Project 1.2: Learning to Rate

Nishi Mehta
Graduate student, Computer Science
#Person 50291671
nishimeh@buffalo.edu

Abstract

This project aims to solve the Learning to Rank problem which arises in Information Retrieval through linear regression. The two methods used for training the linear regression model are closed form solution and stochastic gradient descent. Through this project, we try to find the optimal solution to this problem by comparing different values for hyperparameters on the linear regression model.

1 LeToR

Learning to Rank is an application of Machine Learning which uses supervised, semi-supervised or reinforcement learning in the construction of ranking models for Information Retrieval Systems. Ranking is a very crucial task in Information Retrieval problems. In this project, we use a supervised learning model. We train our model through the data and then predict values for other data.

There are various approaches to solve a LeToR problem. Some of the widely used approaches are mentioned below.

1.1 Pointwise approach

In this case, it is assumed that each query-document pair in the training data has a numerical or ordinal score. Then the learning-to-rank problem can be approximated by a regression problem — given a single query-document pair, predict its score.

1.2 Pairwise approach

In this case, the learning-to-rank problem is approximated by a classification problem, learning a binary classifier that can tell which document is better in a given pair of documents. The goal is to minimize the average number of inversions in ranking.

We use the pointwise approach and solve the problem using linear regression.

2 Dataset

For this problem we use the Microsoft LETOR 4.0 Dataset which is a dataset released by Microsoft Research Asia. It contains 8 datasets for four ranking settings derived from the two query sets and the Gov2 web page collection. We use the 'querylevelnorm' version of the dataset. The entire dataset contains 69623 query document pairs each having 46 features. Each row contains the query id, the document id, the rank and the value for the 46 features.

2.1 Data Preprocessing

In order to use this data for regression we need to preprocess the data. One row of the data in the text file looks like this:

```
2 qid:10032 1:0.056537 2:0.000000 3:0.666667 4:1.000000 ...  46:0.076923

    #docid = GX029-35-5894638 inc = 0.0119881192468859 prob = 0.139842
```

Here 2 is the rank of the data, 'qid' is the Query ID and it's value is 10032, Document ID is represented by 'docid'. Rest of the values represent the feature vector which has values for 46 features.

Firstly, we parse this data and generate two files. The first file contains the value for all the ranks that is, target data. The second file that is the data file, contains all the 46 features for all 69623 queries.

Secondly, we delete the features which have 0 variance values. There are five columns in the feature table which have 0 value for all the 69623 queries. We delete those columns from the data as all the values for this feature are 0. Also, when we will compute the inverse of the design matrix the diagonal value will be zero if we keep these features and we won't be able to compute it's inverse. Hence, we delete the zero-valued features.

Thirdly, we separate the data into three parts. The first part contains 80% of the data which is kept for training. The second part is 10% of the data and is used for validation. The remaining 10% is used for testing. We keep validation data in order to keep track of how our model is performing during training. When we tune the hyperparameters, we can check whether the model is overfitting or underfitting the data through validation accuracy and error.
Size of training data: 55699 data points
Size of validation data: 6962 data points
Size of testing data: 6962 data points

3 Closed form solution

The first approach that we use to solve the LeToR problem is the closed form solution.
The closed form solution has the form:

$$t = w^T \emptyset(x)$$

Here 't' represents the target vector which we need to obtain. 'w' represents the weights which satisfy the closed form solution and $\emptyset(x)$ is the value for one data point 'x' in the design matrix.

To obtain this we go through a number of steps:

3.1 Design Matrix

To compute the design matrix we take a radial basis function. The reason we use a design matrix and a basis function is because there is non-linearity in the data and without using a basis function it is not possible to obtain an optimal result.

We use the gaussian radial basis function which is defined as:

$$\emptyset_1(x_1) = e^{\frac{-(x_1-\mu_1)^2}{2\sigma^2}}$$

As our input data is in the form of vectors, we convert the above formula and use it as:

$$\emptyset_1(x_1) = e^{\frac{-(x_1-\mu_1)^T\Sigma^{-1}(x_1-\mu_1)}{2}}$$

$\emptyset$ is the design matrix and $\emptyset_1(x_1)$ is the value of the first basis function for x_1.
Here, x_1 is the feature vector for the 1st data-point and is of the dimension 1x41.

We calculate the centroids of the data and all the centroids are represented by μ. Here, μ_1 is the centroid for the first cluster of the data and is of the dimension 1x41.
Σ^{-1} is the covariance matrix which represents the variance of a particular feature.

Suppose, we have ten basis functions. Then we need to calculate 10 centroids for the data. Hence, we divide the data into 10 clusters. For generating these clusters, we use the simple k-means algorithm. We use k-means because it is a fast converging and simple algorithm.

The dimensions of the design matrix: the number of rows is equal to the number of data points and the number of columns is equal to the number of basis functions. In this case we use 10 basis functions.
Hence, the dimensions of the design matrix for
training data: (55699, 10)
validation data: (6962, 10)
testing data: (6962, 10)

The design matrix can be represented by:

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

As we can see, we calculate various basis function outputs for each data point.

The covariance matrix is a 41x41 dimension matrix as we have 41 features if we eliminate features with 0 variance. The matrix is of the form:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_D^2 \end{pmatrix}$$

Here, $\sigma_1{}^2$ signifies the variance of the 1st feature. We keep all the other values in the matrix as 0. This is because we don't need the variances between different feature vectors. Also, a bigger variance acts as a general classifier whereas a smaller variance acts as a local classifier. In order to make it a general classifier we do a dot product of the covariance matrix with a random large value.

Hence the exponential term in the calculation of the $\emptyset(x)$ value becomes a scalar value and we get $\emptyset(x)$ as a scalar.

3.2 Computation of weights

After computing the design matrix, next we compute the weight matrix. The number of weights is the same as the number of basis functions used. In this case we suppose that we have used 10 basis functions, then the dimension of the weight matrix is 10x1.
We compute the weights through this formula:

$$w = (\emptyset^T \emptyset)^{-1} \emptyset^T t$$

This quantity is known as the Moore-Penrose pseudo inverse of the design matrix. We use this pseudo inverse because the design matrix is not a square matrix and we cannot perform inverse on it.

Here 'w' is the weight vector, '$\emptyset$' is the design matrix and 't' = {t_1, t_2 …. t_n} are the target output values of the training data.

To prevent overfitting of the data, we add least-square regularization to the weights. This regularization helps the model to unlearn some of the data.

Finally, the weight matrix is computed by,

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

Where 'I' is the identity matrix and λ is the regularization value.

3.3 Actual output values and E_{RMS}

After computing the design matrix and the weights, we compute the actual target values through this equation.

$$t = w^T \emptyset(x)$$

Next, we calculate the root mean square error to judge how well the model is performing.

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

Where,

$$E(w^*) = ½\,(t_n - \widehat{t_n})^2$$

And

$$\widehat{t_n} = w^T \emptyset(x)$$

Hence,

$$E_{RMS} = \sqrt{(t_n - w^T \emptyset(x))^2/N_v}$$

We calculate E_{RMS} for the training, testing and the validation data.

4 Stochastic Gradient Descent

Stochastic gradient descent (often shortened to SGD), also known as incremental gradient descent, is an iterative method for optimizing an objective function.

In this method, we update the weights through multiple iterations to decrease the error of the model. We update the weights based on a parameter known as learning rate.

Hence, this can be represented through the below equation,

$$w_{i+1} = w_i + \eta \Delta w$$

Suppose we take the case of one such weight w_1,

$$(w_1)^{i+1} = (w_1)^i + \eta \Delta w_1$$

Where,

$$\Delta w_1 = \frac{\partial}{\partial w_1} E$$

$$\Delta w_1 = \frac{\partial}{\partial w_1} ½\,(t_n - \widehat{t_n})^2$$

$$\Delta \mathrm{w}_1 = \frac{\partial}{\partial w_1} ½\,(\mathrm{t}_n - w^T \emptyset(x))^2$$

$$\Delta \mathrm{w}_1 = ½ * -2(\mathrm{t}_n - w^T \emptyset(x)) \frac{\partial}{\partial w_1} w^T \emptyset(x)$$

$$\Delta \mathrm{w}_1 = -(\mathrm{t}_n - w^T \emptyset(x)) \frac{\partial}{\partial w_1} (w_1 \emptyset_1(x) + w_2 \emptyset_2(x) + \cdots w_n \emptyset_n(x))$$

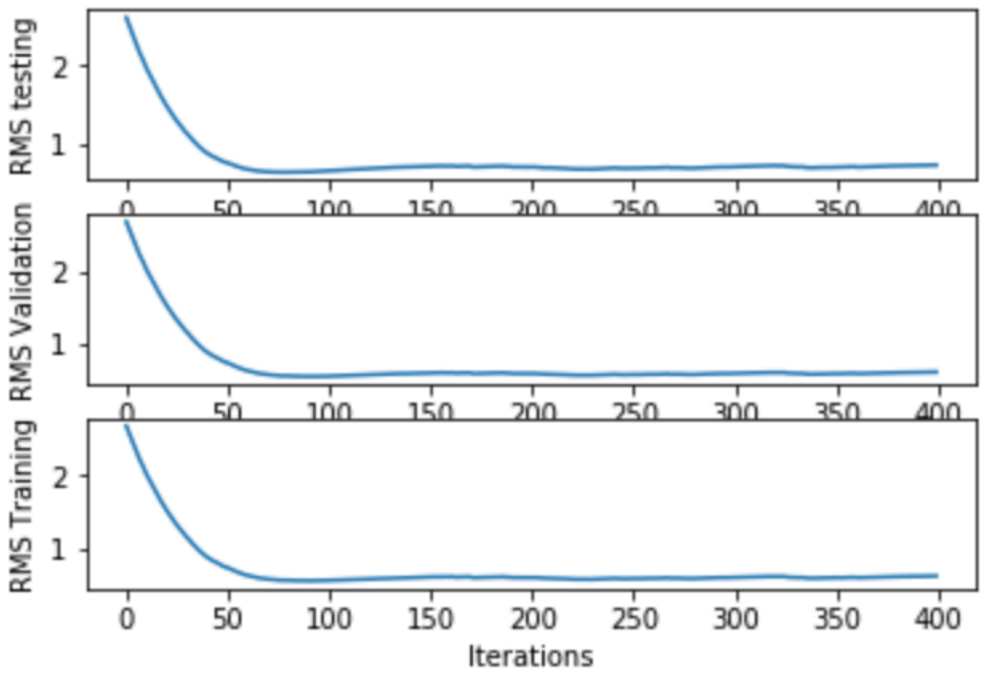
Hence,

$$\Delta \mathrm{w}_1 = -(\mathrm{t}_n - w^T \emptyset(x))(\emptyset_1(x))$$

Therefore,

$$(w_1)^{i+1} = (w_1)^i - \eta(\mathrm{t}_n - w^T \emptyset(x))(\emptyset_1(x))$$

The above formula is the one we use for updating the weights.
For gradient descent, we calculate the output values and the E_{RMS} in the exactly same way as we did for the closed form solution.
In the below figure we can see, how E_{RMS} decreases with the number of iterations.

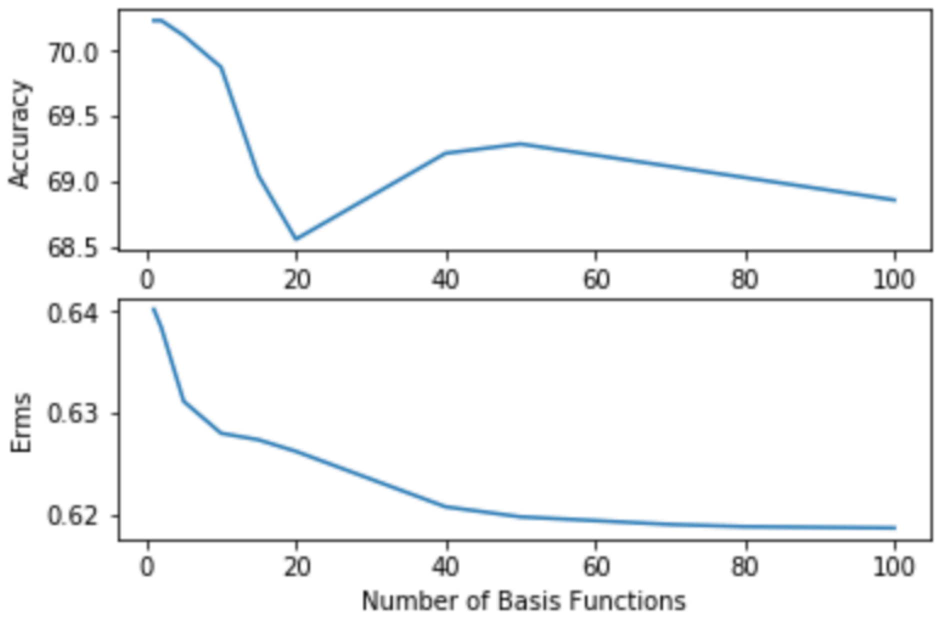


5 Tuning Hyperparameters

To achieve the optimal solution to the LeToR problem, we tune a number of hyperparameters and observe the change in the output.

5.1 Number of Basis Functions M

We change the number of basis functions from 1 to 100 and observe the output on testing data.

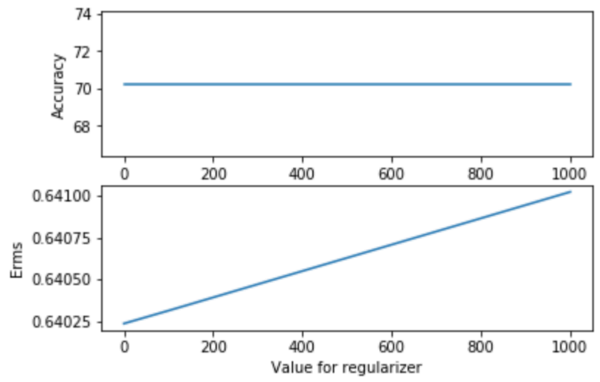


As we can observe, the highest accuracy is when the number of basis function is 1. This is because the nature of the dataset is such that it needs only one basis function. The dataset contains most of the target values as ‘0’, and only a few target values as ‘1’ and ‘2’. Due to this irregularity in the dataset, using one basis function is optimal.

5.2 Regularization term λ

The regularization term is used to prevent overfitting of data. It is added to the weights when computing the closed form solution.

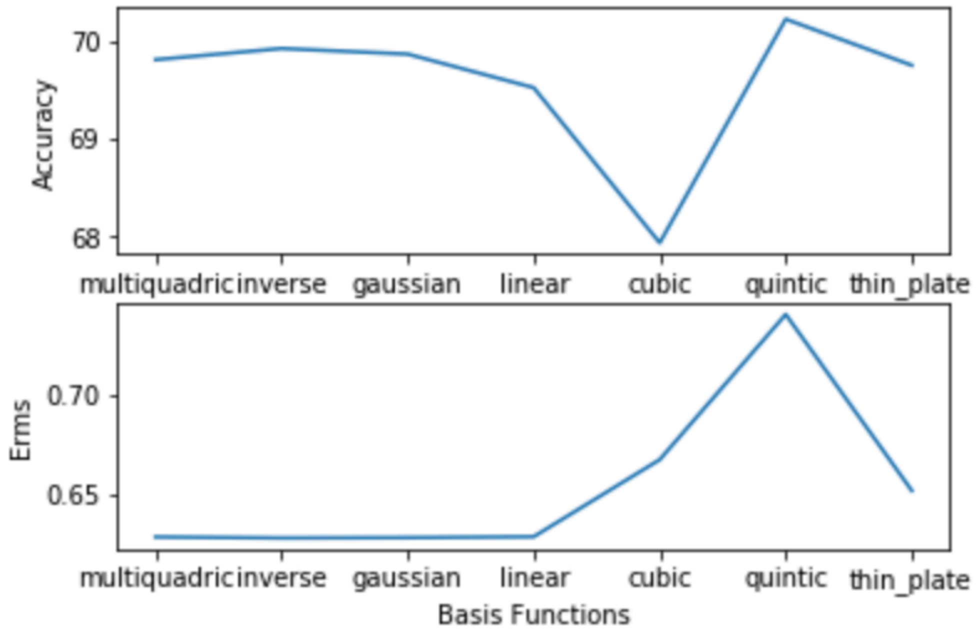
We change the value of λ from 0 to 1000 and observe the output on testing data.
Below is a graph showing the same.



The accuracy remains the same. However, the value for E_{RMS} increases. This is because when λ is increased very much it might lead to underfitting of data.

5.3 Choice of Basis Function

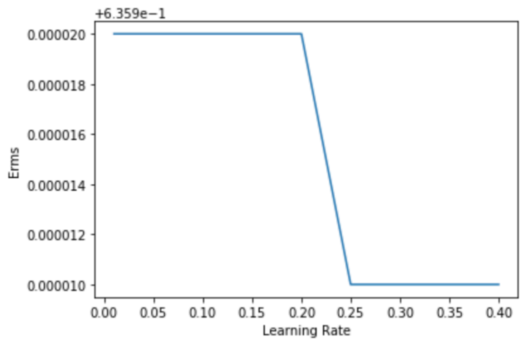
We use a number of basis function and compare output values for the same.
The types of basis functions we use are: {multiquadric, inverse, gaussian, linear, cubic, quintic, thin plate}
Below is a graph showing performance of all basis functions on testing data.



We can observe that accuracy is the highest when we use the quantic basis function. The dataset has 46 features. It is difficult to perform linear regression when we have so many features. Hence, a polynomial basis function will perform well in such scenarios. Even the gaussian basis function performs well with both accuracy and RMS error as optimal.

5.4 Learning Rate

The learning rate determines how gradually or quickly the weights of the model are updated. A high learning rate might not be able to finetune the network, however a low learning rate might become very slow for the parameters to adapt.
We compare the output of the gradient descent algorithm by changing the learning rate.
Below is the graph depicting the same.



As the learning rate increases, the error has remained almost the same.

6 Conclusion

Through this project, we solved the LeToR problem with linear regression using closed form solution and gradient descent approach. We achieve the highest accuracy of around 70% using the optimal set of hyperparameters and lowest value of $E_{RMS} = 0.63591$. However, there is more scope to this problem. As we have discussed, the nature of the dataset might be the problem as to why we have gained such performance. As the target values are majorly comprised of 0 the model is not able to fit properly to the data. We also conclude how various hyperparameters affect the model.

References

[1] https://en.wikipedia.org/wiki/Stochastic_gradient_descent
[2] https://arxiv.org/pdf/1705.07563.pdf
[3]https://compscicenter.ru/media/slides/information_retrieval_2016_autumn/2016_11_19_information_retrieval_2016_autumn_szTyoc5.pdf
[4] https://en.wikipedia.org/wiki/Learning_to_rank
[5] https://opensourceconnections.com/blog/2017/04/01/learning-to-rank-linear-models/
[6] https://en.wikipedia.org/wiki/Radial_basis_function
[7] https://www.cs.cmu.edu/afs/cs/academic/class/15883-f17/slides/rbf.pdf