Project Report - Learning to Rank using Linear Regression

Chaitanya Chembolu - 50248987 *Kautuk R Desai - 50247648*

Problem Statement

The project aims to rank the search engine webpages based on the features of each web page, the method is known as Learning to Rank (LeToR). We train a linear regression model using a closed-form solution and stochastic gradient decent method.

Approach

We load the LETOR data and synthetic data using numpy package. While importing the synthetic data we encountered an issue related to line endings. To fix this issue we changed the line endings to windows from macOS (default).

Now we have two datasets, we first partition those datasets into training data, validation data, testing data with proportions 80%, 10% 10% respectively.

In order to determine the value of hyper parameters, complexity of basis function and regularization constant lambda, we compute error for each pair of M (1 - 100) and lambda (0 - 1).

Once we get the optimal value of M and Lambda we cluster the training data into M clusters using k-means to get the centroids of each cluster. We then find the variance of each cluster which we use as the spread for each radial basis function. Using the optimal hyper parameters, we calculate the design matrix of the training input data.

Now, since we have the design matrix we compute the weights using two methods namely

1. Closed-form solution with regularization term
2. Stochastic gradient descent

a. Here we implemented the early stopping with different set of values for patience, epochs, and mini-batch size.

Finally, with the optimal hyper parameters we calculated the root mean squared error for training, validation and test data set using the weights obtained from the above mentioned methods. The results were computed for LETOR and Synthetic dataset.

Results & Observations

After computing the grid search on M and lambda values in the range 1 to 100 and 0 to 1 respectively the values with minimum errors for the two dataset we found to be

LETOR Data: M = 56; lambda = 0

Synthetic Data: M = 33; lambda = 0

Further, we plot the values of E_RMS for each set of M and lambda. Since the difference between the errors values is not much we normalize the values to be able to plot it properly. The plots are attached below

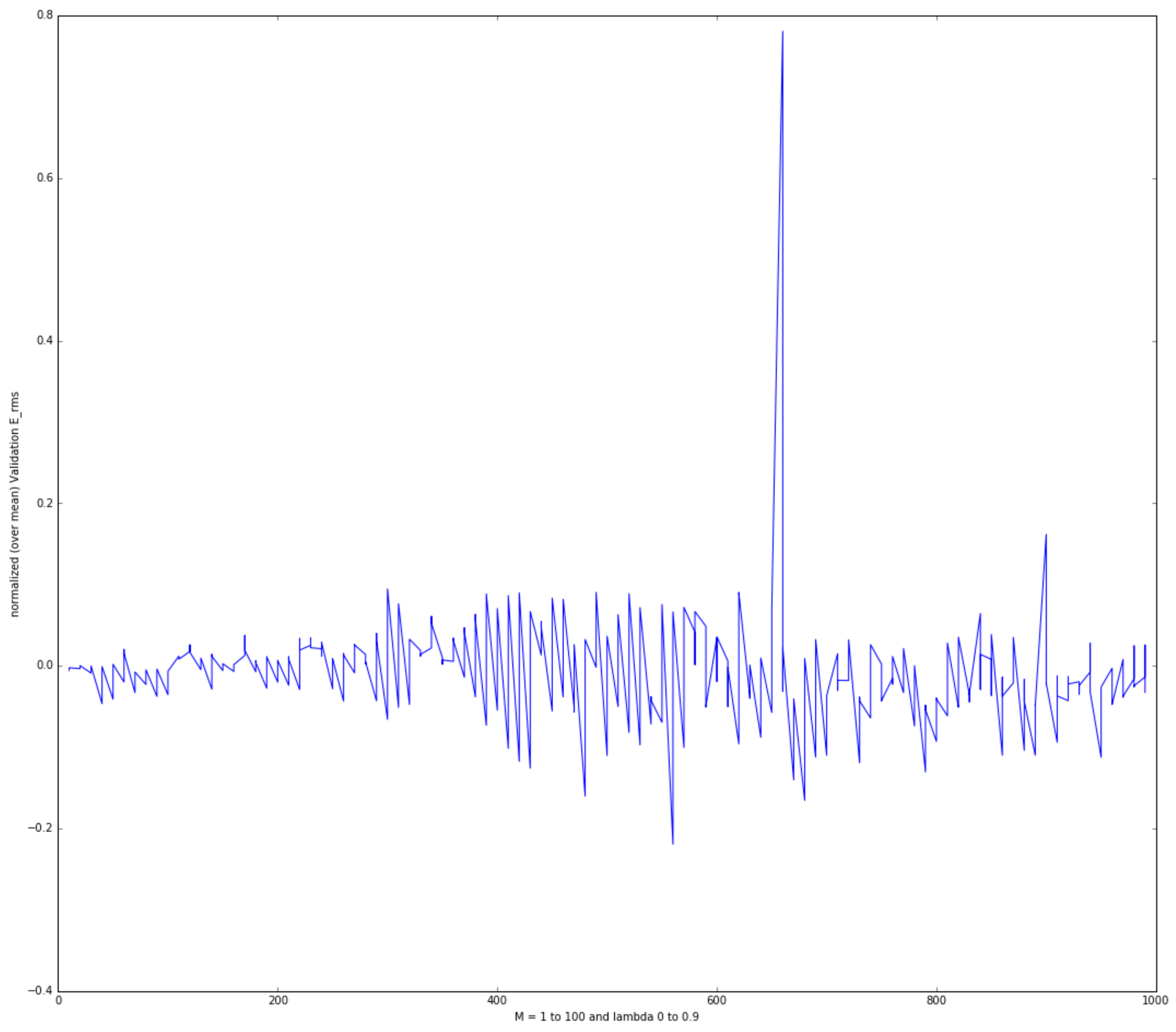

Fig1: LETOR data; Model complexity vs RMS error

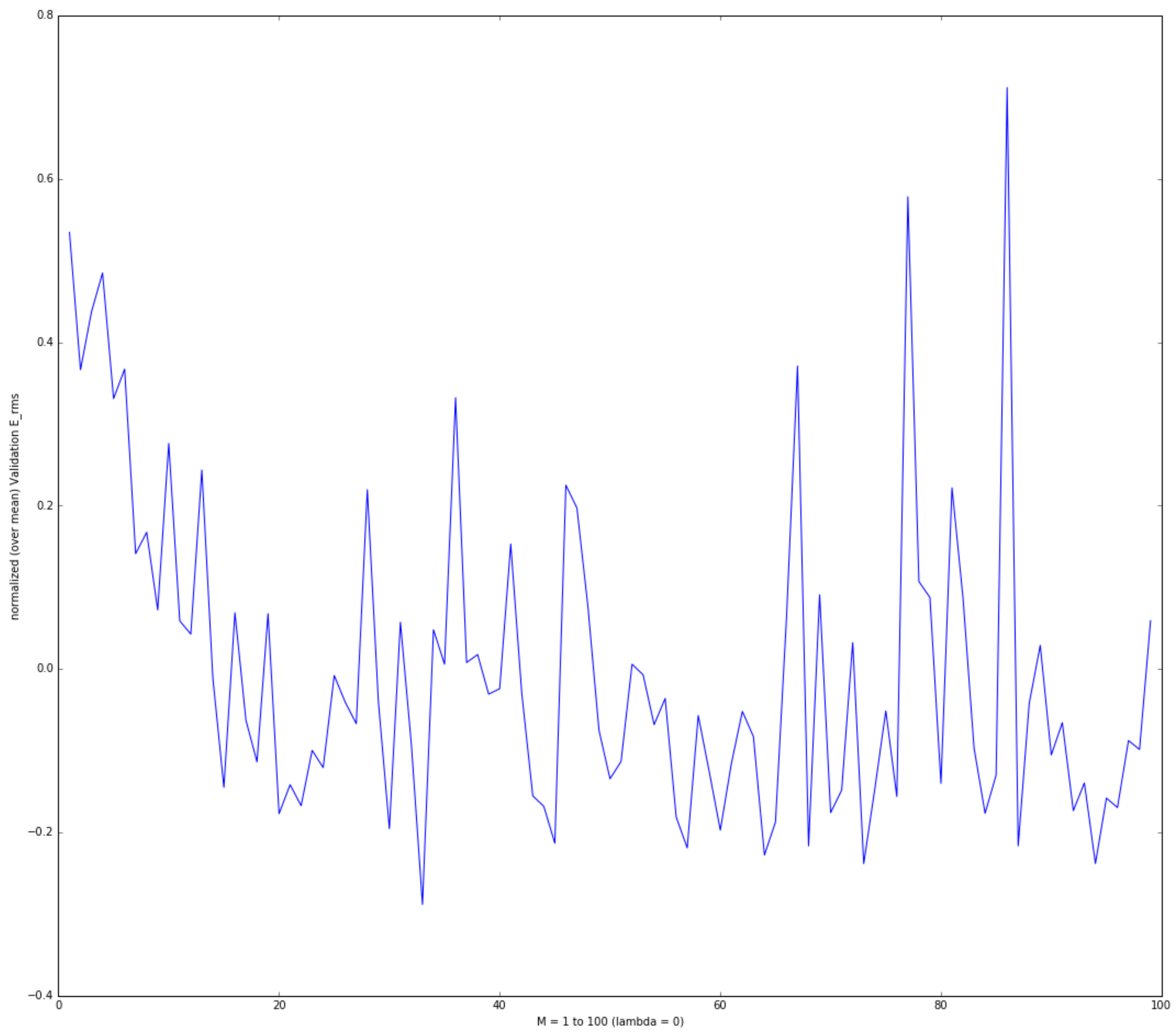

Fig2: Synthetic dataset: Model complexity vs RMS error

From the above plots, we can clearly see that dip indicates the optimal value of hyper-parameter with minimum RMS error.

After trying different set of values for patience, epochs, learning rates and mini batch sizes for stochastic gradient algorithm. The algorithm converged quickly and efficiently with the following set of values

1. Learning rate = 1
2. Epochs = 100
3. Mini batch size = 500
4. Patience = 25

	LETOR Dataset		**Synthetic Dataset**	
Errors \ Methods	Closed-form solution	Stochastic Gradient Descent	Closed-form solution	Stochastic Gradient Descent
Training Error RMS	0.562956607412	0.564228185967	0.70738994533	0.711020625616
Validation Error RMS	0.552962970599	0.553936136576	0.723643006016	0.719707182198
Test Error RMS	0.637878610991	0.63570204165	0.708766051056	0.711639849261

Note:

Running main.py file will print the weights along with the RMS error for partitioned data for closed-form and SGD method and for each dataset.

References:

1. https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)#Grid_search
2. Hyper-parameter Optimization Machines: **Published in:** Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on
3. lecture slides