Abdul Rehman Liaqat

Lab Course Machine Learning

Exercise Sheet 5

Report

Data Cleaning and Pre-Processing:

A snapshot of head.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5

To further clean and standardize the data following steps were performed:

1- Dropping rows with NA values using built in “dropna” function.
2- Checking the presence of any categorical variable using “colUnique” function.
3- Replacing categorical variables to numeric using “dummy” function which converts any non-numeric value to numeric one.
4- Normalizing all the columns using “normalize” function.
5- Shuffling the data with respect to rows with “shuffleit” function.
6- Separating the data into training and test data which are 80% and 20% of total data respectively.

Abdul Rehman Liaqat

Exercise 1: Regularization

Following hyperparameters were used:

Step size (α) = $[\,9\times10^{-4}\,, 9\times10^{-5}, 9\times10^{-6}\,]$

Regularization Constant (λ) = $[\,0.1\,, 0.475,\ 0.85\,]$

Batch size = 50

Mini-BGD function

```
def mbgd(x,y,beta,alpha,imax,epsilon,batches,lamb,xtest,ytest):

    r,c=x.shape
    betaold=beta
    betanew=np.zeros(c)
    rmsetest=np.zeros(imax)
    rmsetrain=np.zeros(imax)
# Epoch Loop
    for itr in range(imax):
# batch Loop
        for re in range(int(r/batches)):
            betanew=beta-alpha*derivative(x.iloc[re*batches:(re+1)*batches]
                                          ,y.iloc[re*batches:(re+1)*batches],beta,lamb)
            beta=betanew
# Convergance condition
            if (abs(lossfunc(x,y,betanew)-lossfunc(x,y,betaold))<epsilon:
                rmsetrain[itr]=rmse(x,y,betanew)
                rmsetest[itr]=rmse(xtest,ytest,betanew)
                return betanew,rmsetrain,rmsetest,itr

        rmsetrain[itr]=rmse(x,y,betanew)
        rmsetest[itr]=rmse(xtest,ytest,betanew)
        betaold=betanew
    return betanew,rmsetrain,rmsetest,itr
```

Functions associated to this function are:

- “derivative”
- “rmse”
- “lossfunc”

Fixed step size was used for all minimizations to reduce the computing load and time.

Abdul Rehman Liaqat

Step size (α) = $[\, 9\times10^{-4}, 9\times10^{-5}, 9\times10^{-6} \,]$

Regularization Constant (λ) = $[\, 0.1, 0.475, 0.85 \,]$

Using regularization constant and step size following 9 combinations were used:

combination 1: $[\, 0.1 \;,\; 9\times10^{-4} \,]$

combination 2: $[\, 0.475 \;,\; 9\times10^{-4} \,]$

combination 3: $[\, 0.85 \;,\; 9\times10^{-4} \,]$

combination 4: $[\, 0.1 \;,\; 9\times10^{-5} \,]$

combination 5: $[\, 0.475 \;,\; 9\times10^{-5} \,]$

combination 6: $[\, 0.85 \;,\; 9\times10^{-5} \,]$

combination 7: $[\, 0.1 \;,\; 9\times10^{-6} \,]$

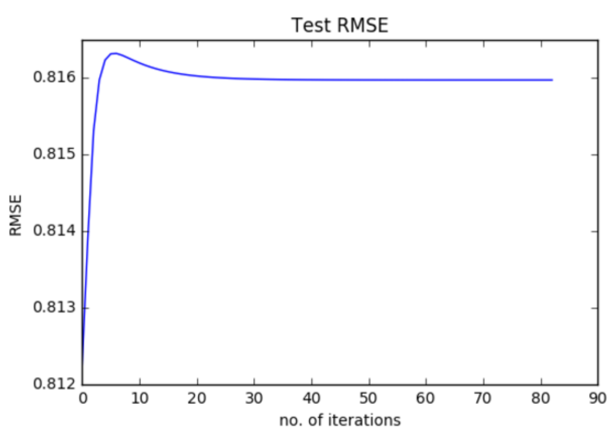
combination 8: $[\, 0.475 \;,\; 9\times10^{-6} \,]$

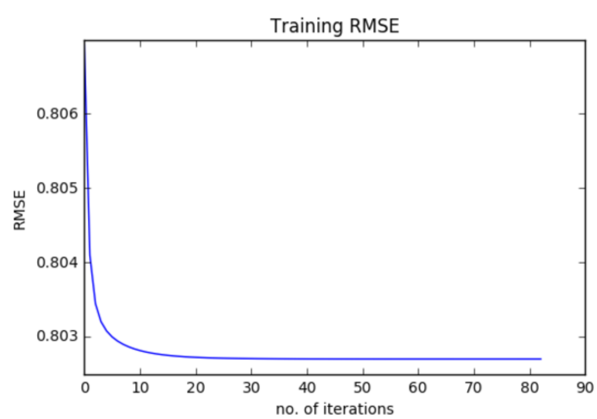
combination 9: $[\, 0.85 \;,\; 9\times10^{-6} \,]$

Plot of Test data RMSE and Train data RMSE achieved on each step follow:

Note: Since most of the plots had different variance hence it was not visually pleasing to draw these two against same y-axis.

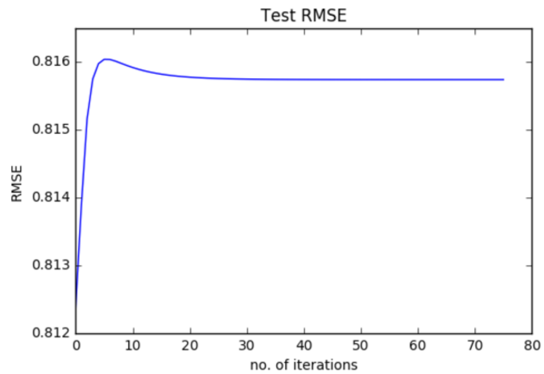
Combination 1:

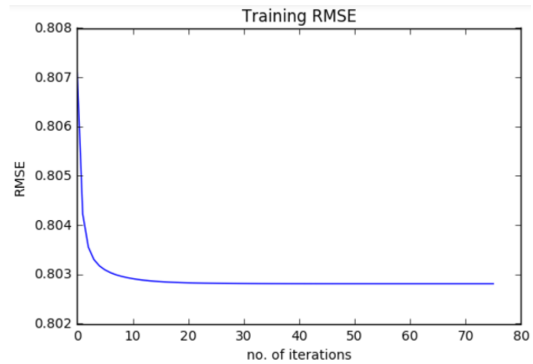




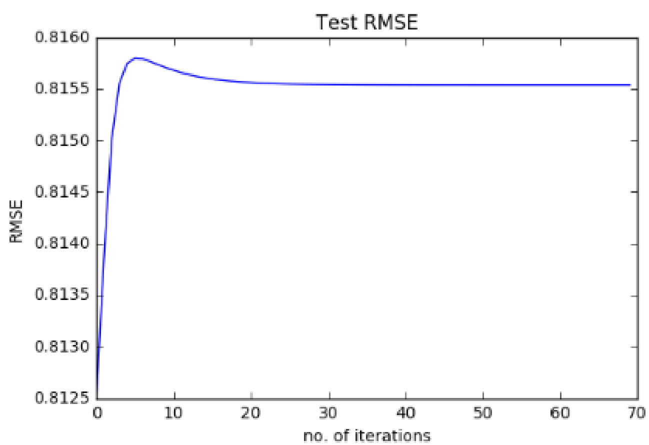
Abdul Rehman Liaqat

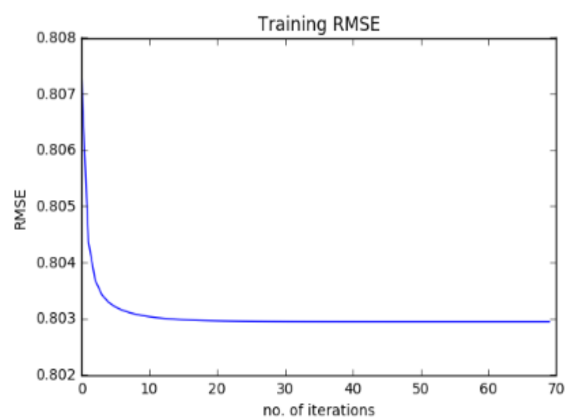
Combination 2:





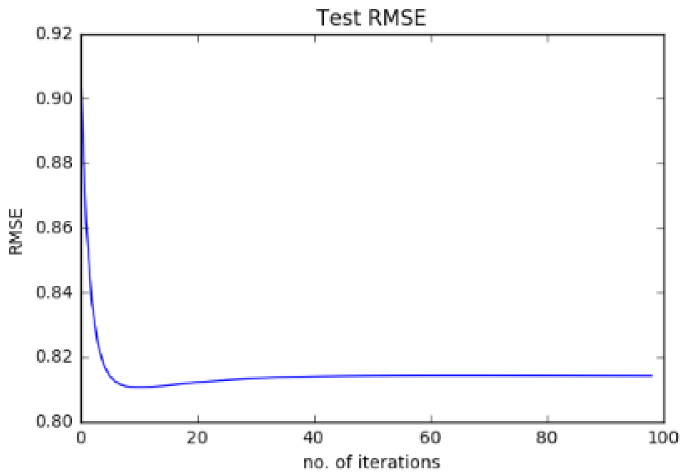
Combination 3:

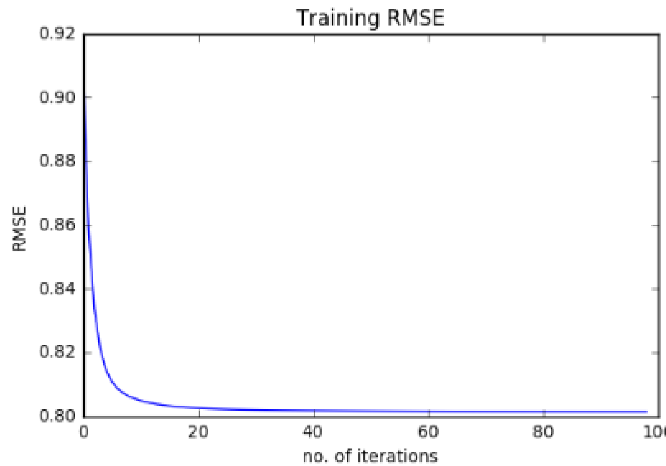




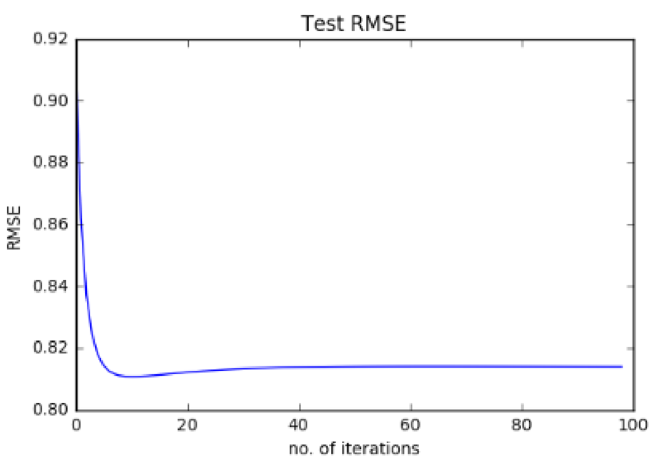
Abdul Rehman Liaqat

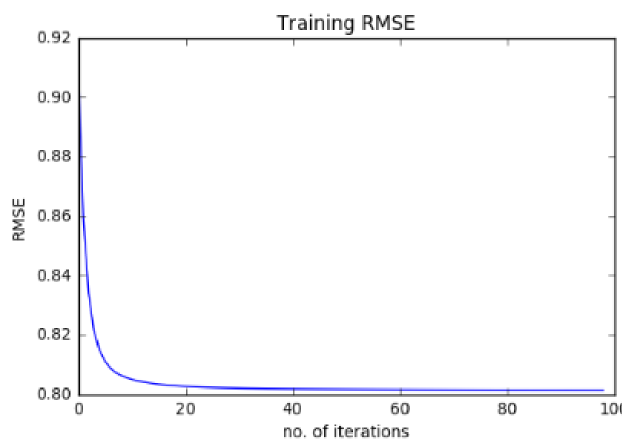
Combination 4:





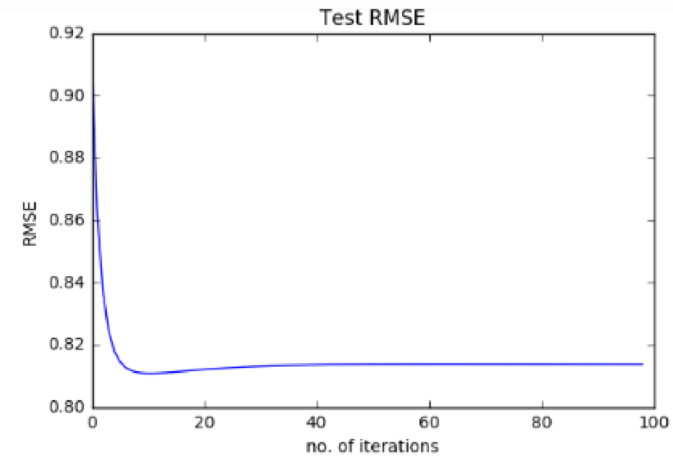
Combination 5:

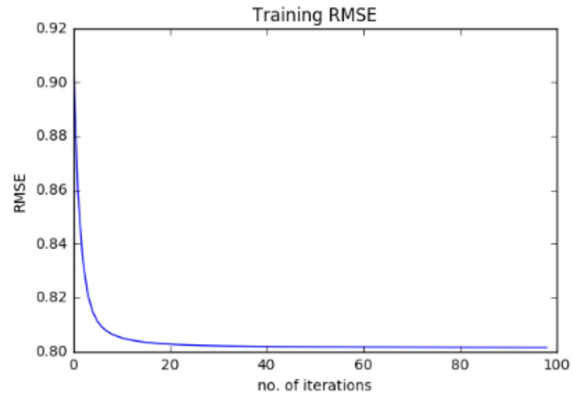




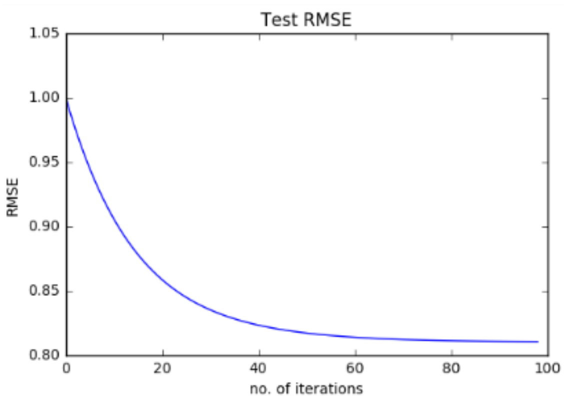
Abdul Rehman Liaqat

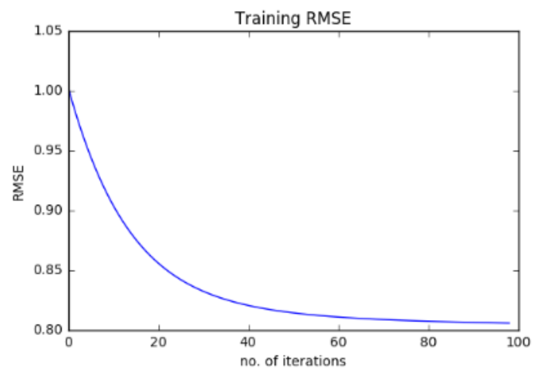
Combination 6:



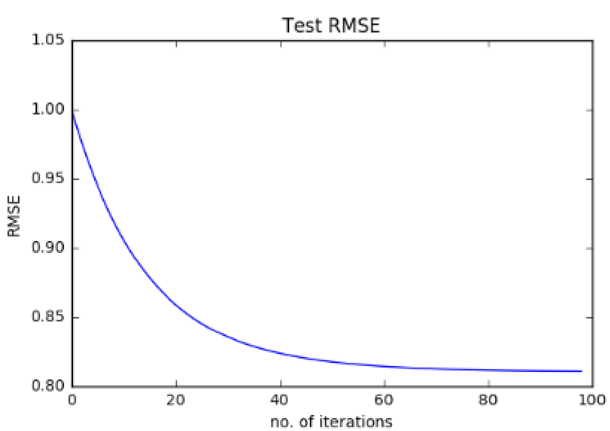


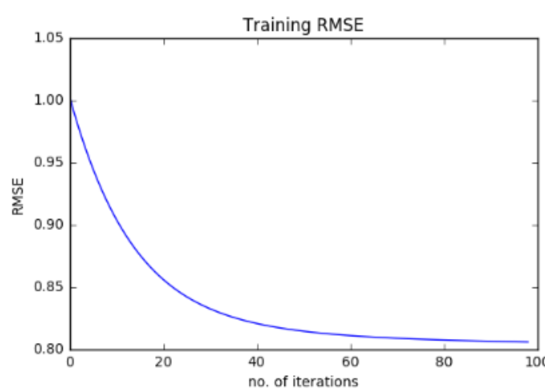
Combination 7:





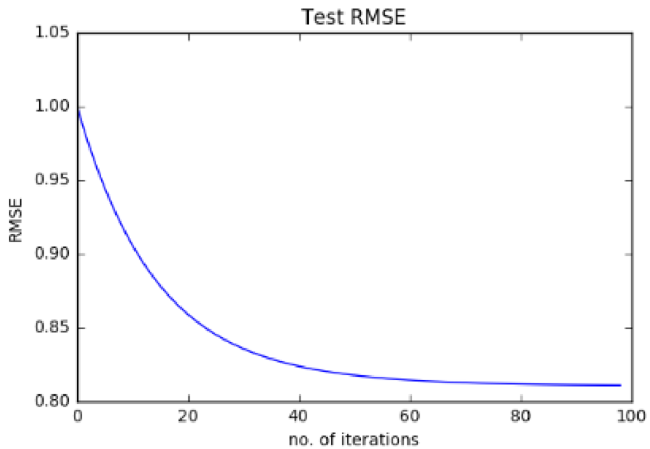
Combination 8:

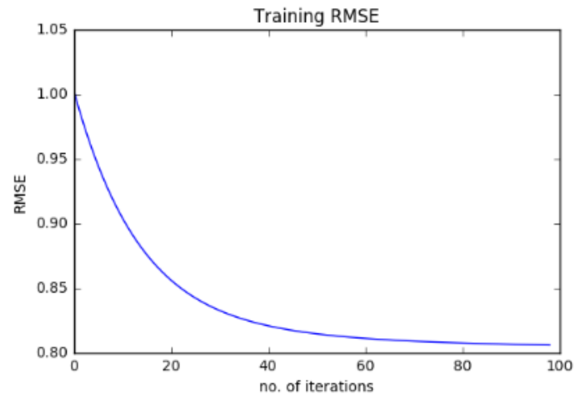




<u>Abdul Rehman Liaqat</u>

Combination 9:





Abdul Rehman Liaqat

Exercise 2: Hyper Parameter Tuning and Cross-Validation

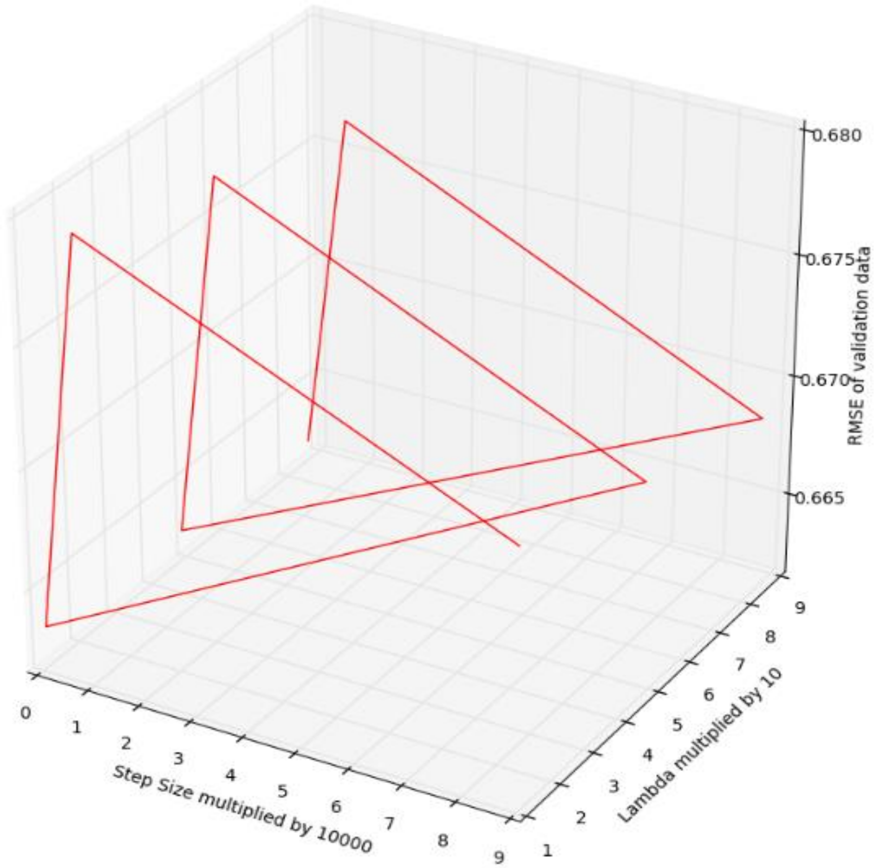
Continuing with the previous exercise, k-fold cross-validation for model selection was performed. All the hyperparameter were same as pervious exercise plus a new variable K was introduced. Its value was 5.

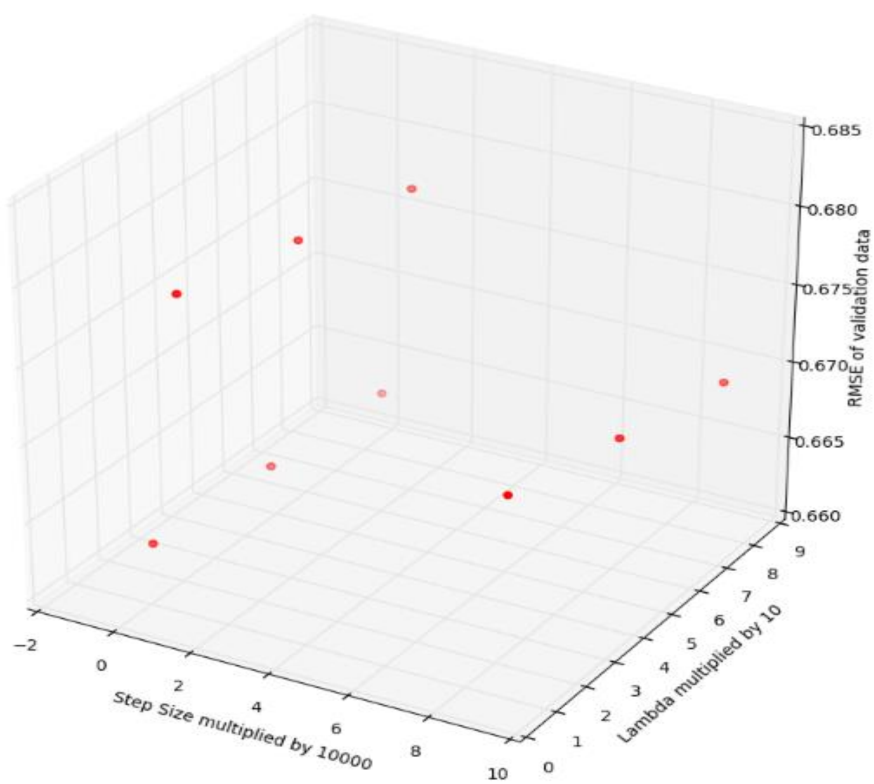
```
error=np.zeros(k*len(lamb)*len(stepsize))
counter=0

#loop to vary the value of regularization constant
for lam in range(len(lamb)):
#loop to vary the value of stepsize
    for alp in range(len(stepsize)):
#loop to perform k-fold operation
        for x in range(k):
#separating training and validation data
            ktrain,kvalid=separate(train,k,x+1)
#further separating x and y from training and validation data
            kxtrain,kytrain=sepxy(ktrain,ytopre)
            kxvalid,kyvalid=sepxy(kvalid,ytopre)
#mini-Bactch Gradient descent
            r,c=kxtrain.shape
            beta=np.zeros(c)
            betaPre=mbgd(kxtrain,kytrain,beta,stepsize[alp],imax,epsilon,batchsize,lamb[lam])
#RMSE collection. Tested on validatoin data.
            error[counter]=rmse(kxvalid,kyvalid,betaPre)
            counter=counter+1
```

3d line plot and scatter plot of each combination against respective RMSE value when tested on validation data follows:

Abdul Rehman Liaqat



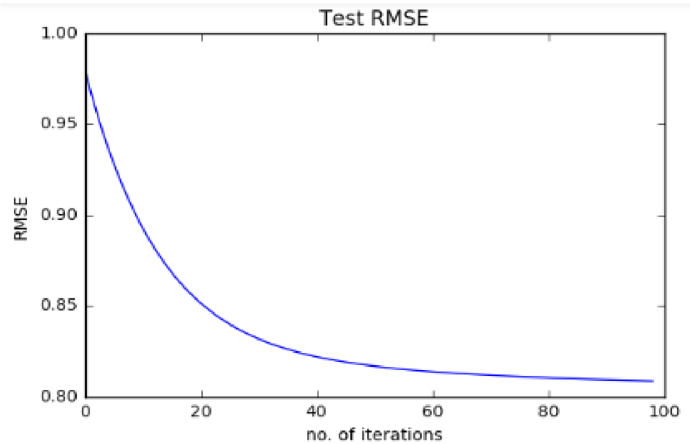


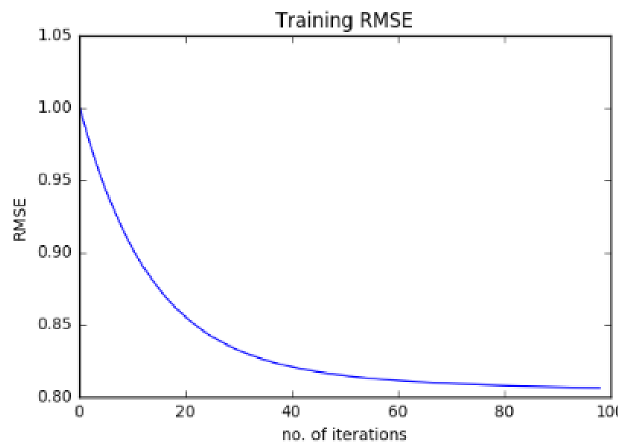
Abdul Rehman Liaqat

After availing RMSE tested on validation data of each combination of hyper parameters following were found to perform the best:

```
Best RMSE tested on validation data =  0.662684250555
best combination of hyper parameter:  Lambda =  8.5  Step Size =  0.09
```

After using these hyperparameter to train the training data we get following RMSE of training and test data.





And according to this model our minimum RMSE are following:

```
Best Training data RMSE  =    0.805835426055
Best Test data RMSE  =  0.808468492828
```

If we compare these two graphs with all the 9 plots of the previous exercise it becomes obvious that these parameters are one of the few combinations which provides minimum values.