Name: Heramb Pawar

Roll No:67

ML Practical 3 Output

Import Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Import Dataset

```
data = 'pulsar_data_train.csv'

df = pd.read_csv(data)
```

Exploratory data analysis

```
df.shape
```

```
(12528, 9)
```

```
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	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve
0	121.156250	48.372971	0.375485	-0.013165	3.168896	18.399367	7.449874	65.159298
1	76.968750	36.175557	0.712898	3.388719	2.399666	17.570997	9.414652	102.722975
2	130.585938	53.229534	0.133408	-0.297242	2.743311	22.362553	8.508364	74.031324
3	156.398438	48.865942	-0.215989	-0.171294	17.471572	NaN	2.958066	7.197842
4	84.804688	36.117659	0.825013	3.274125	2.790134	20.618009	8.405008	76.291128

```
col_names = df.columns

col_names
```

```
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')
```

```
df.columns = df.columns.str.strip()

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')
```

```
df.columns = ['IP Mean', 'IP Sd', 'IP Kurtosis', 'IP Skewness',
              'DM-SNR Mean', 'DM-SNR Sd', 'DM-SNR Kurtosis', 'DM-SNR Skewness', 'target_class']
```

```
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')
```

```
df['target_class'].value_counts()
```

```
0.0    11375
1.0     1153
Name: target_class, dtype: int64
```

```
df['target_class'].value_counts()/np.float(len(df))
```

```
0.0    0.907966
1.0    0.092034
Name: target_class, dtype: float64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12528 entries, 0 to 12527
Data columns (total 9 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   IP Mean          12528 non-null  float64
 1   IP Sd            12528 non-null  float64
 2   IP Kurtosis      10793 non-null  float64
 3   IP Skewness      12528 non-null  float64
 4   DM-SNR Mean      12528 non-null  float64
 5   DM-SNR Sd        11350 non-null  float64
 6   DM-SNR Kurtosis  12528 non-null  float64
 7   DM-SNR Skewness  11903 non-null  float64
 8   target_class     12528 non-null  float64
dtypes: float64(9)
memory usage: 881.0 KB
```

```
df.isnull().sum()
```

```
IP Mean               0
IP Sd                 0
IP Kurtosis        1735
IP Skewness           0
DM-SNR Mean           0
DM-SNR Sd          1178
DM-SNR Kurtosis       0
DM-SNR Skewness     625
target_class          0
dtype: int64
```

```
df = df.fillna(0)
```

```
df.isnull().sum()
```

```
IP Mean            0
IP Sd              0
IP Kurtosis        0
IP Skewness        0
DM-SNR Mean        0
DM-SNR Sd          0
DM-SNR Kurtosis    0
DM-SNR Skewness    0
target_class       0
dtype: int64
```

```
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_c
count	12528.00	12528.00	12528.00	12528.00	12528.00	12528.00	12528.00	12528.00	125
mean	111.04	46.52	0.41	1.78	12.67	23.87	8.33	100.26	
std	25.67	6.80	1.00	6.21	29.61	20.19	4.54	107.18	
min	5.81	24.77	-1.74	-1.79	0.21	0.00	-3.14	-1.98	
25%	100.87	42.36	0.00	-0.19	1.91	13.27	5.80	26.39	
50%	115.18	46.93	0.16	0.20	2.79	17.41	8.45	78.43	
75%	127.11	50.98	0.42	0.93	5.41	26.47	10.73	135.77	
max	189.73	91.81	8.07	68.10	222.42	110.64	34.54	1191.00	

Declare feature vector and target variable

```
X = df.drop(['target_class'], axis=1)
```

```
y = df['target_class']
```

Split data into separate 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 = 0.2, random_state = 0)
```

```
X_train.shape, X_test.shape
```

```
((10022, 8), (2506, 8))
```

Feature Scaling

```
cols = X_train.columns
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns=[cols])
```

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

```
X_train.describe()
```

	IP Mean	IP Sd	IP Kurtosis	IP Skewness	DM-SNR Mean	DM-SNR Sd	D K
count	1.002200e+04	1.002200e+04	1.002200e+04	1.002200e+04	1.002200e+04	1.002200e+04	
std	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	

Run SVM with default hyperparameters

```
# 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_te
```

```
Model accuracy score with default hyperparameters: 0.9785
```

Run SVM with rbf kernel and C=**100.0**

```
# instantiate classifier with rbf kernel 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 kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_te
```

```
Model accuracy score with rbf kernel and C=100.0 : 0.9792
```

Run SVM with rbf kernel and C=**1000.0**

```
# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)
```

```
# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.9765
```

Run SVM with linear kernel

Run SVM with linear kernel and C=**1.0**

```
# instantiate classifier with linear kernel 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 kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_t
```

```
Model accuracy score with linear kernel and C=1.0 : 0.9765
```

Run SVM with linear kernel and C=**100.0**

```
# instantiate classifier with linear kernel and C=100.0
linear_svc100=SVC(kernel='linear', C=100.0)


# fit classifier to training set
linear_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_
```

```
Model accuracy score with linear kernel and C=100.0 : 0.9765
```

Run SVM with linear kernel and C=**1000.0**

```
# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)
```

```
# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.9765
```

Compare the train-set and test-set accuracy

```
y_pred_train = linear_svc.predict(X_train)

y_pred_train
```

```
array([0., 0., 0., ..., 1., 0., 0.])
```

```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.9802
```

Check for overfitting and underfitting

```
# print the scores 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: 0.9751
Test set score: 0.9765
```

Compare model accuracy with null accuracy

```
# check class distribution in test set

y_test.value_counts()
```

```
0.0    2285
1.0     221
Name: target_class, dtype: int64
```

```
# check null accuracy score

null_accuracy = (3306/(3306+274))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.9235
```

Run SVM with polynomial kernel

Run SVM with polynomial kernel and C=**1.0**

```
# instantiate classifier with polynomial kernel 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 predictions on test set
y_pred=poly_svc.predict(X_test)
```

```
# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. format(accuracy_score
```

```
Model accuracy score with polynomial kernel and C=1.0 : 0.9749
```

Run SVM with polynomial kernel and C=**100.0**

```
# instantiate classifier with polynomial kernel and C=100.0
poly_svc100=SVC(kernel='poly', C=100.0)


# fit classifier to training set
poly_svc100.fit(X_train, y_train)


# make predictions on test set
y_pred=poly_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. format(accuracy_score
```

```
Model accuracy score with polynomial kernel and C=1.0 : 0.9789
```

Run SVM with sigmoid kernel

Run SVM with sigmoid kernel and C=**1.0**

```
# instantiate classifier with sigmoid kernel 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 predictions on test set
y_pred=sigmoid_svc.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_
```

```
Model accuracy score with sigmoid kernel and C=1.0 : 0.8787
```

Run SVM with sigmoid kernel and C=**100.0**

```
# instantiate classifier with sigmoid kernel and C=100.0
sigmoid_svc100=SVC(kernel='sigmoid', C=100.0)


# fit classifier to training set
sigmoid_svc100.fit(X_train,y_train)


# make predictions on test set
y_pred=sigmoid_svc100.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with sigmoid kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(
```

```
Model accuracy score with sigmoid kernel and C=100.0 : 0.8783
```

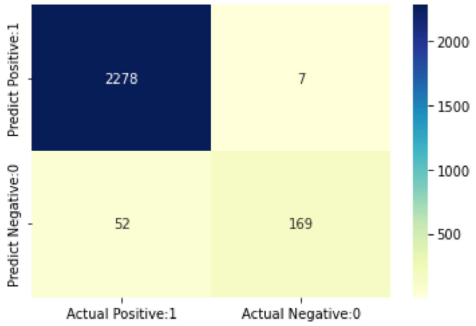
Confusion matrix

```
# Print the Confusion Matrix and slice it into four pieces

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 Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[2278    7]
 [  52  169]]

True Positives(TP) =  2278

True Negatives(TN) =  169

False Positives(FP) =  7

False Negatives(FN) =  52
```

```
# visualize confusion matrix with seaborn heatmap

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')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fefe91bbdf0>
```



Classification metrices

Classification Report

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99      2285
         1.0       0.96      0.76      0.85       221

    accuracy                           0.98      2506
```

```
   macro avg       0.97      0.88      0.92      2506
weighted avg       0.98      0.98      0.98      2506
```

Classification accuracy

```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]

# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.9765
```

Classification error

```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.0235
```

Precision

```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9969
```

Recall

```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.9777
```

True Positive Rate

```
true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

```
True Positive Rate : 0.9777
```

False Positive Rate

```
false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.0398
```

Specificity

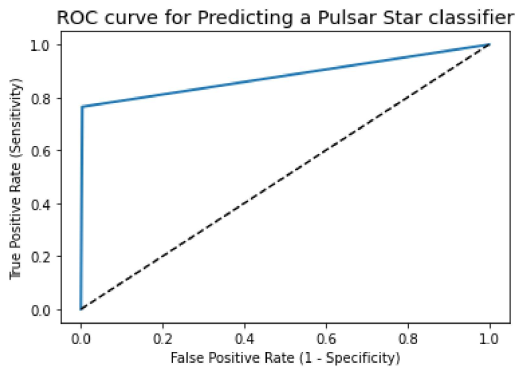
```
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.9602
```

ROC - AUC

ROC Curve

```
# plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Predicting a Pulsar Star classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



ROC AUC

```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred_test)
```

```
print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.8808
```

```
# calculate cross-validated ROC AUC

from sklearn.model_selection import cross_val_score

Cross_validated_ROC_AUC = cross_val_score(linear_svc1000, X_train, y_train, cv=10, scoring='roc_

print('Cross validated ROC AUC : {:.4f}'.format(Cross_validated_ROC_AUC))
```

```
Cross validated ROC AUC : 0.9622
```

Stratified k-fold Cross Validation with shuffle split

```
from sklearn.model_selection import KFold


kfold=KFold(n_splits=5, shuffle=True, random_state=0)


linear_svc=SVC(kernel='linear')


linear_scores = cross_val_score(linear_svc, X, y, cv=kfold)

# print cross-validation scores with linear kernel

print('Stratified cross-validation scores with linear kernel:\n\n{}'.format(linear_scores))
```

```
Stratified cross-validation scores with linear kernel:

[0.9764565  0.97765363 0.97486034 0.97365269 0.9744511 ]
```

```
# print average cross-validation score with linear kernel

print('Average stratified cross-validation score with linear kernel:{:.4f}'.format(linear_score
```

```
Average stratified cross-validation score with linear kernel:0.9754
```

Stratified k-Fold Cross Validation with shuffle split with rbf kernel

```
rbf_svc=SVC(kernel='rbf')


rbf_scores = cross_val_score(rbf_svc, X, y, cv=kfold)

# print cross-validation scores with rbf kernel

print('Stratified Cross-validation scores with rbf kernel:\n\n{}'.format(rbf_scores))
```

```
Stratified Cross-validation scores with rbf kernel:

[0.97206704 0.97286512 0.96847566 0.97285429 0.96846307]
```

```
# print average cross-validation score with rbf kernel

print('Average stratified cross-validation score with rbf kernel:{:.4f}'.format(rbf_scores.mean
```

```
Average stratified cross-validation score with rbf kernel:0.9709
```

```
# **Hyperparameter·Optimization·using·GridSearch·CV**
```

Hyperparameter Optimization using GridSearch CV

```
# import GridSearchCV
from sklearn.model_selection import GridSearchCV


# import SVC classifier
from sklearn.svm import SVC


# instantiate classifier with default hyperparameters with kernel=rbf, C=1.0 and gamma=auto
svc=SVC()



# declare parameters for hyperparameter tuning
parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
               {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6
               {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4] ,'gamma':[0.01,0.0
              ]




grid_search = GridSearchCV(estimator = svc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)


grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                         {'C': [1, 10, 100, 1000],
                          'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
                                    0.9],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'degree': [2, 3, 4],
                          'gamma': [0.01, 0.02, 0.03, 0.04, 0.05],
                          'kernel': ['poly']}],
             scoring='accuracy')
```

```
# examine the best model


# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))


# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params_))


# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.9785


Parameters that give the best results :

 {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

```
Estimator that was chosen by the search :

 SVC(C=10, gamma=0.1)
```

```
# calculate GridSearch CV score on test set

print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
GridSearch CV score on test set: 0.9796
```

0s completed at 01:12