The datasets used in this is the **Boston housing data.**

After having the data I did pre-processing by the sklearn scaling pre-processing function

**SVR\_class(rh-svr)**

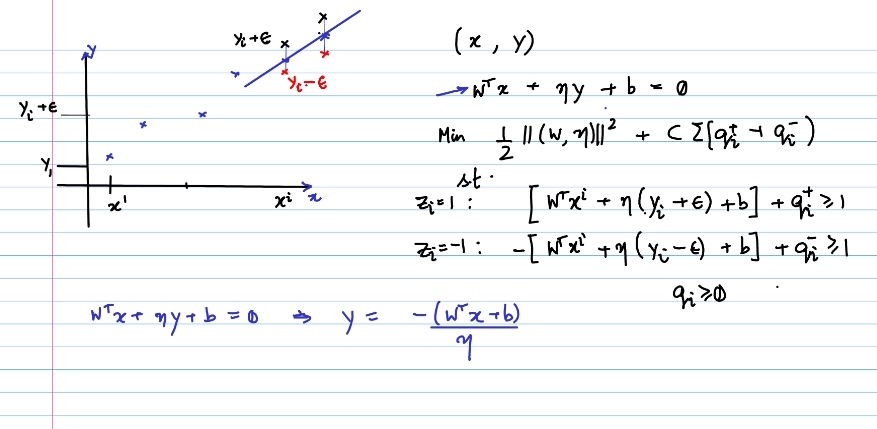
**Methods to call the program:**

* **test\_linear(X,y,C=1,e=0.3,ratio=0.2)**
* **test\_soft(X,y,C=1000.1,e=0.5,ratio=0.2)**
* **k\_fold(X,y,C=1,e=0.3,n\_folds=5):**
* **SVM initialisation:-** **\_\_init\_\_(self, kernel=linear\_kernel, C=1 ,epsilon=0.3):**

**PRE-REQ:-**

I have implemented this by using cvxopt solver.

This method have implemented on the basis of the idea you have given in the class by considering y-e and y+e new features and with values - 1 and +1 respectively and evaluating y with y=-(w\*x+b)/(nu).



I have implemented this method only for linear types because for nonlinear kernels evaluating y was difficult by above formula of y.

**Algorithm:-**

Basically, in this method I have just implemented SVM with additional features y-e and y+e and you and using the line separating both features as the required regressor.

For training and validation I have implemented two types of comparison algorithm namely training test validation split and k fold cross validation method.

In this algorithm, soft margin classifier as well as hard margin classifier can be achieved.

The error function which I have used in this is also mean squared error.

|  |  |  |
| --- | --- | --- |
| C | Epsilon (e) | Graph with error(linear kernel) |
| 1.00 | 0.1 |  |
| 1.00 | 0.3 |  |
| 1.00 | 0.5 |  |
| 1.00 | 1.0 |  |
| 10.00 | 0.2 |  |
| 10.00 | 0.3 |  |
| 10.00 | 0.5 |  |
| 10.00 | 1.0 |  |
| 100.00 | 0.1 |  |
| 100.00 | 0.3 |  |
| 100.00 | 0.5 |  |
| 100.00 | 1.0 |  |
| 1000.00 | 0.1 |  |
| 1000.00 | 0.3 |  |
| 1000.00 | 0.5 |  |

**SVR\_sklearn**

**Methods to call:**

* **train\_test\_split(C,e,ratio)**
* **Kfold(C,e,n\_folds)**

For the comparing purpose I have used the sklearn SVR with kernel rbf.

For cross validation and training I have used the Kfold cross validation method also it is taken from the sklearn inbuilt Kfold validation split function.

For error measurement I have used mean-squared-error.

**SVR\_paper**

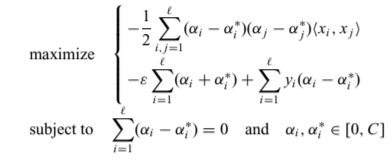
**Methods to call the program:-**

* **test\_linear(X,y,C=1,e=0.3,ratio=0.2)**
* **test\_non\_linear(X,y,kernel,C,e,ratio)**
* **test\_soft(X,y,e,ratio)**
* **k\_fold(X,y,C=1,e=0.3,n\_folds=5)**
* **SVR initialisation:- \_\_init\_\_(self, kernel=linear\_kernel, C=1 ,epsilon=0.3)**

**PRE\_REQ:-**

I have tried implemented this by using cvxopt solver.

This method which I have implemented is directly based on solving the minimising problem which we got from epsilon tube assumption.



This method is quite general can be utilised for multiple kernels like linear kernel, polynomial kernel as well as Gaussian kernel.

**Algorithm:-**

To formulate this and to overcome ai and ai\* I used a single 2n length array of which first n are ai and rest are ai\*, so therefore the equivalent p matrix became [K(x,y) -K(x,y),-K(x,y) K(x,y)]. Similarly I made the rest matrices accordingly and later just formulated what was given in the paper, you given.

In this algorithm, soft margin classifier as well as hard margin classifier can be achieved.

For checking and validation the methods implemented to be used are training test validation split as well as k fold cross validation split.

Error function is used is mean squared error.

KERNEL COMPARISON

|  |  |  |  |
| --- | --- | --- | --- |
| epsilon | C | kernel | mse |
| 0.1 | 1 | linear | 0.10175823768346721 |
| 0.1 | 1 | Non-linear(polynomial of degree 2 ) | 0.2387300022402307 |
| 0.1 | 1 | Non-linear(gaussian) | 0.06294591603726271 |
| 0.1 | 1000 | linear | 0.09785255526179185 |
| 0.1 | 1000 | Non-linear(polynomial of degree 2 ) | 0.2332568233305947 |
| 0.1 | 1000 | Non-linear(gaussian) | 0.1250140509160475 |

Comparison between methods implemented

|  |  |  |  |
| --- | --- | --- | --- |
| epsilon | C | method | mse |
| 0.1 | 1 | Sklearn svr | 0.09863247347965674 |
| 0.1 | 1 | Cvxopt\_class(rh-svr) | 0.15353084 |
| 0.1 | 1 | Cvxopt\_paper | 0.10175823768346721 |
| 0.1 | 1000 | Sklearn svr | 0.09732685894805433 |
| 0.1 | 1000 | Cvxopt\_class(rh-svr) | 0.15915849 |
| 0.1 | 1000 | Cvxopt\_paper | 0.09785255526179185 |
| 0.3 | 1 | Sklearn svr | 0.11429262409837403 |
| 0.3 | 1 | Cvxopt\_class(rh-svr) | 0.11067671 |
| 0.3 | 1 | Cvxopt\_paper | 0.11530027298295902 |
| 0.3 | 1000 | Sklearn svr | 0.11487747676734845 |
| 0.3 | 1000 | Cvxopt\_class(rh-svr) | 0.10934239 |
| 0.3 | 1000 | Cvxopt\_paper | 0.1432871077820389 |

OBSERVATION:-

Smaller epsilon as well as much bigger epsilon were giving greater error, for me epsilon=0.3 worked well averagely.

Increasing C and reducing epsilon were increasing the no. of support vectors, likely to be overfitting and underfitting for the vice versa.

For me, Cvxopt\_paper with gaussian kernel give the best result with C=1 and epsilon=0.1 gave the best results.