Introduction

This projects targets the Learning to Rank (LeToR) problem where we map an input vector x to a real-valued scalar target y(x, w). x is the features derived from a query-document pair and y is the relevance label. We use two different approaches to solve learn the weights, w:

1. Closed form solution
2. Stochastic Gradient Descent.

Description of the Regression Model Used:

Our regression model looks is as follows:
$y(x, w) = w^T \phi(x)$,
Where $\phi(x)$ is the basis function, in our case we have used the radial basis function.

Why use radial basis function?

Since, our data is complex and the features hold a non-linear relation with the target variable, we need a model that is non-linear in **x,** so that we can fit a non-linear curve to our data, which will in turn reduce the cost function(error) and hence, potentially increase the accuracy. However, we still call it a linear model because its linear in **w.**

Gaussian radial basis function:

The Gaussian Radial basis function is defined as:
$\phi_j(x) = \exp\ (-1/2\ (x - \mu_j)^T \Sigma^{-1}{}_j\ (x - \mu_j))$

Here, μ_j is the mean of the cluster j, as we first divide our training data set into k clusters, using k-means. The value of this k is a hyper parameter, which we will tune during this project.

The gaussian radial basis function actually computes the similarity of each data point with each of the cluster centroids (mean of each cluster). This similarity measure is basically given by the euclidean distance between the cluster centroid and the data point. Since Euclidean distance is in the power of 2, this gives us a higher degree (degree =2) non-linear curve to fit our data and hence, enables us to model a complex data set.

Hence, we use the **design matrix** $\phi_j(x)$, instead of simple feature vectors, in our model.

However, our model is linear with respect to **w,** the weights.

Implementational Objective for our model:

While fitting the curve to our data, our main objective is to minimize the cost which is defined by the total error made by our model. This total error or cost of our model is defined by Root-Mean-Square Error (RMS). To prevent overfitting, we add the regularization term as well and hence our overall cost function becomes:

$E(w) = E_D(w) + \lambda E_W(w)$
Where, λ is our regularization parameter which we will be tuning for optimal value.

To train the weights, we have implemented two different approaches:

1. **Closed Form Solution**:

The closed form solution with regularization gives the optimal weights using the Moore-Penrose pseudo-inverse of the matrix ϕ as :

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

Where, t is the target label (target value) for data point **x** and λ is the regularization parameter.
Once we compute the RHS of the above equation, our **w** gets populated with most optimum weights.

2. **Gradient Descent Solution:**

Starting with the random weights, $w^{(0)}$, we iteratively update the weights as :

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

where $\mathbf{w^{(\tau+1)}}$ is the weight matrix for τ+1 th iteration of the update rule.

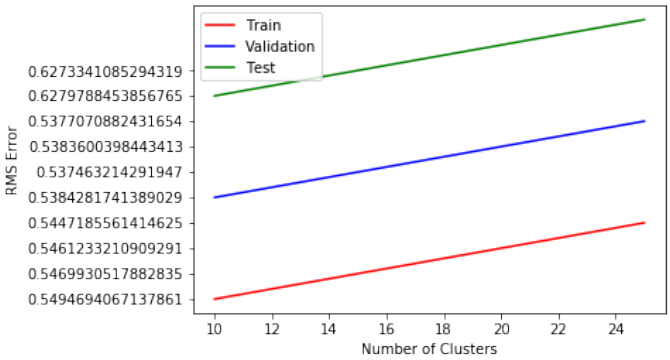
$\Delta w(\tau) = -\eta(\tau)\nabla E$. Here η is the learning rate, which is another hyperparameter that decides how fast the model learns or how aggressively we update the weights in each iteration.

After repeating the update rule for sufficient number of times, we get the most optimal values for weights, which can then be used to make predictions on unseen data.

$\nabla E = \nabla E_D + \lambda \nabla E_W$,
where $\nabla E_D = -(t_n - w^{(\tau)T} \phi(x_n))\phi(x_n)$, which is the differential for cost function without regularization term.
$\nabla E_W = w^{(\tau)}$ is the differential for regularization term.

Tuning the hyper parameters:

Tuning Number of Clusters for Closed Form Solution:
Here as we change the number of clusters, the number of radial basis functions also changes, thereby generation different number of features and hence different design matrices. As we can see from the graph, the RMS error increases as number of clusters increases.

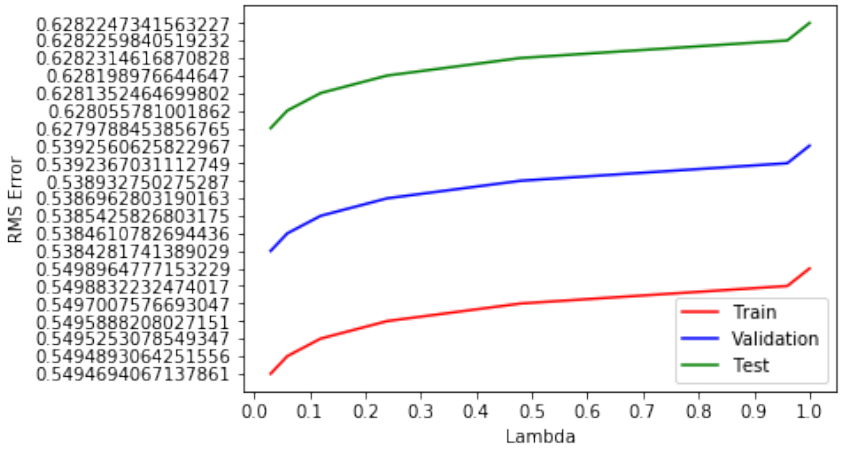


Hence, we settle down for 10 clusters only for the optimal performance.

Tuning the regularization parameter (Lambda) for Closed Form Solution:

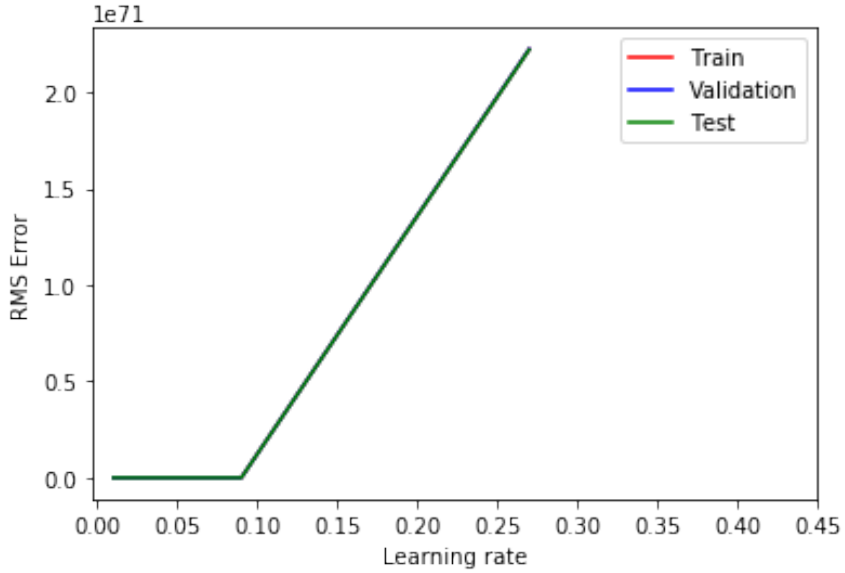
Here as we change the Lambda by increasing it by the multiples of 2, starting from 0.01 and ending at 1. As we can see from the graph, the RMS error increases as lambda increases. However, the Test accuracy is best for Lambda =0.03. So we choose 0.03, as

our most optimal Lambda value.



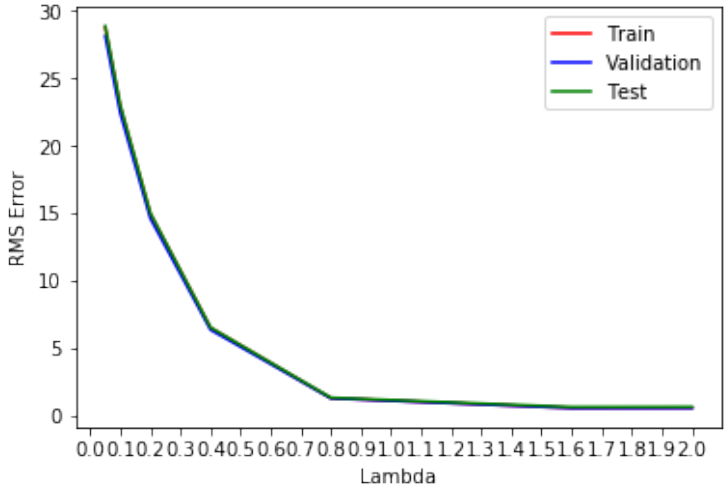
Tuning the Learning Rate for Stochastic Gradient Descent:

Here we change the Learning Rate parameter from 0.01 to 0.27 by increasing it by multiples of 3. As we can see from the graph, the RMS error initially remains almost the same till neta=0.09, and then increases steeply as lambda increases. We choose 0.01 as our optimal value as it gives the best accuracy on testing data.

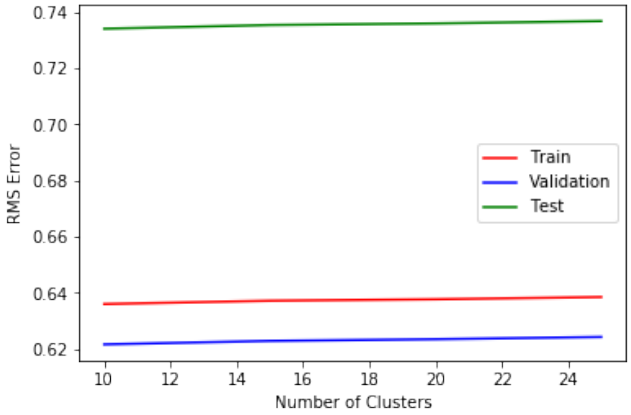


Tuning the Lambda(Regularization Parameter) for Stochastic Gradient Descent Solution:

Here we change the Lambda from 0.05 to 2 by increasing it by multiples of 2. As we can see from the graph, the RMS error decreases as Lambda increases and stagnates at lambda =2. Hence we choose Lambda =2 as our optimal value.



Tuning Number of Clusters for Stochastic Gradient Descent:
Here as we change the number of clusters, the number of radial basis functions also changes, thereby generation different number of features and hence different design matrices. As we can see from the graph, the RMS error increases as number of clusters increases.



Hence, we settle down for 10 clusters only for the optimal performance.

For all the above stochastic gradient descent solutions, we have used 400 iterations to train the weights.

Conclusion:

After experimenting with hyper parameters and testing performance with both closed form solution and gradient descent solution on test data, **we find stochastic gradient**

descent solution with Lambda = 2, Clusters =10, and Learning Rate = 0.01 as the most accurate model.