Project 1: Regression

PROJECT REPORT

Machine Learning [CSE – 574]

Prof. Sargur N. Srihari

RAVI KARAN SHARMA

UB PERSON# 50097193

1 OVERVIEW

This project is to implement regression on web search ranking data set LETOR 4.0. We are given a dataset of vectors and target values. There are 46 features for each sample in the dataset. We need to learn the weight of each feature which can give us the best result given a new sample. We use the technique of regression to learn these weights by using the samples given. We use the Gaussian Model of regression and vary the degree or complexity of the model until it gives us the closest prediction we can get on a previously unseen keyword.

2 REGRESSION MODEL

A Regression Model is first fed some sample data, to proceed by arriving at a tentative set of values of the weights. This is called the training set. The training set should contain a substantial number of samples for fine tuning the weights of the regression model. The more the samples, the better an estimate do we get about what the values of the weights will be. This set of values are evaluated for accuracy against a smaller subset of the data called the validation set, whose purpose is to test how well is the model performing with its current complexity or degree. This is a small set which is just used to evaluate the error the model is generating on unlearned data. We chose that degree of the model which gives us the minimum error, and we designate that as the most accurate complexity of our model, which gives us the closest prediction given an unlearned value, from the rest of the dataset which we call the testing set.

For this purpose, we partition the dataset into 3 parts containing 40%, 10% and 50% of the total values respectively. We use the 40% partition as the training set, 10% as a validation set and the remaining 50% as the validation set. We separate out the first column as it has the target values for all the samples, we refer to as target vector t.

With our data sets ready, we proceed to apply the technique of regression on training set. Regression is a general way of obtaining a linear model that fits a given set of data points as accurately as possible. This estimation is subject to error from the actual values, which we use to train the model to improve its accuracy and provide a general model which minimizes the error across all the given data values.

Given: a set of training examples (x_n, t_n), n =1 to N, where N is the total number of samples
Goal: learn a function y(x) to minimize a loss function (error function): E(y,t)

We use the linear basis function model which is given by:

$$y(x,w) = w_0 + {}_{j=1}\sum^{M-1} w_j \phi_j(x) = \phi_j(x)w$$

Where M is the degree of the model
w_0, w_1 ... to w_{M-1} are the weights we need to calculate
ϕ is the basis function we use to calculate w.

In the simplest case $\phi_j(x) = x_j$. But we take the value of ϕ from Gaussian distribution as

$$\phi(x) = \exp\{ - (x - \mu)^2/2s^2\} \qquad \text{where } \mu \text{ is the mean and s is the variance}$$

μ and s are called the hyper-parameters of the model with degree M. For a model with degree M there are M basis functions. Typically $\phi_0(x)$ is 1, so w_0 acts as a bias parameter.

We calculate μ and s from the data in the training set and prepare the design matrix for the data by substituting each data value in each basis-function successively. Hence, a model of degree M will have a design matrix of $1+(M-1)\times D$ columns long, where D = 46 in our case. We get each set of 46 columns from each of the basis-functions ϕ_1, ϕ_2, ... ,ϕ_M respectively.

Hence the design matrix ϕ is

$$\Phi = \begin{array}{cccccccccccccc}
1 & \phi_1(x_1^1) & \phi_1(x_1^2) & \ldots & \phi_1(x_1^D) & \phi_2(x_1^1) & \phi_2(x_1^2) & \ldots & \phi_2(x_1^D) & \ldots & \phi_M(x_1^1) & \phi_M(x_1^2) & \ldots & \phi_M(x_1^D) \\
1 & \phi_1(x_2^1) & \phi_1(x_2^2) & \ldots & \phi_1(x_2^D) & \phi_2(x_2^1) & \phi_2(x_2^2) & \ldots & \phi_2(x_2^D) & \ldots & \phi_M(x_2^1) & \phi_M(x_2^2) & \ldots & \phi_M(x_2^D) \\
\vdots & & \vdots & & \vdots & & \vdots & & \vdots & & & \vdots & & \vdots \\
1 & \phi_1(x_{N-1}^1) & \phi_1(x_{N-1}^2) & \ldots & \phi_1(x_{N-1}^D) & \phi_2(x_{N-1}^1) & \phi_2(x_{N-1}^2) & \ldots & \phi_2(x_{N-1}^D) & \ldots & \phi_M(x_{N-1}^1) & \phi_M(x_{N-1}^2) & \ldots & \phi_M(x_{N-1}^D) \\
1 & \phi_M(x_N^1) & \phi_1(x_N^2) & \ldots & \phi_1(x_N^D) & \phi_2(x_N^1) & \phi_2(x_N^2) & \ldots & \phi_2(x_N^D) & \ldots & \phi_M(x_N^1) & \phi_M(x_N^2) & \ldots & \phi_M(x_N^D)
\end{array}$$

Using the design matrix Φ, we can calculate the weights by

$$w^* = (\Phi^T \Phi)^{-1} \Phi^T t \qquad \text{where t is the target vector}$$

On calculating the values for each sample given by the model, we can find out the error between the model's values and the actual values that are given in the data set. We measure this error by calculating the root means square error or E_{RMS} given as

$$E_{RMS}(w) = [(\Phi^T w - t)(\Phi w - t)/N]^{1/2}$$

To prevent against a higher degree we use a regularization parameter λ to limit the increase in M, due to which the model can become quite complex. We introduce λ in our equation for w and thus into E_{RMS} as

$$w^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T t \qquad \text{where I is an identity matrix}$$

Small weight values of λ are encouraged.

For every value of M and λ, we obtain the basis functions $\phi_1, \phi_2, \ldots, \phi_M$, Φ and w* from the training set and proceed to validate our model on the validating set. With increasing M the RMS error value on the validation set denoted as $E_{validation}$, decreased, but we were facing the problem of over-fitting, so after introducing λ to small value the $E_{validation}$ decreased further, on a lesser value of M, thus making the model simpler while decreasing the error at the same time.

If we keep increasing the M, after considering the regularization parameter, the $E_{validation}$ begins to increase after a certain limit. This indicates that an optimal value of M and λ was reached in the previous iteration. Hence, we arrive at the optimal complexity for the model. We go further to show how this model performs on the testing set and represent the actual performance through E_{test}.

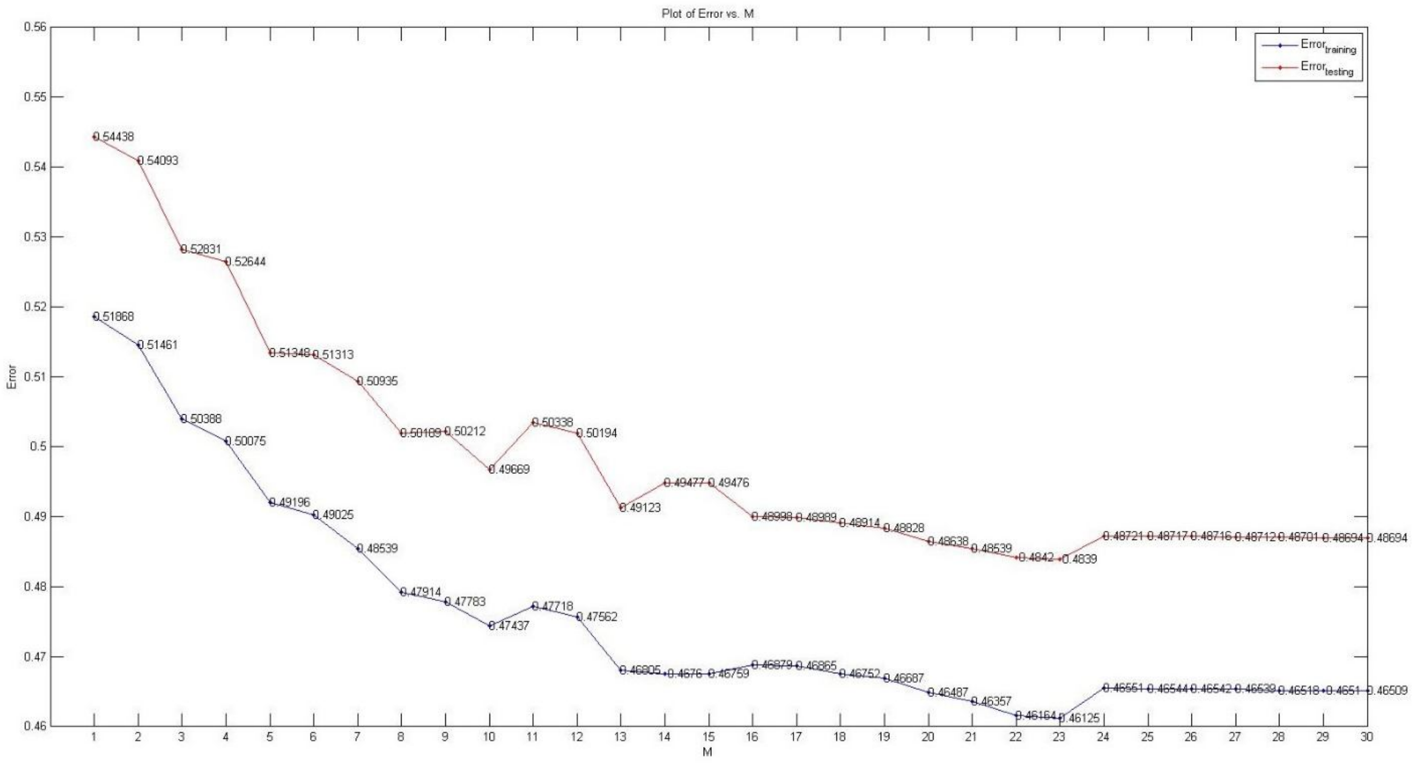
A graph plot comparing E_{test} and $E_{training}$ represents the choice of the optimal value of M, also depicting the variation of error for each value of M.

3 RESULTS

The degree of the model was increased incrementally from 1 through 30, which sufficiently showed the optimal value of M required.
The regularization factor λ was varied beginning from 0.001 to 1000 in multiples of 10, but a lower error and thus the optimal performance of the model was found to be at λ = 0.001 and M = 23.

The E_{RMS} found while training and testing has been plotted as $E_{TRAINING}$ and $E_{TESTING}$ below

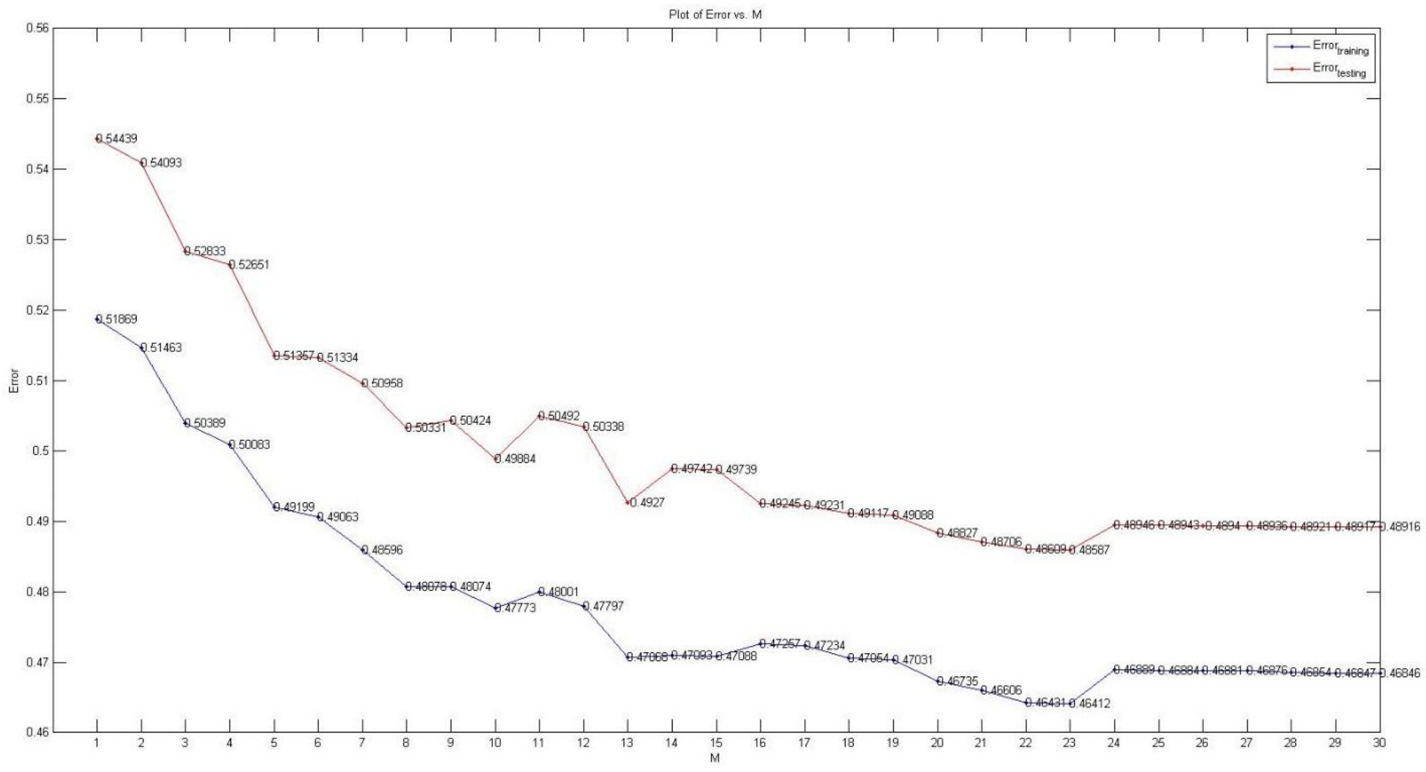
(a) λ = 0.001, lowest $E_{TRAINING}$ = 0.46125, $E_{TESTING}$= 0.483897 at M = 23



M	$E_{training}$	$E_{validation}$	$E_{testing}$
1	5.186797048723698e-01	5.638027955575883e-01	5.443849837677810e-01
2	5.146059940277609e-01	5.520892994170843e-01	5.409259548847909e-01
3	5.038751953765578e-01	5.291218576451967e-01	5.283110769208825e-01
4	5.007496984213286e-01	5.232166604953455e-01	5.264438020822320e-01
5	4.919642477136846e-01	5.046302061525215e-01	5.134826751382824e-01
6	4.902532847219582e-01	4.959784871744563e-01	5.131257292874940e-01
7	4.853883308909278e-01	4.913527911061465e-01	5.093524224834856e-01
8	4.791425475755370e-01	4.778784872912280e-01	5.018896195577332e-01
9	4.778305529774431e-01	4.780954638318321e-01	5.021249000536183e-01
10	4.743742054503036e-01	4.657833540813698e-01	4.966852137918562e-01
11	4.771812301378062e-01	4.744166559507284e-01	5.033802736460739e-01

M	$E_{training}$	$E_{validation}$	$E_{testing}$
12	4.756203264036242e-01	4.763804255870249e-01	5.019409308614727e-01
13	4.680530455295407e-01	4.518244682436994e-01	4.912289910495785e-01
14	4.676040842009522e-01	4.542647795386769e-01	4.947724184096032e-01
15	4.675889272567534e-01	4.541706327934181e-01	4.947587732005506e-01
16	4.687913316302289e-01	4.531609298246553e-01	4.899766574360553e-01
17	4.686546210548604e-01	4.527494571277752e-01	4.898916315384269e-01
18	4.675203819746091e-01	4.487510700568578e-01	4.891430987859650e-01
19	4.668657253988908e-01	4.475465700595802e-01	4.882829043418859e-01
20	4.648698519217474e-01	4.373228603497542e-01	4.863826401992739e-01
21	4.635659324829633e-01	4.313665763514336e-01	4.853884178930770e-01
22	4.616368993067721e-01	4.256406240780540e-01	4.841991977609569e-01
23	4.612517735036478e-01	4.247815754056850e-01	4.838969031294655e-01
24	4.655066042851860e-01	4.373412041355178e-01	4.872084629398934e-01
25	4.654431757477869e-01	4.372758067533812e-01	4.871742971095665e-01
26	4.654249195515598e-01	4.371613299577424e-01	4.871560885732053e-01
27	4.653918591565296e-01	4.369618895659668e-01	4.871205161463469e-01
28	4.651849695399236e-01	4.365366309633880e-01	4.870116959800021e-01
29	4.650983212381221e-01	4.363124691667473e-01	4.869383907488272e-01
30	4.650928240196638e-01	4.362904812432711e-01	4.869353838306385e-01

(b) $\lambda = 0.01$, lowest $E_{TRAINING} = 0.46412$, $E_{TESTING} = 0.485868$ at M = 23



M	$E_{training}$	$E_{validation}$	$E_{testing}$
1	5.186860758788447e-01	5.638039398905580e-01	5.443850270082992e-01
2	5.146319926770008e-01	5.521263409786361e-01	5.409283196137998e-01
3	5.038891393719434e-01	5.292753026656046e-01	5.283310671961934e-01
4	5.008323827626583e-01	5.234044044871623e-01	5.265070829970423e-01
5	4.919939606431696e-01	5.054261248765561e-01	5.135697823129998e-01
6	4.906344499593356e-01	4.997661221256461e-01	5.133371938413496e-01
7	4.859618959965688e-01	4.958228313485885e-01	5.095800277371385e-01
8	4.807797496765794e-01	4.827234611316175e-01	5.033142379574830e-01
9	4.807366230524948e-01	4.854598289305036e-01	5.042394749496254e-01
10	4.777313591543985e-01	4.751618976571516e-01	4.988360609885618e-01
11	4.800080243258420e-01	4.840885601697407e-01	5.049177385367396e-01
12	4.779737965341788e-01	4.814949806027689e-01	5.033809442564626e-01
13	4.706780508127311e-01	4.591452180213631e-01	4.926987906015891e-01
14	4.709260982393237e-01	4.687400364916310e-01	4.974245923295709e-01
15	4.708782237949115e-01	4.686062300570440e-01	4.973889440846083e-01
16	4.725698472656348e-01	4.617128114177028e-01	4.924538028834318e-01
17	4.723438963096357e-01	4.611812569320125e-01	4.923113795673981e-01
18	4.705403865946789e-01	4.585025761434120e-01	4.911710390769334e-01
19	4.703092507177740e-01	4.578627891718992e-01	4.908823367030767e-01
20	4.673504022255783e-01	4.502668611888950e-01	4.882721281011061e-01
21	4.660561178426236e-01	4.431101537696890e-01	4.870561644979785e-01
22	4.643078684418528e-01	4.385142052308869e-01	4.860877252188579e-01
23	4.641168319720431e-01	4.377354300861953e-01	4.858678119274081e-01
24	4.688943152263068e-01	4.474761304670346e-01	4.894582106552299e-01
25	4.688445375689853e-01	4.474503235566116e-01	4.894275178178785e-01
26	4.688137470613614e-01	4.472107622494411e-01	4.893968343067798e-01
27	4.687595125782130e-01	4.468705616790984e-01	4.893558693826873e-01
28	4.685359194289024e-01	4.463342482910368e-01	4.892146340427713e-01
29	4.684683012061832e-01	4.462340076465877e-01	4.891652377120978e-01
30	4.684628568835681e-01	4.462093349814193e-01	4.891608960419921e-01

Minimization of Error

As it is evident from comparing the two plots for different values of λ, we have got lesser $E_{testing}$ for lower values of λ. After determining an optimal weight value for λ, we see that on increasing the complexity or the degree of the model, the error keeps decreasing, and then becomes almost constant and starts rising again. At that point, it shows that the particular value of M is the appropriate degree of our model. We also find that $E_{VALIDATION}$ is minimum for M = 23. Here, after M = 23, the graph begins to rise again indicating the increasing values of error if we increase the degree any further. Hence the optimal values the model parameters are:

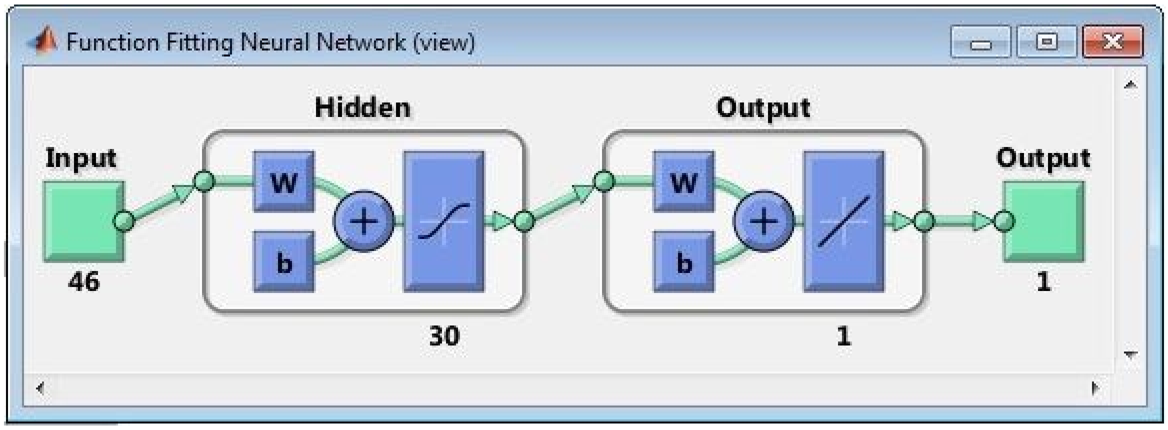
M = 23, λ = 0.001 and E_{RMS} = 0.483897

4 COMPARISON WITH STANDARD PACKAGES

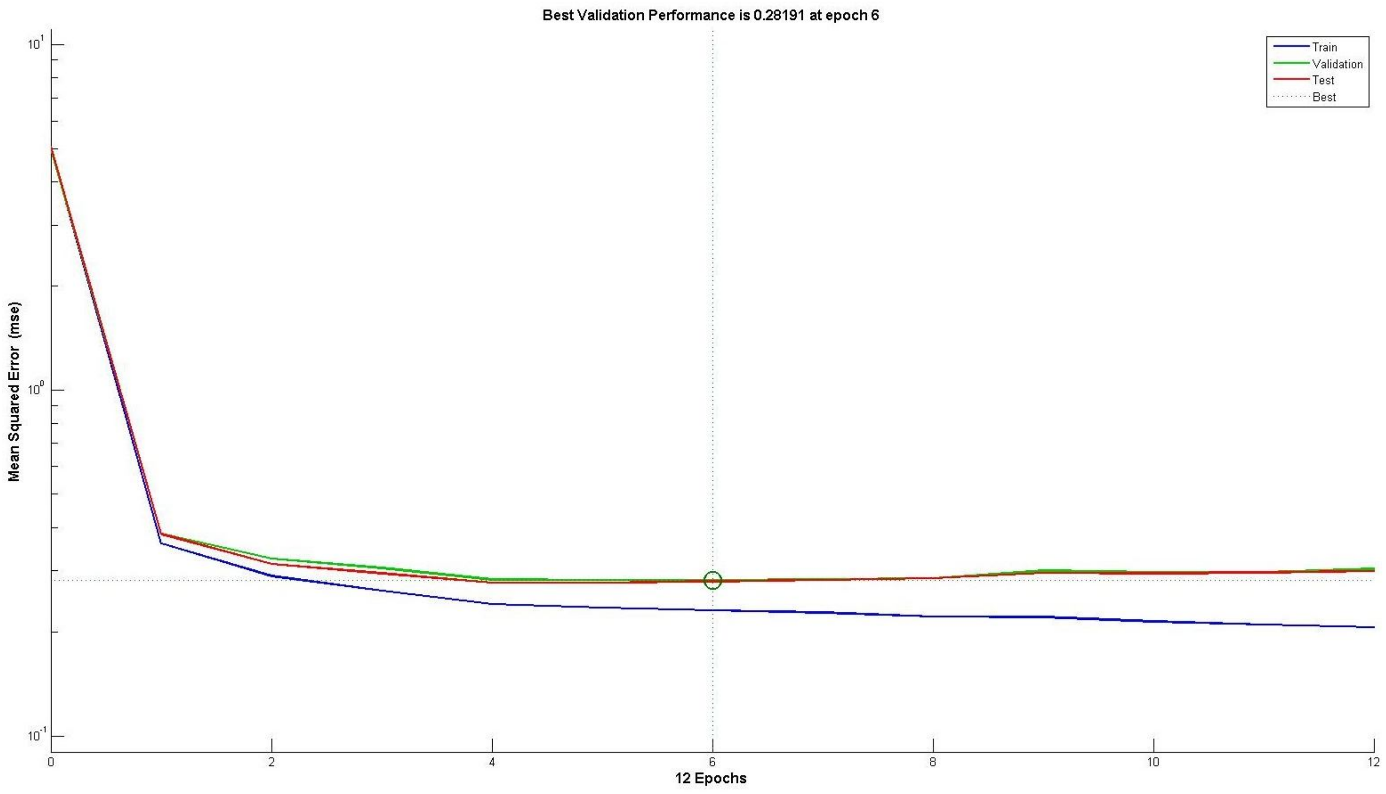
These results were compared with the standard packages available publically. The Neural Network package available with Matlab was chosen and Function Fitting and Pattern Recognition packages were used to compare with the linear regression model used above with varying results.

A. FUNCTION FITTING NEURAL NETWORK

A neural network of 30 hidden nodes was trained on the same LETOR 4.0 data set of 46 features as shown and given the target vector from the data set. It performed poorly compared to the regression model. Learning was done on 70% data as training, 10% as validation and 20% as testing.



RESULTS: The Neural Network gave the following performance (Mean Squared Error) values

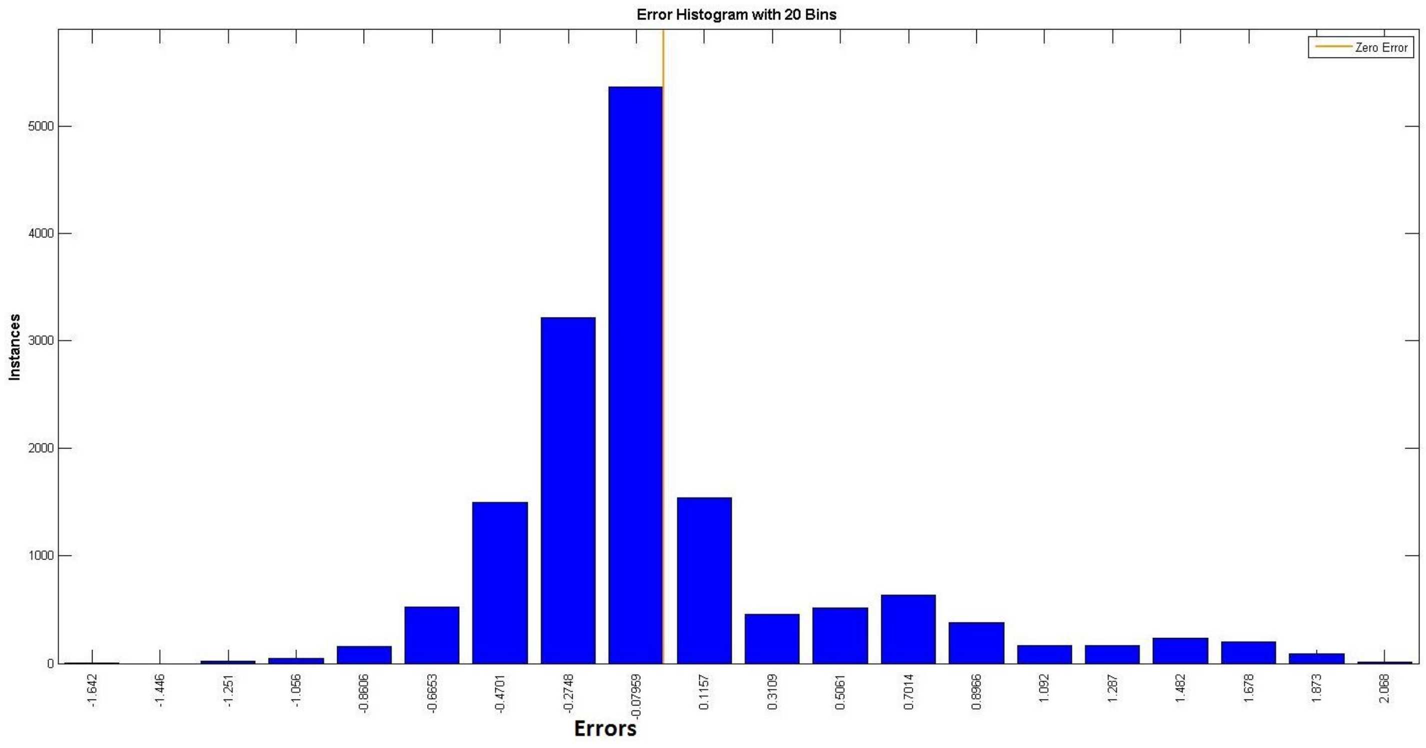


MEAN SQUARED ERRORS (PERFORMANCE)

The network was tested to find the initial performance of:
Mean Squared Error(MSE)/Performance = 2.452782115867008e-01 ≈ 0.245278

The training, validation and test performances of the network were then recalculated:
$MSE_{TRAINING}$ /Training Performance = 2.302837806807008e-01 ≈ 0.230284
$MSE_{VALIDATION}$/Validation Performance = 2.819139397830747e-01 ≈ 0.281914
$MSE_{TESTING}$/Testing Performance = 2.794457847952461e-01 ≈ 0.279446

ERROR HISTOGRAM



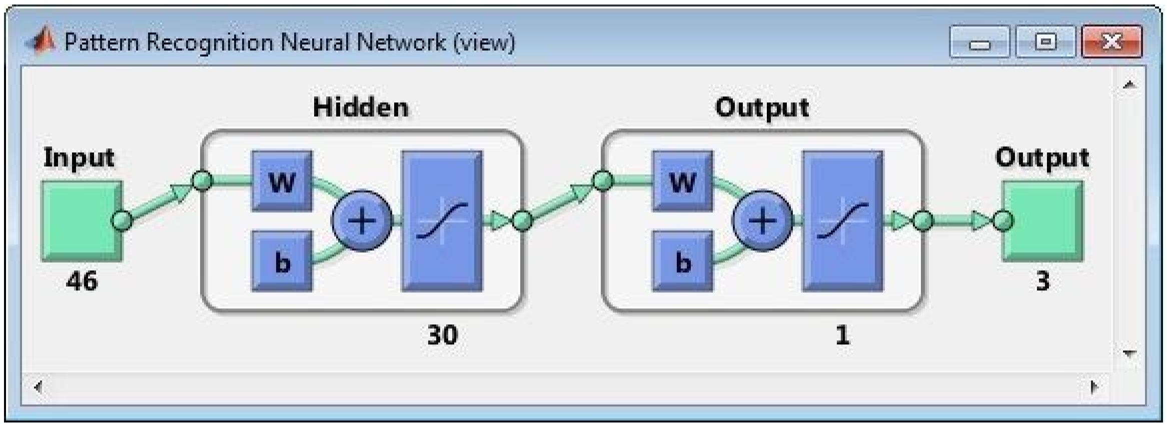
As the E_{RMS} for this neural network is $(MSE_{TESTING})^{1/2}$, the preformance is evaluated on the testing samples which the network has not seen before, the error for comes out to be

$$\mathbf{E_{RMS} = 0.528626}$$

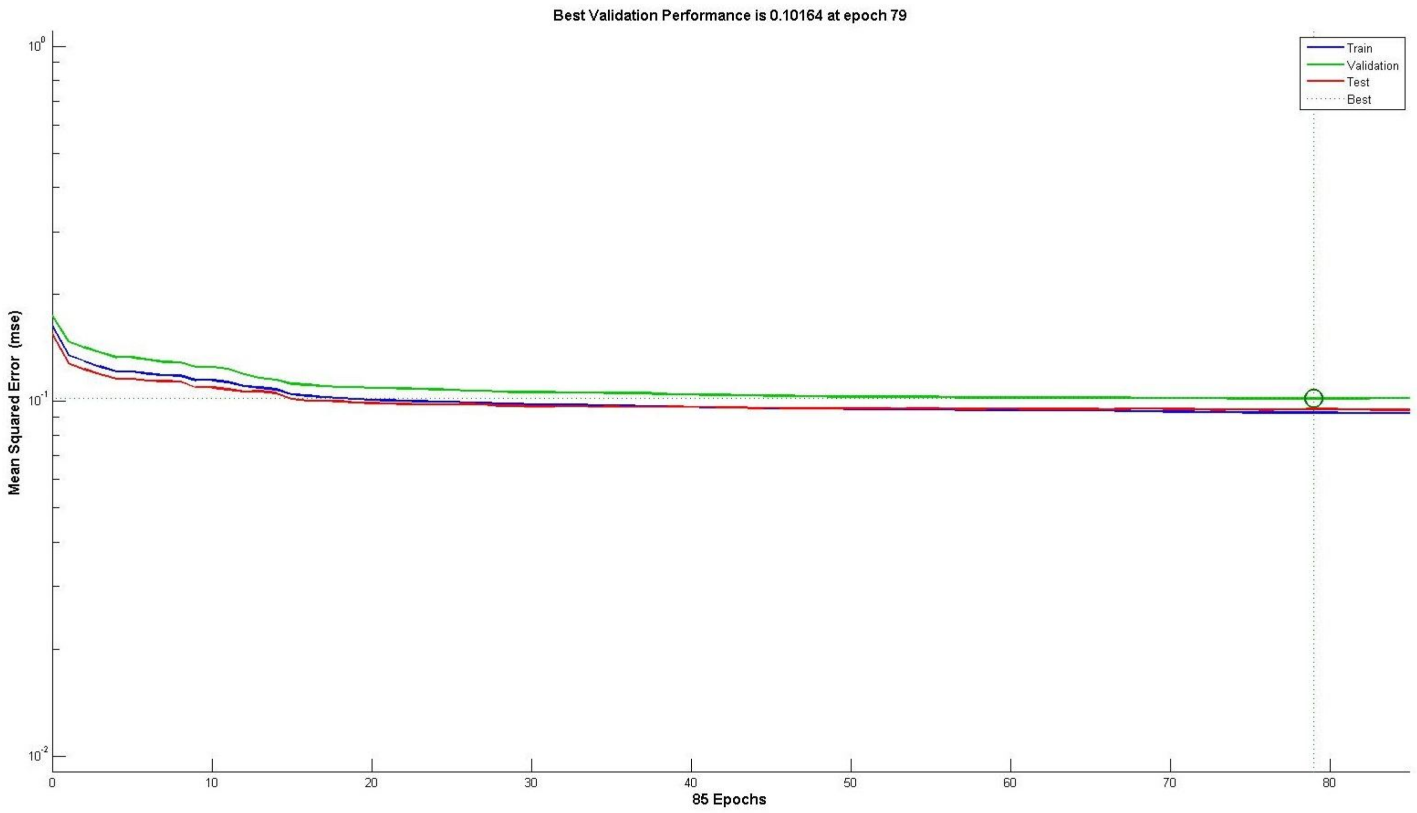
This package *performs worse* than the optimal linear regression model devised above as the $E_{TESTING}$ for the regression model is lesser than the error of this package.

B. Pattern Recognition

A pattern recognition neural network of 30 nodes was used to learn from our original dataset of 46 features as showing in the figure. Learning was done on 70% as training, 10% as validation and rest 20% as testing, the performance as much better than the Function Fitting package above.



Results: The following performance (Mean Squared Error) values were obtained



Mean Squared Errors (Performance)

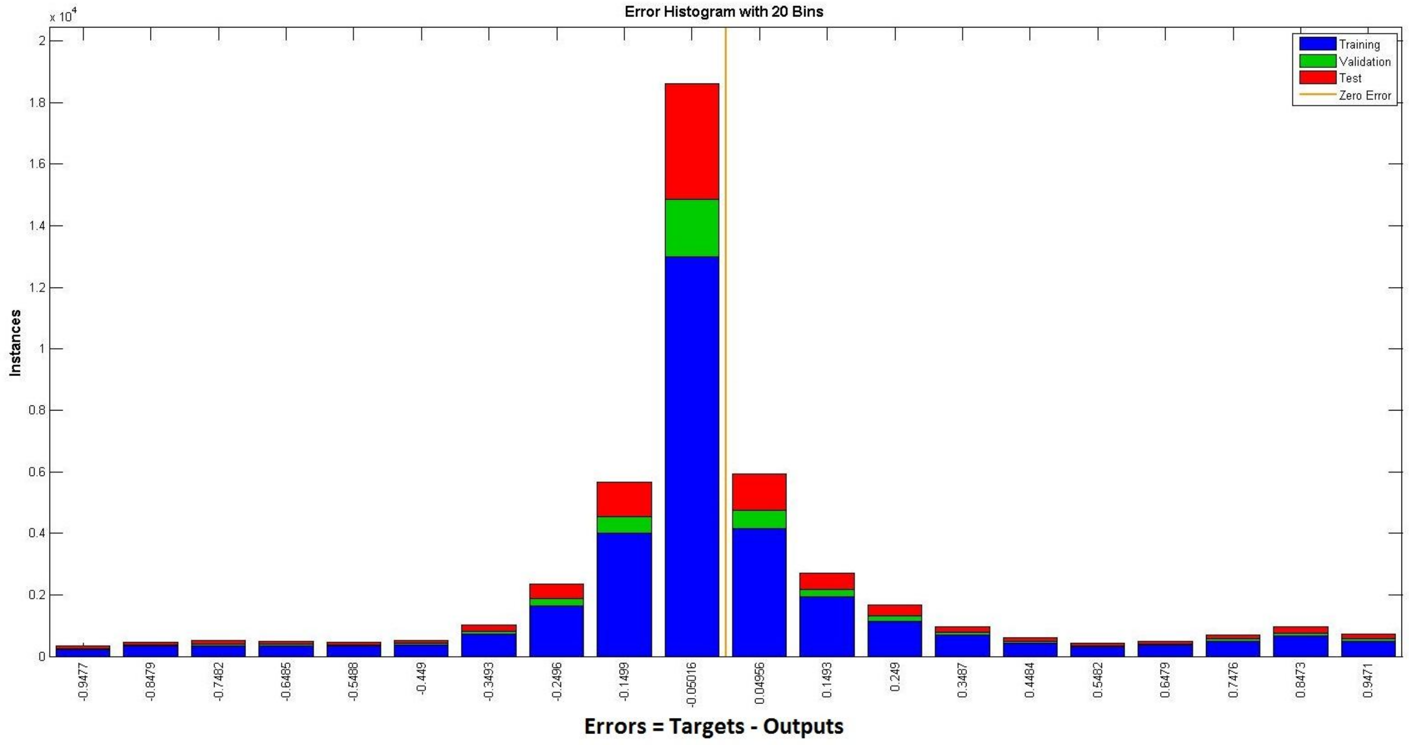
The network was tested to find the initial performance of:

Mean Squared Error(MSE)/Performance = 9.379583909908705e-02 ≈ 0.093796

The training, validation and test performances of the network were then recalculated:
$MSE_{TRAINING}$ /*Training Performance* = 9.243435739514944e-02 ≈ 0.092434
$MSE_{VALIDATION}$/*Validation Performance* = 1.016386849320456e-01 ≈ 0.101639
$MSE_{TESTING}$/*Testing Performance* = 9.464004970776442e-02 ≈ 0.09464

As evident the values were significantly better than the Function Fitting package.

Error values are calculated similarly on performance as E_{RMS} = (performance value)$^{1/2}$



As the E_{RMS} for this neural network is ($MSE_{TESTING}$)$^{1/2}$, the preformance is evaluated on the testing samples which the network has not seen before, the error for comes out to be

E_{RMS} = 0.307636

This package performs approximately *1.5 times better* than the optimal linear regression model formulated above, as the $E^{RMS}_{TESTING}$ is almost half of that found by the regression model.