ELL409: MACHINE LEARNING

ASSIGNMENT 3

Support Vector Regression

Vivek Muskan
2017MT10755

Dataset is Boston House Pricing Data, consisting of 506 rows and 13 features and a class containing the Median value of owner-occupied homes to be predicted later.

First will look after SVR implementation using CVXOPT

Theory and Algorithm

We need to solve a dual problem in this, as discussed in the class. (From Bishop Book)

$$\begin{aligned}\widetilde{L}(\mathbf{a}, \widehat{\mathbf{a}}) &= -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(a_n - \widehat{a}_n)(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) \\ &\quad -\epsilon\sum_{n=1}^{N}(a_n + \widehat{a}_n) + \sum_{n=1}^{N}(a_n - \widehat{a}_n)t_n\end{aligned}$$

We need to maximize this equation with respect to constraints

$$\sum_{n=1}^{N}(a_n - \widehat{a}_n) = 0$$

$$0 \leqslant a_n \leqslant C$$
$$0 \leqslant \widehat{a}_n \leqslant C$$

Where C and epsilon are hyperparameters.

Using the value of “a” and “a^” found we can find the value of y(x), which is as follows:

$$y(\mathbf{x}) = \sum_{n=1}^{N}(a_n - \widehat{a}_n)k(\mathbf{x}, \mathbf{x}_n) + b$$

Where b is

$$\begin{aligned} b &= t_n - \epsilon - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n) \\ &= t_n - \epsilon - \sum_{m=1}^{N}(a_m - \widehat{a}_m)k(\mathbf{x}_n, \mathbf{x}_m) \end{aligned}$$

Then we took the average of all the b for which points are support vectors.

The above dual problem is solved using a QP Solver, whose standard form is given below:

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x^{\top}Px + q^{\top}x \\ \text{subject to} \quad & Gx \preceq h \\ & Ax = b \end{aligned}$$

We can get the value of x using the following equations:

```
sol = solvers.qp(P,q,G,h,A,b)
sol['x']
```

Hence, we need to convert our dual problem in the above form. So, took x as a matrix of [a, a^] and then changes each equation accordingly, which is shown below.

Implemented SVR

- At first Data framing and Normalization has been done and then in Order to frame optimization problem into Standard form, Kernel Function is defined.

-

The different kernel function is introduced as:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

Polynomial kernel equation

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Gaussian radial basis function (RBF)

```
[ ] def Kernel(name, x, y, gamma):
      if name == 'linear':
        return np.dot(x,y)

      elif name == 'rbf':
        #gamma = 1/(np.var(x) + np.var(y))
        return np.exp(-gamma*np.matmul((x-y).transpose(),(x-y)))

      elif name == 'poly':
        return (np.dot(x,y)+1)**(2) # Quadratic Kernel

      else:
        print("Please Enter - 'linear' or 'poly' or 'rbf' ")
```

- After that predictor function which will help in predicting the median using the equation of Hyperplane as y(x):

$$y(\mathbf{x}) = \sum_{n=1}^{N} (a_n - \widehat{a}_n) k(\mathbf{x}, \mathbf{x}_n) + b$$

```
[ ] def predictor(X_train,x,Alpha,B,s, gamma):
      ans = 0
      N = int(len(Alpha)/2)
      for i in range(N):
        ans += (Alpha[i]-Alpha[N+i])*Kernel(s,x,X_train[i], gamma)
      return ans
```

- Then we splitted data into Training and Test dataset for K-Fold Cross Validation and Extracted (X,y) for both Training and Testing.

```python
# K-Fold
k_fold_size = 101
no_of_fold = int(len(data)/k_fold_size)
count = 0
MSE = 0
for i in range(no_of_fold):

    count+=1
    print("Validation No : ",count," -------------------------------------------------")


    X_test = np.array([data[j][:-1] for j in range(i*k_fold_size,(i+1)*k_fold_size)])
    Y_test = np.array([data[j][-1] for j in range(i*k_fold_size,(i+1)*k_fold_size)])

    X_train = []
    Y_train = []
    for j in range(len(data)):
        if (j<i*k_fold_size) or (j>(i+1)*k_fold_size):
            X_train.append(data[j][:-1])
            Y_train.append(data[j][-1])
    X_train = np.array(X_train)
    Y_train = np.array(Y_train)
```

- After that I fixed Hyper-parameters and then conversion of optimization equation into standard Matrix form. This was the crux of all steps.

  IDEA: Equation consist (a, a^) as variable so take them as single vector [a a^] and then from dimension of P and backtracking summation I got following equation for every Matrix:

-

$$P = \begin{bmatrix} K & -K \\ -K & K \end{bmatrix} \qquad q = \begin{bmatrix} y-\varepsilon \\ -y-\varepsilon \end{bmatrix}$$

$I_{N\times N}$ - Identity
$O_{N\times N}$ - Zero
$y_{N\times 1}$ - Label
$K_{N\times N}$ - Kernel

$$G = \begin{bmatrix} -I & O \\ O & -I \\ I & O \\ O & I \end{bmatrix} \qquad h = \begin{bmatrix} O \\ O \\ cI \\ cI \end{bmatrix}$$

$$A = \begin{bmatrix} I & -I \end{bmatrix} \qquad b = \begin{bmatrix} 0 \end{bmatrix}$$

```python
# Hyper-parameters
epsilon = 0.1
C = 100
ker = 'linear'
gamma = 0.1

# Optimization COnversion
K = []
for x1 in X_train:
  row = []
  for x2 in X_train:
    row.append(Kernel(ker,x1,x2,gamma))
  K.append(row)

# Sub matrices
N = len(X_train)
K = matrix(np.array(K),tc='d')
I = matrix(np.eye(N),tc='d')
O = matrix(np.zeros([N,N]),tc='d')
Ones_N_1 = matrix(np.ones([N,1]),tc='d')
Zeros_N_1 = matrix(np.zeros([N,1]),tc='d')
y = matrix(Y_train,tc='d')


# Transformation into Standard Form
P = (1/2)*matrix([[K,-K],[-K,K]],tc='d')
q = matrix([(y-epsilon*Ones_N_1), -(y+epsilon*Ones_N_1)],tc='d')
G = matrix([[-I,O,I,O],[O,-I,O,I]],tc='d')
h = matrix([Zeros_N_1,Zeros_N_1,C*Ones_N_1,C*Ones_N_1],tc='d')
A = matrix([[matrix(1,(1,N),tc='d')],[matrix(-1, (1,N),tc='d')]],tc='d')
b = matrix([0],tc='d')
```

All the useful matrices like Identity, Zeros, Ones and Kernel etc. is defined under Sub-matrices section of the code.

- Then Using CVXOPT, this problem has been solved to get [a, a^] and then B has been obtained.
- Then Using prediction on test data, I calculated the MSE keeping a margin of epsilon.

```
# Solving Optimization Problem
sol = solvers.qp(P,q,G,h,A,b)

x = sol['x']
B = matrix(1,(1,N),tc='d')*(matrix([[K],[-K]],tc='d')*x + matrix([(y-epsilon*Ones_N_1)]))
B /= N

# print(x.size)
# print(B.size)

# Prediction Stage
Y_predicted = []
sq_err = 0
for t,y in zip(X_test,Y_test):
  fx = predictor(X_train, t, x, B, ker,gamma)
  Y_predicted.append(fx)

  if(abs(y-fx) > epsilon):
    sq_err += (abs(y-fx)-epsilon)**2

Y_predicted = np.array(Y_predicted)
MSE += sq_err
#print("Squared Err from ",count,"th fold = ",sq_err)

# Data Visualisation
plt.scatter(X_test[:,0], Y_predicted, color = 'm',label = "Predicted")
plt.scatter(X_test[:,0], Y_test, color = 'g', label = "Actual")
plt.xlabel("1st Column of X")
plt.ylabel("Median Val")
plt.title("Validation Fold No : %d\nKernel=%s, C=%d, epsilon=%f, Gamma=%f\n MSE=%f"%(i+1,ker,C,epsilon,gamma,sq_err))
plt.legend()
plt.show()
```

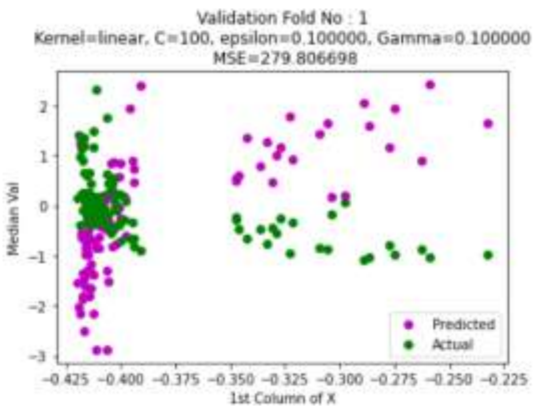
- I took average of MSE over all folds and a scatter plot of predicted values (PINK) has been plotted versus 1st column of test data along with actual label (GREEN) to observe the deviation of prediction from actual value.

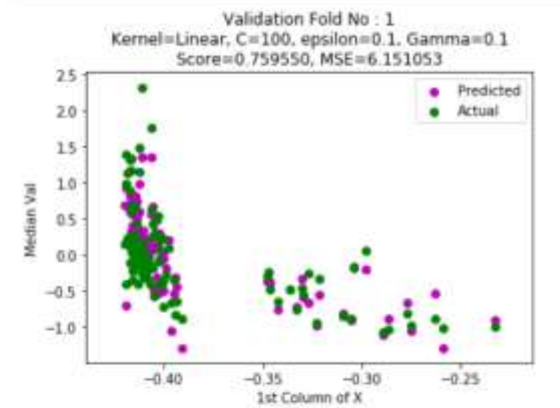
- Keeping all other aspect same, using Sci-kit learn, SVR has been used to plot similar graphs to compare.

Comparative Graph Plotting

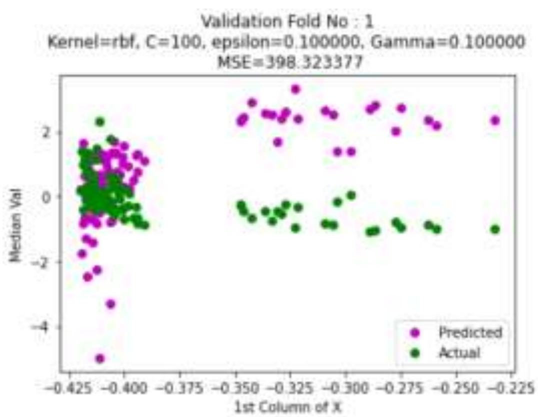
Implemented SVR **SK-learn**

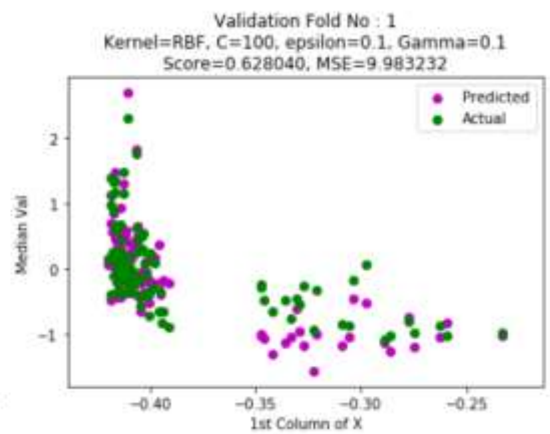
Kernel: Linear



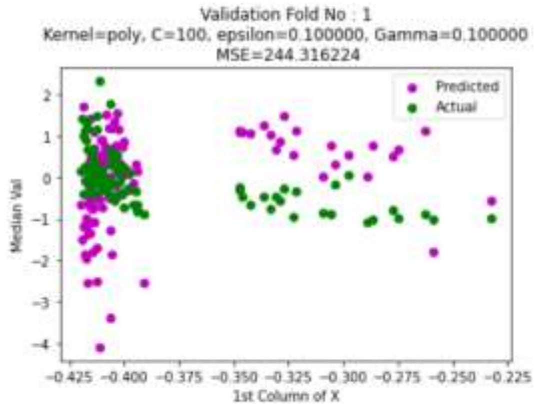


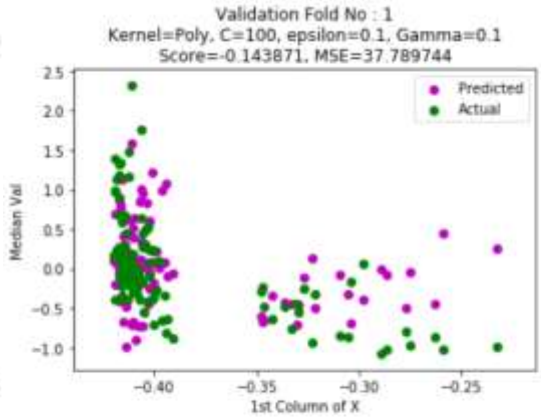
Kernel: RBF



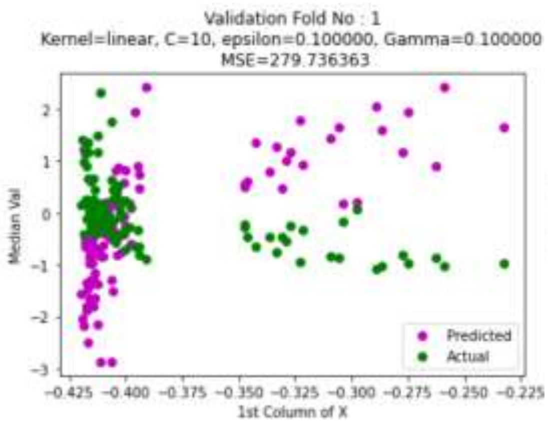


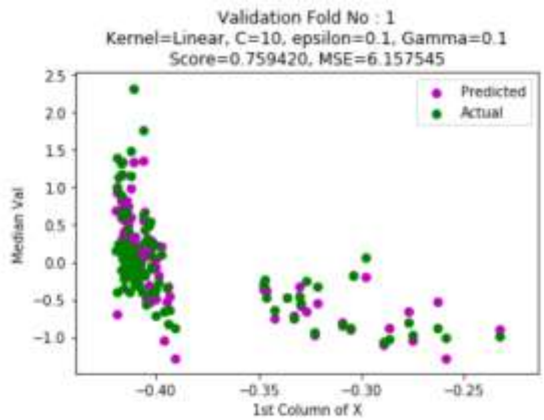
Kernel: Poly 2^{nd} Degree



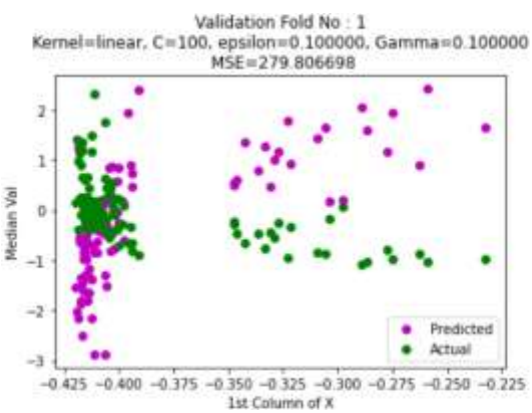


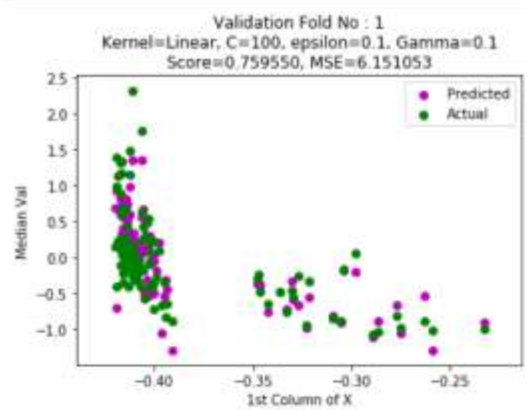
__C = 10__



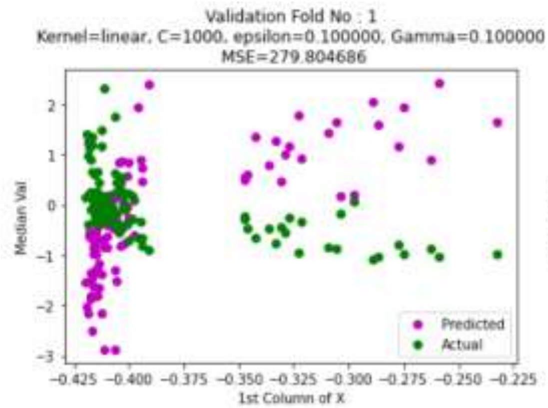


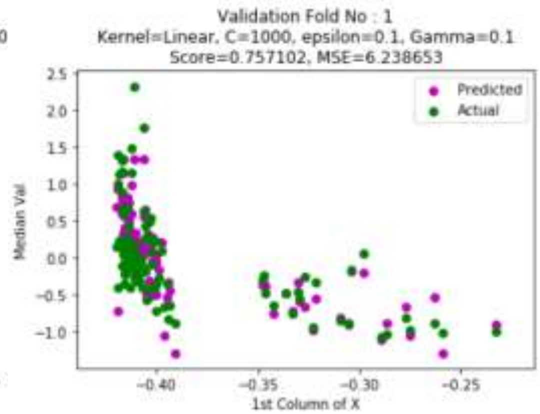
__C = 100__



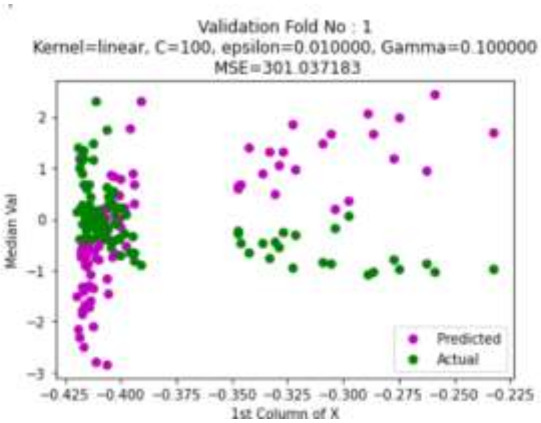


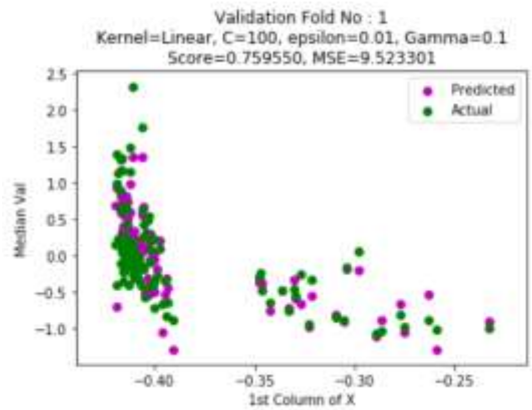
__C = 1000__



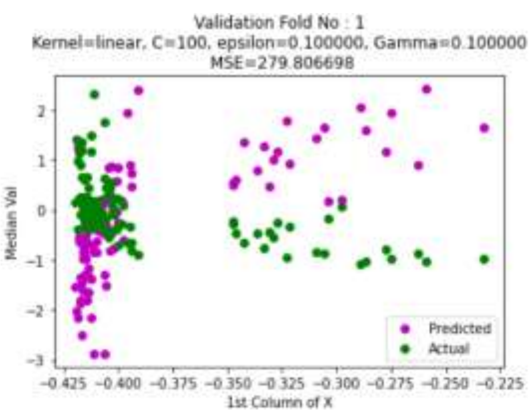


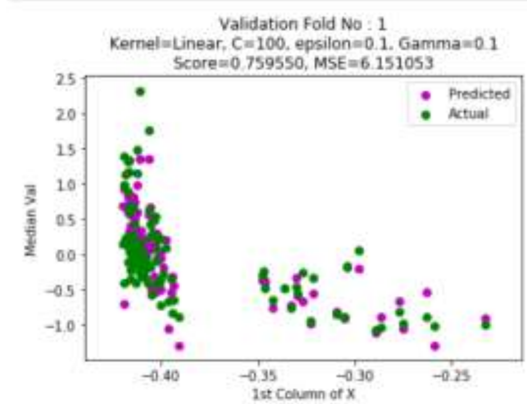
__Epsilon = 0.01__



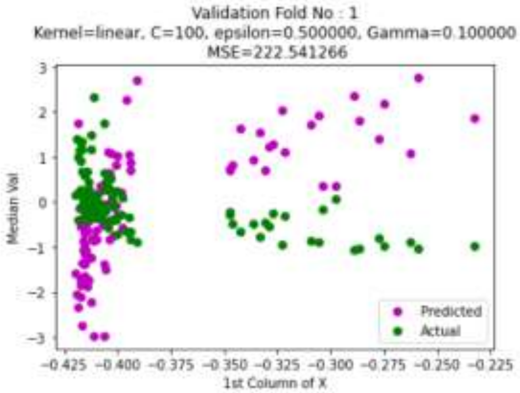


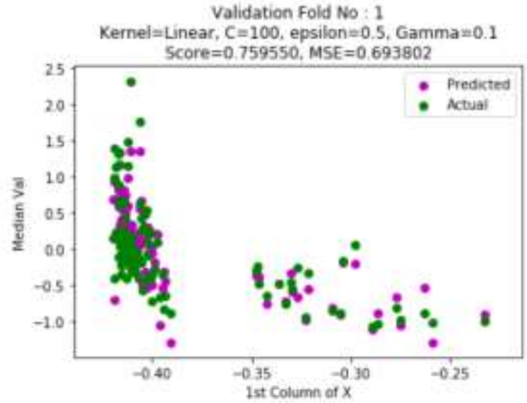
__Epsilon = 0.1__





__Epsilon = 0.5__





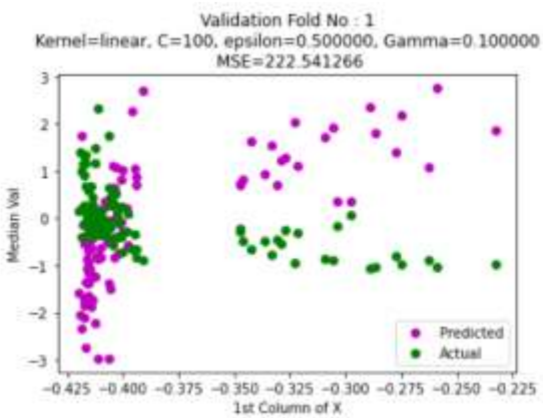
Best Parameters came out to be:

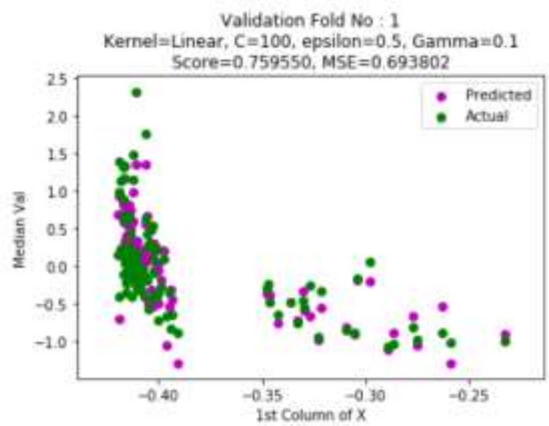
Kernel: Linear

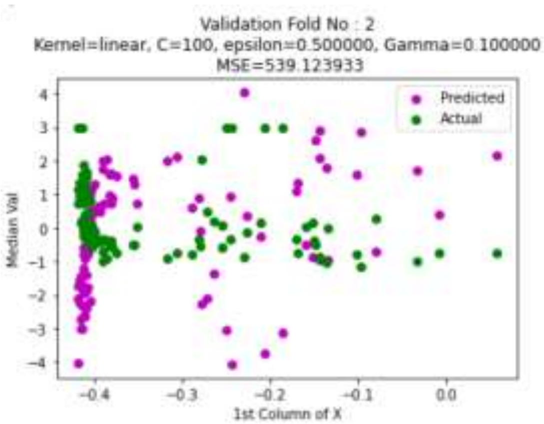
C = 100, Epsilon = 0.5

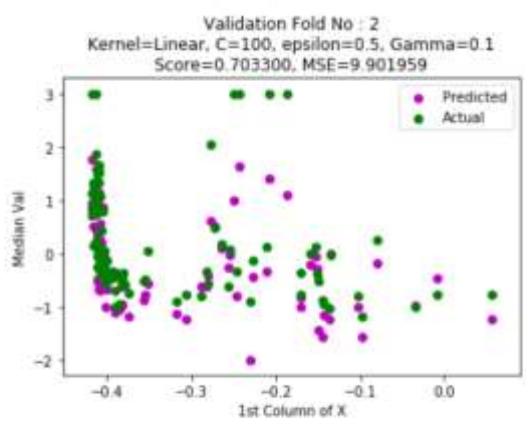
So, here is a plot of each validation fold for above parameters:

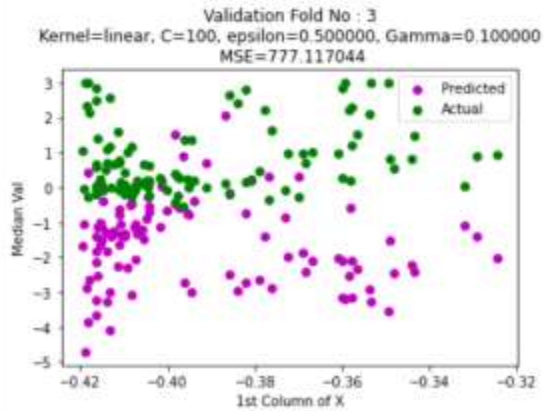
Implemented SVR **SK-Learn**

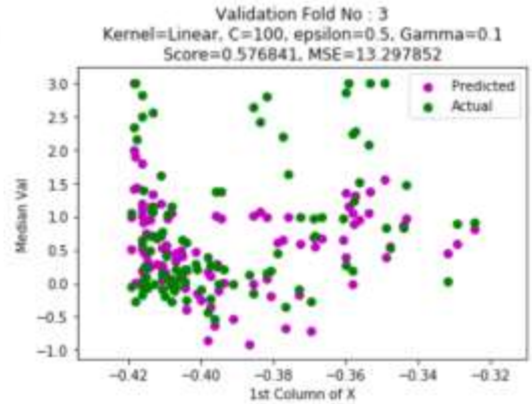


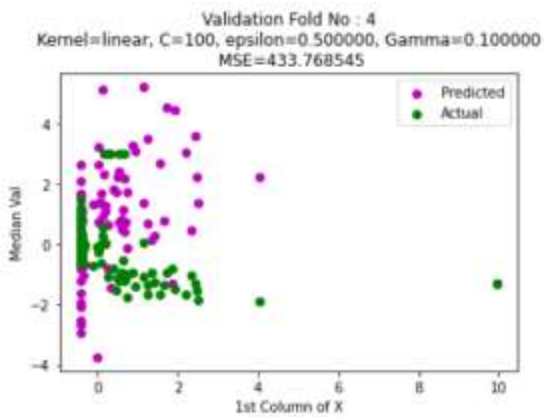


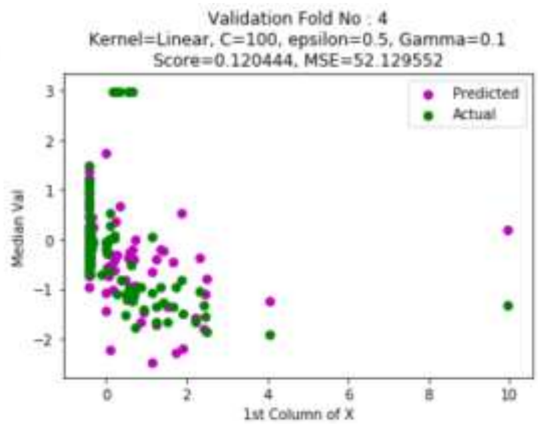


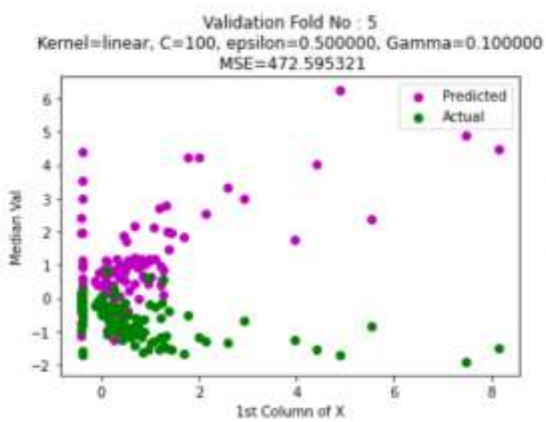


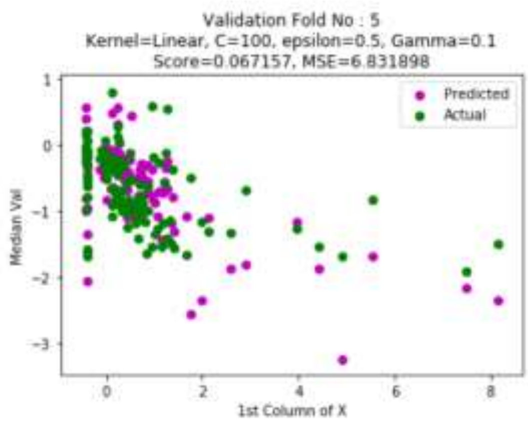

Validation Fold No : 3
Kernel=linear, C=100, epsilon=0.500000, Gamma=0.100000
MSE=777.117044
Predicted
Actual
Median Val
1st Column of X


Validation Fold No : 3
Kernel=Linear, C=100, epsilon=0.5, Gamma=0.1
Score=0.576841, MSE=13.297852
Predicted
Actual
Median Val
1st Column of X


Validation Fold No : 4
Kernel=linear, C=100, epsilon=0.500000, Gamma=0.100000
MSE=433.768545
Predicted
Actual
Median Val
1st Column of X


Validation Fold No : 4
Kernel=Linear, C=100, epsilon=0.5, Gamma=0.1
Score=0.120444, MSE=52.129552
Predicted
Actual
Median Val
1st Column of X


Validation Fold No : 5
Kernel=linear, C=100, epsilon=0.500000, Gamma=0.100000
MSE=472.595321
Predicted
Actual
Median Val
1st Column of X


Validation Fold No : 5
Kernel=Linear, C=100, epsilon=0.5, Gamma=0.1
Score=0.067157, MSE=6.831898
Predicted
Actual
Median Val
1st Column of X