Learning to Rank using Linear Regression

Hrishikesh Nitturkar
University at Buffalo
hnitturk@buffalo.edu

Abstract

In this paper, we focus on using a Linear Regression model to solve the LeToR problem (Learning to Rank) that arises in Information Retrieval. This involves mapping an input vector x to real scalar value returned by a function y(x,w) where 'w' denotes the vector of weights assigned to each input. The methods used to address the LeToR problem in this case are 'Closed Form Solution' and 'Stochastic Gradient Descent(SGD)'.

1 Introduction

Learning to Rank using machine learning is a way of constructing ranking models for information retrieval which computes the relevance of the items in the datasets for a particular query. It consists of a dataset used by a machine learning algorithm to produce a numerical score or binary judgement ('relevant' or 'not relevant') on the inputs in the dataset.

In this case, we address this issue using linear regression by mapping an input vector x to a real-valued scalar target given by y(x,w) using two methods - Closed Form Solution and Stochastic Gradient Descent.

The training data consists for pairs of input values x and target values t. The input values are real valued vectors. The target values are scalar values limited to 0, 1 and 2 which are the relevance labels. Higher the relevance label, the better is the match between query and result. Using real values for linear regression is often better since it avoids all the queries hitting only 3 values. Real values provide probabilities of better matches between query.

1.1 Model

The model of linear regression in this problem is given by

$$\boxed{y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^{\top}\phi(\mathbf{x})} \tag{1}$$

where,
w = (w_0,w_1,w_2.......w_{n-1}) is a vector of weights to be derived or learnt from the training samples as the training is carried out by assigning multiple weights to match the input better to a output.
$\phi = (\phi_0, \phi_1, \phi_2.......\phi_{n-1})$ is a vector of M radial basis functions. Each basis function convers the input vector x into scalar quantity.

1.2 Radial Basis Function

A radial basis function is a real-valued function that satisfies the condition $\phi(\mathbf{x}) = \phi(||\mathbf{x}||)$; or alternatively on the distance of some other point c, called the center, so that $\phi(\mathbf{x}, \mathbf{c}) = \phi(||\mathbf{x} - \mathbf{c}||)$. It gives values to a point in space based on their distance from the origin.

1.3 Gaussian Radial Basis Function

In this experiment, we are going to use the Gaussian Radial Basis Function which is represented by

$$\boxed{\phi_j(\mathbf{x}) = exp\left(-\frac{(x-\mu_j)^2}{2\sigma^2}\right)} \tag{2}$$

which is a scalar quantity equation that returns a scalar value for all the values of inputs, x, that are given to it. Its vector representation is given by

$$\boxed{\phi_j(\mathbf{x}) = exp\left(-\frac{1}{2}(\mathbf{x}-\mu_j)^\top \Sigma_j^{-1}(\mathbf{x}-\mu_j)\right)} \tag{3}$$

where μ_j is the center of the basis function and Σ_j describes how broadly the basis function spreads.

The Gaussian model gives rise to multi-dimensional covariance matrix required for the radial basis functions. The number of dimensions in the covariance matrix is given by the number of features in the model. In our case, the number of features is 41. Hence we get a covariance matrix of order 41x41 as shown by the following matrix.

$$Co-variance\ Matrix\ \Sigma = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \sigma_{13}^2 & \dots & \sigma_{141}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 & \sigma_{23}^2 & \dots & \sigma_{241}^2 \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ \sigma_{411} & \dots\dots\dots\dots & & & \sigma_{41}^2 \end{bmatrix}$$

Here σ_{12} represents th value of feature 1 with respect to feature 2. Since the inputs provided are not dependant on the other inputs, we set the elements other than the diagonal elements of the covariance matrix to '0'

1.4 Likelihood function

We are using a probabilistic model in which the output target value is subject to noise and has a normal distribution, with mean $y(\mathbf{x}, \mathbf{w})$ and precision β. Input samples X =$\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ and target values t = $\{t_1, t_2, ...t_n, \}$, the likelihood functions has the form,

$$\boxed{p(\mathbf{t}|X, \mathbf{w}, \beta) = \prod_{n=1}^{N} N(t_n|\mathbf{w}^\top \phi(\mathbf{x}_n), \beta^{-1})} \tag{4}$$

The above function returns a probable value of t with respect to input samples X, weight vector $\mathbf{w}$ and precision β by calcuating the product of all the variabilities of the target variables based on our linear regression model.

The likelihood function defines the likelihood of an input of being of a particular type. Maximizing the likelihood function is same as minimizing the error in the distribution which here is, sum of squares error given by,

$$\boxed{E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \{t_n - \mathbf{w}^\top \phi(\mathbf{x}_n)\}^2} \tag{5}$$

A regularization term is added to the above error function to improve generalization and avoid overfitting, with the form:

$$\boxed{E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})} \tag{6}$$

where λ governs the relative importance of the regularization term. We have to minimize the above equation by finding $\mathbf{w}^*$ by taking derivative of the equation with respect to $\mathbf{w}$, setting it to 0 and solving for $\mathbf{w}$.

And the weight decay regularizer is ,

$$\boxed{E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w}} \tag{7}$$

The weight decay regularizer is used to limit the size of the weights. It keeps the weights small unlss there are big error derivatives. This term specifically improves generalization and stop overfitting the error data and it makes the model smoother where output changes more slowly s the input changes.

To solve for $\mathbf{w}$ as discussed, we use two linear regression solutions: Closed Form Solution and Stochastic Gradient Descent.

2 Description

2.1 Closed Form Solution

Closed Form Solution is one of the solutions to the linear regression problem. We start by considering the linear regression form as discussed above.

$$\boxed{y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \phi(\mathbf{x})} \tag{8}$$

Since we have a target variable $\mathbf{t}$ the above equation changes to

$$\boxed{\mathbf{t} = \mathbf{w}^\top \phi(\mathbf{x})} \tag{9}$$

where w = vector of weights $\mathbf{w_1}, \mathbf{w_2}, ...$ and ϕ is vector of basis functions $\phi_\mathbf{1}, \phi_\mathbf{2}, ...$

which then translates to,

$$\boxed{\mathbf{t} = \mathbf{w_1}\phi_1(\mathbf{x}) + \mathbf{w_2}\phi_2(\mathbf{x}) + ...} \tag{10}$$

Now targeting for the weight vector which is the goal of this solution, we have

$$\boxed{\mathbf{w} = \phi^{-1}(\mathbf{x})\mathbf{t}} \tag{11}$$

which we later translate to

$$\boxed{\mathbf{w} = (\phi^\top \phi)^{-1}\phi^\top \mathbf{t}} \tag{12}$$

The above equation is the Moore-Penrose inverse of the matrix ϕ. The Moore-Penrose inverse, a pseudoinverse of a matrix is a generalization of the inverse matrix ϕ.

In quantity (12) $\mathbf{t}$ = $\{\mathbf{t_1}, \mathbf{t_2}, ...\mathbf{t_N}\}$ and ϕ is the design matrix:

$$\Phi = \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) & \dots & \phi_M(x_1) \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots \end{bmatrix}$$

The above matrix has the order of $\mathbf{nxM}$ where n is the number of inputs and M is the number of radial basis functions.

Each value in the matrix is a scalar value returned by a basis function by operating on the input vector or values x and x is the vector of values of inputs $\mathbf{x_1}$ with repect to the number of features in the experiment. Hence x is given by $\mathbf{x_1} = [e_{11} \quad e_{12}.... \quad e_{141}]$

So, it can be represented as,

$$\boxed{x_1 \rightarrow \Phi \rightarrow \phi_1(x_1)} \tag{13}$$

In our experiment we are using a target vector where an input can collide to make the prediction. It is given by $\mathbf{t} = [0 \quad 1 \quad 2]$. Higher the output for an input vaariable, the better is the match. To increase the chances of making better predictions, we use real values bounded by 0 and 2.

2.2 Stochastic Gradient Descent

In linear regression our goal is to find the minimum value of the cost function so that the error is minimized. In other words, gradient descent helps us make smaller adjustments to the weights and find a minimum change that has to be done to the initial weights so that the error function is minimized. The magnitude and direction of the weight update is computed by taking a step in the opposite direction of the cost gradient.

In GD optimization, we compute the cost gradient based on the complete training set; hence, we sometimes also call it batch GD. In case of very large datasets, using GD can be quite costly since we are only taking a single step for one pass over the training set - thus, the larger the training set, the slower our algorithm updates the weights and the longer it may take until it converges to the global cost minimum.

In Stochastic Gradient Descent, we don't accumulate the weight updates for every epoch as in gradient descent. Instead we update the weights after each training sample. Here, the term "stochastic" comes from the fact that the gradient based on a single training sample is a "stochastic approximation" of the "true" cost gradient. Due to its stochastic nature, the path towards the global cost minimum is not "direct" as in GD, but may go "zig-zag" if we are visualizing the cost surface in a 2D space. However, it has been shown that SGD almost surely converges to the global cost minimum if the cost function is convex

So we take a random value to be our initial value $\mathbf{w}_{(0)}$. The change required to this value is given by

$$\boxed{\Delta\mathbf{w}^{(\tau)} = -\eta^{(\tau)}\nabla E} \tag{14}$$

It goes along the opposite direction of the error. $-\eta^{(\tau)}$ is the learning rate and decides how big or small the change should be.

Thus the weights are updated using the equation:

$$\boxed{\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}} \tag{15}$$

Because of linearity we have

$$\boxed{\nabla E = \nabla E_D + \lambda\nabla E_W} \tag{16}$$

in which

$$\boxed{\nabla E_D = -(t_n - \mathbf{w}^{(\tau)^\top}\phi(\mathbf{x}_n))\phi(\mathbf{x}_n)} \tag{17}$$

$$\boxed{\nabla E_{\mathbf{w}} = \mathbf{w}^{(\tau)}} \tag{18}$$

The above solution can be evaluated on a test set using Root Mean Squared (RMS) error, defined as

$$\boxed{E_RMS = \sqrt{2E(\mathbf{w}^*)/N_v}} \tag{19}$$

where $\mathbf{w}^*$ is the solution and $N_{\mathbf{v}}$ is the size of the dataset.

3 Algorithm or Experiment Execution

In our task of Learning to Rank, we use the a dataaset to implement linear regression on. We are using a Microsoft LETOR 4.0 Dataset for this experiment. The entire dataset contains 69623 query document pairs each having 46 features out of which 5 are zeroes. Hence we operate with 41 features. They are the 46-dimensional input vector x for our linear regression model. All the features are normalized to fall in the interval of [0, 1].

We get the data files from 'Querylevelnorm_X.csv' and 'Querylevelnorm_t.csv' files. 'Querylevelnorm_t.csv' file gives the relevance labels and 'Querylevelnorm_X.csv' files gives us the query document pairs or the dataset required. The first column in the dataset is the relevance label of the row. It takes discrete values 0, 1 or 2. The larger the relevance label, the better is the match between query and document. The objective output y of our linear regression will give a continuous value rather than a discrete one so as to give a fine-grained distinction. This forms our target vector t.

We partition the dataset into three parts - a training set, a validation set and a testing set with 80%, 10% and 10% of the dataset respectively. As a result we have training matrix, testing matrix and validation matrix of order 41 x 55699, 41 x 6962 and 41 x 6962 respectively. Then the model is trained is performed on the training set using the hyper-parameters such as $\Sigma_j, \lambda, \eta^\tau, \mu_j, M$.

We start with setting the values of M as 10 and C_Lambda as 0.9. Then prepare the dataset and partition the data as discussed above. The training data matrix comes out to be of the order 41 x 55699. The validation data matrix has the order 41 x 6962. The testing data also has the same order where 41 denotes the number of features and the other number denotes the number of inputs in each set.

Then according to the closed form solution, the dataset is clustered using kmeans clustering.kmeans clustering is used to choose basis radial functions by clustering the data to create labels. In this project, we keep the number of clusters equal to number of radial basis function. Then we randomly choose M data centers as centers for gaussian radial basis function and spread for gaussian radial basis functions is found next. Using these details we generate the BigSigma matrix which is the co-variance matrix. Using this matrix, the design matrix (Phi matrix) for the training set can be found. Using this information, we calculate weights required to train the model using closed for solution which ultimately serves the purpose of this project. Then we have to evaluate the solution on the root mean squared error. Using these values accuracy of the model for training, testing and validation datasets. For the hyper parameters mentioned above, we get the following E_rms values.

E_rms Training = 0.5498628487549042
E_rms Validation = 0.5392063570106923
E_rms Testing = 0.6282278135552603

Now let us change the value of M to a 100. The accuracy are affected a little. The values that we get are

E_rms Training = 0.5416668385449263
E_rms Validation = 0.5363864364027545
E_rms Testing = 0.6210347810546649

when we raise the lambda value to 2.0, the accuracy decreases just a little.

E_rms Training = 0.5428452341882228
E_rms Validation = 0.5366761061650219
E_rms Testing = 0.6223687219111481

When M = 400 and lambda = 2.0 Order of Mu is 400 x 41, co variance matrix has the order 41 x 41, design matrices has orders of 55699 x 400, 6962 x 400, 6962 x 400 for training data, validation data and testing data respectively.

The accuracy values generated are, M = 400 Lambda = 2.0 E_rms Training = 0.540273556497256
E_rms Validation = 0.5361089922972998
E_rms Testing = 0.6197952507928114

When M = 50 and lambda = 0.3

M = 400
Lambda = 0.3
E_rms Training = 0.5383525326610082
E_rms Validation = 0.5350153274925858
E_rms Testing = 0.618505089657095

Now let us consider the stochastic gradient descent solution for the same problem. We start the execution by initializing the hyper parameters - learning rate to 0.01 and lambda to 2 and multiplying the weight matrix generated above by a scalar 220. We do this to randomize the weights obtained by the closed form solution. Then we randomly select the number of datapoints to be used and start the process. Now for every datapoint weights are updated using previous value and learning rate to try reach the minimum change that is necessry to increase the acccuracy. We have to be careful about regulating learning rate since we can either overshoot and miss the global minimum of the gradient descent graph if the learning rate is too high or if learning rate is too low, it may take a lot of time to reach the minimum. We have to exepriment and try to optimize the learning rate.

The errors are greatly increased. We get the following values,

M = 400
Lambda = 2.0
eta=0.01
E_rms Training = 34.84838
E_rms Validation = 34.75518
E_rms Testing = 35.74967

When we change the values of few parameters such as lambda and learning rate, and also change the number of datapoints to use to 10 the accuracy decreases enormously

M = 400
Lambda = 0.6
eta=0.5
E_rms Training = 1242.91849
E_rms Validation = 1243.26216
E_rms Testing = 1241.46983

When M = 15, Lambda = 0.0001 and learning rate = 0.01,

M = 15
Lambda = 0.6
eta=0.5
E_rms Training = 0.54582
E_rms Validation = 0.53915
E_rms Testing = 0.62227

4 Observations

In Closed Form solution:

Increase in number of basis functions in the closed form solution for this linear regression model resulted in increase of the accuracy because the error function was minimized. This is because increase in the number of basis functions make the inputs go through more processing and thus have more number of weights associated to each input that can be used to find a better match of query to the document. Since we are using real values, it is possible there may be very less change in the accuracy values. But even this little changes can contribute signnificantly towards the accuracy of the model. It is important to note that number of basis functions is directly proportional to the accuracy of the model.

Increase the regularization rate affects the model inversely which means, that when regularization rate is high, the accuracy is less. This is because, the main function of a regularization is to reduce ovefitting of the data. But when we increase it, it may reduce th overfitting but also decreases the accuracy. And when we decrease it, the accuracy may be increased at the risk of overfitting.

In Stochastic Gradient Descent solution:

When we change the number of datapoints to be used for the experiment, it affects the accuracy of the model greatly. For example, with a low number of data points the errors are increased enormously. This is because datapoints are sample points from the large collection of data. Since running every data point through the algorthm consumes a lot of time, it is better to consider few data points that can represent all the others with similar features and this helps us in making a suitable prediction about the other points in our data space.

Increasing the learning rate gradually helps us get close to the minimum change value and the change starts to increase after a certain point. This means that we have increased the learning rate too much and now it is time to decrease it in smaller intervals till we attain the best possible minimum value for the learning rate.

Acknowledgments

This report is supported by the entire teaching faculty of CSE Department for the course CSE 574 - Introduction to Machine Learning for Fall 2018 semester

References

[1] ShareLaTex concepts for various mathematical representation retrieved from *https://www.sharelatex.com/*

[2] Machine Learning: Cost Function and Gradient Descent Optimization concepts, retrieved from *https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd*

[3] Linear Gaussian Function retrieved from *https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions and https://en.wikipedia.org/wiki/Gaussian_function*

[4] Gradient Descent vs Stochastic Gradient Descent retrieved from *https://www.quora.com/Whats-the-difference-between-gradient-descent-and-stochastic-gradient-descent*