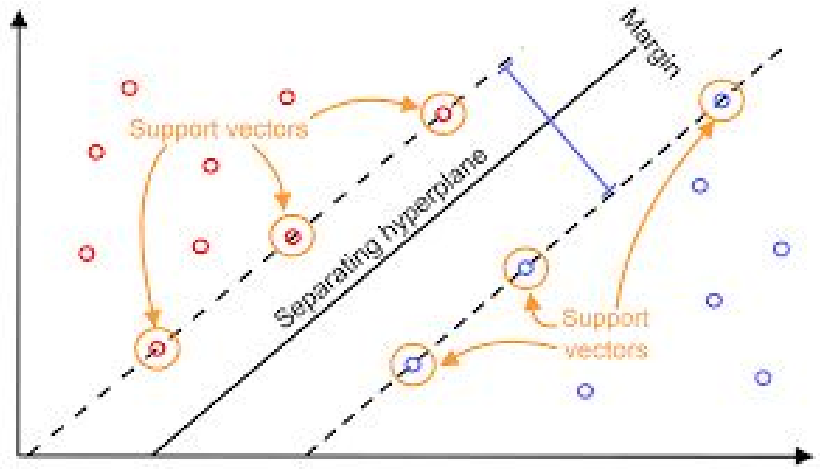
ELL-409

ASSIGNMENT-3

Support Vector Regression

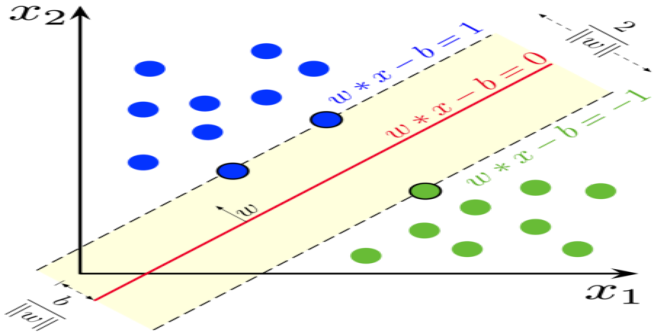
-Amokh Varma(2018MT60527)



Introduction

Support Vector Machines are a method to classify data into different classes. It's importance stems from its ability to scale to higher dimensions without the need for storing those dimensions explicitly . In the course of this assignment, we will be looking into the adaptation of SVM into regression problems. This is called Support Vector Regression.

Theory



The basic principle behind SVM is to create a decision boundary to separate different classes. The way to do that is to maximise the width of the above shaded area. This is equivalent to minimising $|w|$. To account for external points, slack variables are used . Now this optimisation essentially reduces to

$$min\ (0.5 * |w| + (\sum_{1}^{n} q_i)\ )\quad \text{s.t}\quad y_i\,(w^T x + b) \geq 1 - q_i \ and\ q_i \geq 0$$

Using the method of Lagrange multipliers and KKT condition we can find the required value for w and b.

Note that decision boundary is defined as :- (class = 1) if $w^T x + b > 0$.
To adapt this to regression, we use a concept called $\varepsilon - SVR$. We basically do the above steps but try to fit most of the data within an ε margin .

The new question reduces to the following :-

$$min\ (0.5 * |w| + (\sum_{1}^{n} q^+_i + q^-_i)\ )\quad s.t\ \ y_i - w^T x - b \leq \varepsilon + \xi^+_i\ \ ,\ \ y_i - w^T x - b \leq \varepsilon + \xi_i{}^-$$

$$\xi^+_i\ ,\ \xi^-_i \geq 0$$

Using the KKT conditions , the problem is converted to its dual and it reduces to the following equation :-

$$max\ -\tfrac{1}{2}\sum_{1}^{n}\sum_{1}^{n}(\alpha_i - \alpha^*_i)(\alpha_j - \alpha^*_j)\,x_i^T x_j\ -\ \varepsilon\sum_{1}^{n}(\alpha_i + \alpha^*_i\ )\ +\ \sum_{1}^{n} y_i(\alpha_i - \alpha^*_i\ )$$

s.t. $\alpha_i^{(*)} \geq 0\ \ and\ \sum_{1}^{n}(\alpha_i - \alpha^*_i) = 0$

This equation is given to a quadratic optimisation algorithm which returns the alphas.
One noteworthy point here is that for “unimportant” data, ($\alpha_i - \alpha^*_i) = 0$ which means that the vector has no contribution to the weight vector. All other vectors are therefore “Supporting” the training process and hence the name Support Vector Machines.

Kernel Trick :-

As we can see, the equation depends only on the dot-product of the vectors . So in order to take the vectors to higher dimensions,we do not need the actual coordinates,we only need the scalar product . Some very common kernels are listed below and in this assignment we

will be using all 3 of them and comparing them using our own implementation as well as scikit.sklearn.svr .

Polynomial Kernel :- $K(x1, x2) = (x1^T x2 + d)^n$

Linear Kernel :- $K(x1, x2) = x1^T x2$

Radial Basis Function (rbf) :- $K(x1, x2) = exp(-\gamma ||x1 - x2||^2)$

Result

The results have been compared using 2 different metrics, Mean squared error and R-2 score. K-crossValidation with K=5 has been done to check for overfitting and underfitting. Lower MSE and closer to 1 score are preferred.

$$\text{MSE} = \frac{1}{n} \sum_{1}^{n} (y_i - ypred_i)^2$$

$$(\text{R-2 score})^2 = \left(\sum_{1}^{n} (ypred_i - y_{mean,observed})^2 / (\sum_{1}^{n} (y_i - y_{mean,observed})^2 \right)$$

Note that the error and score are calculated on the normalised data. (for both scikit-sklearn and own implementation.)

Different hyperparameters used here are kernel and ε.

a) C = 0.3 , $\varepsilon = 0.1$, kernel = linear
b) C = 0.3 , $\varepsilon = 0.1$, kernel = polynomial(d = 2)
c) C = 0.3 , $\varepsilon = 0.1$, kernel = rbf
d) C = 0.3 , $\varepsilon = 0.25$, kernel = linear
e) C = 0.3 , $\varepsilon = 0.25$, kernel = polynomial(d=2)
f) C = 0.3 , $\varepsilon = 0.25$, kernel = rbf

The table below shows the different average metrics and K-cross outputs of metrics for the above 6 cases for both implementations.

Data from Different values of hyperparameters

Sno	***5-cross validation score***	***Average score***	***5-cross validation MSE***	***Average MSE***
a1	0.8786, 0.8396, 0.651, 0.7718, 1.1876	0.8657	0.2318, 0.3742, 0.4649 0.5568, 0.4134	0.4081

a2 (*)	0.8187, 0.3814, 0.2383, 0.5677, 0.4192	0.4851	0.5951, 0.5214, 0.7435, 0.6432, 0.384	0.5774
b1	0.3986, 0.282, 0.9273 0.5633, 1.7014	0.7746	0.5455, 0.7735, 0.9047 0.7284, 0.9269	0.7757
b2	1.6117, 0.9566, 0.626 1.5708, 1.3659 (bad fit)	1.226	1.115, 1.0822, 0.983 0.8075, 0.8927	0.9761
c1	0.6465 0.5519 0.6836 0.4725 0.6788	0.6067	0.251 0.4285 0.6595 0.5001 0.4986	0.4675
c2	0.1863 0.0437 0.4963 0.2738 1.0925	0.4185	0.496 0.7769 0.7724 0.7671 0.6933	0.7011
d1	0.9189 0.7471 0.6061 0.8128 1.1814	0.8533	0.2405 0.3253 0.4532 0.5951 0.4327	0.4093
d2	0.9694 0.3771 0.2532 0.5816 0.6537	0.5672	0.6867 0.593 0.7719 0.7288 0.4422	0.6445
e1	0.3792 0.2806 0.8992 0.5692 1.6322	0.752	0.5485 0.7617 0.8867 0.7399 0.8992	0.7672
e2	1.7265 0.9878 0.6807 1.5509 1.4244	1.2740	1.1699 1.118 0.9961 0.8774 0.8742	1.0071
f1	0.6175 0.506 0.6525 0.4772 0.7086	0.592	0.2762 0.4258 0.658 0.537 0.5072	0.4808
f2	0.4195 0.104 0.3442 0.4208 1.3591	0.5295	0.5613 0.83 0.7669 0.8499 0.8124	0.7641

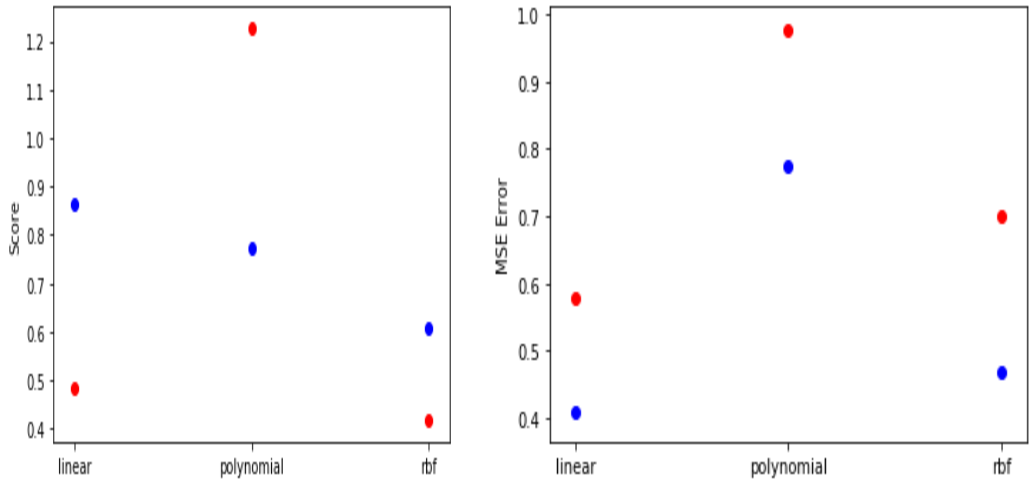
* 1 index refers to scikit-sklearn and 2 refers to own implementation.

* **score > 1 and score $\ll 1$ is representative of poor learning.**

* rbf, polynomial kernels don't always score well due to over fitting as it has many dimensions.

To visualise the same :-

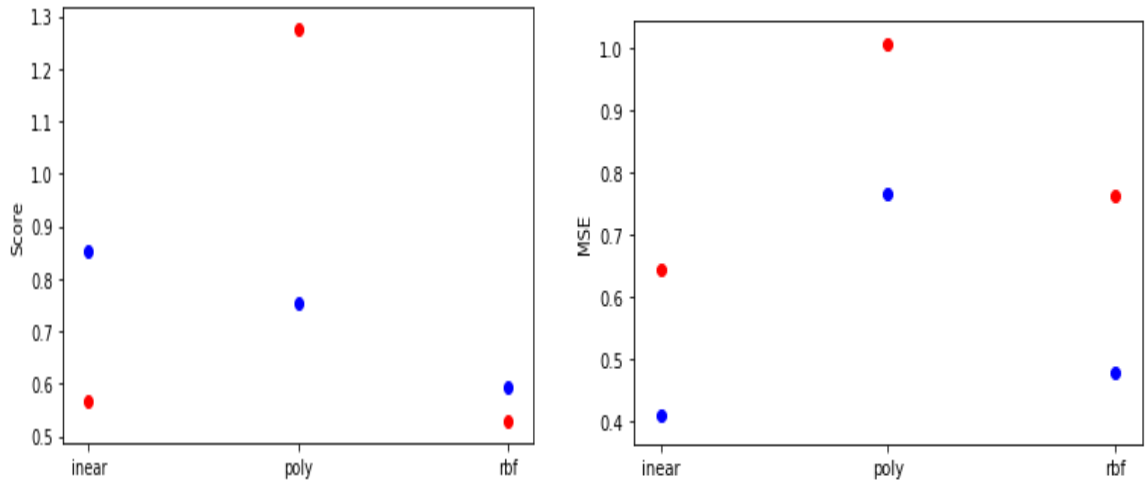
$\varepsilon = 0.1$ Characteristics



Red - Own implementation

Blue- Scikit implementation

$\varepsilon = 0.25$ Characteristics



Red - Own implementation

Blue- Scikit implementation

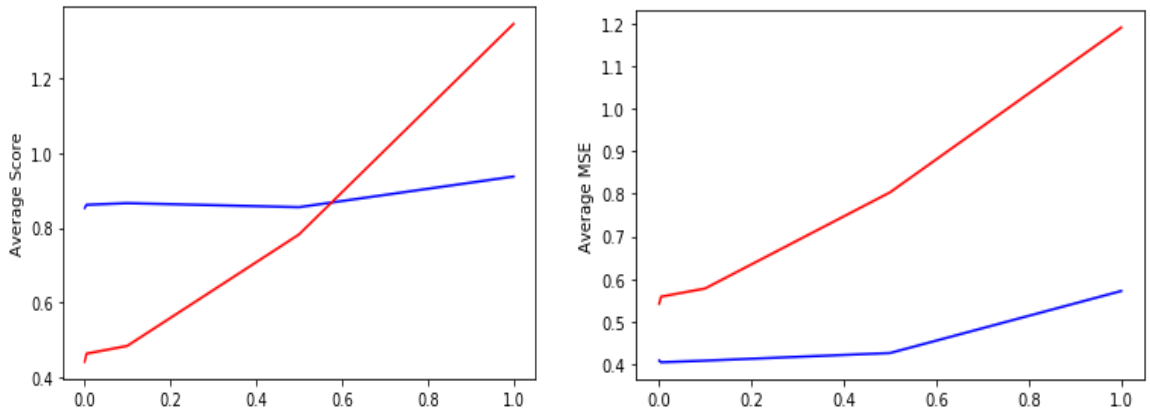
Interpretation :- The better results can be obtained using rbf and linear kernels. Some kernels have very high score,which suggests that training has not taken place properly

which can be due to low number of test samples. We can also notice that very high and very low score values are associated with high Mean Square Errors.

Now we look at different values of epsilon for linear kernel :-

Epsilon	Implementation	Average Score	Average MSE
1	Scikit	0.9367	0.5716
1	Own	1.344	1.19
0.5	Scikit	0.855	0.426
0.5	Own	0.7822	0.8031
0.1	Scikit	0.8657	0.4081
0.1	Own	0.4851	0.5774
0.05	Scikit	0.8613	0.4043
0.05	Own	0.4644	0.5583
0.001	Scikit	0.8529	0.4082
0.001	Own	0.4429	0.5416

To visualise this :-



Red - Own implementation

Blue- Scikit implementation

Conclusion :-

We can clearly see that the own implementation has much lesser accuracy compared to scikit implementation. This is because scikit implementation uses a combination of different SVR algorithms to achieve it's result and also has the advantage of much higher numerical stability. The SVR algorithm is clearly a very useful algorithm to perform regression and further, we can notice that even with high dimensional kernels,the space occupancy is not high. The SVM and its variations rely upon geometric properties to perform classification while other methods like logistic regression generally tend to be statistical in their approach.The SVR algorithm however suffers from having to undergo heavy calculations and hence cannot be easily scaled up for large data.

References :-

1) Sci-kit sklearn documentation
2) Tutorial on Support Vector Regressionby Alex J. Smola and Bernhard Scholkopf
3) Pattern Recognition and Machine Learning by Christopher M Bishop
4) Burges SVM Tutorial