Submitted by

ADITYA MISHRA

SEC-C , ROLL NO-56

REG NO-201800612

Intelligent Systems Lab Assignment – 6

Question: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

ON 1st DATA SET

Source Code:

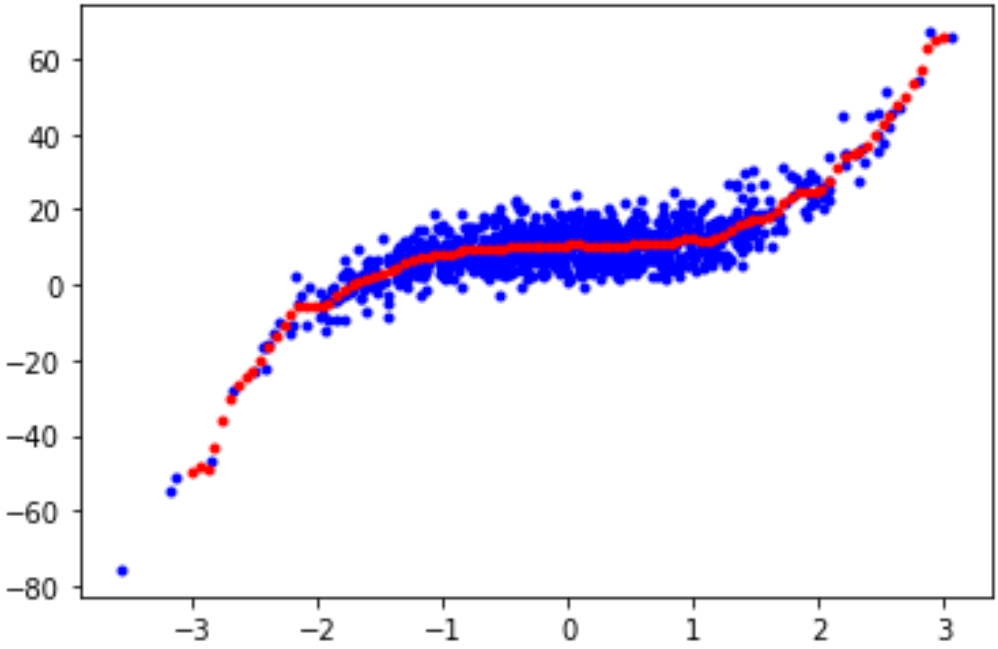
```
import numpy as np import
matplotlib.pyplot as plt
np.random.seed(8)
X = np.random.randn(1000,1)
y = 2*(X**3) + 10 + 4.6*np.random.randn(1000,1)
 # Weight Matrix in code. It is a diagonal matrix.
def wm(point, X, tau):

  # tau --> bandwidth
# X --> Training data.
  # point --> the x where we want to make the prediction.

  # m is the No of training examples .
    m = X.shape[0]

  # Initialising W as an identity matrix.
```

```
 w = np.mat(np.eye(m))

  # Calculating weights for all training examples [x(i)'s].
 for i in range(m):
        xi = X[i]
 d = (-2 * tau * tau)
 w[i, i] = np.exp(np.dot((xi-point), (xi-point).T)/d)
return w def predict(X, y, point, tau):

   # m = number of training examples.
 m = X.shape[0]

   # Appending a cloumn of ones in X to add the bias term.
 # Just one parameter: theta, that's why adding a column of ones
# to X and also adding a 1 for the point where we want to
# predict.
    X_ = np.append(X, np.ones(m).reshape(m,1), axis=1)

   # point is the x where we want to make the prediction.
 point_ = np.array([point, 1])

   # Calculating the weight matrix using the wm function we wrote   earlier.
    w = wm(point_, X_, tau)

  # Calculating parameter theta using the formula.
    theta = np.linalg.pinv(X_.T*(w * X_))*(X_.T*(w * y))

  # Calculating predictions.
 pred = np.dot(point_, theta)

   # Returning the theta and predictions
return theta, pred
```

```
def plot_predictions(X, y, tau, nval):
  # X --> Training data.
  # y --> Output sequence.
  # nval --> number of values/points for which we are going to
predict.
  # tau --> the bandwidth.
   # The values for which we are going to predict.
  # X_test includes nval evenly spaced values in the domain of X.
   X_test = np.linspace(-3, 3, nval)

  # Empty list for storing predictions.
 preds = []

  # Predicting for all nval values and storing them in preds.
 for point in X_test:
theta, pred = predict(X, y, point, tau)
preds.append(pred)

  # Reshaping X_test and preds
   X_test = np.array(X_test).reshape(nval,1)
preds = np.array(preds).reshape(nval,1)
  # Plotting
plt.plot(X, y, 'b.')
 plt.plot(X_test, preds, 'r.') # Predictions in red color.
  plt.show()
plot_predictions(X, y, 0.08, 100)
```

Expected Output

Graph:



ON 2nd SET OF DATA:

Source Code:

```
import matplotlib.pyplot as plt
import pandas as pd import numpy
as np
 def kernel(point, xmat, k):
m,n = np.shape(xmat)
weights = np.mat(np.eye((m)))
for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
 return weights
 def localWeight(point, xmat, ymat, k):
wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
return W
def localWeightRegression(xmat, ymat, k):
    m,n = np.shape(xmat)
ypred = np.zeros(m)
 for i in range(m):
    ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
return ypred
# load data points
 data = pd.read_csv('10-dataset.csv')
bill = np.array(data.total_bill)
tip = np.array(data.tip)

#preparing and add 1 in bill
mbill = np.mat(bill)
mtip = np.mat(tip)
```

```
 m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))
 #set k here
ypred = localWeightRegression(X,mtip,0.5)
SortIndex = X[:,1].argsort(0)
 xsort = X[SortIndex][:,0]
 fig = plt.figure() ax = fig.add_subplot(1,1,1)
 ax.scatter(bill,tip, color='green')
ax.plot(xsort[:,1],ypred[SortIndex], color = 'red', linewidth=5)
plt.xlabel('Total bill')
plt.ylabel('Tip')
 plt.show();
```

Expected Output

Graph: