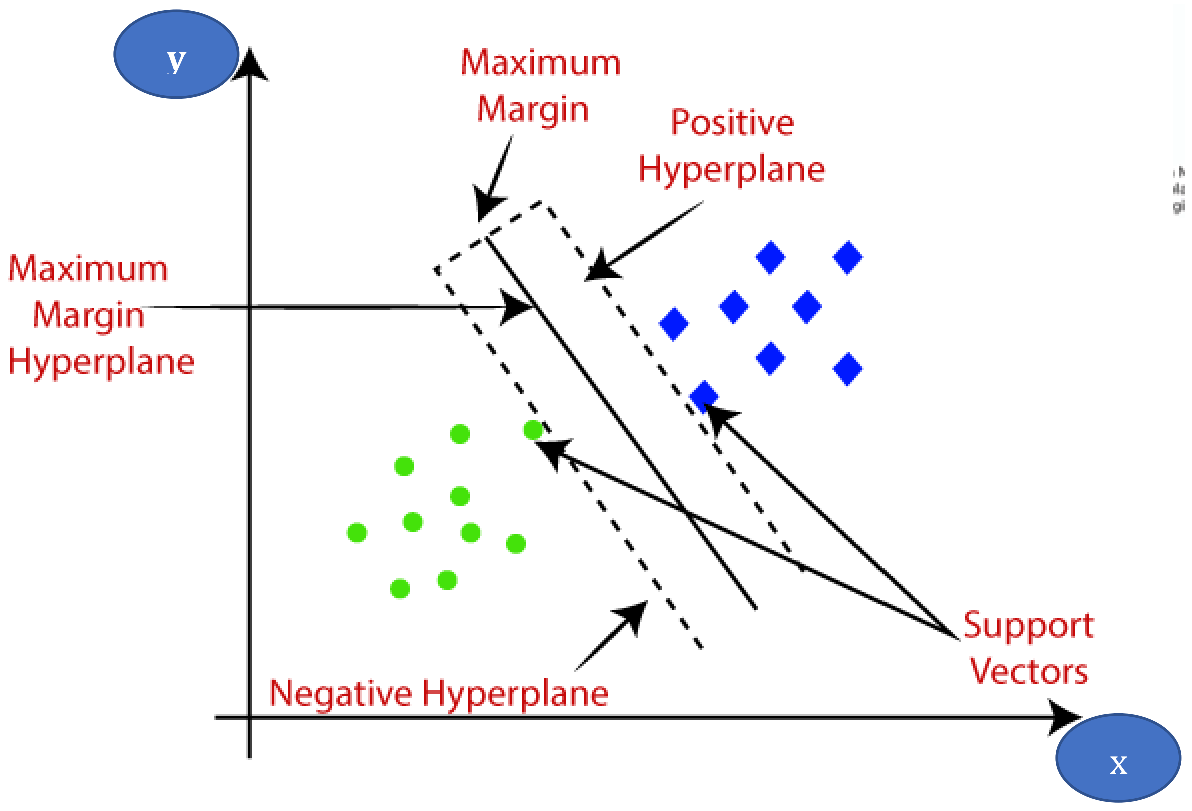
4. Support Vector Machine

- Supervised Machine Learning Model
- Used for both Classification and Regression
- Hyperplane
- Support Vectors

Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression tasks. It is a powerful and versatile algorithm that aims to find an optimal hyperplane or decision boundary in a high-dimensional feature space to separate different classes or predict numerical values.

The fundamental idea behind SVM is to find the hyperplane that maximally separates the data points of different classes. The hyperplane is selected such that the distance between the hyperplane and the closest data points from each class, known as support vectors, is maximized. This distance is called the margin. The support vectors and the hyperplane are the key components of SVM. The support vectors are the crucial data points that influence the construction of the hyperplane, which in turn determines the separation between different classes and enables accurate classification or regression.



Support Vectors: In Support Vector Machine (SVM), support vectors are the data points that lie closest to the decision boundary, known as the hyperplane. These support vectors play a crucial role in defining the decision boundary and determining the optimal hyperplane that maximizes the margin.

The support vectors are the subset of training data points that have the most influence on the construction of the hyperplane. They are the points that are located on or near the margin, as well as the points that are misclassified. These data points are crucial because they define the separation between different classes and contribute to the calculation of the margin.

The choice of support vectors is determined during the training process of the SVM algorithm. The algorithm selects the support vectors based on their distance from the decision boundary. Only the support vectors are necessary to define the hyperplane and make predictions, rather than using all the training data points. This property of SVM makes it memory-efficient and computationally efficient.

Hyperplane: In SVM, the hyperplane is a decision boundary that separates different classes in the feature space. For binary classification tasks, the hyperplane is a (d-1)-dimensional subspace in a d-dimensional feature space.

In a linear SVM, the hyperplane is a linear combination of the input features. Mathematically, it can be represented as:

w^T x + b = 0

where w is the weight vector perpendicular to the hyperplane, x is the input feature vector, and b is the bias term. The weight vector w determines the orientation of the hyperplane, while the bias term b shifts the hyperplane.

The objective of SVM is to find the optimal hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class, i.e., the support vectors. The hyperplane that achieves the maximum margin is considered the best decision boundary, as it provides better generalization to unseen data.

In cases where the data is not linearly separable, SVM uses the kernel trick to transform the feature space into a higher-dimensional space. In this higher-dimensional space, a hyperplane is sought to separate the transformed data. The kernel function computes the inner products of the transformed feature vectors without explicitly calculating the transformation. This allows SVM to capture complex non-linear decision boundaries.

For linearly separable data, SVM finds the hyperplane that achieves the maximum margin. However, when the data is not linearly separable, SVM uses a technique called the kernel trick to transform the original feature space into a higher-dimensional space, where the classes can be separated by a hyperplane.

The kernel trick allows SVM to implicitly map the data into a higher-dimensional space without explicitly calculating the transformed feature vectors. This is computationally efficient and enables SVM to capture complex non-linear relationships between features.

Advantages of Support Vector Machine:

- **Effective in high-dimensional spaces:** SVM performs well even in cases where the number of features is much greater than the number of samples. This makes it suitable for tasks involving a large number of features, such as text classification or image recognition.
- **Versatility:** SVM supports different kernel functions, such as linear, polynomial, and radial basis function (RBF), allowing flexibility in capturing non-linear relationships. This makes SVM adaptable to various types of data and problem domains.
- **Regularization:** SVM includes a regularization parameter (C) that controls the trade-off between maximizing the margin and minimizing the classification errors. This parameter helps prevent overfitting and allows the model to generalize well to unseen data.
- **Robust to outliers:** SVM is less sensitive to outliers compared to other classification algorithms like logistic regression. The use of support vectors, which are the closest data points to the decision boundary, makes the model less affected by outliers.

Limitations of Support Vector Machine:

- **Computationally intensive:** SVM can be computationally expensive, especially when dealing with large datasets. Training time and memory requirements can increase significantly as the number of samples and features grows.
- **Parameter selection:** SVM has several parameters, including the choice of kernel function, regularization parameter (C), and kernel-specific parameters. Selecting appropriate values for these parameters can be challenging and often requires careful tuning.
- **Interpretability:** While SVM can provide accurate predictions, it is not as interpretable as some other models like decision trees or logistic regression. The learned model does not directly provide insights into the relationship between individual features and the target variable.

Applications of Support Vector Machine:

- **Text and document classification:** SVM is widely used for tasks such as sentiment analysis, spam detection, topic classification, and document categorization in natural language processing.
- **Image classification:** SVM has been successfully applied to image recognition tasks, including object detection, facial expression recognition, and handwritten digit recognition.
- **Bioinformatics:** SVM is used in protein structure prediction, gene expression analysis, and disease classification based on genomic data.
- **Financial analysis:** SVM can be applied to credit scoring, stock market prediction, fraud detection, and anomaly detection in financial data.
- **Medical diagnosis:** SVM has been employed in medical diagnosis, including cancer classification, disease prognosis, and identification of genetic markers.
- **Remote sensing:** SVM is used in satellite image analysis, land cover classification, and pattern recognition in remote sensing applications.

Implementing SVM using Python

Dataset Required

https://drive.google.com/file/d/1V6yFU3nDdx9R56yOzy6GxHPq-Dav2A4K/view?usp=share_link

Importing Required Libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from sklearn import metrics
import seaborn as sn
```

Importing (Reading) Datasets

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin',
             'bmi', 'pedigree', 'age', 'label']
data = pd.read_csv('/content/diabets.csv', header= None,
       names=col_names)
print(data.shape)
data.head()
```

```
(768, 9)
```

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Checking for any null values in dataset

```
data.isnull().sum()
```

```
pregnant    0
glucose     0
bp          0
skin        0
insulin     0
bmi         0
pedigree    0
age         0
label       0
dtype: int64
```

Assigning dependent and independent variables

```
feature_cols = ['pregnant','insulin', 'bmi',
                'age','glucose','bp', 'pedigree']
x=data[feature_cols]
y=data.label
```

splitting the dataset into Training and Testing Dataset

```
x_train, x_test, y_train, y_test = train_test_split(x,y,
     test_size=0.3, random_state=5)
display(x_train.shape, y_train.shape, x_test.shape,
     y_test.shape)
```

```
(537, 7)
(537,)
(231, 7)
(231,)
```

Preprocessing Data with StandardScaler

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

Standardization (Z-score normalization) scales the features of a dataset so that they have zero mean and unit variance. This transformation centers the data around the mean and scales it by the standard deviation. It does not enforce a specific range for the transformed values. Normalization, on the other hand, scales the features to a specific range, often between 0 and 1 or -1 and 1. It is achieved by dividing each value by the maximum value in the feature range or by applying other normalization techniques.

Fitting the Model (SVM) using 'rbf' kernel

```
model= SVC(kernel='rbf',random_state=0)
 model.fit(x_train, y_train)
svc_prediction=model.predict(x_test)
print('svc_prediction:  ', svc_prediction)
```

```
svc_prediction:  [1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0
0 0 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0
 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 1 1 0 1 0 0 0 0 0]
```

Evaluation Metrics for 'rbf' kernel

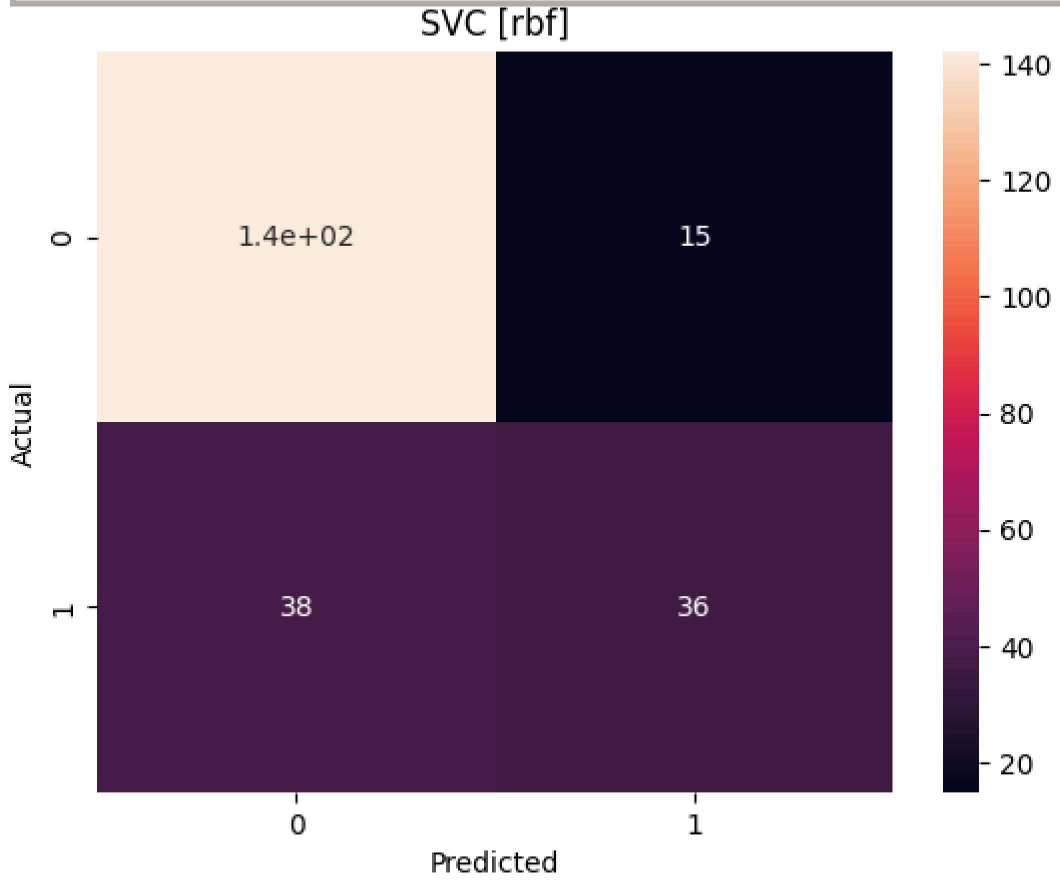
```
conf_mat=metrics.confusion_matrix(y_test, svc_prediction)
 print('SVC [ kernerl - rbf ]')
print('Confusion Matrix : \n', conf_mat)
Accuracy_score=metrics.accuracy_score(y_test, svc_prediction)
print('Accuracy Score : ', Accuracy_score)
print('Accuracy in Percentage : ', int(Accuracy_score*100),'%')
print(classification_report(svc_prediction,y_test))
```

```
SVC [ kernerl - rbf ]
Confusion Matrix :
 [[142  15]
 [ 38  36]]
Accuracy Score :  0.7705627705627706
Accuracy in Percentage :  77 %
              precision    recall  f1-score   support

           0       0.90      0.79      0.84       180
           1       0.49      0.71      0.58        51

    accuracy                           0.77       231
   macro avg       0.70      0.75      0.71       231
weighted avg       0.81      0.77      0.78       231
```

```
conf_mat=pd.crosstab(y_test, y_pred, rownames=['Actual'],
    colnames=['Predicted'])
sn.heatmap(conf_mat, annot=True).set(title='SVC [rbf]')
```



Fitting the Model (SVM) using 'Linear' kernel

```
model= SVC(kernel='linear',random_state=0)
model.fit(x_train, y_train)
svc_prediction=model.predict(x_test)
print('svc_prediction: ', svc_prediction)
```

```
svc_prediction:  [1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0
0 0 0 1 0
 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 0 0 1
 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0
 0 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0
 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0
 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 1 1 1 0 0 0 0 0]
```

Evaluation Metrics for 'Linear' kernel

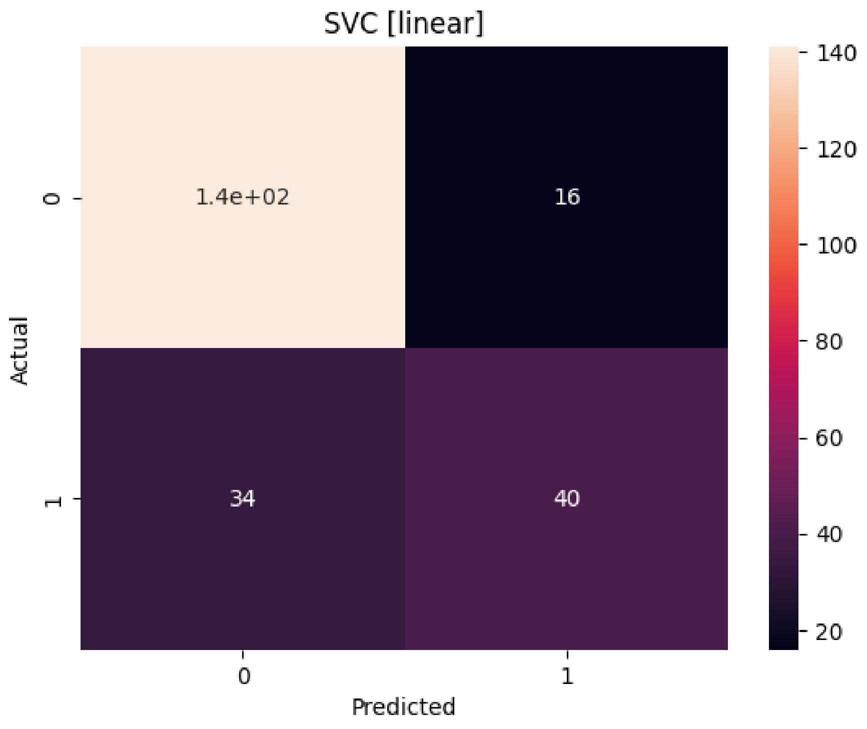
```
conf_mat=metrics.confusion_matrix(y_test, svc_prediction)
print('SVC [ kernerl - linear ]')
print('Confusion Matrix : \n', conf_mat)
Accuracy_score=metrics.accuracy_score(y_test, svc_prediction)
print('Accuracy Score : ', Accuracy_score)
print('Accuracy in Percentage : ', int(Accuracy_score*100),'%')
print(classification_report(svc_prediction,y_test))
```

```
SVC [ kernerl - linear ]
Confusion Matrix :
 [[141  16]
 [ 34  40]]
Accuracy Score :  0.7835497835497836
Accuracy in Percentage :  78 %
              precision    recall  f1-score   support

           0       0.90      0.81      0.85       175
           1       0.54      0.71      0.62        56

    accuracy                           0.78       231
   macro avg       0.72      0.76      0.73       231
weighted avg       0.81      0.78      0.79       231
```

```
conf_mat=pd.crosstab(y_test,  y_pred,  rownames=['Actual'],
     colnames=['Predicted'])
sn.heatmap(conf_mat, annot=True).set(title='SVC [linear]')
```


SVC [linear]
1.4e+02
Actual
Predicted