Project 2

Learning to Rank using Linear Regression

CSE474/574: Introduction to Machine Learning (Fall 2016)

Instructor: Sargur N. Srihari

Teaching Assistants: Jun Chu, Junfei Wang and Kyung Won Lee

Sugosh Nagavara Ravindra

sugoshna@buffalo.edu

Person#: 50207357

Table of Contents

Abstract

Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value which is usually discrete. Linear regression is an algorithm under supervised learning in which the input data is linearly related to its corresponding output. In this project we try to use linear regression algorithm to learn the LeToR dataset. Since LeToR is a classification dataset, we can linear regression to output discrete values to the input. We further implement the linear regression model on a Synthetic data in which the data is modeled better

We rate the goodness of fit for the model based on the ERMS value (root mean square error).

Introduction

Linear regression is the most basic and commonly used predictive analysis. Regression estimates are used to describe data and to explain the relationship between one dependent variable and one or more independent variables. At the center of the regression analysis is the task of fitting a single line through a scatter plot. The simplest form with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent, c = constant, b = regression coefficients, and x = independent variable. In this project we have more than one independent variable (LeToR= 46, Synthetic= 10). We make use of Gaussian Radial Basis Function which is a real valued function whose value depends on its distance to the center of the cluster. We have two methods to determine the weight vector, Moore Penrose inverse and Gradient Descent. In this project we implement both the methods and report the ERMS values.

We have various hyper-parameters such as M, λ, μj, Σj, $\eta(\tau)$ which we will tell how to be tuned later on. Our aim in this project is to find the optimal values for hyper-parameters and fit a linear regression model on both LeToR dataset and the synthetic dataset which provides the least value of ERMS as possible.

Data Partition

We have two datasets, LeToR and Synthetic dataset. We partition both the datasets in similar fashion. We first shuffle the dataset and store it separately to normalize it further. We then partition the dataset as follows.

80%= Training Set

10%= Validation Set (Used to train Hyper-parameters)

10%= Test Set (Invisible till the testing stage)

We follow the above partition style as to tune the hyper-parameters for various conditions including over fitting.

LeToR dataset

```
## TRAINING = 80%
X_train=X[0:55698]
Y_train=Y[0:55698];

## VALIDATION = 10%
X_validate=X[55698:62660]
Y_validate=Y[55698:62660];

## TESTING = 10%
X_test=X[62660:69623];
Y_test=Y[62660:69623];
```

Synthetic Dataset

```
## TRAINING = 80%
X_train=X[0:16000]
Y_train=Y[0:16000];

## VALIDATION = 10%
X_validate=X[16000:18000]
Y_validate=Y[16000:18000];

## TESTING = 10%
X_test=X[18000:20000];
Y_test=Y[18000:20000];
```

Linear Regression Model

In this section we look at how to design the linear regression model using various hyper-parameter and in the next section we look at how to tune them.

We first list out all the formulae needed and then go in detail on how to implement them.

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

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

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_n)\}^2 \tag{3}$$

$$\mathbf{w}_{\mathrm{ML}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{t} \tag{4}$$

$$\mathbf{w}^* = (\lambda \mathbf{I} + \boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \mathbf{t} \tag{5}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta \mathbf{w}^{(\tau)} \tag{6}$$

$$\nabla E_D = -(t_n - \mathbf{w}^{(\tau)\top}\boldsymbol{\phi}(\mathbf{x}_n))\boldsymbol{\phi}(\mathbf{x}_n) \tag{7}$$

$$\nabla E_W = \mathbf{w}^{(\tau)} \tag{8}$$

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^*)/N_{\mathrm{V}}} \tag{9}$$

We look at each of the formulae individually.

1) Figure 1 is the linear regression function where **w** is the weight vector and Φ is the design matrix.
2) Formula 2 represents the Gaussian radial basis function where μ is the mean vector and Σ is the covariance matrix representing the spread of radial basis function. Σ is a diagonal matrix with variance as its diagonal values

3) Formula 3 gives us the squared error function which is a measure of error, basically the summation squared of the difference between predicted target value and the actual target value. This value is not normalized to the size of the dataset, hence we make use of ERMS, i.e. Root mean square error.
4) Formula 4 gives us the function for Moore–Penrose pseudoinverse or Closed form solution. We use this to predict the **w** vector which can be used in formula 1 to predict the target value.
5) Formula 5 gives us the regularized function of the closed form solution with a parameter Lambda which is used to avoid over fitting of data on training samples.
6) Formula 6 is the generalized overview of Gradient Descent, which is another method to determine weights for the regression model.
7) Formula 7 and Formula 8 constitute $\mathbf{w}^{t}$ which constitutes the Gradient Descent function. We will look in detail about this during the implementation phase. There are also two types under this, Stochastic Gradient Descent and Batch Gradient Descent.
8) Formula 9 gives us the Root Mean Square error which is the measure of the goodness of our model in this project. This value is normalized and hence preferred over Squared error function.

The glue holding all these functions is the design matrix which is as follows:

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

Instead of using the raw dataset as it is for the input, we first come up with a design matrix using radial basis function and then use these values as input to our model. The design matrix is designed as follows.

We consider the LeToR dataset for this example

- Based on the 'M' value you have taken, divide the training set X into M different batches.
- For each batch select a μ value, either a random sample or mean of the whole batch. (μ_j will be 1x46). We can also perform K-Means clustering to obtain μ values
- Now you will have M such values for μ (μ=Mx46, μ0=1x46, μ1=1x46 ..... μM=1x46)
- Now for PHI design matrix use the above μ values.

- For Φ0(x0) use μ0 and x0 (x0 is the first sample of shape 1x46) so, Φ0(x0)= (x0-μ0) * Σ-1 * (x0- μ0)T === 1x46 * 46x46 *46x1 ==1x1 Scalar Value.
- Similarly Φ1(x0) uses μ1, Φ2(x0) uses μ2... so on till ΦM(x0) uses μM. Now this is for x0. You have do it for N(55698) samples.
- So the final design matrix Φ=NxM == 55698x10

These are the steps to define a design matrix.

Implementation

1) Mean μ: Mean can be either selected in random or we can divide the input into M samples and fine the mean of each batch. Following is the code to do the latter.

```
def mean_x (x):
   k=0;
   z=x[0:int(math.floor((16000/M)*M))];
   a=np.vsplit(z,M);
   mu=np.zeros((M,10))
   sigma=np.zeros((10,10))
   for i in range(M):
      for j in range(10):
         mu[i][j]=np.asarray(a[i]).T[j].mean()
   return mu
```

We can also implement the K-means clustering for better accuracy. The K-means clustering algorithm divides the dataset into M clusters taking into account its Euclidian Distance to each other. Following the code to implement K means clustering

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=M, random_state=0).fit(X_train)
mu = kmeans.cluster_centers_
```

2) Design Matrix Φ: Following is the code to implement the design matrix

```
def phi(X_train,mu,SIGMA):
   PHI=np.zeros((len(X_train),len(mu)));
   for i in range(len(X_train)):
      for j in range(len(mu)):
         PHI[i][j]=phi_calc(X_train[i],mu[j],SIGMA)
   return PHI;

def phi_calc(x_1,mu,SIGMA):
   a=np.subtract(x_1,mu);
   temp=np.dot(a.T,SIGMA)
   a=np.dot(temp,a.T)
   #w/o SIGMA
   #a=np.dot(a.T,a)
   a=a*(-0.5)
   return math.exp(a)
```

3) Weight **w:** Following is the code to calculate weight using Closed form solution

```
def weight_calc(PHI,Y):
    lamda=ld*np.identity(M)
    a=np.dot(PHI.T,PHI);
    a=np.add(lamda,a)
    a_mat=np.asmatrix(a);
    a_mat=a_mat.I;
    rc=np.dot(PHI.T,Y)
    weight=np.dot(a_mat,rc);
    return weight;
```

4) Closed for Solution: Following is the code to implement Closed form solution and output the ERMS values.

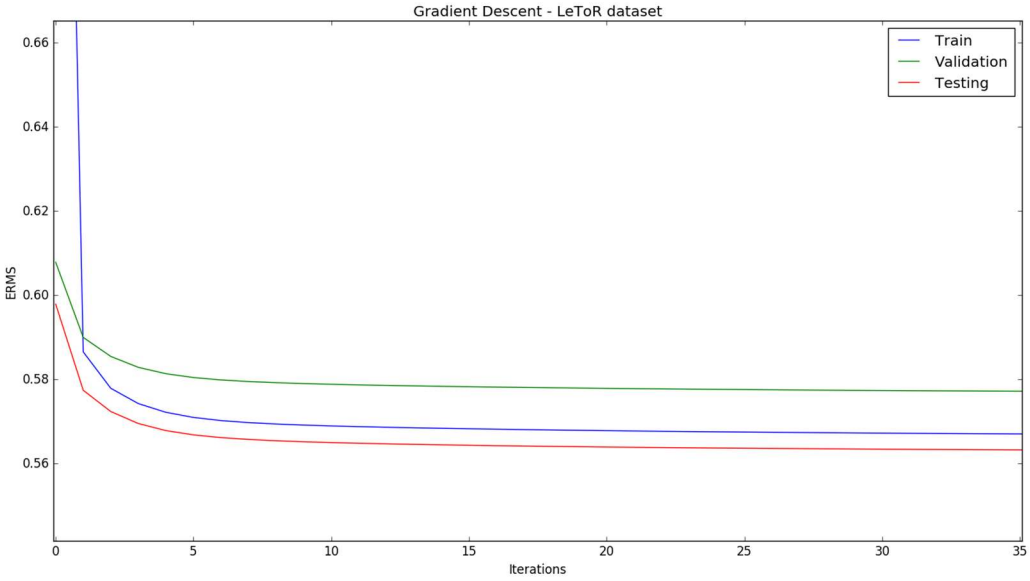
```
def closed_form(weight,PHI_train,Y_train,PHI_validate,Y_validate,PHI_test,Y_test):
    print "CLOSED FORM SOLUTION"
    sse=0;
    for i in range(55690):
        y_hat=float(np.dot(weight,PHI_train[i]));
        sse+=math.pow((y_hat-float(Y_train[i])),2)
    sse/=2;
    print "Train ERMS:"
    print math.sqrt((2*sse)/55690)
    print ""
    print "VALIDATION"
    sse=0;
    for i in range(6962):
        y_hat=float(np.dot(weight,PHI_validate[i]));
        sse+=math.pow((y_hat-float(Y_validate[i])),2)
    sse/=2;
    print "Validate ERMS:"
    print math.sqrt((2*sse)/6962)
    print ""

    print "TEST"
    sse=0;
    count=0;
    for i in range(6962):
        y_hat=float(np.dot(weight,PHI_test[i]));
        sse+=math.pow((y_hat-float(Y_test[i])),2)
    sse/=2;
    print "Test ERMS:"
    print math.sqrt((2*sse)/6962)
```

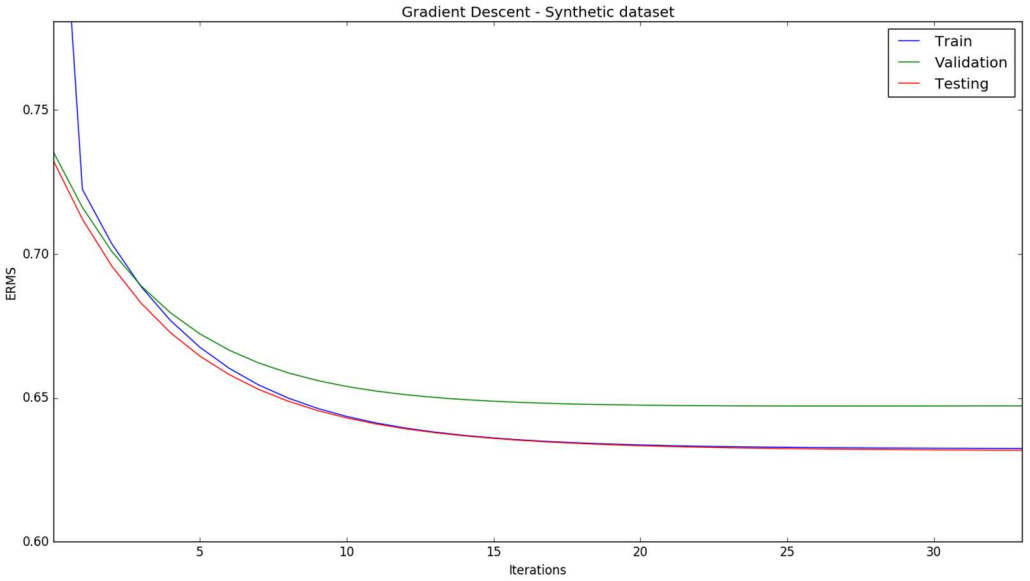
5) Stochastic Gradient Descent: Following is the code to implement Stochastic Gradient Descent.

```
weight=np.ones((1,M))
print "GRADIENT DESCENT"
ED=np.zeros(M);
for iterations in range(MAX_ITER):
   print "\nITERATION %d"%iterations
   sse=0.0;
   count=0;
   for i in range(55698):
      y_hat=float(np.dot(weight,PHI_train[i]));
      alpha=1;
      sse+=math.pow((y_hat-float(Y_train[i])),2)
      if(round(y_hat)==float(Y_train[i])):
         count=count+1
      weight_temp=np.zeros(M)
      if iterations != MAX_ITER:
         for k in range(M):
            ED[k]=(float((y_hat)-float(Y_train[i]))*PHI_train[i][k])
      weight=weight-0.02*ED#+ld*weight)
   sse/=2;
   print "Train ERMS:"
   print math.sqrt((2*sse)/55690)
   print ""

   print "VALIDATION"
   sse=0;
   count=0;
   for i in range(6962):
      y_hat=float(np.dot(weight,PHI_validate[i]));
      sse+=math.pow((y_hat-float(Y_validate[i])),2)
      if(round(float(y_hat))==float(Y_validate[i])):
         count=count+1
   sse/=2;
   print "Validate ERMS:"
   print math.sqrt((2*sse)/6962)

   print "TEST"
   sse=0;
   count=0;
   for i in range(6962):
      y_hat=float(np.dot(weight,PHI_test[i]));
      sse+=math.pow((y_hat-float(Y_test[i])),2)
```

```
    if(round(y_hat)==float(Y_test[i])):
        count=count+1
sse/=2;
print "Test ERMS:"
print math.sqrt((2*sse)/6962)
```

The above implementation codes are for the LeToR dataset and can further be generalized to synthetic dataset by just changing the limit values.





ERMS is converging quick since we are doing Stochastic Gradient Descent. We set our initial weight values as ones and let it converge to minima. No matter how many iterations, the ERMS is at its minima

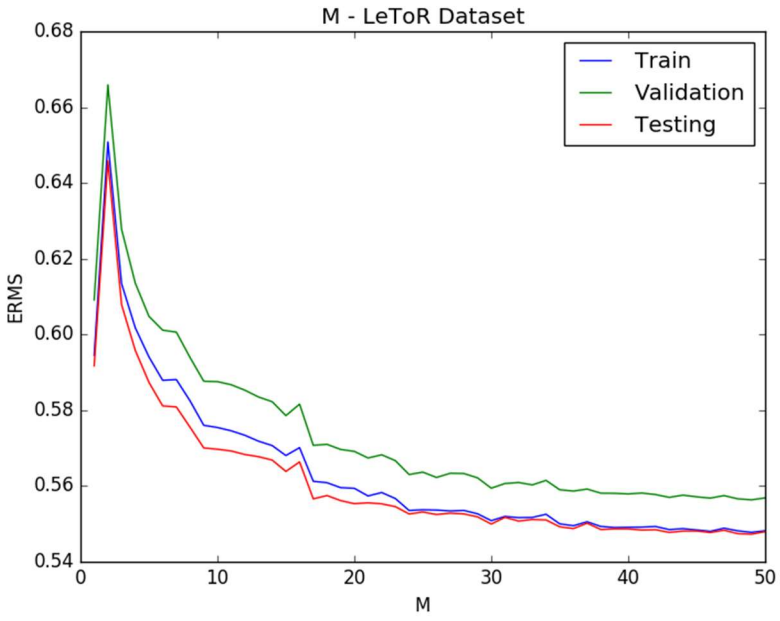
Hyper-Parameters Tuning

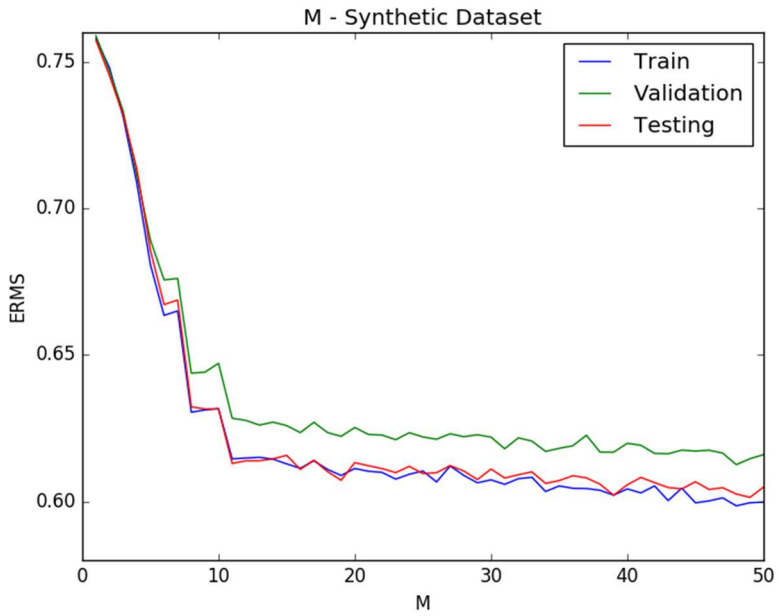
1) M: M represent the number of basis functions in the model. One of the purposes of radial basis function is to reduce the dimensionality of the input data. We can reduce the dataset into M features with the help of design matrix Φ whose dimension is NxM. One way I tuned M value is to brute force all values to M in a range and select the one with the least ERMS value. The code to do that is as follows:

```
for M in range(1,51):
    closed_form(f,weight,PHI_train,Y_train,PHI_validate,Y_validate,PHI_test,Y_test)
```

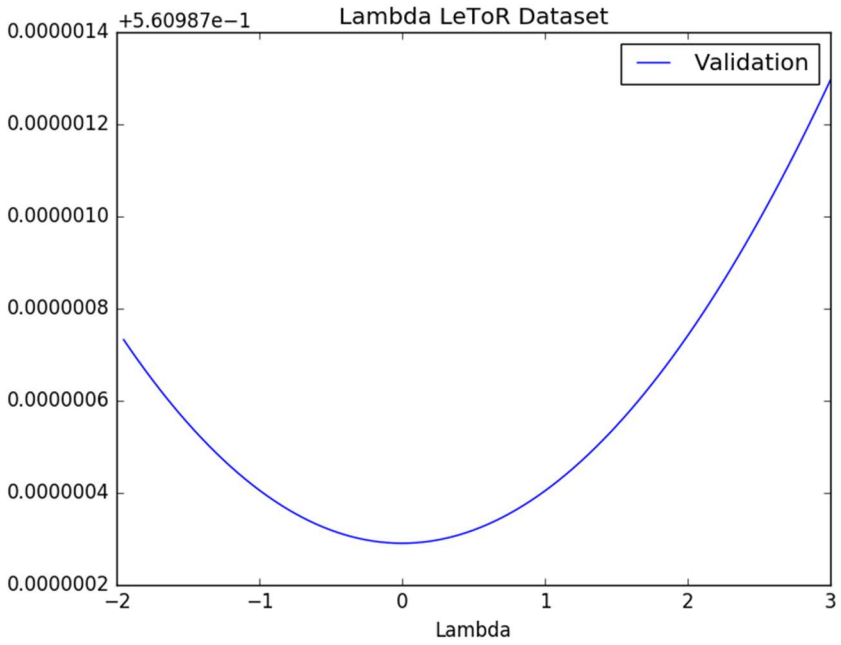
The ERMS values are recorded at each stage of M and the one with the least value of ERMS is selected.

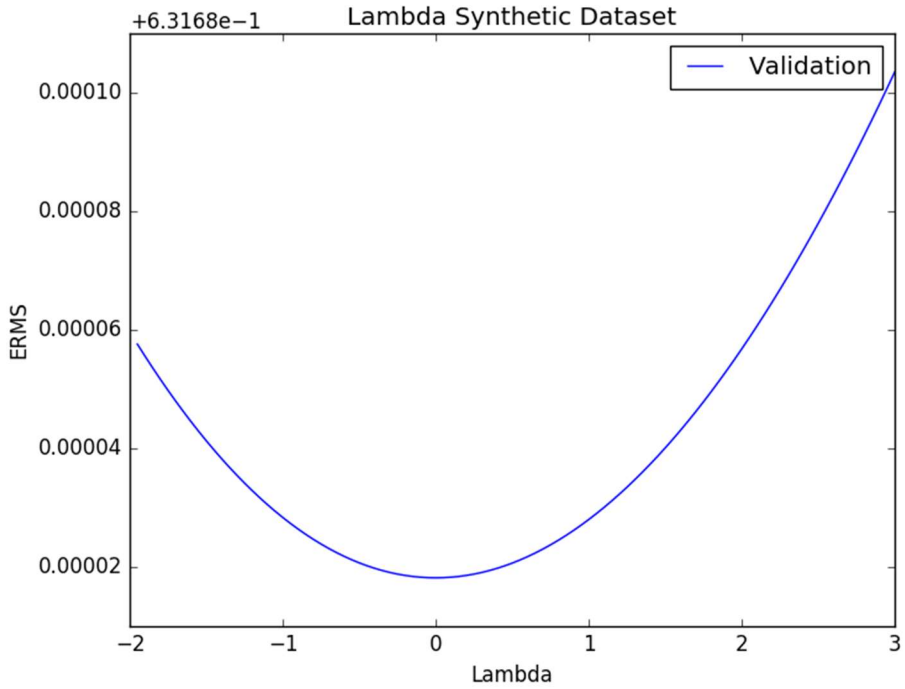
In the case of LeToR it was found that when M=49 we got the optimal value of ERMS, but the others were close. It is also seen that as we increase M the ERMS-Training keeps decreasing gradually. We can take high values of M but it is not feasible to do so as it can be computationally expensive and negates the purpose of radial basis functions. For LeToR we shall take M value of **10**, and synthetic data we keep M as **10**.
LeToR => M=100 ERMS-testing=**0.555421601334**
Synthetic Data=> M=100 ERMS-testing=**0.593471310539,**
M=1000 ERMS-testing=**0.440676940771**





2) λ: The Regularization coefficient λ is to be selected such a way as to avoid over fitting the data. It was observed that assigning values to λ did more harm to the model than any good, it is best to keep the λ value as **0** for both LeToR and Synthetic dataset.





3) μ : The mean μ can be calculated in many ways, few of them are,

- Choosing random
  Code:

```
import random
mu=random.sample(X_train,M);
```

- Creating batches of M and averaging the samples
  Code:

```
def mean_x (x):
   k=0;
   z=x[0:int(math.floor((16000/M)*M))];
   a=np.vsplit(z,M);
   mu=np.zeros((M,10))
   sigma=np.zeros((10,10))
   for i in range(M):
      for j in range(10):
         mu[i][j]=np.asarray(a[i]).T[j].mean()
 return mu
```

- K-Means clustering: Cluster the dataset based on their Euclidian distances
  Code:

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=M, random_state=0).fit(X_train)
mu = kmeans.cluster_centers_
```

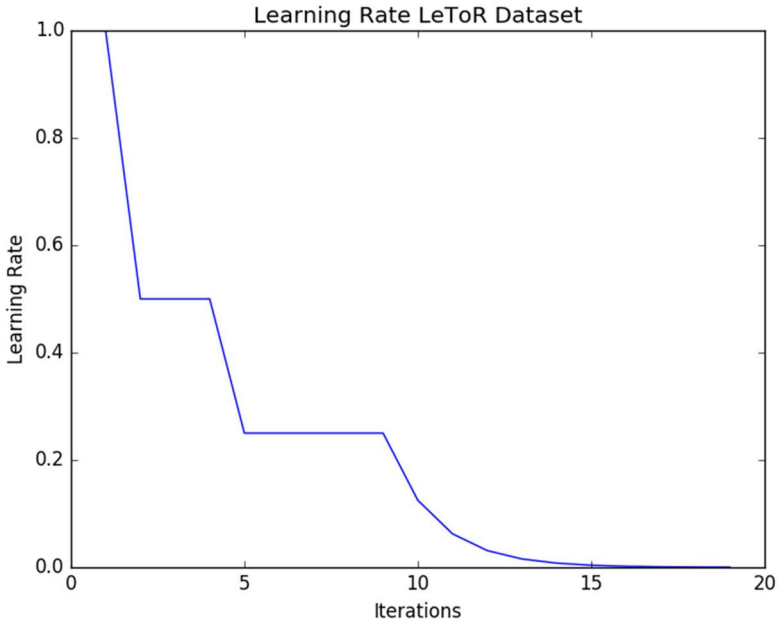
It was found that implementing K-Means Clustering gave the optimal result based on ERMS values and hence we shall be using mu values calculated from K-means clustering

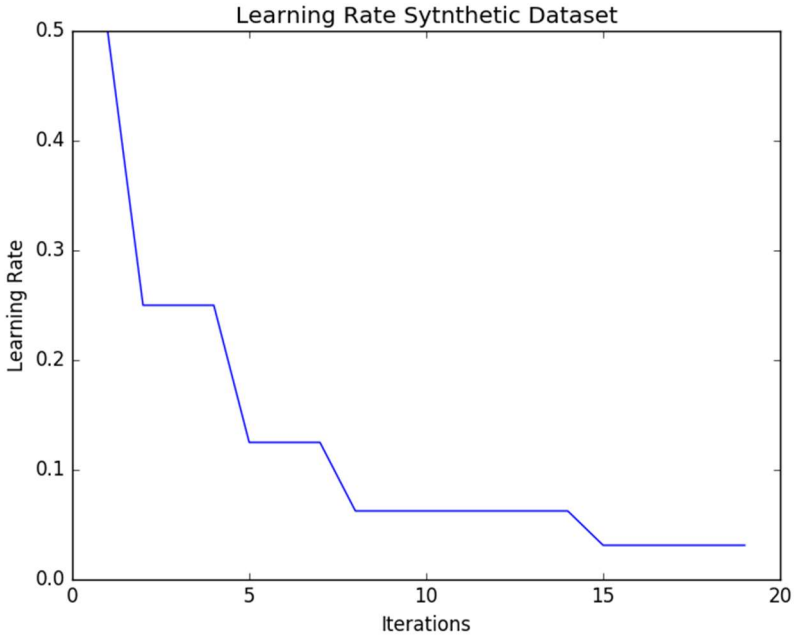
4) Σ : Covariance matrix provides the spread for Gaussian radial basis function. Various different operations were done to select the optimal value for Σ. They are:
    - Find variance of each of the centers from the K-means clustering (vertically) and assign a diagonal matrix.
    - Find variance of each M divided batch and assign a diagonal matrix
    - Find variance of the whole dataset and assign a diagonal matrix
    - Assign an offset of 0.001 in place of 0 to avoid singular matrix
    - Multiply Σ to be scalable
    - Use identity matrix for Σ

    It was also observed that the weight values were too huge if we did not do any manipulation to Covariance matrix, which then shall be highly unfeasible for gradient descent. Of all the above methods, keeping Σ as an identity matrix gave the optimal value for ERMS.

5) η: Eta value represents the learning rate for the Gradient Descent algorithm.
    I implemented both Batch Gradient descent and Stochastic Gradient descent and observed that Stochastic Gradient Descent provided the best result in both feasibility and time complexity.
    Batch Gradient Descent involves summing up the error x inputs for all the sample and subtracting it with the original weight after every iteration.
    Stochastic Gradient Descent involves subtraction of weights after every sample is processed.
    The code provided earlier has both the implementation (Batch Gradient Descent in comments)
    It was observed that η has a great impact on gradient descent and we made sure that if we found any divergence from the minimum ERMS already found, then break the iteration and report the weights and ERMS value. Optimal values are:
    η-LeToR= 0.0001
    η-Synthetic= 0.001

    There are two ways of tuning η:
    - Keeping η constant so the gradient term ∇E_d gradually decreases and we don't need to decrease η for the error to converge to local minima.
    - Another method is to keep η as 1 initially and whenever there is an increase in ERMS value we decrease the η by half.


Learning Rate LeToR Dataset
Learning Rate
Iterations
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20


Learning Rate Sytnthetic Dataset
Learning Rate
Iterations
0.5
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20

Results

LeToR Dataset:

Closed Form Solution
Weights:
[[0.22534689 0.46893694 0.56879095 0.4938407 0.44166351 0.67481558 -0.42749369 0.6643669 0.42540614 -0.6571124]]

ERMS-Train: **0.56626649605**
ERMS-Validation: **0.576488171046**
ERMS-Test: **0.562692111385**

Stochastic Gradient Descent
Weights:
[[0.24815469 0.50460191 0.4890162 0.46836926 0.45151006 0.56694302
-0.38545839 0.70571992 0.30856664 -0.50233488]]

ERMS-Train: **0.566878112155**
ERMS-Validation: **0.576248196706**
ERMS-Test: **0.562674720832**

Synthetic Dataset:

Closed Form Solution
Weights:
[[0.78637378 3.10847465 -2.32084459 -0.70811311 -1.07282424 -0.56594611
1.16715306 -0.39506556 -0.42623686 2.07396753]]

ERMS-Train: **0.631698174236**
ERMS-Validation: **0.647140988969**
ERMS-Test: **0.631581936135**

Stochastic Gradient Descent
Weights:
[[0.78646335 3.10686661 -2.32549113 -0.70789397 -1.07206935 -0.5740586
1.16589558 -0.39578118 -0.43189033 2.07093916]]

ERMS-Train: **0.632287515117**
ERMS-Validation: **0.647453168818**
ERMS-Test: **0.631442623961**

Even though the gradient descent ERMS seems to be close to closed form solution it is gradient descent is preferred over closed form solution since closed form solution is computationally expensive. Gradient Descent converges to minimum fairly quickly compared to closed form solution.

From the above result we can arrive at following conclusions

- The ERMS values for all the three partitions are in close range so we can say that the model is not over fitting the data.
- To back the above point we have taken the regularization term to be **zero** to get the optimal result.
- Further we observe that the learning rate is too small since we are performing stochastic gradient descent which updates the weights after each sample.
- We can obtain lower ERMS values by choosing higher values for M, but it can be computationally expensive and it might over fit the training data.
- Accuracy can be calculated by rounding off the predicted value and comparing with the expected value. ERMS value provide better insight hence they are being reported.