Assignment 2

Name : Harleen Kaur Taunque Student Id : 20811951

Name : Somesh Kumar Gupta Student Id : 20817245

In [1]:
```
# Import Packages
import pandas as pd
import numpy as np
from scipy.io import loadmat
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
from sklearn.metrics import classification_report
```

Question 1

In [2]:
```
# Load the Dataset DataA.mat
mat_dataA = loadmat('Dataset/DataA.mat')
```

In [3]:
```
# DataA.mat
print("Dataset A: ", mat_dataA)
```

```
Dataset A:  {'__header__': b'MATLAB 5.0 MAT-file, Platform: PCWIN64, Created on: Tue Oct 21 15:46:48 2014', '__v
ersion__': '1.0', '__globals__': [], 'fea': array([[1, 2, 1, ..., 2, 2, 1],
       [3, 3, 4, ..., 1, 3, 4],
       [4, 1, 4, ..., 2, 4, 4],
       ...,
       [3, 4, 4, ..., 3, 1, 2],
       [1, 4, 3, ..., 4, 3, 4],
       [4, 1, 3, ..., 2, 4, 3]], dtype=uint8), 'gnd': array([[1],
       [1],
       [1],
       ...,
       [1],
       [1],
       [1]], dtype=int16)}
```

In [4]:
```
# Get the Features from the Matrix
mat_features_dataA = mat_dataA['fea']
```

In [5]:
```
# Get the Labels from the Matrix
mat_label_dataA = mat_dataA['gnd']
```

In [6]:
```
# Convert the Feature Matrix data into Pandas Dataframe
df_features_dataA = pd.DataFrame(mat_features_dataA)
print("Features from the Dataset: \n", df_features_dataA)
```

```
Features from the Dataset:
       0  1  2  3  4  5  6  7  8  9  ...  47  48  49  50  51  52  53  \
0      1  2  1  2  1  2  3  3  3  3  ...   4   3   2   3   4   2   2
1      3  3  4  2  1  2  2  4  3  3  ...   3   1   3   4   3   4   4
2      4  1  4  4  4  4  1  1  2  1  ...   1   1   2   1   1   4   2
3      1  4  1  1  3  3  1  3  4  1  ...   2   1   3   3   4   1   3
4      4  3  1  1  4  1  4  4  1  4  ...   4   3   1   3   2   1   4
...   .. .. .. .. .. .. .. .. .. ..  ...  ..  ..  ..  ..  ..  ..  ..
2195   3  3  3  4  2  2  3  3  4  4  ...   2   1   4   3   3   4   1
2196   3  3  1  1  2  2  3  4  2  2  ...   4   2   1   3   1   2   3
2197   3  4  4  3  3  4  1  3  4  3  ...   2   4   4   3   4   1   2
2198   1  4  3  3  4  1  3  4  2  4  ...   4   2   4   2   2   2   4
2199   4  3  1  2  1  2  3  3  3  3  ...   3   4   3   3   4   3   4

      54  55  56
0      2   2   1
1      1   3   4
2      2   4   4
3      3   4   2
4      2   1   1
...   ..  ..  ..
2195   2   1   3
2196   4   2   3
2197   3   1   2
2198   1   3   4
2199   2   4   3

[2200 rows x 57 columns]
```

In [7]:
```
# Statistical Description of the Dataset
print(df_features_dataA.describe())
```

```
                 0            1            2            3            4  \
count  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000
mean      2.493182     2.529545     2.506818     2.526818     2.547727
std       1.103943     1.086542     1.098160     1.118780     1.081699
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%       1.000000     2.000000     1.000000     2.000000     2.000000
50%       3.000000     3.000000     3.000000     3.000000     3.000000
75%       3.000000     4.000000     3.000000     4.000000     3.000000
max       4.000000     4.000000     4.000000     4.000000     4.000000

                 5            6            7            8           9  ...  \
count  2200.000000  2200.000000  2200.000000  2200.000000  2200.00000  ...
mean      2.492727     2.550000     2.541364     2.553182     2.56500  ...
std       1.099399     1.109818     1.103600     1.126751     1.10822  ...
min       1.000000     1.000000     1.000000     1.000000     1.00000  ...
25%       2.000000     2.000000     2.000000     2.000000     2.00000  ...
50%       3.000000     3.000000     3.000000     3.000000     3.00000  ...
75%       3.000000     4.000000     3.000000     4.000000     4.00000  ...
max       4.000000     4.000000     4.000000     4.000000     4.00000  ...

                47           48           49           50           51  \
count  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000
mean      2.475000     2.498636     2.497273     2.439091     2.495909
std       1.070635     1.104786     1.070499     1.093583     1.087354
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%       2.000000     1.000000     2.000000     1.000000     1.000000
50%       2.000000     3.000000     2.000000     2.000000     2.000000
75%       3.000000     3.000000     3.000000     3.000000     3.000000
max       4.000000     4.000000     4.000000     4.000000     4.000000

                52           53           54           55           56
count  2200.000000  2200.000000  2200.000000  2200.000000  2200.000000
mean      2.482273     2.435909     2.500455     2.456364     2.512273
std       1.065672     1.089652     1.091952     1.093162     1.086456
min       1.000000     1.000000     1.000000     1.000000     1.000000
25%       2.000000     1.000000     2.000000     1.000000     2.000000
50%       2.000000     2.000000     3.000000     2.000000     3.000000
75%       3.000000     3.000000     3.000000     3.000000     3.000000
max       4.000000     4.000000     4.000000     4.000000     4.000000

[8 rows x 57 columns]
```

Check For Missing Values in the Dataset

In [8]:
```
# Number of Missing Values in the Dataset
print("Total Number of Missing Values in the Dataset: ",df_features_dataA.isna().sum().sum())
```

```
Total Number of Missing Values in the Dataset:  0
```

There are no missing values in the dataset

In [9]:
```
# Convert the Labels to Dataframe and Check for Unique Values
df_label_dataA = pd.DataFrame(mat_label_dataA)
print("Number of Unique Lables in the Dataset: ", df_label_dataA[0].unique())
```

```
Number of Unique Lables in the Dataset:  [ 1 -1]
```

There are two unique labels in the data set -1 and 1

Question 1.1: - Z-Score Normalization

In [10]:
```
# Z-Score Normalization
ss = StandardScaler()
ss.fit(df_features_dataA)
z_standardized_dataA = ss.transform(df_features_dataA)
```

In [11]:
```
# Standardised Dataset
print("Standardised DataA: \n",z_standardized_dataA)
```

```
Standardised DataA:
 [[-1.35289759 -0.48747864 -1.37244139 ... -0.4584159  -0.41756618
  -1.39224875]
 [ 0.45920268  0.43308188  1.36002111 ... -1.37441497  0.49741947
   1.36965138]
 [ 1.36525282 -1.40803915  1.36002111 ... -0.4584159   1.41240513
   1.36965138]
 ...
 [ 0.45920268  1.35364239  1.36002111 ...  0.45758317 -1.33255183
  -0.47161537]
 [-1.35289759  1.35364239  0.44920027 ...  1.37358225  0.49741947
   1.36965138]
 [ 1.36525282  0.43308188 -1.37244139 ... -0.4584159   1.41240513
   0.44901801]]
```

In [12]:
```
# Convert Labels to numpy array
np_label_dataA = df_label_dataA.to_numpy()
print("converted array is :",np_label_dataA)
```

```
converted array is : [[1]
 [1]
 [1]
 ...
 [1]
 [1]
 [1]]
```

Randomly spliting the data 70% of samples for taining and other 30% for testing

using hold-out scheme

In [13]:
```
# Split the Dataset using Train Test Split
x_train_dataA, x_test_dataA, y_train_dataA, y_test_dataA = train_test_split(z_standardized_dataA,np_lab
el_dataA.ravel(),test_size = 0.3, random_state = 42)

print(" \n Features for Train Dataset: \n", x_train_dataA)
print(" \n Labels for the Train Dataset: \n", y_train_dataA)
```

```
 Features for Train Dataset:
 [[ 1.36525282 -0.48747864  0.44920027 ...  1.37358225  1.41240513
   1.36965138]
 [-0.44684745 -0.48747864 -1.37244139 ...  1.37358225 -1.33255183
   1.36965138]
 [-0.44684745  1.35364239 -1.37244139 ...  0.45758317 -1.33255183
   0.44901801]
 ...
 [ 1.36525282 -0.48747864 -0.46162056 ... -0.4584159  -0.41756618
   0.44901801]
 [ 0.45920268  0.43308188  0.44920027 ... -1.37441497  0.49741947
   0.44901801]
 [-0.44684745  1.35364239 -0.46162056 ... -1.37441497 -0.41756618
  -0.47161537]]

 Labels for the Train Dataset:
 [ 1  1 -1 ... -1 -1  1]
```

The hold out method is kind of cross validation method where the dataset is split into training and test set. Later the model is trained on only traning set and then output values are predicted using the same model for the Test set.

Advantages:

Easier to compute

Disadvantage:

1. Sometimes evaluation has higher variance
2. It depends on what datapoints are in test set and what in training set so it depends on split heavily.

Question 1.2: 5-fold Cross Validation on Training set to select the Parameters k for k_NN from given set and plot

In [14]:
```
# Create a list for k from 1 to 31
k_list=list(range(1,32,2))
print("Set of parameters for k value is: \n",k_list)
```

```
Set of parameters for k value is:
 [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31]
```

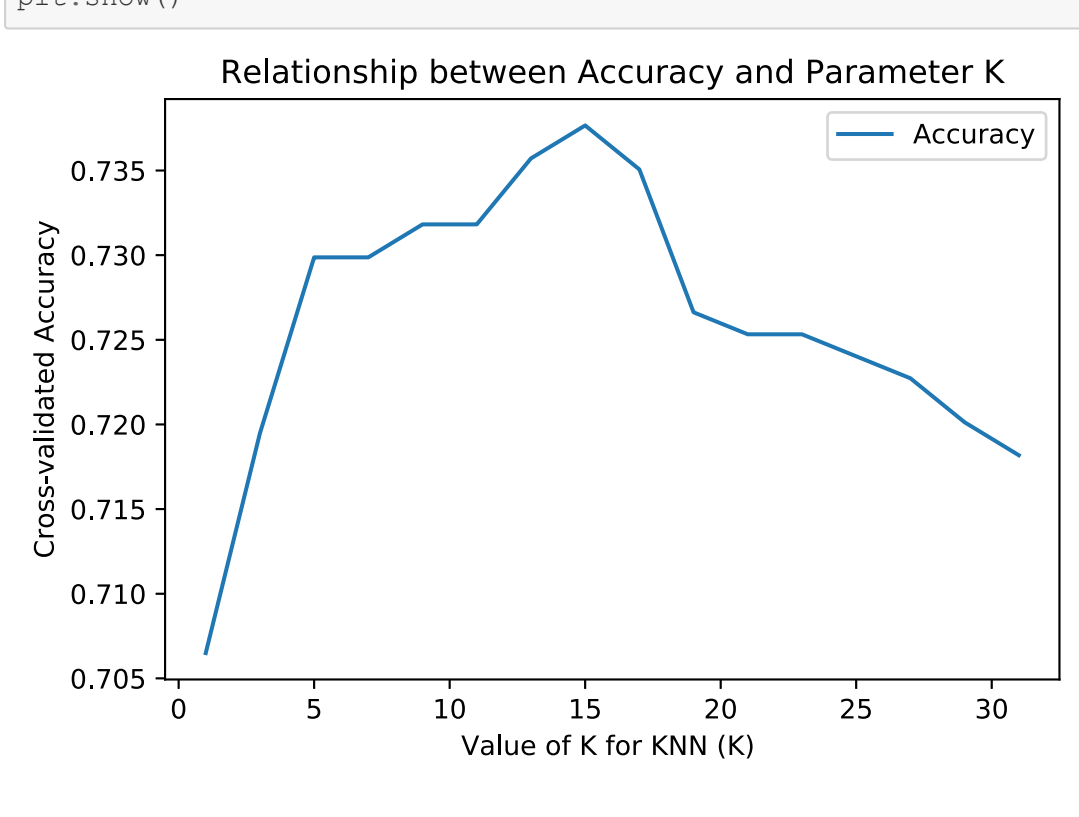
In [15]:
```
# empty list to store accuracy
K_accuracy=[]

# looping through given values of k
for k in k_list:
    # 2. KNeighborsClassifier with k neighbours
    knn = KNeighborsClassifier(n_neighbors = k)
    # 3. cross_val_score for KNeighborsClassifier with k neighbours
    accuracy = cross_val_score(knn, x_train_dataA, y_train_dataA, cv=5, scoring = 'accuracy')
    print("For k: {0}, Accuracies: {1}".format(k, accuracy))
    #4. append mean of scores for k neighbours to k_accuracy list
    K_accuracy.append(accuracy.mean())

print("Mean of accuracy scores: \n", K_accuracy)
```

```
For k: 1, Accuracies: [0.69805195 0.69805195 0.69155844 0.72077922 0.72402597]
For k: 3, Accuracies: [0.7012987  0.73376623 0.70779221 0.73701299 0.71753247]
For k: 5, Accuracies: [0.72727273 0.73376623 0.75324675 0.72402597 0.71103896]
For k: 7, Accuracies: [0.73701299 0.74675325 0.73376623 0.73051948 0.7012987 ]
For k: 9, Accuracies: [0.74675325 0.75324675 0.73701299 0.72402597 0.69805195]
For k: 11, Accuracies: [0.75649351 0.76298701 0.71103896 0.73051948 0.69805195]
For k: 13, Accuracies: [0.73701299 0.77272727 0.71103896 0.74350649 0.71428571]
For k: 15, Accuracies: [0.72727273 0.75324675 0.73376623 0.75324675 0.72077922]
For k: 17, Accuracies: [0.72077922 0.75       0.72727273 0.76298701 0.71428571]
For k: 19, Accuracies: [0.72077922 0.73701299 0.72077922 0.75       0.70454545]
For k: 21, Accuracies: [0.71103896 0.75       0.70779221 0.75974026 0.69805195]
For k: 23, Accuracies: [0.68831169 0.75       0.71103896 0.76623377 0.71103896]
For k: 25, Accuracies: [0.69155844 0.75974026 0.71428571 0.74350649 0.71103896]
For k: 27, Accuracies: [0.69155844 0.76298701 0.71753247 0.74025974 0.7012987 ]
For k: 29, Accuracies: [0.69805195 0.74025974 0.71428571 0.73701299 0.71103896]
For k: 31, Accuracies: [0.69805195 0.74350649 0.70454545 0.73376623 0.71103896]
Mean of accuracy scores:
 [0.7064935064935064, 0.7194805194805195, 0.7298701298701299, 0.7298701298701298, 0.7318181818181817,
0.7318181818181817, 0.7357142857142858, 0.7376623376623377, 0.7350649350649351, 0.7266233766233767,
0.7253246753246754, 0.7253246753246754, 0.724025974025974, 0.7227272727272728, 0.7201298701298702, 0.
7181818181818181]
```

In [16]:
```
plt.plot(k_list, K_accuracy, label = 'Accuracy')
plt.xlabel('Value of K for KNN (K) ')
plt.ylabel('Cross-validated Accuracy')
plt.title('Relationship between Accuracy and Parameter K')
plt.legend(loc = 'best')
plt.show()
```



In [17]:
```
# Get the Maximum Value of Accuracy Score
print('The best Calculation accuracy is :' , max(K_accuracy))
```

```
The best Calculation accuracy is : 0.7376623376623377
```

The Best Calculation accuracy of 0.7376623376623377 ie. 73.77% is best for value of K = 15

Hence the Best K in term of Classification Accuracy is 15.

Question 1.3: RBF Kernel best parameter selection in terms of classification accuracy

In [18]:
```
# Dicitionary of Parameters C and Gamma
c = [0.1, 0.5, 1, 2, 5, 10, 20, 50]
gamma = [0.01, 0.05 , 0.1, 0.5, 1, 2, 5, 10]
param_dict = dict(C = c, gamma = gamma)
print("Dicitionary of Parameters C and Gamma: \n ",param_dict)
```

```
Dicitionary of Parameters C and Gamma:
  {'C': [0.1, 0.5, 1, 2, 5, 10, 20, 50], 'gamma': [0.01, 0.05, 0.1, 0.5, 1, 2, 5, 10]}
```

In [19]:
```
# Data fitted with best parameters
clf = svm.SVC(kernel = 'rbf')
search = GridSearchCV(clf, param_dict, cv = 5, scoring = 'accuracy').fit(x_train_dataA, y_train_dataA)
```

In [20]:
```
# Best Score for the Search
print(" Best Score from Grid Search: ",search.best_score_)
```

```
Best Score from Grid Search:  0.9045454545454547
```

In [21]:
```
# Best Parameters from the Grid Search
print("Best Parameters from the Grid Search: \n", search.best_params_)
```

```
Best Parameters from the Grid Search:
 {'C': 10, 'gamma': 0.01}
```

The best parameters are : The best Soft Margin penality 'c' is 10 and gamma is 0.01

Question 1.4:

In [22]:
```
# Feature Test set and Label Test Set
print("Feature Test Set: \n",x_test_dataA)
print("Label Test Set: \n",y_test_dataA)
```

```
Feature Test Set:
 [[ 1.36525282 -0.48747864 -0.46162056 ... -1.37441497 -0.41756618
   0.44901801]
 [-1.35289759  1.35364239 -1.37244139 ...  1.37358225  0.49741947
  -1.39224875]
 [ 0.45920268  0.43308188  1.36002111 ...  1.37358225  1.41240513
   0.44901801]
 ...
 [ 1.36525282 -1.40803915  1.36002111 ...  1.37358225 -0.41756618
   1.36965138]
 [ 0.45920268  1.35364239  0.44920027 ...  1.37358225  1.41240513
   0.44901801]
 [ 1.36525282  1.35364239  0.44920027 ...  0.45758317  1.41240513
   0.44901801]]
Label Test Set:
 [-1  1  1 -1  1 -1  1 -1 -1 -1 -1 -1  1  1  1 -1  1  1  1 -1 -1  1 -1 -1
 -1  1  1  1  1 -1  1 -1 -1  1 -1 -1 -1  1 -1 -1  1 -1  1 -1  1  1 -1 -1
  1 -1  1  1  1 -1  1 -1  1  1  1 -1  1  1 -1 -1 -1  1 -1 -1  1  1  1 -1
 -1 -1  1  1  1  1 -1  1  1  1  1 -1 -1  1  1 -1 -1 -1 -1  1  1  1  1 -1
  1  1 -1  1  1 -1  1  1  1 -1  1 -1 -1  1 -1  1  1 -1  1 -1  1  1  1 -1
  1 -1 -1  1  1  1 -1 -1  1 -1  1  1 -1 -1 -1 -1  1  1  1  1  1  1  1 -1
  1  1  1  1  1  1 -1 -1 -1  1 -1  1  1 -1 -1  1  1  1 -1  1  1 -1 -1 -1
  1  1 -1  1  1  1 -1  1  1 -1 -1 -1 -1 -1 -1  1 -1  1 -1  1  1  1 -1  1
 -1  1  1  1  1  1 -1  1  1  1  1 -1 -1 -1  1  1 -1  1  1  1  1  1  1 -1
 -1 -1 -1  1 -1 -1  1  1 -1 -1  1  1  1 -1  1  1 -1  1 -1 -1  1  1 -1  1
 -1 -1 -1  1  1 -1 -1  1  1 -1  1 -1 -1  1 -1  1  1  1 -1  1 -1  1 -1  1
 -1 -1  1 -1  1  1  1  1 -1  1  1 -1  1  1  1  1 -1 -1 -1  1  1 -1  1 -1
 -1 -1 -1  1  1 -1  1  1  1 -1  1 -1 -1 -1  1 -1 -1  1  1  1 -1  1 -1 -1
  1 -1 -1  1  1 -1 -1 -1  1  1 -1  1 -1 -1  1 -1  1  1  1 -1 -1 -1  1  1
 -1  1  1 -1  1 -1  1  1  1 -1 -1  1 -1  1 -1 -1  1  1 -1  1  1 -1 -1  1
 -1 -1 -1 -1  1 -1 -1  1  1  1  1  1  1 -1 -1  1 -1 -1  1 -1 -1  1  1 -1
  1 -1  1  1  1 -1  1 -1 -1  1 -1 -1  1 -1  1 -1 -1  1  1 -1  1  1  1 -1
 -1  1 -1  1  1 -1 -1 -1  1 -1  1  1  1  1  1 -1 -1 -1  1 -1 -1 -1  1  1
 -1  1  1  1  1 -1  1  1  1 -1 -1  1 -1 -1 -1 -1  1 -1 -1  1 -1 -1  1  1
 -1 -1 -1 -1 -1  1 -1  1  1 -1 -1  1  1 -1  1 -1 -1 -1 -1 -1  1 -1  1  1
  1  1 -1 -1 -1 -1 -1  1 -1  1 -1 -1  1  1 -1  1  1 -1  1 -1  1 -1 -1 -1
 -1  1 -1  1 -1 -1 -1  1 -1  1 -1 -1  1  1  1 -1  1  1  1  1  1  1 -1 -1
 -1 -1 -1 -1  1 -1 -1  1  1  1 -1  1  1 -1  1 -1  1  1  1  1  1 -1 -1 -1
 -1 -1  1 -1  1  1 -1  1 -1 -1  1  1 -1 -1 -1 -1 -1  1 -1  1 -1  1 -1  1
  1  1 -1 -1 -1  1 -1 -1  1  1 -1  1 -1  1 -1 -1  1  1 -1  1  1 -1  1  1
  1  1 -1 -1 -1 -1  1 -1 -1 -1  1  1  1  1 -1 -1  1  1 -1 -1 -1  1 -1  1
  1 -1  1 -1  1 -1 -1 -1  1  1  1  1 -1 -1  1 -1  1  1 -1 -1 -1 -1  1  1
  1  1  1 -1  1 -1 -1 -1  1 -1  1  1 -1 -1  1 -1  1  1  1  1 -1  1  1  1
 -1 -1 -1  1 -1 -1 -1  1 -1  1 -1  1]
```

KNN Classifier

In [23]:
```
# For K-NN using above parameters to classify the test set
# k = 15 gave best classification accuracy

Knn_model = KNeighborsClassifier(n_neighbors = 15)
Knn_model.fit(x_train_dataA,y_train_dataA)

y_pred_KNN = Knn_model.predict(x_test_dataA)

# comparison between actual and predicted response
print("Accuracy is :",metrics.accuracy_score(y_test_dataA, y_pred_KNN))
print("Precision is :",metrics.precision_score(y_test_dataA, y_pred_KNN))
print("Recall is :",metrics.recall_score(y_test_dataA, y_pred_KNN))
print("F1-Score is :",metrics.f1_score(y_test_dataA, y_pred_KNN))
```

```
Accuracy is : 0.75
Precision is : 0.9202127659574468
Recall is : 0.5356037151702786
F1-Score is : 0.6771037181996086
```

In [24]:
```
# Classification Report for Each Class for KNN
print("Classification Report for KNN: \n", classification_report(y_test_dataA, y_pred_KNN))
```

```
Classification Report for KNN:
               precision    recall  f1-score   support

          -1       0.68      0.96      0.80       337
           1       0.92      0.54      0.68       323

    accuracy                           0.75       660
   macro avg       0.80      0.75      0.74       660
weighted avg       0.80      0.75      0.74       660
```

SVM Classifier

In [25]:
```
# For SVM using above parameters to classify the test set
SVM_model = SVC(C = 10,gamma = 0.01)
SVM_model.fit(x_train_dataA,y_train_dataA)

y_pred_SVM = SVM_model.predict(x_test_dataA)

# comparison between actual and predicted response
print("Accuracy is :",metrics.accuracy_score(y_test_dataA, y_pred_SVM))
print("Precision is :",metrics.precision_score(y_test_dataA, y_pred_SVM))
print("Recall is :",metrics.recall_score(y_test_dataA, y_pred_SVM))
print("F1-Score is :",metrics.f1_score(y_test_dataA, y_pred_SVM))
```

```
Accuracy is : 0.9075757575757576
Precision is : 0.9145569620253164
Recall is : 0.8947368421052632
F1-Score is : 0.9045383411580594
```

In [26]:
```
# Classification Report for Each Class for SVM
print("Classification Report for SVM: \n", classification_report(y_test_dataA, y_pred_SVM))
```

```
Classification Report for SVM:
               precision    recall  f1-score   support

          -1       0.90      0.92      0.91       337
           1       0.91      0.89      0.90       323

    accuracy                           0.91       660
   macro avg       0.91      0.91      0.91       660
weighted avg       0.91      0.91      0.91       660
```

Gaussian Naive Bayes

In [27]:
```
# Using default setups for Naive Bayes Classifier to classify test set
NB_model = GaussianNB()
NB_model.fit(x_train_dataA,y_train_dataA)

y_pred_NB = NB_model.predict(x_test_dataA)

# comparison between actual and predicted response
print("Accuracy is :",metrics.accuracy_score(y_test_dataA, y_pred_NB))
print("Precision is :",metrics.precision_score(y_test_dataA, y_pred_NB))
print("Recall is :",metrics.recall_score(y_test_dataA, y_pred_NB))
print("F1-Score is :",metrics.f1_score(y_test_dataA, y_pred_NB))
```

```
Accuracy is : 0.8818181818181818
Precision is : 0.8792569659442725
Recall is : 0.8792569659442725
F1-Score is : 0.8792569659442725
```

In [28]:
```
# Classification Report for Each Class for Gaussian Naive Bayes
print("Classification Report for  Gaussian Naive Bayes: \n", classification_report(y_test_dataA, y_pred
_NB))
```

```
Classification Report for  Gaussian Naive Bayes:
               precision    recall  f1-score   support

          -1       0.88      0.88      0.88       337
           1       0.88      0.88      0.88       323

    accuracy                           0.88       660
   macro avg       0.88      0.88      0.88       660
weighted avg       0.88      0.88      0.88       660
```

Decision Tree

In [29]:
```
# Using default setups for Decision Trees to classify test set
DT_model = DecisionTreeClassifier()
DT_model.fit(x_train_dataA,y_train_dataA)

y_pred_DT = DT_model.predict(x_test_dataA)

# comparison between actual and predicted response
print("Accuracy is :",metrics.accuracy_score(y_test_dataA, y_pred_DT))
print("Precision is :",metrics.precision_score(y_test_dataA, y_pred_DT))
print("Recall is :",metrics.recall_score(y_test_dataA, y_pred_DT))
print("F1-Score is :",metrics.f1_score(y_test_dataA, y_pred_DT))
```

```
Accuracy is : 0.9333333333333333
Precision is : 0.9428571428571428
Recall is : 0.9195046439628483
F1-Score is : 0.9310344827586207
```

In [30]:
```
# Classification Report for Each Class for Decision Trees
print("Classification Report for Decision Trees: \n", classification_report(y_test_dataA, y_pred_DT))
```

```
Classification Report for Decision Trees:
               precision    recall  f1-score   support

          -1       0.92      0.95      0.94       337
           1       0.94      0.92      0.93       323

    accuracy                           0.93       660
   macro avg       0.93      0.93      0.93       660
weighted avg       0.93      0.93      0.93       660
```

KNN Classifier with 20 Iterations

In [31]:
```
# Repeating above steps 20 times by varying the split of training-test set as in Step 1

# for KNN
Knn_model2 = KNeighborsClassifier(n_neighbors=15)
# Classification based on KNN
random_seed_iterations = range(1,42,2)
KNN_accuracy_list=[]
KNN_precision_list=[]
KNN_recall_list=[]
KNN_f1_list=[]

# Iteration for Retraining
for i in random_seed_iterations:

    # Random Dataset Split
    x_train_dataA_KNN, x_test_dataA_KNN, y_train_dataA_KNN, y_test_dataA_KNN = train_test_split(z_stand
ardized_dataA,np_label_dataA.ravel(),test_size =0.3, random_state = i)

    # Training the Model
    Knn_model2.fit(x_train_dataA_KNN, y_train_dataA_KNN)

    # Test Dataset Prediction
    y_predict_KNN_list = Knn_model2.predict(x_test_dataA_KNN)

    # Classification Accuracy
    accuracy = metrics.accuracy_score(y_test_dataA_KNN, y_predict_KNN_list)
    KNN_accuracy_list.append(accuracy)

    # Precision
    precision = metrics.precision_score(y_test_dataA_KNN, y_predict_KNN_list)
    KNN_precision_list.append(precision)

    # Recall
    recall = metrics.recall_score(y_test_dataA_KNN, y_predict_KNN_list)
    KNN_recall_list.append(recall)

    # F1-score
    f1score = metrics.f1_score(y_test_dataA_KNN, y_predict_KNN_list)
    KNN_f1_list.append(f1score)


print("Acuracy list is :",KNN_accuracy_list)
print("\n")
print("Precision list is :",KNN_precision_list)
print("\n")
print("Recall list is:",KNN_recall_list)
print("\n")
print("F1-Score list is:",KNN_f1_list)

# Average and standard Deviation of Classification values

mean_accuracy_list_KNN = sum(KNN_accuracy_list)/len(KNN_accuracy_list)
variance_accuracy_list_KNN = sum([((x-mean_accuracy_list_KNN)**2) for x in KNN_accuracy_list])/len(KNN_
accuracy_list)
sd_accuracy_list_KNN = variance_accuracy_list_KNN**0.5

print("Mean for accuracy in case of KNN is: ",mean_accuracy_list_KNN)
print("Standard Deviation for accuracy in case of KNN is: ",sd_accuracy_list_KNN)
print("\n")

mean_precision_list_KNN=sum(KNN_precision_list)/len(KNN_precision_list)
variance_precision_list_KNN=sum([((x-mean_precision_list_KNN)**2) for x in KNN_precision_list])/len(KNN
_precision_list)
sd_precision_list_KNN=variance_precision_list_KNN**0.5

print("Mean for precision list in case of KNN is: ",mean_precision_list_KNN)
print("Standard Deviation for precision list in case of KNN is: ",sd_precision_list_KNN)
print("\n")

mean_recall_score_list_KNN=sum(KNN_recall_list)/len(KNN_recall_list)
variance_recall_score_list_KNN=sum([((x-mean_recall_score_list_KNN)**2) for x in KNN_recall_list])/len(
KNN_recall_list)
sd_recall_score_list_KNN=variance_recall_score_list_KNN**0.5

print("Mean for recall score in case of KNN is: ",mean_recall_score_list_KNN)
print("Standard Deviation for recall score in case of KNN is: ",sd_recall_score_list_KNN)
print("\n")

mean_f1_score_list_KNN=sum(KNN_f1_list)/len(KNN_f1_list)
variance_f1_score_list_KNN=sum([((x-mean_f1_score_list_KNN)**2) for x in KNN_f1_list])/len(KNN_f1_list)
sd_f1_score_list_KNN=variance_f1_score_list_KNN**0.5

print("Mean for f1 score in case of KNN is: ",mean_f1_score_list_KNN)
print("Standard Deviation for f1 score in case of KNN is: ",sd_f1_score_list_KNN)
```

```
Acuracy list is : [0.7818181818181819, 0.7545454545454545, 0.7227272727272728, 0.759090909090909, 0.7
575757575757576, 0.7378787878787879, 0.7803030303030303, 0.75, 0.7272727272727273, 0.775757575757575
8, 0.7151515151515152, 0.7045454545454546, 0.7166666666666667, 0.7681818181818182, 0.743939393939393
9, 0.7484848484848485, 0.7424242424242424, 0.7333333333333333, 0.746969696969697,
 0.7378787878787879]


Precision list is : [0.9748743718592965, 0.9572192513368984, 0.9714285714285714, 0.9637305699481865,
 0.9575471698113207, 0.9521276595744681, 0.9660194174757282, 0.9421052631578948,
 0.9690721649484536, 0.9828571428571429, 0.9378531073446328, 0.9662921348314607, 0.9635416666666666,
 0.9896907216494846, 0.9567567567567568, 0.9646464646464646, 0.9651162790697675, 0.9606741573033708,
 0.9646464646464646, 0.9779005524861878]


Recall list is: [0.5825825825825826, 0.5375375375375375, 0.4885057471264368, 0.5502958579881657, 0.57
1734463276836158, 0.521865889212828, 0.5905044510385756, 0.5511363636363636, 0.514367816091954, 0.56969
69696969697, 0.4817927170868347, 0.4742857142857143, 0.48725212464589235, 0.5589123867069486, 0.5348
189415041783, 0.5283582089552239, 0.5395480225988701, 0.4853801169590643, 0.5029411764705882, 0.5441595
441595442, 0.5115606936416185]


F1-Score list is: [0.7293233082706767, 0.6884615384615385, 0.6500956022944551, 0.7005649717514124, 0.
7173144876325089, 0.6741996233521657, 0.7329650092081003, 0.7016274864376131, 0.6654275092936803, 0.71
7557251908397, 0.6466165413533834, 0.6299810246679318, 0.647834274952919, 0.7074569789674953, 0.69439
42133855553, 0.6807692307692309, 0.6920289855072463, 0.6459143968871596, 0.6602316602316602, 0.695810
5646630237, 0.6717267552182162]
Mean for accuracy in case of KNN is:  0.7442279942279942
Standard Deviation for accuracy in case of KNN is:  0.021132705635035186


Mean for precision list in case of KNN is:  0.964251436739261


Standard Deviation for precision list in case of KNN is:  0.011700384104702295


Mean for recall score in case of KNN is:  0.5299499614099764


Standard Deviation for recall score in case of KNN is:  0.03402303748050454


Mean for f1 score in case of KNN is:  0.6833476864385892


Standard Deviation for f1 score in case of KNN is:  0.02889766938216869
```

SVM Classifier with 20 Iterations

In [32]:

```python
# For SVM
SVM_model2 = SVC(C = 10,gamma = 0.01)

# Classification based on SVM
SVM_accuracy_list=[]
SVM_precision_list=[]
SVM_recall_list=[]
SVM_f1_list=[]

# Iteration for Retraining
for i in random_seed_iterations:
    # Random Dataset Split
    x_train_dataA_SVM, x_test_dataA_SVM, y_train_dataA_SVM, y_test_dataA_SVM = train_test_split(z_standardized_dataA, np_label_dataA.ravel(), test_size =0.3, random_state = i)
    # Training the Model
    SVM_model2.fit(x_train_dataA_SVM, y_train_dataA_SVM)

    # Test Dataset Prediction
    y_predict_SVM_list = SVM_model2.predict(x_test_dataA_SVM)

    # Classification Accuracy
    accuracy = metrics.accuracy_score(y_test_dataA_SVM, y_predict_SVM_list)
    SVM_accuracy_list.append(accuracy)

    #Precision
    precision = metrics.precision_score(y_test_dataA_SVM, y_predict_SVM_list)
    SVM_precision_list.append(precision)

    #Recall
    recall = metrics.recall_score(y_test_dataA_SVM, y_predict_SVM_list)
    SVM_recall_list.append(recall)

    #F1-score
    f1score = metrics.f1_score(y_test_dataA_SVM, y_predict_SVM_list)
    SVM_f1_list.append(f1score)

print("Acuracy list is :",SVM_accuracy_list)
print("\n")
print("Precision list is :",SVM_precision_list)
print("\n")
print("Recall list is:",SVM_recall_list)
print("\n")
print("F1-Score list is:",SVM_f1_list)
print("\n")

#Average and standard Deviation of Classification values
mean_accuracy_list_SVM = sum(SVM_accuracy_list)/len(SVM_accuracy_list)
variance_accuracy_list_SVM = sum([((x-mean_accuracy_list_SVM)**2) for x in SVM_accuracy_list])/len(SVM_accuracy_list)
sd_accuracy_list_SVM = variance_accuracy_list_SVM**0.5

print("mean for accuracy in case of SVM is: ",mean_accuracy_list_SVM)
print("\n")
print("Standard Deviation for accuracy in case of SVM is: ",sd_accuracy_list_SVM)
print("\n")
mean_precision_list_SVM=sum(SVM_precision_list)/len(SVM_precision_list)
variance_precision_list_SVM=sum([((x-mean_precision_list_SVM)**2) for x in SVM_precision_list])/len(SVM_precision_list)
sd_precision_list_SVM=variance_precision_list_SVM**0.5

print("mean for precision list in case of SVM is: ",mean_precision_list_SVM)
print("\n")
print("Standard Deviation for precision list in case of SVM is: ",sd_precision_list_SVM)
print("\n")

mean_recall_score_list_SVM=sum(SVM_recall_list)/len(SVM_recall_list)
variance_recall_score_list_SVM=sum([((x-mean_recall_score_list_SVM)**2) for x in SVM_recall_list])/len(SVM_recall_list)
sd_recall_score_list_SVM=variance_recall_score_list_SVM**0.5

print("mean for recall score in case of SVM is: ",mean_recall_score_list_SVM)
print("\n")
print("Standard Deviation for recall score in case of SVM is: ",sd_recall_score_list_SVM)
print("\n")

mean_f1_score_list_SVM=sum(SVM_f1_list)/len(SVM_f1_list)
variance_f1_score_list_SVM=sum([((x-mean_f1_score_list_SVM)**2) for x in SVM_f1_list])/len(SVM_f1_list)
sd_f1_score_list_SVM=variance_f1_score_list_SVM**0.5

print("mean for f1 score in case of SVM is: ",mean_f1_score_list_SVM)
print("\n")
print("Standard Deviation for f1 score in case of SVM is: ",sd_f1_score_list_SVM)
print("\n")
```

```
Acuracy list is : [0.9227272727272727, 0.9121212121212121, 0.9121212121212121, 0.9075757575757576, 0.9, 0.8742424242424243, 0.8939393939393939, 0.9181818181818182, 0.906060606060606, 0.9136363636363637, 0.8893939393939394, 0.8893939393939394, 0.9090909090909091, 0.9166666666666666, 0.9196969696969697, 0.9136363636363637, 0.9075757575757576, 0.9, 0.8954545454545455, 0.9045454545454545, 0.9106060606060606]


Precision list is : [0.937888198757764, 0.8962536023054755, 0.9289940828402367, 0.9287925696594427, 0.9022346368715084, 0.8846153846153846, 0.8985074626865671, 0.9408284023668639, 0.9255952380952381, 0.9174311926605505, 0.9303030303030303, 0.9134328358208955, 0.9347181008902077, 0.9394904458598726, 0.9371428571428572, 0.9237804878048781, 0.9197707736389685, 0.8942857142857142, 0.9118541033434651, 0.9285714285714286, 0.9309309309309309]


Recall list is: [0.9069069069069069, 0.933933933933934, 0.9022988505747126, 0.8875739644970414, 0.9124293785310734, 0.8717201166180758, 0.8931750741839762, 0.9034090909090909, 0.8936781609195402, 0.9090909090909091, 0.8599439775910365, 0.8742857142857143, 0.8923512747875354, 0.8912386706948641, 0.9136490250696379, 0.9044776119402985, 0.9067796610169492, 0.9152046783625731, 0.8823529411764706, 0.8888888888888888, 0.8959537572254336]


F1-Score list is: [0.9221374045801528, 0.9147058823529413, 0.9154518950437318, 0.9077155824508321, 0.9073033707865168, 0.8781204111600587, 0.8958333333333333, 0.9217391304347827, 0.9093567251461988, 0.9132420091324202, 0.8937409024745271, 0.8934306569343066, 0.9130434782608695, 0.9147286821705426, 0.9252468265162199, 0.914027149321267, 0.9132290184921764, 0.9046242774566474, 0.8968609865470851, 0.9082969432314411, 0.9131075110456554]


mean for accuracy in case of SVM is:  0.9055555555555556


Standard Deviation for accuracy in case of SVM is:  0.011636533989453692


mean for precision list in case of SVM is:  0.9202581656881561


Standard Deviation for precision list in case of SVM is:  0.01620781017136178


mean for recall score in case of SVM is:  0.8971115517716507


Standard Deviation for recall score in case of SVM is:  0.01640915447683593


mean for f1 score in case of SVM is:  0.9083781988986527


Standard Deviation for f1 score in case of SVM is:  0.011053761601686089
```

Naive Bayes Classifier with 20 Iterations

In [33]:

```python
# For Naive Bayes
NB_model2 = GaussianNB()

# Classification based on NB
NB_accuracy_list = []
NB_precision_list = []
NB_recall_list = []
NB_f1_list = []

# Iteration for Retraining
for i in random_seed_iterations:
    # Random Dataset Split
    x_train_dataA_NB, x_test_dataA_NB, y_train_dataA_NB, y_test_dataA_NB = train_test_split(z_standardized_dataA, np_label_dataA.ravel(), test_size = 0.3, random_state = i)

    #Training the Model
    NB_model2.fit(x_train_dataA_NB, y_train_dataA_NB)

    #Test Dataset Prediction
    y_predict_NB_list = NB_model2.predict(x_test_dataA_NB)

    #Classification Accuracy
    accuracy = metrics.accuracy_score(y_test_dataA_NB, y_predict_NB_list)
    NB_accuracy_list.append(accuracy)

    #Precision
    precision = metrics.precision_score(y_test_dataA_NB, y_predict_NB_list)
    NB_precision_list.append(precision)

    #Recall
    recall = metrics.recall_score(y_test_dataA_NB, y_predict_NB_list)
    NB_recall_list.append(recall)

    #F1-score
    f1score = metrics.f1_score(y_test_dataA_NB, y_predict_NB_list)
    NB_f1_list.append(f1score)

print("Acuracy list is :",NB_accuracy_list)
print("\n")
print("Precision list is :",NB_precision_list)
print("\n")
print("Recall list is:",NB_recall_list)
print("\n")
print("F1-Score list is:",NB_f1_list)
print("\n")

#Average and standard Deviation of Classification values
mean_accuracy_list_NB = sum(NB_accuracy_list)/len(NB_accuracy_list)
variance_accuracy_list_NB = sum([((x-mean_accuracy_list_NB)**2) for x in NB_accuracy_list])/len(NB_accuracy_list)
sd_accuracy_list_NB = variance_accuracy_list_NB**0.5

print("Mean for accuracy in case of Naive Bayes is: ",mean_accuracy_list_NB)
print("\n")
print("Standard Deviation for accuracy in case of Naive Bayes is: ",sd_accuracy_list_NB)
print("\n")
mean_precision_list_NB=sum(NB_precision_list)/len(NB_precision_list)
variance_precision_list_NB=sum([((x-mean_precision_list_NB)**2) for x in NB_precision_list])/len(NB_precision_list)
sd_precision_list_NB=variance_precision_list_NB**0.5

print("Mean for precision list in case of Naive Bayes is: ",mean_precision_list_NB)
print("\n")
print("Standard Deviation for precision list in case of Naive Bayes is: ",sd_precision_list_NB)
print("\n")

mean_recall_score_list_NB=sum(NB_recall_list)/len(NB_recall_list)
variance_recall_score_list_NB=sum([((x-mean_recall_score_list_NB)**2) for x in NB_recall_list])/len(NB_recall_list)
sd_recall_score_list_NB=variance_recall_score_list_NB**0.5

print("Mean for recall score in case of Naive Bayes is: ",mean_recall_score_list_NB)
print("\n")
print("Standard Deviation for recall score in case of Naive Bayes is: ",sd_recall_score_list_NB)
print("\n")

mean_f1_score_list_NB=sum(NB_f1_list)/len(NB_f1_list)
variance_f1_score_list_NB=sum([((x-mean_f1_score_list_NB)**2) for x in NB_f1_list])/len(NB_f1_list)
sd_f1_score_list_NB=variance_f1_score_list_NB**0.5

print("Mean for f1 score in case of Naive Bayes is: ",mean_f1_score_list_NB)
print("\n")
print("Standard Deviation for f1 score in case of Naive Bayes is: ",sd_f1_score_list_NB)
print("\n")
```

```
Acuracy list is : [0.8772727272727273, 0.8590909090909091, 0.8787878787878788, 0.8742424242424243, 0.853030303030303, 0.8545454545454545, 0.8712121212121212, 0.8696969696969697, 0.8878787878787879, 0.8727272727272727, 0.8742424242424243, 0.8742424242424243, 0.8590909090909091, 0.8909090909090909, 0.8727272727272727, 0.8681818181818182, 0.8772727272727273, 0.8575757575757575, 0.8469696969696969, 0.8681818181818182, 0.8696969696969697]


Precision list is : [0.8539325842696629, 0.835195530726257, 0.8806818181818182, 0.8875379939209727, 0.8599439775910365, 0.8518518518518519, 0.8442622950819673, 0.882183908045977, 0.8914285714285715, 0.8514285714285714, 0.8870056497175142, 0.8803418803418803, 0.8611111111111112, 0.8888888888888888, 0.862796833773087, 0.8583815028901735, 0.8601583113456465, 0.8463687150837989, 0.8385269121813032, 0.8771428571428571, 0.8651685393258427]


Recall list is: [0.9129129129129129, 0.8978978978978979, 0.8908045977011494, 0.863905325443787, 0.867231638418079, 0.8717201166180758, 0.9169139465875371, 0.8721590909090909, 0.896551724137931, 0.9030303030303031, 0.8795518207282913, 0.8828571428571429, 0.8781869688385269, 0.8942598187311178, 0.9108635097493036, 0.8865671641791045, 0.9209039548022598, 0.8859649122807017, 0.8705882352941177, 0.8746438746438746, 0.8901734104046243]


F1-Score list is: [0.8824383164005805, 0.8654124457308249, 0.8857142857142858, 0.8755622188905547, 0.8635724331926863, 0.861671469740634, 0.8790896159317213, 0.8771428571428572, 0.8939828080229227, 0.8764705882352941, 0.8832630098452883, 0.8815977175463623, 0.8695652173913044, 0.8915662650602408, 0.8861788617886179, 0.8722466960352423, 0.8894952251023192, 0.8657142857142858, 0.8542568542568543, 0.8758915834522111, 0.8774928774928775]


Mean for accuracy in case of Naive Bayes is:  0.8694083694083694


Standard Deviation for accuracy in case of Naive Bayes is:  0.01082154877878943


Mean for precision list in case of Naive Bayes is:  0.8649684906823234


Standard Deviation for precision list in case of Naive Bayes is:  0.017104767454261264


Mean for recall score in case of Naive Bayes is:  0.8889375412459919


Standard Deviation for recall score in case of Naive Bayes is:  0.016478153080216752


Mean for f1 score in case of Naive Bayes is:  0.8765869348899032


Standard Deviation for f1 score in case of Naive Bayes is:  0.010196866726204193
```

Decision Tree with 20 Iterations

In [34]:

```python
# Initialize Decision Tree
DT_model2 = DecisionTreeClassifier()

# Classification based on DT
DT_accuracy_list = []
DT_precision_list = []
DT_recall_list = []
DT_f1_list = []

#Iteration for Retraining
for i in random_seed_iterations:
    #Random Dataset Split
    x_train_dataA_DT, x_test_dataA_DT, y_train_dataA_DT, y_test_dataA_DT = train_test_split(z_standardized_dataA, np_label_dataA.ravel(),test_size = 0.3, random_state = i)

    # Training the Model
    DT_model2.fit(x_train_dataA_DT, y_train_dataA_DT)

    # Test Dataset Prediction
    y_predict_DT_list = DT_model2.predict(x_test_dataA_DT)

    # Classification Accuracy
    accuracy = metrics.accuracy_score(y_test_dataA_DT, y_predict_DT_list)
    DT_accuracy_list.append(accuracy)

    #Precision
    precision = metrics.precision_score(y_test_dataA_DT, y_predict_DT_list)
    DT_precision_list.append(precision)

    #Recall
    recall = metrics.recall_score(y_test_dataA_DT, y_predict_DT_list)
    DT_recall_list.append(recall)

    #F1-score
    f1score = metrics.f1_score(y_test_dataA_DT, y_predict_DT_list)
    DT_f1_list.append(f1score)

print("Acuracy list is :",DT_accuracy_list)
print("\n")
print("Precision list is :",DT_precision_list)
print("\n")
print("Recall is list :",DT_recall_list)
print("\n")
print("F1-Score is list :",DT_f1_list)
print("\n")

#Average and standard Deviation of Classification values
mean_accuracy_list_DT=sum(DT_accuracy_list)/len(DT_accuracy_list)
variance_accuracy_list_DT=sum([((x-mean_accuracy_list_DT)**2) for x in DT_accuracy_list])/len(DT_accuracy_list)
sd_accuracy_list_DT=variance_accuracy_list_DT**0.5

print("Mean for accuracy in case of Decision Tree is: ",mean_accuracy_list_DT)
print("\n")
print("Standard Deviation for accuracy in case of Decision Tree is: ",sd_accuracy_list_DT)
print("\n")
mean_precision_list_DT=sum(DT_precision_list)/len(DT_precision_list)
variance_precision_list_DT=sum([((x-mean_precision_list_DT)**2) for x in DT_precision_list])/len(DT_precision_list)
sd_precision_list_DT=variance_precision_list_DT**0.5

print("Mean for precision list in case of Decision Tree is: ",mean_precision_list_DT)
print("\n")
print("Standard Deviation for precision list in case of Decision Tree is: ",sd_precision_list_DT)
print("\n")

mean_recall_score_list_DT=sum(DT_recall_list)/len(DT_recall_list)
variance_recall_score_list_DT=sum([((x-mean_recall_score_list_DT)**2) for x in DT_recall_list])/len(DT_recall_list)
sd_recall_score_list_DT=variance_recall_score_list_DT**0.5

print("Mean for recall score in case of Decision Tree is: ",mean_recall_score_list_DT)
print("\n")
print("Standard Deviation for recall score in case of Decision Tree is: ",sd_recall_score_list_DT)
print("\n")

mean_f1_score_list_DT=sum(DT_f1_list)/len(DT_f1_list)
variance_f1_score_list_DT=sum([((x-mean_f1_score_list_DT)**2) for x in DT_f1_list])/len(DT_f1_list)
sd_f1_score_list_DT=variance_f1_score_list_DT**0.5

print("Mean for f1 score in case of Decision Tree is: ",mean_f1_score_list_DT)
print("\n")
print("Standard Deviation for f1 score in case of Decision Tree is: ",sd_f1_score_list_DT)
print("\n")
```

```
Acuracy list is : [0.9318181818181818, 0.9227272727272727, 0.9378787878787879, 0.9424242424242424, 0.9151515151515152, 0.9378787878787879, 0.9166666666666666, 0.9212121212121213, 0.9363636363636364, 0.9212121212121213, 0.9409090909090909, 0.9348484848484848, 0.9333333333333333, 0.9439393939393939, 0.9454545454545454, 0.9318181818181818, 0.9378787878787879, 0.9378787878787879, 0.9257575757575758, 0.9090909090909091, 0.9287878787878788]


Precision list is : [0.9260355029585798, 0.9246987951807228, 0.9449275362318841, 0.9491017964071856, 0.9382352941176471, 0.9520958083832335, 0.9028571428571428, 0.9411764705882353, 0.9273743016759777, 0.9040697674418605, 0.9542857142857143, 0.9528023598820059, 0.9376770538243626, 0.950920245398773, 0.9498607242339833, 0.9447852760736196, 0.9510086455331412, 0.9465875370919882, 0.9343283582089552, 0.9169054441260746, 0.9333333333333333]


Recall is list : [0.93993993993994, 0.9219219219219219, 0.9367816091954023, 0.9378698224852071, 0.9011299435028248, 0.9271137026239067, 0.9376854599406528, 0.9090909090909091, 0.9540229885057471, 0.9424242424242424, 0.9355742296918768, 0.9228571428571428, 0.9376770538243626, 0.9365558912386707, 0.9498607242339833, 0.9194029850746268, 0.9322033898305084, 0.9327485380116959, 0.9205882352941176, 0.9116809116809117, 0.930635838150289]


F1-Score is list : [0.932935916542474, 0.9233082706766917, 0.9408369408369408, 0.9434523809523809, 0.9193083573487031, 0.9394387001477105, 0.9199417758369723, 0.9248554913294799, 0.9405099150141643, 0.9228486646884273, 0.9448373408769448, 0.9375907111756168, 0.9376770538243626, 0.943683409436834, 0.9498607242339833, 0.9319213313161876, 0.9415121255349501, 0.939617083946981, 0.9274074074074073, 0.9142857142857144, 0.9319826338639653]


Mean for accuracy in case of Decision Tree is:  0.9310966810966812


Standard Deviation for accuracy in case of Decision Tree is:  0.009936822684379112


Mean for precision list in case of Decision Tree is:  0.9372889098968774


Standard Deviation for precision list in case of Decision Tree is:  0.01503965354025902


Mean for recall score in case of Decision Tree is:  0.9303697847389972


Standard Deviation for recall score in case of Decision Tree is:  0.012847793854870058


Mean for f1 score in case of Decision Tree is:  0.9337053309179472


Standard Deviation for f1 score in case of Decision Tree is:  0.009662905260889483
```

Question 1.5: Comment on obtained results

Question 1.2:

If we choose smaller value of k then the smaller value can be noisy while a higher value can be computationally very expensive as it would have smoother boundaries i.e. lower variance and higher bias. So we used cross validation to choose and take small data chunks from the dataset as validation dataset and use it against possible valuse of k which in our case were given. Then each label was predicted for every option in validation set using k = 1, 3.... 31 so that we can get best performance and take that value as best k value to minimize error in validation. Usually we take k = Sqrt(N) to chosse best value whrere N is no of samples in training dataset and the value of k is kept odd so that there is no confusion between 2 classes of data. The best k for k-NN from the given set is 15 as the Accuracy is higher among all other k after performing Cross Validation.

Question 1.3:

Penality Parameter C is how much miss classified points you need to avoid or how much you actually care about the misclassification. So a higher C value would mean that all the data points need to be classified correctly. In case of RBF kernel large values of C can result in over-fitting and can form classification isolated regions. Running GridSearchCV helped us to find the ideal value of C as 10. Using this value will prevent isolated regions and over-fitting.

The gamma parameter tells how quickly the Gaussian class boundaries disappear as they move away from support vector. So larger values of gamma would decrease the effect caused by any individual support vector. However, choosing a higher value of gamma results in future datapoints to fall in small clusters. So gamma tuning is needed. Hence we found the best value of gamma to avoid this issue. The best parameter selection is needed not only to get highest accuracy but also to correctly identify the situation that is whether clustering of future information should be together or what should be the margins between classes. The best value of gamma parameter obtained after GridSearchCV is 0.01.

In Question 1.4:

In the Case of KNN Classifier, for k = 15 the accuracy score, Precision, Recall, F1-Score is calculated. The accuracy score for the model is 75% for both the classes. The Recall and F1-score for Class label '-1' is higher than that of class label '1', whereas the Precision is higher for class label '1' than of class label '-1'. By evaluating the f1-score from the KNN classification report it is clear that the kNN algorithm is unable to classify the class label '1' as the score is lower than that of class label '-1'. After varying the randomness in the dataset for 20 Iteration the Mean accuracy score for KNN is 74.423% and the Standard Deviation is 0.021. The Mean F1-score is 0.683, Mean precision is 0.964 and Mean recall is 0.53 approximately. This values are nearly the same when compared to the values obtained from that of without Iteration for the k-NN classifier.

In the case of SVM Classifier, evaluation metrics precision, recall, f1-score and accuracy score are all very similar. It generalizes much better than k-NN classifier. The f1-score and precision for class label '-1' and '1' are very similar and they differ by 0.01 in each case. In addition, the evaluation metrics values are higher than that of k-NN classifier. The average accuracy score for SVM classifier is around 91%. The Result from SVM classifier after randomly varying the Training set for 20 iteration is shown above. In this case the Mean accuracy score is 90.56% and Standard Deviation is 0.012. The Mean Precision is 0.920, Mean Recall is 0.897 and Mean F1-Score is 0.908. The results when compared to without iteration is almost the same. SVM perform better even when there are outliers than that of KNN. KNN performs better than that of SVM when the number of training (m) samples are greater than the number of features (n) (m>>n).

For Guassian Naive Bayes Classifier, the evaluation metrics have all the same values of 0.88 for both the class labels. The accuracy score is 88% which is higher than k-NN and lower than that of SVM Classifier. The Result from Naïve Bayes classifier after randomly varying the Training set for 20 iteration is shown above. In this case the Mean accuracy score is 86.94% and Standard Deviation is 0.011. The Mean Precision is 0.865, Mean Recall is 0.889 and Mean F1-Score is 0.877. The results when compared to without iteration is almost the same. In this it was assumed that the features follow Gaussian Distribution.

The results obtained from Decision Tree classifier is better than that of all the other classifiers mentioned above. The f1-score for both the class labels is 0.93 which is highest among all the algorithm. The precision and recall is very similar in values and differ by 0.2 and 0.1 between two class labels for precision and recall respectively. This values suggest that Decision Tree is performing better than other mentioned algorithms in this case. The Mean accuracy score after 20 iterations of randomly varying the dataset is 93.174% and the Standard Deviation is 0.011. The Mean Precision is 0.937, Mean Recall is 0.932 and the Mean F1-Score is 0.934. Which is again similar to that of the results obtained without iterations. This values however are higher than other algorithms. Decision Trees understands colinearity better than SVM which makes them have higher performance in categorical data. Decision trees use hyper-rectangles in input space whereas, SVM uses kernel trick which makes decsion trees better in performance.