**S. B. JAIN INSTITUTE OF TECHNOLOGY,
MANAGEMENT & RESEARCH, NAGPUR.**

Practical No. 8

Aim: Apply the knowledge to study and implement the Support Vector Machine.

Name of Student : ______________________

Roll No. : ______________________

Semester/Year : ______________________

Academic Session : ______________________

Date of Performance : ______________________

Date of Submission : ______________________

OBJECTIVE/EXPECTED LEARNING OUTCOME:

The objectives and expected learning outcome of this practical are:

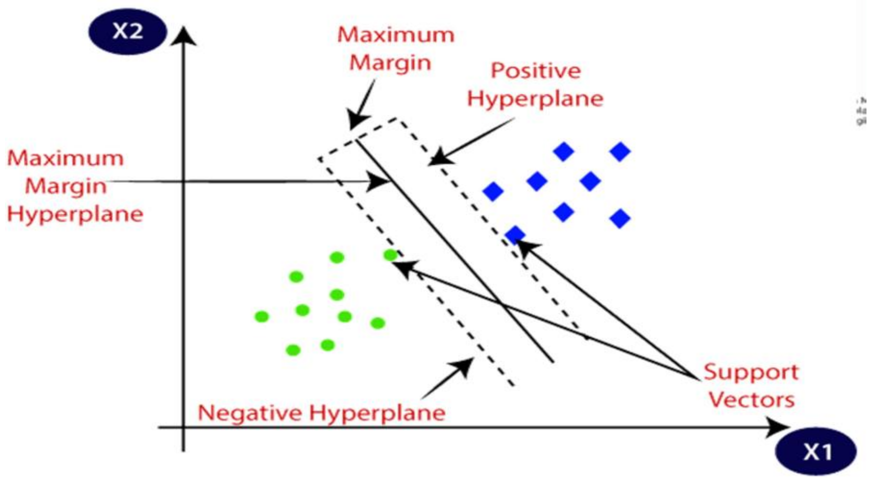
- The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.
- The dimension of the hyperplane depends upon the number of features.
- If the number of input features is two, then the hyperplane is just a line.
- If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

THEORY:

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:



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.

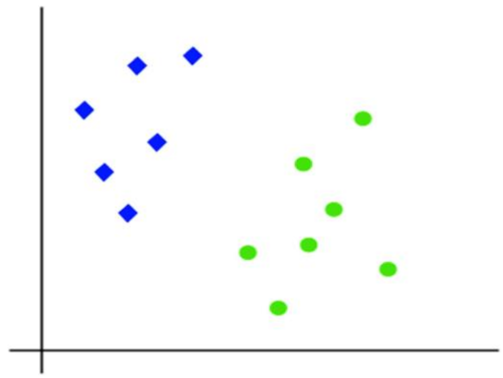
Support Vectors:

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.

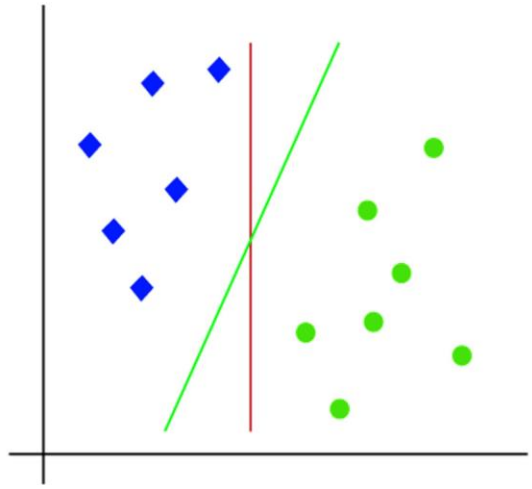
How does SVM works?

Linear SVM:

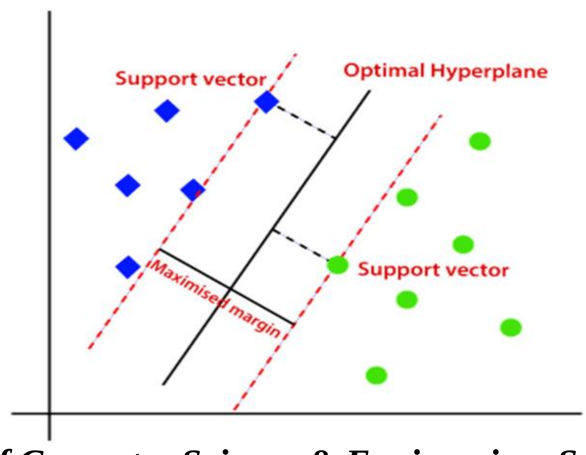
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



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:

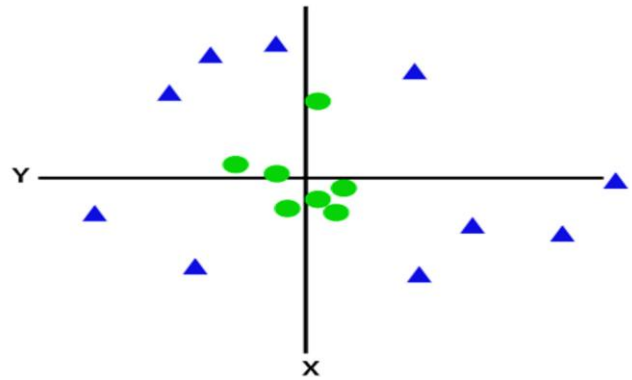


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane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

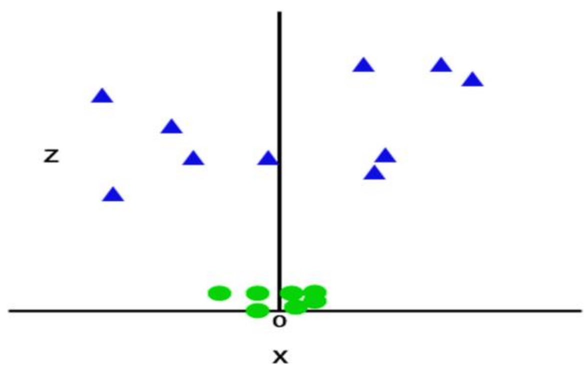
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



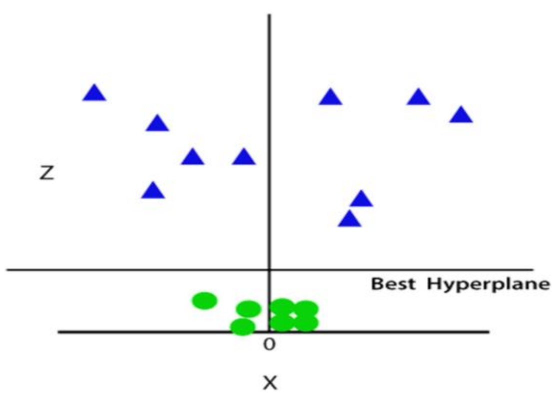
So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

$z=x^2+y^2$

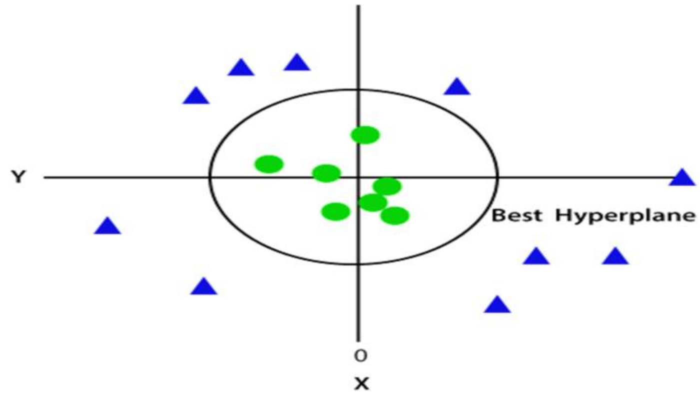
By adding the third dimension, the sample space will become as below image:



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.

Advantages of SVM

1. SVM works better when the data is Linear
2. It is more effective in high dimensions
3. With the help of the kernel trick, we can solve any complex problem
4. SVM is not sensitive to outliers
5. Can help us with Image classification

Disadvantages of SVM

1. Choosing a good kernel is not easy
2. It doesn't show good results on a big dataset
3. The SVM hyperparameters are Cost -C and gamma. It is not that easy to fine-tune these hyper-parameters. It is hard to visualize their impact.

PROGRAM CODE:

OUTPUT (SCREENSHOT):

Practical 8(SVM).ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
#Shrutika Pradeep Bagdi (CS22130)
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Shrutika Pradeep Bagdi (CS22130)
import warnings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
#Shrutika Pradeep Bagdi (CS22130)
path="/content/drive/MyDrive/Machine Learning (ML)/pulsar_stars.csv"
df=pd.read_csv(path)
df.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve
0	140.562500	55.683782	-0.234571	-0.699648	3.199833
1	102.507812	58.882430	0.465318	-0.515088	1.677258
2	103.015625	39.341649	0.323328	1.051164	3.121237
3	136.750000	57.178449	-0.068415	-0.636238	3.642977
4	88.726562	40.672225	0.600866	1.123492	1.178930

```
df
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile
0	140.562500	55.683782	-0.234571	-0.699648
1	102.507812	58.882430	0.465318	-0.515088
2	103.015625	39.341649	0.323328	1.051164
3	136.750000	57.178449	-0.068415	-0.636238
4	88.726562	40.672225	0.600866	1.123492
...	...	...	...	...
17893	136.429688	59.847421	-0.187846	-0.738123
17894	122.554688	49.485605	0.127978	0.323061
17895	119.335938	59.935939	0.159363	-0.743025
17896	114.507812	53.902400	0.201161	-0.024789
17897	57.062500	85.797340	1.406391	0.089520

17898 rows × 9 columns

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df.isnull().sum()
```

	0
Mean of the integrated profile	0
Standard deviation of the integrated profile	0
Excess kurtosis of the integrated profile	0
Skewness of the integrated profile	0
Mean of the DM-SNR curve	0
Standard deviation of the DM-SNR curve	0
Excess kurtosis of the DM-SNR curve	0
Skewness of the DM-SNR curve	0
target_class	0

dtype: int64

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df.shape
```

```
(17898, 9)
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17898 entries, 0 to 17897
Data columns (total 9 columns):
 #   Column                                          Non-Null Count  Dtype  
---  ------                                          --------------  -----  
 0    Mean of the integrated profile                 17898 non-null  float64
 1    Standard deviation of the integrated profile   17898 non-null  float64
 2    Excess kurtosis of the integrated profile      17898 non-null  float64
 3    Skewness of the integrated profile             17898 non-null  float64
 4    Mean of the DM-SNR curve                       17898 non-null  float64
 5    Standard deviation of the DM-SNR curve         17898 non-null  float64
 6    Excess kurtosis of the DM-SNR curve            17898 non-null  float64
 7    Skewness of the DM-SNR curve                   17898 non-null  float64
 8   target_class                                    17898 non-null  int64  
dtypes: float64(8), int64(1)
memory usage: 1.2 MB
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df.columns
```

```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    #Remove leading space
    df.columns = df.columns.str.strip()
```

```
#Shrutika Pradeep Bagdi (CS22130)
df.columns
```

```
Index(['Mean of the integrated profile',
       'Standard deviation of the integrated profile',
       'Excess kurtosis of the integrated profile',
       'Skewness of the integrated profile', 'Mean of the DM-SNR curve',
       'Standard deviation of the DM-SNR curve',
       'Excess kurtosis of the DM-SNR curve', 'Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness', 'DM-SNR Mean', 'DM-SNR Sd',
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df.columns
```

```
Index(['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness', 'DM-SNR Mean',
       'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class'],
      dtype='object')
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    df['target_class'].value_counts()
```

target_class	count
0	16259
1	1639

dtype: int64

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    #view the percenage distribution of targrt_class column
    df['target_class'].value_counts()/np.float64(len(df))
```

target_class	count
0	0.908426
1	0.091574

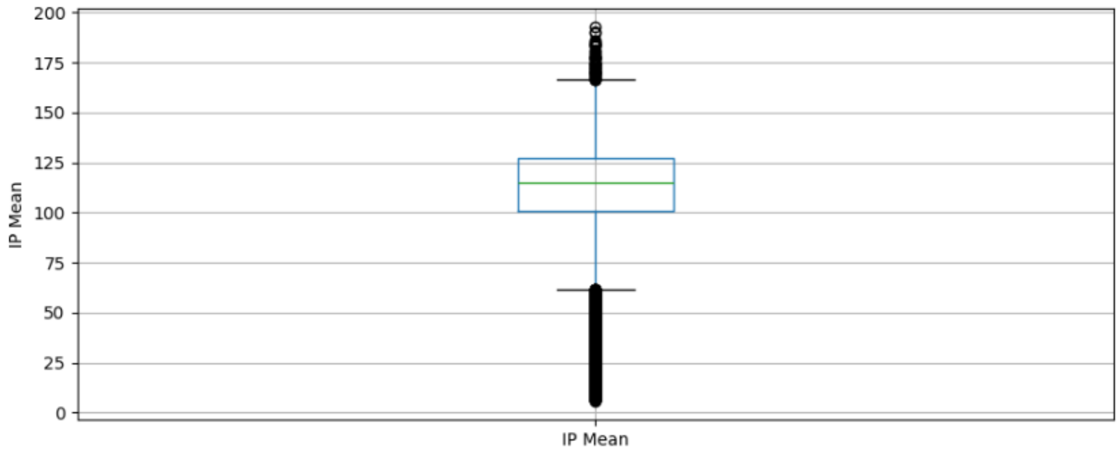
dtype: float64

```
#Shrutika Pradeep Bagdi (CS22130)
# view summary statistics in numerical variables
round(df.describe(),2)
```

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness	target_class
count	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00	17898.00
mean	111.08	46.55	0.48	1.77	12.61	26.33	8.30	104.86	0.09
std	25.65	6.84	1.06	6.17	29.47	19.47	4.51	106.51	0.29
min	5.81	24.77	-1.88	-1.79	0.21	7.37	-3.14	-1.98	0.00
25%	100.93	42.38	0.03	-0.19	1.92	14.44	5.78	34.96	0.00
50%	115.08	46.95	0.22	0.20	2.80	18.46	8.43	83.06	0.00
75%	127.09	51.02	0.47	0.93	5.46	28.43	10.70	139.31	0.00

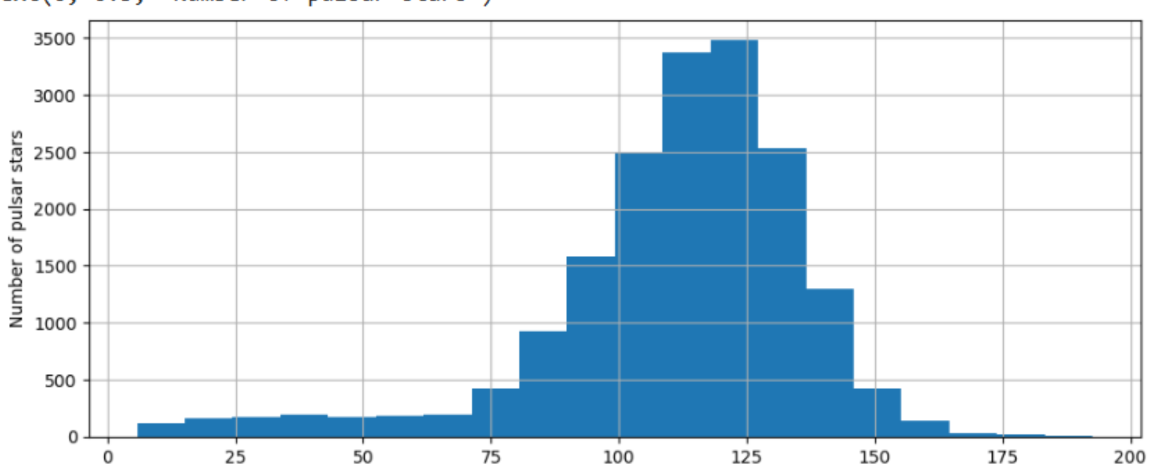
```
#Shrutika Pradeep Bagdi (CS22130)
plt.figure(figsize=(24,20))

plt.subplot(4, 2, 1)
fig = df.boxplot(column='IP Mean')
fig.set_title('')
fig.set_ylabel('IP Mean')

plt.subplot(4, 2, 2)
fig = df.boxplot(column='IP Sd')
fig.set_title('')
fig.set_ylabel('IP Sd')
```

```
Text(0, 0.5, 'DM-SNR Skewness')
```




```
#Shrutika Pradeep Bagdi (CS22130)
# plot histogrM TO CHECK DISTRIBUTION

plt.figure(figsize=(24,20))

plt.subplot(4, 2, 1)
fig = df['IP Mean'].hist(bins=20)
fig.set_xlabel('IP Mean')
fig.set_ylabel('Number of pulsar stars')

plt.subplot(4, 2, 2)
fig = df['IP Sd'].hist(bins=20)
fig.set_xlabel('IP Sd')
fig.set_ylabel('Number of pulsar stars')
```

```
Text(0, 0.5, 'Number of pulsar stars')
```



```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    X = df.drop(['target_class'], axis=1)
    y = df['target_class']
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    X_train.shape, X_test.shape
```

```
((14318, 8), (3580, 8))
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    cols = X_train.columns
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    from sklearn.preprocessing import StandardScaler
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```
#Shrutika Pradeep Bagdi (CS22130)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    X_train.describe()
```

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	DM-SNR Kurtosis	DM-SNR Skewness
count	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04	1.431800e+04
mean	1.908113e-16	-6.550610e-16	1.042143e-17	3.870815e-17	-8.734147e-17	-1.617802e-16	-1.513588e-17	1.122785e-16
std	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00	1.000035e+00
min	-4.035499e+00	-3.181033e+00	-2.185946e+00	-5.744051e-01	-4.239001e-01	-9.733707e-01	-2.455649e+00	-1.003411e+00
25%	-3.896291e-01	-6.069473e-01	-4.256221e-01	-3.188054e-01	-3.664918e-01	-6.125457e-01	-5.641035e-01	-6.627590e-01
50%	1.587461e-01	5.846646e-02	-2.453172e-01	-2.578142e-01	-3.372294e-01	-4.067482e-01	3.170446e-02	-2.059136e-01
75%	6.267059e-01	6.501017e-01	-1.001238e-02	-1.419621e-01	-2.463724e-01	1.078934e-01	5.362759e-01	3.256217e-01
max	3.151882e+00	7.621116e+00	7.008906e+00	1.054430e+01	7.025568e+00	4.292181e+00	5.818557e+00	1.024613e+01

```
#Shrutika Pradeep Bagdi (CS22130)
#Default
#import SVC classifier
from sklearn.svm import SVC
#import metrics to compute accuracy
from sklearn.metrics import accuracy_score
#instantiate classifier with default hyperparameters
svc = SVC()
#fit classifier to training set
svc.fit(X_train, y_train)
#make predictions on test set
y_pred = svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with default hyperparameters: 0.9827
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    #instantiate classifier with rbf kernal and C=1000
    svc = SVC(C=10000.0)
    #fit classifier to training set
    svc.fit(X_train, y_train)
    #make predictions on test set
    y_pred = svc.predict(X_test)
    #compute and print accuracy score
    print('Model accuracy score with rbf kernal and C=1000.0: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernal and C=1000.0: 0.9796
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    #instantiate classifier with rbf kernal and C=100
    svc = SVC(C=100.0)
    #fit classifier to training set
    svc.fit(X_train, y_train)
    #make predictions on test set
    y_pred = svc.predict(X_test)
    #compute and print accuracy score
    print('Model accuracy score with rbf kernal and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with rbf kernal and C=100.0 : 0.9832
```

Run SVM with linear Kernal

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    ### RUN SVM with linear kernal
    # instantiate classifier with linear kernal and C=1.0
    linear_svc=SVC(kernel='linear', C=1.0)
    # fit classifier to training set
    linear_svc.fit(X_train,y_train)
    # make predictions on test set
    y_pred_test=linear_svc.predict(X_test)
    #compute and print accuracy score
    print('Model accuracy score with linear kernal and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernal and C=1.0 : 0.9832
```

```
#Shrutika Pradeep Bagdi (CS22130)
### RUN SVM with linear kernal
# instantiate classifier with linear kernal and C=1.0
linear_svc=SVC(kernel='linear', C=100.0)
# fit classifier to training set
linear_svc.fit(X_train,y_train)
# make predictions on test set
y_pred_test=linear_svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with linear kernal and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernal and C=100.0 : 0.9832
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    ### RUN SVM with linear kernal
    # instantiate classifier with linear kernal and C=1.0
    linear_svc=SVC(kernel='linear', C=1000.0)
    # fit classifier to training set
    linear_svc.fit(X_train,y_train)
    # make predictions on test set
    y_pred_test=linear_svc.predict(X_test)
    #compute and print accuracy score
    print('Model accuracy score with linear kernal and C=1.0000 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernal and C=1.0000 : 0.9832
```

```
#Shrutika Pradeep Bagdi (CS22130)
#CHEACK FOR OVERFITTINF AND UNDERFITTING

#print the score on training and test set
print ('Training set score: {:.4f}'.format(linear_svc.score(X_train, y_train)))
print ('Test set score: {:.4f}'.format(linear_svc.score(X_test, y_test)))
```

```
Training set score: 0.9785
Test set score: 0.9832
```

RUN SVM WITH PLOYNOMIAL KERNAL

[ ]
```
#Shrutika Pradeep Bagdi (CS22130)
###RUN SVM WITH PLOYNOMIAL KERNAL

#instantiate classifier with polynomial kranal and C=1.0
poly_svc=SVC(kernel='poly', C=1.0)
# fit classifier to training set
poly_svc.fit(X_train,y_train)
#make pradictions on test set
y_pred=poly_svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with polynomial kernal and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernal and C=1.0 : 0.9807
```

```
#Shrutika Pradeep Bagdi (CS22130)
###RUN SVM WITH PLOYNOMIAL KERNAL

#instantiate classifier with polynomial kranal and C=100.0
poly_svc=SVC(kernel='poly', C=100.0)
# fit classifier to training set
poly_svc.fit(X_train,y_train)
#make pradictions on test set
y_pred=poly_svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with polynomial kernal and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernal and C=100.0 : 0.9824
```

[ ]
```
#Shrutika Pradeep Bagdi (CS22130)
###RUN SVM WITH PLOYNOMIAL KERNAL

#instantiate classifier with polynomial kranal and C=1000.0
poly_svc=SVC(kernel='poly', C=1000.0)
# fit classifier to training set
poly_svc.fit(X_train,y_train)
#make pradictions on test set
y_pred=poly_svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with polynomial kernal and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernal and C=1000.0 : 0.9838
```

RUN SVM WITH SIGMIOD KERNAL

```
#Shrutika Pradeep Bagdi (CS22130)
### RUN SVM WITH SIGMIOD KERNAL

#instantiate classifier with sigmoid kernal and C=1.0
sigmoid_svc=SVC(kernel='sigmoid',C=1.0)
# fit classifier to training set
sigmoid_svc.fit(X_train,y_train)
#make pradictions on test set
y_pred=sigmoid_svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with polynomial kernal and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernal and C=1.0 : 0.8858
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    ### RUN SVM WITH SIGMIOD KERNAL

    #instantiate classifier with sigmoid kernal and C=100.0
    sigmoid_svc=SVC(kernel='sigmoid',C=100.0)
    # fit classifier to training set
    sigmoid_svc.fit(X_train,y_train)
    #make pradictions on test set
    y_pred=sigmoid_svc.predict(X_test)
    #compute and print accuracy score
    print('Model accuracy score with polynomial kernal and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernal and C=100.0 : 0.8855
```

```
#Shrutika Pradeep Bagdi (CS22130)
### RUN SVM WITH SIGMIOD KERNAL

#instantiate classifier with sigmoid kernal and C=1000.0
sigmoid_svc=SVC(kernel='sigmoid',C=1000.0)
# fit classifier to training set
sigmoid_svc.fit(X_train,y_train)
#make pradictions on test set
y_pred=sigmoid_svc.predict(X_test)
#compute and print accuracy score
print('Model accuracy score with polynomial kernal and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with polynomial kernal and C=1000.0 : 0.8855
```

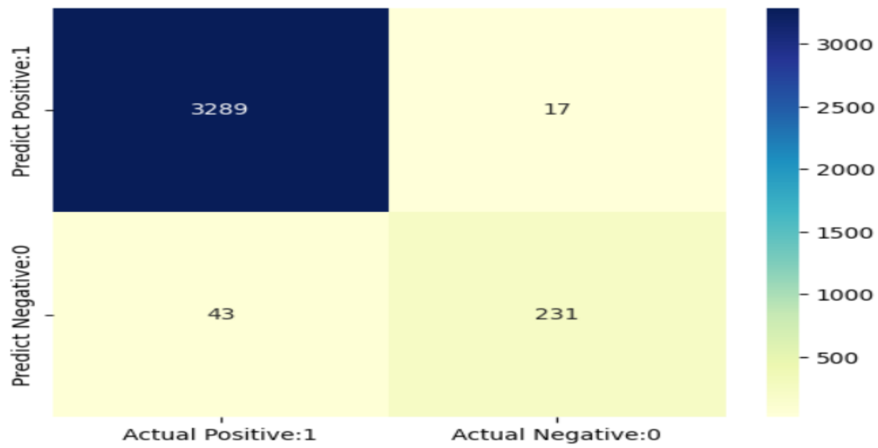
```
#Shrutika Pradeep Bagdi (CS22130)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Neagatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[3289   17]
 [  43  231]]

True Positives(TP) =  3289

True Negatives(TN) =  231

False Positives(FP) =  17

False Neagatives(FN) =  43
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    import seaborn as sns
```

```
#Shrutika Pradeep Bagdi (CS22130)
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                         index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```



```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    from sklearn.metrics import classification_report
    print(classification_report(y_test, y_pred_test))

                  precision    recall  f1-score   support

               0       0.99      0.99      0.99      3306
               1       0.93      0.84      0.89       274

        accuracy                           0.98      3580
       macro avg       0.96      0.92      0.94      3580
    weighted avg       0.98      0.98      0.98      3580
```

```
#Shrutika Pradeep Bagdi (CS22130)
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

```
[ ] #Shrutika Pradeep Bagdi (CS22130)
    classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
    print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

    Classification accuracy : 0.9832
```

Classification error

```
#Shrutika Pradeep Bagdi (CS22130)
classification_error = (FP + FN) / float(TP + TN + FP + FN)
print('Classification error : {0:0.4f}'.format(classification_error))

Classification error : 0.0168
```

CONCLUSION:

_ _

_ _

_ _

_ _

_ _

DISCUSSION AND VIVA VOCE:

Q1. What are Support Vector Machines (SVMs)?

Q2. What is the basic principle of a Support Vector Machine?

Q3. What are Support Vectors in SVMs?

Q4. What do you mean by Hinge loss?

Q5. What is the role of the C hyper-parameter in SVM? Does it affect the bias/variance trade-off?

Q6. Explain different types of kernel functions.

REFERENCE

- https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/
- https://www.geeksforgeeks.org/support-vector-machine-algorithm/

https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm