CS F320 : FODS

Assignment 1

Date : 29th October, 2021

Team Members

R Adarsh	2019A7PS0230H
Abhinav Talesra	2019AAPS0223H
Sahaj Gupta	2019A7PS0148H

Problem 1 Polynomial Regression

1.1. Model Description and implementation

Dataset given:

The dataset consists of two features i.e. ‘Strength’ and ‘Temperature’ applied to a certain piece of plastic. Expected to predict how much ‘Pressure’ that the plastic can stand. It consists of 1650 rows.

Data Processing:
Data processing is the collection and manipulation of items of data to produce meaningful information. Pre-process your data includes :

- Shuffling the data
- Standardizing/normalizing the values

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)} \qquad x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- Creating a random 70-30 split to aid in training and testing respectively.
- Creating polynomial features depending on the degree of the polynomial. A polynomial with degree d will have (d+2)C2 features.

Loss Function:
Loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. The error function used for regression task is mean squared error given by the formula,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Optimization:

Our objective is to find the optimum set of parameters that minimize the cost function. We have Gradient Descent and Stochastic Batch gradient descent as optimization algorithms.

A) Standard/ Batch Gradient Descent:

1. We start by initializing random weights
2. In each iteration we update the parameters,

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

3. By iterating over the training examples until convergence we are able to reach the optimum parameters that minimize the cost.

B) Stochastic Gradient Descent:

Usually when we deal with bigger datasets, Gradient Descent turns out to be slow computationally since for every iteration it passes the whole dataset to calculate the gradient of loss. Hence we use SGD in which we compute the gradient for just a single example at each iteration. This brings down the computation time.

1. Pick a single data point
2. Compute gradient over that single point
3. Update parameters

1.3. The final train and test metrics

1.3.1 Polynomial regression with degrees 0-9 using GD and SGD

GD (Learning rate : 0.1|| # iterations = 10000 || Converging criteria = |curr_error - prev_error| <= 5e-5)

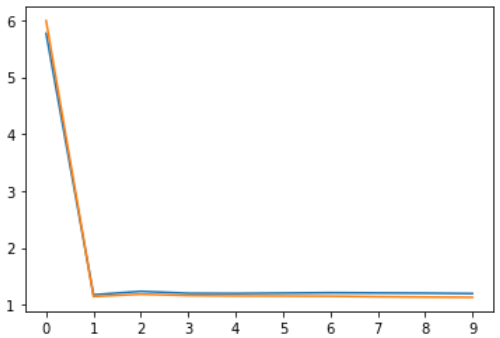
	Degree	Train Error	Avg Train Error	Test Error	Avg Test Error	Min Avg Train Error	Min Avg Test error	W
0	0	6662.798268	5.768657	2966.061348	5.992043	5.768657	5.992043	[14.931601731601724]
1	1	1358.874650	1.176515	567.044875	1.145545	1.176515	1.145382	[21.012254487641624, 16.126840363543394, -28.8...
2	2	1428.010618	1.236373	587.332523	1.186530	1.236373	1.186530	[19.949431569181137, 12.485295323911634, 2.312...
3	3	1388.652869	1.202297	575.552090	1.162731	1.202297	1.162731	[20.228220282333716, 13.366836714423343, 2.669...
4	4	1385.743549	1.199778	572.354672	1.156272	1.199778	1.156272	[20.168350508775248, 13.379371068301175, 4.104...
5	5	1393.093515	1.206142	571.617712	1.154783	1.206142	1.154783	[19.976654983562447, 13.411743448041939, 4.002...
6	6	1399.882561	1.212020	569.953966	1.151422	1.212020	1.151422	[19.87359519474215, 12.96177715071445, 4.13499...
7	7	1396.045925	1.208698	565.723490	1.142876	1.208698	1.142876	[19.885917905781888, 12.892151720401223, 3.908...
8	8	1391.460141	1.204727	562.620594	1.136607	1.204727	1.136607	[19.87822086144206, 12.858329261116408, 3.8837...
9	9	1383.348464	1.197704	560.230505	1.131779	1.197704	1.131779	[19.929714110605413, 12.986854431359077, 3.499...

SGD (Learning rate : 0.1|| # iterations = 5000 || Converging criteria = |curr_error - prev_error| <= 5e-5**)**

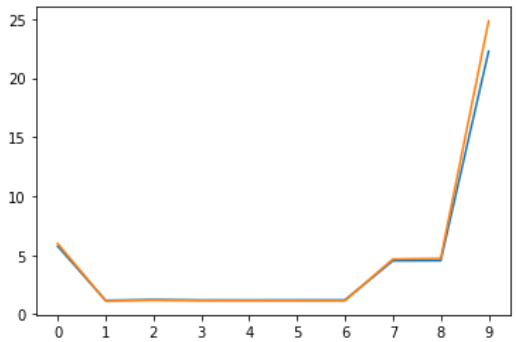
	Degree	Train Error	Avg Train Error	Test Error	Avg Test Error	Min Avg Train Error	Min Avg Test error	W
0	0	6688.247328	5.790690	2953.276786	5.966216	5.790690	5.966216	[14.931601731601724]
1	1	1380.493247	1.195232	585.356232	1.182538	1.194860	1.181514	[21.01222231800195, 16.126745641069185, -28.89...
2	2	1397.782465	1.210201	602.821714	1.217822	1.210201	1.217822	[19.94685783398814, 12.4577262155485, 2.258464...
3	3	1422.941328	1.231984	620.215217	1.252960	1.231984	1.252573	[20.23638824077044, 13.39763907981587, 2.50696...
4	4	1411.218326	1.221834	615.436549	1.243306	1.221834	1.243306	[20.199006065134416, 13.471998000076209, 3.882...
5	5	1453.609394	1.258536	636.636631	1.286135	1.258536	1.277665	[20.017485797105937, 13.25743611843667, 4.1102...
6	6	1471.689923	1.274190	644.366713	1.301751	1.267287	1.262584	[19.865420323080457, 13.034729522066481, 3.833...
7	7	1512.738528	1.309730	663.712179	1.340833	1.256066	1.231779	[19.84685210749558, 13.05138818551076, 3.53516...
8	8	1439.093237	1.245968	610.806353	1.233952	1.245968	1.231971	[19.926670176562027, 12.819613951024143, 3.860...
9	9	1435.259987	1.242649	600.524824	1.213181	1.242649	1.205040	[19.95336453729375, 13.148349675254392, 3.1355...

MSE VS DEGREE PLOT

* On X-axis = MSE
  On Y-axis = degree of polynomial



learning rate = 0.1



learning rate = 0.00001

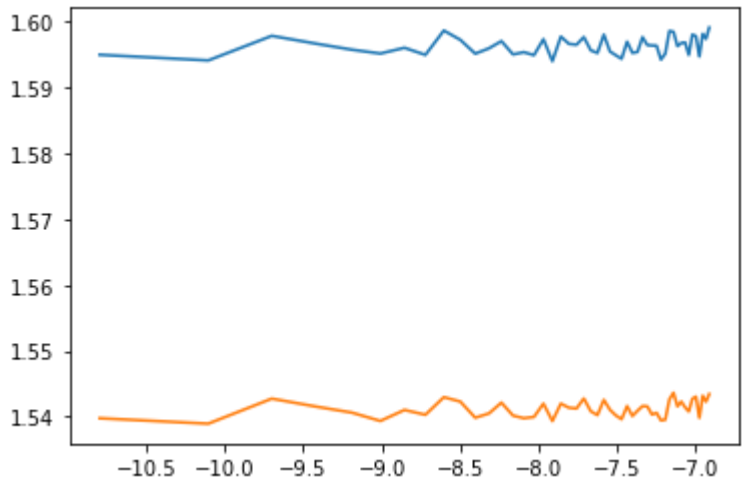
1.3.2 Lasso and Ridge Regression with polynomial of degree 9

GD -LASSO(Learning rate : 0.05|| # iterations = 10000 || Converging criteria = |curr_error - prev_error| <= 5e-5**)**

	Lambda	Avg Train Error	Avg Test Error	RMSE_Train	RMSE_test
0	0.000000	1.277395	1.190189	1.598371	1.542847
1	0.000020	1.271960	1.185432	1.594967	1.539761
2	0.000041	1.270593	1.184160	1.594110	1.538935
3	0.000061	1.276547	1.190021	1.597841	1.542738
4	0.000082	1.274620	1.188127	1.596634	1.541510
5	0.000102	1.273196	1.186755	1.595742	1.540620
6	0.000122	1.272261	1.184799	1.595156	1.539350
7	0.000143	1.273648	1.187371	1.596025	1.541020
8	0.000163	1.271935	1.186238	1.594951	1.540285
9	0.000184	1.277830	1.190365	1.598643	1.542961
10	0.000204	1.275581	1.189331	1.597235	1.542291

RMSE (X-axis) Vs ln(Lambda) (Y-axis) plot.

* BLUE : Train RMSE | ORANGE : Test RMSE

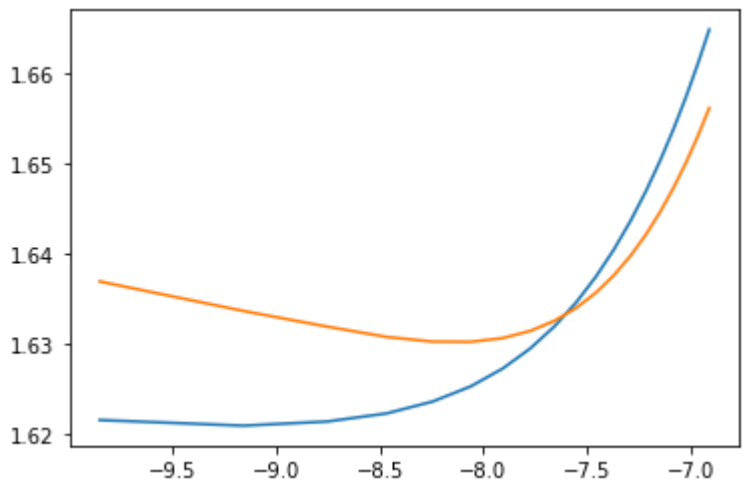


SGD-LASSO (Learning rate : 0.05|| # iterations = 5000 || Converging criteria = |curr_error - prev_error| <= 5e-5**)**

	Lambda	Avg Train Error	Avg Test Error	RMSE_Train	RMSE_test
0	0.000000	1.330465	1.364530	1.631236	1.651987
1	0.000053	1.314778	1.339823	1.621590	1.636962
2	0.000105	1.313762	1.334477	1.620964	1.633693
3	0.000158	1.314509	1.331633	1.621424	1.631952
4	0.000211	1.315969	1.329765	1.622325	1.630807
5	0.000263	1.318119	1.328917	1.623650	1.630287
6	0.000316	1.320832	1.328881	1.625319	1.630264
7	0.000368	1.324032	1.329551	1.627287	1.630675
8	0.000421	1.327668	1.330847	1.629520	1.631470
9	0.000474	1.331692	1.332695	1.631988	1.632602
10	0.000526	1.336059	1.335033	1.634661	1.634034

RMSE (X-axis) Vs ln(Lambda) (Y-axis) plot.

* BLUE : Train RMSE | ORANGE : Test RMSE

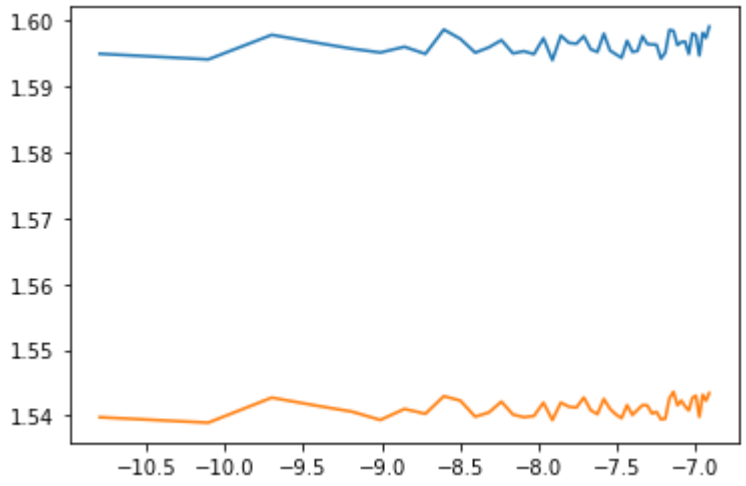


GD -RIDGE (Learning rate : 0.051|| # iterations = 10000 || Converging criteria = |curr_error - prev_error| <= 5e-5**)**

	Lambda	Avg Train Error	Avg Test Error	RMSE_Train	RMSE_test
0	0.000000	1.277395	1.190189	1.598371	1.542847
1	0.000020	1.271960	1.185432	1.594967	1.539761
2	0.000041	1.270593	1.184160	1.594110	1.538935
3	0.000061	1.276547	1.190021	1.597841	1.542738
4	0.000082	1.274620	1.188127	1.596634	1.541510
5	0.000102	1.273196	1.186755	1.595742	1.540620
6	0.000122	1.272261	1.184799	1.595156	1.539350
7	0.000143	1.273648	1.187371	1.596025	1.541020
8	0.000163	1.271935	1.186238	1.594951	1.540285
9	0.000184	1.277830	1.190365	1.598643	1.542961
10	0.000204	1.275581	1.189331	1.597235	1.542291

RMSE (X-axis) Vs ln(Lambda) (Y-axis) plot.

* BLUE : Train RMSE | ORANGE : Test RMSE

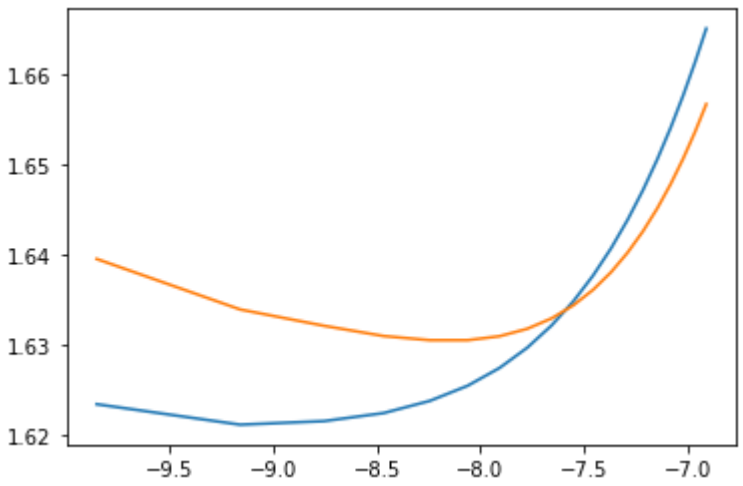


SGD -RIDGE (Learning rate : 0.05|| # iterations = 5000 || Converging criteria = |curr_error - prev_error| <= 5e-5**)**

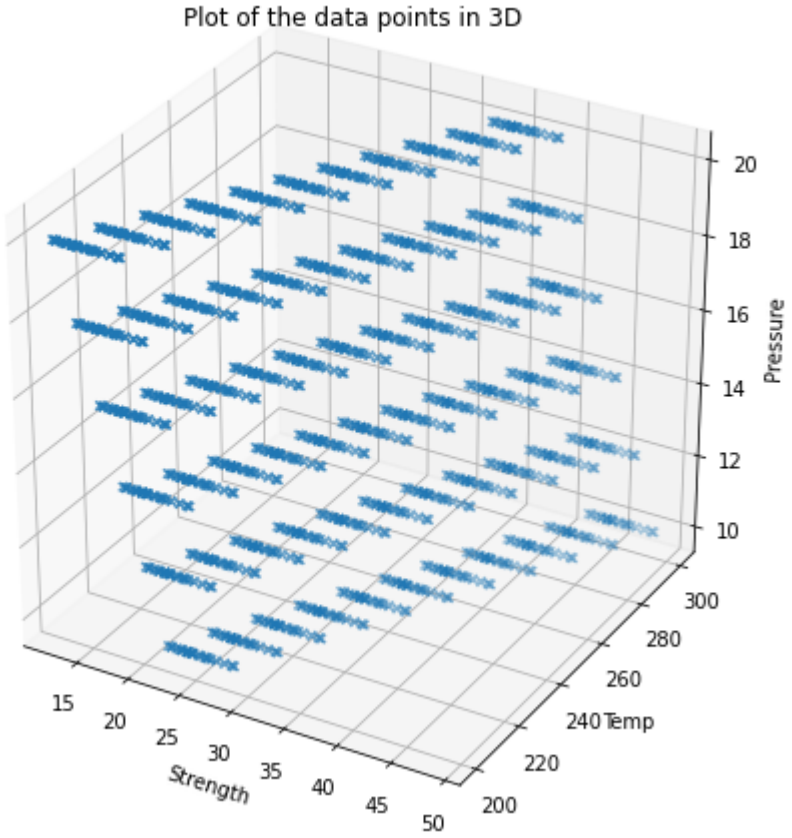
	Lambda	Avg Train Error	Avg Test Error	RMSE_Train	RMSE_test
0	0.000000	1.331010	1.365415	1.631570	1.652522
1	0.000053	1.317648	1.343989	1.623359	1.639506
2	0.000105	1.313955	1.334797	1.621083	1.633889
3	0.000158	1.314601	1.331831	1.621482	1.632073
4	0.000211	1.316051	1.329974	1.622376	1.630935
5	0.000263	1.318252	1.329225	1.623732	1.630475
6	0.000316	1.320978	1.329225	1.625409	1.630475
7	0.000368	1.324199	1.329942	1.627390	1.630915
8	0.000421	1.327857	1.331286	1.629636	1.631739
9	0.000474	1.331898	1.333176	1.632114	1.632897
10	0.000526	1.336283	1.335556	1.634798	1.634354

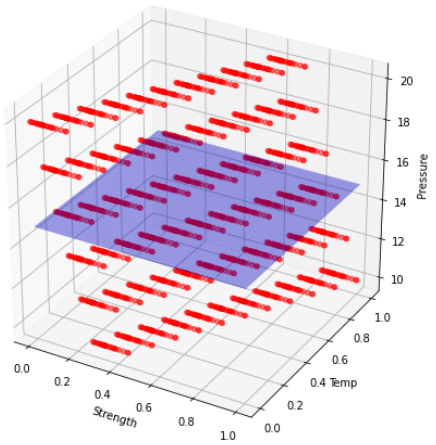
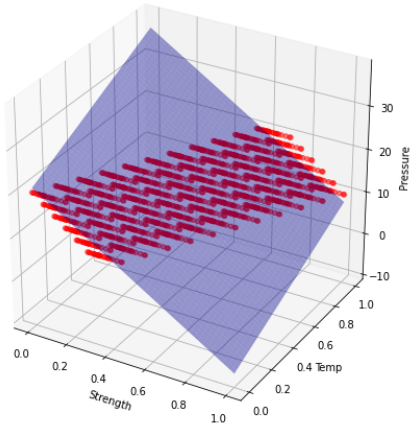
RMSE (X-axis) Vs ln(Lambda) (Y-axis) plot.

* BLUE : Train RMSE | ORANGE : Test RMSE

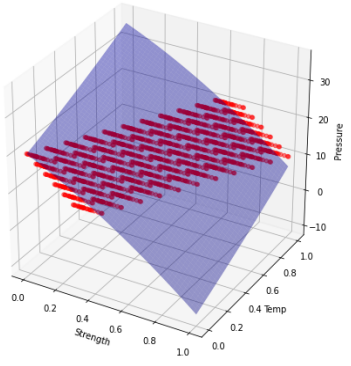


1.4. Surface Plots for different degrees of polynomial using weights from GD

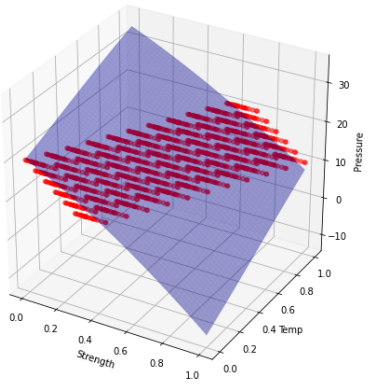


DEGREE 0	DEGREE 1
	

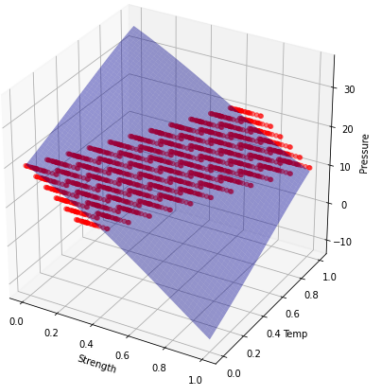
DEGREE 2



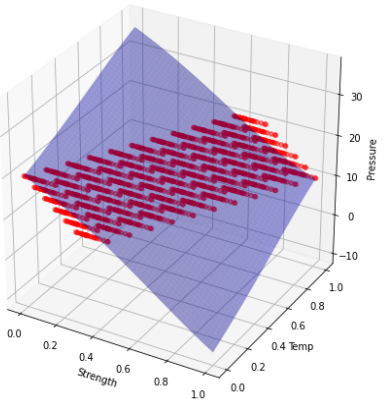
DEGREE 3



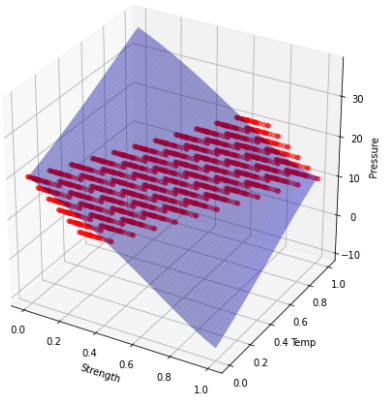
DEGREE 4



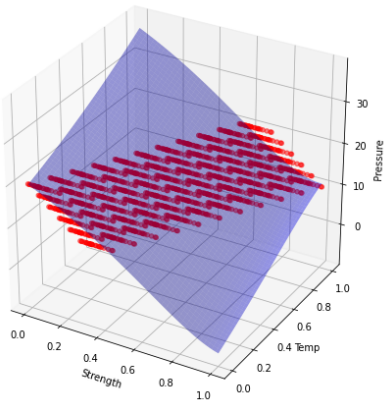
DEGREE 5

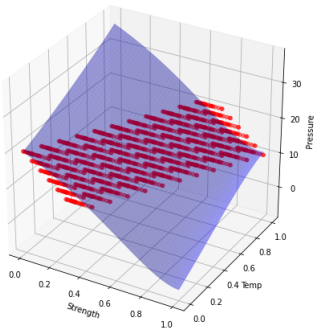
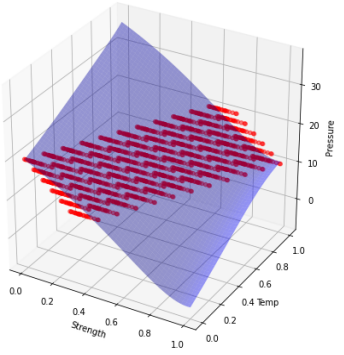


DEGREE 6



DEGREE 7



DEGREE 8	DEGREE 9
	

1.5. RESULTS AND CONCLUSIONS:

A) From the MSE values obtained after running the gradient descent and SGD algorithm with appropriate learning rates to fit polynomials of degrees varying between 0 to 9 on the given data points, we get lower training and testing error for polynomials of degrees 1,2,3, 4,5 with minimum being for that of degree 1. So for the given data the best fit would be a polynomial of degree 1.

Higher degree polynomials like degee 7,8,9 we get slightly higher training errors for some learning rates and their testing error is higher than training error indicating the presence of some overfitting. Intuitively, with more flexible polynomials of higher degrees become increasingly tuned to random noise on target values.

B) After applying regularization, we see that the MSE for polynomials of degree 9 has reduced slightly as compared to those values without regularization. From the values it seems that the model with regularization seems to be performing better than that of linear degree in some cases. Thus a complex model with regularization with appropriate lambda may perform better than a simple linear model to fit the points. We see that as lambda increases, the weights get smaller and the testing error reduces thereby suppressing the effect of overfitting. The RMSE VS ln(lambda) graph shows that the desired effect of reducing overfitting has been achieved.