8) SVM

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

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

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 df=pd.read_csv('/content/drive/MyDrive/HCP/kyphosis.csv')
6 df
7 # df.tail()
8 # df.head()
```

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15
...	...	...	...	...
76	present	157	3	13
77	absent	26	7	13
78	absent	120	2	13
79	present	42	7	6
80	absent	36	4	13

81 rows × 4 columns

```
1 from sklearn.model_selection import train_test_split
2
3 X=df.drop('Kyphosis',axis=1)
4 y=df['Kyphosis']
5
5 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
6
1  #   X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_st
```

Import Support vector classifier

```
from sklearn.svm import SVC

model=SVC()
model.fit(X_train,y_train)

#--C=controls the cost of misclassification of the data.Large 'c' gives low
#--(Low bias because the cost of misclassification is penalized),With a smal

#---'gamma' (small value of gamma that means gaussian with a large variance
#---large gamma leads to high bias and low variance(implying that support ve
#---'Kernel' is the type of kernel that you want to use in the classifier, d
```

```
SVC()
```

----Arguments----

SVC(C=1.0 (- Regularization parameter),
kernel='rbf'(---'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable),
degree=3 (---Degree of the polynomial kernel function ('poly')),
gamma='scale' (---{'scale', 'auto'}),
coef0=0.0 (--- Independent term in kernel function. Significant when 'poly' and 'sigmoid' is used
shrinking=True (---doubt-shriking heuristic),
probability=False(---Whether to enable probability estimates),
tol=0.001 (---Tolerance for stopping criterion),
cache_size=200 (---Specify the size of the kernel cache (in MB)),
class_weight=None(---{dict, 'balanced'},Set the parameter C of class i to class_weight[i]*C for S
verbose=False,
max_iter=-1 (--- Hard limit on iterations within solver, or -1 for no limit),
decision_function_shape='ovr'(---'ovo', 'ovr', default='ovr'.If decision_function_shape=’ovo’, th
break_ties=False,
random_state=None,
)

```
#---Predictions of test set
predictions=model.predict(X_test)
```

```
#---Printing Confusion Matrix
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
  print(classification_report(y_test,predictions))
```

```
[[18  0]
 [ 7  0]]
              precision    recall  f1-score   support

      absent       0.72      1.00      0.84        18
```

```
     present       0.00      0.00      0.00         7

    accuracy                           0.72        25
   macro avg       0.36      0.50      0.42        25
weighted avg       0.52      0.72      0.60        25

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1318: Unde
  _warn_prf(average, modifier, msg_start, len(result))
```

Grid Search (we can adjust 'c' and 'gamma' parameters using Grid Search)

```
from sklearn.model_selection import GridSearchCV

#---Dictionary  for  parameters
   param_grid={'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001]}
grid=GridSearchCV(SVC(),param_grid,verbose=4)
#---high value of verbose will give us high text description while running t
grid.fit(X_train,y_train)
```

```
[CV 1/5] END ..................C=1000, gamma=1;, score=0.833 total time=   0.0s
[CV 2/5] END ..................C=1000, gamma=1;, score=0.818 total time=   0.0s
[CV 3/5] END ..................C=1000, gamma=1;, score=0.818 total time=   0.0s
[CV 4/5] END ..................C=1000, gamma=1;, score=0.818 total time=   0.0s
[CV 5/5] END ..................C=1000, gamma=1;, score=0.818 total time=   0.0s
[CV 1/5] END ................C=1000, gamma=0.1;, score=0.833 total time=   0.0s
[CV 2/5] END ................C=1000, gamma=0.1;, score=0.727 total time=   0.0s
[CV 3/5] END ................C=1000, gamma=0.1;, score=0.727 total time=   0.0s
[CV 4/5] END ................C=1000, gamma=0.1;, score=0.727 total time=   0.0s
[CV 5/5] END ................C=1000, gamma=0.1;, score=0.818 total time=   0.0s
[CV 1/5] END ...............C=1000, gamma=0.01;, score=0.833 total time=   0.0s
[CV 2/5] END ...............C=1000, gamma=0.01;, score=0.727 total time=   0.0s
[CV 3/5] END ...............C=1000, gamma=0.01;, score=0.636 total time=   0.0s
[CV 4/5] END ...............C=1000, gamma=0.01;, score=0.727 total time=   0.0s
[CV 5/5] END ...............C=1000, gamma=0.01;, score=0.818 total time=   0.0s
[CV 1/5] END ..............C=1000, gamma=0.001;, score=0.750 total time=   0.0s
[CV 2/5] END ..............C=1000, gamma=0.001;, score=0.818 total time=   0.0s
[CV 3/5] END ..............C=1000, gamma=0.001;, score=0.818 total time=   0.0s
[CV 4/5] END ..............C=1000, gamma=0.001;, score=0.636 total time=   0.0s
[CV 5/5] END ..............C=1000, gamma=0.001;, score=0.727 total time=   0.0s
[CV 1/5] END .............C=1000, gamma=0.0001;, score=0.917 total time=   0.0s
[CV 2/5] END .............C=1000, gamma=0.0001;, score=0.818 total time=   0.0s
```

```
[CV 3/5] END .............C=1000, gamma=0.0001;, score=0.727 total time=   0.0s
[CV 4/5] END .............C=1000, gamma=0.0001;, score=0.636 total time=   0.0s
[CV 5/5] END .............C=1000, gamma=0.0001;, score=0.818 total time=   0.0s
GridSearchCV(estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001]},
             verbose=4)
```

```
grid.best_params_

#---returns the parameters combination which has best cross validation score
```

```
{'C': 0.1, 'gamma': 1}
```

```
grid.best_estimator_
#---fetches the best estimator
```

```
SVC(C=0.1, gamma=1)
```

```
grid_predictions=grid.predict(X_test)
print(confusion_matrix(y_test,grid_predictions))
print(classification_report(y_test,grid_predictions))

#--- After the grid search, we can see that model performance has increased.
```

```
[[18  0]
 [ 7  0]]
              precision    recall  f1-score   support

      absent       0.72      1.00      0.84        18
     present       0.00      0.00      0.00         7

    accuracy                           0.72        25
```