LEarning TO Rank(LETOR) using Linear Regression

Raghavendra Prakash Nayak

***Abstract*—the project involves implementing linear regression to solve regression problem and evaluate its performance. In this project we map the input vector x to its scalar target t for a real world dataset released by Microsoft Asia and a synthetic dataset. Both the datasets were trained using closed form solutions and stochastic gradient descent.**

.

I. Introduction

LEarning TO Rank (LETOR) or machine learned ranking is an application in machine learning, typically involved in construction of ranking models for information retrieval systems. Training data consists of lists of items with some partial order specified between items in each list. This order is typically induced by giving a numerical or ordinal score or a binary judgment (e.g. "relevant" or "not relevant") for each item. The ranking model's purpose is to rank, i.e. produce a permutation of items in new, unseen lists in a way which is "similar" to rankings in the training data in some sense.

In an information retrieval system, when machine learning is applied to it, the query-document pairs are usually represented by numerical vectors, which are called feature vectors. Such an approach is sometimes called bag of features and is analogous to the bag of words model and vector space model used in information retrieval for representation of documents.

In this project we focus on a dataset of university rankings, tuition etc. based on a survey conducted by US News and Chronical to construct a Bayesian network with high log-likelihood. Furthermore we try to use these Bayesian networks to determine some conditional probabilities.

II. The Dataset

For this project, I have used of 2 datasets. The first dataset, is a real world dataset- Microsoft LETOR 4.0 Dataset, made available by Microsoft Asia. LETOR is a package of benchmark data sets for research on Learning To Rank released by Microsoft Research Asia.

It contains 8 datasets for four ranking settings derived from the two query sets and the Gov2 webpage collection. For this project, I have downloaded MQ2007. And from this, I have selected the “QueryLevelNorm.txt" for implementing the linear regression models. The entire dataset consists of 69623 query-document pairs(rows), each having 46 features.

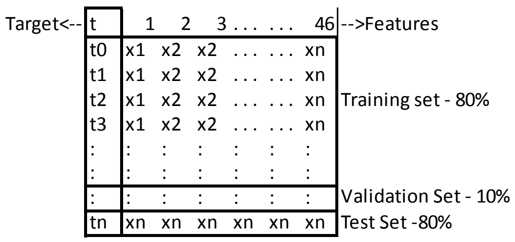
The meaning of each column are as follows. The first column is the relevance label of the row. It takes value 0, 1 or 2. This is the objective output y we expect our linear regression to give. The second column qid is the query id. However, this is not used in this project. The following 46 columns are the features. They represent the 46-dimensional input vector x for the linear regression model. All the features are normalized to fall in the interval of [0; 1]. The last columns contain the docid, inc and prob, these are not used in this project and are hence omitted out during document processing.

The second dataset that was used in this project was, a synthetic dataset with 10 features and 2000 query-documents. A 2000x10 matrix, along with a target dataset of 2000x1 matrix.

III. The Method

The real world dataset “Querylevelnorm.txt” was processed in MATLAB, where only the input data, target data and the features were imported into a matrix of size 69623x47. This was then segregated into the target dataset (69623x1) and the input dataset (69623x46). The input dataset was then split into 3 parts, with 80% of the dataset being stored for training, 10% for validation and the last 10% for testing. The synthetic dataset was also divided into training, validation and testing as done to the real world dataset.

Partition of Dataset:
The dataset has been partitioned as shown below;
Real world dataset LETOR (MQ2007) - Microsoft Asia



Both the real world and the synthetic dataset has been partitioned as per the figure shown. The final partition of these are given below:

Real world dataset:

Input Training Set	= 80% of the entire input data set
	= 55698 x 46 matrix
Target Training Set	= 80% of the entire input data set
	= 55698 x 1 matrix
Input Validation Set	= 10% of the entire input data set
	= 6962 x 46 matrix
Target Validation Set	= 10% of the entire input data set
	= 6962 x 1 matrix
Input Test Set	= 10% of the entire input data set
	= 6961 x 46 matrix
Target Test Set	= 10% of the entire input data set
	= 6961 x 1 matrix

Synthetic world dataset:

Input Training Set	= 80% of the entire input data set
	= 1600 x 10 matrix
Target Training Set	= 80% of the entire input data set
	= 1600 x 1 matrix
Input Validation Set	= 10% of the entire input data set
	= 200 x 10 matrix
Target Validation Set	= 10% of the entire input data set
	= 200 x 1 matrix
Input Test Set	= 10% of the entire input data set
	= 200 x 10 matrix
Target Test Set	= 10% of the entire input data set
	= 200 x 1 matrix

In this project, I have used two models; namely the Closed Form Solution and the Stochastic Gradient Descent model:

A. *Closed Form Solution:*

This linear regression function y(x,w) has the form;

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$$

Where w is the weight vector to be learnt from training samples and phi is the vector of M basis functions.

Basis Function:

The closed-form solution is the simplest type of regression model. For the closed form solution I have created a NxM size matrix. Where N stands for number of training samples and M stands for the number of Basis functions. We call this matrix the design matrix or f. The basis function I have used is;

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^\top \Sigma_j^{-1}(\mathbf{x} - \boldsymbol{\mu}_j)\right)$$

The Gaussian basis function gives one scalar value for one row/document/observation. I have used kmeans in MATLAB to find the means that are non-zero. As µj is the amplitude/height of the Gaussian function and sigma is the width of Gaussian function takes. So if µj takes the mean of every known features we get one scalar value for that feature which is then marginalized by the standard deviation to give the best result.

Design Matrix:

I have varied the value of M from 2 to 10. I first column of the Design matrix is constant and I have fixed the value to 1.

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

So as my Design matrix dimensions change I vary my calculations accordingly. As the number of Basis functions increase the complexity of the model increases. The design matrix is then used by the closed form solution to calculate the weight matrix. I also save the Design matrix to disk so I can use it later for other models. This is necessary especially in the case of random number Gaussian function. Also it makes the program run a lot faster.

Weight Vector:

I have then calculated the weight vector from the Design matrix. Here is the formula for the weight vector.

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

I have made the Design matrix of the form of NxM and the value of t is the relevance labels of the training dataset. So we get our weight matrix which has the size of 1xM. As the weight vector is the weight assigned to the scalar values from the Gaussian functions. Lambda is there to stop over-fitting.

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

Next, I calculated the error of the predictions minus the actual value whole squared and then summed. Of course this is our least square error for the entire data set.

Lambda:

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

The challenge is to make sure we don't over-fit our training dataset to the notion of lambda is introduced. The process is to vary the value of lambda over our calculated value of the weight vector to give us the best possible answer. Lambda provides protection for over-fitting by adding the squared and summed value of the weights to the real error.

$$E_W(\mathbf{w}) = \frac{1}{2}\sum_{j=0}^{M-1} |w_j|^q$$

Thereby, making harder to over-fit as we increase the complexity of the model the lambda part of the equation increases and we can control the complexity of the model. The value of lambda should be small otherwise the weights might overplay their part in the final error calculation. I have varied the value for lambda in the project from 0.01 to 0.1.

Error Root Mean Square:

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

So, as to get the idea of the error per document/observation. We divide the final error by number of documents and get the Erms value. The value comes out between 0.54 to 0.56 for my closed-form model. The value is affected by the random numbers so it changes a slightly sometimes. But it does decrease as the complexity increases but only to a certain extent. After that the regularization causes the error to increase. Here is the graph for Erms X M X lambda.

B. *Stochastic Gradient Descent:*

The gradient descent algorithm takes a row from the dataset and calculates the weight vector according to that row in the dataset. Then that weight vector is applied to a subset of the rows to give some error and we go on a new row, new weights and new error till the error converges.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} + \eta(t_n - \mathbf{w}^{(\tau)T}\phi_n)\phi_n$$

Design Matrix:
The design matrix was calculated the same way as in case of Closed Form Solution.

Weight Vector:
What is different here is that the initial weight vector is of random values. It is used in the initial iteration to find the next weight vector and so on till convergence.

Learning Rate η:
As we take steps towards convergence we do so in iterations. η helps measuring those steps. We take big steps we can overshoot our target and if our steps are too small it takes a long time for convergence.

C. *Setting Model Paramaters:*

One important aspect of the project is to choose correct values for mu and sigma for the basis functions. For this case, we have partitioned the feature matrix X into M partitions. Then we have randomly selected a feature from each partition and computed its mu and sigma. In case of sigma, we added a small offset value because there are features in the dataset for which the value is zero for all data points.

Selecting M, Lambda and Eeta:
At first we vary M and keep λ fixed. Then we can observe the following trend. If we increase the value of M then the error also increase and becomes constant. If we decrease M, then also the error keeps increasing. The values of M were hardcoded into the MATLAB program for up to a model complexity of 10. All of the parameters were hardcoded and tuned and then the most appropriate values were selected in the program.

IV. RESULTS

The outcome of the learning models are reported in terms of estimated error. The rms error is calculated using the formula

$$E_{rms}(w) = \sqrt{\frac{2E_d}{N}}$$

Following are the results;
Real world dataset:
Model Complexity = 10;
Lambda = 0.00000073
Eeta = 0.01
Erms(Training Set)= 0.5450

Synthetic dataset:
Model Complexity = 10;
Lambda = 0.00000073
Eeta = 0.1
Erms(Training Set)= 0.1137

V. REFERENES

1. ‘Learning to Rank’ –wikipedia - https://en.wikipedia.org/wiki/Learning_to_rank