STA 6106 - Project 1 : Anmol Sureshkumar Panchal ; UID:4446829

Question 1: -

We have a dataset “Charlie.csv” which has Data Column having values 1 and -1 which are Original and New respectively with x1, x2, x3, x4 as other columns. We consider x = { x1, x2, x3, x4} and y = {Data} from the dataset “Charlie.csv”.

We generated our custom gaussian kernel by using following equation:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{y}||^2}{\sigma^2}\right)$$

In def kernel() function as shown below:

```
data = pd.read_csv("charlie.csv")
data.head()#just generates head of the dataset
x = np.array(data.iloc[:,1:5])#Columns x1,x2,x3,x4 in x as features
y = np.array(data.iloc[:,0])
sigma = 1
gamma = 0.6 #Value of C
def Kernel(x, y, sigma):
    return np.exp(-np.linalg.norm(x-y)**2 / ( (sigma ** 2)))
```

We use def omega() function which generates matrix of omega using x , y and sigma values which is **omega = y*K(xi,xj).**

```
def Omega(x, y, sigma):
    length = len(x)
    omega = np.zeros((length, length))
    for i in range(length):
        for j in range(length):
            omega[i, j] = y[i] * y[j] * Kernel(x[i], x[j], sigma)
    return omega
```

```
Omega(x, y, sigma)
```

```
array([[1.00000000e+00, 3.01530220e-01, 2.10709397e-01, 1.00838101e-02,
        2.28919590e-05, 5.67102360e-02, 1.09327574e-03, 1.76819662e-02,
        2.13555468e-01, 1.75746710e-01, 1.91218240e-01, 7.84005241e-02,
        4.51020542e-03, 6.44948200e-01, 1.91447000e-03, 6.41904566e-01,
        9.75685964e-05, 8.29608813e-03, 8.50331915e-01, 1.88083345e-03],
       [3.01530220e-01, 1.00000000e+00, 9.68778098e-01, 2.85556110e-01,
        1.80130810e-04, 4.20599607e-04, 4.76115401e-02, 8.97421404e-03,
        8.22389027e-01, 9.90058345e-02, 6.54938047e-01, 7.19908994e-02,
        2.04883891e-03, 6.00344804e-01, 5.40248183e-02, 9.08103615e-02,
        1.46053067e-04, 5.44113824e-04, 5.95440569e-01, 1.58941474e-04],
       [2.10709397e-01, 9.68778098e-01, 1.00000000e+00, 3.33401702e-01,
        1.11437490e-04, 1.80355012e-04, 8.41874003e-02, 9.92502432e-03,
        8.85252240e-01, 9.67812937e-02, 5.61521906e-01, 7.85842906e-02,
        9.64565844e-04, 5.30474024e-01, 9.20219303e-02, 6.39452450e-02,
        2.25391992e-04, 2.06251523e-04, 4.56048493e-01, 1.50063949e-04],
       [1.00838101e-02, 2.85556110e-01, 3.33401702e-01, 1.00000000e+00,
        2.61668881e-03, 4.66835690e-07, 3.35942894e-02, 5.49276677e-05,
        1.67098529e-01, 1.43292753e-03, 3.89793774e-01, 1.26355657e-03,
        1.22012362e-03, 3.50138305e-02, 2.29087254e-02, 8.32401159e-04],
```

Then we will design a classifier function def classifier() which uses x,y,sigma and gamma value by train() function to generate alpha, para and beta values.

```
def Classifier(x, y):
    para = Train(x, y, sigma, gamma)
    beta = para[0]
    alpha = para[1 : len(para)]
    return alpha, beta, para
```

```
alpha
```

```
<function __main__.alpha()>
```

```
beta
```

```
array([0.35491091])
```

```
para
```

```
array([[0.35491091],
       [0.30293236],
       [0.22055716],
       [0.50809158],
       [0.24179211],
       [0.50809094],
       [0.50809152],
       [0.2419071 ],
```

The training function is written in the logic applying by omega + identity matrix / gamma = y.y.T

```
def Train(x, y, sigma, gamma):
    length = len(x) + 1
    A = np.zeros((length, length))
    A[0][0] = 0
    A[0, 1:length] = y.T
    A[1:length, 0] = y
    A[1:length, 1:length] = Omega(x, y, sigma) + np.eye(length - 1) / gamma
    B = np.ones((length, 1))
    B[0][0] = 0
    return np.linalg.solve(A, B)
```

Then we create the Prediction function def pred() which generates predicted data from the test data . Then we Split the dataset into 75% for training and 25% for testing.

```
def Pred(x_test):
    preds = []
    temp_pred = []
    for i in range(len(x_test)):
        for j in range(len(X_train)):
            ans = alpha[j] * y_train[j] * Kernel(X_train[j], x_test[i], sigma)
            temp_pred.append(ans)
        preds.append(np.sum(temp_pred)+beta)
        temp_pred = []
    return preds, temp_pred
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)#Split the dataset into 75% for training and 25% for testing
```

```
X_train, X_test, y_train, y_test
```

```
(array([[ 0.944697,  0.504711],
        [ 0.839022,  1.469579],
        [ 0.411311, -0.21893 ],
        [ 0.294281,  0.491533],
        [ 0.291681, -0.6034  ],
        [-0.12378 , -0.08767 ],
        [-2.14662 , -1.17849 ],
        [-0.99211 ,  1.670464],
        [-0.14246 ,  0.560808],
        [ 0.197337,  0.640937],
        [ 2.36528 , -1.30494 ],
        [-0.99498 , -0.31493 ],
        [-0.27825 , -0.94763 ],
        [ 2.608666, -0.42176 ],
        [ 3.204876,  0.879172]]), array([[-1.70241 , -0.36089 ],
        [-2.65608 ,  0.135288],
        [-1.10423 ,  1.472593],
        [ 0.203271, -2.29514 ],
        [-1.2195  , -0.09129 ]]), array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64), array([1, 1, 1, 1, 1], dty
pe=int64))
```

We will now generate predicted and test data from classification of training data using classifier function on training data of x and y.

alpha, beta, para = Classifier(X_train, y_train)

pred, test = Pred(X_test)

```
pred,test
```

```
([array([0.35775112]),
  array([0.35263764]),
  array([0.36664188]),
  array([0.360435]),
  array([0.35531268]),
  array([0.24995662]),
  array([0.35491444]),
  array([0.37834558])],
 [])
```

We use def Normalization() to normalize the predicted data. We generate confusion matrix which computes confusion matrix to evaluate the accuracy of a classification. By definition a confusion matrix is such that is equal to the number of observations known to be in group but predicted to be in group .Thus in binary classification, the count of true negatives is , false negatives is , true positives is and false positives is .

#Normalization

y_pred_vals = []

for i in range(0, len(pred)):

```
    deno = 0.9*(np.sign(pred[i])*(max(pred) + min(pred)) + (max(pred) - min(pred)))
    num = 2*(np.abs(pred[i]))
    yt = 1 - (num/deno)
    if yt>0:
        y_pred_vals.append(1)
    elif yt<0:
        y_pred_vals.append(-1)
```

Then compute the F1 score, also known as balanced F-score or F-measure of the test data and predicted values by using f1_score function. F1 Score. The F1 Score is the 2*((precision*recall)/(precision+recall)). It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true = y_test, y_pred = y_pred_vals)
```

```
array([[0, 1],
       [1, 6]], dtype=int64)
```

```
f1_score(y_true = y_test, y_pred = y_pred_vals)
#Compute the F1 score, also known as balanced F-score or F-measure
```

```
0.8571428571428571
```

#Some metrics are essentially defined for binary classification tasks (e.g. f1_score). In these cases, by default only the positive label is evaluated, assuming by default that the positive class is labelled 1 (though this may be configurable through the pos_label parameter).

Code:

```
import pandas as pd
import numpy as np
import math
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
import matplotlib as mp
import matplotlib.pyplot as plt


data = pd.read_csv("charlie.csv")
data.head()#just generates head of the dataset
x = np.array(data.iloc[:,1:5])#Columns x1,x2,x3,x4 in x as features
y = np.array(data.iloc[:,0])
sigma = 1
gamma = 0.6 #Value of C
def Kernel(x, y, sigma):
    return np.exp(-np.linalg.norm(x-y)**2 / ( (sigma ** 2)))
def Omega(x, y, sigma):
    length = len(x)
    omega = np.zeros((length, length))
    for i in range(length):
        for j in range(length):
            omega[i, j] = y[i] * y[j] * Kernel(x[i], x[j], sigma)
    return omega
def Classifier(x, y):
    para = Train(x, y, sigma, gamma)
    beta = para[0]
    alpha = para[1 : len(para)]
    return alpha, beta, para
```

```
def Train(x, y, sigma, gamma):
    length = len(x) + 1
    A = np.zeros((length, length))
    A[0][0] = 0
    A[0, 1:length] = y.T
    A[1:length, 0] = y
    A[1:length, 1:length] = Omega(x, y, sigma) + np.eye(length - 1) / gamma
    B = np.ones((length, 1))
    B[0][0] = 0
    return np.linalg.solve(A, B)
def Pred(x_test):
    preds = []
    temp_pred = []
    for i in range(len(x_test)):
        for j in range(len(X_train)):
            ans = alpha[j] * y_train[j] * Kernel(X_train[j], x_test[i], sigma)
            temp_pred.append(ans)
        preds.append(np.sum(temp_pred)+beta)
        temp_pred = []
    return preds, temp_pred
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)#Split the dataset into 75% for training and 25% for testing
alpha, beta, para = Classifier(X_train, y_train)
pred, test = Pred(X_test)


#Normalization
y_pred_vals = []
for i in range(0, len(pred)):
    deno = 0.9*(np.sign(pred[i])*(max(pred) + min(pred)) + (max(pred) - min(pred)))
    num = 2*(np.abs(pred[i]))
    yt = 1 - (num/deno)
```

```
    if yt>0:
        y_pred_vals.append(1)
    elif yt<0:
        y_pred_vals.append(-1)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true = y_test, y_pred = y_pred_vals)
f1_score(y_true = y_test, y_pred = y_pred_vals)
#Compute the F1 score, also known as balanced F-score or F-measure
#Some metrics are essentially defined for binary classification tasks (e.g. f1_score). In these cases, by default only the positive label is evaluated, assuming by default that the positive class is labelled 1 (though this may be configurable through the pos_label parameter).
```

Question 2:

We have the dataset “charlie1.csv” where we will use y as Data Column but since this is one class classification SVM we will only use Original.Data Data = 1 and x as columns {z1,z2}.

```
data = pd.read_csv("charlie1.csv")
x = np.array(data.iloc[:,1:3])
y = np.array(data.iloc[:,0])
x
y
x_out = x[21:]
y_out = y[21:]
x = x[0:20]
y = y[y>0]
sigma = 1
gamma = 0.1257360775345962
```

we use this as we are only using Original(1) Dataset rows so rest with New(-1) as X_out and y_out to detect anomalies and outliers.

Now we are going to compute the rho equation :

$$\rho = \frac{1}{\mathbf{e}'\mathbf{H}^{-1}\mathbf{e}}.$$

and

$$\boldsymbol{\alpha} = \frac{\mathbf{H}^{-1}}{\mathbf{e}'\mathbf{H}^{-1}\mathbf{e}}\mathbf{e}. \tag{4}$$

where $\mathbf{H} = \mathbf{K} + \frac{1}{C}\mathbf{I}_N$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)'$ represents the column vector of Lagrange multipliers. The matrix $\mathbf{K}$ is the Gram matrix and has entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{e} = (1, \ldots, 1)'$, and $\mathbf{I}_N$ denotes the identity matrix.

We are using sigma = 1 and gamma = 0.1257360775345962 to generate most possible accuracy. We calculated gamma by taking a maximum covariance value from covariance matrix of a Gram Matrix. Gram matrix is a square matrix which has Kij entries.

```
def Gram_Matrix(x):
    K = np.zeros((len(x),len(x)))
    for i in range(0, len(x)):
        for j in range(0, len(x)):
            K[i, j] = Kernel(x[i], x[j], sigma)

    return K
Gram_Matrix(x)
```

```
Gram_Matrix(x)
```

```
array([[1.00000000e+00, 3.01530220e-01, 2.10709397e-01, 1.00838101e-02,
        2.28919590e-05, 5.67102360e-02, 1.09327574e-03, 1.76819662e-02,
        2.13555468e-01, 1.75746710e-01, 1.91218240e-01, 7.84005241e-02,
        4.51020542e-03, 6.44948200e-01, 1.91447000e-03, 6.41904566e-01,
        9.75685964e-05, 8.29608813e-03, 8.50331915e-01, 1.88083345e-03],
       [3.01530220e-01, 1.00000000e+00, 9.68778098e-01, 2.85556110e-01,
        1.80130810e-04, 4.20599607e-04, 4.76115401e-02, 8.97421404e-03,
        8.22389027e-01, 9.90058345e-02, 6.54938047e-01, 7.19908994e-02,
        2.04883891e-03, 6.00344804e-01, 5.40248183e-02, 9.08103615e-02,
        1.46053067e-04, 5.44113824e-04, 5.95440569e-01, 1.58941474e-04],
       [2.10709397e-01, 9.68778098e-01, 1.00000000e+00, 3.33401702e-01,
        1.11437490e-04, 1.80355012e-04, 8.41874003e-02, 9.92502432e-03,
        8.85252240e-01, 9.67812937e-02, 5.61521906e-01, 7.85842906e-02,
        9.64565844e-04, 5.30474024e-01, 9.20219303e-02, 6.39452450e-02,
        2.25391992e-04, 2.06251523e-04, 4.56048493e-01, 1.50063949e-04],
       [1.00838101e-02, 2.85556110e-01, 3.33401702e-01, 1.00000000e+00,
        2.61668881e-03, 4.66835690e-07, 3.35942894e-02, 5.49276677e-05,
        1.67098529e-01, 1.43292753e-03, 3.89793774e-01, 1.26355657e-03,
        1.22012362e-03, 3.50138305e-02, 2.29087254e-02, 8.32401159e-04,
        8.34802670e-07, 4.41720128e-05, 4.81227932e-02, 1.21138953e-07],
       [2.28919590e-05, 1.80130810e-04, 1.11437490e-04, 2.61668881e-03,
        1.00000000e+00, 5.14166310e-09, 1.19703328e-08, 7.47684915e-12,
        1.22987376e-05, 5.25193106e-09, 5.25471162e-03, 1.22920345e-09,
        1.29006765e-01, 6.05434586e-06, 6.06614210e-09, 1.91309696e-07,
```

Then we calculate the H,K,alpha and rho using the given equations.

```
def H(x):
    mat = np.zeros((len(x), len(x)))
    mat[0:len(x), 0:len(x)] = Gram_Matrix(x) + np.eye(len(x))/gamma
    return mat
```

```
H(x)
```

```
array([[2.66666667e+00, 4.03840951e-10, 3.97625480e-04, 4.21553451e-15,
        2.33233998e-22, 3.66231570e-07, 1.71717542e-08, 1.13666172e-07,
        1.93545100e-04, 8.54349510e-02, 2.32916669e-06, 1.87824977e-04,
        2.49805033e-06, 3.23033256e-01, 5.43720878e-09, 4.47316249e-07,
        1.54176130e-05, 3.84877640e-03, 1.37049253e-02, 4.42022810e-14,
        2.02419114e-02, 2.14527090e-09, 2.32631845e-21, 5.41653948e-23,
        3.65534800e-14, 7.09547416e-23, 7.87163536e-09, 3.66605725e-19,
        1.82994691e-21, 8.96749353e-13],
       [4.03840951e-10, 2.66666667e+00, 5.31397622e-04, 1.03847706e-03,
        4.87083722e-09, 3.98438353e-28, 5.06789460e-05, 1.17127822e-07,
        6.60452671e-03, 4.93252452e-10, 1.41223024e-02, 1.04528226e-21,
        3.63773592e-16, 1.15105711e-08, 2.52882629e-03, 6.36069649e-31,
        1.39646854e-17, 6.04019335e-11, 1.18264716e-03, 3.21740105e-42,
        1.40542588e-04, 2.00432645e-33, 1.60883539e-06, 4.10572608e-28,
        3.51082457e-10, 3.03271161e-23, 1.40829902e-28, 1.58020216e-30,
        1.31487827e-44, 8.36692935e-21],
       [3.97625480e-04, 5.31397622e-04, 2.66666667e+00, 8.19009474e-05,
        8.57144237e-11, 1.37591198e-19, 4.20035979e-02, 2.89581282e-09,
        3.60594940e-01, 2.89482733e-05, 2.14381101e-01, 1.77864338e-10,
```

```
def alpha():

   a = np.dot(np.divide(np.linalg.inv(H_mat),np.dot(np.dot(e.T, np.linalg.inv(H_mat)),e)),e)

   return a
```

```
[0.06327363 0.07324714 0.05527159 0.05249269 0.06046767 0.05279316
 0.08041784 0.07935801 0.05638258 0.05436132 0.07779936 0.07003023
 0.06716941 0.07646087 0.0804745 ]
```

```
 def rho():

   p = np.divide(1, np.dot(np.dot(e.T, np.linalg.inv(H_mat)), e))

   return p
```

```
0.7312403505181224
```

Then we Split the dataset into 75% for training and 25% for testing.

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)
```

```
X_train, X_test, y_train, y_test
```

```
(array([[11.5, 21.8, 19.3, 12.1],
        [ 9.5, 19.6, 13.6, 14.5],
        [ 9.8, 25.8, 14.8, 15. ],
        [12.6, 23.9, 17.1, 14.2],
        [ 8.9, 19. ,  8.5, 14.7],
        [ 9.9, 20. , 15.4, 15.9],
        [10.5, 19.9, 18.1, 14.8],
        [12.8, 26.3, 13.5, 13.7],
        [ 9.7, 20.1, 10. , 16.6],
        [11.3, 21.6, 14. , 18.7],
        [ 9.5, 19.3, 15.3, 12.2],
        [ 9.9, 23.7, 11.9, 18.1],
        [14.9, 25. , 16.3, 16.6],
        [10. , 20.7, 13.6, 15.5],
        [ 9.7, 20. , 16.1, 16.5],
        [15.9, 24.6, 14.7, 15.3],
        [10.5, 20.3, 17. , 16.5],
        [ 8.7, 19. ,  9.9, 16.8],
        [10.1, 19.4, 16.2, 15.8],
        [10. , 19.8, 14. , 15.9],
        [ 8.7, 18.8, 16.9, 16.8],
        [11.9, 21.8, 14.1, 16.2]]), array([[10.3, 20.5, 15.6, 15.1],
        [ 9.2, 19. , 11.5, 16.3],
        [ 8.5, 19.2, 17.4, 15.8],
        [11.7, 21.5, 19.8, 18.3],
```

Then we generate prediction values using pred(data) function and then normalizing it so that outliers are mapped correctly while plotting. We use points from “new” to check the accuracy of your prediction. And to perform a prediction. The classification function is given in equation (5) below:

$$y_t = b + \sum_{j=1}^{N} a_j K(\mathbf{x}_t, \mathbf{x}_j) \qquad (5)$$

has to be normalized in this case to yield useful results:

$$\bar{y}_t = 1 - \frac{2\,|y_t|}{C\,(\mathrm{sgn}(y_t)(y_{max} + y_{min}) + (y_{max} - y_{min}))} \qquad (6)$$

e = np.ones(len(X_train))

H_mat = H(X_train)

```
a = alpha()
rho = rho()
def preds(data):
    p = []
    temp = []
    for i in range(len(data)):
        for j in range(len(X_train)):
            temp.append(a[j]*Kernel(data[i], X_train[j], sigma))
        p.append(sum(temp)+rho)
        temp = []
```

We use def Normalization() to normalize the predicted data. We generate confusion matrix which computes confusion matrix to evaluate the accuracy of a classification. By definition a confusion matrix is such that is equal to the number of observations known to be in group but predicted to be in group .Thus in binary classification, the count of true negatives is , false negatives is , true positives is and false positives is .

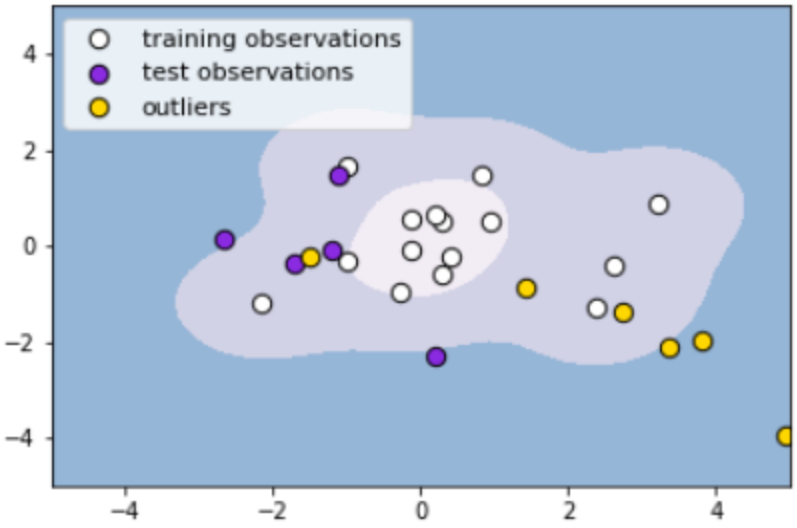
```
    #Normalization
    y_pred = []
    Z = []
    for i in range(0, len(p)):
        denom = gamma*(np.sign(p[i])*(max(p) + min(p)) + (max(p) - min(p)))
        num = 2*(np.abs(p[i]))
        yt = 1 - (num/denom)
        Z.append(yt)
        if yt>0:
            y_pred.append(1)
        elif yt<0:
            y_pred.append(-1)
    return y_pred, Z
```

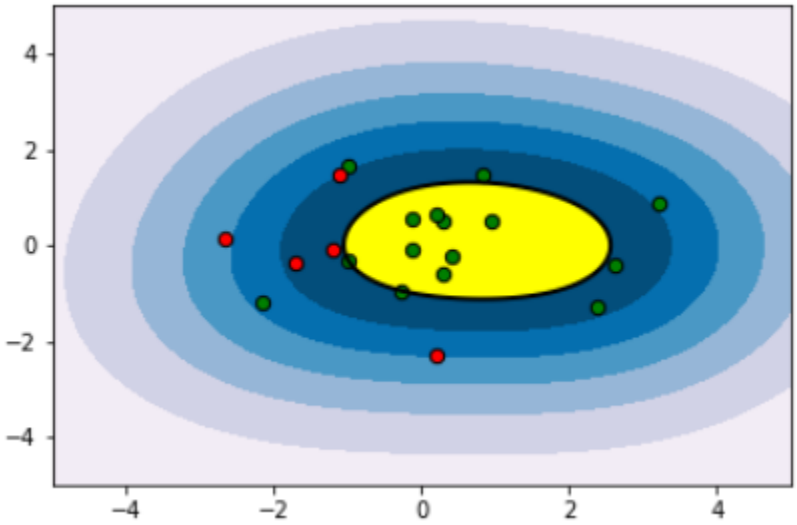
We plot the predicted test data and predicted data by using One class SVM scikit learn with gaussian kernel on z1 and z2 with Data (1) values. We detect outliers and we follow the given below code to plot the contour plots.

```
p, Z = preds(x_out)
```

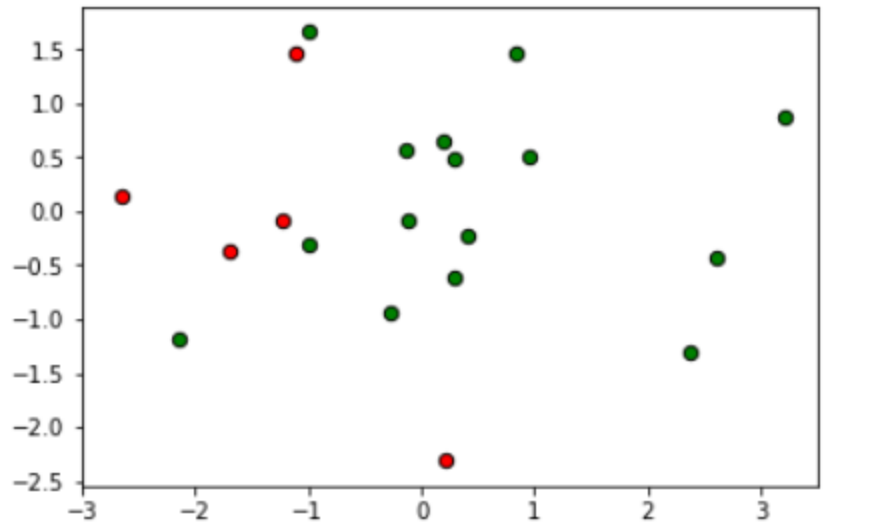
```
p = np.array(p)
n_error_outliers = p[p == 1].size
print("Number of errors = ",n_error_outliers,"/",y_out.size)
xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))
p, Z = preds(np.c_[xx.ravel(), yy.ravel()])
Z = np.array(Z).reshape(xx.shape)
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.PuBu)
a = plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='darkred')

s = 70
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='white', s=s, edgecolors='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='blueviolet', s=s,
             edgecolors='k')
c = plt.scatter(x_out[:, 0], x_out[:, 1], c='gold', s=s,
            edgecolors='k')

plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.legend([b1, b2, c],
        [ "training observations",
         "test observations", "outliers"],
        loc="upper left",
        prop=mp.font_manager.FontProperties(size=11))
plt.show()
from sklearn import svm
clf = svm.OneClassSVM(kernel = 'rbf', gamma = gamma)
clf.fit(X_train, y_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_test
n_error_train = y_pred_train[y_pred_train == -1].size
```

```
n_error_test = y_pred_test[y_pred_test == -1].size
xx, yy = np.meshgrid(np.linspace(-5, 5, 500), np.linspace(-5, 5, 500))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
clf.predict(X_test)
y_pred_test
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.PuBu)
a = plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.contourf(xx, yy, Z, levels=[0, Z.max()], colors='yellow')

s = 40
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='green', s=s, edgecolors='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='red', s=s,
            edgecolors='k')
plt.axis('tight')
plt.xlim((-5, 5))
plt.ylim((-5, 5))

plt.show()
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='green', s=s, edgecolors='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='red', s=s, edgecolors='k')
plt.show()
```





Here in above image red dot represents the outliers and green dots as proper data.



Here in above image red dot represents the outliers and green dots as proper data.

Code:

```
data = pd.read_csv("charlie1.csv")

x = np.array(data.iloc[:,1:3])

y = np.array(data.iloc[:,0])

x

y

x_out = x[21:]

y_out = y[21:]

x = x[0:20]

y = y[y>0]

sigma = 1

gamma = 0.1257360775345962

def Gram_Matrix(x):
    K = np.zeros((len(x),len(x)))
    for i in range(0, len(x)):
        for j in range(0, len(x)):
            K[i, j] = Kernel(x[i], x[j], sigma)
```

```
    return K
Gram_Matrix(x)
np.max(np.cov(Gram_Matrix(x)))
def H(x):
    mat = np.zeros((len(x), len(x)))
    mat[0:len(x), 0:len(x)] = Gram_Matrix(x) + np.eye(len(x))/gamma
    return mat
def alpha():
    a = np.dot(np.divide(np.linalg.inv(H_mat),np.dot(np.dot(e.T, np.linalg.inv(H_mat)),e)),e)
    return a
def rho():
    p = np.divide(1, np.dot(np.dot(e.T, np.linalg.inv(H_mat)), e))
    return p
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.25)
e = np.ones(len(X_train))
H_mat = H(X_train)
a = alpha()
rho = rho()
def preds(data):
    p = []
    temp = []
    for i in range(len(data)):
        for j in range(len(X_train)):
            temp.append(a[j]*Kernel(data[i], X_train[j], sigma))
        p.append(sum(temp)+rho)
        temp = []

    #Normalization
    y_pred = []
    Z = []
```

```
    for i in range(0, len(p)):
        denom = gamma*(np.sign(p[i])*(max(p) + min(p)) + (max(p) - min(p)))
        num = 2*(np.abs(p[i]))
        yt = 1 - (num/denom)
        Z.append(yt)
        if yt>0:
            y_pred.append(1)
        elif yt<0:
            y_pred.append(-1)
    return y_pred, Z
p, Z = preds(x_out)
p = np.array(p)
n_error_outliers = p[p == 1].size
print("Number of errors = ",n_error_outliers,"/",y_out.size)
xx, yy = np.meshgrid(np.linspace(-5, 5, 50), np.linspace(-5, 5, 50))
p, Z = preds(np.c_[xx.ravel(), yy.ravel()])
Z = np.array(Z).reshape(xx.shape)
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.PuBu)
a = plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='darkred')

s = 70
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='white', s=s, edgecolors='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='blueviolet', s=s,
                 edgecolors='k')
c = plt.scatter(x_out[:, 0], x_out[:, 1], c='gold', s=s,
                edgecolors='k')

plt.xlim((-5, 5))
plt.ylim((-5, 5))
plt.legend([b1, b2, c],
           [ "training observations",
```

```
          "test observations", "outliers"],
          loc="upper left",
          prop=mp.font_manager.FontProperties(size=11))
plt.show()
from sklearn import svm
clf = svm.OneClassSVM(kernel = 'rbf', gamma = gamma)
clf.fit(X_train, y_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
y_pred_test
n_error_train = y_pred_train[y_pred_train == -1].size
n_error_test = y_pred_test[y_pred_test == -1].size
xx, yy = np.meshgrid(np.linspace(-5, 5, 500), np.linspace(-5, 5, 500))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
clf.predict(X_test)
y_pred_test
plt.contourf(xx, yy, Z, levels=np.linspace(Z.min(), 0, 7), cmap=plt.cm.PuBu)
a = plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
plt.contourf(xx, yy, Z, levels=[0, Z.max()], colors='yellow')

s = 40
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='green', s=s, edgecolors='k')
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='red', s=s,
            edgecolors='k')
plt.axis('tight')
plt.xlim((-5, 5))
plt.ylim((-5, 5))

plt.show()
b1 = plt.scatter(X_train[:, 0], X_train[:, 1], c='green', s=s, edgecolors='k')
```

```
b2 = plt.scatter(X_test[:, 0], X_test[:, 1], c='red', s=s, edgecolors='k')

plt.show()
```