Assignment 3

Achint Kumar Aggarwal
2018EE10433

Assignment Report

This report consists of various SVR and kernel implementations on a given dataset (The Boston house-price data of Harrison) consisting of 13 feature-input for 506 samples.

1. Support Vector Regression Overview

Inspired from the success of SVM algorithm used for classification problems, SVR reformulates a given regression problem into a convex optimization problem, which can be easily solved using various optimizers provided by the python libraries.

1.1 Ordinary Least Squares Regression (OLR) with Regularization:

$$Given\ a\ data-set\ \{x^i, y^i\}\ where\ x^i \in R^d\ and\ y^i \in R\ \forall\ i \in \{1, 2, 3...n\}, let:$$

$$y^i = f(x^i) + \eta^i$$

Our aim is to minimize a cumulative metric (squared error loss in this case) of η^i, at the same time enhancing the generalization of f (parameterized by θ) by introducing a regularization term.

$$\text{minimize} \quad \Sigma(\eta^i)^2 + ||\theta^2|| \implies \text{minimize} \quad \Sigma(y^i - f(x^i))^2 + ||\theta^2||$$

Henceforth, we find f parameterized by some θ which minimizes the objective function.

1.2 Support Vector Regression (SVR):

1.2.1 Motivation:

On certain instances, we can allow some tunable amount of error to be present if having an error less than the maximum allowed value helps in better generalization over the samples in the given feature space. To achieve the same we implement SVR, inspired by the concepts from both SVM and OLR.

1.2.2 Formulation:

Converting the OLR problem to a constrained optimization problem we have:

$$\text{minimize } \frac{1}{2}||\omega||^2 \text{ given that } |y^i - \omega^T.x^i - b| < \epsilon\ \forall\ i$$

Here, ϵ is chosen by the user as per the maximum allowable error in the particular case while still minimizing the regularization term.

However, this model is susceptible to outliers and fails to work in linearly non-separable cases. Fortunately, with a small modification the model can be made viable in the latter cases as well.

MODIFICATION:

$$\text{minimize } \frac{1}{2}||\omega||^2 + C.\Sigma_i\ \eta_i \text{ given that } |y^i - \omega^T.x^i - b| < \epsilon + \eta_i \ and\ \eta_i > 0 \forall\ i$$

The same has been implemented in python using various kernels and libraries which have been talked upon in detail in the following sections.

2. Implementation1 - Using CVXOPT Library:

2.1 Python code:

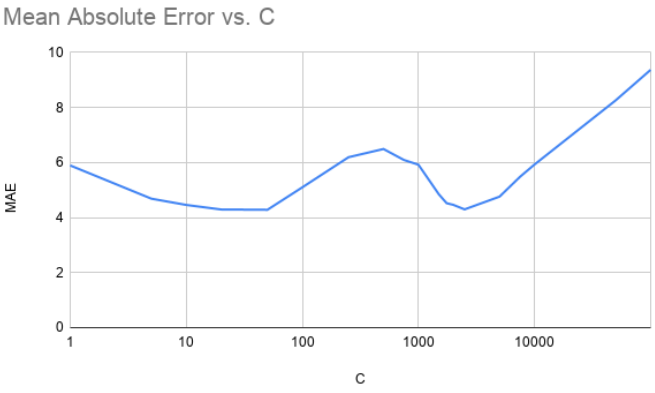
```
# FILE1: using SVR + CVXOPT
import numpy as np
import matplotlib
import csv
from cvxopt import matrix
from cvxopt import solvers

# INPUT
filename = "BostonHousing.csv"
data = []
with open(filename, 'r') as csvfile:
  csvreader = csv.reader(csvfile)
  for row in csvreader:
    data.append(row)
data = data[1:]
for i in range(len(data)):
  for j in range(14):
    data[i][j] = float(data[i][j])

# Separating Training and Testing data
train_data = data[0:355]
test_data = data[355:]

# Normalization
X = [0]*(len(train_data))
y = [0]*(len(train_data))
for i in range(len(train_data)):
  X[i] = train_data[i][0:13] + [1]
  y[i] = train_data[i][13]
for i in range(len(y)):
  y[i] = float(y[i])
for i in range(len(X)):
  for j in range(14):
    X[i][j] = float(X[i][j])
mu = np.mean(X)
std = np.std(X)
for i in range(len(X)):
  for j in range(14):
    X[i][j] = (X[i][j]-mu)/ std

# Computing the required matrices
ephsilon = 0.1
c = 100.0

```

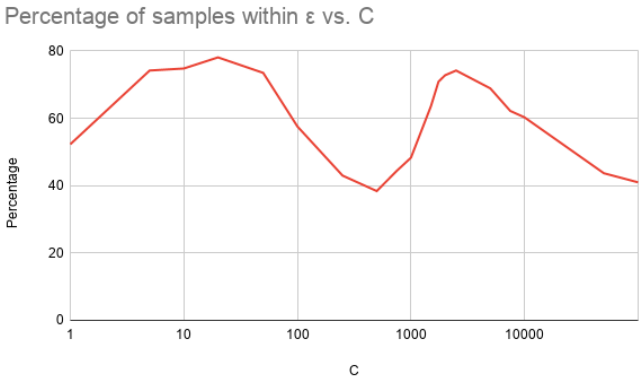
```
P = []
for i in range(14):
  temp_vec = [0.0] * (14 + len(train_data))
  temp_vec[i] = 1.0
  P.append(temp_vec)
rest = [[0.0 for i in range(14 + len(train_data))] for j in range(len(
    train_data))]
P = P + rest

q = [0.0 for i in range(14)] + [c for i in range(len(train_data))]

n = len(train_data)
h = [ephsilon]*(3 * len(train_data))
for i in range(n):
  h[i] = h[i] + y[i]
for i in range(n):
  h[i+n] = h[i+n] - y[i]
for i in range(n):
  h[i+2*n] = h[i+2*n] - ephsilon

n = len(train_data)
G = [0]*(3*n)
for i in range(n):
  temp_vec = [0.0]*(n)
  temp_vec[i] = -1.0
  G[i] = X[i] + temp_vec
for i in range(n):
  temp_vec = [0.0]*(n)
  temp_vec[i] = -1.0
  temp = [0.0]*14
  for j in range(14):
    temp[j] = -X[i][j]
  G[i+n] = temp + temp_vec
for i in range(n):
  temp_vec = [0.0]*(n)
  temp_vec[i] = -1.0
  temp = [0.0]*14
  G[i + 2*n] = temp + temp_vec
G = np.array(G)

sol = solvers.qp(matrix(P), matrix(q), matrix(G, tc = 'd'), matrix(h))
```

2.2 Results:

Table 1: Varying C for a chosen ϵ with a 7:3 :: training: testing validation

S. No.	C	ϵ	MAE	Fraction of sample with $\eta^i = 0$
1.	1.0	5.0	5.895	0.523
2.	5.0	5.0	4.688	0.742
3.	10.0	5.0	4.460	0.748
4.	20.0	5.0	4.295	0.781
5.	50.0	5.0	4.284	0.735
6.	100.0	5.0	5.101	0.576
7.	250.0	5.0	6.195	0.430
8.	500.0	5.0	6.493	0.384
9.	750.0	5.0	6.094	0.444
10.	1000.0	5.0	5.923	0.483
11.	1500.0	5.0	4.846	0.636
12.	1750.0	5.0	4.524	0.709
13.	2000.0	5.0	4.463	0.728
14.	2500.0	5.0	4.299	0.742
15.	5000.0	5.0	4.762	0.689
16.	7500.0	5.0	5.482	0.622
17.	10000.0	5.0	5.929	0.603
18.	50000.0	5.0	8.259	0.437
19.	100000.0	5.0	9.368	0.410





2.3 Implications for practical purposes

The MAE v/s C curve happens to be bi-modal (if were to consider the minimas). However, our aim, besides minimization of the overall error (risk) invloved, is to maximize the percentage of samples within the permissible error range. The corresponding graph happens to have two peaks, one near C = 30 and another one around C = 2500 or C = 3000.

Assuming the problem at hand is the estimation of a function which provides results within a permissible error range, given by ϵ, our aim is to tune the parameter C for the given ϵ such that we are able to estimate maximum number of samples correctly with occasional outliers. Thus, we use graph2 and choose the higher peak. One may argue the ignorance of MAE v/s C curve. Thus, it becomes necessary to clarify that the same hasn't been completely ignored. A high percentage of samples within the permissible range automatically implies a reduction in error, however, it may happen that the point with lesser percentage enjoys lower cumulative error. Herein, we prefer the point with the higher percentage arguing that we prefer more number of samples with a high restricted error with occasional outliers over having a fewer number of samples with lesser restricted errors. This analysis thus allows us to choose C for a given value of ϵ, which in this case happens to be somewhere around 30.

2.4 Inferences:

Table 2: Comparing MAE and '%' samples for Test and Training data

S. No.	C	ϵ	MAE_{Test}	$\text{'\%'}_{Test}$	$\text{MAE}_{Training}$	$\text{'\%'}_{Training}$
1.	1.0	5.0	5.895	0.523	5.560	0.589
2.	5.0	5.0	4.688	0.742	4.988	0.639
3.	10.0	5.0	4.460	0.748	4.701	0.670
4.	20.0	5.0	4.295	0.781	4.219	0.696
5.	50.0	5.0	4.284	0.735	4.009	0.732
6.	100.0	5.0	5.101	0.576	3.813	0.758
7.	250.0	5.0	6.195	0.430	3.539	0.769
8.	500.0	5.0	6.493	0.384	3.426	0.777
9.	750.0	5.0	6.094	0.444	3.338	0.786
10.	1000.0	5.0	5.923	0.483	3.292	0.789
11.	1500.0	5.0	4.846	0.636	3.171	0.808
12.	1750.0	5.0	4.524	0.709	3.13	0.806
13.	2000.0	5.0	4.463	0.728	3.088	0.814
14.	2500.0	5.0	4.299	0.742	3.004	0.808

While we have analysed the performance on test data already, the performance of the model on the training data ends up being quite different as compared to the former case. The MAE in this case decreases almost monotonically as C is increased accompanied by a steady increase with the number of samples within permissible error range. This implies C helps as a regulator for regularization (and hence the generalization) of the model. This

is coherent with what should have been expected. As suggested in the initial stages of this report, SVR is inspired by two ideas, OLR and a SVM. The corresponding objective function for minimization hence contains two terms, one inspired by each of the two ideas. Since the first term (i.e. $\frac{1}{2}||w||^2$) is derived from the regularization term in OLR and the second term is L1 penalty for any deviation from the maximum permissible error, a high value of C results in a compromise of generalization at the cost of lower error rates, thus resulting in **over-fitting** of data.

Similarly, an extremely low value of C means a compromise on the error values of samples and thus gives high MAE values for both test and training data sets. This represents model operation in the **under-fitting** regime. Thus, an optimal value of C is found somewhere in between (happens to be around 30 in this case).

3. Implementation2 - Using sklearn Library:

3.1 Python Code

```
# File2: SVR using customized solvers (from sklearn)
import csv
import numpy as np
from sklearn.svm import SVR
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# INPUT
filename = "BostonHousing.csv"
data = []
with open(filename, 'r') as csvfile:
  csvreader = csv.reader(csvfile)
  for row in csvreader:
    data.append(row)
data = data[1:]
for i in range(len(data)):
  for j in range(14):
    data[i][j] = float(data[i][j])

# Separating Training and Testing data
train_data = data[0:355]
test_data = data[355:]

# Finding the required matrices
X = [0]*(len(train_data))
y = [0]*(len(train_data))
for i in range(len(train_data)):
  X[i] = train_data[i][0:13]
  y[i] = train_data[i][13]
ephsilon = 10.0
c = 1.0

# Normalization:
scale_X = StandardScaler()
X = scale_X.fit_transform(X)

```

```
# Regression
regressor = SVR(kernel = 'linear', C = 100.0, gamma = 'auto', epsilon = 0.1)
regressor.fit(X, y)

# Prediction
y_pred = regressor.predict(X)
```

3.2 Implementation with various Kernels:

3.2.1 Linear Kernel

```
regressor = SVR(kernel = 'linear', C = c, gamma = 'auto', epsilon = epsilon)
```

Table 3: MAE and Fraction of samples with error at most ϵ

S. No.	C	ϵ	MAE	Required Fraction
1.	1.0	5.0	11.210	0.222
2.	5.0	5.0	11.285	0.215
3.	10.0	5.0	11.284	0.215
4.	50.0	5.0	11.286	0.215
5.	100.0	5.0	11.286	0.215
6.	500.0	5.0	11.285	0.215
7.	1000.0	5.0	11.284	0.215

In this implementation, surprisingly, very little variance is observed both in MAE and the required fraction with variance of C for a given ϵ. The program was run on test data to obtain the results.

3.2.2 Radial Basis Function Kernel

```
regressor = SVR(kernel = 'rbf', C = c, gamma = 'auto', epsilon = epsilon)
```

Table 4: MAE and Fraction of samples with error at most ϵ

S. No.	C	ϵ	MAE	Required Fraction
1.	1.0	5.0	9.746	0.212
2.	5.0	5.0	9.932	0.172
3.	10.0	5.0	9.905	0.159
4.	50.0	5.0	9.898	0.152
5.	100.0	5.0	9.958	0.139
6.	500.0	5.0	9.945	0.139
7.	1000.0	5.0	9.945	0.139

Very little variance is observed in MAE though the required fraction shows a steady decrease followed by saturation with increase in C. The program was run on test data to obtain the results.

3.2.3 Polynomial (degree 2) Kernel

```
regressor = SVR(kernel = 'poly', degree = 2, C = c, gamma = 'auto', epsilon =
    epsilon)
```

Table 5: MAE and Fraction of samples with error at most ϵ

S. No.	C	ϵ	MAE	Required Fraction
1.	1.0	5.0	9.745	0.265
2.	5.0	5.0	9.742	0.278
3.	10.0	5.0	9.500	0.298
4.	50.0	5.0	8.892	0.291
5.	100.0	5.0	8.797	0.291
6.	500.0	5.0	9.106	0.272
7.	1000.0	5.0	9.722	0.291

Once again, the required percentage tends to saturate preceded by a steady decrease. MAE on the other hand tends to hover around a certain value with an occasional decrease from the central tendency. The results were obtained on test data.

3.3 Inferences

The sklearn SVR optimizer tends to give similar results whatever may the test data be, suggesting achievement of a good amount of generalization of the estimated function. However, the optimizer seems to have been programmed to expect larger data-sets and thus, tends to under-fit the current training set of 355 samples, tested on 151 other samples.

This thus motivates us to train the model on the entire data-set. As can be implied from the above results, our data isn't large enough to obtain a good fit and the model is nicely adapted to render generalized solutions.

As far as the kernel choice is concerned, linear kernel gave higher MAE values than rbf kernel. However, results of the 'Required Fraction' column suggested rbf kernel to be better. Nevertheless, the 2nd degree polynomial performed well in the both the departments and is hence our choice when the model is run on the entire data.

The following results show remarkable improvement from the previous ones. This strongly suggests that the sklearn optimizer is data hungry and the larger the data set the better (to a very good extent!). Given the power to obtain a generalized estimate of the required function besides being compatible with various kernels makes SVR a strong and indispensable tool for the associated problems. *(Please find the table on the next page.)*

Table 6: MAE and Fraction of samples with error at most ϵ

S. No.	C	ϵ	MAE	Required Fraction
1.	1.0	5.0	4.593	0.672
2.	5.0	5.0	3.914	0.733
3.	10.0	5.0	3.689	0.737
4.	50.0	5.0	3.445	0.796
5.	100.0	5.0	3.412	0.792
6.	500.0	5.0	3.283	0.800
7.	1000.0	5.0	3.310	0.790

4. Conclusion:

SVR was successfully implemented in the required ways using various python libraries. Various differences and similarities in the two cases were noted and explained on the basis of the background theory. The End.

References

- **Medium Article by Tom Sharp** - https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2
- **For sklearn documentation** - https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/
- **Mariette Awad et Rahul Khanna's Efficient Learning Machines** - https://link.springer.com/chapter/10.1007/978-1-4302-5990-94
- **ELL409 assignment 3 PDF** - And listed references