

Support Vector Machine

Support Vector Machine

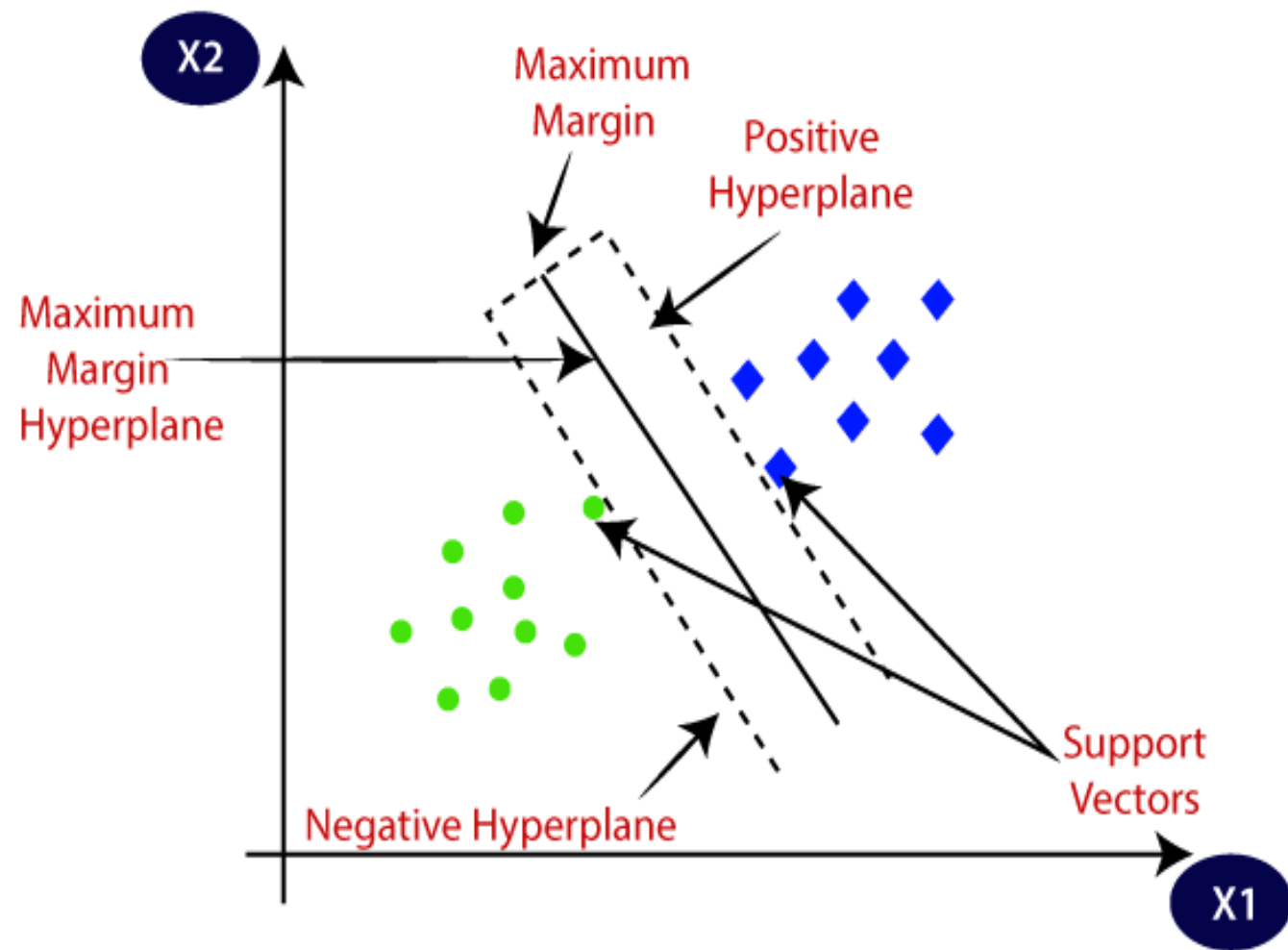
- **Support vector machines (SVMs)** are a set of supervised learning methods used for classification and regression.
- Though we say regression problems as well its best suited for **classification**.
- The goal of the SVM algorithm is to create the **best line or decision boundary** that can segregate 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.

Support Vector Machine

- SVMs is useful because they can find complex relationships between your data without you needing to do a lot of transformations on your own. It's a great option when you are working with smaller datasets that have tens to hundreds of thousands of features. They typically find more accurate results when compared to other algorithms because of their ability to handle small, complex datasets.
- SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, and in web pages.

How an SVM works

- A simple linear SVM classifier works by making a straight line between two classes.
- That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category.
- This means there can be an infinite number of lines to choose from.
- The specialty of SVM is that it chooses the best line to classify your data points. It chooses the line that separates the data and is the furthest away from the closet data points as possible.

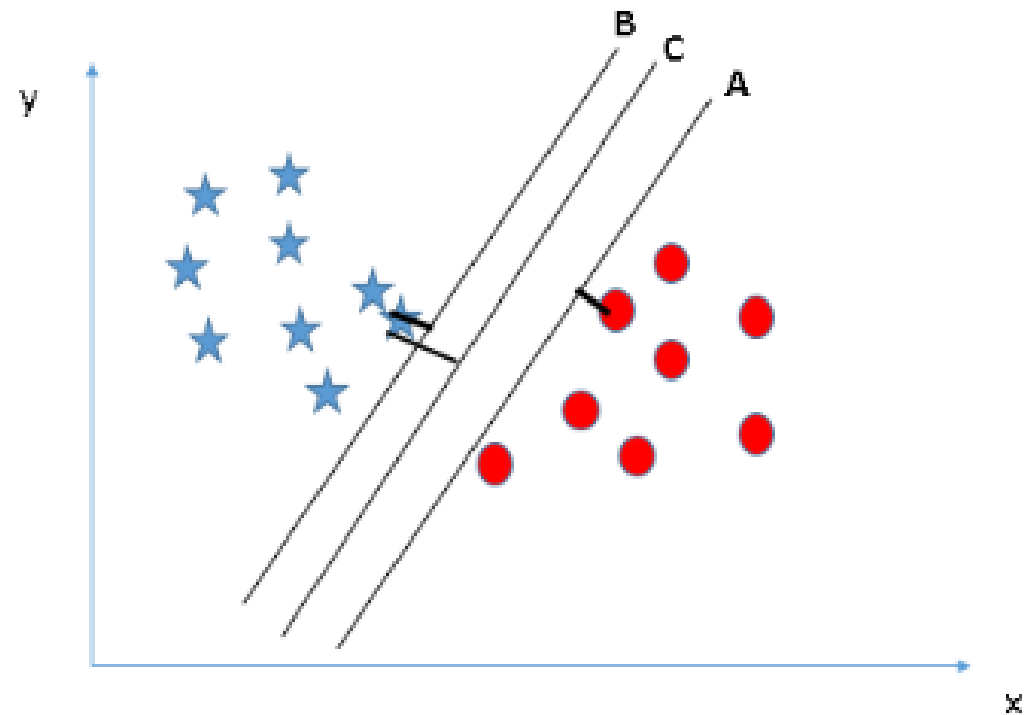
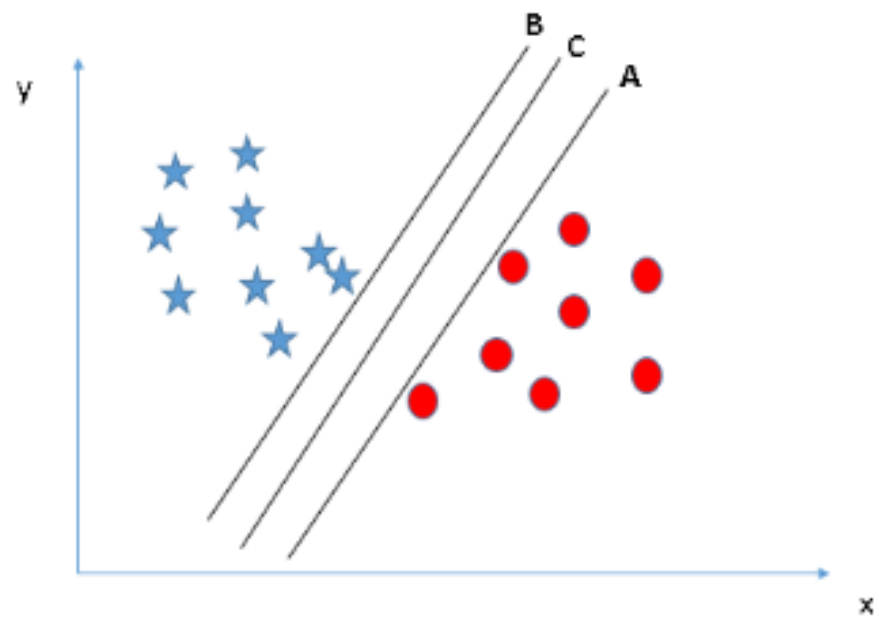
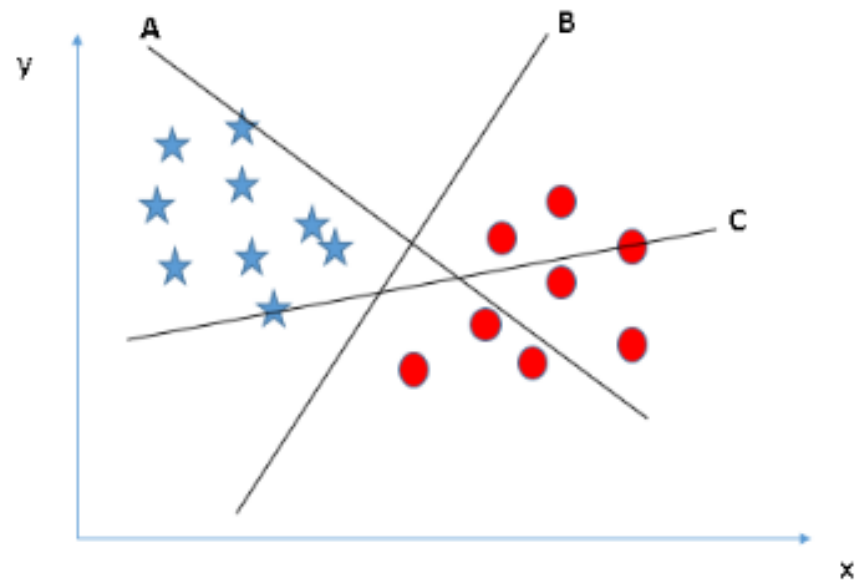


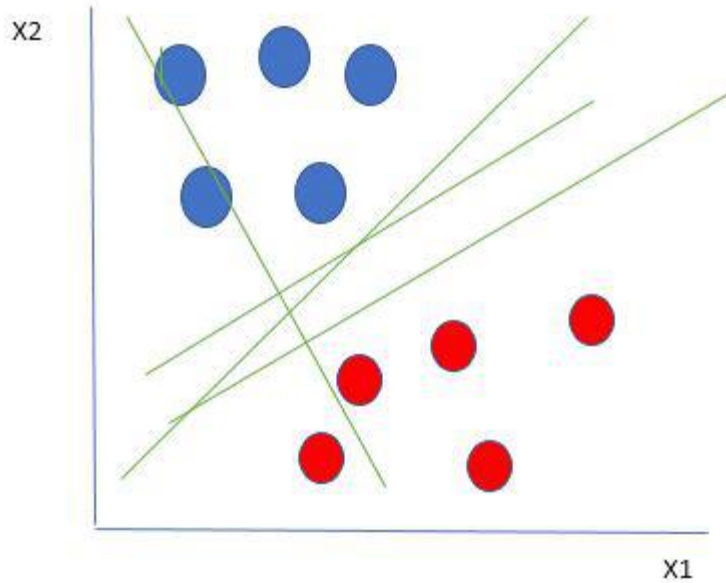
Hyperplane

- 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.
- Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes.
- This best boundary is known as the hyperplane of SVM.

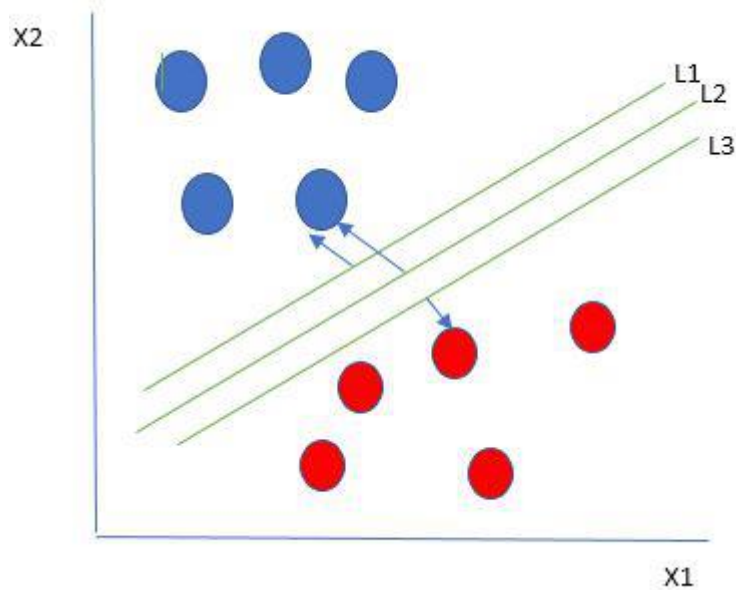
Hyperplane

- The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features, 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.
- Reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is **high chance of miss-classification**.



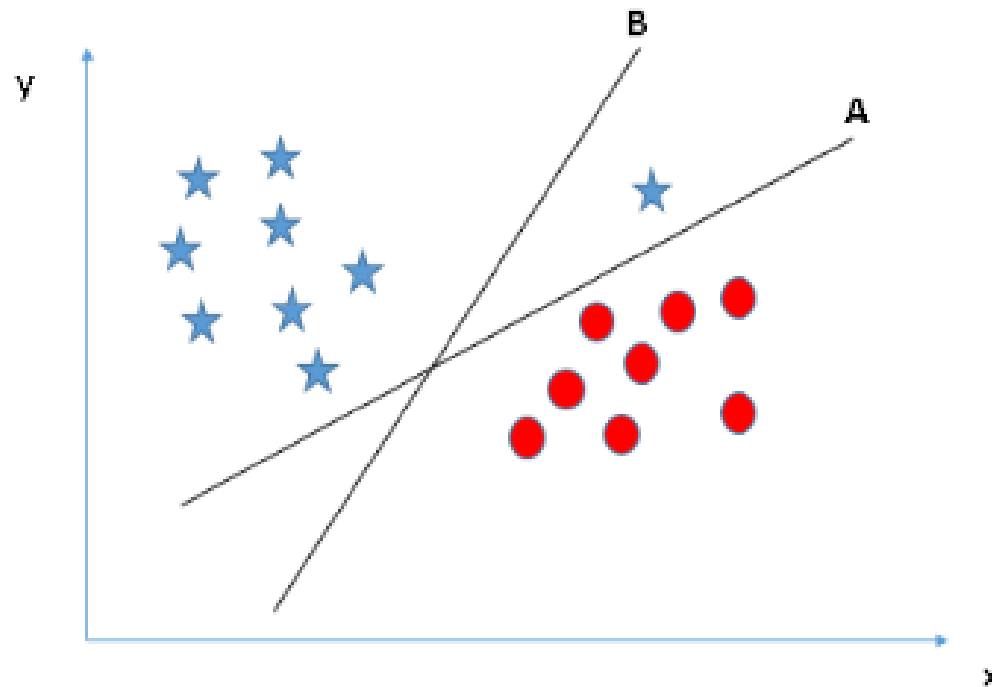


- There are multiple lines that segregates our data points or does a classification between red and blue circles.
- So how do we choose the best line or in general the best hyperplane that segregates our data points.

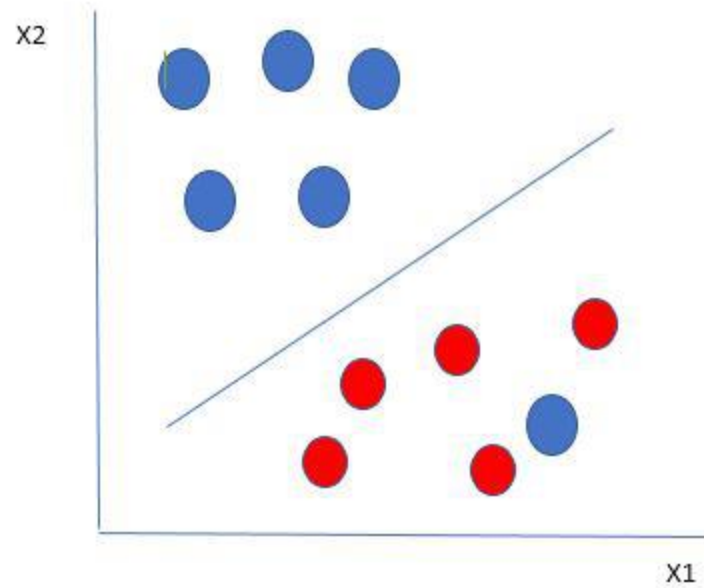
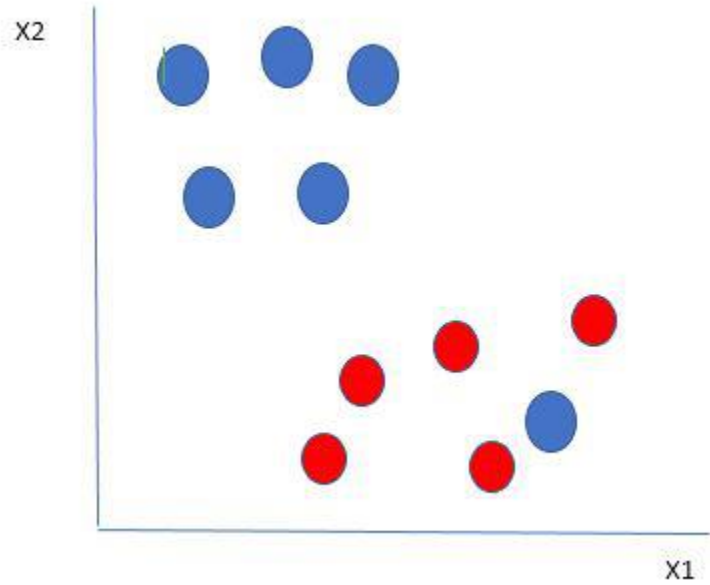


- We choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the maximum-margin hyperplane/hard margin.
- From the figure, we choose L_2 .

Variations in Hyperplane

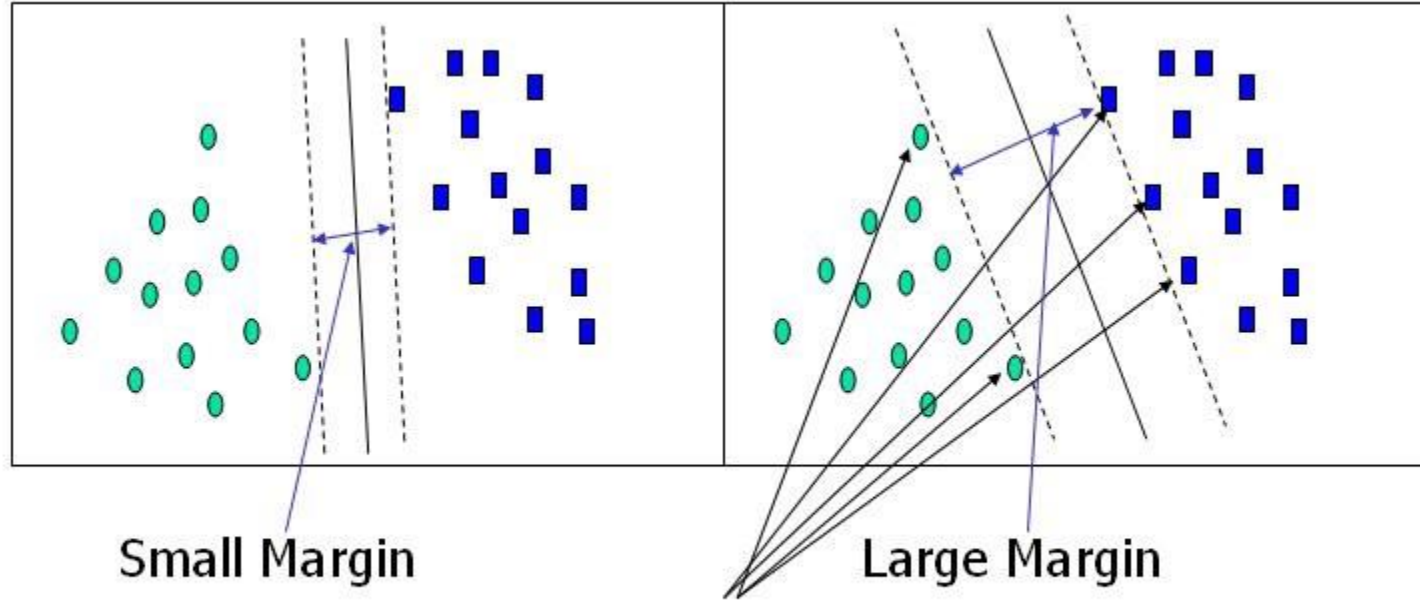


Variations in Hyperplane



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.
- Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.



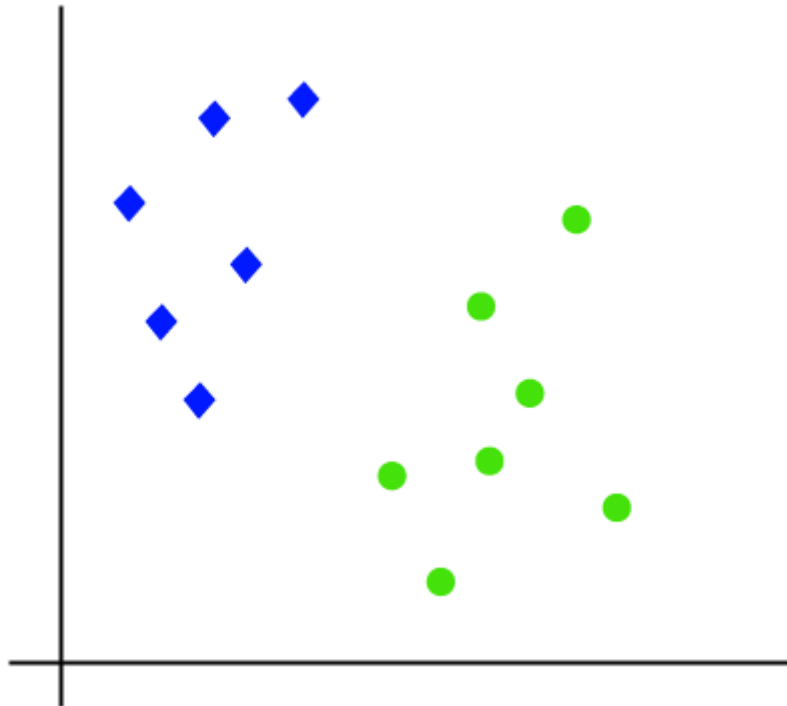
Small Margin

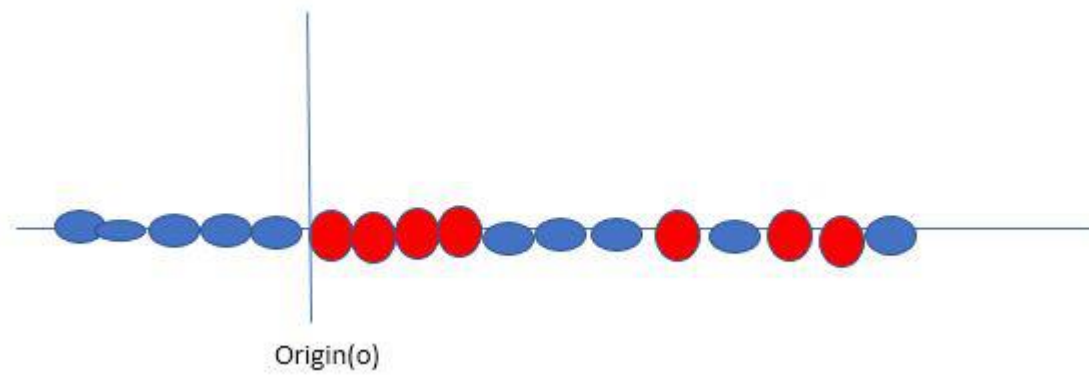
Large Margin

Support Vectors

Types of SVM

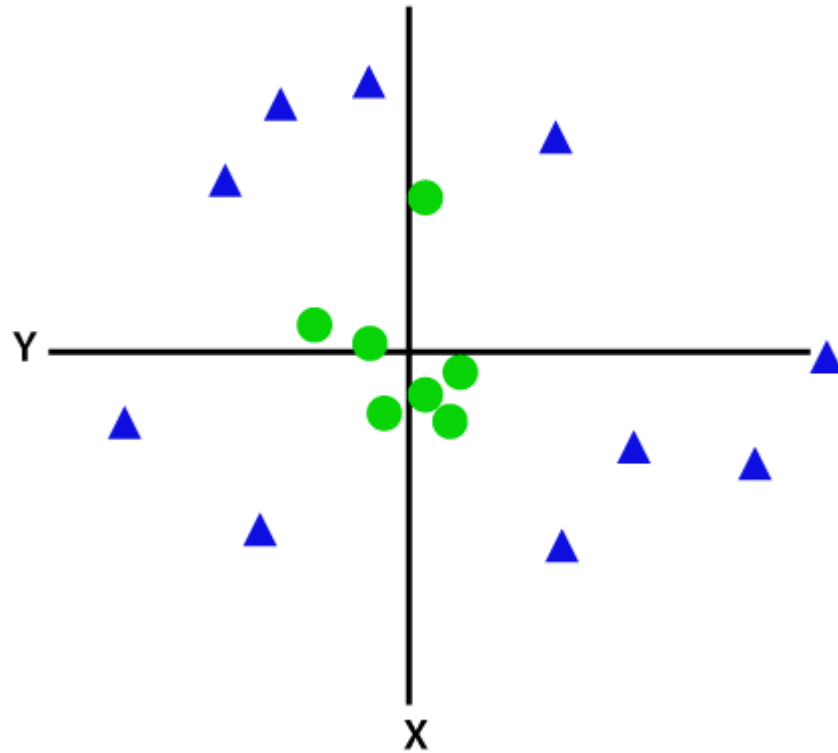
- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

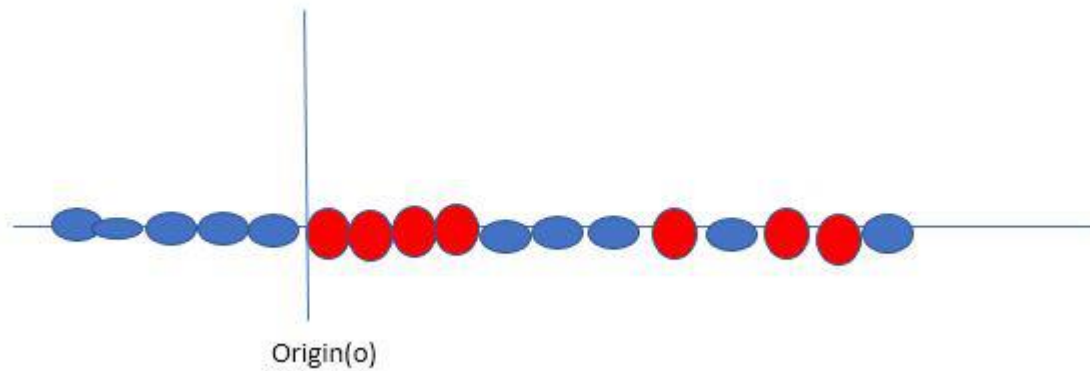




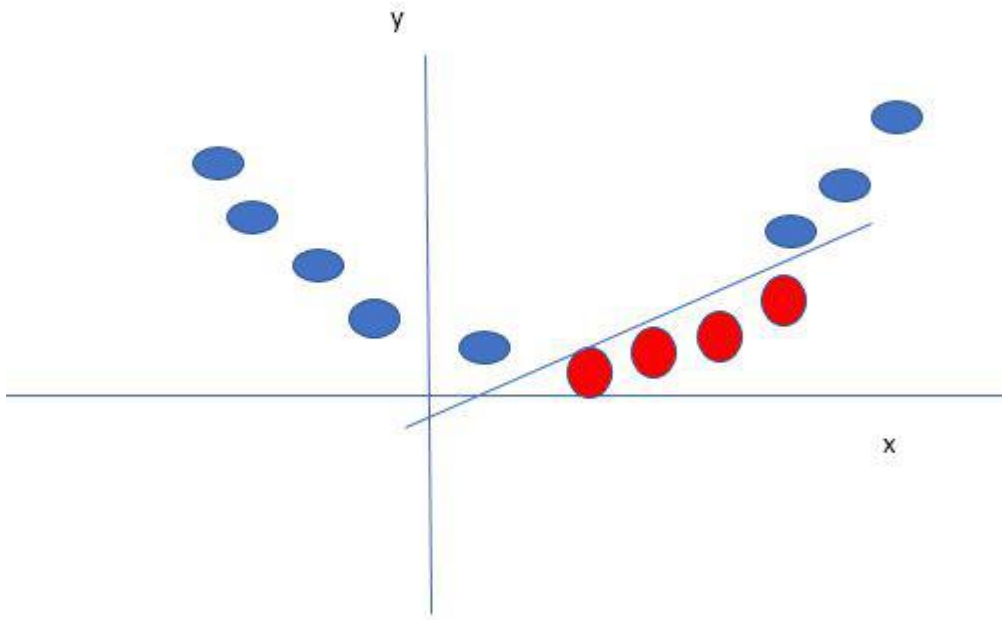
Types of SVM

- **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.



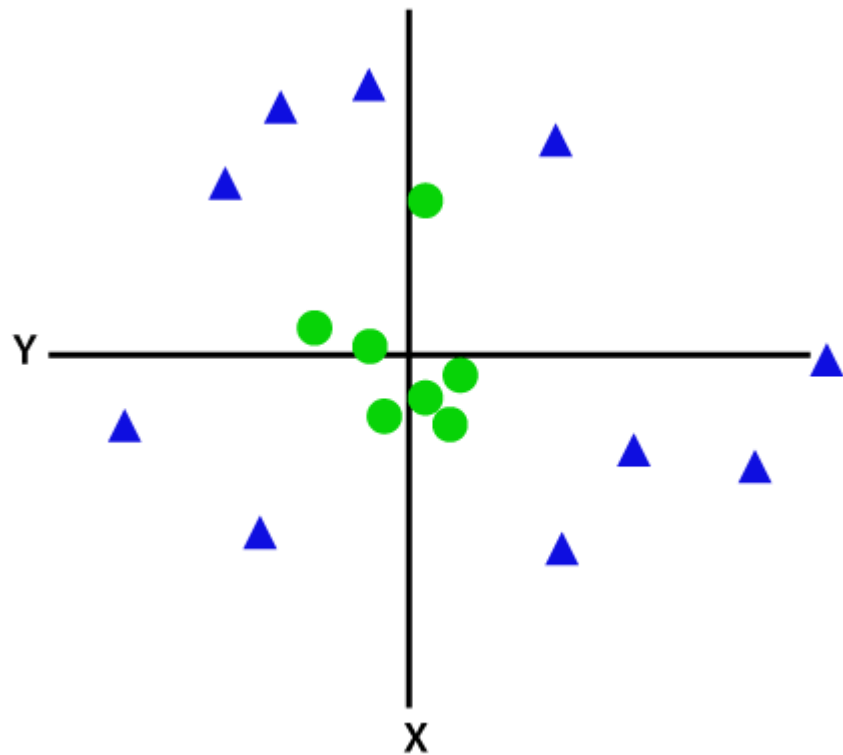


- SVM solves this by creating a new variable using a kernel.
- A non-linear function that creates a new variable is referred to as **kernel**.

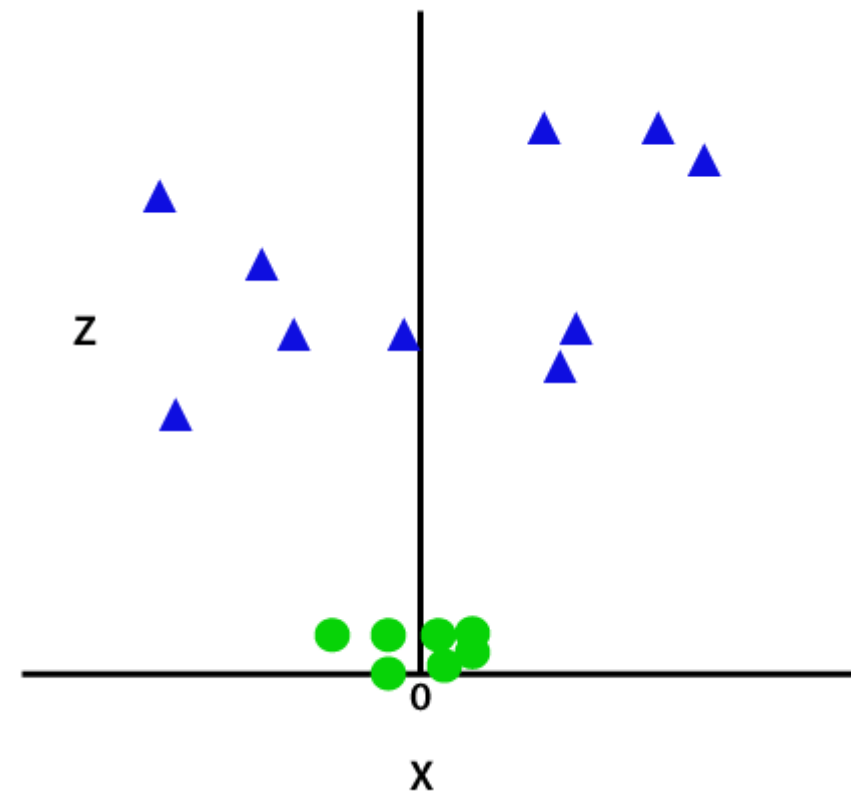


SVM Kernel

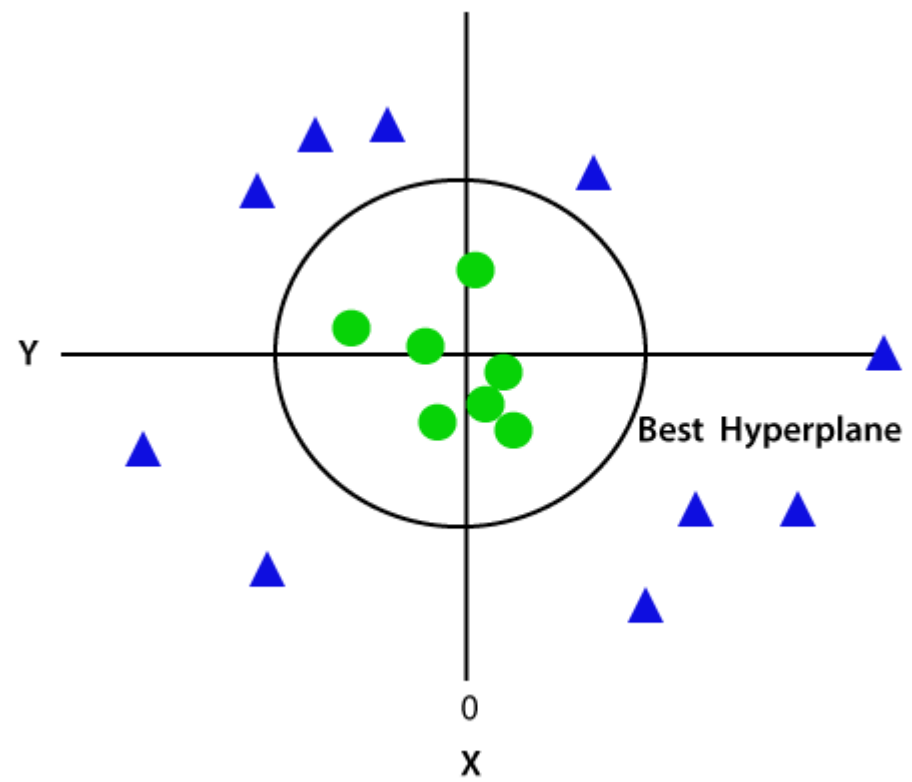
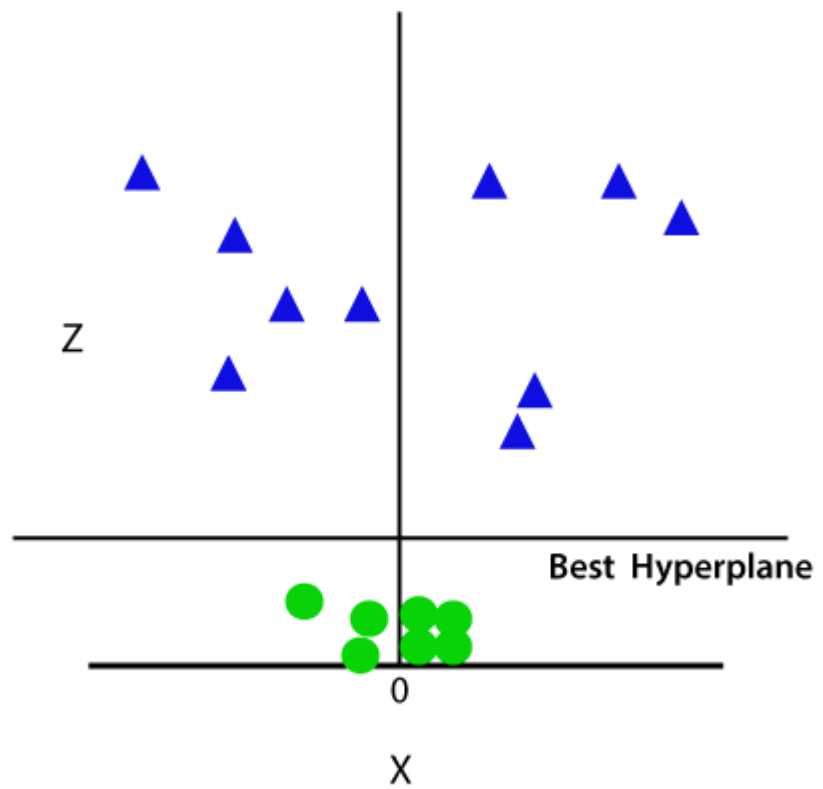
- The SVM kernel is a function that takes low dimensional input space and transforms it into higher-dimensional space.
- It converts not separable problem to separable problem. It is mostly useful in non-linear separation problems.
- Simply put the kernel, it does some extremely complex data transformations then finds out the process to separate the data based on the labels or outputs defined.



- we need to add one more dimension.
- $z = x^2 + y^2$



Sample Space after adding third dimension



Kernel Functions

- Linear
- Polynomial
- *Radial Basis Function* (RBF)
- Sigmoid
 - More useful in neural networks than in support vector machines, but there are occasional specific use cases.

Steps for Data Preprocessing

- **Getting the dataset**
- **Importing libraries**
- **Importing datasets**
- **Finding Missing Data**
- **Encoding Categorical Data**
- **Splitting dataset into training and test set**
- **Feature scaling**

Steps

- **Getting the dataset: CSV Files**
- **Importing Libraries**
 - Import pandas as pd
 - is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library
 - import matplotlib as mp
 - Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code.
 - import numpy as nm
 - Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices

- Importing the Datasets
 - `data_set= pd.read_csv('Dataset.csv')`
 - `data_set.head()`
 - `data_set.tail()`
 - `data_set.shape()`
 - `data_set['class'].value_counts()`

Sepal.Length	Sepal.width	Petal.Length	Petal.width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3	1.4	0.1	setosa
4.3	3	1.1	0.1	setosa
5.8	4	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa
5.4	3.4	1.7	0.2	setosa
5.1	3.7	1.5	0.4	setosa
4.6	3.6	1	0.2	setosa
5.1	3.3	1.7	0.5	setosa
4.8	3.4	1.9	0.2	setosa
5	3	1.6	0.2	setosa
5	3.4	1.6	0.4	setosa
5.2	3.5	1.5	0.2	setosa
5.2	3.4	1.4	0.2	setosa
4.7	3.2	1.6	0.2	setosa
4.8	3.1	1.6	0.2	setosa
5.4	3.4	1.5	0.4	setosa
5.2	4.1	1.5	0.1	setosa
5.5	4.2	1.4	0.2	setosa
4.9	3.1	1.5	0.2	setosa
5	3.2	1.2	0.2	setosa
5.5	3.5	1.3	0.2	setosa
4.9	3.6	1.4	0.1	setosa
4.4	3	1.3	0.2	setosa
5.1	3.4	1.5	0.2	setosa
5	3.5	1.3	0.3	setosa
4.5	2.3	1.3	0.3	setosa
4.4	3.2	1.3	0.2	setosa
5	3.5	1.6	0.6	setosa
5.1	3.8	1.9	0.4	setosa

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
data_set.head()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
data_set.tail()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
data_set.tail(10)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
140	6.7	3.1	5.6	2.4	virginica
141	6.9	3.1	5.1	2.3	virginica
142	5.8	2.7	5.1	1.9	virginica
143	6.8	3.2	5.9	2.3	virginica
144	6.7	3.3	5.7	2.5	virginica
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
data_set.shape
```

(150, 5)

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
data_set['Species'].value_counts()
```

```
setosa      50
versicolor  50
virginica   50
Name: Species, dtype: int64
```

```
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')
df['Petal.Width'].value_counts()
```

```
0.2    29
1.3    13
1.8    12
1.5    12
1.4     8
2.3     8
1.0     7
0.4     7
0.3     7
2.1     6
2.0     6
0.1     5
1.2     5
1.9     5
1.6     4
2.5     3
2.2     3
2.4     3
1.1     3
1.7     2
0.6     1
0.5     1
Name: Petal.Width, dtype: int64
```

- **Extracting dependent and independent variables:**
- To extract an independent variable, we will use **`iloc[]`** method of Pandas library.
- It enables us to select a particular cell of the dataset, that is, it helps us select a value that belongs to a particular row or column from a set of values of a data frame or dataset.
- It is used to extract the required rows and columns from the dataset.
- `x= data_set.iloc[:, :-1].values`
- In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
x=data_set.iloc[:,:].values
x
```

```
array([[5.1, 3.5, 1.4, 0.2, 'setosa'],
       [4.9, 3.0, 1.4, 0.2, 'setosa'],
       [4.7, 3.2, 1.3, 0.2, 'setosa'],
       [4.6, 3.1, 1.5, 0.2, 'setosa'],
       [5.0, 3.6, 1.4, 0.2, 'setosa'],
       [5.4, 3.9, 1.7, 0.4, 'setosa'],
       [4.6, 3.4, 1.4, 0.3, 'setosa'],
       [5.0, 3.4, 1.5, 0.2, 'setosa'],
       [4.4, 2.9, 1.4, 0.2, 'setosa'],
       [4.9, 3.1, 1.5, 0.1, 'setosa'],
       [5.4, 3.7, 1.5, 0.2, 'setosa'],
       [4.8, 3.4, 1.6, 0.2, 'setosa'],
       [4.8, 3.0, 1.4, 0.1, 'setosa'],
       [4.3, 3.0, 1.1, 0.1, 'setosa'],
       [5.8, 4.0, 1.2, 0.2, 'setosa'],
       _
```

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as num
data_set=pd.read_csv('/content/sample_data/iris.csv')
x=data_set.iloc[:, :-1].values
x
```

```
[5.8, 2.6, 4. , 1.2],
[5. , 2.3, 3.3, 1. ],
[5.6, 2.7, 4.2, 1.3],
[5.7, 3. , 4.2, 1.2],
[5.7, 2.9, 4.2, 1.3],
[6.2, 2.9, 4.3, 1.3],
[5.1, 2.5, 3. , 1.1],
[5.7, 2.8, 4.1, 1.3],
[6.3, 3.3, 6. , 2.5],
[5.8, 2.7, 5.1, 1.9],
[7.1, 3. , 5.9, 2.1],
[6.3, 2.9, 5.6, 1.8],
```

- **Extracting dependent variable:**
- To extract dependent variables, we will use Pandas .iloc[] method.
- `y= data_set.iloc[:,3].values`
- Here we have taken all the rows with the last column only. It will give the array of dependent variables.

- **Handling Missing data:**
- **By deleting the particular row:** The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values.
- **By calculating the mean:** In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value.
- Another method is 'most frequent'

```
import numpy as np

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing_values = np.nan,
                        strategy = 'mean')

data = [[12, np.nan, 34], [10, 32, np.nan],
        [np.nan, 11, 20]]
print("Original Data : \n", data)
# Fitting the data to the imputer object
imputer = imputer.fit(data)

# Imputing the data
data = imputer.transform(data)

print("Imputed Data : \n", data)
```

```
Original Data :
[[12, nan, 34], [10, 32, nan], [nan, 11, 20]]
Imputed Data :
[[12.  21.5 34. ]
 [10.  32.  27. ]
 [11.  11.  20. ]]
```

Encoding Categorical Data

- What is categorical data?
- Categorical variables are usually represented as '**strings**' or '**categories**' and are finite in number.
- The city where a person lives: Delhi, Mumbai, Ahmedabad, Bangalore, etc.
- The department a person works in: Finance, Human resources, IT, Production.
- The highest degree a person has: High school, Diploma, Bachelors, Masters, PhD.
- The grades of a student: A+, A, B+, B, B- etc.
- There are two kinds of categorical data-
- **Ordinal Data:** The categories have an inherent order
- **Nominal Data:** The categories do not have an inherent order

- In Ordinal data, while encoding, one should retain the information regarding the order in which the category is provided. Like in the above example the highest degree a person possesses, gives vital information about his qualification.
- While encoding Nominal data, we have to consider the presence or absence of a feature. In such a case, no notion of order is present. For example, the city a person lives in.
- **One Hot Encoding**
- `from sklearn.preprocessing import LabelEncoder, OneHotEncoder`

Index	Animal
0	Dog
1	Cat
2	Sheep
3	Horse
4	Lion

One-Hot code



Index	Dog	Cat	Sheep	Lion	Horse
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0

Splitting dataset into training and test set

- [train_test_split\(\) function](#).
- The function takes a loaded dataset as input and returns the dataset split into two subsets.

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train
```

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=0)
X_train
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
3	4.6	3.1	1.5	0.2
149	5.9	3.0	5.1	1.8
98	5.1	2.5	3.0	1.1
6	4.6	3.4	1.4	0.3
68	6.2	2.2	4.5	1.5
...
9	4.9	3.1	1.5	0.1
103	6.3	2.9	5.6	1.8
67	5.8	2.7	4.1	1.0
117	7.7	3.8	6.7	2.2
47	4.6	3.2	1.4	0.2

75 rows × 4 columns

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')

X = df.iloc[:, :-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
X_train
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
137	6.4	3.1	5.5	1.8
84	5.4	3.0	4.5	1.5
27	5.2	3.5	1.5	0.2
127	6.1	3.0	4.9	1.8
132	6.4	2.8	5.6	2.2
...
9	4.9	3.1	1.5	0.1
103	6.3	2.9	5.6	1.8
67	5.8	2.7	4.1	1.0
117	7.7	3.8	6.7	2.2
47	4.6	3.2	1.4	0.2

120 rows × 4 columns

Feature Scaling

- Feature Scaling is a technique to standardize the independent features present in the data in a fixed range.
- It is performed during the data pre-processing to handle highly varying magnitudes or values or units.
- If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.
- We use Feature Scaling to bring all values to the same magnitudes and thus, tackle this issue. we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

Age	Salary	Purchased
38	68000	No
43	45000	Yes
30	54000	No
48	65000	No
40	nan	Yes
35	58000	Yes
nan	53000	No
49	79000	Yes
50	88000	No
37	77000	Yes

- As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**.
- If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

Standardization

$$X' = \frac{x - \text{mean}(x)}{a}$$

Diagram illustrating the Standardization formula:

- new value** points to X' .
- original value** points to x .
- mean** points to $\text{mean}(x)$.
- Standard deviation** points to a .

Normalization

$$X' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Diagram illustrating the Normalization formula:

- new value** points to X' .
- original value** points to x .

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1.0	790	99
Mitsubishi	Space Star	1.2	1160	95
Skoda	Citigo	1.0	929	95
Fiat	500	0.9	865	90
Mini	Cooper	1.5	1140	105
VW	Up!	1.0	929	105
Skoda	Fabia	1.4	1109	90
Mercedes	A-Class	1.5	1365	92
Ford	Fiesta	1.5	1112	98
Audi	A1	1.6	1150	99

$$(790 - \underline{1292.23}) / \underline{238.74} = -2.1$$

Feature Scaling on Weight

$$(1.0 - \underline{1.61}) / \underline{0.38} = -1.59$$

Feature Scaling on Volume

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('/content/sample_data/iris.csv')
X=df.iloc[:,[2,3]]
X
```

	Petal.Length	Petal.Width
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2
...
145	5.2	2.3
146	5.0	1.9
147	5.0	2.0

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
df = pd.read_csv('/content/sample_data/iris.csv')
X=df.iloc[:,[2,3]]
X
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
scaledX = scale.fit_transform(X)
```

```
print(scaledX)
```

```
[ 1.37546573e-01  8.77547895e-04]
[-2.60315415e-01 -2.62386821e-01]
[ 2.51221427e-01  1.32509732e-01]
[ 2.51221427e-01  8.77547895e-04]
[ 2.51221427e-01  1.32509732e-01]
[ 3.08058854e-01  1.32509732e-01]
[-4.30827696e-01 -1.30754636e-01]
[ 1.94384000e-01  1.32509732e-01]
[ 1.27429511e+00  1.71209594e+00]
[ 7.62758269e-01  9.22302838e-01]
[ 1.21745768e+00  1.18556721e+00]
[ 1.04694540e+00  7.90670654e-01]
[ 1.16062026e+00  1.31719939e+00]
[ 1.61531967e+00  1.18556721e+00]
[ 4.21733708e-01  6.59038469e-01]
```

- **Python predict() function** enables us to **predict the labels of the data values** on the basis of the trained model.
- The predict() function **accepts only a single argument** which is usually the data to be tested.
- `Y_pred=classifier.predict(X_test)`

```
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')
X=df.iloc[:, :-1]
y=df.iloc[:, -1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
from sklearn.svm import SVC
classifier=SVC(kernel='rbf')
classifier.fit(X_train,y_train)
Y_pred=classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,Y_pred)
accuracy = float(cm.diagonal().sum())/len(y_test)
print("\nAccuracy Of SVM For The Given Dataset : ", accuracy)
```

Accuracy Of SVM For The Given Dataset : 0.9

```

import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')
X=df.iloc[:, :-1]
y=df.iloc[:, -1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
from sklearn.svm import SVC
classifier=SVC(kernel='linear')
classifier.fit(X_train,y_train)
Y_pred=classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,Y_pred)
accuracy = float(cm.diagonal().sum())/len(y_test)
print("\nAccuracy Of SVM For The Given Dataset : ", accuracy)
cm

```

Accuracy of the model= $14+7+8/30$
Accuracy=0.96

```

Accuracy Of SVM For The Given Dataset :  0.9666666666666667
array([[14,  0,  0],
       [ 0,  7,  1],
       [ 0,  0,  8]])

```

- Although accuracy is an important metric, it is not always adequate to measure a model's performance and is best relied upon when there is no class imbalance (i.e., when the proportion of instances of all the classes is the same) which rarely is the case in any real scenario.
- When there is a class imbalance, accuracy can be misleading too. Let's assume that we have a sample of 100 fruits of which 90 are apples and 5 are bananas and 5 are custard apples. Even if our model predicts all the fruits as apples, the accuracy will be 90% while the truth is that our model is not actually doing a great job!
- Hence, the need for other metrics.
- Accuracy is calculated for the model as a whole but recall and precision are calculated for individual classes. We use macro or micro or weighted scores of recall, precision and F1 score of a model for multiclass classification problems.

```

import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv('/content/sample_data/iris.csv')
X=df.iloc[:, :-1]
y=df.iloc[:, -1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
from sklearn.svm import SVC
classifier=SVC()
classifier.fit(X_train,y_train)
y_predict=classifier.predict(X_test)
from sklearn.metrics import classification_report
print(classification_report(y_test,y_predict))

```







	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	11
versicolor	0.67	1.00	0.80	6
virginica	1.00	0.77	0.87	13
accuracy			0.90	30
macro avg	0.89	0.92	0.89	30
weighted avg	0.93	0.90	0.90	30

Confusion matrix

- A 'Confusion Matrix' is a consolidation of the number of times a model gives a correct or an incorrect inference or simply, the number of times a model rightly identifies the truth (actual classes) and the number of times it gets confused in identifying one class from another.
- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they don't have the disease.
- **false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

Applications of confusion matrix

- A confusion matrix helps measure the performance of a classification problem with the help of different metrics that can be calculated from it.
- As an example, let's assume that our model predicted the fruit types- Apple (A), Banana(B) and Custard apple(C) and gave the confusion matrix
- It gives us the below information about our problem and the model:

		Predicted (what our model says))			
Actual (what the data says)	CLASSES	 A	 B	 C	Row totals
	 A	### ₅	2	3	10
	 B	2	### ₆	0	8
	 C	3	2	₂	7
	Column Totals	10	10	5	25

Diagonal numbers are rightly classified observations

Total number of observations/records

#MLmuse
CLAIRVOYANT

- Total number of observations in the data (i.e., fruits) = 25
- Number of A, B, C type fruits the data has = 10, 8, 7 respectively
- Model saw more instances of apples (A = 10) than other fruit types
- The diagonal observations are the true positives of each class i.e., the number of times the model correctly identified A as A, B as B and C as C
- All other non-diagonal observations are incorrect classifications made by the model

- **Precision**

- Of all the positive predictions I made, how many of them are truly positive?

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall**

- Of all the actual positive examples out there, how many of them did I correctly predict to be positive?

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1 Score**
- To evaluate model performance comprehensively, we should examine **both** precision and recall. The F1 score serves as a helpful metric that considers both of them.
- **Definition:** Harmonic mean of precision and recall for a more balanced summarization of model performance.

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Example

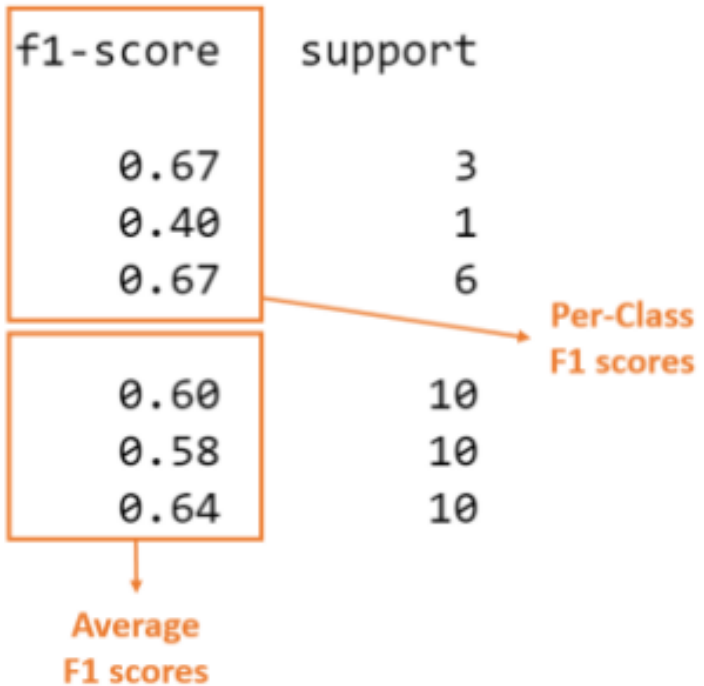
- Imagine we have trained an **image classification model** on a **multi-class** dataset containing images of **three** classes: **Airplane**, **Boat**, and **Car**.

No	Actual	Predicted	Match
1	Airplane	Airplane	✓
2	Car	Boat	✗
3	Car	Car	✓
4	Car	Car	✓
5	Car	Boat	✗
6	Airplane	Boat	✗
7	Boat	Boat	✓
8	Car	Airplane	✗
9	Airplane	Airplane	✓
10	Car	Car	✓







- Upon running `sklearn.metrics.classification_report`




	precision	recall	f1-score	support	
Aeroplane	0.67	0.67	0.67	3	
Boat	0.25	1.00	0.40	1	
Car	1.00	0.50	0.67	6	
accuracy			0.60	10	Per-Class F1 scores
macro avg	0.64	0.72	0.58	10	
weighted avg	0.82	0.60	0.64	10	

Average
F1 scores






Confusion Matrix

		Predicted		
		 Airplane	 Boat	 Car
Actual	 Airplane	2	1	0
	 Boat	0	1	0
	 Car	1	2	3

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)
 Airplane	2	1	1
 Boat	1	3	0
 Car	3	0	3




- F1 score

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$




Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
 Airplane	2	1	1	0.67	0.67	$2 * (0.67 * 0.67) / (0.67 + 0.67)$ = 0.67
 Boat	1	3	0	0.25	1.00	$2 * (0.25 * 1.00) / (0.25 + 1.00)$ = 0.40
 Car	3	0	3	1.00	0.50	$2 * (1.00 * 0.50) / (1.00 + 0.50)$ = 0.67

- **Macro Average**




- The macro-averaged F1 score (or macro F1 score) is computed using the arithmetic mean (aka **unweighted** mean) of all the per-class F1 scores.
- This method treats all classes equally regardless of their **support** values.

Label		Per-Class F1 Score	Macro-Averaged F1 Score
	Airplane	0.67	$\frac{0.67 + 0.40 + 0.67}{3}$ $= \mathbf{0.58}$
	Boat	0.40	
	Car	0.67	

- **Weighted Average**
- The **weighted-averaged** F1 score is calculated by taking the mean of all per-class F1 scores **while considering each class's support**.
- **Support** refers to the number of actual occurrences of the class in the dataset.
 - Aeroplane-3
 - Boat-1
 - Car-6

Label	Per-Class F1 Score	Support	Support Proportion	Weighted Average F1 Score
 Airplane	0.67	3	0.3	$ \begin{aligned} &(0.67 * 0.3) + \\ &(0.40 * 0.1) + \\ &(0.67 * 0.6) \\ &= \mathbf{0.64} \end{aligned} $
 Boat	0.40	1	0.1	
 Car	0.67	6	0.6	
Total	-	10	1.0	

- **Micro Average**
- Micro averaging computes a **global average** F1 score by counting the **sums** of the True Positives (**TP**), False Negatives (**FN**), and False Positives (**FP**).

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Micro-Averaged F1 Score
 Airplane	2	1	1	$\frac{TP}{TP + \frac{1}{2} (FP + FN)} = \frac{6}{6 + \frac{1}{2} (4 + 4)}$ $= 0.60$
 Boat	1	3	0	
 Car	3	0	3	
TOTAL	6	4	4	

- **Which average should I choose?**

- In general, if you are working with an imbalanced dataset where all classes are equally important, using the **macro** average would be a good choice as it treats all classes equally.
- It means that for our example involving the classification of airplanes, boats, and cars, we would use the macro-F1 score.
- If you have an imbalanced dataset but want to assign greater contribution to classes with more examples in the dataset, then the **weighted** average is preferred.
- This is because, in weighted averaging, the contribution of each class to the F1 average is weighted by its size.
- If you have a balanced dataset and want an easily understandable metric for overall performance regardless of the class. In that case, you can go with accuracy, which is essentially our **micro** F1 score.