Computational Statistics Assignment

By :

Raja Kumar Dubey
15MA20033

INDEX

1.a – PCA Regression

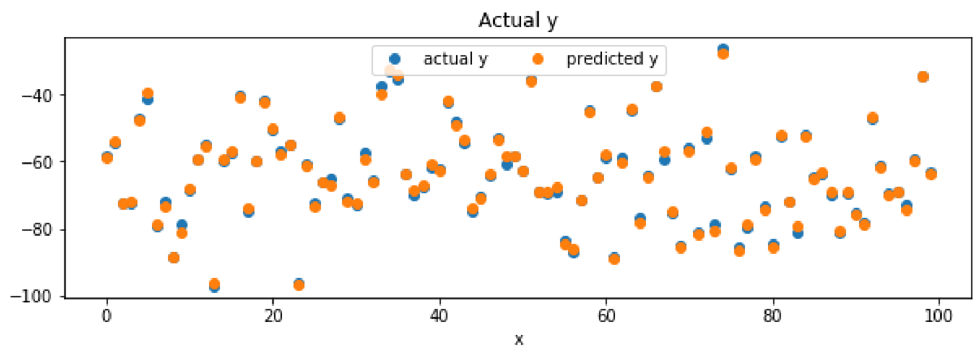
The data contained 4-dimensional input and 1 dimensional output.

1.The least square estimate gave following results-

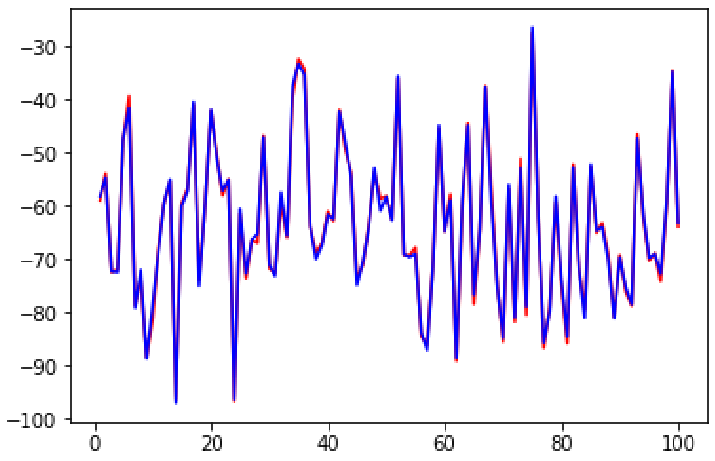
RMSE = `0.938995291966`

Coefficients = `[ b0= 3.31034229, b1= -11.55135449, b2 = -7.23717124, b3 = -14.41072607, b4 = 13.80669419])`

`Plot of predicted and actual values:`



So estimator



2. Checking for Multicollinearity:

condition numbers are

```
5.16558698417 9.39286938742 66.7547028642 2145.30406831

So clearly the Multicollinearity is present.
```

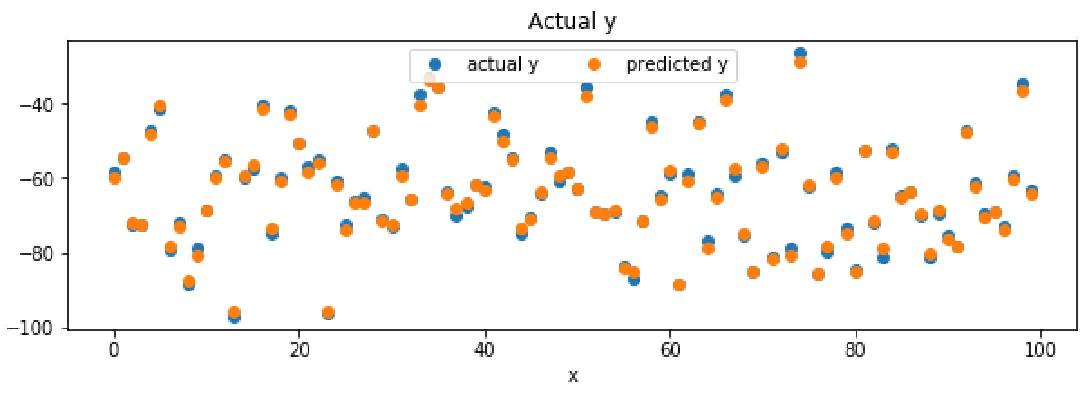
3. So we decide to remove it , the variance explained are

```
0.76016785312
0.907327867161
0.988258179483
0.999645659623
1.0

so lets take take first 3 singular values and divide matrix U and V
accordingly. For U we take first 60 column vectors and first 3 row vector
for V.

4. Now we generate the estimator using SVD. We divide the above matrix into
noise and non-noise part. And the estimator gave following results
```

RMSE = 1.0952521979

```
coefficients = [-0.42664246,  1.82105009, -6.94679601, -1.30434264,
0.51410469] (b0-b4)
```

Plots are as follows:

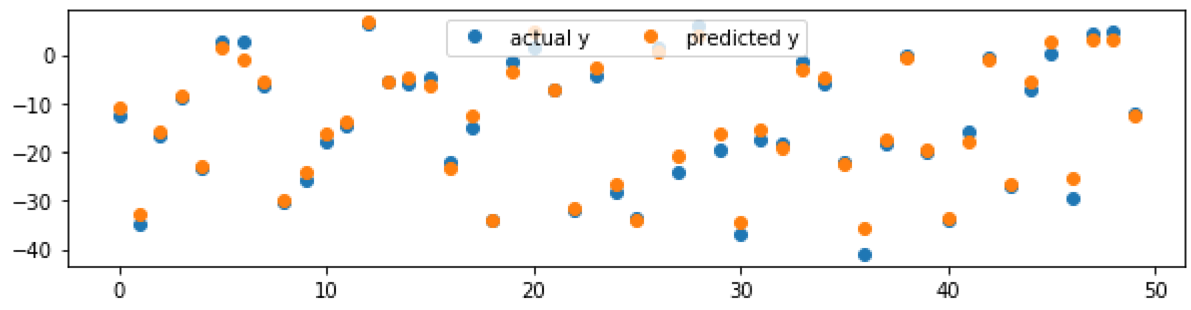


So clearly PCR estimator had almost same RMSE but removed Multicollinearity

1.b – RLS Regression

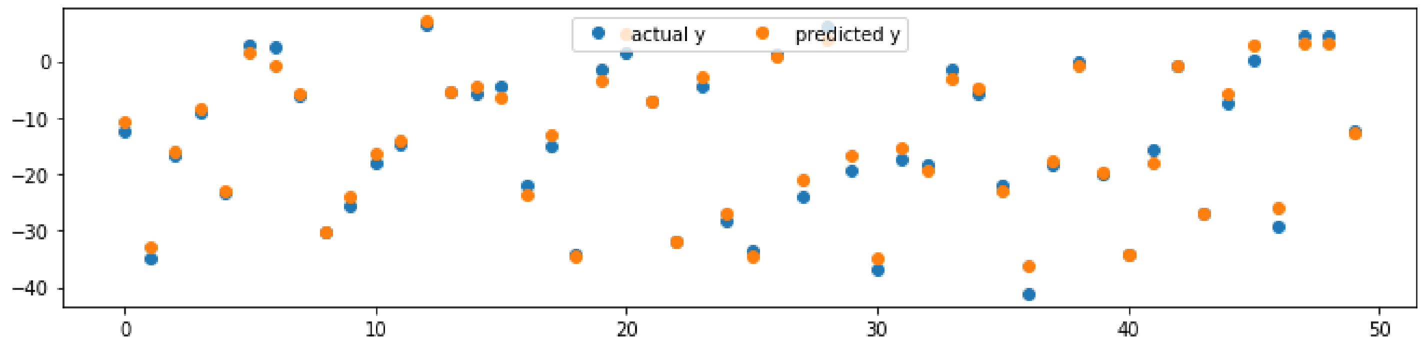
RMSE from Least square estimator = `1.82303968121`
coefficients = `array([[-12.56629909],[  0.34782342],[ -0.7261294 ]])(b0-b2)`

```
The plot from Least square model is:
```

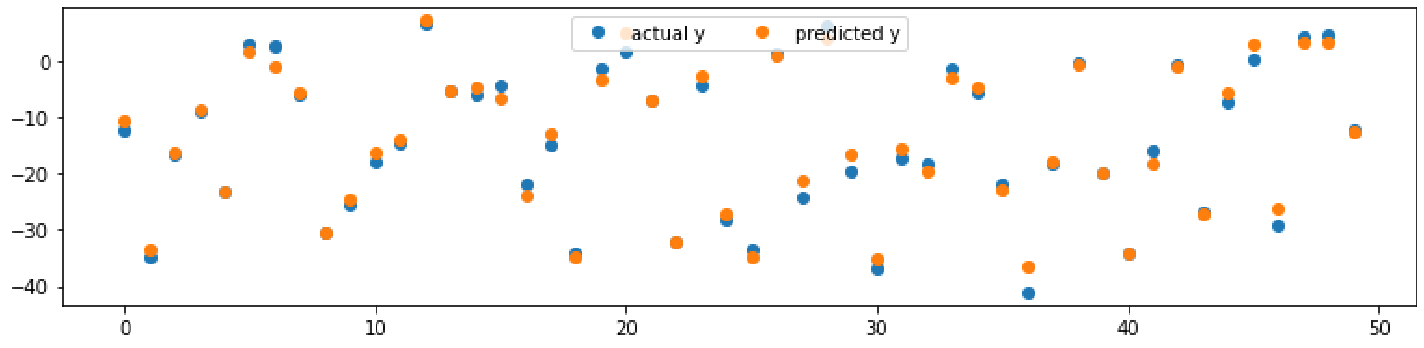


Result from Recursive least square model is as follows:

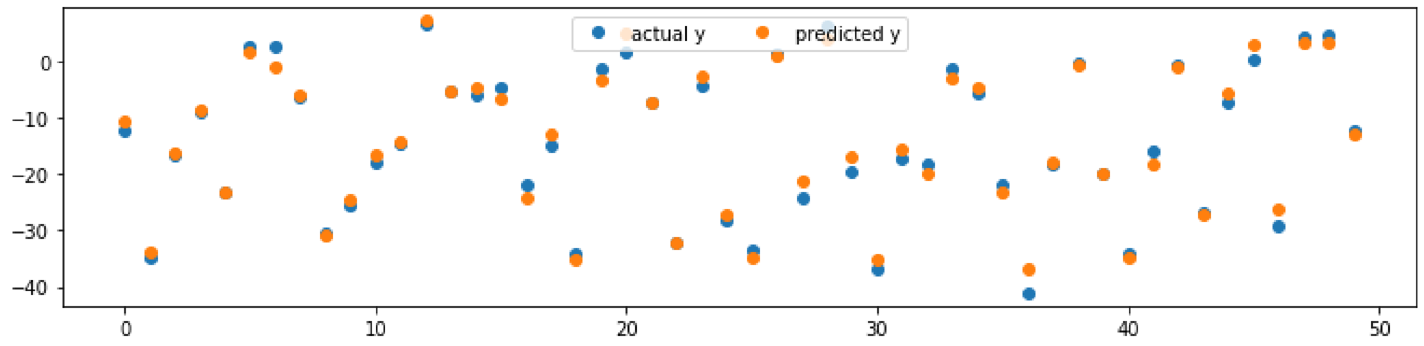
```
With lambada as forgetting factor
```
a.
```
This is for lambada value 1
The error is 1.75388245306
The graphical description is as follows:
```



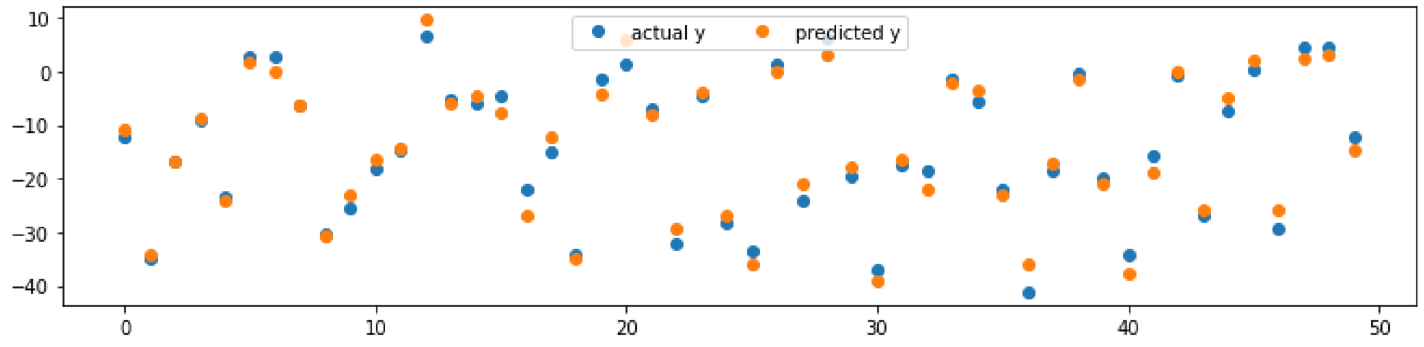
b.
```
This is for lambada value 0.99
The error is 1.68946133833
The graphical description is as follows:
```



c.
```
This is for lambada value 0.95
The error is 1.67315242421
The graphical description is as follows:
```



```
d.
This is for lambada value 0.2
The error is 2.26474148773
The graphical description is as follows:
```



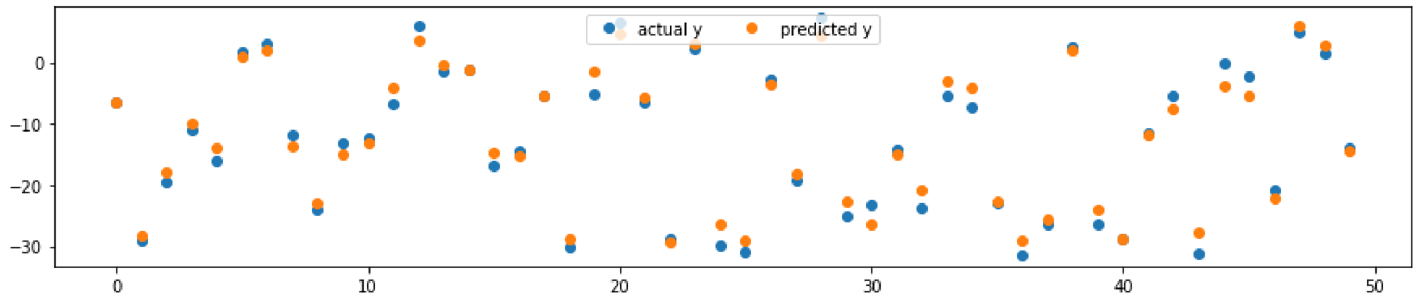
So clearly the performance(RMSE wise) was

lambada = 0.95 > lambada = 0.99 >lambada = 1 > Least square > lambada = 0.2

1.c – RLS Regression

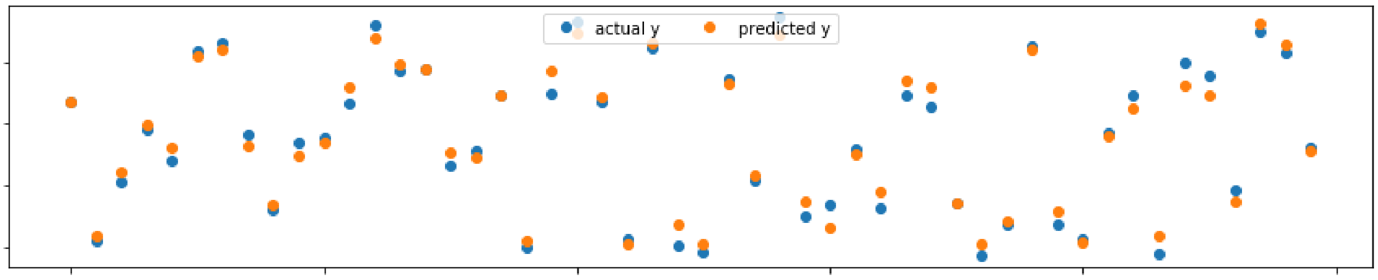
RMSE from Least square estimator = `1.89282800605`
coefficients = `array([[-11.96449068],[ -0.01691908],[ -0.76234721]])(b0-b2)`

```
The plot from Least square model is:
```

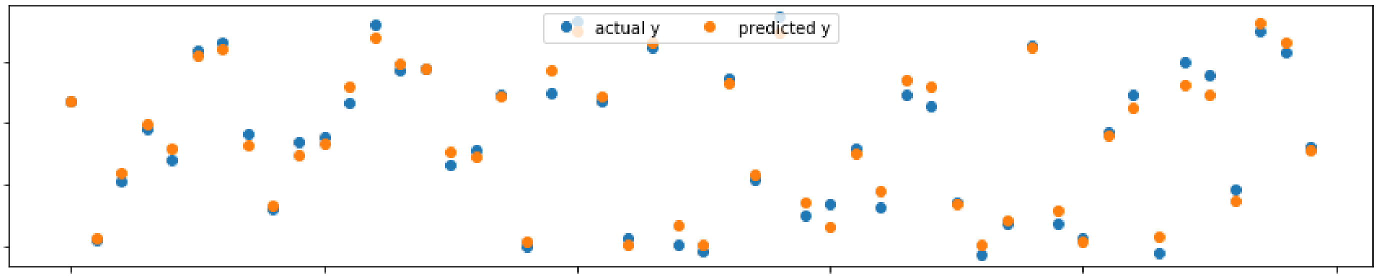


Result from Recursive least square model is as follows：

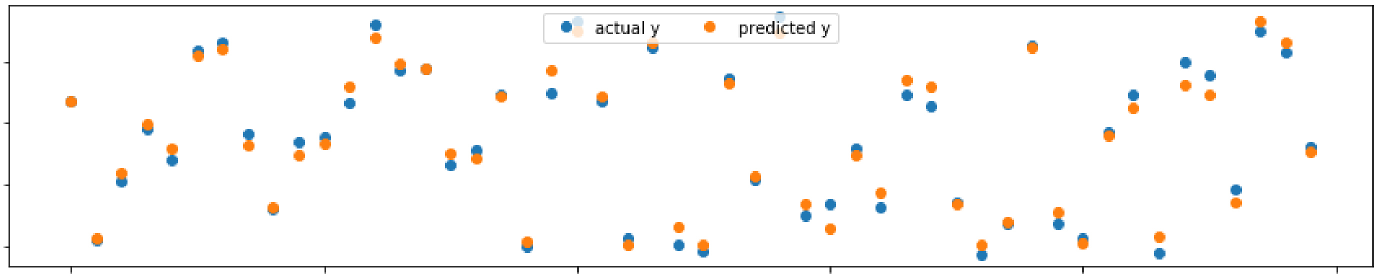
```
With lambada as forgetting factor
a.
This is for lambada value 1
The error is 1.86971133096
The graphical description is as follows:
```



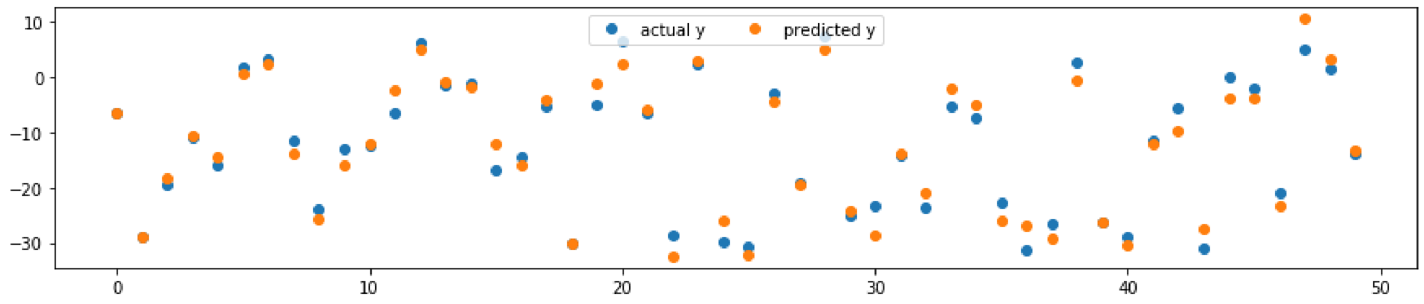
```
b.
This is for lambada value 0.99
The error is 1.82757110812
The graphical description is as follows:
```



```
c.
This is for lambada value 0.95
The error is 1.82592998912
The graphical description is as follows:
```



```
d.
This is for lambada value 0.2
The error is 2.56410119291
The graphical description is as follows:
```



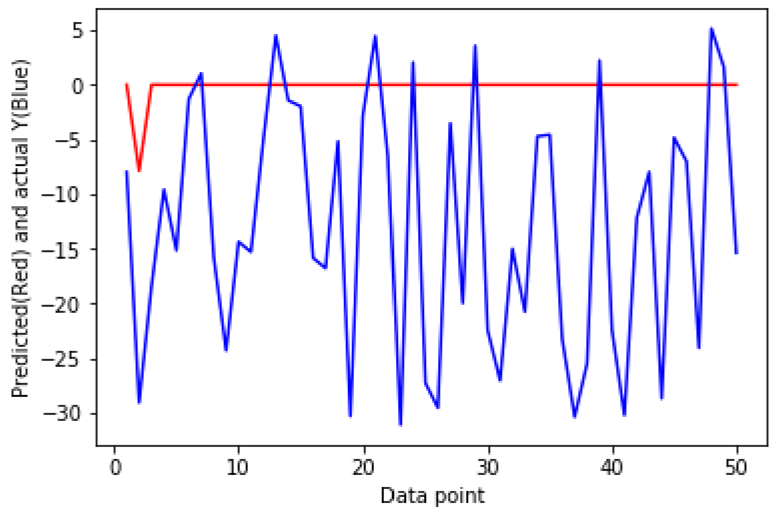
So clearly the performance(RMSE wise)was

lambada = 0.95 > lambada = 0.99 >lambada = 1 > Least square>lambada = .2

1.d – LW Regression

1.
For the first function we selected the points having weights 1 and used them to predict the particular data point.

RMSE without any tuning = `17.0141325039`



We modeled for different values insteat of 1and got following RMSE

```
D = ([ 10.0827665 ,  14.72855027,  18.81187423,  22.31142199,25.91781807,
29.12573541,  33.48873716,  38.32762274,43.63625574,  61.54754169])

Respective RMSE
RMSE =
6.04760682445
0.951587715293
0.904875682561
0.860949097348
0.849865035695
0.841473279668
0.820338051881
0.803163881336
0.797752716068
```

0.799837755533

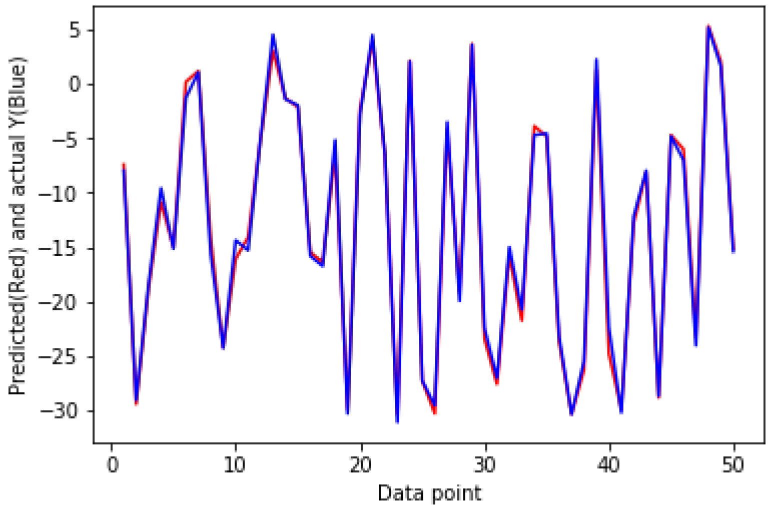
The initial distance used for weight is performing bad.
Clearly by incresing the minimum distance for weight the model is performing good.

2. For second weighting function.

Due to explosion of weights I had to normalize the weight square by 2 before taking it's exponent.
RMSE obtained = 0.798647576896

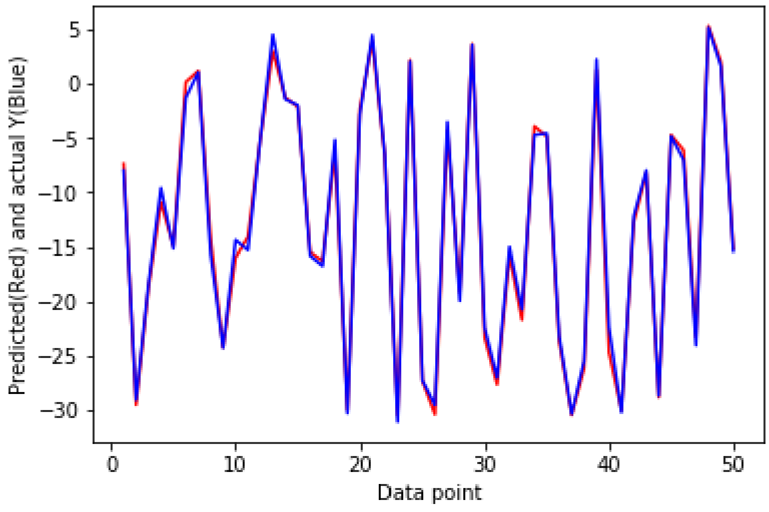
So clearly the model performed slightly better than Least square(RMSE =0.0.799837755533)

The plot is as given below:



3. For the third weighting function:

We again used the normalisation before using the weights.
The RMSE obtained is by far the best.
RMSE = 0.0.780428369406 better that LS(RMSE =0.799837755533)



2. Code

2.a – PCA Regression

```
import pandas as pd
import numpy as np
data = pd.read_csv("/home/stark/Downloads/cs_ass1/x1234y.csv")
input1 = np.array(data)
input1[:,0] = 1
ytest,ytrain = input1[:100:,5],input1[100:,5]
xtest,xtrain = input1[:100,0:5],input1[100:,0:5]
from numpy.linalg import inv
a = (xtrain.T).dot(xtrain)
ainv = inv(a)
a1 = ainv.dot(xtrain.T)
b = a1.dot(ytrain)
ycap = xtest.dot(b)
import matplotlib.pyplot as plt
%matplotlib inline
plt.subplot(3, 1, 1)
plt.title('Actual y')
plt.xlabel('x')
plt.subplot(3, 1, 1)
plt.plot(ytest, 'o', label='actual y')
plt.plot(ycap, 'o', label='predicted y')
plt.subplot(3, 1, 1)
plt.legend(loc='upper center', ncol=4)
plt.gcf().set_size_inches(10, 10)
plt.show()
print(np.sqrt(np.mean((ytest-ycap)**2)))
U, s, V = np.linalg.svd(xtrain, full_matrices=True)
S = np.zeros(xtrain.shape)
S[:5,:5] = np.diag(s)
print(s[0]/s[1],s[0]/s[2],s[0]/s[3],s[0]/s[4])
print(s[0]/np.sum(s))
print((s[0]+s[1])/np.sum(s))
print((s[0]+s[1]+s[2])/np.sum(s))
print((s[0]+s[1]+s[2]+s[3])/np.sum(s))
print((s[0]+s[1]+s[2]+s[3]+s[4])/np.sum(s))
U1 = U[:,:60]
S1 = S[:60,:3]
V1 = V[:3,:]
U2 = U[:,60:]
S2 = S[60:,3:]
V2 = V[3:,:]
usefull = np.dot(U1,np.dot(S1,V1))
noise = np.dot(U2,np.dot(S2,V2))
w = np.dot(xtrain,V1.T)
r1 = np.dot(w.T,w)
r2 = inv(r1)
r3 = np.dot(r2,w.T)
r = np.dot(r3,ytrain)
```

```
bnew = np.dot(V1.T,r)
ycap1 = np.dot(xtest,bnew)
print(np.sqrt(np.mean((ytest-ycap1)**2)))
import matplotlib.pyplot as plt
%matplotlib inline
plt.subplot(3, 1, 1)
plt.title('Actual y')
plt.xlabel('x')
plt.subplot(3, 1, 1)
plt.plot(ytest, 'o', label='actual y')
plt.plot(ycap1, 'o', label='predicted y')
plt.subplot(3, 1, 1)
plt.legend(loc='upper center', ncol=4)
plt.gcf().set_size_inches(10, 10)
plt.show()
```

2.b – RLS Regression

```
import pandas as pd
import numpy as np
input_train = pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Input_Training.csv")
output_train =
pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Output_Training.csv",header =
None)
input_test = pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Input_Test.csv")
output_test =
pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Output_Test.csv",header = None)
input_train = np.array(input_train)
input_train[:,0] = 1
output_train = np.array(output_train)
input_test = np.array(input_test)
input_test[:,0] = 1
output_test = np.array(output_test)
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots()
x = list(range(1,51))
y = output_train
ax.plot(x,y)
fig.suptitle('test title')
ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')
ax.xaxis.label.set_size(20)
plt.draw()
#generating Least square model
from numpy.linalg import inv
a = (input_train.T).dot(input_train)
ainv = inv(a)
a1 = ainv.dot(input_train.T)
b = a1.dot(output_train)
ycap = input_test.dot(b)
e = np.sqrt(np.sum(np.square(output_test-ycap))/50)
```

```
print(e)
import matplotlib.pyplot as plt
%matplotlib inline

plt.title('Actual y')
plt.xlabel('x')
plt.subplot(3, 1, 1)
plt.plot(output_test, 'o', label='actual y')
plt.plot(ycap, 'o', label='predicted y')
plt.subplot(3, 1, 1)
plt.legend(loc='upper center', ncol=4)
plt.gcf().set_size_inches(10, 8)
plt.show()
inp = np.zeros([100,3])
out = np.zeros([100,1])
inp[:50,:] = input_train
inp[50:,:] = input_test
out[:50,:] = output_train
out[50:,:] = output_test
#inp = np.array(inp,dtype=np.float32)
#out = np.array(out,dtype=np.float32)
y_pred = np.zeros([50,1])
import matplotlib.pyplot as plt
%matplotlib inline
lam = [1,0.99,0.95,0.2]
a = (input_train.T).dot(input_train)
ainv = inv(a)
a1 = ainv.dot(input_train.T)
b0 = a1.dot(output_train)
y_pred[0] = inp[50].dot(b0)
M0 = ainv
for l in lam:
   for i in range(50):
      D1 = l
      D2 = inp[50+i].dot(M0)
      D2 = D2.dot(inp[50+i].T)
      D = D1+D2
      N = M0.dot(inp[50+i].T)
      N = N.dot(inp[50+i])
      N = np.dot(N,M0)
      M1 = M0 - N/D
      M1 = M1/D1
      C = M1.dot(inp[50+i].T)
      P = out[50+i]-np.dot(inp[50+i],b0)
      T2 = C*P
      TR = np.reshape(T2,b0.shape)
      b1 = b0 + TR
      if(i!=49):
         y_pred[i+1] = inp[51+i].dot(b1)
      b0 = b1
      M0 = M1
   e = np.sqrt(np.sum(np.square(output_test-y_pred))/50)
```

```
    print("This is for lambada value "+str(l))
    print("The error is "+str(e))
    print("The graphical description is as follows:")
    plt.title('Actual y')
    plt.xlabel('x')
    plt.subplot(3, 1, 1)
    plt.plot(output_test, 'o', label='actual y')
    plt.plot(y_pred, 'o', label='predicted y')
    plt.subplot(3, 1, 1)
    plt.legend(loc='upper center', ncol=4)
    plt.gcf().set_size_inches(13, 10)
    plt.show()
```

2.c – RLS Regression

```
import pandas as pd
import numpy as np
input_train = pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Input_Training.csv")
output_train =
pd.read_csv("/home/stark/Downloads/cs_ass2/Problem2_Output_Training.csv",header =
None)
input_test = pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Input_Test.csv")
output_test =
pd.read_csv("/home/stark/Downloads/cs_ass2/Problem2_Output_Test.csv",header = None)
input_train = np.array(input_train)
input_train[:,0] = 1
output_train = np.array(output_train)
input_test = np.array(input_test)
input_test[:,0] = 1
output_test = np.array(output_test)
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
x = list(range(1,51))
y = output_train
ax.plot(x,y)
fig.suptitle('test title')
ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')
ax.xaxis.label.set_size(20)
plt.draw()
#plotting input
import matplotlib.pyplot as plt
%matplotlib inline

plt.title('Actual y')
plt.xlabel('x')
plt.subplot(3, 1, 1)
plt.plot(output_train, 'o', label='actual y')
plt.legend(loc='upper center', ncol=4)
plt.gcf().set_size_inches(15, 10)
plt.show()
```

```
#generating Least square model
from numpy.linalg import inv
a = (input_train.T).dot(input_train)
ainv = inv(a)
a1 = ainv.dot(input_train.T)
b = a1.dot(output_train)
ycap = input_test.dot(b)
e = np.sqrt(np.sum(np.square(output_test-ycap))/50)
print(e)
import matplotlib.pyplot as plt
%matplotlib inline

plt.title('Actual y')
plt.xlabel('x')
plt.subplot(3, 1, 1)
plt.plot(output_test, 'o', label='actual y')
plt.plot(ycap, 'o', label='predicted y')
plt.subplot(3, 1, 1)
plt.legend(loc='upper center', ncol=4)
plt.gcf().set_size_inches(15, 10)
plt.show()
# Recursive least square algorithm
inp = np.zeros([100,3])
out = np.zeros([100,1])
inp[:50,:] = input_train
inp[50:,:] = input_test
out[:50,:] = output_train
out[50:,:] = output_test
#inp = np.array(inp,dtype=np.float32)
#out = np.array(out,dtype=np.float32)
y_pred = np.zeros([50,1])
import matplotlib.pyplot as plt
%matplotlib inline
lam = [1,0.99,0.95,0.2]
a = (input_train.T).dot(input_train)
ainv = inv(a)
a1 = ainv.dot(input_train.T)
b0 = a1.dot(output_train)
y_pred[0] = inp[50].dot(b0)
M0 = ainv
for l in lam:
    for i in range(50):
        D1 = l
        D2 = inp[50+i].dot(M0)
        D2 = D2.dot(inp[50+i].T)
        D = D1+D2
        N = M0.dot(inp[50+i].T)
        N = N.dot(inp[50+i])
        N = np.dot(N,M0)
        M1 = M0 - N/D
        M1 = M1/D1
        C = M1.dot(inp[50+i].T)
        P = out[50+i]-np.dot(inp[50+i],b0)
        T2 = C*P
        TR = np.reshape(T2,b0.shape)
        b1 = b0 + TR
        if(i!=49):
            y_pred[i+1] = inp[51+i].dot(b1)
        b0 = b1
        M0 = M1
    e = np.sqrt(np.sum(np.square(output_test-y_pred))/50)
    print("This is for lambada value "+str(l))
    print("The error is "+str(e))
    print("The graphical description is as follows:")
    plt.title('Actual y')
    plt.xlabel('x')
    plt.subplot(3, 1, 1)
    plt.plot(output_test, 'o', label='actual y')
    plt.plot(y_pred, 'o', label='predicted y')
    plt.subplot(3, 1, 1)
    plt.legend(loc='upper center', ncol=4)
    plt.gcf().set_size_inches(15, 10)
    plt.show()
```

2.d – LW Regression

```
import pandas as pd
import numpy as np
from numpy.linalg import inv
input_train = pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Input_Training.csv")
output_train =
pd.read_csv("/home/stark/Downloads/cs_ass2/Problem3_Output_Training.csv",header =
None)
input_test = pd.read_csv("/home/stark/Downloads/cs_ass2/Problem1_Input_Test.csv")
output_test =
pd.read_csv("/home/stark/Downloads/cs_ass2/Problem3_Output_Test.csv",header = None)
input_train = np.array(input_train)
input_train[:,0] = 1
output_train = np.array(output_train)
input_test = np.array(input_test)
input_test[:,0] = 1
output_test = np.array(output_test)
import matplotlib.pyplot as plt
%matplotlib inline
fig, ax = plt.subplots()
x = list(range(1,51))
y = output_train
ax.plot(x,y)
fig.suptitle('test title')
ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')
ax.xaxis.label.set_size(20)
plt.draw()
dist_mat = np.zeros([input_test.shape[0],input_train.shape[0]])
```

```
for te in range(input_test.shape[0]):
    for tr in range(input_train.shape[0]):
        dist_mat[te][tr] = np.sqrt(np.sum(np.square(input_test[te]- input_train[tr])))
percent = np.zeros([10])
for i in range(10):
    percent[i] = (np.percentile(dist_mat,10*(i+1)))
from numpy.linalg import inv
x_inp = np.zeros([50,3])
y_out = np.zeros([50,1])
y_final = np.zeros([50,1])
k = percent
for l in k:
    for i in range(input_test.shape[0]):
        count = 0
        for j in range(input_train.shape[0]):
            if(dist_mat[i][j]<=l):
                x_inp[count,:] = input_train[j,:]
                y_out[count,:] = output_train[j,:]
                count = count+1
        x = x_inp[:count,:]
        y = y_out[:count,:]
        a = (x.T).dot(x)
        ainv = inv(a)
        a1 = ainv.dot(x.T)
        b = a1.dot(y)
        y_final[i] = input_test[i].dot(b)
    e = np.sqrt(np.sum(np.square(output_test-y_final))/50)
    print(e)
    from numpy import *
    import matplotlib.pyplot as plt
    %matplotlib inline
    x = list(range(1,51))
    plt.plot(x,y_final,'r')
    plt.plot(x,output_test,'b')
    #plt.show()
#print(y_final-output_test)
#two dimensional data hard to visualise
#clearly from the plot of the data it can be said that local points are not able to predict the
point.
for i in range(input_test.shape[0]):
    coun = 0
    for j in range(input_train.shape[0]):
        if(dist_mat[i][j]<=7):
            coun = coun+1
    #print(coun)
# we need atleast 4% of the test to make a staright line.
from numpy.linalg import inv
x_inp = np.zeros([50,3])
y_out = np.zeros([50,1])
y_final = np.zeros([50,1])
for i in range(input_test.shape[0]):
    count = 0
    for j in range(input_train.shape[0]):
        if(dist_mat[i][j]<=1):
            x_inp[count,:] = input_train[j,:]
            y_out[count,:] = output_train[j,:]
            count = count+1
    if(count<2):
        ind = np.argpartition(dist_mat[i,:], 2)[:2]
        for i in range(2):
            x_inp[i,:] = input_train[ind[i],:]
            y_out[i,:] = output_train[ind[i],:]
        count = 2
    x = x_inp[:count,:]
    y = y_out[:count,:]
    a = (x.T).dot(x)
    ainv = inv(a)
    a1 = ainv.dot(x.T)
    b = a1.dot(y)
    y_final[i] = input_test[i].dot(b)
    #print(y_final[i])
e = np.sqrt(np.sum(np.square(output_test-y_final))/50)
print(e)
from numpy import *
import matplotlib.pyplot as plt
%matplotlib inline
x = list(range(1,51))
plt.plot(x,y_final,'r')
plt.plot(x,output_test,'b')
plt.xlabel('Data point')
plt.ylabel('Predicted(Red) and actual Y(Blue)')
plt.show()
#only 2% improvement
#even after finding the minimum distanced 2 points in case when no points exist in radius of
1.very less improvement
y_final = np.zeros([50,1])
x = np.zeros(input_train.shape[0])
for i in range(input_test.shape[0]):
    for j in range(input_train.shape[0]):
        x[j] = (-0.5*dist_mat[i][j]*dist_mat[i][j])
    norms = np.linalg.norm(x, keepdims = True)
    x /= norms
    x = np.exp(x)
    #print(x)
    dist_new = np.diag(x)
    winv = inv(dist_new)
    a1 = (input_train.T).dot(winv)
    a2 = a1.dot(input_train)
    ainv = inv(a2)
    a3 = ainv.dot(input_train.T)
    a4 = a3.dot(winv)
    a5 = a4.dot(output_train)
    y_final[i] = input_test[i].dot(a5)
e = np.sqrt(np.sum(np.square(output_test-y_final))/50)
```

```
print(e)
from numpy import *
import matplotlib.pyplot as plt
%matplotlib inline
x = list(range(1,51))
plt.plot(x,y_final,'r')
plt.plot(x,output_test,'b')
plt.xlabel('Data point')
plt.ylabel('Predicted(Red) and actual Y(Blue)')
plt.show()
y_final = np.zeros([50,1])
x = np.zeros(input_train.shape[0])
for i in range(input_test.shape[0]):
    for j in range(input_train.shape[0]):
        if(1/dist_mat[i][j]>10):
            x[j] = 10
        else:
            x[j] = 1/dist_mat[i][j]
    norms = np.linalg.norm(x, keepdims = True)
    x /= norms
    dist_new = np.diag(x)
    winv = inv(dist_new)
    a1 = (input_train.T).dot(winv)
    a2 = a1.dot(input_train)
    ainv = inv(a2)
    a3 = ainv.dot(input_train.T)
    a4 = a3.dot(winv)
    a5 = a4.dot(output_train)
    y_final[i] = input_test[i].dot(a5)
e = np.sqrt(np.sum(np.square(output_test-y_final))/50)
print(e)
from numpy import *
import matplotlib.pyplot as plt
%matplotlib inline
x = list(range(1,51))
plt.plot(x,y_final,'r')
plt.plot(x,output_test,'b')
plt.xlabel('Data point')
plt.ylabel('Predicted(Red) and actual Y(Blue)')
plt.show()
# from LSE
from numpy.linalg import inv
a = (input_train.T).dot(input_train)
ainv = inv(a)
a1 = ainv.dot(input_train.T)
b = a1.dot(output_train)
ycap = input_test.dot(b)
e = np.sqrt(np.sum(np.square(output_test-ycap))/50)
print(e)
from numpy import *
import matplotlib.pyplot as plt
%matplotlib inline
```

```
x = list(range(1,51))
plt.plot(x,ycap,'r')
plt.plot(x,output_test,'b')
plt.show()
```

THE END

- RAJA KUMAR DUBEY
-15MA20033