CSE574 Assignment 1.2 Report

LeToR: Learning to Rank

Tenzin Norden
UB person# 50096989
October 10, 2018

1 Introduction

In this project, we use two different supervised machine learning approaches to solve Learning to Rank (LeToR) problem which arises in Information Retrieval. For example, we use a lot of search engines like google, where every search displays the relevant search results which are ranked based on their relevance. The ranking and the relevancy of the search result depends on a lot of parameters called as features. Our objective is to formulate LeToR problem as linear regression and map an input vector **x** to a real-valued scalar target y(**x**,**w**). Here a linear regression model is trained on LeToR datasets using a 'Closed-form solution' and Stochastic gradient descent (SGD) solution.

2 Dataset

Our objective is to implement linear regression on a LeToR dataset. We use a benchmark dataset – QueryLevelNorm version of LeTOR 4.0 MQ2007 supervised ranking provided by Microsoft Research Asia.

The input vector of the LeToR dataset is a query-URL pair and the target scalar are value assigned about how well the URL corresponds to the query. The dataset consists of a total of 69623 query-document pairs(rows) each having 46-dimensional feature vectors. Following is a sample field of the input dataset:

```
2 qid:10032 1:0.056537 2:0.000000 3:0.666667 4:1.000000 5:0.067138 ... 45:0.000000 46:0.076923
        #docid = GX029-35-5894638 inc = 0.0119881192468859 prob = 0.139842
0 qid:10032 1:0.279152 2:0.000000 3:0.000000 4:0.000000 5:0.279152 ... 45:0.250000 46:1.000000
        #docid = GX030-77-6315042 inc = 1 prob = 0.341364
0 qid:10032 1:0.130742 2:0.000000 3:0.333333 4:0.000000 5:0.134276 ... 45:0.750000 46:1.000000
        #docid = GX140-98-13566007 inc = 1 prob = 0.0701303
1 qid:10032 1:0.593640 2:1.000000 3:0.000000 4:0.000000 5:0.600707 ... 45:0.500000 46:0.000000
        #docid = GX256-43-0740276 inc = 0.0136292023050293 prob = 0.400738
```

The first column contains a relevance label of the row with discrete values 0, 1, or 2. For instance, larger relevance label corresponds to a better match between query and document. The 46-dimensional feature input vector **x** for linear regression are real numbers that are normalized to be in the range of 0 to 1. Additional fields in the dataset such as the query id (qid), document id (docid), inc, and prob are ignored in this project.

The dataset.csv file is parsed and the irrelevant fields are removed. The first column feature label is extracted and stored as a target Numpy vector **t**. Out of the 46 feature columns, 5 columns with values as 0 are removed. The remaining 41 feature columns are processed into Numpy matrix input vector **x**. The data is then partitioned as 80% to a training set, 10% to a validation set and 10% to a testing set. The three datasets do not overlap. The training set is used to learn the weights or the parameters. The validation set is used to validate the learned weights and parameters. The test set is used at last to test the accuracy of the model with the learned weights.

3 Model Description

In this project, we model the linear regression function y(**x**,**w**) using M basis functions and has the form:

$$y(\mathrm{x}, \mathrm{w}) = \mathrm{w}^{\mathrm{T}}\phi(\mathrm{x}) = w_0\phi_0 + w_1\phi_1 + \cdots w_{M-1}\phi_{M-1}$$

where $\mathrm{w} = (w_0, w_1, \ldots., w_{M-1})$ is weight vector to be learnt from the training samples and $\phi = (\phi_0, \phi_1, .., \phi_{M-1})^{\mathrm{T}}$ is a vector of M basis functions. The M basis functions are non-linear functions of input variables and brings a non-linearity to the model. Therefore, M can represent the complexity of the model. We assume $\phi_0(\mathrm{x}) = 1$ for all inputs to introduce w_0 as the bias term in the model.

Here we chose the basis function as Gaussian radial basis function

$$\phi_j = \exp\big((x - \mu_j)^T {\Sigma_j}^{-1} (x - \mu_j)\big)$$

where μ_j is the center of the basis function and Σ_j decides how broadly the basis functions are spread. Each basis function turns the input vector into a scalar value.

1. In this project, k-means clustering is used to choose basis functions wherein the observations are clustered into M clusters, and their centroids are centers for Gaussian radial basis functions μ_j.

$$\begin{bmatrix} \mu_{11} & \mu_{12} \cdots & \mu_{1,N} \\ \mu_{21} & \mu_{22} \cdots & \mu_{2,N} \\ \vdots & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \mu_{M1} & \mu_{M2} & \mu_{MN} \end{bmatrix}$$

The center of the Gaussian radial basis function μ_j is formed into a vector of MxN dimensions where N is 41 and M is a tunable hyperparameter number of basis function and it also determines the complexity of the model.

2. A uniform spread for all basis functions $\Sigma_j = \Sigma$ is considered in this project and it is constrained to a diagonal 41x41 dimension matrix where each diagonal element represents the variance σ_D^2 from each 41 feature columns.

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & \cdot \\ \vdots & \sigma_2^2 \cdots & \cdot \\ \vdots & & \\ \cdot & \cdots & \sigma_D^2 \end{bmatrix}$$

3. Each basis function then turns the input vectors into a scalar value. For example, the terms in the exponent of Gaussian radial basis function undergo matrix multiplications for every cluster 1 to M resulting into a scalar value as described below.

$$(x_i^1 - \mu_1)^T {\Sigma_j}^{-1} (x_i^1 - \mu_1) = \begin{bmatrix} (x_1^j - \mu_1) & (x_2^j - \mu_2) & \ldots\ldots & (x_i^j - \mu_j) \end{bmatrix} \begin{bmatrix} \sigma_1^2 & \cdots & \cdot \\ \vdots & \sigma_2^2 \cdots & \cdot \\ \vdots & & \\ \cdot & \cdots & \sigma_D^2 \end{bmatrix}^{-1} \begin{bmatrix} (x_1^j - \mu_0) \\ (x_2^j - \mu_1) \\ \\ (x_i^j - \mu_j) \end{bmatrix}$$

where for $\mathbf{x}_i^j$ is the value of feature **i** in $\mathbf{j}^{th}$ training example. A matrix multiplication above is the form of **[1x41] [41x41] [41x1]** resulting in a scalar output [1x1]. Each Gaussian radial basis function is a scalar value depending on the parameters such as its centers $\boldsymbol{\mu_j}$, variance $\boldsymbol{\sigma_D^2}$ and number of basis function **M** (model complexity).

4. After calculating the basis function $\phi(\mathbf{x}_n)$, it is then multiplied with the weights to perform linear regression. The function **y(x, w)** is still linear in **w** and $\mathbf{x}_n$ but non-linear with respect to $\phi(\mathbf{x}_n)$.

The output is assumed to have a probabilistic normal distribution with its mean as y(**x**,**w**) and noise β which is assumed constant. The likelihood function has the form:

$$p(\mathbf{t}|\mathbf{x}, \mathrm{w}, \beta) = \prod_{n=1}^{N} N(t_n|\mathbf{w}^T \phi(\mathbf{x}_n), \beta^{-1}) = \exp\left(-\frac{\sum_{i=1}^{N}[t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)]^2}{2\beta^2}\right)$$

where $\mathbf{x} = \{x_1, \ldots, x_n\}$ are the input samples and $\mathbf{t} = \{t_1, \ldots, t_n\}$ are target values. **Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error.**

$$E_D = \frac{1}{2}\sum_{i=1}^{N}[t_n - \mathbf{w}^{\mathrm{T}}\phi(\mathbf{x}_n)]^2$$

5. In order to smoothen out the curve and prevent overfitting, a regularization term or a penalty term is added to the error function as follows.

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_{\mathrm{w}}(\mathbf{w})$$

where λ is a tunable hyper parameter which governs the importance of the regularization term and the weight decay regularizer is

$$E_{\mathrm{w}}(\mathbf{w}) = \frac{1}{2}\mathbf{w}^{\mathbf{T}}\mathbf{w}$$

The goal in this project is to minimize the error function while containing overfitting at the same time using regularization term. To find the proper weight $\mathbf{w}^*$ which minimizes the weight, we use two different linear regression solutions.

3.1 Closed Form Solution

1. After setting the model complexity M, and clustering, the model hyperparameters such as variance and mean are computed as shown above. Every Gaussian radial function is computed for each cluster and for every input samples, the Design matrix **Φ** is computed which is of dimension NxM where M is the model complexity and N is number of input sample.

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_o(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \phi_o(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{bmatrix}$$

2. After computing the design matrix, we can calculate the closed form solution of maximum likelihood to the linear regression using Moore-Penrose pseudo-inverse of the

matrix $\mathbf{\Phi}$ as

$$\mathbf{w}_{ML} = (\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$

where $\mathbf{w}_{ML}$ is the weight function and $\mathbf{t}$ represent the target vector $\mathbf{t} = \{t_1, t_2, \dots t_N\}$.
A regularization term is added in order to contain overfitting and the regularized weights $\mathbf{w}^*$ is calculated as

$$\mathbf{w}^* = (\lambda\mathbf{I} + \mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$

where I is an identity matrix.

3. We can now calculate minimized sum-of-squared errors by using the calculated regularized $\mathbf{w}^*$ and evaluate the root mean square error E_{RMS} given by

$$\boldsymbol{E_{RMS}} = \sqrt{2E(\mathbf{w}^*)/N_v}$$

where N_v is the size of the dataset. We repeat the same process by changing hyperparameters such as the regularization term λ and model complexity or number or number of Gaussian basis function M for the validation and testing dataset. The $\boldsymbol{E_{RMS}}$ is recorded after tuning the hyperparameters and plot in the experiment section of the report.

3.2 Stochastic Gradient Descent (SGD)

In stochastic gradient descent approach the weights are randomly initialized $\mathbf{w}^{(0)}$ which is then updated after minimizing the first order differentiation of the error function ∇E. The updated weight is done as follows

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau)} - \eta^{(\tau)}\nabla E$$

where $\eta^{(\tau)}$ is the learning rate, which decides the step size of the gradient descent. The weights are updated over many iterations until the global minimum of the error function is reached. The gradient of the error function is given by

$$\nabla E_D = -(\mathbf{t}_n - \mathbf{w}^{(\tau)\mathrm{T}}\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)$$

And a regularization term is added to contain overfitting as follows.

$$\nabla E = \nabla E_D + \lambda\nabla E_W$$

where $\nabla E_W = \mathbf{w}^{(\tau)}$. $\boldsymbol{E_{RMS}}$ is again calculated after minimizing the error as

$$\boldsymbol{E_{RMS}} = \sqrt{2E(\mathbf{w}^*)/N_v}$$

We again tune the hyperparameters such as M, learning rate $\eta^{(\tau)}$, and λ and try to find the optimum value which give the lowest $\boldsymbol{E_{RMS}}$ value.

4 Experiment

4.1 E_{RMS} vs λ

In the first experiment, to obtain the lowest E_{RMS} value, I tuned the regularization coefficient 'λ' in Closed Form solution (C_lambda) and (La) in Stochastic Gradient Descent solution independently with λ = 0.01, 0.05, 0.1, 0.5, 1, 5, and 10. Other hyperparameters such as number of basis function M and learning rate are kept constant to find the most effective of regularization term for Closed From solution and Stochastic Gradient Descent.

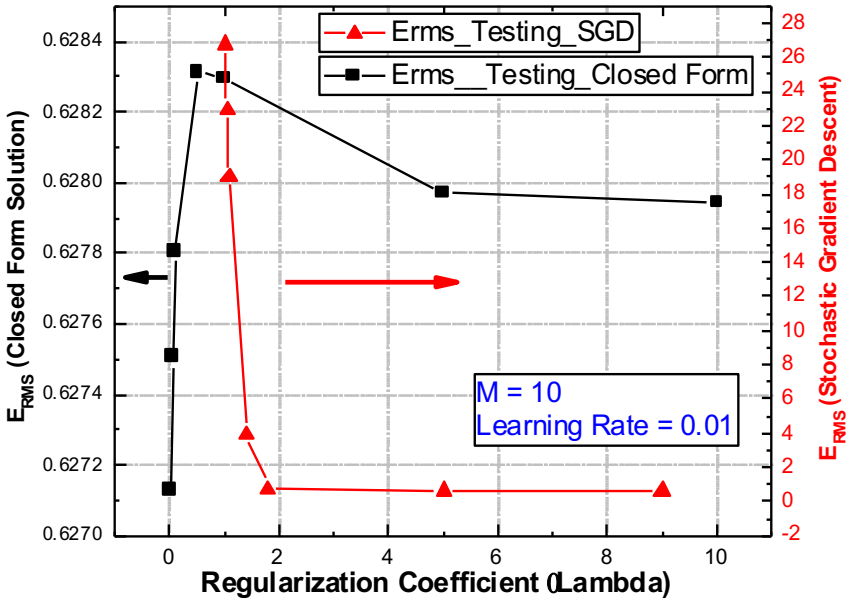

Figure 1. E_{RMS} **vs** λ. E_{RMS} for Closed From Solution is scaled on the left vertical axis (black squares) and E_{RMS} **from Stochastic Gradient Descent is scaled on the right vertical axis (red triangles).**

In Figure 1, I plot the testing dataset E_{RMS} as a function of λ for Closed Form Solution and Stochastic Descent together to compare the dependency of E_{RMS} on regularization between these two different approaches.

It is observed that in closed form solution, E_{RMS} does not change as much as compared to SGD solution. In closed form solution approach, even though the change is very small E_{RMS} increases drastically with λ and then it decreases after λ = 0.5 and then saturates for higher values of λ. The lowest E_{RMS} is 0.62713 for $\lambda = 0.01$. It appears that smaller λ contains overfitting better for the current set up of fixed parameters.

On the other hand, in Stochastic Gradient Descent approach, learning rate is kept fixed at 0.01 and 400 iterations were carried out. E_{RMS} of the testing dataset decreases exponentially with increasing λ and then saturates for λ = 1 and above. The lowest E_{RMS} found is 0.63546 for $\lambda = 5$ for 400 iterations.

4.2 E_{RMS} vs M

In Figure 2, I plot E_{RMS} testing as a function of number of basis function or model complexity '**M**' for Closed Form solution (blue) and Stochastic Gradient Descent solution (red). For the Closed form solution, I kept $\lambda = 0.03$ fixed and recorded the corresponding E_{RMS} for the testing dataset. I observed that with increasing **M**, the performance of the model becomes better as E_{RMS} reduces as can be seen in Figure 2. E_{RMS} then saturates for large values of M and does not decrease as much. For **M = 100**, E_{RMS} = 0.61861 for the set up.

Similarly, in SGD approach, E_{RMS} also decreases with increasing **M** as compared to Closed Form solution. It also means the performance of the Stochastic Gradient Descent improve with the complexity of the model. The lowest E_{RMS} value is obtained for **M = 100 is** 0.63024.

It is clear that by increasing **M**, we are also automatically increasing the value of other hyperparameters such as number of clusters and variance. In doing so, we are maximizing the likelihood function, and hence, minimizing E_{RMS}. On the other hand, for small values of M, the value of variance is large and will result in smaller value of likelihood function which is equivalent to larger E_{RMS}.

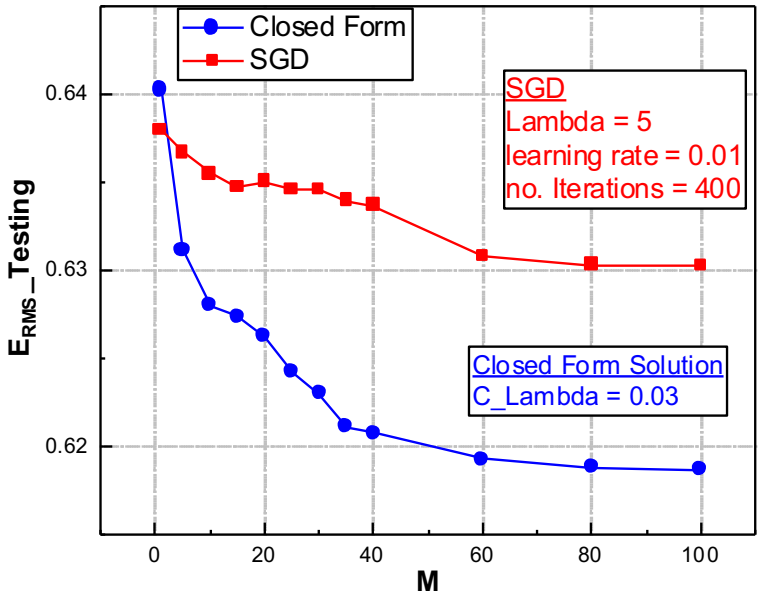

Figure 2. E_{RMS} for testing data plotted as function of M. E_{RMS} for Closed Form solution is shown with blue circles and E_{RMS} for Stochastic Gradient Descent is shown as red squares.

4.3 E_{RMS} vs Learning Rate: Stochastic Gradient Descent Solution

Here E_{RMS} is plotted as a function of learning rate 'η' tuned from 0.01 to 0.1 as shown in figure 3. In Stochastic Gradient Descent Solution, learning rate decides the step size of the gradient descent which makes it a very important parameter. Finding proper learning rate is of utmost importance in in Gradient descent approaches because small learning rate may cost lot of computing power and time while large learning rate may over shoot the global minimum and the solution may not converge.

In figure 3, I plot $\boldsymbol{E_{RMS}}$ of testing, validation and training sets to compare them. $\boldsymbol{E_{RMS}}$ seems to remain relatively constant with small changes for the fixed parameter M = 15 and λ = 5.

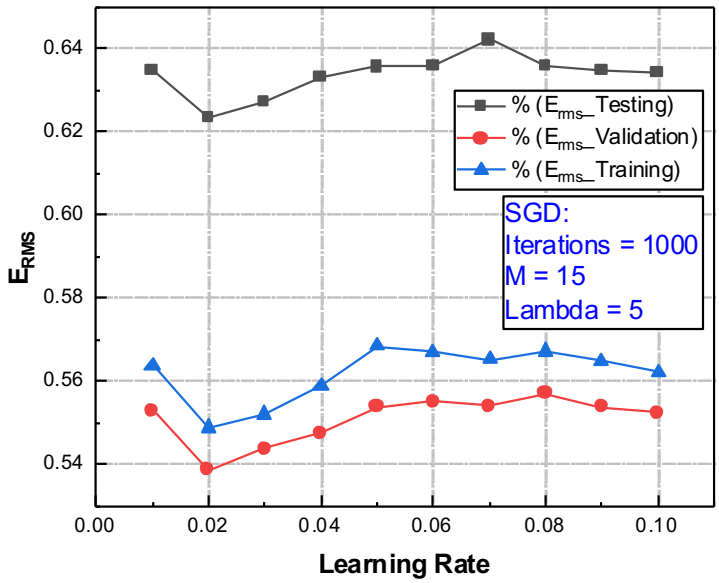

Figure 3. $\boldsymbol{E_{RMS}}$ Vs Learning rate for Stochastic Gradient Decent solution. $\boldsymbol{E_{RMS}}$ for testing (black square), validation (red circles) and training (blue triangles) are plotted as function of learning rate.

4. 4 $\boldsymbol{E_{RMS}}$ vs iteration: Stochastic Gradient Descent Solution

In figure 4, $\boldsymbol{E_{RMS}}$ training and testing is plotted as a function of iterations to check the minimization of error and at which iteration $\boldsymbol{E_{RMS}}$ becomes stagnant. In this experiment, the learning rate is not kept at a constant value but treated as automatic self-adjustable parameter with every iteration. Learning rate = $0.1/\sqrt{5 * i}$ where **i** number of iterations set at 1000. I kept M = 15 and λ = 5 as fixed value. We can visually see that the $\boldsymbol{E_{RMS}}$ is decreasing with number of iterations.

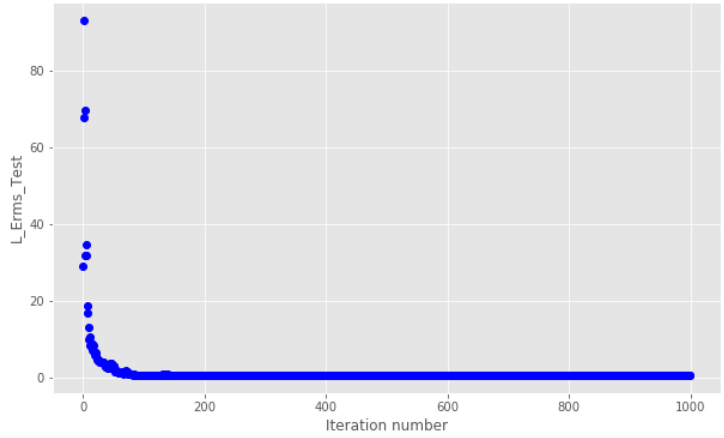

Figure 4. $\boldsymbol{E_{RMS}}$ is plotted as function of iteration by changing Learning rate automatically.

5. Conclusion

In this project, a benchmark dataset from Microsoft to solve Learning to Rank (LeToR) problem which arises in information retrieval using Closed Form solution and Stochastic Descent Solution. LeToR problem is formulated as linear regression and map an input vector **x** to a real-valued scalar target y(**x**,**w**). A Gaussian radial basis function is used in this project to give a non-linearity to the model.

In closed form approach, proper weight to minimize $\boldsymbol{E_{RMS}}$ is found by solving Moore-Penrose pseudo-inverse added with regularization term to contain overfitting. On the other hand, in Stochastic Gradient Descent, random weights are chosen initially which then updated after every iteration of linear regression until optimal performance of the model is reached. $\boldsymbol{E_{RMS}}$ is minimized by changing various combinations of hyperparameters.