CSE 574 – INTRODUCTION TO MACHINE LEARNING – PROJECT 1.2

Submitted By: Keerthana Baskaran

UB# 50288944

Contents:

1. Introduction

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 use linear regression solution for this problem by taking the input vector x to a real value scalar target y(x;w). The target values are scalar values that take on of the three values (0,1,2). We use linear regression to obtain real values which helps in ranking. We solve this linear regression problem by two methods,

1. Linear regression using closed form solution
2. Linear regression using stochastic gradient descent (SGD) solution

Given Dataset

In this project we use the QuerylevelNorm version of LETOR dataset provided by microsoft. The dataset has 69923 rows of 46 features.

2. Implementation

1. We use two approaches, first we do the closed form solution and the first step in closed form solution is to define the number of basis function used in solving the problem. The data usually are not linear and cannot be modeled using linear function. So, we use **Radial Basis function** to introduce a non-linearity component in the data. We create multiple Gaussian basis functions and then fit a linear model for each of these functions. The

hyperparameters here includes the number of basis function, the µ value using K-means clustering and the BigSigma (Σ) which is spread of gaussian function.

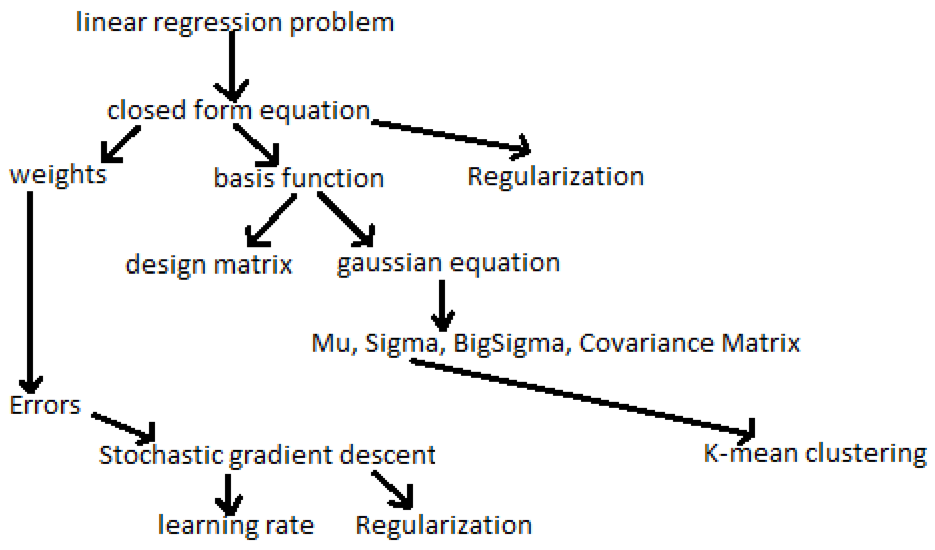
2. The next step is to **split the dataset** into training, validation and testing sets. Training set will have 80% = 55700 samples, validation set will have 10%=6961 samples, test set will have 10%=6961 samples. Training set is used to learn the weights and parameters. The validation set is used to validate the learned weights and test set is used to find how the learned weights generalizes.

3. The next step is to formulate the **design matrix** using gaussian normal distribution. We have to choose the center for basis function. Here we use M basis functions instead of one polynomial function by dividing the feature space into M regions and applying M basis functions on input vectors. We use Gaussian equation to find center by diving the space into M clusters. These clusters are found using the **k-means algorithm**. The centroid or the mean of one cluster is taken as the center for one basis function. That way we have M centroids forming the centers of these basis functions.

4. We then find the **weights** by using the closed form likelihood solution and compute the model complexity and lambda is additional regularization term used. The error (**Erms**) is found by calculating the difference between the predicted value and the actual value. The goal is to minimize this error. We use **stochastic gradient descent** solution to minimize this error by using learning rate.

3. Conceptual understanding

3.1 Terms and concepts used in solving linear regression



Linear Regression model using closed form solution

The equation for linear regression is

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

This equation is rewritten using **Moore's pseudo inverse** term as,

$((\Phi\}^{T} \Phi)^{-1} \Phi^{T} y(x,w)$

Where W is weight vector to be learnt from training samples
Phi is vector of M basis functions and X is input vector
From the above equation we find the equation for weights that is given by,

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

In our project we use **gaussian function** which is of form,

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

Where **Mu** is center for basis function and **sigma** is the spread of basis function.
First, we try to formulate the design matrix. The **dimension** of design matrix is number of training data x number of clusters. Which is 55699x10 in our case.

Each element in the **Design matrix** Φ(X) is calculated by the following steps. In the formula of gaussian equation, the **X** is an input vector of dimension 1x41, **Mu** is the mean of basis function that is 41x1. Sigma is represented by **BigSigma** which is a covariance matrix.
A **covariance matrix** is a diagonal matrix with the diagonals having sigma value so that while multiplying the matrix with the other gaussian equation terms the sigma square value is obtained. The dimension of covariance matrix is 41x41. The gaussian equation gives a **scalar** value after the multiplication. This accounts to one term of the design matrix. The entire matrix is formulated by this method.

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2}(\underset{1x41}{\mathbf{x}} - \underset{41x1}{\boldsymbol{\mu}_j})^{\top} \underset{41x41}{\Sigma_j^{-1}} (\underset{1x41}{\mathbf{x}} - \underset{41x1}{\boldsymbol{\mu}_j})\right) \Rightarrow \text{scalar value}$$

Basis function

Every continuous function can be represented as **combination** of basis function. It is used to find similarity in the function. Depending on the number of basis function we need, the data set is divided into clusters using **K-Mean clustering**. The Mu and sigma value vectors are returned to the design matrix and gaussian normal distribution equation for calculation.

K-Mean clustering

When data points are given, divide the points randomly depending on the number of **clusters** needed. Then arbitrarily choose center points and the computer distance with each point and find the middle. Depending on the average move the **center points** of each cluster. Now again compute distance of all points with the center. Do the same process again until all points in one cluster are near to the center. These points are the Mu value and the 41 features of data point together form a vector of dimension 41x1. This is returned to the gaussian equation.

Root Mean square error

Now we get the design matrix and we find the weight value using the closed form equation. The dimension of weight vector after multiplication is 10x1. The next step is to find the error rate. This is calculated by removing the **predicted value** from the **actual target value**. The resulting value is the error. The equation is given by,

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

Gradient descent solution

For any machine learning algorithm, we come up with a cost function and use gradient descent to **minimize the cost function**. Gradient Descent works by starting with random values for each coefficient (with an initial set of parameter values and iteratively moves toward a set of parameter values that minimize the function). The **sum of the squared errors** is calculated for each pair of input and output values. A **learning rate** is used as a scale factor and the coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved. We must select a learning rate (alpha) parameter that determines the size of the improvement step to take on each iteration of the procedure. This iterative minimization is achieved using calculus, taking steps in the **negative direction** of the function gradient.

There are 3 types of gradient descent.

1. **Batch gradient descent** calculates the error for each example within the training dataset, but only after all training examples have been evaluated, the model gets updated. Advantages is that it produces a stable error gradient and a stable convergence. The disadvantage is that the stable error gradient can sometimes result in a state of convergence that isn't the best the model can achieve.

2. **Stochastic gradient descent** (SGD) calculates the error for **each example** within the training dataset. This means that it updates the parameters for each training example, one by one. Sometimes the frequency of the updates can also result in noisy gradients, which may cause the error rate to jump around, instead of slowly decreasing. Here the **weights** are **randomly initialized** and then iteratively update the weights.

As the algorithm goes through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm **converges**. If this is done, the data can be shuffled for each pass to prevent cycles. Typical implementations may use an adaptive **learning rate** so that the algorithm converges.

The equation is given as,

$$w := w - \eta \nabla Q(w)$$

It is also written as

$$= w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

Choose an initial vector of parameter w and learning rate n. Repeat until an approximate minimum is obtained.

3. **Mini-batch Gradient Descent** simply splits the training dataset into small batches and performs an update for each of these batches. Therefore, it creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. Common mini-batch sizes range between 50 and 256.

Learning Rate

In order for Gradient Descent to reach the local minimum, we have to set the learning rate to an appropriate value, which is **neither too low nor too high**.
This is because if the steps it takes are too big, it maybe will not reach the local minimum because it just bounces back and forth. If you set the learning rate to a very small value, gradient descent will eventually reach the **local minimum,** but it will maybe take too much time

How to find if gradient descent works fine?

A good way to make sure that Gradient Descent runs properly is by plotting the cost function as Gradient Descent runs. When we plot the number of iterations in x-axes and the value of cost function in Y-axes then the cost function should decrease after every iteration.

Regularization

This technique discourages learning a more complex or flexible model, to **avoid** the risk of **overfitting**. This will adjust the coefficients based on your training data. If there is noise in the training data, then the estimated coefficients won't generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.

The equation is given by,
E(w) = ED(w) + Λ EW(w)

4. Tuning the model

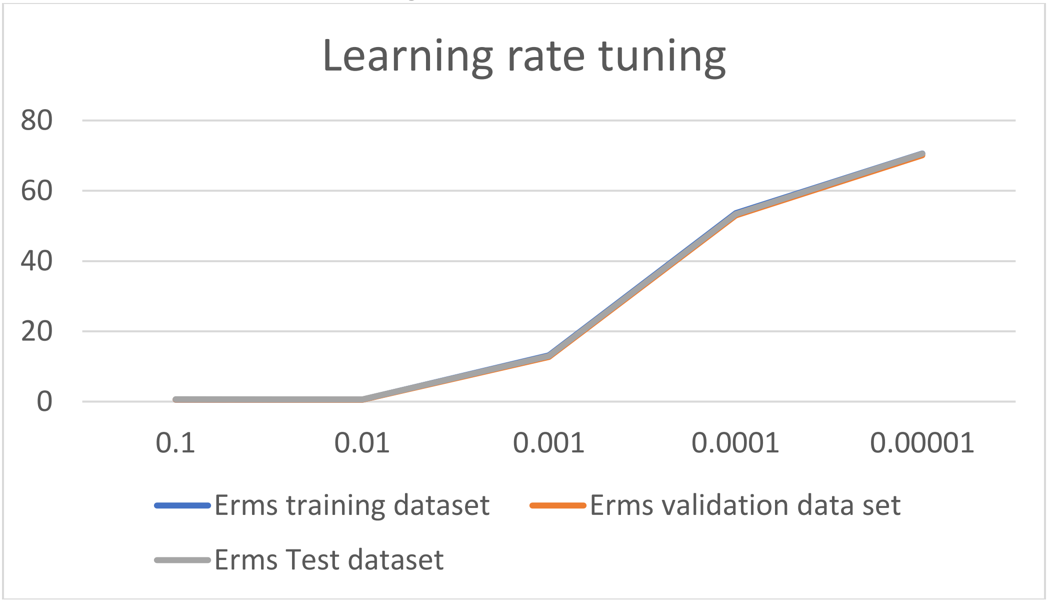
Available Hyper Parameters

1. Lambda (Regularization term)
2. Eta (Learning rate)
3. M (Number of Basis function)

Changing the learning rate

Learning rate	Erms training dataset (SGD)	Erms Validation dataset SGD)	Erms Test dataset (SGD)
0.1	0.56	0.55	0.63
0.01	0.56	0.54	0.62
0.001	13.12	12.7	13.00
0.0001	53.6	53.0	53.4
0.00001	70.6	70.1	70.55
0.2	5.7	5.5	5.6
0.3	12.3	11.9	12.2

In order for the gradient descent to work we must set the learning rate to an **appropriate value**. This parameter determines how fast or slow we will move towards the optimal weights. So, using good learning rate is crucial one way to resolve this problem is to **divide** the learning rate value by N where N is the size of **training data**.



Observation

- From the table, as the learning rate is set from **0.1 to 0.01** there is no much change in the error or accuracy. As we go high or low there is drastic change in the accuracy. This is because the model started overfitting the data or is not learning accurately.

Inference

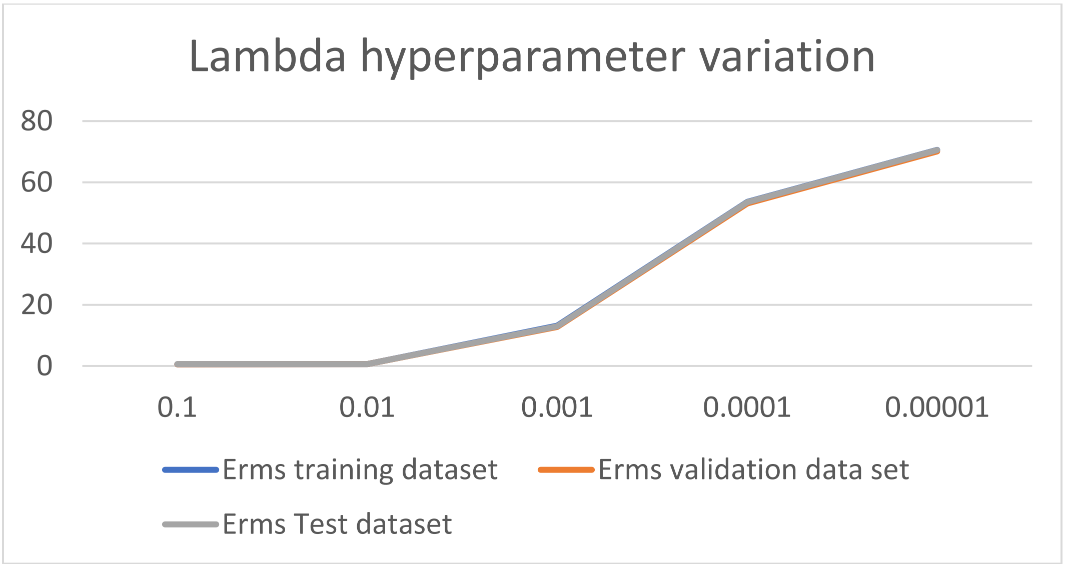
- If the learning rate is **too small**, the optimization will need to be run a lot of times (taking a long time and potentially never reaching the optimum).
- If the learning rate is **too big** then the optimization may be unstable (bouncing around the optimum, and maybe even getting worse rather than better).

Changing Lambda for closed form solution

Lambda	Erms Training	Erms Validation	Erms Test
0.001	0.54	0.53	0.62
0.0001	0.54	0.53	0.62
0.03	0.54	0.53	0.62

Changing Lambda for SGD

Lambda	Erms Training	Erms Validation	Erms Test
0.01	0.58	0.57	0.64
0.001	13.12	12.7	13.00
0.0001	53.6	53.08	53.5



Observation

As the learning rate is decreased the **error function increases**. This proves that learning rate must be set optimum. We should try plotting the learning rate for different values and when the graph seems increasing then should stop experimenting.

Inference

It is always recommended to take **random number** mostly 0 as Lambda for a subsample of data and look at the variation in the estimation. Then repeat the process for a slightly larger value of lambda to see how it affects the variability of your estimate. Whatever value of lambda we decide is appropriate for your subsampled data, we can use a smaller value to achieve comparable regularization on the full data set.

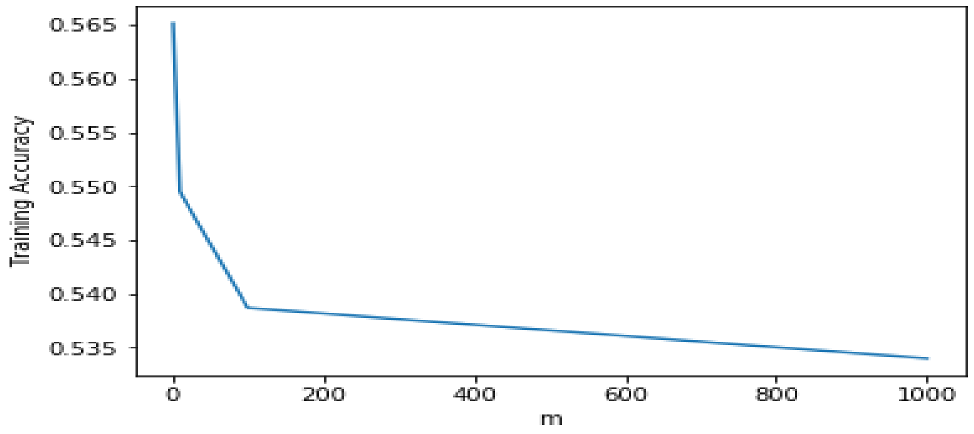
Changing the number of basis function

	Closed form solution	SGD

Number of Basis function	Erms Training	Erms Validation	Erms Testing	Erms Trainng	Erms Validation	Erms Testing
1	0.56	0.55	0.64	0.56	0.55	0.63
10	0.53	0.53	0.62	0.56	0.54	0.62
100	0.53	0.53	0.61	0.53	0.53	0.61
1000	0.54	0.53	0.62	0.54	0.53	0.62
10000	0.54	0.53	0.62	0.54	0.53	0.62

Observation

As the number of **basis function increases** the **error decreases**, this is because the higher the number of basis function the lower goes the variance value and so the error goes low. In our given dataset there is a huge difference in the variance of each data point. If we try using each data point as a separate cluster by giving that many basis functions, then we would get even higher accuracy.



Inference

Tuning the basis function to certain extent will not affect the model much but once we use large number of bases we will start overfitting the data. As the number of basis function increases more the variance decreases and so the error rate also decreases

What will happen if we use single standard deviation to all our bases. We will get a smooth curve in this case. This is because the gaussian that make up the reconstruction all have the same standard deviation.

5. Conclusion

By implementing two methods we find that in closed form solution we calculate the design matrix mxn for each value of m where m is the number of training samples and it is constant for all features. In stochastic gradient descent we assume the weights, results in a much faster implementation of the gradient descent approach and will be much **faster than closed form solution approach**.

One analysis done in the project is that the **dataset is scattered enough** and so 1 basis function and considerable number of basis function say 10000 gives the **same error rate**. If we try keeping all the datapoints as separate cluster then the model will learn more.

Our results are not too great when we **increase the number of basis function** to very few value. But if we use a larger number of basis then the accuracy will increase, and error rate will go low. Using a very large number of basis then we start overfitting. The **learning rate affects the error** and accuracy significantly, hence choosing optimal once is important. If we use single standard deviation for all our basis then the plot is much **smoother**. This is because Gaussian that make up the reconstruction all have the same standard deviation.

References

1. https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Iterative_method
2. https://www.hackerearth.com/blog/machine-learning/gradient-descent-algorithm-linear-regression/
3. https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd
4. https://medium.com/@lachlanmiller_52885/understanding-and-calculating-the-cost-function-for-linear-regression-39b8a3519fcb
5. https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10
6. https://github.com/ravioactive/LETOR
7. https://pythonmachinelearning.pro/using-neural-networks-for-regression-radial-basis-function-networks/
8. https://github.com/nandakishorek/LETOR
9. https://stackoverflow.com/questions/12182063/how-to-calculate-the-regularization-parameter-in-linear-regression/12182410
10. https://en.wikipedia.org/wiki/Stochastic_gradient_descent