Introduction to machine learning

Project 1.2

Cse 574

Ubit : v33 person#:50292192

Introduction

In this project, we are provided with learning to rank data set. Our goal is to predict page relavence using all the other features. We formulate this as linear regression problem and solve using closed form and gradient descent method .

Data preprocessing :

The given data is in text format and cannot be manipulated or used for prediction. So, we convert this data into csv file for computational purpose. Target values and all the features used for prediction are stored in two different csv files.

Deleting features: Among the 46 features of data available to us we need to delete few of the features for our computational purposes.The features which produce a zero variance are being deleted and this would cause us a problem during finding the inverse of the matrix in closed form solution. So there are a total of 5 such features in our data which form a zero variance so those 5 features are being deleted and the number of features are being reduced to 41. Now the transpose of the data matrix is being formed and the features are being stored along the rows and the samples along the columns. The dimensions of the new matrix formed is 41*69623

After deleting the columns , 80% of the data is considered for training and 10% validation . Rest of the data is used for testing .

Closed form solution :

The dataset contains 41 features and it is difficult to represent the target variable as linaer combination of these features. So we transform the coordinate system to 10 dimensions from 41 dimension. This is done by using basis functions. We assume that the data is combination of 10 gaussian distributions with different mean. Where mean can be visualized as center of each gaussian distribution. To get ten different gaussian distributions, we cluster the data into 10 clusters using k means algorithm. Now we write the equation in the linear form of basis functions.

$t = w_1f_1+w_2f_2\ldots\ldots\ldots\ldots+w_{41}f_{41}$ **to** $t = t=W_1\Phi_1(x) + W_2\Phi_2(x)+ \ldots\ldots\ldots\ldots+W_{10}\Phi_{10}(x)$. where

m: number of basis functions **phi**: it is the basis function.

t : target value **f**: feature

Caluclation of new design matrix :
To calculate the basis function we use the vector form of the Gaussian function which is represented in the box.

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

Input data matrix having 41 features with dimension(1*41).(41 features because we remove the zero variance features)

Is the center of the basis function ie.,mean of the phij(x) having 41 features with dimension(1*41)

μ: The mean for each feature is different so in total we form 41 means in each cluster and a matrix is formed to store the mean of all the features and the matrix has a dimension of 10*41

Big-Sigma : Covariance matrix of the input features with dimensions (41*41). The covariance matrix is the diagonal matrix because all other elements are zero as the features are independent of one another. In the covariance matrix we also scale the values of variances with a factor so that we can have a better usage of the values. Here we have chosen 200 as the scaling factor.

The inverse of a covariance matrix gives

$$\boldsymbol{D} = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_2 \end{bmatrix} \Rightarrow \boldsymbol{D}^{-1} = \begin{bmatrix} 1/d_1 & 0 & \cdots & 0 \\ 0 & 1/d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/d_n \end{bmatrix}$$

Now using the above equation we can get the phi matrix with dimensions(58000*10).(with taking m=10 and and training input as 58000). That is the design matrix:

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

Here N=58000 and M=10 Thus the design matrix is formed.

Thus we have 58000x10 matrix of training data. If we see the scatterplot matrix (fig1)we observe that some of the variables in the transformed design matrix are highly correlated. When variables are highly correlated, the weights of a single(multiple also) predictor can go very large to compensate with negative coefficients on correlated counterparts. This can lead to very high variance among parameters. So, we penalize sum of squares of weights to find out the optimum weights. This will make the model little biased but reduces lot of varience(sum of squares).The amount of bias is determined by lambda

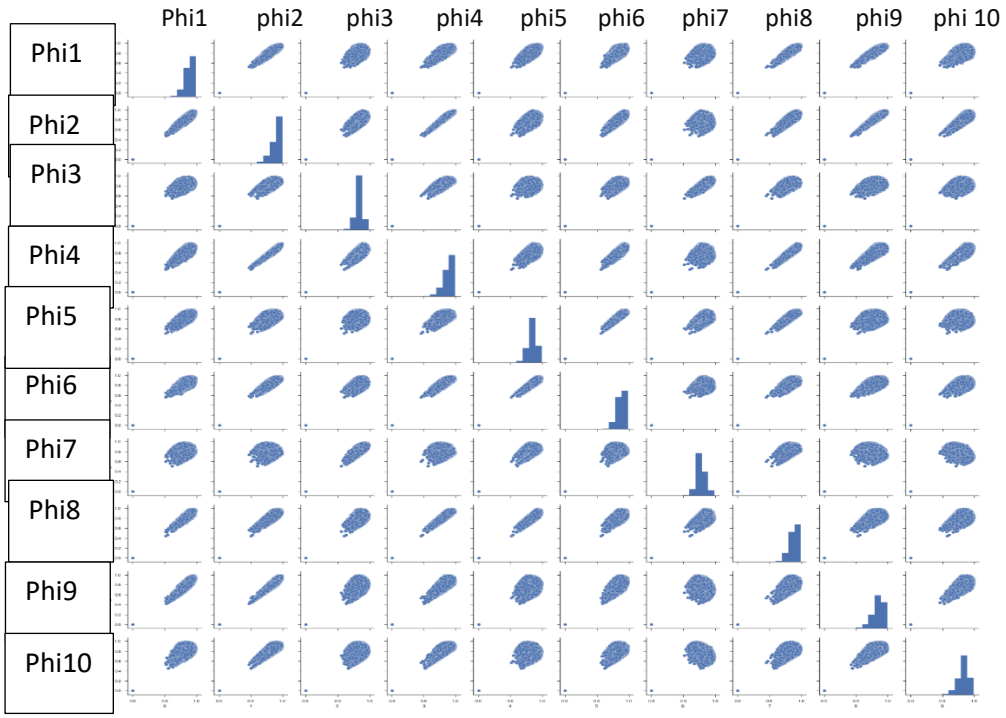


Fig 1

For example , consider phi2 and phi4 . The scatter plot between them extremely correlated . As phi2 increases ph4 also increases .With this correlation if we train the data the weights of the corresponding variables will be large and we get a large rms error. So we define the cost function as sum of squares plus square of weights times the lambda. This condition ensures that weights of the variable are not huge. This kind of regularization is called ridge regression.

Gradient descent solution :

In this method we define the cost function to be the sum of squares of differences between predicted and actual output.

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

Where θ is set of weights. This cost function can be visualized as n dimentional curve where n is the number of weights. Each set of weights correspond to a point on this cost function. Our goal is to find the minimum of the of this cost function. Minimum can be obtained by decreasing each weights iteratively until we reach the minimum value of J(θ). This is done by calculating the gradient of the function and decreasing the weights in the opposite direction of the gradient. Amount of decrement is determined by α (learning rate) .

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta).$$

Above update is performed for all weights (j=1,2……n) until minimum is obtained

Repeat until convergence {

$$\theta_j := \theta_j + \alpha\sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)} \qquad (\text{for every } j).$$

}

This method of updating the weights has a disadvantage. for every iteration all the rows in the training data set are read. This will take a lot of time to run if the data set is huge. In such cases stochastic gradient descent is used which we repeatedly run through the training set, and each time we encounter a training example, we update the parameters according to the gradient of the error with respect to that single training example only

RESULTS:

Log(lambda)	No_of_clusters(M)	Training error (rms)	Testing Error (rms)
-4	10	0.5528	0.6311
-2	10	0.549	0.6275
0	10	0.5501	0.626
1	10	0.552	0.631
2	10	0.555	0.632
6	10	0.605	0.7000
0	5	0.559	0.6324
0	2	0.5732	0.638
0	15	0.547	0.627
0	30	0.543	0.623

Let us analyse each of the following results

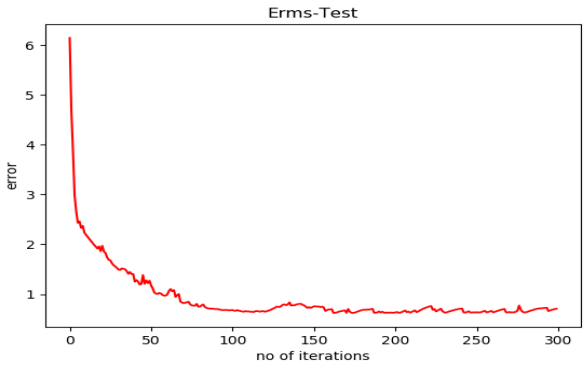
Since the values in our dataset are extremely low (all of the values are in between 0 to 1) , even a small change in the rms error is a significant one.

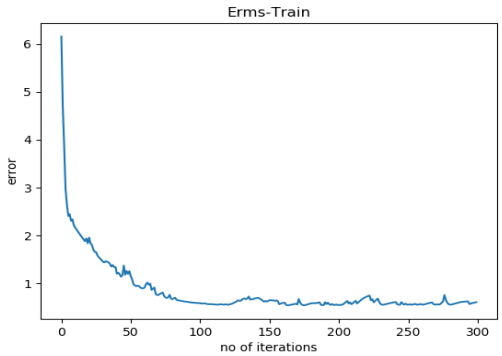
Initially keeping the no of clusters as 10 , we vary the lambda's from 10^-4 to 10^6 . It is clear from the table that when lambda is tending to zero, Rms error increases. When lambda is zero, Rms error corresponds to unregularized ordinary least squares error. As we increase lambda the error slowly decreases (i.e by making trade off between varience and bias). After a certain point due to increasing bias the error increases . So we should choose lambda such that bias and varience are optimum. In our data, lambda between 0.01 and 1 seems to be the right choice.

As we vary the number of clusters ,we are assuming the data is represented by corresponding number of gaussian distributions. From the table we can see that 15 gaussian distributions (M=15) is the correct representation of the dataset.

For gradient descent :

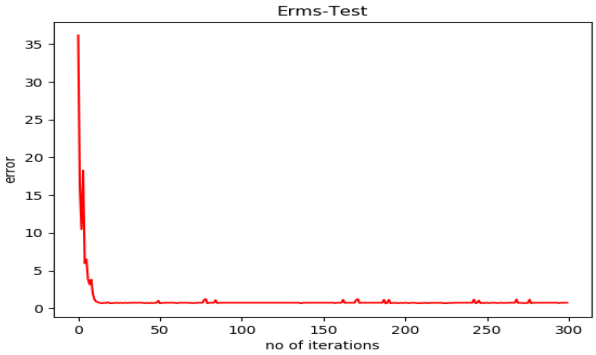
1.For M=10, C_lambda = 0.01,learning rate =0.02, iterations=400

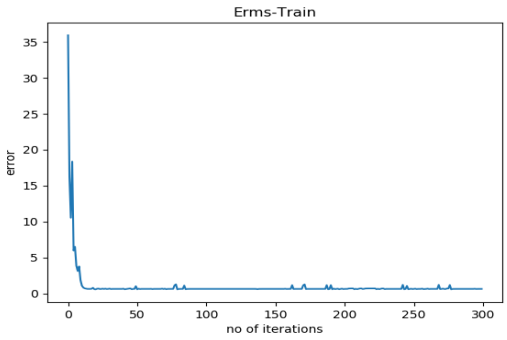




For above parameters the Erms testing = 0.623. Erms training= 0.5526

2.For M=10, C_lambda = 0.01, learning rate=0.06, iterations=300





erms traning :0.57631 erms testing :0.64359

In gradient descent method , as we increase the learning rate , step taken in the direction opposite to gradient can overshoot minimum cross it . so, the error increases as we increase the learning rate