PROJECT 1.2 Learning to Rank using Linear Regression

Krishna Sehgal
50291124

1. OBJECTIVE

The goal of this project is to use machine learning to solve a problem that arises in Information Retrieval, one known as the Learning to Rank (LeToR) problem. We formulate this as a problem of linear regression where we map an input vector x to a real-valued scalar target y(x, w).

There are two tasks:

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).

2. SOFTWARE USED

- Anaconda- Navigator
- Jupyter Notebook
- Python 3.6

3. LIBRARIES

- Scikit-learn
- NumPy
- CSV
- Math
- Matplot

4. DATA SET

In this project we are applying the concept of linear regression on a learning to rank (LeToR) dataset. In the LeToR dataset the input vector is derived from a query-URL pair and the target value is human value assignment about how well the URL corresponds to the query. It uses the Gov2 web page collection (~25M pages) and two query sets from Million Query track1 of TREC 2007 and TREC 2008. In our data-set, the first column represents the relevance label of the row. It takes one of the discrete values 0, 1 or 2. The larger the relevance label, the better is the match between query and document followed by forty six columns representing 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].

5. HYPER PARAMETERS

- Number of Basis Function (M)
- Learning Rate
- Regularizer (λ)

6. TASK

Linear Regression is a kind of supervised learning technique in which training is done on an existing data-set. It is used to find relationship between a dependent variable and one or more independent variables. The training data given to us on learning to rank (LeToR) consist of 46 features and 69,623

rows out of which we will use 80% of our data-set for training, 10% for validation and 10% to test whether model is working correctly or not.

6.1 Train a Linear Regression model on LeToR dataset using a closed-form solution

Neural networks that are used to solve a classification problem makes use of the equation $y=wx+b$ where w stands for weights, x for independent variable and b for bias. In case of Linear Regression, we cannot make use of this equation to predict value on a particular input this is because the function would be increasing and decreasing and is not linear so we can represent such curve with a set of curves known as basis function represented by Φ(x).

Step 1: K - means Clustering
Since we are working on a large data-set containing 55698 rows (training data), it is difficult to apply linear regression directly on such a large data-set so we first perform clustering which is a unsupervised form of learning. In our model we have use K-means clustering where we choose random points as centroids of M number of clusters and after every iteration we improve our cluster set by finding mean repetitively. So we will first reduce our data-set by the process of K-means clustering.

By the process of K-means we will find 10 clusters and will get mean values of these ten clusters that will be used in future to evaluate basis function. Also, within each cluster we will find mean of a particular feature. This way we will be having a 46x10 matrix where the rows represent the total clusters and columns represent mean value of clustered data rows across each feature.

Step 2: Linear Regression Model

Linear Regression model has the form,

$$T=W^T\Phi(x)$$

where T : Target Class
W : Weight Vector
Φ(x) : Basis Function

This equation will be used to predict value of weights since we know the basis function and our target class T having values 0,1,2. To find weights we will use the equation,

$$W=\Phi^{-1}(x)\ T$$

In order to find value of Φ^{-1} we use Moore Penrose pseudo-inverse of the matrix Φ which is show as,

$$\Phi = (\Phi^\top\Phi)^{-1}\Phi^\top$$

The closed form solution can be shown as,

$$W= (\Phi^\top\Phi)^{-1}\Phi^\top\ T$$

In order to perform matrix multiplication in the closed form solution, the terms should have the following dimensions,

Φ: 55k x 10
$\Phi^\top$: 10 x 55k
T: 55k x 1
W= [10 x 55k][55k x 10][10 x 55k][55k x 1] = [10x1]

W that stands for weights will be a 10x1 matrix.

$T= w_1 \Phi_1(x)+ w_2 \Phi_2(x)+ w_3 \Phi_3(x)+.....w_{10}\Phi_{10}(x)$

Φ is the design matrix,

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

In the design matrix Φ, we will have 10 columns representing $\Phi_1 \Phi_2 \Phi_3 \cdots \Phi_{10}$. All the 46 features will be given input to the basis function,

$$(f_1 f_2 f_3 ....f_{46})y \sim (\Phi_1(x) \Phi_2(x) \Phi_3(x) .....\Phi_{10}(x))y$$

The basis function are used to reduce the 46 feature columns into 10 columns that is we are reducing 46 features into a set of 10 basis function so as to reduce our computation. In the design matrix we have 55,690 rows where each data row is given as an input to the basis function i.e

$\Phi_1(x_1) \Phi_2(x_1) \Phi_3(x_1) .....\Phi_{10}(x_1)$
$\Phi_1(x_2) \Phi_2(x_2) \Phi_3(x_2) .....\Phi_{10}(x_2)$
.
.
.
.
.
.
$\Phi_1(x_{55k}) \Phi_2(x_{55k}) \Phi_3(x_{55k}) .....\Phi_{10}(x_{55k})$

We have used a Gaussian Radial Basis Function that is shown by

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

where μ_j = Centroid of each cluster
$\sum^{-1}{}_j$ = Variance - Covariance Matrix

In order to calculate Φ, we have to perform a matrix multiplication that should result into a scale 1x1 value. In order to do so we should have the following dimensions of terms in the gaussian radial basis function,
x: 46x1
μ: 46x1
∑: 46x46

Therefore, Φ: $[46x1]^T[46x46][46x1]=[1x1]$

In the Gaussian Radial Basis function we have used a variance- covariance matrix which is a result of matrix multiplication of the feature square matrix with itself i.e all the 46 features are multiplied with all the other features and feature itself. Since we require co-variance that is the matrix multiplication of a feature with itself so we will make all the other variances as zero thus resulting in a diagonal covariance matrix shown by,

$$\Sigma = \begin{pmatrix} \sigma_1^2 & & & \\ & \sigma_2^2 & & \\ & & \ddots & \\ & & & \sigma_D^2 \end{pmatrix}$$

where σ stands for the spread of each basis function.

Use of Regularizer

We make use of regularizer to avoid the problem of over-fitting that is, if the basis functions try to cover each and every point present in the graph then there might be some redundant values or points due to which our model will learn wrong things and will make wrong predictions based on these.

The closed form solution with an add-on regularizer term to remove redundant terms is given by,

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

where The coefficient λ governs the relative importance of the regularization term. In the given closed form solution all the redundant values will be ignored by the model. Regularization can be motivated as a technique to improve the generalizability of a learned model. Regularization is used in order to prevent the problem of overfitting.

Error - root mean square

Root mean square represents the square root of the second sample moment of the differences between predicted values by our model and observed values or the quadratic mean of these differences. We have calculated the ERMS value after training, validation and testing so as to check whether our model has been improved or not by checking on sets of data.

6.2 Train a linear regression model on the LeToR dataset using stochastic gradient descent

In our project we have used another method to find the weights in the linear regression equation termed as stochastic gradient descent (SGD). In this concept we are updating weights in the equation after every datapoint so as to minimize the error. Minimization of error is done by differentiating the error function so as to minimize it and update it accordingly.

Comparison between Gradient Descent and Stochastic Gradient Descent

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.

- for one or more epochs, or until approx. cost minimum is reached:
  - for training sample i:
    - for each weight j
      - $w_j := w + \Delta w_j$, where: $\Delta w_j = \eta(\text{target}^{(i)} - \text{output}^{(i)})x_j^{(i)}$

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

Learning rate
Learning Rate refers to the amount of time taken by our algorithm to learn. If we reduce value of learning rate, classifier takes longer time to learn but with an improved accuracy.

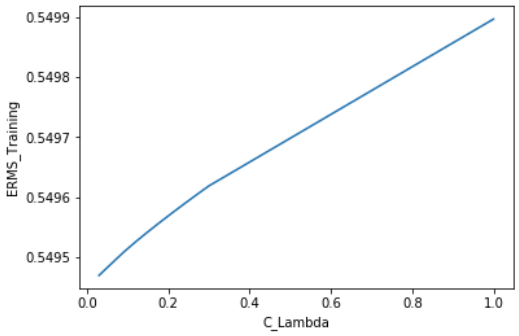
7. GRAPHICAL REPRESENTATION

7.1 Lambda vs ERMS_Training

Hyper-parameters [Number of Clusters=10,Lambda=Variable]

In [11]:
```
import matplotlib.pyplot as plt
plt.plot([0.03,0.09,0.12,0.15,0.18,0.21,0.24,0.27,0.3,1],[0.549469407,0.5495078,0.549525308,0.549541913,0.549557954
plt.ylabel('ERMS_Training')
plt.xlabel('C_Lambda')
plt.show()


```

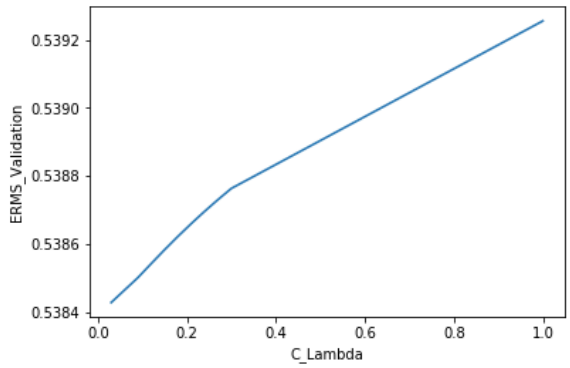


7.2 Lambda vs ERMS_Validation

Hyper-parameters [Number of Clusters=10,Lambda=Variable]

In [19]:

```
import matplotlib.pyplot as plt
plt.plot([0.03,0.09,0.12,0.15,0.18,0.21,0.24,0.27,0.3,1],[0.538428174,0.538501237,0.538542583,0.53858324,0.53862253
plt.ylabel('ERMS_Validation')
plt.xlabel('C_Lambda')
plt.show()
```

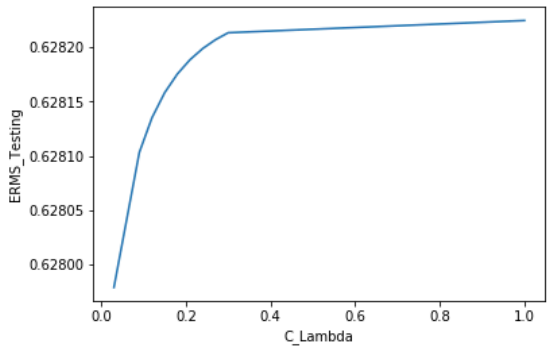


7.3 Lambda vs ERMS_Testing

Hyper-parameters [Number of Clusters=10,Lambda=Variable]

In [20]:

```
import matplotlib.pyplot as plt
plt.plot([0.03,0.09,0.12,0.15,0.18,0.21,0.24,0.27,0.3,1],[0.627978845,0.62810316,0.628135246,0.6281583,0.628175514,
plt.ylabel('ERMS_Testing')
plt.xlabel('C_Lambda')
plt.show()
```

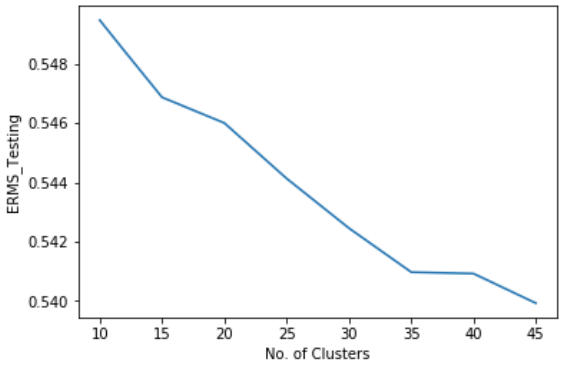


7.4 Number of Clusters vs ERMS_Training

Hyper-parameters [Number of Clusters=Variable,Lambda=0.03]

In [21]:

```
import matplotlib.pyplot as plt
plt.plot([10,12,15,20,25,30,35,40,45],[0.549469407,0.548420577,0.546868421,0.546001727,0.544131428,0.54245746,0.540
plt.ylabel('ERMS_Training')
plt.xlabel('No. of Clusters')
plt.show()
```

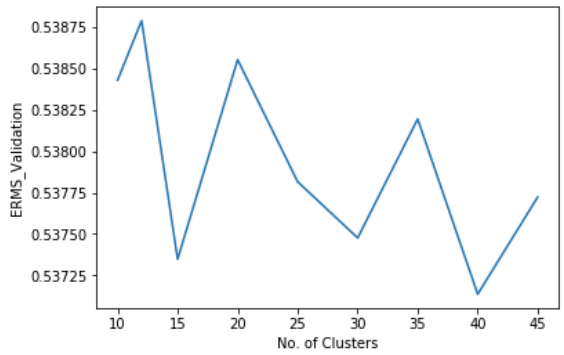


7.5 Number of Clusters vs ERMS_Validation

Hyper-parameters [Number of Clusters=Variable,Lambda=0.03]

In [23]:

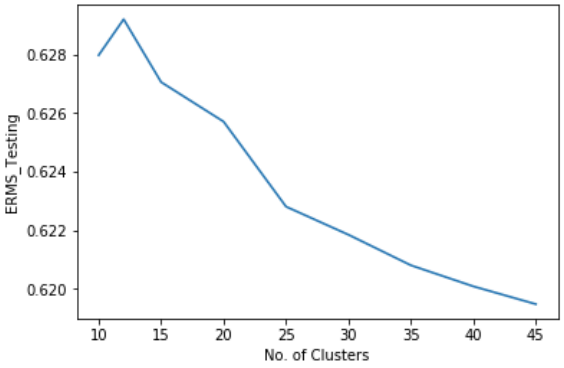
```
import matplotlib.pyplot as plt
plt.plot([10,12,15,20,25,30,35,40,45],[0.538428174,0.538786889,0.537347952,0.538551279,0.537815366,0.537476563,0.53
plt.ylabel('ERMS_Validation')
plt.xlabel('No. of Clusters')
plt.show()
```



7.6 Number of Clusters vs ERMS_Testing

Hyper-parameters [Number of Clusters=Variable,Lambda=0.03]

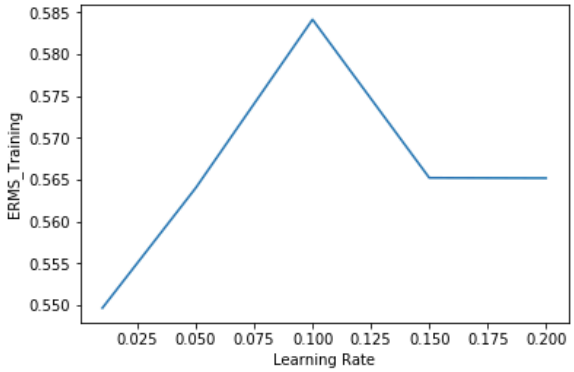
In [24]:
```
import matplotlib.pyplot as plt
plt.plot([10,12,15,20,25,30,35,40,45],[0.627978845,0.629206555,0.627057998,0.625708085,0.622803384,0.621842466,0.62
plt.ylabel('ERMS_Testing')
plt.xlabel('No. of Clusters')
plt.show()
```



7.7 Learning Rate vs ERMS_Training

Hyper-parameters [Number of Clusters=10,Learning Rate= Variable]

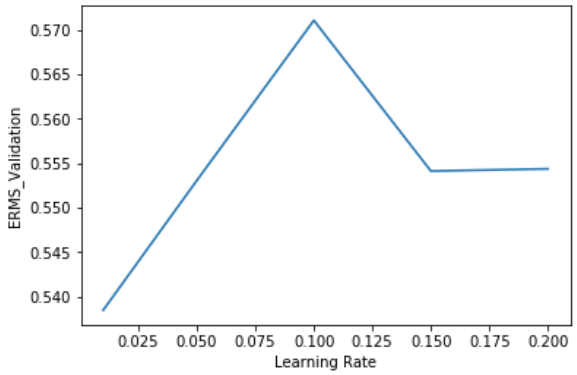
In [25]:
```
import matplotlib.pyplot as plt
plt.plot([0.01,0.05,0.1,0.15, 0.20],[0.54964, 0.56402, 0.58417, 0.56522, 0.56518])
plt.xlabel('Learning Rate')
plt.ylabel('ERMS_Training')
plt.show()
```



7.8 Learning Rate vs ERMS_Validation

Hyper-parameters [Number of Clusters=10,Learning Rate=Variable]

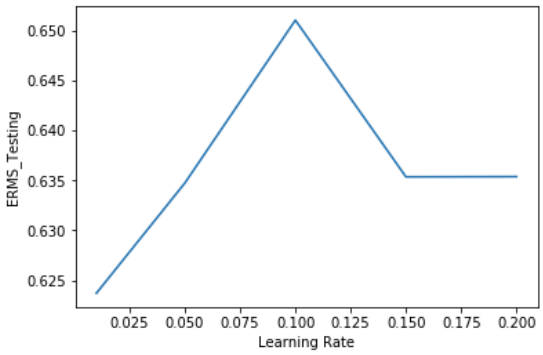
In [26]:
```
import matplotlib.pyplot as plt
plt.plot([0.01,0.05,0.1,0.15, 0.20],[0.53846, 0.55304, 0.57105, 0.5541, 0.55435])
plt.xlabel('Learning Rate')
plt.ylabel('ERMS_Validation')
plt.show()
```



7.9 Learning Rate vs ERMS_Testing

Hyper-parameters [Number of Clusters=10,Learning Rate=Variable]

In [27]:
```
import matplotlib.pyplot as plt
plt.plot([0.01,0.05,0.1,0.15, 0.20],[0.62372, 0.63473, 0.65101, 0.63535, 0.63537])
plt.xlabel('Learning Rate')
plt.ylabel('ERMS_Testing')
plt.show()
```



8. RESULT

The Accuracy and Erms values are as follows :

8.1 Closed Form Solution

```
UBITname      = Krishna Sehgal
Person Number = 50291124
-------------------------------------------------
-----------------LeToR Data----------------------
-------------------------------------------------
-------Closed Form with Radial Basis Function-------
-------------------------------------------------
M = 10
Lambda = 0.9
E_rms Training   = 0.5494694067137939
E_rms Validation = 0.5384281741394163
E_rms Testing    = 0.6279788453832134
Accuracy Training   = 73.92233253738846
Accuracy Validation = 74.6768170066073
Accuracy Testing    = 69.87501795719005
----------Gradient Descent Solution--------------------
E_rms Training   = 0.56518
E_rms Validation = 0.55435
E_rms Testing    = 0.63537
[0.54964, 0.56402, 0.58417, 0.56522, 0.56518]
[0.53846, 0.55304, 0.57105, 0.5541, 0.55435]
[0.62372, 0.63473, 0.65101, 0.63535, 0.63537]
```

9. CONCLUSION

We have compared the ERMS values on training, validation and testing dataset. The Closed form solution gives comparatively low root mean square error value as compared to gradient descent solution.