Learning to rank using linear regression

Shruti Bendale
Person Number: 50289048,
School of Engineering and Applied Sciences,
University at Buffalo, Buffalo, NY, 14214
shrutitu@buffalo.edu

Abstract

The purpose of this project is to use machine learning to solve the Learning to Rank (LeToR) problem. We use two linear regression techniques to solve this problem of ranking. The tasks given for this project are:
1. Train a linear regression model on LeToR dataset using a closed-form solution.
2. Train a linear regression model on the LeToR dataset using stochastic gradient descent (SGD).

1. Introduction

Learning to Rank (LeToR)

In this project, we use the 'QueryLevelNorm' version of LeToR 4.0 package provided by Microsoft. This dataset contains 69923 vectors of 46 dimensions each. Each dimension corresponds to a feature namely IDF of terms in the body, anchor, title, etc. of documents.

Data Partition

80% of the total data was used as the training dataset, 10% of the data was used as the validating dataset and the remaining 10% was used to test the accuracy of the model.
55699 vectors were used for training.
6962 vectors were used for validation.
6962 vectors were used for testing.

Hyperparameters:

A hyperparameter is a configuration whose value cannot be estimated from data. They are often specified manually by searching for the best values using trial and error method. Hyperparameters have to be tuned to fit the model for generating maximum accuracy. Some examples of hyperparameters are the learning rate, batch size, activation functions, dropout, number of epochs, loss function, regularizer, number of layers in the model, etc. The values of the hyperparameters are set manually before learning as contrast to the 'parameters' where the values are learned during the training (E.g.: weights).

2. Experimentation

For the purpose of this project, some hyperparameters were chosen to observe the change in accuracy of the model for different values of the hyperparameters. The hyperparameters that were experimented with are:

i. Number of Basis Functions (M)
ii. Regularization Factor (λ)
iii. Learning Rate (η)

2.1. Closed Form Solution:

The closed-form solution should be preferred for "smaller" datasets when computing matrix inverse is not a concern. For very large datasets, or datasets where the inverse of X^TX may not exist Gradient Descent or Stochastic Gradient Descent approaches are to be preferred. The linear regression model is defined as:

$$y = w_0x_0 + w_1x_1 + ... + w_mx_m = \sum_{j=0}^{m} = \mathbf{w}^T\mathbf{x}$$

where *y* is the target variable, *x* is an *m*-dimensional sample vector, and *w* is the weight vector. w_0 represents the y-axis intercept of the model and therefore $x_0=1$. Using the closed-form solution (normal equation), we compute the weights of the model as follows:

$$w = (\Phi^T\Phi)^{-1}\Phi^Ty$$

Where y is the target vector and Φ is the design matrix. A model of degree M will have a design matrix of 1+(M-1)×D columns long, where D = 46 in our case. We get each set of 46 columns from each of the basis-functions Φ1, Φ2, … , ΦM respectively. Thus, the design matrix Φ is given as:

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

We use a regularization parameter λ to limit the increase in M which can make the model quite complex. We introduce λ in our equation for w and thus into ERMS as

$$w = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T y, \quad \text{where I is an identity matrix.}$$

After calculating the values of each target variable of the model, we can find out the error between the model's values and the actual values that are given in the data set. We measure this error by calculating the root means square error or E_{RMS} given as

$$E_{RMS}(w) = [(\Phi^Tw - y)(\Phi w - y)/N]^{1/2}$$

Choosing Number of Basis Functions (M) and Regularization Factor (λ):
For tuning M and λ, I used the grid search method of hyperparameter optimization. I started from small values of M and λ and gradually tried bigger values. The following table displays the values of E_{RMS} as a result of the grid search for the testing dataset:

M/ λ	0.0001	0.1	10
1	0.6402356065897246	0.6402356829691717	0.6402433226260125
3	0.6317830355082548	0.6317810747987833	0.6317075883939716
5	0.6311326003015552	0.6311306597723515	0.63108616668108
7	0.6289628277016461	0.6287923734178563	0.6284108561509404
9	0.629305490864239	0.62912780854054	0.628442309413297
10	0.6278388560960106	0.6281151520218362	0.628264930645502
20	0.6241022417804644	0.626836035673036	0.6278569681309243
40	0.6199751941770781	0.6219525988137692	0.6278728289688089

For the above set of parameters, when M=40 and λ=0.0001, the least error of 0.6199751941770781 was observed.

2.2. Stochastic Gradient Descent:

The way this optimization algorithm works is that instead of looking at the entire dataset at once, each training instance is shown to the model one at a time. The model makes a prediction for a training instance, the error is calculated, and the model is updated in order to reduce the error for the next prediction. This process is repeated for a fixed number of iterations.

This procedure can be used to find the set of coefficients in a model that result in the smallest error for the model on the training data. Each iteration, the weights (w) are updated using the equation:

$$w^{(\tau+1)} = w^{(\tau)} + \Delta w^{(\tau)}$$

$$\text{where } \Delta w^{(\tau)} = -\eta^{(\tau)} \nabla E$$

Where w is the weight being optimized, η is the learning rate that we'll be configuring, ∇E is the prediction error for the model on the training data.

Configuring the learning rate (η):

The learning rate was adjusted from 0.0001 through 0.1 with increments of 10 keeping the value of M as 1. The following table shows the values of η vs. the error for the testing dataset:

η/ E_{RMS}	E_{RMS}
0.0001	59.73416
0.001	21.36789
0.01	0.63593
0.1	21.36789

From the above table we can infer that the learning rate of 0.01 generates an optimal solution.

3. Conclusion

Two linear regression techniques were used to train our model: the closed form and stochastic gradient descent. The following observations were made:

- The closed form solution is more efficient if used for smaller datasets which is why for the datasets like the one used in this project, the stochastic gradient descent technique is preferred.
- The stochastic descent generates less error value as compared to the closed form but takes more computation time.
- Higher values of M generate better accuracy but take more computational time. Thus we must decide on a value of M which is optimal but doesn't take very long to train the model.

4. References

1) https://ublearns.buffalo.edu/bbcswebdav/pid-4714290-dt-content-rid-19922272_1/courses/2189_24904_COMB/project1.2.pdf
2) https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html
3) https://machinelearningmastery.com/implement-linear-regression-stochastic-gradient-descent-scratch-python/