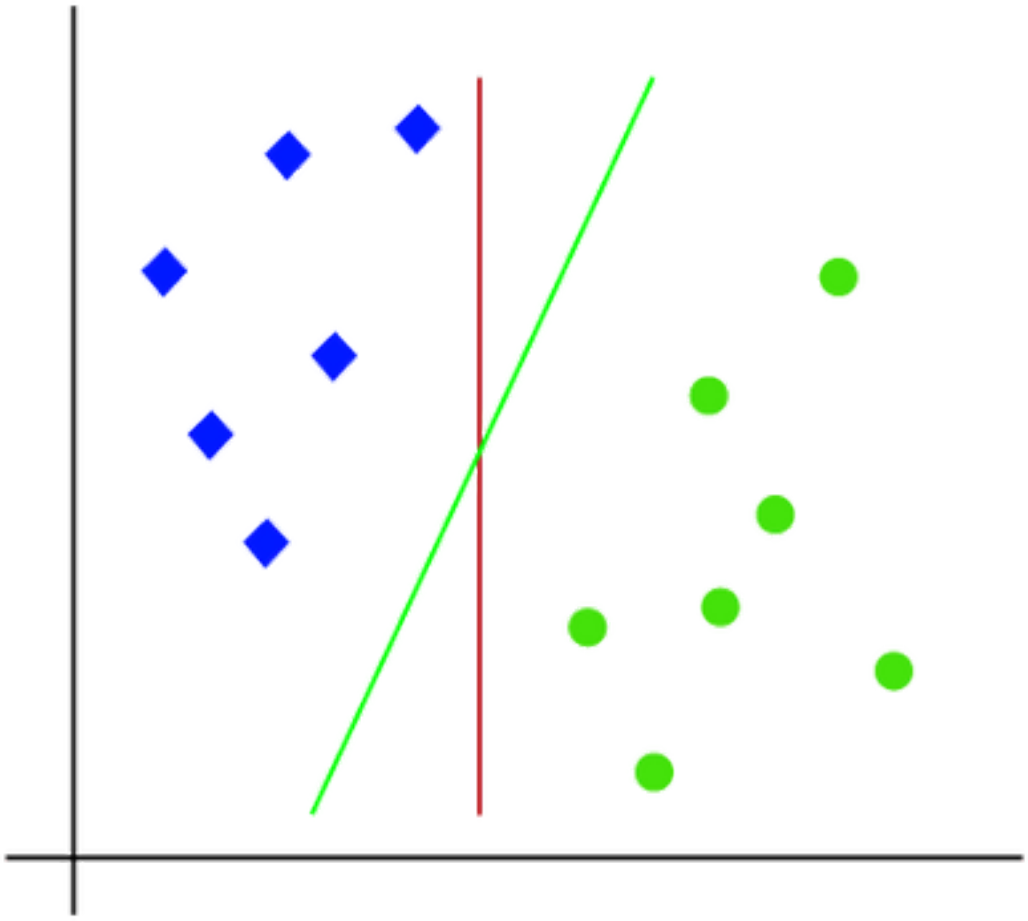
How does SVM works?

Linear SVM:

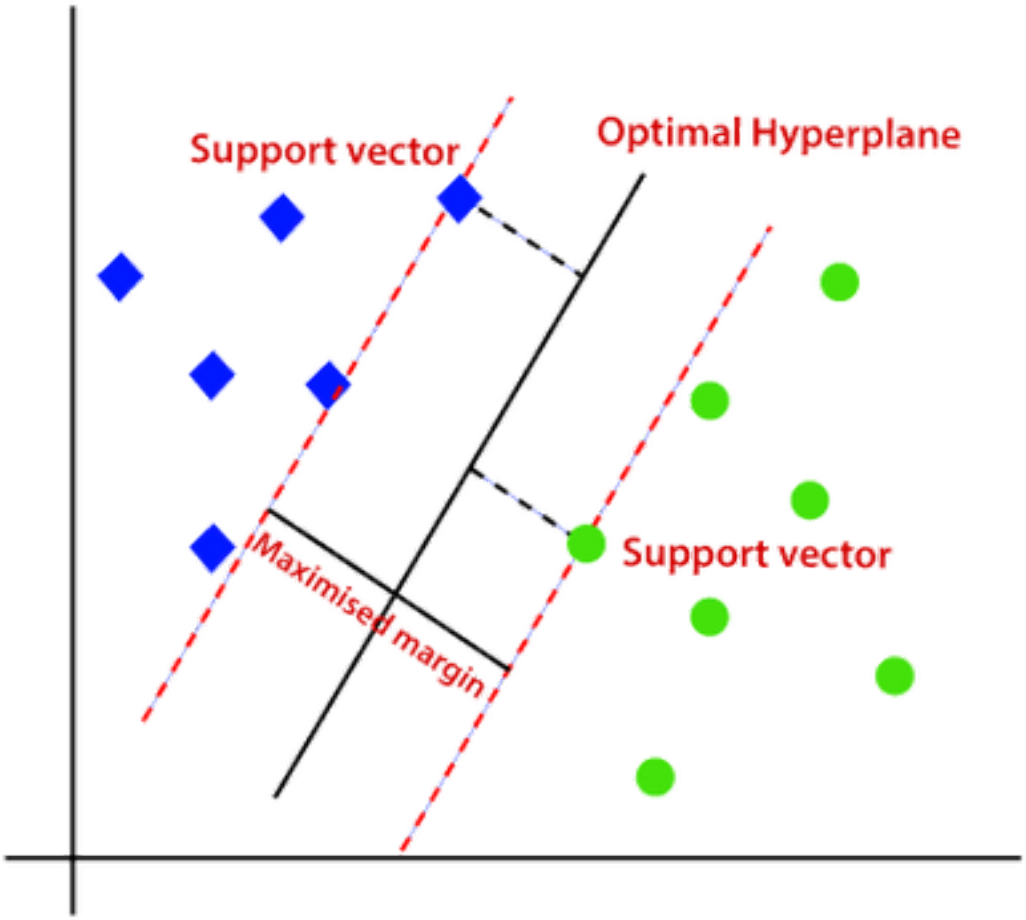
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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:

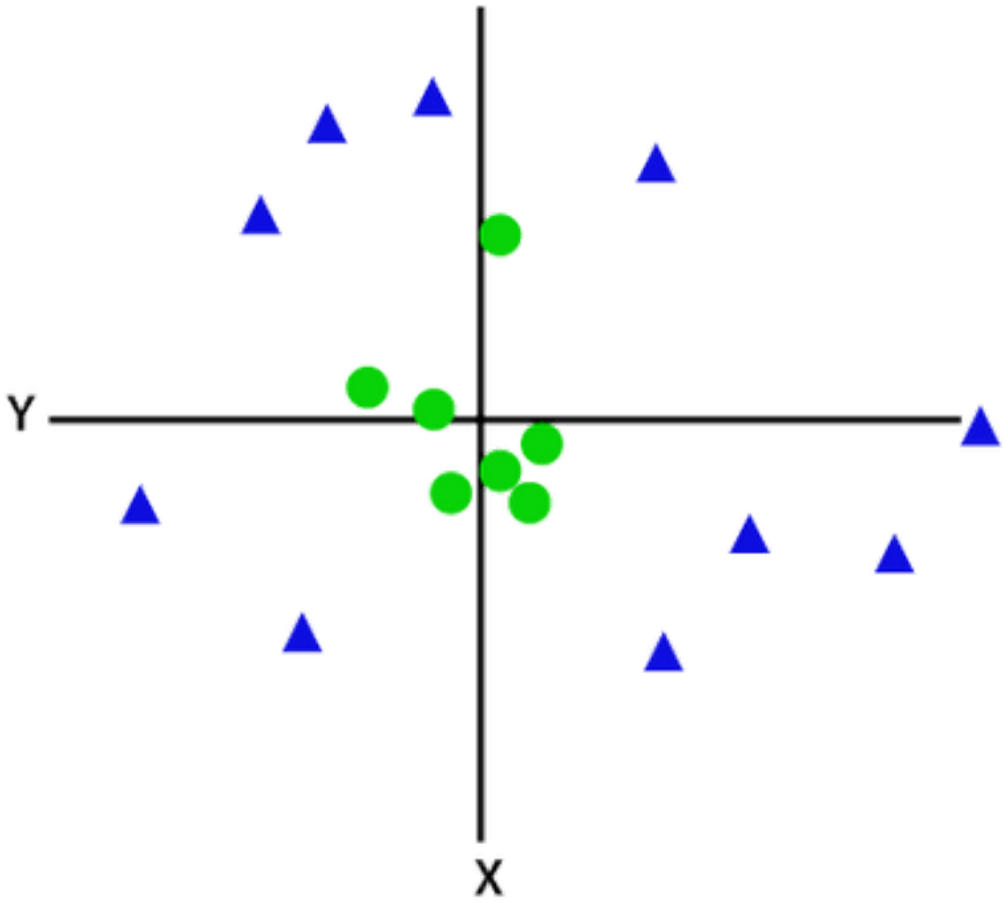


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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

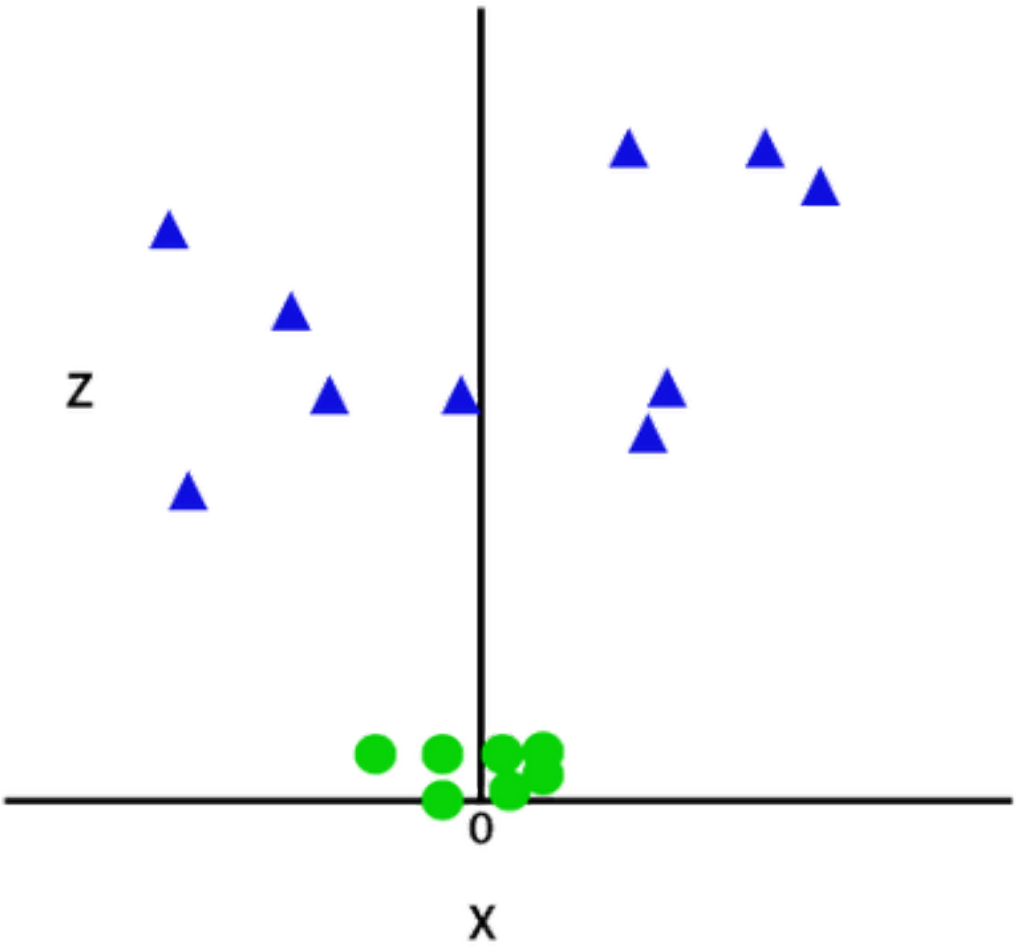
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



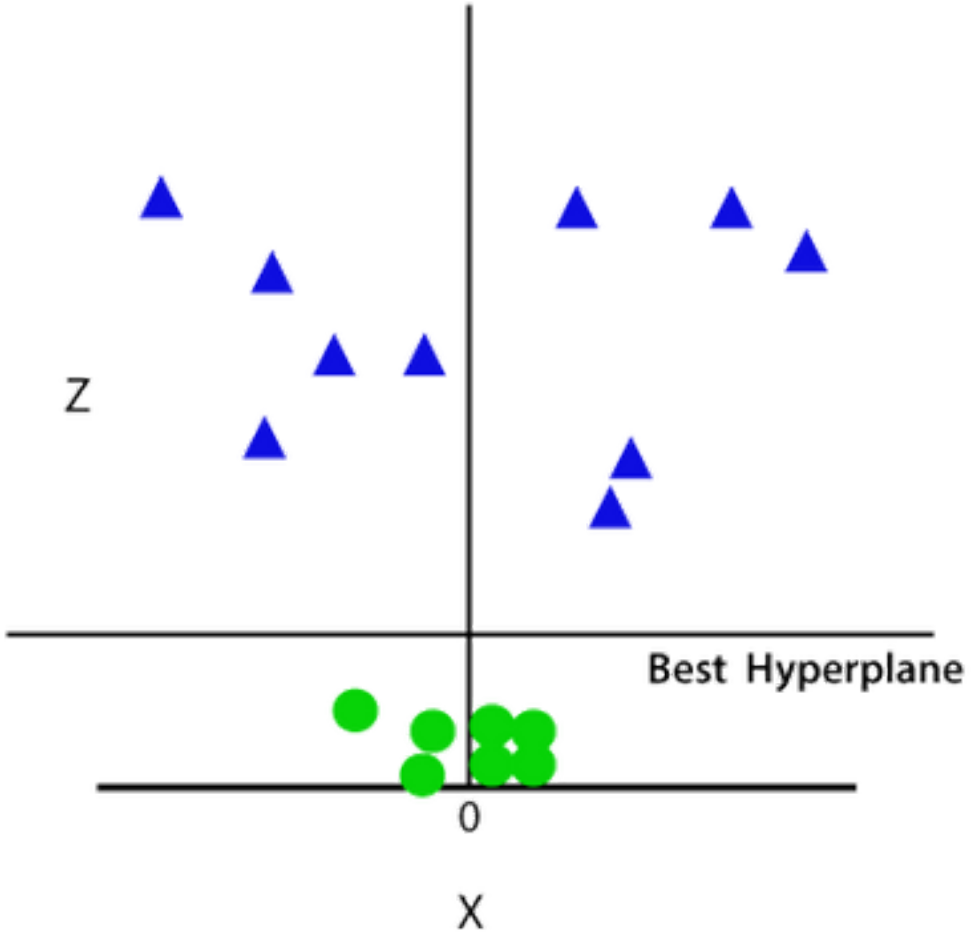
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

z=x2 +y2

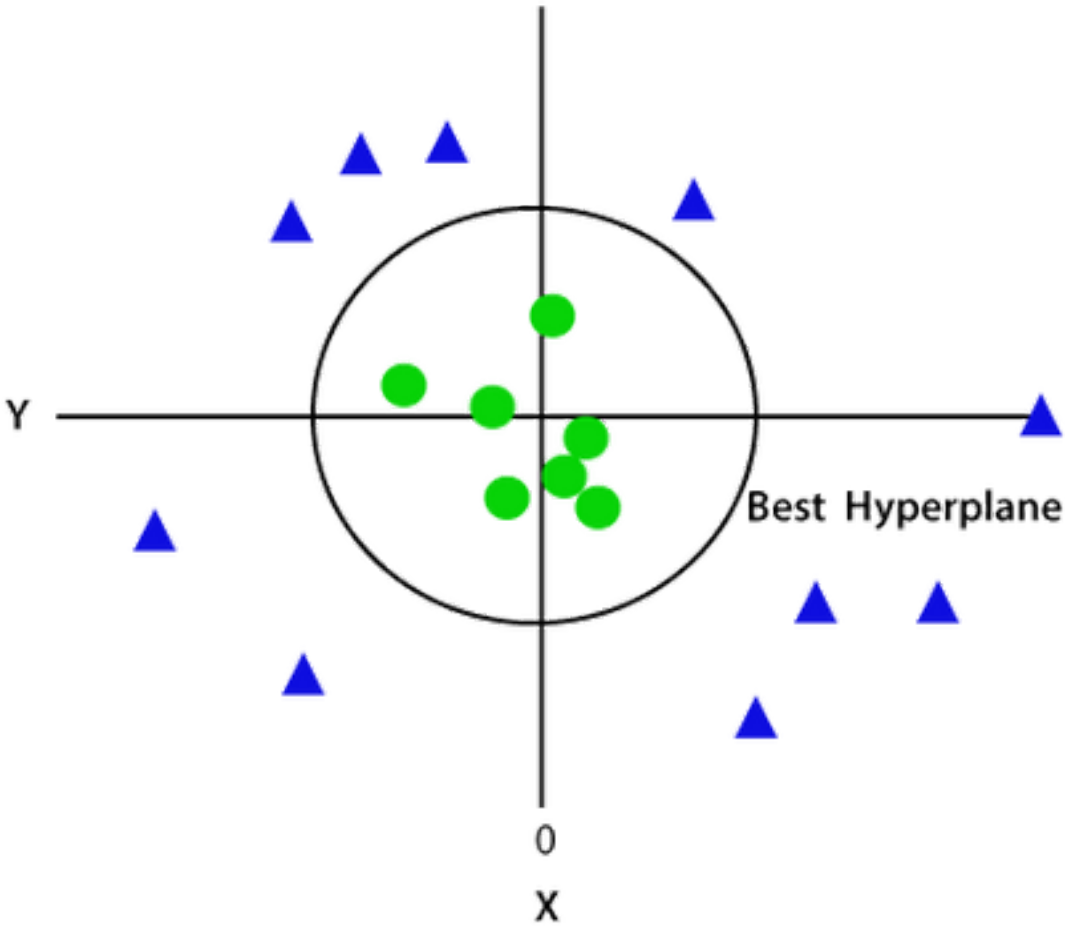
By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way. Consider the below image:



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:



Hence we get a circumference of radius 1 in case of non-linear data.

Python Implementation of Support Vector Machine

Now we will implement the SVM algorithm using Python. Here we will use the same dataset **user_data**, which we have used in Logistic regression and KNN classification.

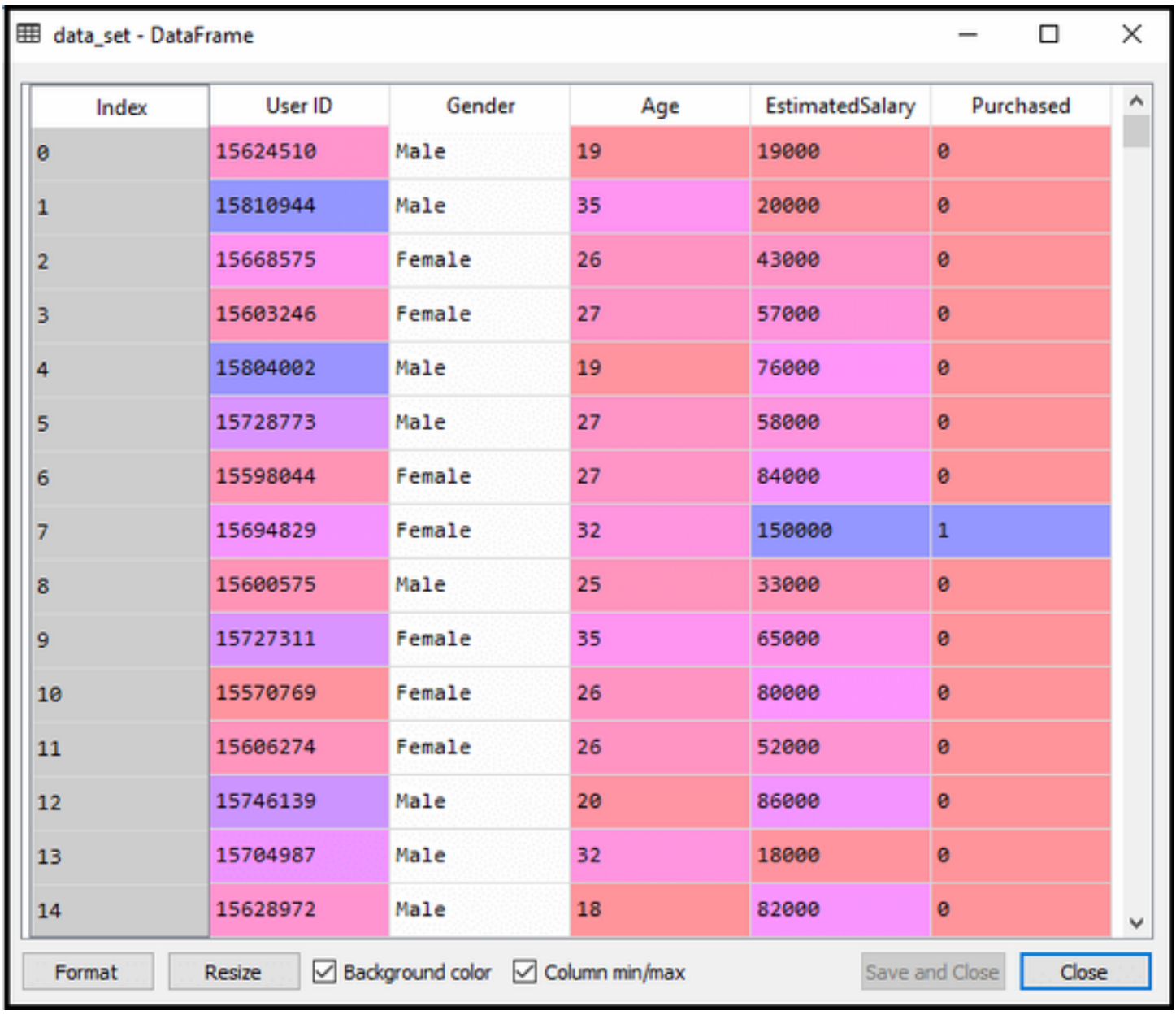
- **Data Pre-processing step**

Till the Data pre-processing step, the code will remain the same. Below is the code:

```
1. #Data Pre-processing Step
2. # importing libraries
3. import numpy as nm
4. import matplotlib.pyplot as mtp
5. import pandas as pd
6.
7. #importing datasets
8. data_set= pd.read_csv('user_data.csv')
9.
10. #Extracting Independent and dependent Variable
11. x= data_set.iloc[:, [2,3]].values
12. y= data_set.iloc[:, 4].values
13.
```

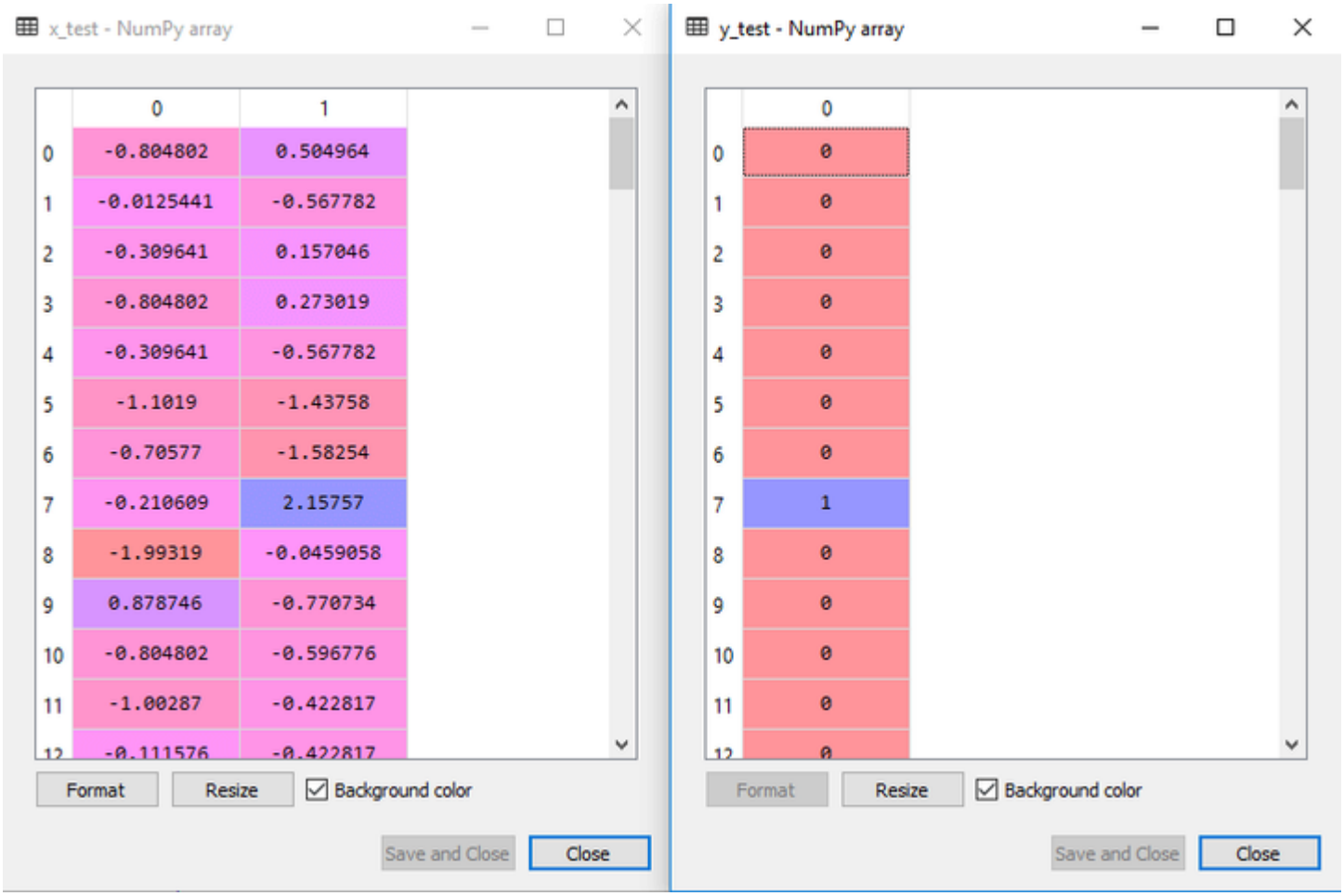
```
14. # Splitting the dataset into training and test set.
15. from sklearn.model_selection import train_test_split
16. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
21. x_test= st_x.transform(x_test)
```

After executing the above code, we will pre-process the data. The code will give the dataset 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

The scaled output for the test set will be:



Fitting the SVM classifier to the training set:

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we will import **SVC** class from **Sklearn.svm** library. Below is the code for it:

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

In the above code, we have used **kernel='linear'**, as here we are creating SVM for linearly separable data. However, we can change it for non-linear data. And then we fitted the classifier to the training dataset(x_train, y_train)

Output:

```
Out[8]:

SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='linear', max_iter=-1, probability=False, random_state=0,
  shrinking=True, tol=0.001, verbose=False)
```

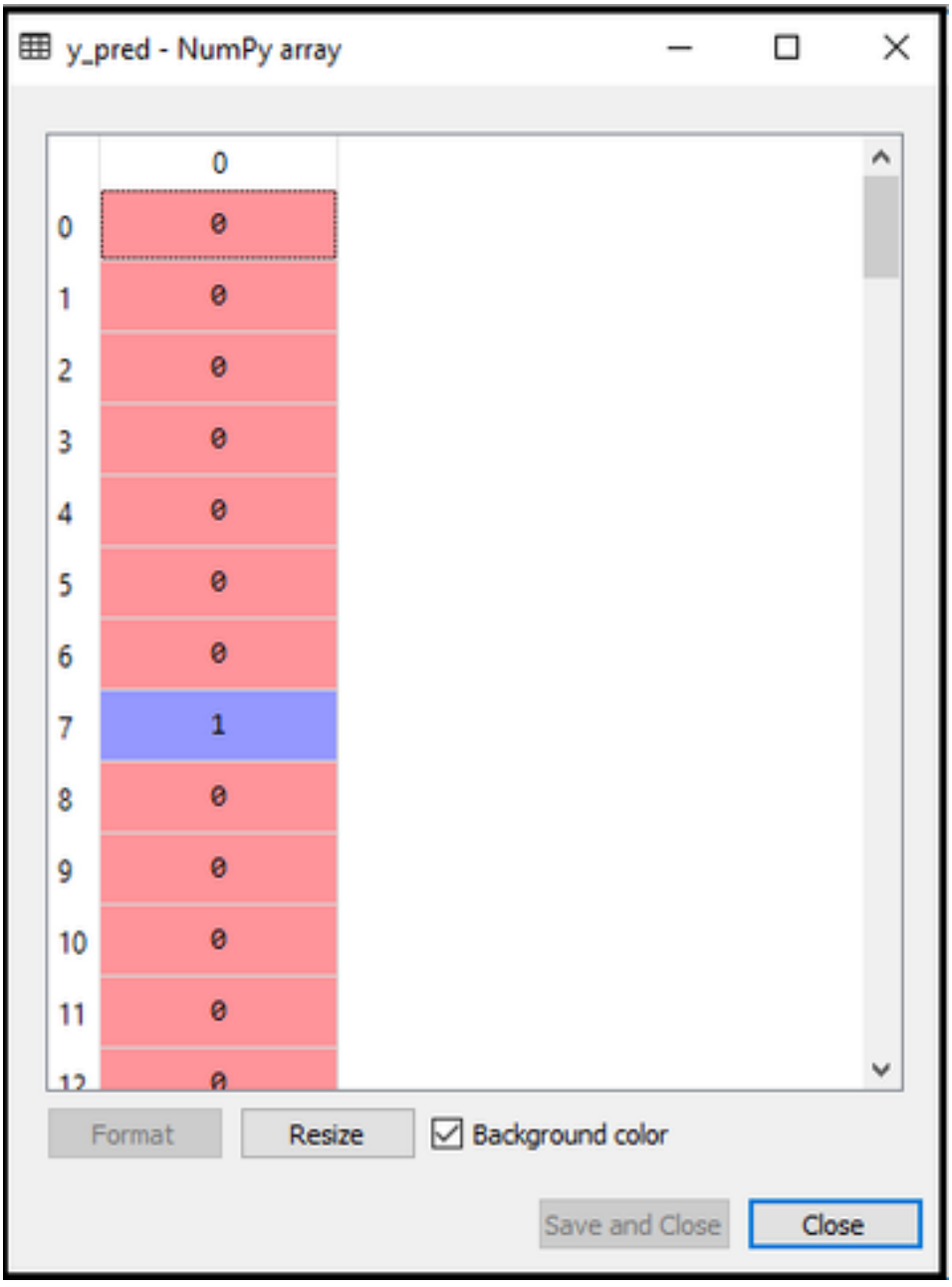
The model performance can be altered by changing the value of **C(Regularization factor), gamma, and kernel**.

- **Predicting the test set result:**Now, we will predict the output for test set. For this, we will create a new vector
  y_pred. Below is the code for it:

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

After getting the y_pred vector, we can compare the result of **y_pred** and **y_test** to check the difference between the actual value and predicted value.

Output: Below is the output for the prediction of the test set:



- **Creating the confusion matrix:**Now we will see the performance of the SVM classifier that how many incorrect predictions are there as compared to the Logistic regression classifier. To create the confusion matrix, we need to import the

confusion_matrix function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y_true**(the actual values) and **y_pred** (the targeted value return by the classifier). 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 output image, there are 66+24= 90 correct predictions and 8+2= 10 correct predictions. Therefore we can say that our SVM model improved as compared to the Logistic regression model.

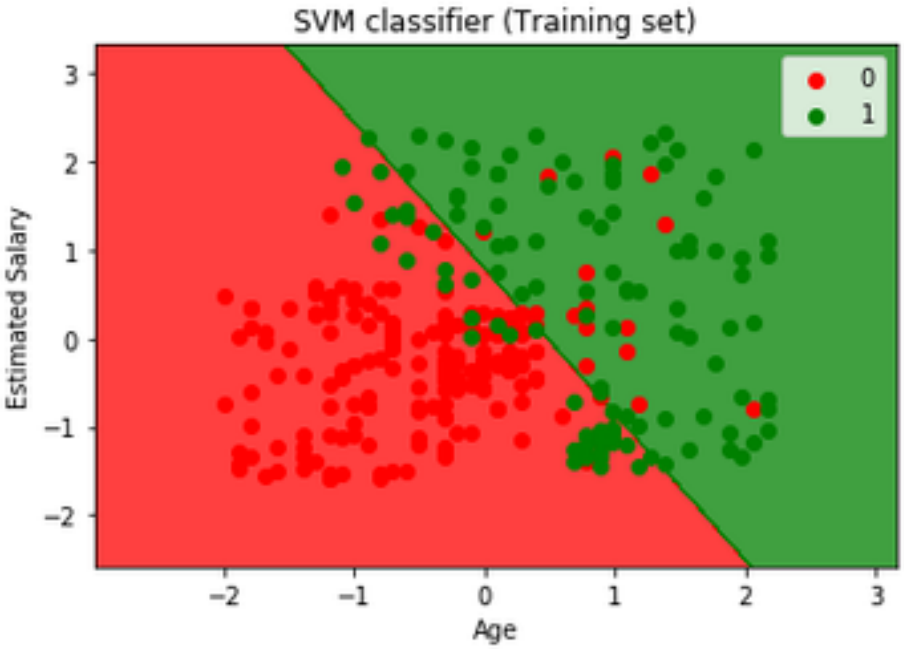
- **Visualizing the training set result:**Now we will visualize the training set result, below is the code for it:

```
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.shape),
6. alpha = 0.75, cmap = ListedColormap(('red', 'green')))
7. mtp.xlim(x1.min(), x1.max())
8. 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(('red', 'green'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:

By executing the above code, we will get the output as:



As we can see, the above output is appearing similar to the Logistic regression output. In the output, we got the straight line as hyperplane because we have **used a linear kernel in the classifier**. And we have also discussed above that for the 2d space, the hyperplane in SVM is a straight line.

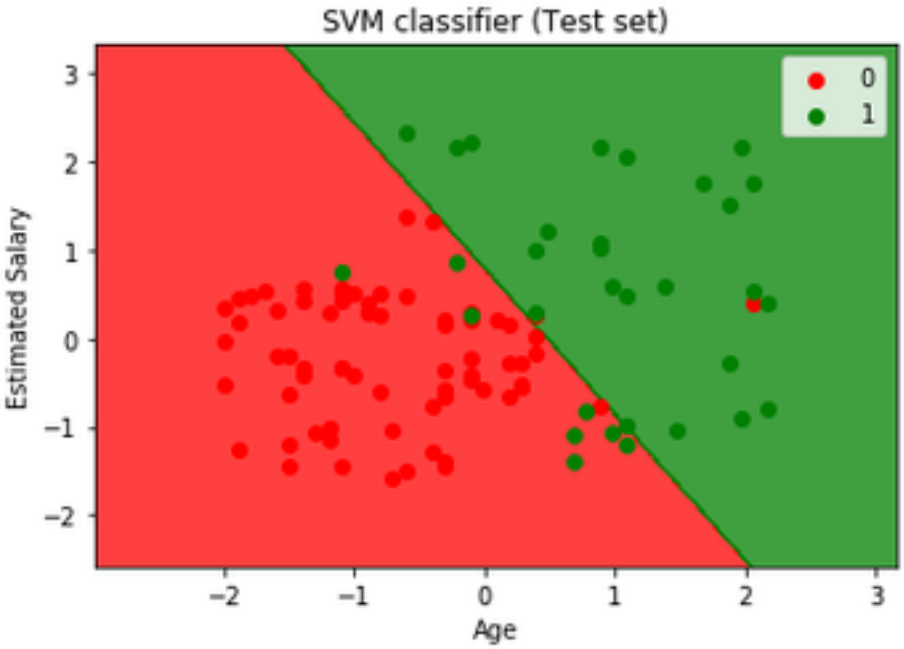
- **Visualizing the test set result:**

```
#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(('red','green' )))
mtp.xlim(x1.min(), x1.max())
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(('red', 'green'))(i), label = j)
13. mtp.title('SVM classifier (Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()
```

Output:

By executing the above code, we will get the output as:



As we can see in the above output image, the SVM classifier has divided the users into two regions (Purchased or Not purchased). Users who purchased the SUV are in the red region with the red scatter points. And users who did not purchase the SUV are in the green region with green scatter points. The hyperplane has divided the two classes into Purchased and not purchased variable.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

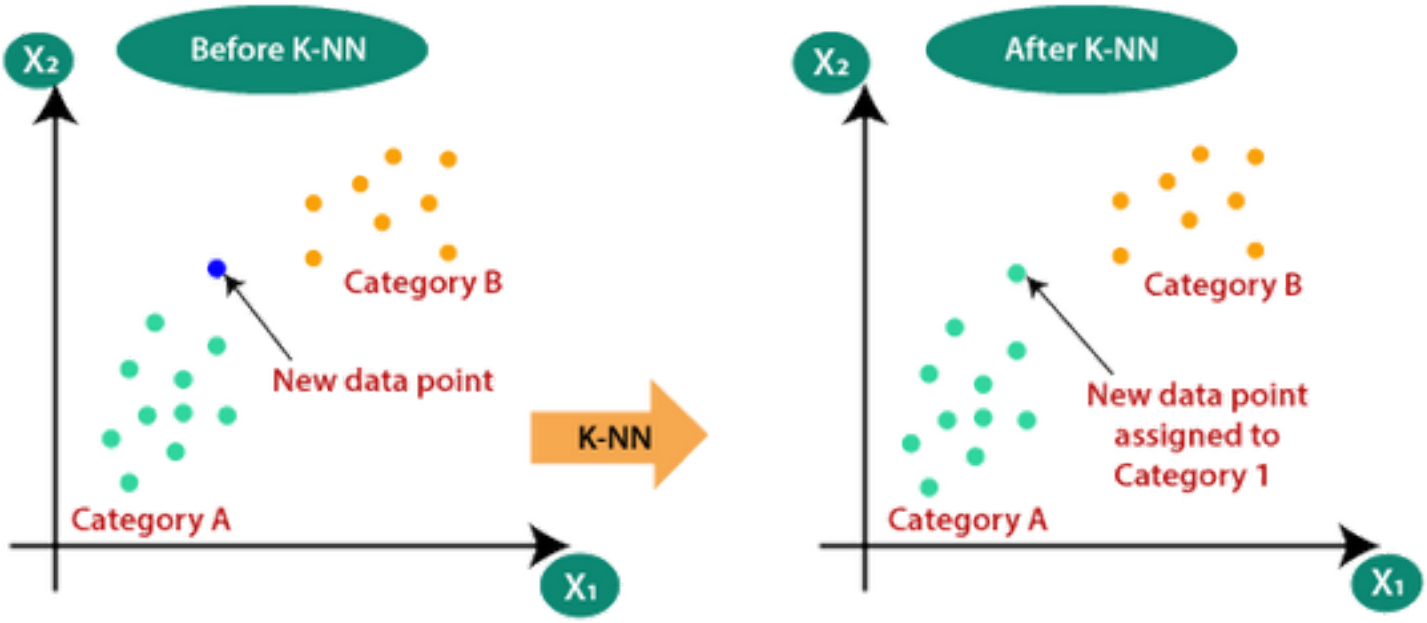
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm 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.
- **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.

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:

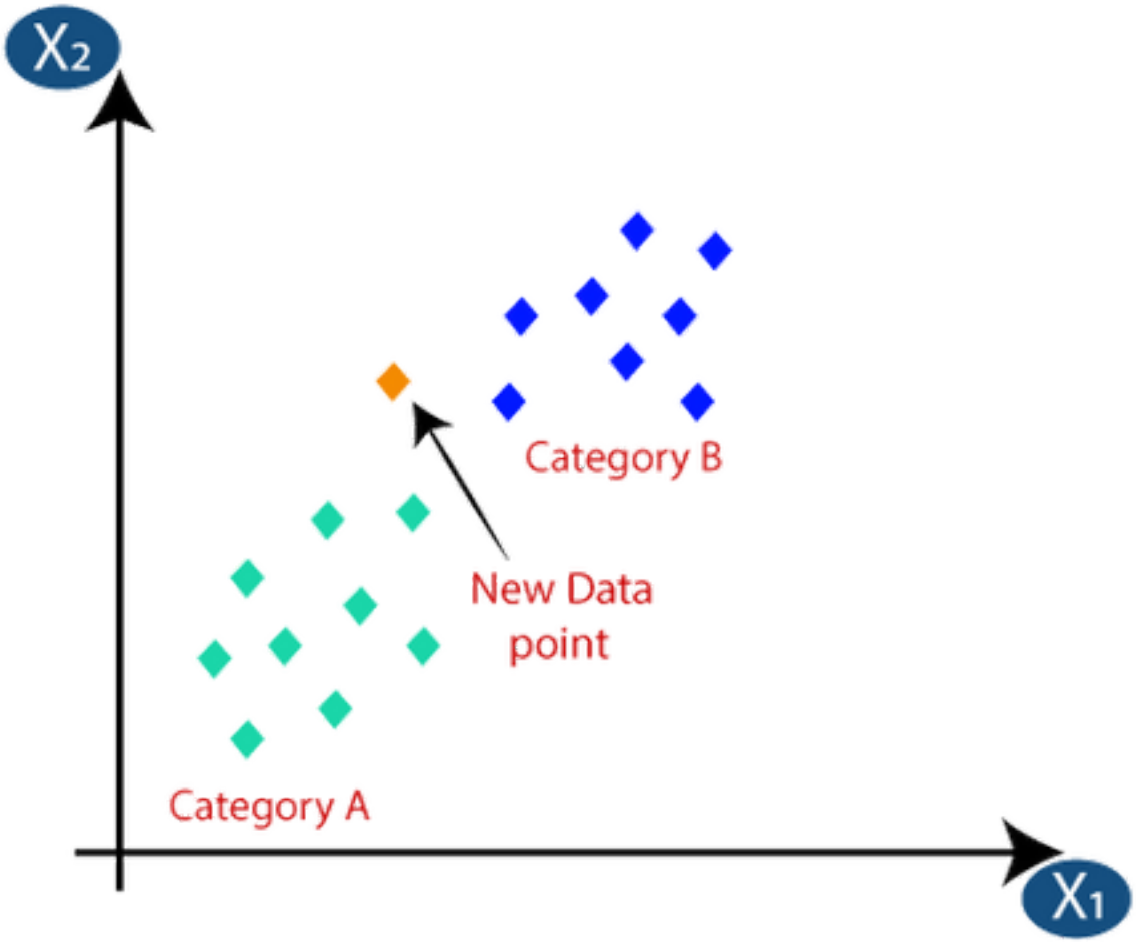


How does K-NN work?

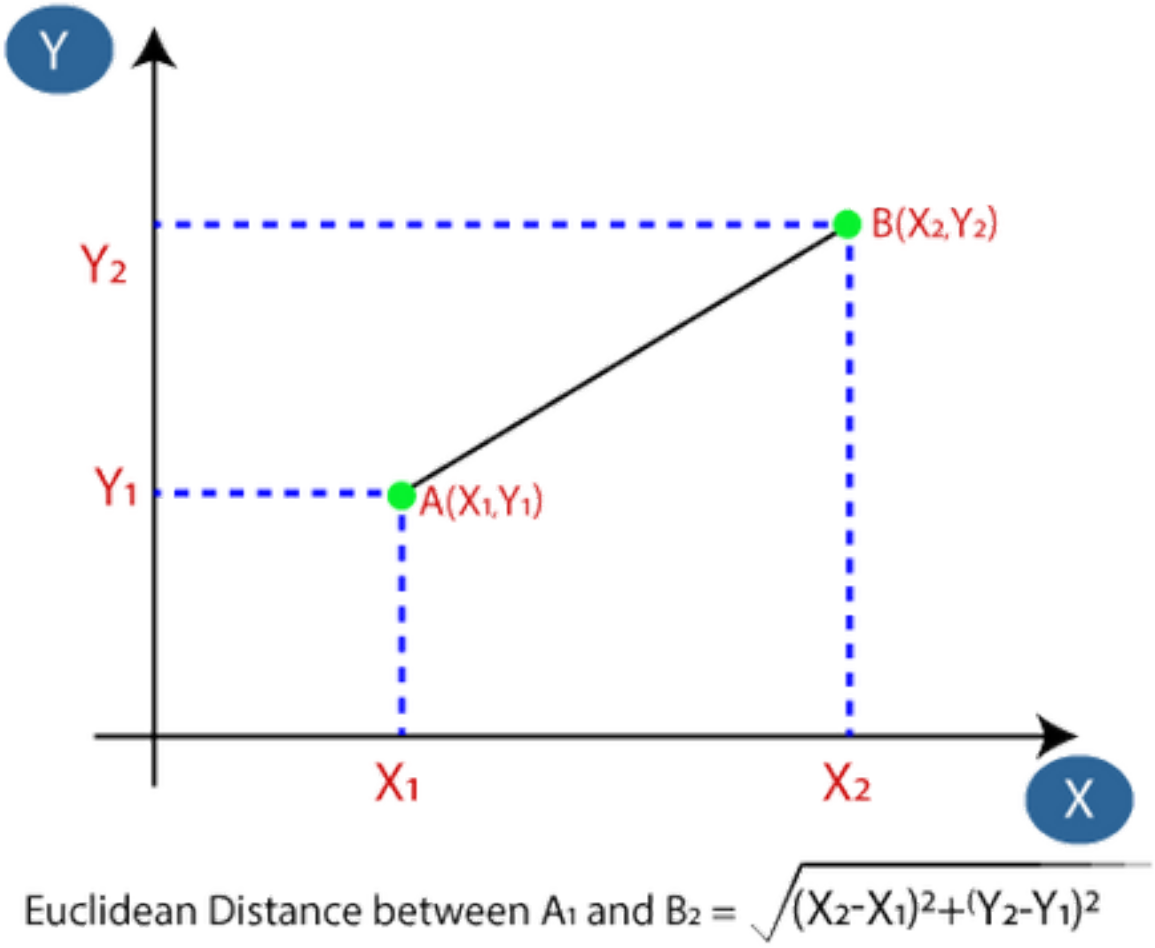
The K-NN working can be explained on the basis of the below algorithm:

- **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.

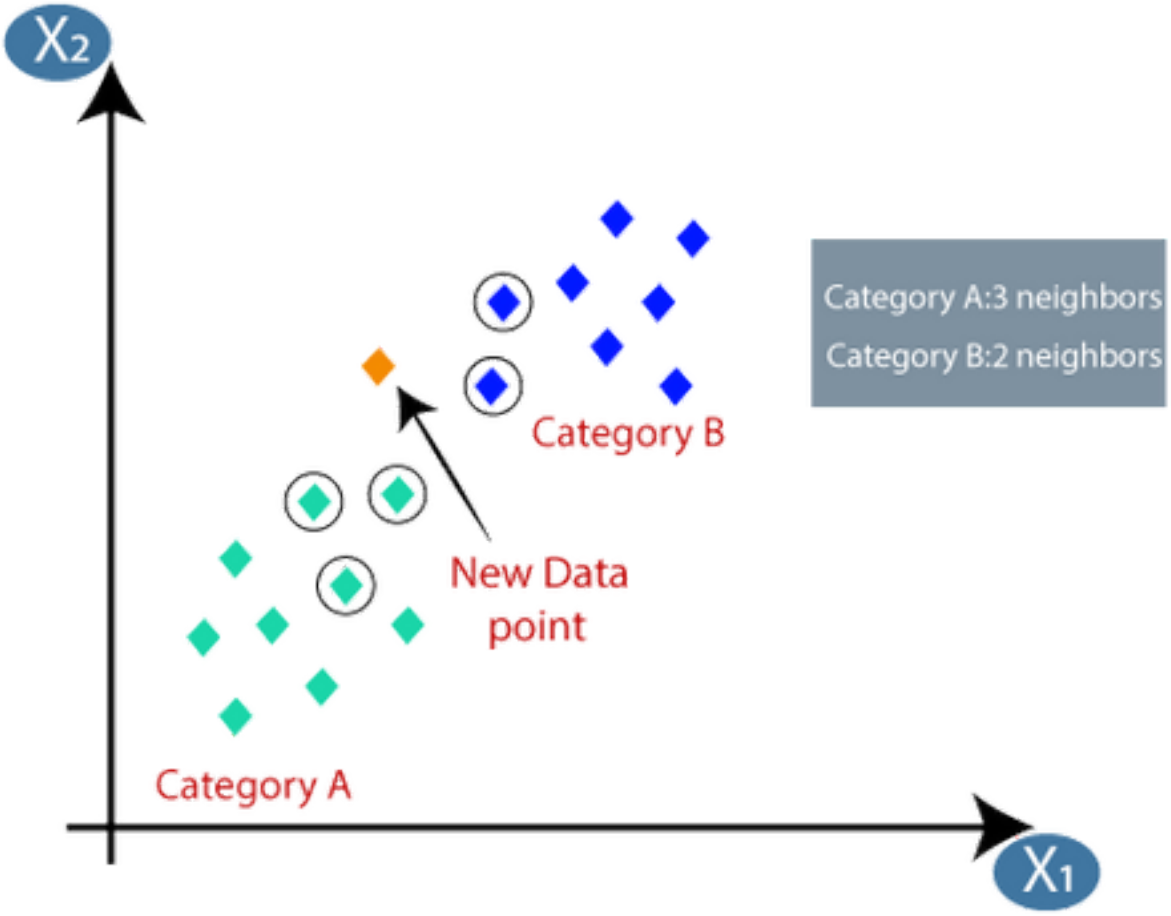
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- 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:



- 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.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

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.

Python implementation of the KNN algorithm

To do the Python implementation of the K-NN algorithm, we will use the same problem and dataset which we have used in Logistic Regression. But here we will improve the performance of the model. Below is the problem description:

Problem for K-NN Algorithm: There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network. The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

Steps to implement the K-NN algorithm:

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result

- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

Data Pre-Processing Step:

The Data Pre-processing step will remain exactly the same as Logistic Regression. Below is the code for it:

```
# 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)
```

By executing the above code, our dataset is imported to our program and well pre-processed. After feature scaling our test dataset will look like:

From the above output image, we can see that our data is successfully scaled.

- **Fitting K-NN classifier to the Training data:**Now we will fit the K-NN classifier to the training data. To do this we will import the **KNeighborsClassifier** class of **Sklearn Neighbors** library. After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be
  - **n_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 5.
  - **metric='minkowski':** This is the default parameter and it decides the distance between the points.
  - **p=2:** It is equivalent to the standard Euclidean metric.

And then we will fit the classifier to the training data. Below is the code for it:

```
#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)
```

Output: By executing the above code, we will get the output as:

```
Out[10]:

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
```

metric_params=None, n_jobs=None, n_neighbors=5, p=2,
weights='uniform')

- **Predicting the Test Result:** To predict the test set result, we will create a **y_pred** vector as we did in Logistic Regression. Below is the code for it:

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

Output:

The output for the above code will be:

- **Creating the Confusion Matrix:**Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

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

In above code, we have imported the confusion_matrix function and called it using the variable cm.

Output: By executing the above code, we will get the matrix as below:

In the above image, we can see there are 64+29= 93 correct predictions and 3+4= 7 incorrect predictions, whereas, in Logistic Regression, there were 11 incorrect predictions. So we can say that the performance of the model is improved by using the K-NN algorithm.

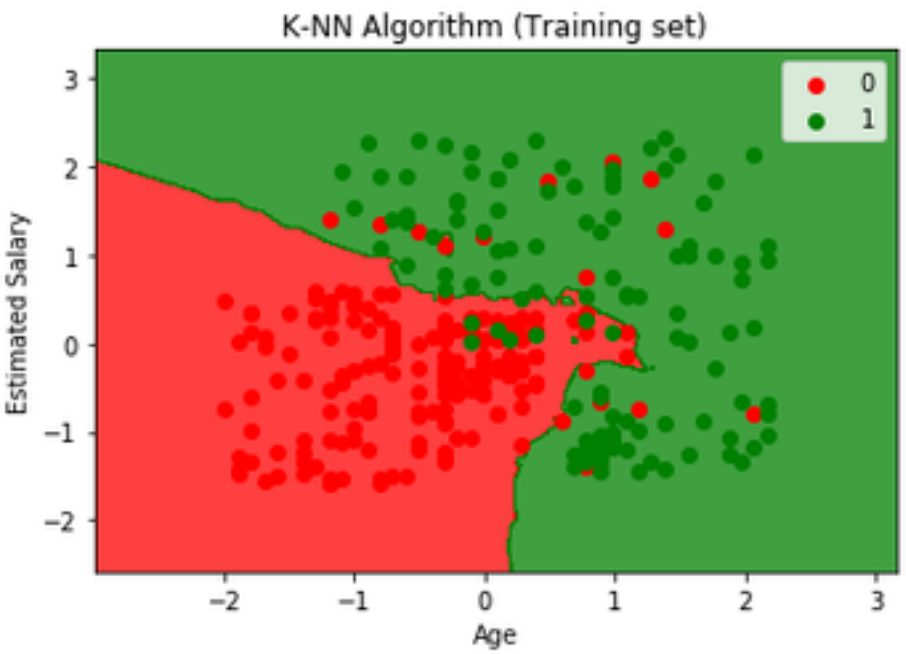
- **Visualizing the Training set result:**Now, we will visualize the training set result for K-NN model. The code will remain same as we did in Logistic Regression, except the name of the graph. Below is the code for it:

```
#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(('red','green' )))
mtp.xlim(x1.min(), x1.max())
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(('red', 'green'))(i), label = j)
13. mtp.title('K-NN Algorithm (Training set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()
```

Output:

By executing the above code, we will get the below graph:



The output graph is different from the graph which we have occurred in Logistic Regression. It can be understood in the below points:

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.
- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
- Hence our model is well trained.

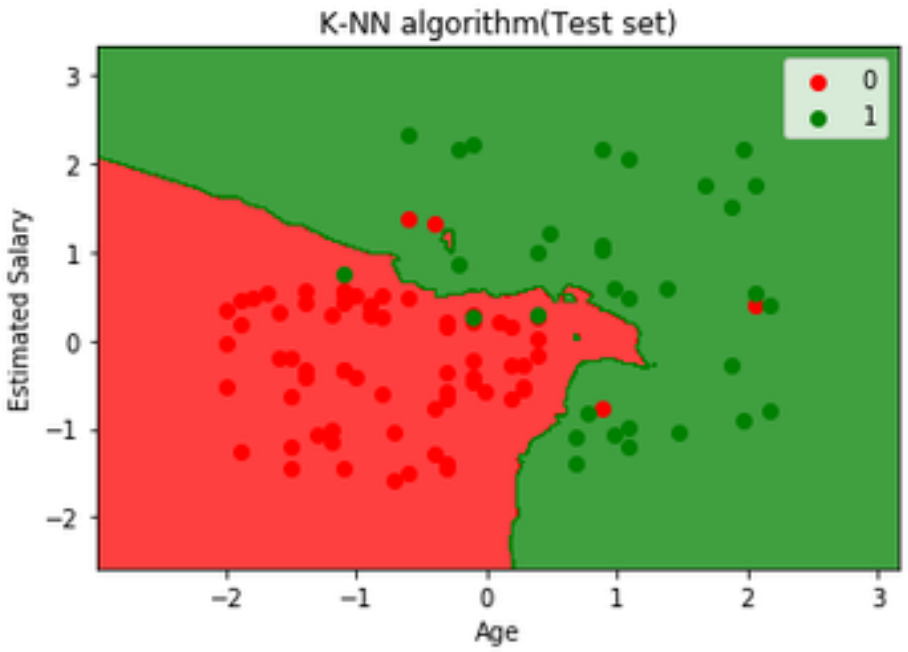
- **Visualizing the Test set result:**After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes:

such as

x_train and y_train will be replaced by **x_test and y_test**.Below is the code for it:

```
#Visualizing 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(('red','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(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

Output:



The above graph is showing the output for the test data set. As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.

However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

Example : https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html

Click for solution

Introduction to Dimensionality Reduction Technique

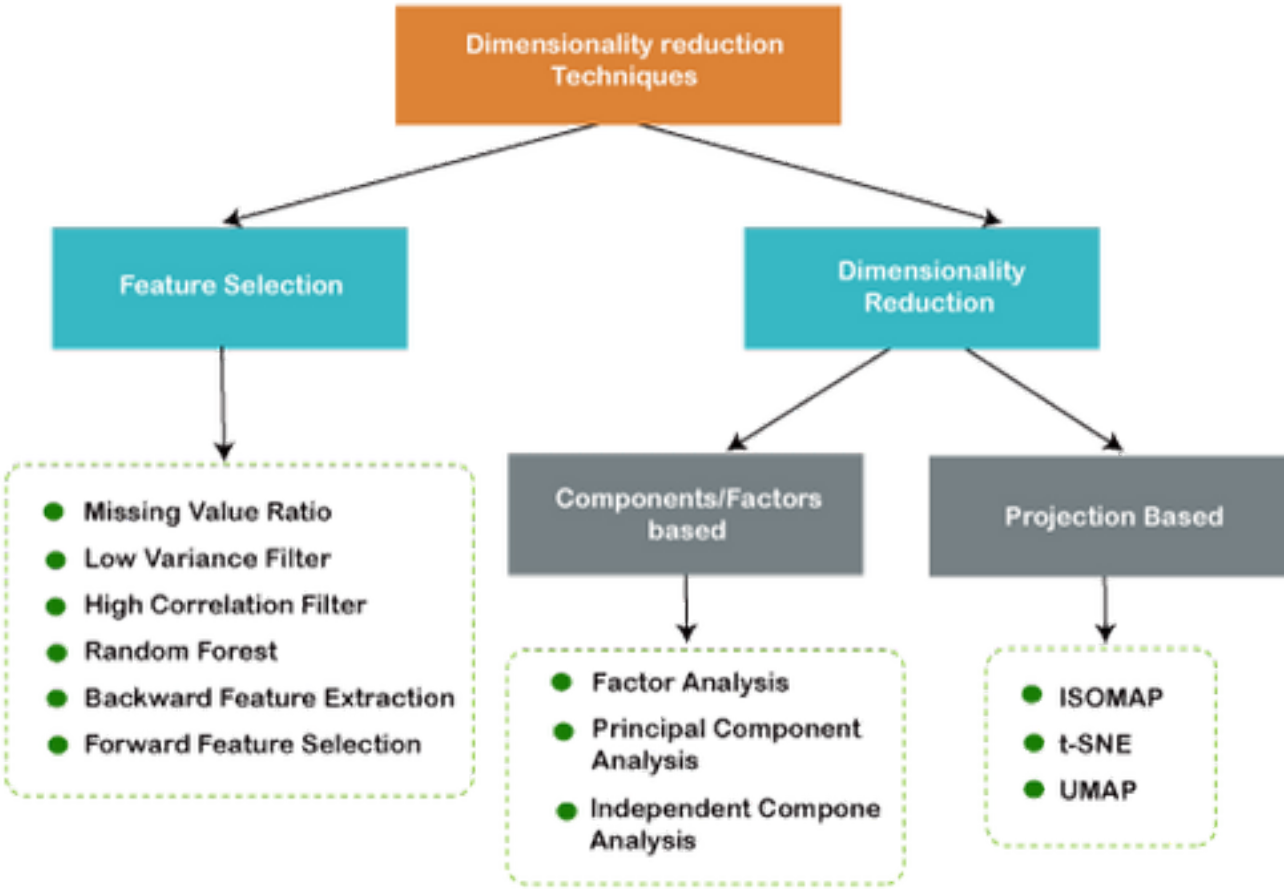
What is Dimensionality Reduction?

The number of input features, variables, or columns present in a given dataset is known as dimensionality, and the process to reduce these features is called dimensionality reduction.

A dataset contains a huge number of input features in various cases, which makes the predictive modeling task more complicated. Because it is very difficult to visualize or make predictions for the training dataset with a high number of features, for such cases, dimensionality reduction techniques are required to use.

Dimensionality reduction technique can be defined as, **"It is a way of converting the higher dimensions dataset into lesser dimensions dataset ensuring that it provides similar information."** These techniques are widely used in machine learning for obtaining a better fit predictive model while solving the classification and regression problems.

It is commonly used in the fields that deal with high-dimensional data, such as speech recognition, signal processing, bioinformatics, etc. It can also be used for data visualization, noise reduction, cluster analysis, etc.



The Curse of Dimensionality

Handling the high-dimensional data is very difficult in practice, commonly known as the curse of dimensionality. If the dimensionality of the input dataset increases, any machine learning algorithm and model becomes more complex. As the number of features increases, the number of samples also gets increased proportionally, and the chance of overfitting also increases. If the machine learning model is trained on high-dimensional data, it becomes overfitted and results in poor performance.

Hence, it is often required to reduce the number of features, which can be done with dimensionality reduction.

Benefits of applying Dimensionality Reduction

Some benefits of applying dimensionality reduction technique to the given dataset are given below:

- By reducing the dimensions of the features, the space required to store the dataset also gets reduced.
- Less Computation training time is required for reduced dimensions of features.
- Reduced dimensions of features of the dataset help in visualizing the data quickly.
- It removes the redundant features (if present) by taking care of multicollinearity.

Disadvantages of dimensionality Reduction

There are also some disadvantages of applying the dimensionality reduction, which are given below:

- Some data may be lost due to dimensionality reduction.

- In the PCA dimensionality reduction technique, sometimes the principal components required to consider are unknown.

Approaches of Dimension Reduction

There are two ways to apply the dimension reduction technique, which are given below:

Feature Selection

Feature selection is the process of selecting the subset of the relevant features and leaving out the irrelevant features present in a dataset to build a model of high accuracy. In other words, it is a way of selecting the optimal features from the input dataset.

Three methods are used for the feature selection:

1. Filters Methods

In this method, the dataset is filtered, and a subset that contains only the relevant features is taken. Some common techniques of filters method are:

- Correlation
- Chi-Square Test
- ANOVA
- Information Gain, etc.

2. Wrappers Methods

The wrapper method has the same goal as the filter method, but it takes a machine learning model for its evaluation. In this method, some features are fed to the ML model, and evaluate the performance. The performance decides whether to add those features or remove to increase the accuracy of the model. This method is more accurate than the filtering method but complex to work. Some common techniques of wrapper methods are:

- Forward Selection
- Backward Selection
- Bi-directional Elimination

3. Embedded Methods: Embedded methods check the different training iterations of the machine learning model and evaluate the importance of each feature. Some common techniques of Embedded methods are:

- LASSO
- Elastic Net
- Ridge Regression, etc.

Feature Extraction:

Feature extraction is the process of transforming the space containing many dimensions into space with fewer dimensions. This approach is useful when we want to keep the whole information but use fewer resources while processing the information.

Some common feature extraction techniques are:

1. Principal Component Analysis
2. Linear Discriminant Analysis

3. Kernel PCA
4. Quadratic Discriminant Analysis

Common techniques of Dimensionality Reduction

1. Principal Component Analysis
2. Backward Elimination
3. Forward Selection
4. Score comparison
5. Missing Value Ratio
6. Low Variance Filter
7. High Correlation Filter
8. Random Forest
9. Factor Analysis
10. Auto-Encoder

Principal Component Analysis (PCA)

Principal Component Analysis is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components. It is one of the popular tools that is used for exploratory data analysis and predictive modeling.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are **image processing, movie recommendation system, optimizing the power allocation in various communication channels.**

Backward Feature Elimination

The backward feature elimination technique is mainly used while developing Linear Regression or Logistic Regression model. Below steps are performed in this technique to reduce the dimensionality or in feature selection:

- In this technique, firstly, all the n variables of the given dataset are taken to train the model.
- The performance of the model is checked.
- Now we will remove one feature each time and train the model on n-1 features for n times, and will compute the performance of the model.
- We will check the variable that has made the smallest or no change in the performance of the model, and then we will drop that variable or features; after that, we will be left with n-1 features.
- Repeat the complete process until no feature can be dropped.

In this technique, by selecting the optimum performance of the model and maximum tolerable error rate, we can define the optimal number of features require for the machine learning algorithms.

Forward Feature Selection

Forward feature selection follows the inverse process of the backward elimination process. It means, in this technique, we don't eliminate the feature; instead, we will find the best features that can produce the highest increase in the performance of the model. Below steps are performed in this technique:

- We start with a single feature only, and progressively we will add each feature at a time.
- Here we will train the model on each feature separately.
- The feature with the best performance is selected.
- The process will be repeated until we get a significant increase in the performance of the model.

Missing Value Ratio

If a dataset has too many missing values, then we drop those variables as they do not carry much useful information. To perform this, we can set a threshold level, and if a variable has missing values more than that threshold, we will drop that variable. The higher the threshold value, the more efficient the reduction.

Low Variance Filter

As same as missing value ratio technique, data columns with some changes in the data have less information. Therefore, we need to calculate the variance of each variable, and all data columns with variance lower than a given threshold are dropped because low variance features will not affect the target variable.

High Correlation Filter

High Correlation refers to the case when two variables carry approximately similar information. Due to this factor, the performance of the model can be degraded. This correlation between the independent numerical variable gives the calculated value of the correlation coefficient. If this value is higher than the threshold value, we can remove one of the variables from the dataset. We can consider those variables or features that show a high correlation with the target variable.

Random Forest

Random Forest is a popular and very useful feature selection algorithm in machine learning. This algorithm contains an in-built feature importance package, so we do not need to program it separately. In this technique, we need to generate a large set of trees against the target variable, and with the help of usage statistics of each attribute, we need to find the subset of features.

Random forest algorithm takes only numerical variables, so we need to convert the input data into numeric data using hot encoding.

Factor Analysis

Factor analysis is a technique in which each variable is kept within a group according to the correlation with other variables, it means variables within a group can have a high correlation between themselves, but they have a low correlation with variables of other groups.

We can understand it by an example, such as if we have two variables Income and spend. These two variables have a high correlation, which means people with high income spends more, and vice versa. So, such variables are put into a group, and that group is known as the factor. The number of these factors will be reduced as compared to the original dimension of the dataset.

Auto-encoders

One of the popular methods of dimensionality reduction is auto-encoder, which is a type of ANN or artificial neural network, and its main aim is to copy the inputs to their outputs. In this, the input is compressed into latent-space representation, and output is occurred using this representation. It has mainly two parts:

- Encoder: The function of the encoder is to compress the input to form the latent-space representation.

- Decoder: The function of the decoder is to recreate the output from the latent-space representation.

Dimension Reduction-

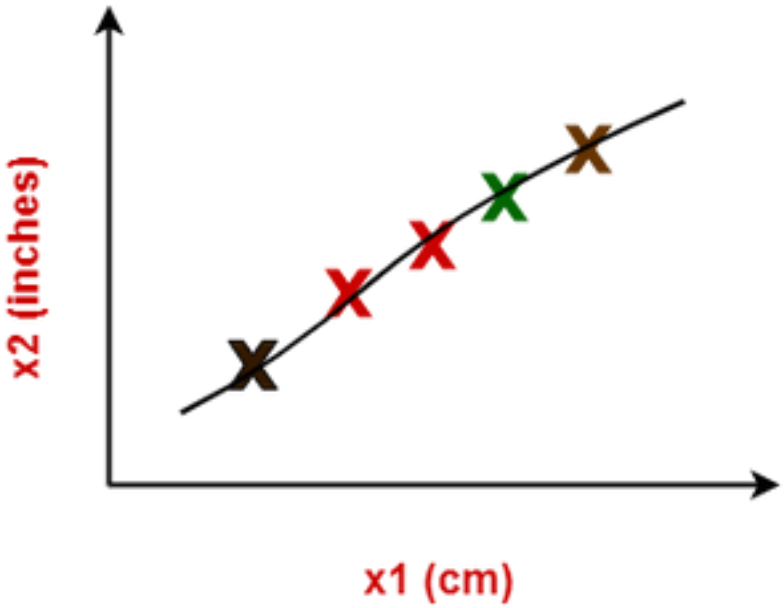
In pattern recognition, Dimension Reduction is defined as-

- It is a process of converting a data set having vast dimensions into a data set with lesser dimensions.
- It ensures that the converted data set conveys similar information concisely.

Example-

Consider the following example-

- The following graph shows two dimensions x1 and x2.
- x1 represents the measurement of several objects in cm.
- x2 represents the measurement of several objects in inches.



In machine learning,

- Using both these dimensions convey similar information.
- Also, they introduce a lot of noise in the system.
- So, it is better to use just one dimension.

Using dimension reduction techniques-

- We convert the dimensions of data from 2 dimensions (x1 and x2) to 1 dimension (z1).
- It makes the data relatively easier to explain.

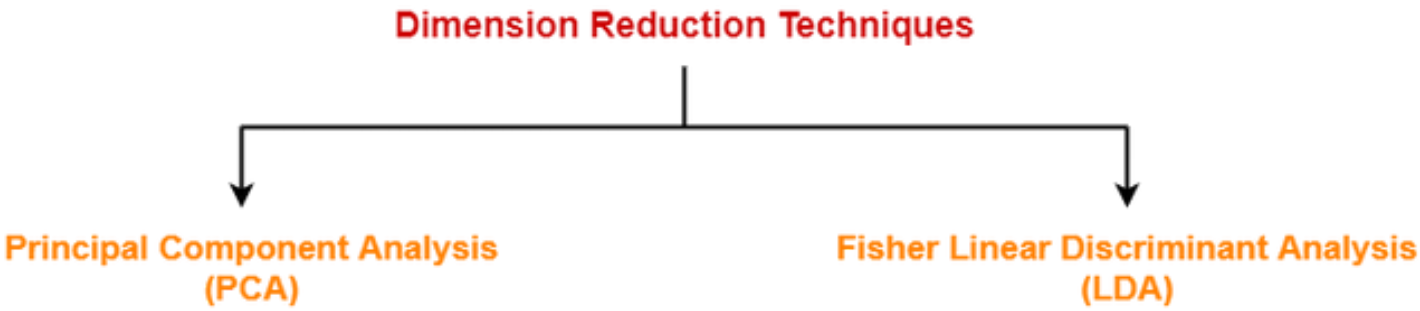
Benefits-

Dimension reduction offers several benefits such as-

- It compresses the data and thus reduces the storage space requirements.
- It reduces the time required for computation since less dimensions require less computation.
- It eliminates the redundant features.
- It improves the model performance.

Dimension Reduction Techniques-

The two popular and well-known dimension reduction techniques are-



1. Principal Component Analysis (PCA)
2. Fisher Linear Discriminant Analysis (LDA)

In this article, we will discuss about Principal Component Analysis.

Principal Component Analysis-

- Principal Component Analysis is a well-known dimension reduction technique.
- It transforms the variables into a new set of variables called as principal components.
- These principal components are linear combination of original variables and are orthogonal.
- The first principal component accounts for most of the possible variation of original data.
- The second principal component does its best to capture the variance in the data.
- There can be only two principal components for a two-dimensional data set.

PCA Algorithm-

The steps involved in PCA Algorithm are as follows-

Step-01: Get data.

Step-02: Compute the mean vector (μ).

Step-03: Subtract mean from the given data.

Step-04: Calculate the covariance matrix.

Step-05: Calculate the eigen vectors and eigen values of the covariance matrix.

Step-06: Choosing components and forming a feature vector.

Step-07: Deriving the new data set.

PRACTICE PROBLEMS BASED ON PRINCIPAL COMPONENT ANALYSIS-

Problem-01:

Given data = { 2, 3, 4, 5, 6, 7 ; 1, 5, 3, 6, 7, 8 }.

Compute the principal component using PCA Algorithm.

OR

Consider the two dimensional patterns (2, 1), (3, 5), (4, 3), (5, 6), (6, 7), (7, 8).

Compute the principal component using PCA Algorithm.

OR

Compute the principal component of following data-

CLASS 1

X = 2 , 3 , 4

Y = 1 , 5 , 3

CLASS 2

X = 5 , 6 , 7

Y = 6 , 7 , 8

Solution-

We use the above discussed PCA Algorithm-

Step-01:

Get data.

The given feature vectors are-

- x1 = (2, 1)
- x2 = (3, 5)
- x3 = (4, 3)
- x4 = (5, 6)
- x5 = (6, 7)
- x6 = (7, 8)

Step-02:

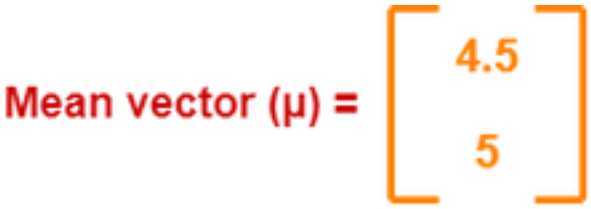
Calculate the mean vector (μ).

Mean vector (μ)

= ((2 + 3 + 4 + 5 + 6 + 7) / 6, (1 + 5 + 3 + 6 + 7 + 8) / 6)

= (4.5, 5)

Thus,



Step-03:

Subtract mean vector (μ) from the given feature vectors.

- x1 – μ = (2 – 4.5, 1 – 5) = (-2.5, -4)
- x2 – μ = (3 – 4.5, 5 – 5) = (-1.5, 0)
- x3 – μ = (4 – 4.5, 3 – 5) = (-0.5, -2)
- x4 – μ = (5 – 4.5, 6 – 5) = (0.5, 1)
- x5 – μ = (6 – 4.5, 7 – 5) = (1.5, 2)
- x6 – μ = (7 – 4.5, 8 – 5) = (2.5, 3)

Feature vectors (xi) after subtracting mean vector (μ) are-

$$\begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 2.5 \\ 3 \end{bmatrix}$$

Step-04:

Calculate the covariance matrix.

Covariance matrix is given by-

$$\text{Covariance Matrix} = \frac{\sum (x_i - \mu)(x_i - \mu)^t}{n}$$

Now,

$$m_1 = (x_1 - \mu)(x_1 - \mu)^t = \begin{bmatrix} -2.5 \\ -4 \end{bmatrix} \begin{bmatrix} -2.5 & -4 \end{bmatrix} = \begin{bmatrix} 6.25 & 10 \\ 10 & 16 \end{bmatrix}$$

$$m_2 = (x_2 - \mu)(x_2 - \mu)^t = \begin{bmatrix} -1.5 \\ 0 \end{bmatrix} \begin{bmatrix} -1.5 & 0 \end{bmatrix} = \begin{bmatrix} 2.25 & 0 \\ 0 & 0 \end{bmatrix}$$

$$m_3 = (x_3 - \mu)(x_3 - \mu)^t = \begin{bmatrix} -0.5 \\ -2 \end{bmatrix} \begin{bmatrix} -0.5 & -2 \end{bmatrix} = \begin{bmatrix} 0.25 & 1 \\ 1 & 4 \end{bmatrix}$$

$$m_4 = (x_4 - \mu)(x_4 - \mu)^t = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} \begin{bmatrix} 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$m_5 = (x_5 - \mu)(x_5 - \mu)^t = \begin{bmatrix} 1.5 \\ 2 \end{bmatrix} \begin{bmatrix} 1.5 & 2 \end{bmatrix} = \begin{bmatrix} 2.25 & 3 \\ 3 & 4 \end{bmatrix}$$

$$m_6 = (x_6 - \mu)(x_6 - \mu)^t = \begin{bmatrix} 2.5 \\ 3 \end{bmatrix} \begin{bmatrix} 2.5 & 3 \end{bmatrix} = \begin{bmatrix} 6.25 & 7.5 \\ 7.5 & 9 \end{bmatrix}$$

Now,

Covariance matrix

= (m1 + m2 + m3 + m4 + m5 + m6) / 6

On adding the above matrices and dividing by 6, we get-

$$\text{Covariance Matrix} = \frac{1}{6}\begin{bmatrix} 17.5 & 22 \\ 22 & 34 \end{bmatrix}$$

$$\text{Covariance Matrix} = \begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix}$$

Step-05:

Calculate the eigen values and eigen vectors of the covariance matrix.

λ is an eigen value for a matrix M if it is a solution of the characteristic equation |M – λI| = 0.

So, we have-

$$\begin{vmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{vmatrix} - \begin{vmatrix} \lambda & 0 \\ 0 & \lambda \end{vmatrix} = 0$$

$$\begin{vmatrix} 2.92 - \lambda & 3.67 \\ 3.67 & 5.67 - \lambda \end{vmatrix} = 0$$

From here,

$(2.92 – \lambda)(5.67 – \lambda) – (3.67 \times 3.67) = 0$

$16.56 – 2.92\lambda – 5.67\lambda + \lambda 2 – 13.47 = 0$

$\lambda 2 – 8.59\lambda + 3.09 = 0$

Solving this quadratic equation, we get $\lambda = 8.22, 0.38$

Thus, two eigen values are $\lambda 1 = 8.22$ and $\lambda 2 = 0.38$.

Clearly, the second eigen value is very small compared to the first eigen value.

So, the second eigen vector can be left out.

Eigen vector corresponding to the greatest eigen value is the principal component for the given data set.

So. we find the eigen vector corresponding to eigen value $\lambda 1$.

We use the following equation to find the eigen vector-

$$MX = \lambda X$$

where-

- M = Covariance Matrix
- X = Eigen vector
- λ = Eigen value

Substituting the values in the above equation, we get-

$$\begin{bmatrix} 2.92 & 3.67 \\ 3.67 & 5.67 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} = 8.22 \begin{bmatrix} X1 \\ X2 \end{bmatrix}$$

Solving these, we get-

$2.92X1 + 3.67X2 = 8.22X1$

3.67X1 + 5.67X2 = 8.22X2

On simplification, we get-

5.3X1 = 3.67X2 ……(1)

3.67X1 = 2.55X2 ……(2)

From (1) and (2), X1 = 0.69X2

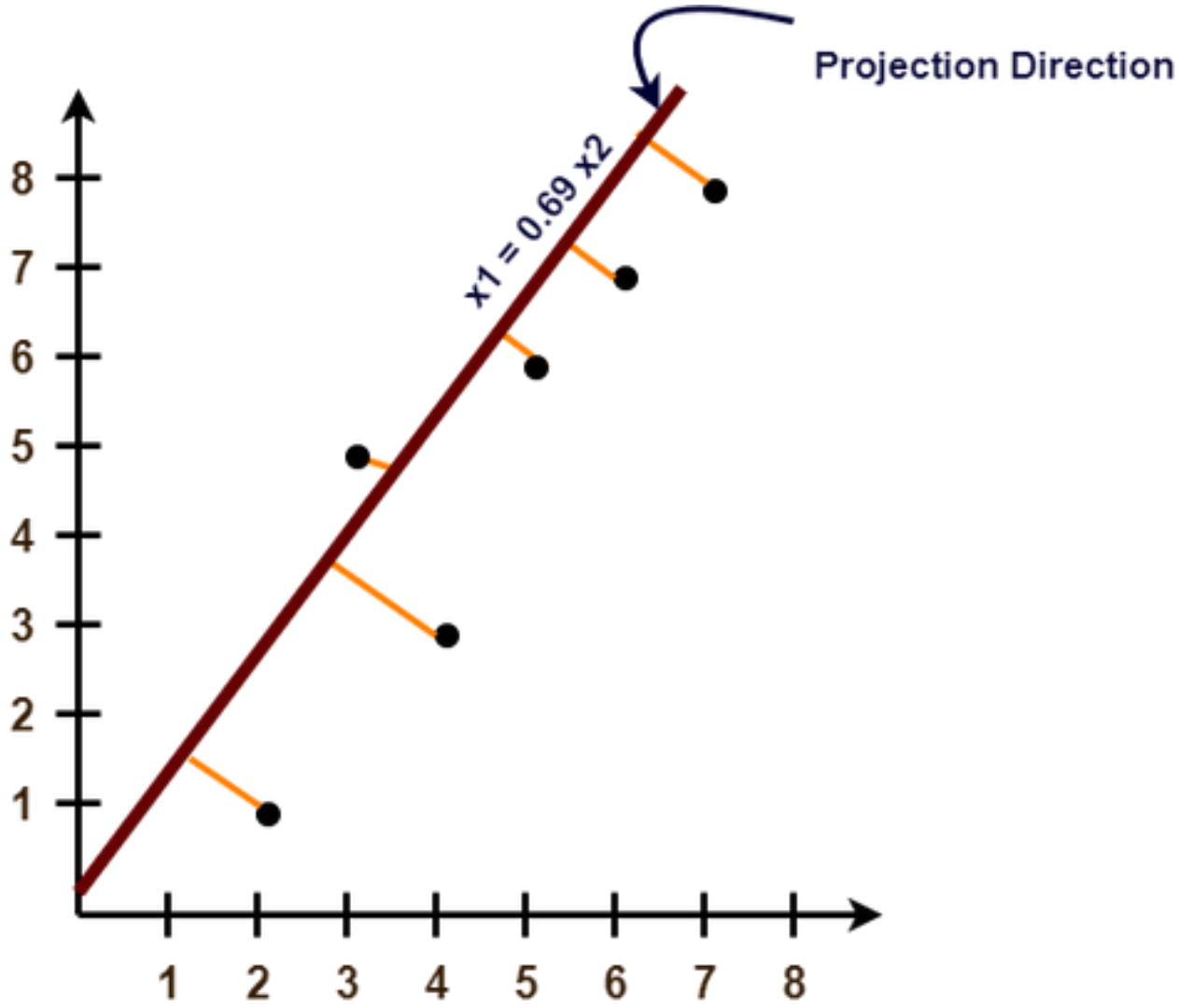
From (2), the eigen vector is-

$$\text{Eigen Vector : } \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

Thus, principal component for the given data set is-

$$\text{Principal Component : } \begin{bmatrix} X1 \\ X2 \end{bmatrix} = \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}$$

Lastly, we project the data points onto the new subspace as-



Problem-02:

Use PCA Algorithm to transform the pattern (2, 1) onto the eigen vector in the previous question.

Solution-

The given feature vector is (2, 1).

Given Feature Vector : $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

The feature vector gets transformed to

= Transpose of Eigen vector x (Feature Vector – Mean Vector)

$$= \begin{bmatrix} 2.55 \\ 3.67 \end{bmatrix}^T \times \left(\begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 4.5 \\ 5 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 2.55 & 3.67 \end{bmatrix} \times \begin{bmatrix} -2.5 \\ -4 \end{bmatrix}$$

$$= -21.055$$