Ashwin Vijayakumar(50249042)
Srivatsa Hegde(50248870)

PROJECT – 2

Learning to Rank using Linear Regression

The following is the set of steps we employed to train a regression model to predict the the appropriate response for two different data sets using a linear regression model created using 'M' gaussian basis functions , where M was determined using 'K' Means clustering .

Handling the input data:

The first step was to import the data files:

a. Synthetic data set (input.csv, output.csv)

b. LETOR data set (Querylevelnorm_X.csv, Querylevelnorm_t.csv)

Next we partition the LETOR data set into training, test and validation sets (in the ratio 80:10:10) by random selection.

	0	1	2	3	4	5	6	7	8	9	...	36	37	38	39	40	41	42	43	44	45
66286	0.075397	0.000000	0.000000	1.0	0.077304	0	0	0	0	0	...	0.999990	0.984020	1.000000	0.975220	0.000000	0.000000	0.100000	0.71429	0.739390	0
25594	0.004505	0.333330	0.500000	0.5	0.006757	0	0	0	0	0	...	0.931530	0.669800	0.548260	0.549680	0.032788	0.192980	0.031802	0.40000	0.406250	0
22677	0.010013	0.000000	0.000000	0.0	0.009975	0	0	0	0	0	...	0.569520	0.142770	0.130570	0.000000	0.000000	0.000000	0.083333	0.71429	1.000000	0
2001	0.051383	0.000000	0.000000	0.0	0.051181	0	0	0	0	0	...	0.828200	0.469870	0.529170	0.460420	0.015553	0.130430	0.000000	0.83333	0.525000	0
36782	0.523030	0.000000	0.000000	0.0	0.522730	0	0	0	0	0	...	0.428050	0.041517	0.074737	0.036004	0.004962	0.066667	0.000000	1.00000	0.690480	0

Train_input data sample (55698 rows × 46 columns) for the LeToR dataset

```
41884    0
20101    0
61663    0
52560    0
59559    0
34305    0
13924    0
25991    0
17309    0
```

Train_output data sample (55698 × 1 columns) for the LeToR dataset

Similarly,
we find the test and validation data sets: (each further divided into input and output matrices)
test_input (6963 rows × 46 columns**)**
test_output (6963 rows × 1 columns)
validation_input (6962 rows × 46 columns)
validation_output (6962 rows×1 columns)

Synthetic data set :

train_input (1600 rows × 10 columns**)**
train_output (1600rows × 1 columns)
test_input (2000 rows × 10 columns**)**
test_output (2000 rows × 1 columns)
validation_input (2000 rows × 10 columns)
validation_output (2000 rows×1 columns)

Calculating Center, Spread and Design Matrix :

In order to find the closed form solution, we first convert the training data which is an N*D (D=46, the number of features) matrix into an N*M design matrix using M Gaussian basis functions. For the selection of M, we use the K-means algorithm with varying values of K (K is a list [3,4,5,7,10] in our example) and calculate the error on the validation set for the best 'K' . We then find the center (dimension : M * 1 * D) and spread matrices (M * D * D) that are used to calculate the Design matrix . The M is derived directly from the K centres and the spread matrix is calculated from diagonalizing a matrix using the variances of each columns multiplied by a constant 'k'.

$$\mathbf{\Phi} = \begin{bmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix}$$

Calculating Closed Form Solution:

Using the formula “wML = (λ + Φ(T) * Φ)(-1) Φ(T) * t ” , we get the closed form solution for the training weights . This computation of this solution may involve dealing with singular matrices , and hence has the potential to lead to non definitive answers . To avoid this and the corresponding difficulty in computing the solution , we use an iterative approach for finding the optimum weights of our regression model .

Stochastic Gradient Descent Solution:

Stochastic gradient descent is an iterative method for minimizing the objective function . The general formula is :

$w(\tau+1) = w(\tau) + \Delta w(\tau)$ (This shows how the weights are modified in each sweep of the function)

and

$\nabla E = \nabla ED + \lambda \nabla EW$ (Where ED is the error on the data and EW is the weight decay. Lambda is the regularization coefficient that determines whether our model overfits or not. We select appropriate lambda based on the validation errors during hyper-parameter training.

Early Stopping :

We use early stopping (a kind of regularization) to avoid overfitting. Validation can be used to detect when overfitting starts during supervised training of a neural network; training is then stopped before convergence to avoid the overfitting (early stopping). The exact criterion used for validation-based early stopping, however, is usually chosen in an ad-hoc fashion or training is stopped interactively. This also helps to minimize the training. We can use early stopping method while training the weights, by testing the validation error for every 'n' number of epochs and stopping the update of the weights when the patience is reached. The patience or the tolerance is reached every time the validation error is more than the previously calculated validation error.

Hyper parameter tuning (Against the validation set):

We use grid search to tune the hyper-parameters by exhaustively using all the parameters (Learning rate, K value , lambda) that are predefined as a populated list . Then we calculate the RMS error on the validation set and store them in a list that contains all the possible errors.

Learning_Rate = [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5]
Choosing a very small value of Learning rate can increase the time for convergence and increases the chances of being stuck at the local minima, while choosing a very high value can cause a NaN (Not a Number) error to occur during training.

K_Vector = [2, 3, 4, 5, 7, 10]
We use K - “means” to partition n observations into k clusters. We generate K centroids (called means), k number of clusters are formed by associating observations with the closest centroid. The centroid of the clusters is the new mean. This is repeated iteratively until convergence. (We use the sk-learn library to implement K means in python) .
Choosing the K value decides how the clustering happens. It’s essential to choose the right K value for an optimal clustering. The centers and the spread are affected by the 'K' value we choose for clustering, hence can be optimized by a simple grid search for the best value. The number 'K' here is also the number of basis functions 'M' that we use in our model.

lambda_vector = [0.001, 0.01, 0.1, 1, 2]

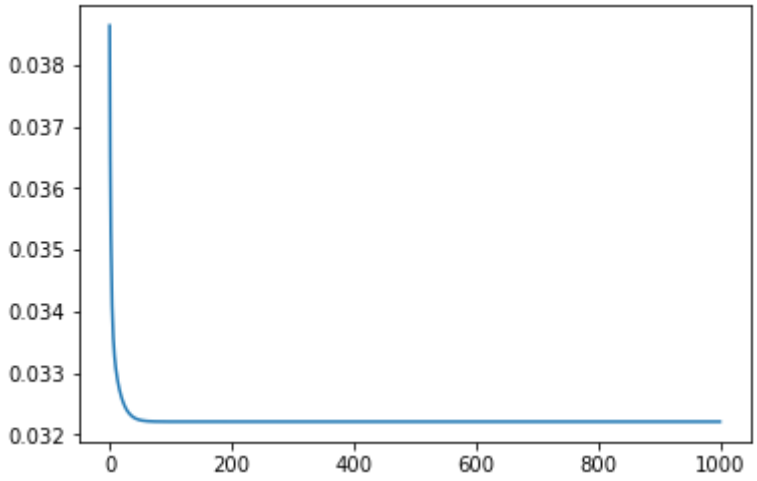
Choosing the 'lambda' value allows us to ensure we do not over fit the data by reducing the variance but with a trade-off for variance. The lower we set the value for lambda, the higher are the chances of the model to over-fit.

Observations while tuning hyper parameters (Performance of the training model):

For the LeToR dataset , we can see from the below graphs (a) and (b) that as the learning rate is increased (from 0.03 to 0.3) the convergence happens faster

We can see that the Gradient Descent Solution approximately matches the closed form solution, we also find that the Stochastic Gradient Descent is more computationally efficient compared to the closed form solution.

a)

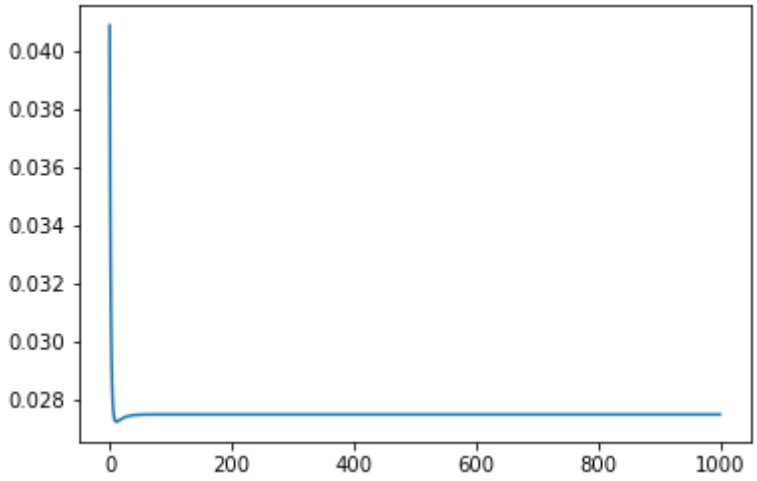


Stochastic Gradient Descent for learning rate = 0.03, lambda = 0.4, epochs = 1000, batch size = 256, K = 3

Closed form	[0.35783817 -0.52926045 0.37790227 0.02994983]
Gradient descent solution	[0.34241062 -0.48521548 0.35043797 0.04887694]

(Observe that the gradient descent solution, being an approximation, is not exactly equal to the closed form solution, but by using an iterative model and playing around with its parameters , we can achieve a good enough solutions that avoids the pitfalls involved in the calculation of a closed form solution . As the latter is often more difficult to do)

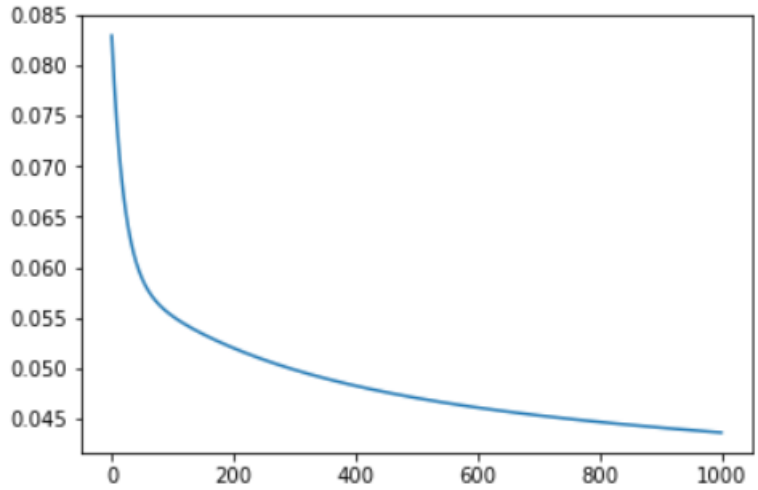
b)



Stochastic Gradient Descent for learning rate = 0.3 , lambda = 0.4 , epochs = 1000 , batch size = 256 K = 3 .
(note: Here we see that upon increasing the learning rate from 0.03 to 0.3, we achieve the convergence in lesser number of epochs and lesser time)

Closed form [0.35875985 -0.53953779 0.38704915 0.02915166]
Gradient descent solution [0.34511087 -0.49489974 0.3579862 0.04987781]

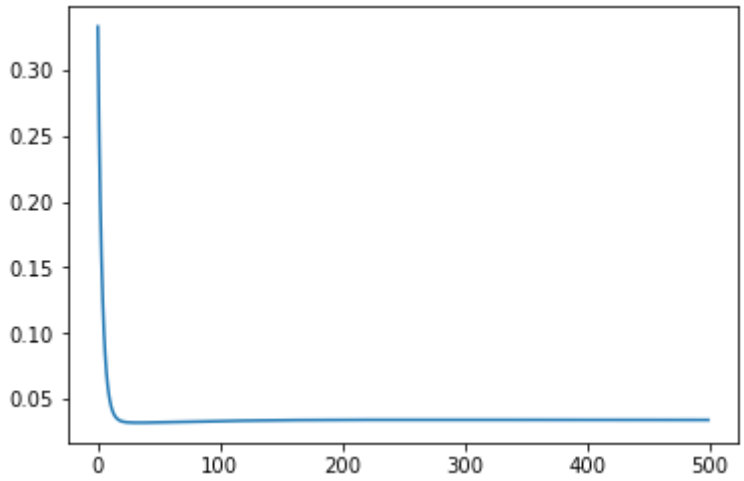
c)



`Closed form solution` of the weights [0.29251349 0.20809477 -0.06390619 -0.49387613 0.07285252 0.11682488
0.15528015 0.10423775]
`Gradient decent solution` of the weights [0.18926757 0.08999812 -0.07948189 -0.18318464 0.10535619 0.15005803
0.06253013 0.23856348]

Stochastic Gradient Descent for learning rate = 0.001 , lambda = 0.01 , epochs = 1000 , batch size = 256 , K = 7 .

d)



Stochastic Gradient Descent for learning rate = 0.0007 , lambda = 0.9 , epochs = 1000 , batch size = 256 , K = 5 .

Closed form

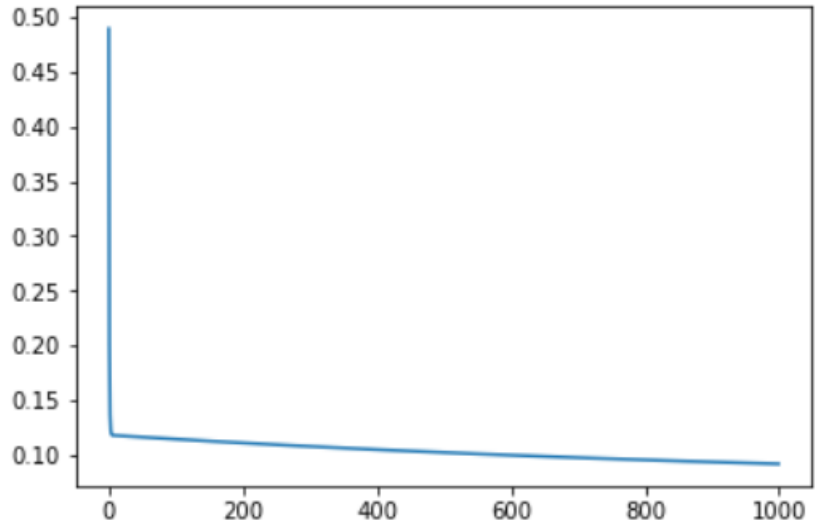
[0.34751965 0.02297788 0.36298298 -0.49344646 -0.10990535 0.13715676]

Gradient descent solution

[0.30478001 0.07990635 0.24876429 -0.35884032 -0.02681095 0.10358377]

The above values are for the LeToR dataset .

The synthetic data set also has a similar behaviour . The following is the graph for `K = 2 lambda= 0.01 Learning Rate = 0.01`



Best set of hyperparameters (LeToR): K =7, Learning rate = 0.0001 , lambda = .01

Best set of hyperparameters (Synthetic): 2

Ouput Console :
LeToR:

```
The best hyperparameters are : K =  7 lambda=  0.01  Learning Rate =  0.0001
Closed form solution of the weights [ 0.29251349  0.20809477 -0.06390619 -0.
49387613  0.07285252  0.11682488
  0.15528015  0.10423775]
Gradient decent solution of the weights [ 0.18926757  0.08999812 -0.07948189
-0.18318464  0.10535619  0.15005803
  0.06253013  0.23856348]
The root mean square test error is   0.578179242256
```

Synthetic:

```
The best hyperparameters are : K =  3 lambda=  2  Learning Rate =  0.001
Closed form solution of the weights [-4.28637567  5.73608854 -1.6640697   2.
29891987]
Gradient decent solution of the weights [ 0.03016315  0.71050351  0.07605661
0.35544574]
The root mean square test error is   0.769939672755
```

Calculating the Test Error:

After splitting the data into train, validation and test, and after selecting and tuning our hyper-parameters, we try to find the test error by applying our model to the test set and calculating the test error.

The RMS error formula is given by:

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N_{\mathrm{V}}}$$

The regression error formula is given by:

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\top}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

The test error for the LeToR dataset we got was: 0.5676
The test error for the Synthetic dataset we got was: 0.769939672755

Interpretation of results :

LeTOR

The LeTOR dataset is a machine learning dataset, in which queries and urls are represented by IDs. The dataset consists of feature vectors extracted from query-url pairs along with relevance judgment labels:

0(not relevant), 1(relevant), 2(highly relevant). In the data files, each row corresponds to a query-url pair. The first column is relevance label of the pair, the second column is query id, and the following columns are features. The larger value the relevance label has, the more relevant the query-url pair is. A query-url pair is represented by a 46-dimensional feature vector.

Our model , when used on the test set , can predict whether the input belongs to irrelevant , relevant or highly relevant with a root mean squared error of 0.578 .

Synthetic data-set

The synthetic data is generated using some sort of a mathematical formula as given below:

y = f(x) + ε

Where f(x) - deterministic function

ε – Random noise, to scatter the

Our model , when used on the test set , can predict the output with a root mean squared error of 0.731 .