

Assignment 2: Regression

CS 589 - ML

Shubham Shetty (shubhamshett@umass.edu)
Brinda Murulidhara (bmurulidhara@umass.edu)
Adarsh Kolya (akolya@umass.edu)

March 2021

Preface

Before running our codes, the following packages are imported and test & training data are assigned to variables. Also, some reusable functions are defined -

```
1 # Import Statements
2 import numpy as np
3 import sklearn as sk
4 from sklearn.neighbors import KNeighborsRegressor
5 from sklearn.tree import DecisionTreeRegressor
6 from sklearn import linear_model
7 from sklearn.metrics import mean_squared_error, mean_absolute_error
8 import pandas as pd
9 import matplotlib.pyplot as plt
10 import math
11
12
13 # Load Data
14 all_data = np.load("data.npz")
15 big_data = np.load("big_data.npz")
16
17
18 # Initialising for data.npz
19
20 ## 3d training inputs
21 X_trn = all_data["X_trn"]
22 ## 1d training outputs
23 y_trn = all_data["y_trn"]
24 ## Test data inputs
25 X_val = all_data["X_val"]
26 ## Test data outputs
27 y_val = all_data["y_val"]
28
29
30 # Initialising for big_data.npz
31
32 ## 3d training inputs
33 X_big_trn = big_data["X_trn"]
34 ## 1d training outputs
35 y_big_trn = big_data["y_trn"]
36 ## Test data inputs
37 X_big_val = big_data["X_val"]
38 ## Test data outputs
39 y_big_val = big_data["y_val"]
40
41
42 # Common Functions
43
44 ## Euclidean Distance
45 def euc_dist(p1, p2, n):
46     tot = 0
```

```
47     for i in range(n):
48         tot += (p1[i] - p2[i])**2
49     return math.sqrt(tot)
50
51 ## Loss Functions
52 ### Mean Squared Error
53 def sq_error(a,b):
54     n = len(a)
55     err = 0
56     for i in range(n):
57         err += (a[i] - b[i])**2
58     return err/n
59 ### Mean Absolute Error
60 def abs_error(a,b):
61     n = len(a)
62     err = 0
63     for i in range(n):
64         err += abs(a[i] - b[i])
65     return err/n
```

Answer 1

Following function runs K-Nearest Neighbours regression on given datasets -

```
1 # KNN Regression
2 def KNN_reg_predict(X_trn, y_trn, x, K):
3
4     # Algorithm -
5     # 1. Calculate distance of x from each element of training data
6     # 2. Find K closest elements
7     # 3. Take average of y for closest elements as final prediction
8
9     # Calculating euclidean distance for each element
10    dist_vector = []
11    for X in X_trn:
12        dist_vector.append(euc_dist(X, x, 3))
13
14    # Sorting by distance to find closest neighbours
15    dist_vector2 = list(enumerate(dist_vector))
16    dist_sorted = sorted(dist_vector2, key=lambda x:x[1])
17
18    # Taking average to return final prediction
19    y_new=0
20    for i in range(K):
21        y_new += y_trn[dist_sorted[i][0]]/K
22    return y_new
```

Answer 2

First, we calculate MSE for training and test data using our KNN function.

```
1 # MSE for KNN Regression
2 def KNN_reg_sq_error(X_trn, y_trn, X_val, y_val, K):
3     eval_sq_error = []
4     for i in range(1, K+1):
5         prediction_trn = [KNN_reg_predict(X_trn, y_trn, j, i) for j
6             in X_trn]
7         prediction_tst = [KNN_reg_predict(X_trn, y_trn, j, i) for j
8             in X_val]
9         eval_sq_error.append([(i), sq_error(prediction_trn, y_trn),
10             sq_error(prediction_tst, y_val)])
11     df = pd.DataFrame(np.array(eval_sq_error), columns=[ 'K' , 'Training
12         Data Error' , 'Test Data Error'])
13     return(df)
```

Following is the observed values for MSE -

K	Training Data Error	Test Data Error
1.0	0.000000	0.031462
2.0	0.008033	0.022590
3.0	0.011284	0.020367
4.0	0.013460	0.020312
5.0	0.014874	0.020941
6.0	0.014836	0.021199
7.0	0.015653	0.021181
8.0	0.015776	0.021703
9.0	0.017731	0.022148
10.0	0.018331	0.022652

Table 1: MSE Values for KNN Regression



Next, we calculate MAE for training and test data using our KNN function.

```

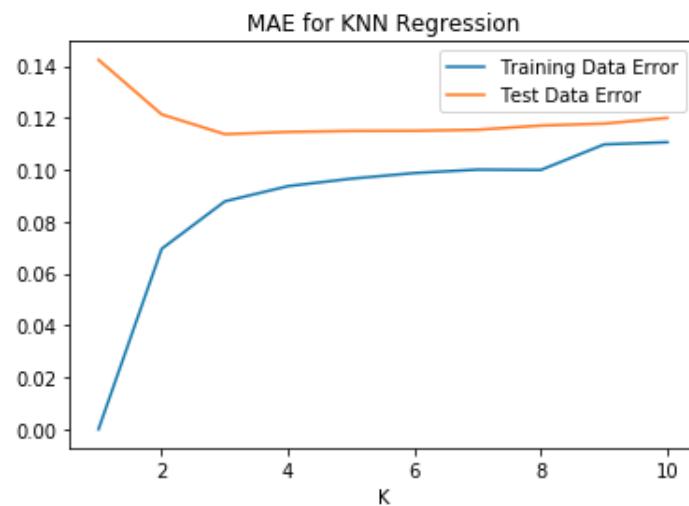
1 # MAE for KNN Regression
2 def KNN_reg_abs_error(X_trn, y_trn, X_val, y_val, K):
3     eval_abs_error = []
4     for i in range(1, K+1):
5         prediction_trn = [KNN_reg_predict(X_trn, y_trn, j, i) for j
6             in X_trn]
6         prediction_tst = [KNN_reg_predict(X_trn, y_trn, j, i) for j
7             in X_val]
7         eval_abs_error.append([int(i), abs_error(prediction_trn,
y_trn), abs_error(prediction_tst,y_val)])
8     df = pd.DataFrame(np.array(eval_abs_error),columns=[ 'K', ,
9     'Training Data Error', 'Test Data Error'])
9     return(df)

```

Following is the observed values for MAE -

K	Training Data Error	Test Data Error
1.0	0.000000	0.142356
2.0	0.069524	0.121296
3.0	0.087810	0.113677
4.0	0.093670	0.114556
5.0	0.096531	0.114927
6.0	0.098717	0.114976
7.0	0.100032	0.115389
8.0	0.099904	0.117003
9.0	0.109744	0.117753
10.0	0.110578	0.119928

Table 2: MAE Values for KNN Regression



Answer 3

Following code is a trivial method to evaluate a linear regression model -

```
1 # Linear Ridge Regression
2 def linear_reg_predict(x, w):
3     y = np.dot(x, w)
4     return y
```

Answer 4

Following method trains a ridge regression model -

```
1 # Training Ridge Regression Model
2 def linear_reg_train(X_trn, y_trn, l):
3     xt = X_trn.transpose()
4     xtx = np.matmul(xt, X_trn)
5     xty = np.matmul(xt, y_trn)
6     li = l*np.identity(X_trn.shape[1])
7     w = np.linalg.solve(xtx+li, xty)
8     return w
```

Answer 5

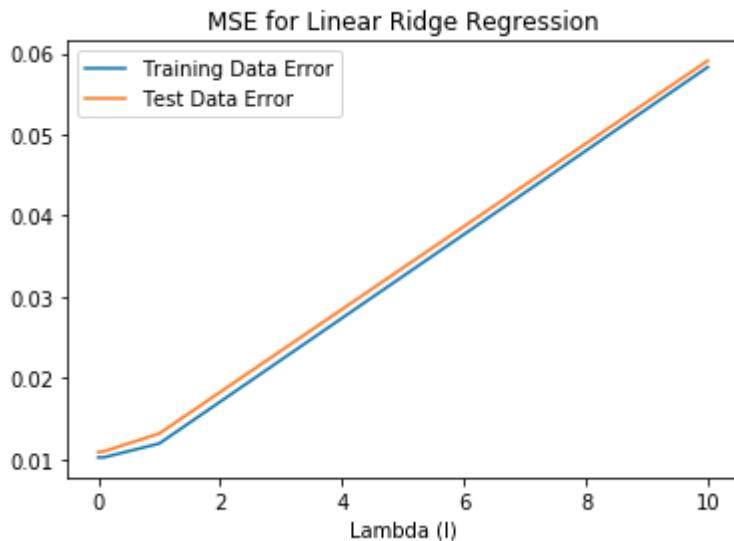
First, we calculate MSE for training and test data using our Ridge Regression model -

```
1 # MSE for Linear Ridge Regression
2 def linear_reg_sq_error(X_trn, y_trn, X_val, y_val):
3     l = [0, 0.001, 0.01, 0.1, 1, 10]
4     eval_sq_error = []
5     for i in l:
6         linear_reg_trn = linear_reg_predict(X_trn, linear_reg_train(
X_trn, y_trn, i))
7         linear_reg_tst = linear_reg_predict(X_val, linear_reg_train(
X_trn, y_trn, i))
8         eval_sq_error.append([i, sq_error(linear_reg_trn, y_trn),
sq_error(linear_reg_tst, y_val)])
9     df = pd.DataFrame(np.array(eval_sq_error), columns=[ 'Lambda (l)', 'Training Data Error', 'Test Data Error'])
10    return(df)
```

Following is the observed values for MSE -

Lambda (l)	Training Data Error	Test Data Error
0.000	0.010242	0.010909
0.001	0.010242	0.010910
0.010	0.010243	0.010917
0.100	0.010263	0.011005
1.000	0.011953	0.013187
10.000	0.058293	0.059082

Table 3: MSE Values for Linear Ridge Regression



Next, we calculate MAE for training and test data using our Ridge Regression model -

```

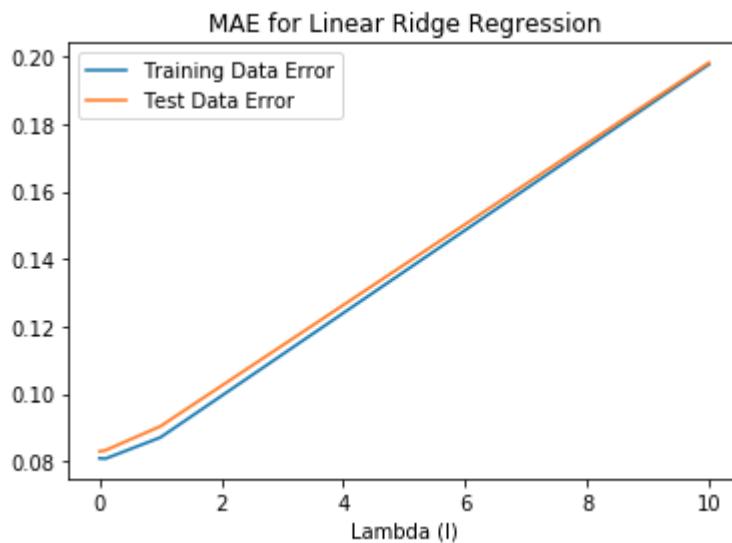
1 # MAE for Linear Ridge Regression
2 def linear_reg_abs_error(X_trn, y_trn, X_val, y_val):
3     l = [0, 0.001, 0.01, 0.1, 1, 10]
4     eval_abs_error = []
5     for i in l:
6         linear_reg_trn = linear_reg_predict(X_trn, linear_reg_train(
7             X_trn, y_trn, i))
8         linear_reg_tst = linear_reg_predict(X_val, linear_reg_train(
9             X_trn, y_trn, i))
10        eval_abs_error.append([i, abs_error(linear_reg_trn, y_trn),
11                               abs_error(linear_reg_tst, y_val)])
12    df = pd.DataFrame(np.array(eval_abs_error), columns=[ 'Lambda (l)', 'Training Data Error', 'Test Data Error'])
13    return(df)

```

Following is the observed values for MAE -

Lambda (l)	Training Data Error	Test Data Error
0.000	0.080886	0.082917
0.001	0.080884	0.082920
0.010	0.080871	0.082945
0.100	0.080776	0.083253
1.000	0.087064	0.090350
10.000	0.197639	0.198164

Table 4: MAE Values for Linear Ridge Regression



Answer 6

1. (A) Squared training error increases as λ increases.

As λ increases, the regression coefficients (w) decrease which indicates that the model is getting less complex (higher bias) which results in greater error.

2. (C) For squared test error, error goes down for a while, then goes up, and may stay constant in either phase.

As λ increases, the coefficients decrease and some coefficients may be nearly 0. As a result, the overall function becomes less complex and overfitting reduces. The MSE decreases. However, on further increasing lambda, overfitting may reduce to an extent that the data is now under fitted. Hence, the MSE increases.

3. (A) Absolute training error increases as λ increases

For the same reason as in 1 (increase in bias due to increasing λ), absolute training error increases with an increase in λ .

4. (C) For absolute test error, error goes down for a while, then goes up, and may stay constant in either phase.

For same reason as in 2, we see that absolute error will reduce initially, reach a minima or optimal point, and then increases as λ increases.

Answer 7

Following method evaluates a regression stump. It returns c_left if $x[\text{dim}] \leq \text{thresh}$ and c_right otherwise.

```
1 # Predicting Value via Regression Stump
2 def reg_stump_predict(x, dim, thresh, c_left, c_right):
3     y = []
4     for i in x:
5         if i[dim] <= thresh:
6             y.append(c_left)
7         else:
8             y.append(c_right)
9     return y
```

Answer 8

Following method trains a regression stump -

```
1 # Training Regression Stump
2 def reg_stump_train(X_trn, y_trn):
3
4     # Algorithm -
5     # 1. For each dimension, sort the data.
6     # 2. Create potential split points as average of adjacent
7     # elements in sorted data.
8     # 3. Define regions with points on either side of split (R1 if
9     # x_i < s else R2).
10    # 4. Find optimal constants c1 & c2
11    # 5. Calculate error for each split.
12    # 6. Return split values for which minimum error is obtained
13    # across the dimensions.
14
15    # Get number of dimensions in input
16    D = X_trn.shape[1]
17
18    # Initialise error as infinity
19    err = float("inf")
20
21    for j in range(D):
22        s = []
23        x = X_trn[:,j]
24        z = np.sort(x)
25
26        # Get potential splits
27        for i in range(len(z)-1):
28            s.append(0.5*(z[i]+z[i+1]))
29
30        for split in s:
31            y1, y2 = 0, 0
32            r1, r2 = [], []
33            # Divide all points into two regions
34            for xn in range(len(x)):
35                if x[xn] < split:
36                    r1.append(xn)
37                    y1 += y_trn[xn]
38                else:
39                    r2.append(xn)
40                    y2 += y_trn[xn]
41
42            # Get constant values for both regions
43            c1 = y1/len(r1)
44            c2 = y2/len(r2)
45
46            # Calculate error
47            err2 = sum([(y_trn[i] - c1)**2 for i in r1]) + sum([(y_trn[i] - c2)**2 for i in r2])
48
49    return s, err2
```

```
46         # Replace terms if error is minimum
47         if(err2<err):
48             err = err2
49             dim = j
50             thresh = split
51             c_left = c1
52             c_right = c2
53
54     return dim, thresh, c_left, c_right
```

Answer 9

Following code generates MSE and MAE for the given training and test data as per the regression stump method defined earlier -

```
1 # MSE & MAE for Regression Stump
2 def reg_stump_sq_error(X_trn, y_trn, X_val, y_val):
3     eval_error = []
4     dim, thresh, c_left, c_right = reg_stump_train(X_trn, y_trn)
5     pred_trn = reg_stump_predict(X_trn, dim, thresh, c_left, c_right)
6     pred_tst = reg_stump_predict(X_val, dim, thresh, c_left, c_right)
7     eval_error.append([sq_error(pred_trn, y_trn), sq_error(pred_tst,
8         y_val)])
9     eval_error.append([abs_error(pred_trn, y_trn), abs_error(pred_tst
10 , y_val)])
11     df = pd.DataFrame(np.array(eval_error), columns=['Training Data
12 Error', 'Test Data Error'], index=['MSE', 'MAE'])
13     return df
14
15
16 # Generate Table
17 reg_stump_sq_error(X_trn, y_trn, X_val, y_val)
```

	Training Data Error	Test Data Error
MSE	0.097324	0.123254
MAE	0.255508	0.288270

Table 5: MSE & MAE Values for Rigression Stump

Answer 10

Following method evaluates KNN Regression on real big data for various values of K. This method utilises KNeighborsRegressor module from scikit-learn.

```
1 # KNN Regression for real data
2 def KNN_reg_real(X_big_trn, y_big_trn, X_big_val, y_big_val):
3     K = [1, 2, 5, 10, 20, 50]
4     sq_errs = []
5     for i in K:
6         neigh = KNeighborsRegressor(n_neighbors=i)
7         neigh.fit(X_big_trn, y_big_trn)
8         sq_errs.append([i, mean_squared_error(y_big_trn, neigh.
predict(X_big_trn)), mean_squared_error(y_big_val, neigh.predict(
X_big_val))])
9     df = pd.DataFrame(np.array(sq_errs), columns=[ 'K', 'Training Data
Error', 'Test Data Error'])
10    return df
11
12 # Generate Table
13 df = KNN_reg_real(X_big_trn, y_big_trn, X_big_val, y_big_val)
14 print(df)
15 df.plot(x="K", title="MSE for KNN Regression on Big Data");
```

K	Training Data Error	Test Data Error
1.0	0.000000	0.063055
2.0	0.013516	0.051343
5.0	0.029755	0.047411
10.0	0.043787	0.054627
20.0	0.059547	0.067552
50.0	0.083329	0.096843

Table 6: MSE Values for KNN Regression on Big Data



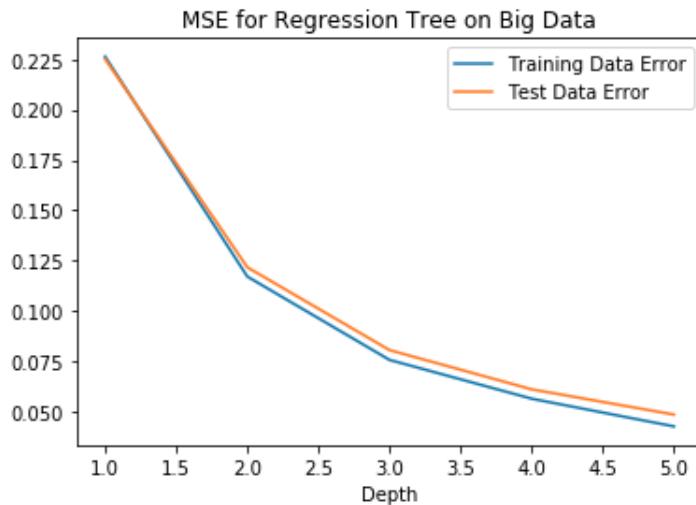
Answer 11

Following method evaluates Regression Tree on real big data for various depths. This method utilises DecisionTreeRegressor module from scikit-learn.

```
1 # Regression Tree for real data
2 def reg_tree_real(X_big_trn, y_big_trn, X_big_val, y_big_val):
3     max_depth = 5
4     sq_errs = []
5
6     # Fit model
7     for i in range(1,max_depth+1):
8         tree1 = DecisionTreeRegressor(max_depth=i)
9         tree1.fit(X_big_trn, y_big_trn)
10        # Predict
11        y_pred_trn = tree1.predict(X_big_trn)
12        y_pred_tst = tree1.predict(X_big_val)
13        # Errors
14        sq_errs.append([i, mean_squared_error(y_big_trn, y_pred_trn),
15                        mean_squared_error(y_big_val, y_pred_tst)])
16    df = pd.DataFrame(np.array(sq_errs),columns=['Depth', 'Training
Data Error', 'Test Data Error'])
17    return df
```

Depth	Training Data Error	Test Data Error
1.0	0.226337	0.225094
2.0	0.117116	0.121711
3.0	0.075729	0.080604
4.0	0.056466	0.061122
5.0	0.042719	0.048591

Table 7: MSE Values for Regression Tree on Big Data



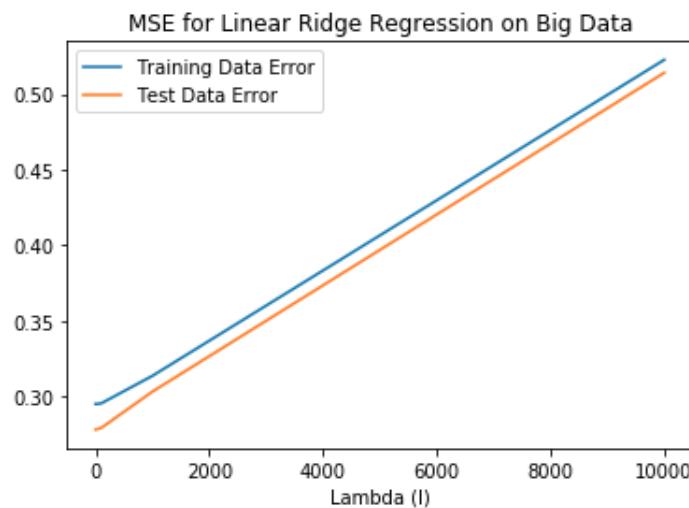
Answer 12

Following method evaluates Linear Ridge Regression on real big data for various lambda values. This method utilises linear_model module from scikit-learn.

```
1 # Linear Ridge Regression for real data
2 def linear_ridge_reg_real(X_big_trn, y_big_trn, X_big_val, y_big_val):
3     :
4     l = [0, 1, 10, 100, 1000, 10000]
5     eval_sq_error = []
6     for i in l:
7         ridge_reg = linear_model.Ridge(alpha=i)
8         ridge_reg.fit(X_big_trn, y_big_trn)
9         linear_reg_trn = linear_reg_predict(X_big_trn, ridge_reg.
10         coef_)
11         linear_reg_tst = linear_reg_predict(X_big_val, ridge_reg.
12         coef_)
13         eval_sq_error.append([i, mean_squared_error(linear_reg_trn,
14             y_big_trn), mean_squared_error(linear_reg_tst, y_big_val)])
15     df = pd.DataFrame(np.array(eval_sq_error), columns=[ 'Lambda (l)' ,
16         'Training Data Error', 'Test Data Error'])
17     return(df)
```

Lambda (l)	Training Data Error	Test Data Error
0.0	0.294976	0.278076
1.0	0.294976	0.278082
10.0	0.294981	0.278144
100.0	0.295388	0.279136
1000.0	0.313595	0.303158
10000.0	0.522598	0.514033

Table 8: MSE Values for Linear Ridge Regression on Big Data



Answer 13

Following method evaluates Linear Lasso Regression on real big data for various lambda values. This method utilises linear_model module from scikit-learn.

```
1 # Linear Lasso Regression for real data
2 def linear_lasso_reg_real(X_big_trn, y_big_trn, X_big_val, y_big_val):
3     :
4     l = [0, 0.1, 1, 10, 100, 1000]
5     N = X_big_trn.shape[0]
6     eval_sq_error = []
7     for i in l:
8         lasso_reg = linear_model.Lasso(alpha=i/(2*N))
9         lasso_reg.fit(X_big_trn, y_big_trn)
10        linear_reg_trn = lasso_reg.predict(X_big_trn)
11        linear_reg_tst = lasso_reg.predict(X_big_val)
12        eval_sq_error.append([i, mean_squared_error(linear_reg_trn,
13 y_big_trn), mean_squared_error(linear_reg_tst,y_big_val)])
14    df = pd.DataFrame(np.array(eval_sq_error),columns=[ 'Lambda (l)' ,
15 'Training Data Error', 'Test Data Error'])
16    return(df)
```

Lambda (l)	Training Data Error	Test Data Error
0.0	0.294967	0.278102
0.1	0.294967	0.278104
1.0	0.294968	0.278118
10.0	0.294998	0.278288
100.0	0.296185	0.280887
1000.0	0.342903	0.338830

Table 9: MSE Values for Linear Lasso Regression on Big Data

