Project-2

Learning to Rank
Using
Liner Regression

Name: Armaan Goyal
UBIT Name: armaango
Person Number: 50170093

Linear Regression

The goal of regression is to predict the value of one or more continuous *target* variables t given the value of a D-dimensional vector x of *input* variables. [1] The simplest form of linear regression models is also linear functions of the input variables. However, we can obtain a much more useful class of functions by taking linear combinations of a fixed set of nonlinear functions of the input variables, known as *basis functions*. Such models are linear functions of the parameter, which gives them simple analytical properties, and yet can be nonlinear with respect to the input variables. [2]

Given a training data set comprising N observations $\{x_n\}$, where n = 1, . . . N, together with corresponding target values $\{t_n\}$, the goal is to predict the value of t for a new value of x. In the simplest approach, this can be done by directly constructing an appropriate function y(x) whose values for new inputs x constitute the predictions for the corresponding values of t. More generally, from a probabilistic perspective, we aim to model the predictive distribution p(t|x) because this expresses our uncertainty about the value of t for each value of x. From this conditional distribution we can make predictions of t, for any new value of x, in such a way as to minimize the expected value of a suitably chosen loss function. [3]

The simplest linear model for regression is one that involves a linear combination of the input variables :

$$\mathbf{y(x,w)=w_0+w_1x_1+...+w_Dx_D}$$

where $\mathbf{x = (x_1, . . . , x_D)^T}$. This is often simply known as *linear regression*. The key property of this model is that it is a linear function of the parameters w_0 , . . w_D . It is also, however, a linear function of the input variables x_i, and this imposes significant limitations on the model. [4]

In this project we are asked to perform linear regression using basis functions.

Choosing the basis Function

Options: Basis function could be one of "Gaussian Basis Function" or "Sigmoidal Basis Function".
Chosen: In our case, "Gaussian Basis Function" has been used because it is non-linear in 'x' and can be more accurate.
Why*?*
Because, In Gaussian Basis Function, we use (x-mu)^2 which makes it non-linear (quadratic) in x. However, "Sigmoidal Basis Function" is linear in 'x' to most part and plays a little below Gaussian Function. Because, Sigmoidal Basis Function uses just (x-mu). So, Gaussian Basis Function is used.

Stochastic Gradient Descent Method:

Batch techniques, such as the maximum likelihood solution , which involve processing the entire training set in one go, can be computationally costly for large data sets. If the data set is sufficiently large, it may be worthwhile to use *sequential* algorithms, also known as *on-line* algorithms, in which the data points are considered one at a time, and the model parameters up- dated after each such presentation. Sequential learning is also appropriate for real- time applications in which the data observations are arriving in a continuous stream, and predictions must be made before all of the data points are seen.

We can obtain a sequential learning algorithm by applying the technique of *stochastic gradient descent*, also known as *sequential gradient descent*, as follows. If the error function comprises a sum over data points $E = {}_n E_n$, then after presentation of pattern n, the stochastic gradient descent algorithm updates the parameter vector w using

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n$$

where τ denotes the iteration number, and η is a learning rate parameter. We shall discuss the choice of value for η shortly. The value of w is initialized to some starting vector $w^{(0)}$. For the case of the sum-of-squares error function, this gives

$$w^{(\tau+1)} = w^{(\tau)} + \eta(t_n - w^{(\tau)T}\phi_n)\phi_n$$

Problem Statement

The problem statement comprises of implementing a linear regression to different data sets and evaluating the performance. The main learning to be gained from the project is how to map an input vector x into a scalar target t. In this project we have been asked to perform 4 tasks.
1. Train a linear regression model on real dataset using batch method.
2. Train a linear regression model on real dataset using stochastic gradient descent.
3. Train a linear regression model on synthetic dataset using batch method and evaluate its performance.
4. Train a linear regression model on synthetic dataset using stochastic gradient descent and evaluate its performance.

Dataset

We were told to implement linear regression on two data sets:

Real World Data Set: A real world dataset from a problem in information retrieval known as Learning to Rank (LeToR, where the input vector is derived from a query-URL pair and the target value is human value assignment about how well he URL corresponds to the query).
Synthetic Data Set: A synthetic dataset generated using a mathematical formula plus some noise.

Solution Approach

We begin by dividing the real world dataset into 3 parts according to the given percentages:

- Training set – consisted of 80% of data
- Validation set – consisted of 10% of the data
- Test set – consisted of 10% of the data

Also we had to partition the synthetic data set into two parts, training as well as validation data set.
For that division I chose to split then in an 80-20% ratio. We didn't need to choose a test set here because the testing of our trained model would be done later using a different test set.

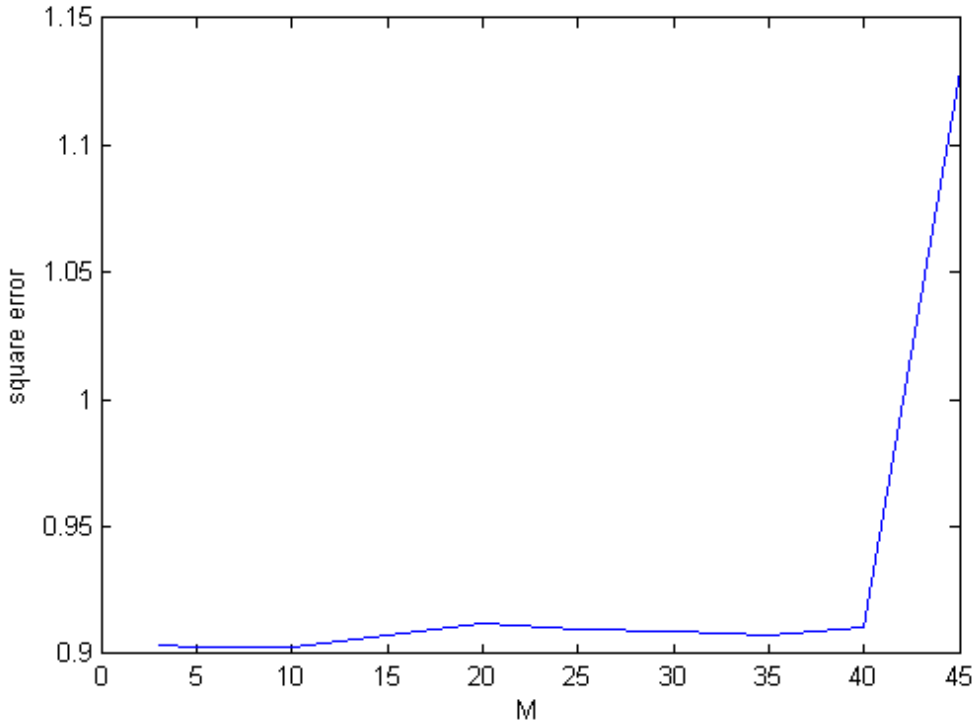
Steps Followed:

I divided all the tasks into two broad categories i.e Real work data set tasks and Synthetic Data set tasks.

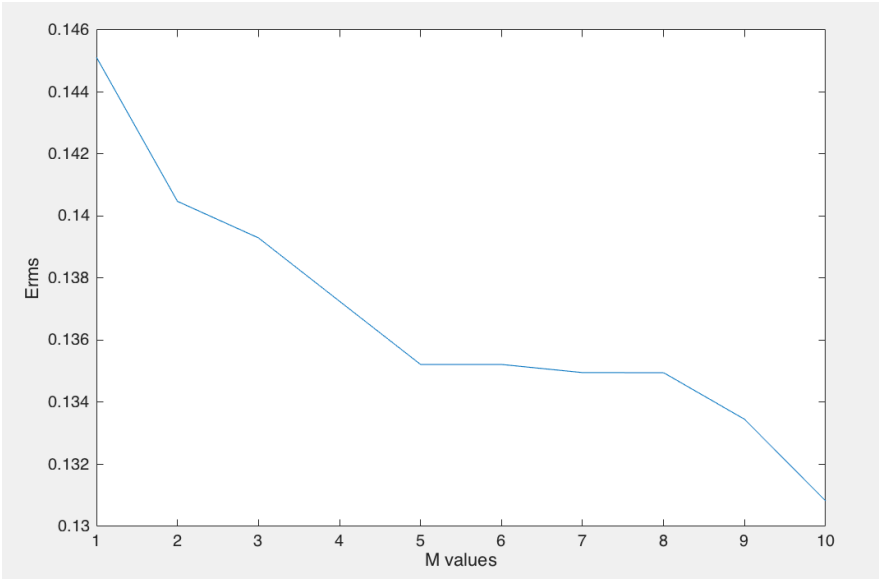
I started with the Real world data set and performed the following tasks on the set.

1. Divided the data set into three parts based on the given percentages. We were asked to make sure that the divided sets did not overlap.

2. The next task was to choose M(number of basis functions). I started off with a random M value of 15. After having a working code, I wanted to tune the value of M for optimal error. So I plotted a curve between the error values and M ranging from 3 to 45 and found that the optimal value of M should be 10 because it gives the lowest error for my model.

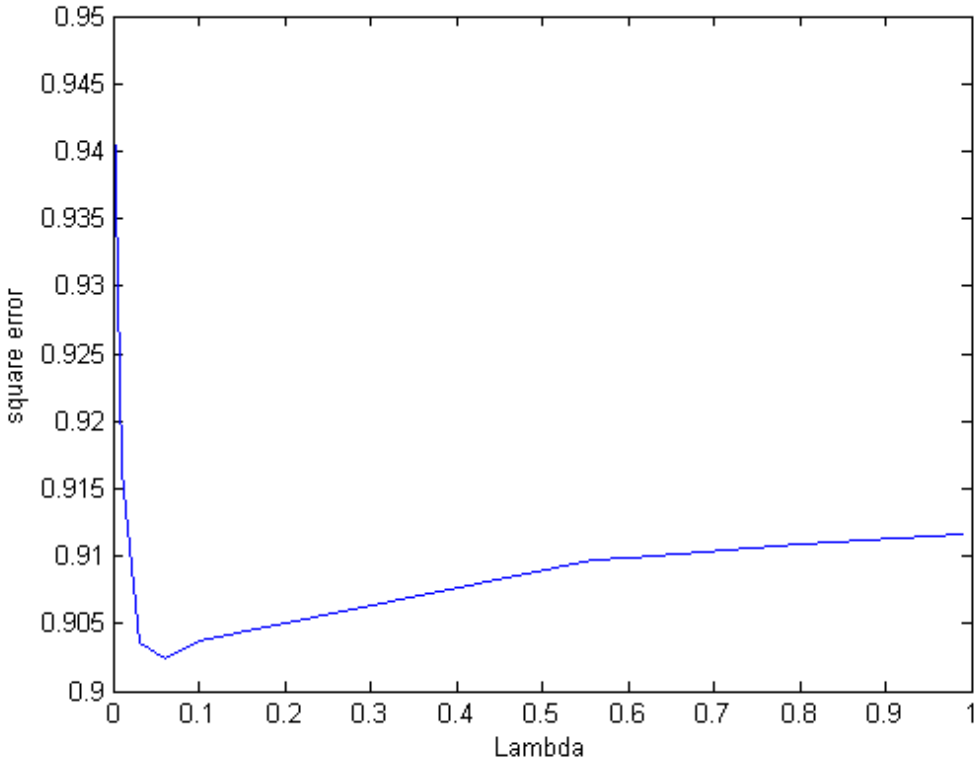


Plot for real data set



Plot for Synthetic data set

3. Choosing the lambda value (**λ**): The regularization factor λ is chosen at random in the beginning but the optimum value is chosen based on the plot between the error and the different lambda values. Further tuning of the model to accommodate other parameters also lead to several changes in the value of lambda until a final value was reached.



4. Choosing mean(mu) was the next task. As asked I picked M random values from the given data set and took them as mu vectors and as a result constructed a mean matrix or mu1.

5. Computing **Σ** (Sigma1 or variance matrix). I computed the variance of the data set rows and had to make sure that the matrix wasn't singular because the matrix had to be inverted later for calculating the basis function values. Some modification was required and I added a noise to my matrix. Later the matrix had to made singular by using the diag function in MATLAB so that only the main diagonal had variance values and all the other values were 0. Lastly this had to be converted into a 3 dimensional matrix as per the requirements. Finally the Sigma1 matrix comes out to be a D*D*M matrix.

6. Creating the design matrix: The next task was to create the design matrix which is an N*M matrix of the basis function values for our training data set. For this task first we had to compute the basis function values for the data set for each of the computed basis functions using the formula:

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

7. The next important task was to calculate the W matrix for the closed form solution. This is the weight matrix and it is computed using the design matrix and the target values of the input data set.

8. The next task was to calculate the square error and then the $\mathbf{E_{rms}}$. This error value is the one based on which we tune our hyper parameters. To calculate the square error we use the following formula

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

Then we calculate the $\mathbf{E_{rms}}$ for the closed form solution using the formula:

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

where $w*$ is the solution and N_V is the size of the validation dataset.

9. Since we are done with the closed form solution, we move on to calculating the weights based on the stochastic gradient descent method. I used the same M and lambda values as used for the closed form solution. For that the first step was to choose an initial weight vector w01. I chose to take random values from 0 to 1 for the weights using the rand() function. But later I had to modify this choice and I had to go ahead with 0 as the value for the initial weights.

10. The next parameter to be chosen for stochastic descent is the eta **(η)** value. There are several ways to choose the eta value. It can be chosen to be a constant for the entire update process or it can be an adaptive learning rate that is varied based on the performance of the model which in turn is judged based on the error values. I started off with a constant small eta values but later modified my code to change the eta values based on the error values. All these values used for eta are put into an eta vector.

11. The next task is to calculate the weight updates. This update also needs to include a regularization term, which is based on the lambda value used in the closed form solution. We are asked to save all these update values in a vector and submit the same as a deliverable. The weight update are calculated using the following expressions:

The stochastic gradient descent algorithm first takes a random initial value $\mathbf{w}^{(0)}$. Then updates the value of $\mathbf{w}$ using

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \quad (9$$

where $\Delta\mathbf{w}^{(\tau)} = -\eta^{(\tau)}\nabla E$ is called the weight updates. It goes along the opposite direction the gradient of the error. $\eta^{(\tau)}$ is the learning rate, deciding how big each update step would b Because of the linearity of differentiation, we have

$$\nabla E = \nabla E_D + \lambda \nabla E_W \quad (10$$

in which

$$\nabla E_D = -(t_n - \mathbf{w}^{(\tau)\top}\boldsymbol{\phi}(\mathbf{x}_n))\boldsymbol{\phi}(\mathbf{x}_n) \quad (11$$

$$\nabla E_W = \mathbf{w}^{(\tau)} \quad (12$$

12. These weight updates are added to the initial weight vector to get the updated weight. I tried to find local minima of the weight updates but ultimately I had to modify my code to run the iterations over the entire data set to get the final updated weight vector. This is where most of the modification of the code was required in order to get satisfactory results from the auto grader. I tried tweaking my parameters in a lot of different ways in order to get the difference between the closed form solution weights and the stochastic gradient descent weights to an acceptable level, but ultimately I had to settle for the minimum difference in values that I could achieve (not optimal as per the grader).

13. After tweaking the parameters on the training data set I calculated the error values on the validation data set and then further on the testing data set to verify the correctness of the regression model.

14. Now the next task is to perform all the above-mentioned steps on the synthetic data set. The only difference is that there is no test set required for the synthetic data set. So we systematically perform all the other steps on the synthetic data set, which is comparatively smaller in size as compared to the real world data set.

Conclusion and Findings

The entire project was a learning exercise to train a model based on input variables and target vectors using Linear Regression model. Due to the multiplicity of the hyper parameters to be tuned for the model, a lot of iterations were required to be performed with each iteration throwing out varying results. Tuning each parameter

individually and then modifying its values based on the other parameters was a grueling task but it provided several learnings, most importantly reinforcing the concept of linear regression and the ways to reduce over fitting by using regularization factor.
I chose the following values for my hyperparameters:
M1=10 lambda1=0.03 mu1=random values eta1=tuned based on error starting from 1 w01=zeros
M2=5 lambda2=0.05 mu2=random values eta2=tuned based on error starting from 1 w02=zeros

Reference:
All references taken from Pattern Recognition and Machine Learning by Christopher M. Bishop.