Lab Course Machine Learning
Exercise 5

1.1 Data preprocessing:

Function name : read_data
Parameter : Filename and column names

This function reads the data and returns the data in the format of data frame along with the headings.

```
def read_data(filename,columns):
    if columns == None:
        data = pd.read_csv(filename, sep='\s+', delimiter = ";")
        return data
    data = pd.read_csv(filename, sep='\s+',header = None)
    data.columns = columns
    return data
```

Output:

Bank data

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown	no
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure	no
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure	no
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown	no
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown	no

Wine data

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
706	7.0	0.780	0.08	2.0	0.093	10.0	19.0	0.99560	3.40	0.47	10.0	5
67	6.6	0.705	0.07	1.6	0.076	6.0	15.0	0.99620	3.44	0.58	10.7	5
1089	11.6	0.410	0.54	1.5	0.095	22.0	41.0	0.99735	3.02	0.76	9.9	7
382	8.3	0.260	0.42	2.0	0.080	11.0	27.0	0.99740	3.21	0.80	9.4	6
589	10.2	0.290	0.49	2.6	0.059	5.0	13.0	0.99760	3.05	0.74	10.5	7

Function name : check_na_null
Parameter : Dataframe and column names

This function checks the presence of null values or NA or nan values within the dataset and removes the same. This function also generates dummies for non-numeric values and returns the same.

```
def check_na_null(data,column_dummies):
    data.isnull().values.any()
    data.dropna(inplace = True)
    data = pd.get_dummies(data=data, columns=column_dummies)
    return data
```

Output:

Dummies and nan or NA value check for Bank data

	age	balance	campaign	pdays	previous	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	...	education_unknown	default_no	default_yes
712	28	102	2	-1	0	0	0	0	0	0	...	0	1	0
3177	32	11797	2	-1	0	0	0	0	0	0	...	0	1	0
4029	37	487	15	-1	0	0	0	0	0	0	...	0	1	0
1074	28	594	3	-1	0	0	0	0	0	0	...	0	1	0
4117	25	8	2	-1	0	1	0	0	0	0	...	0	1	0

5 rows × 34 columns

Function name : text_to_number
Parameter : Dataframe and column names

This function converts the text within a column to numbers. For example, a city with name ABQ will be converted to a specific number for all the available instances.

```
def text_to_number(data,column):
    for i in column:
        Airdata[i] = pd.Categorical(Airdata[i])
        Airdata[i] = Airdata[i].cat.codes
    return data
```

Function name : normalise_data
Parameter : Data to be normalised

Reference: *http://mathworld.wolfram.com/FrobeniusNorm.html*

Numpy.linalg.norm function returns the normalisation of the data. By default, it returns Forbenius norm of the matrix.

$$A_{norm} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|A_{i,j}|^2}$$

Normalised bank data

	age	balance	campaign	pdays	previous	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	...	education_unknown	default_n
3385	0.013296	0.000000	0.010674	0.013116	0.025092	0.043811	0.0	0.000000	0.0	0.0	...	0.0	0.01499
4394	0.015045	0.048746	0.007116	-0.000138	0.000000	0.000000	0.0	0.032513	0.0	0.0	...	0.0	0.01499
4367	0.010147	0.018810	0.003558	-0.000138	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.0	0.01499
979	0.019594	0.013498	0.007116	-0.000138	0.000000	0.000000	0.0	0.032513	0.0	0.0	...	0.0	0.01499
1427	0.017144	0.001600	0.010674	-0.000138	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.0	0.01499

Normalilsed Wine data

	volatile acidity	citric acid	density	sulphates	alcohol	quality
706	0.034998	0.005995	0.024979	0.017295	0.023869	0.021962
67	0.031633	0.005246	0.024994	0.021342	0.025539	0.021962
1089	0.018396	0.040469	0.025023	0.027966	0.023630	0.030746
382	0.011666	0.031476	0.025024	0.029438	0.022436	0.026354
589	0.013012	0.036722	0.025029	0.027230	0.025062	0.030746

Function name : create_Test_Train_data
Parameter : Data of input variables and target

This function splits the input variables to Train and Test data with 80% and 20% of the entire data respectively.

```
def create_Test_Train_data(X_data,Y_data):
    Y_train = Y_data[:math.ceil(0.8*len(Y_data))]
    Y_test = Y_data[math.ceil(0.8*len(Y_data)):]
    X_train = X_data[:math.ceil(0.8*len(X_data))]
    X_test = X_data[math.ceil(0.8*len(X_data)):]
    return Y_train,Y_test,X_train,X_test
```

Output:

Bank data

```
Xtrain:  (3617, 33)
Ytrain:  (3617,)
Xtest:  (904, 33)
Ytest:  (904,)
```

Wine data

```
Xtrain:  (1280, 5)
Ytrain:  (1280,)
Xtest:  (319, 5)
Ytest:  (319,)
```

Regularization:

Bank dataset:

To begin with the problem, first we've to find the correlation between the input variables and the client subscription of a term deposit. As the dataset contains both categorical and numerical values, we can use the spearman correlation coefficient which uses ranked variables and access them with pearson co-efficient correlation between two variables.

Function name : spearman_rank_coefficient
Parameter : Data for correlation

Reference: *https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php*

The Spearman's rank-order correlation is the nonparametric version of the Pearson product-moment correlation. Spearman's correlation coefficient, (ρ, also signified by rs) measures the strength and direction of association between two ranked variables.

Assumption: You need two variables that are either ordinal, interval or ratio

$$r_s = \rho_{rg_X, rg_Y} = \frac{cov(rg_x, rg_y)}{\sigma_{rg_X, rg_Y}}$$

Function name : pearson_coefficient
Parameter : Ranked data for correlation

Reference: *https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php*

The Pearson correlation coefficient (PCC) is a measure of the linear correlation between two variables X and Y. PCC or 'r' has a value between +1 and −1.

Assumptions:

1.The variables must be either interval or ratio measurements.
2.The variables must be approximately normally distributed.
3.There is a linear relationship between the two variables
4.Outliers are either kept to a minimum or are removed entirely.
5.There is homoscedasticity of the data.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

```
def pearson_coefficient(x,y):
    x = x - np.mean(x)
    y = y - np.mean(y)
    return (np.sum(x*y)/np.sqrt(np.sum(x*x)*np.sum(y*y)))
```

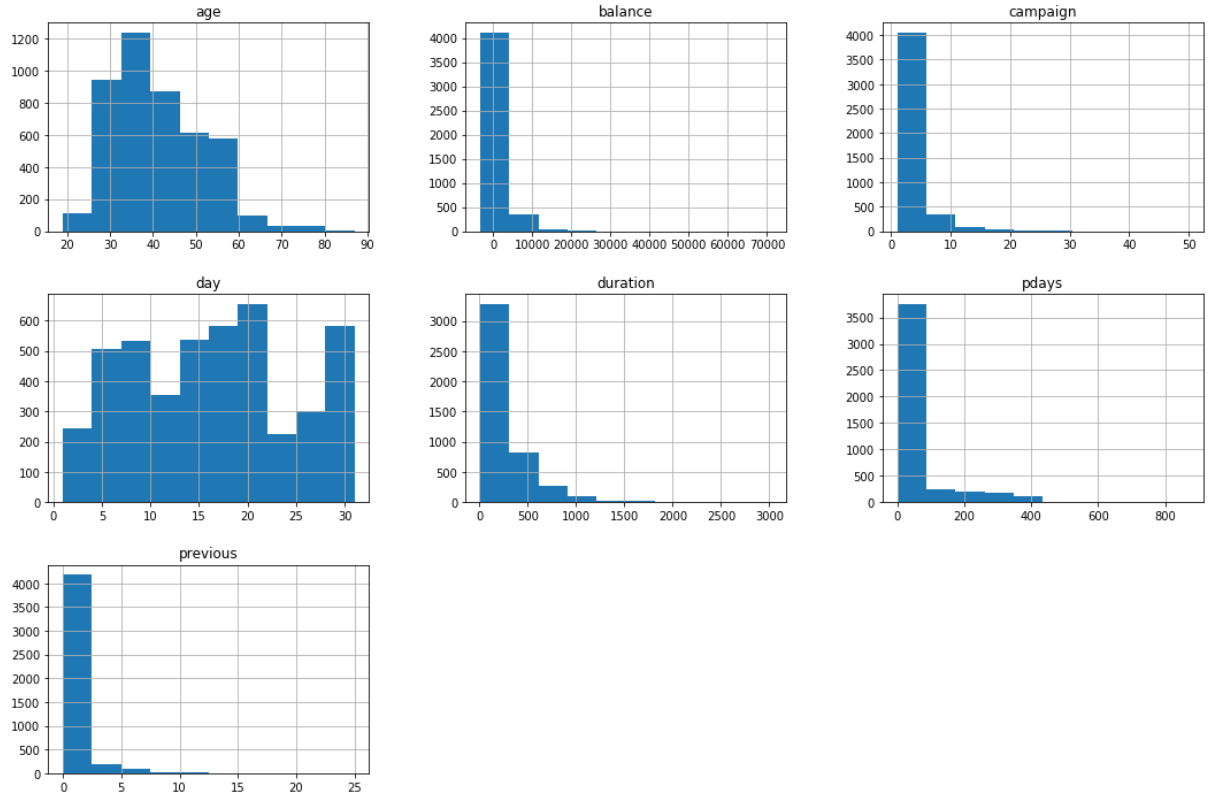
Output:

Correlation between subscription of term deposit and other variables for the bank data

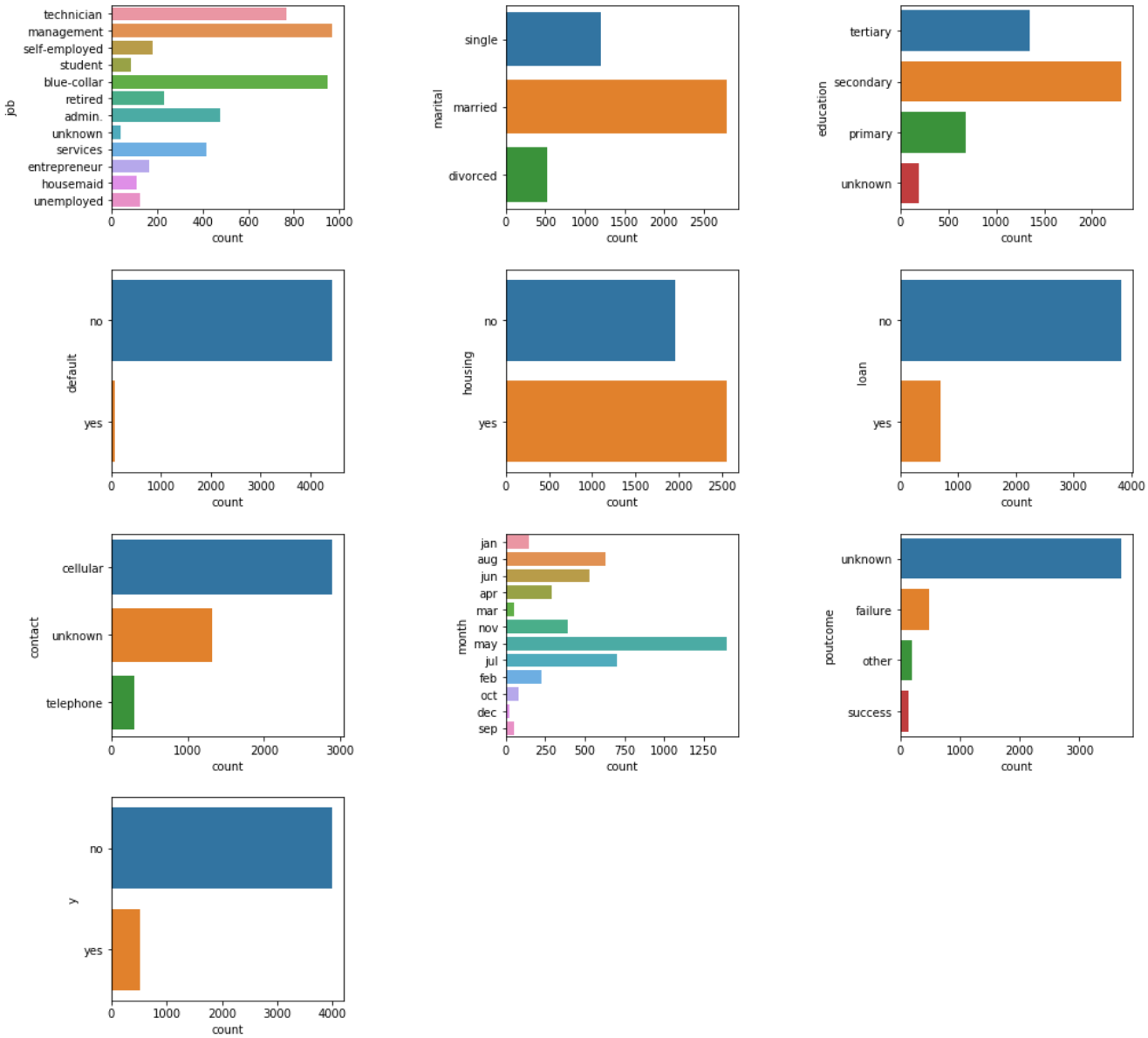
	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	0.0163	0.03	0.0193	0.0481	0.0013	0.079	-0.1047	-0.0705	-0.1295	-0.0118	-0.0326	0.3484	-0.0646	0.1502	0.1653	-0.1413	NaN

From the correlation values between the subscription term deposit and other variables, I'm dropping the columns of day, month, housing, loan and duration from the dataset. The reason why I am dropping duration column is that at the end of the phone call, it is obvious that we get the class of the subscription. Hence it can be dropped.

Visualization of numerical columns of the bank dataset



Visualization of categorical values in the bank dataset




```
Required_bankdata = Bankdata.drop(columns = ['day','month','housing','loan','duration'])
```

Output:

Data after dropping the poorly correlated columns

	age	job	marital	education	default	balance	contact	campaign	pdays	previous	poutcome	y
1143	57	housemaid	married	primary	no	501	cellular	2	-1	0	unknown	0
4471	59	management	married	unknown	no	3534	cellular	4	-1	0	unknown	0
4078	30	management	married	tertiary	no	562	cellular	4	-1	0	unknown	0
1646	38	technician	single	tertiary	no	2273	cellular	1	-1	0	unknown	0
869	56	technician	married	secondary	no	150	cellular	6	349	1	failure	0

Normalised bank data

	age	balance	campaign	pdays	previous	y	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	...	education_unknown	default_n
3385	0.013296	0.000000	0.010674	0.013116	0.025092	0.043811	0.0	0.000000	0.0	0.0	...	0.0	0.01499
4394	0.015045	0.048746	0.007116	-0.000138	0.000000	0.000000	0.0	0.032513	0.0	0.0	...	0.0	0.01499
4367	0.010147	0.018810	0.003558	-0.000138	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.0	0.01499
979	0.019594	0.013498	0.007116	-0.000138	0.000000	0.000000	0.0	0.032513	0.0	0.0	...	0.0	0.01499
1427	0.017144	0.001600	0.010674	-0.000138	0.000000	0.000000	0.0	0.000000	0.0	0.0	...	0.0	0.01499

To begin with the model generation, we have to initialise zero values for beta depending on the number of input variables

```
beta = np.zeros(X_data.shape[1] + 1)
```

Output:

```
(array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 (34,))
```

Function name : loglikelihood
Parameter : Y data and dot product of beta and X

This function returns the loglikelihood of the function to be minimized. This is the function that we've to check for convergence.

$$logL_D(\hat{\beta}) = \sum_{n=1}^{N} y_n(x_n.\hat{\beta}) - log(1 + e^{(x_n.\hat{\beta})})$$

```
def loglikelihood(Y,betaX):
    return  np.sum(np.dot(Y,betaX) - np.log(1 + np.exp(betaX)))
```

Function name : logloss
Parameter : Test data of X,Y and beta values

This function returns the logarithmic loss of the test data from the predicted beta values from one of the classification methods.

Reference: http://wiki.fast.ai/index.php/Log_Loss#Binary_Classification

In a binary classification (M=2), the formula equals:

$$-y(log(p)) + (1 - y)(log(1 - p))$$

```
def logloss(Ytest,Xtest,beta):
    X = np.insert(Xtest, 0, 1, axis = 1)
    betaX = np.dot(beta,X.T)
    Y_pred = 1/(1 + np.exp(-betaX))
    return -np.sum(Ytest * np.log(Y_pred) + (1 - Ytest)
                   * np.log(1 - Y_pred))/len(Ytest)
```

Function name : ridgered_GD
Parameter : training and testing data of X and Y, step length, regularization value, batch size

This function performs the ridge regression using mini Batch Gradient Descent algorithm and calls the function for the same. In mini BGD, the training data is split into batches and is used for training. In our case, we've chosen a batch size of 50. Hence when all the batches are run through an epoch, the process is continued for the maximum epochs. With every batch, the model gets trained and the beta parameter gets updated per iteration and the corresponding log loss for train and test set are recorded for future plotting of the graph.

```
def ridgereg_gd(Y_train,Y_test,X_train,X_test,alpha,reg,minibatchsize,ttype):
    initialbeta = np.zeros(X_data.shape[1] + 1)
    train_graph = list()
    test_graph = list()
    if ttype!='Tune':
        for i in range(0,len(reg)):
            for j in range(0,len(alpha)):
                logloss_train,logloss_test,beta = ridgereg_mini_bgd(alpha[j],i
                train_graph.append(logloss_train)
                test_graph.append(logloss_test)
        return train_graph,test_graph
    else:
        logloss_train,logloss_test,beta = ridgereg_mini_bgd(alpha,initialbeta,
        return logloss_train,logloss_test
```

Function name : ridgered_mini_bgd
Parameter : training and testing data of X and Y, step length, regularization value, batch size

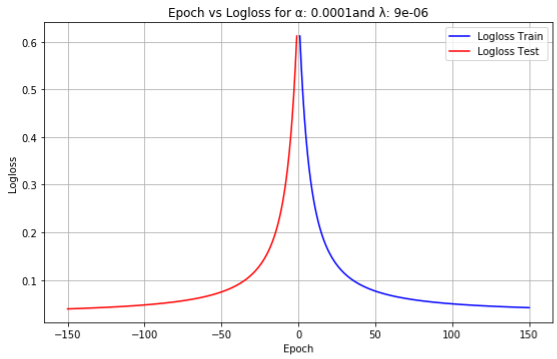
```
def ridgereg_mini_bgd(alpha,beta,minibatchsize,reg,Y_train,Y_test,X_train,X_test,steplength_type):
    X = np.insert(X_train, 0, 1, axis = 1)
    logloss_train = []
    logloss_test = []
    epochs = 150
    history = [0] * len(beta)
    betaX = np.dot(beta,X.T)
    likelihood = loglikelihood(Y_train,betaX)
    batches = math.ceil(len(X)/minibatchsize)
    for iteration in range(0,epochs):
        for i in range(0,batches):
            if i*50+50 < len(X):

                batch_X = X[i*50:i*50+50]
                batch_Y = Y_train[i*50:i*50+50]
            else:

                batch_X = X[i*50:]
                batch_Y = Y_train[i*50]
            func_gradient = - np.dot(batch_X.T,batch_Y - 1/(1 + np.exp(-(np.dot(beta,batch_X.T)))))
            likelihood_new = loglikelihood(batch_Y,np.dot(beta,batch_X.T))
            beta = beta - alpha * (func_gradient - 2*reg*beta)
            betaX = np.dot(beta,batch_X.T)
        if (steplength_type == "Bold"):
            alpha = bold_steplength(alpha,likelihood_new,likelihood)
        likelihood = likelihood_new
        logloss_train.append(logloss(Y_train,X,beta,'Train'))
        logloss_test.append(logloss(Y_test,X_test,beta,'Test'))
    return logloss_train,logloss_test,beta
```

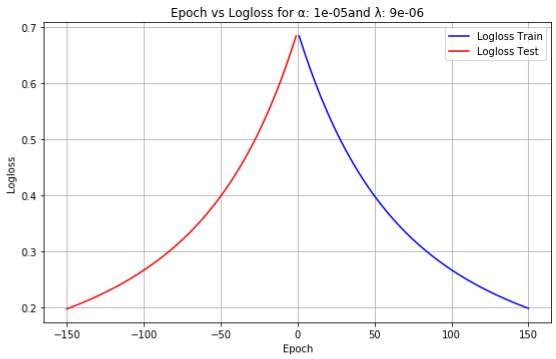
Initializations:

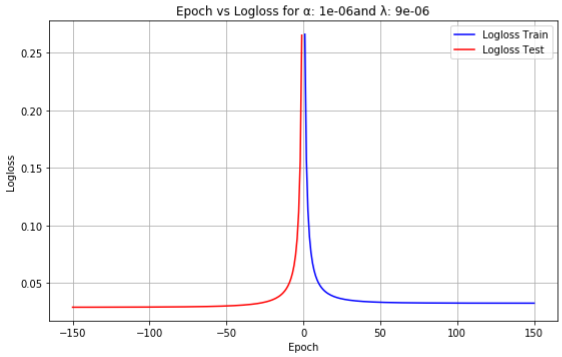
We use the following constant step length, regularization parameters and batch size for training

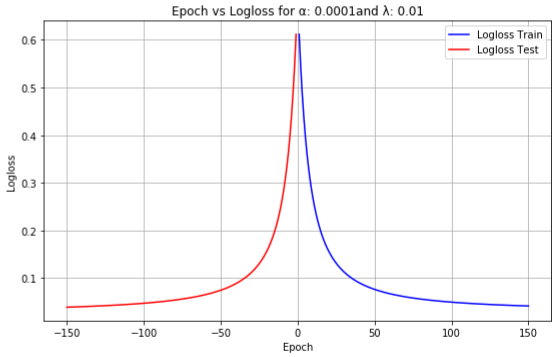
```
alpha = [1e-4,1e-5,1e-3]
reg = [9e-6,0.01,1e-4]
minibatchsize = 50
```

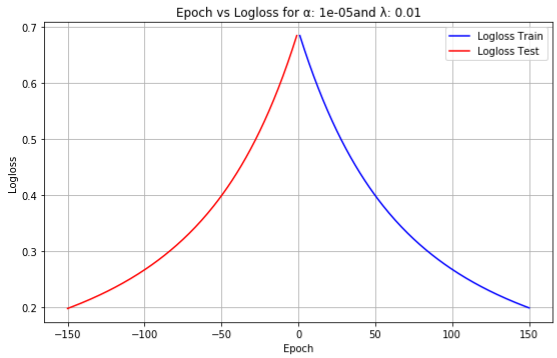
The following are the recorded Logloss train and test data for all the possible combinations.

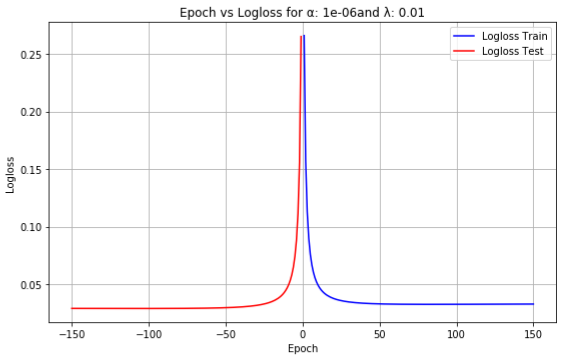


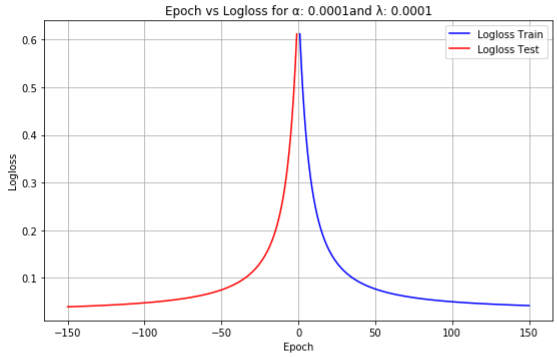


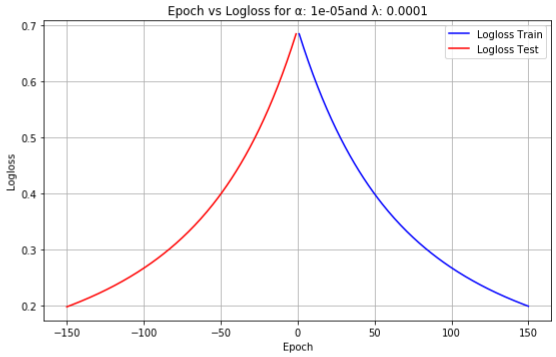


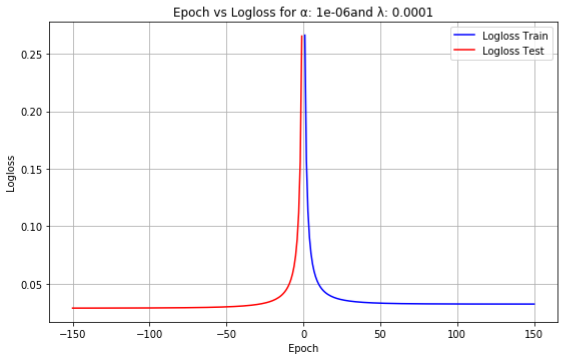


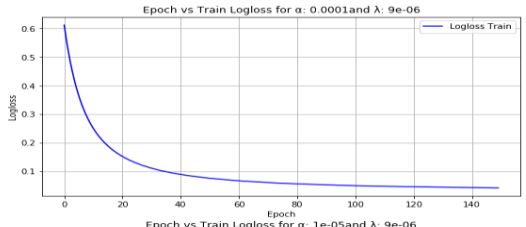


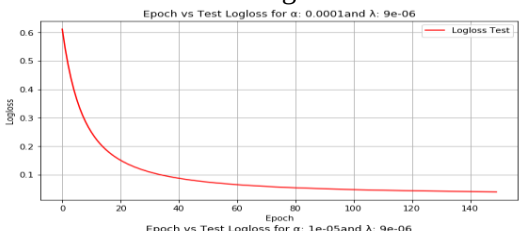


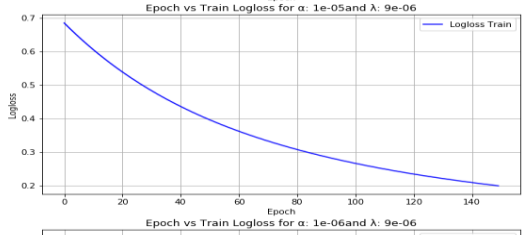


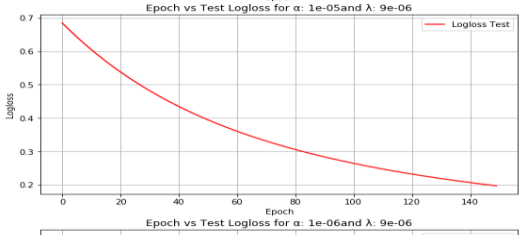


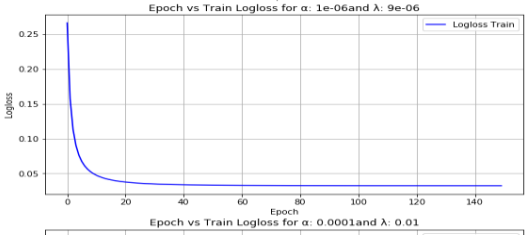


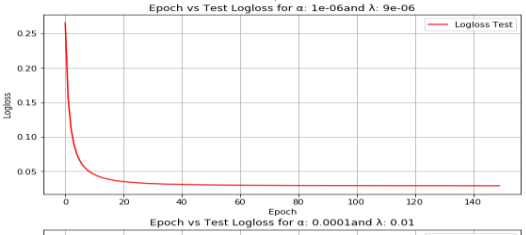

Epoch vs Train Logloss for α: 0.0001and λ: 9e-06
Logloss Train
Logloss
Epoch

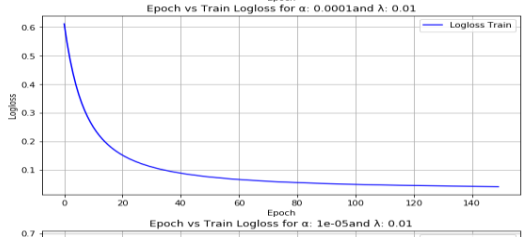

Epoch vs Test Logloss for α: 0.0001and λ: 9e-06
Logloss Test
Logloss
Epoch

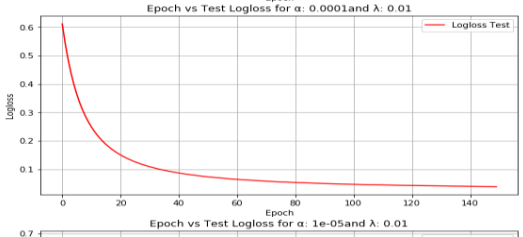

Epoch vs Train Logloss for α: 1e-05and λ: 9e-06
Logloss Train
Logloss
Epoch

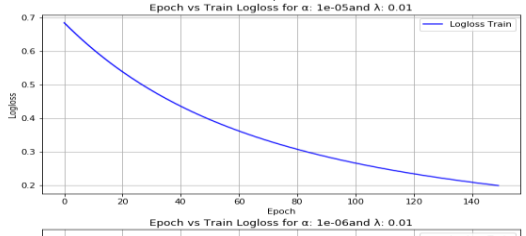

Epoch vs Test Logloss for α: 1e-05and λ: 9e-06
Logloss Test
Logloss
Epoch

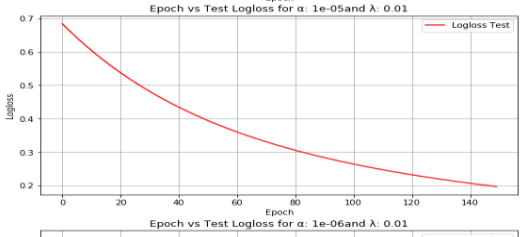

Epoch vs Train Logloss for α: 1e-06and λ: 9e-06
Logloss Train
Logloss
Epoch

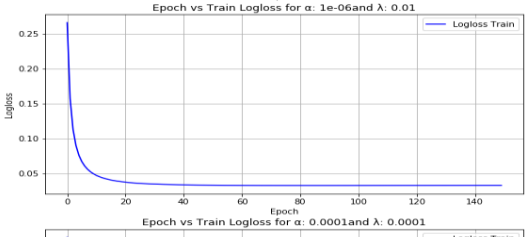

Epoch vs Test Logloss for α: 1e-06and λ: 9e-06
Logloss Test
Logloss
Epoch

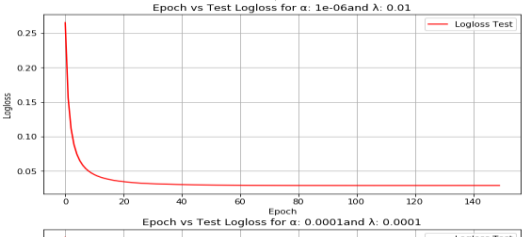

Epoch vs Train Logloss for α: 0.0001and λ: 0.01
Logloss Train
Logloss
Epoch

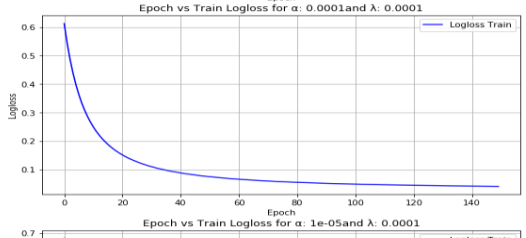

Epoch vs Test Logloss for α: 0.0001and λ: 0.01
Logloss Test
Logloss
Epoch

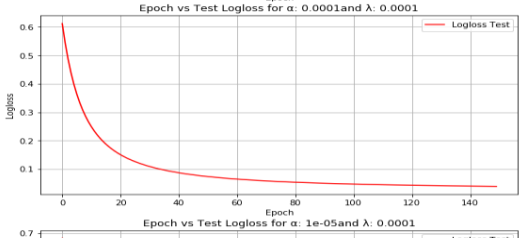

Epoch vs Train Logloss for α: 1e-05and λ: 0.01
Logloss Train
Logloss
Epoch

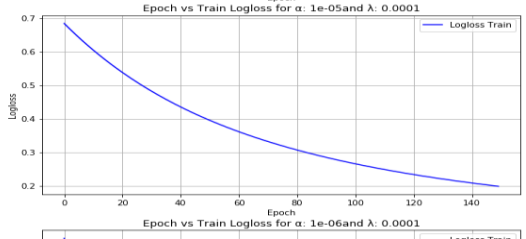

Epoch vs Test Logloss for α: 1e-05and λ: 0.01
Logloss Test
Logloss
Epoch

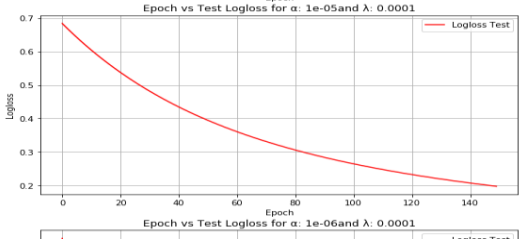

Epoch vs Train Logloss for α: 1e-06and λ: 0.01
Logloss Train
Logloss
Epoch

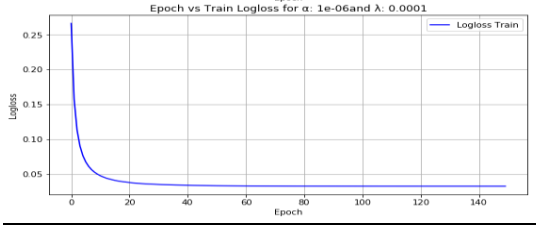

Epoch vs Test Logloss for α: 1e-06and λ: 0.01
Logloss Test
Logloss
Epoch

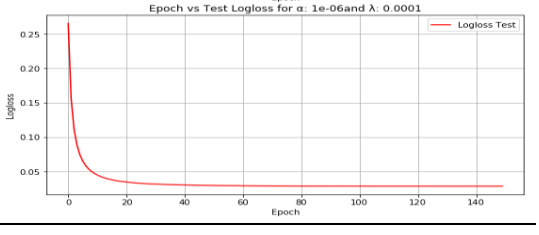

Epoch vs Train Logloss for α: 0.0001and λ: 0.0001
Logloss Train
Logloss
Epoch


Epoch vs Test Logloss for α: 0.0001and λ: 0.0001
Logloss Test
Logloss
Epoch


Epoch vs Train Logloss for α: 1e-05and λ: 0.0001
Logloss Train
Logloss
Epoch


Epoch vs Test Logloss for α: 1e-05and λ: 0.0001
Logloss Test
Logloss
Epoch


Epoch vs Train Logloss for α: 1e-06and λ: 0.0001
Logloss Train
Logloss
Epoch


Epoch vs Test Logloss for α: 1e-06and λ: 0.0001
Logloss Test
Logloss
Epoch

Hyper-parameter tuning:

Function name : gridsearch
Parameter : training and testing data of X and Y, step length, regularization value, batch size, folds

In this section I have implemented the grid search with k-fold cross-validation for model selection i.e. choosing best hyper parameters. I have used Ridge Regression using mini Batch Gradient Descent (mini-BGD) algorithm. The grid search function calls the k-fold validation function from which the log loss of all possible combinations are calculated and thereby plotted on the graph.

```
alpha = [1e-4,5.2e-5,3.7e-3,2e-7]
reg = [9e-16,0.01,1e-4,0.1,1e-9]
```

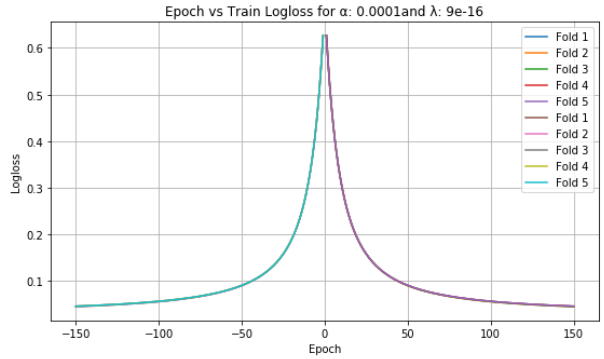
```
def gridsearch(Y_train,X_train,alpha,reg,minibatchsize,fold):

    train_graph = list()
    valid_graph = list()
    logloss_train = list()
    logloss_valid = list()
    for i in range(0,len(reg)):
        for j in range(0,len(alpha)):
            train_graph,valid_graph = kfold(X_train,Y_train,alpha[j],reg[i],minibatchsize,fold)
            logloss_train.append(train_graph)
            logloss_valid.append(valid_graph)
    return logloss_train,logloss_valid
```

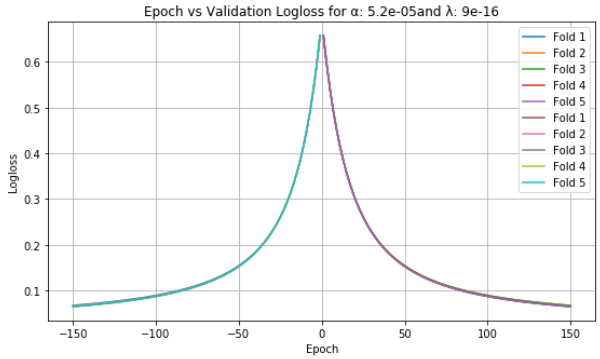
Function name : kfold
Parameter : training data of X and Y, step length, regularization value, batch size, folds

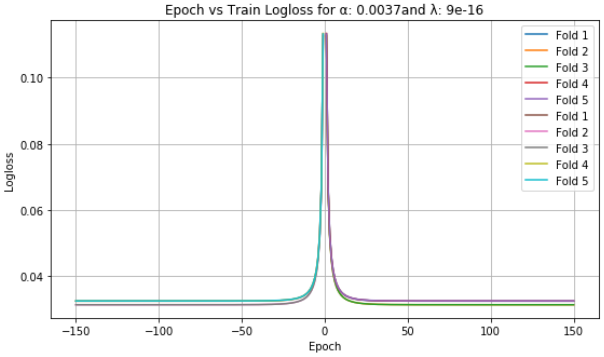
The k-fold cross validation method uses the concept of validating the trained model before building it up. Here, the entire training data is split to k-folds and each time, one of the folds is used for validation and other folds for training. This process continues for k-times which inside uses the mini-BGD for training.

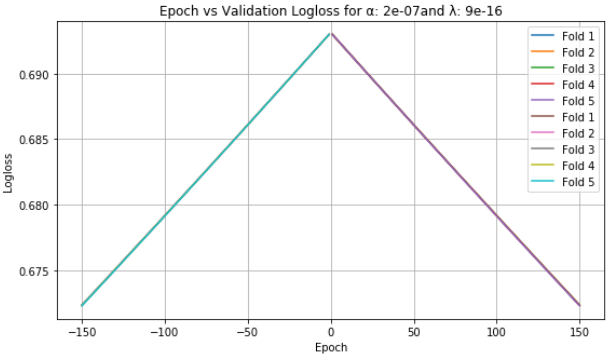
```
def kfold(X_train,Y_train,alpha,reg,minibatchsize,fold):
    split = math.ceil(len(X_train)/fold)
    logloss_train = list()
    logloss_valid = list()
    for i in range (0,fold):
        start = i*split
        end = i*split+split

        if start ==0:
            fold_Xtest = X_train[start:end]
            fold_Xtrain = X_train[end:]
            fold_Ytest = Y_train[start:end]
            fold_Ytrain = Y_train[end:]
            fold_Xtrainold = fold_Xtrain
        else:
            fold_Xtest = X_train[start:end]
            fold_Xtrain1 = X_train[0:start]
            fold_Xtrain2 = X_train[end:-1]
            fold_Xtrain = np.concatenate((fold_Xtrain1,fold_Xtrain2))
            fold_Ytest = Y_train[start:end]
            fold_Ytrain1 = Y_train[0:start]
            fold_Ytrain2 = Y_train[end:-1]
            fold_Ytrain = np.concatenate((fold_Ytrain1,fold_Ytrain2))
        train_graph,valid_graph = ridgereg_gd(fold_Ytrain,fold_Ytest,fold_
        logloss_train.append(train_graph)
        logloss_valid.append(valid_graph)
    return logloss_train,logloss_valid
```

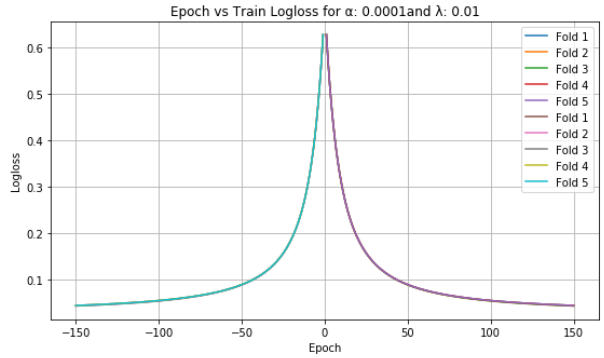
Plot for 5 folds with all possible combinations of steplength and regularization parameters

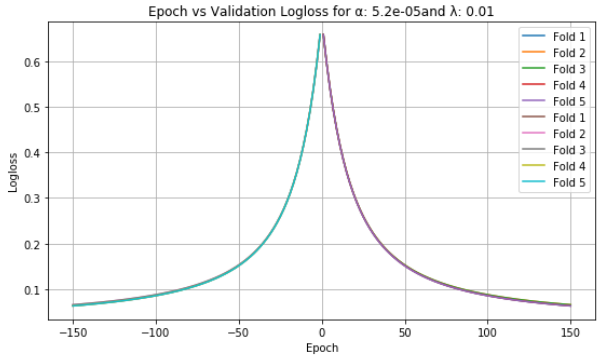


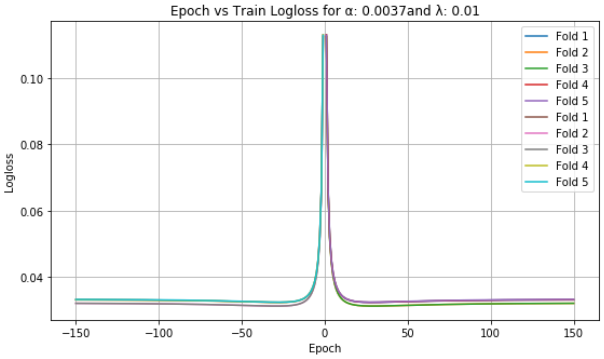


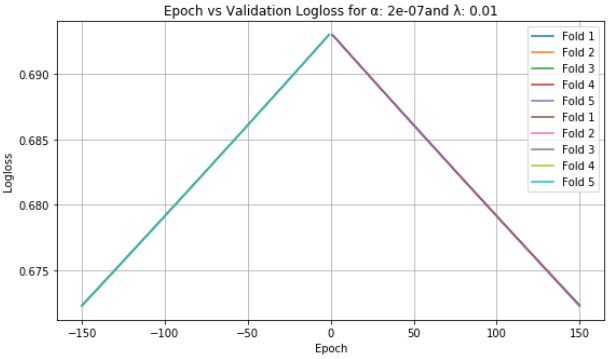


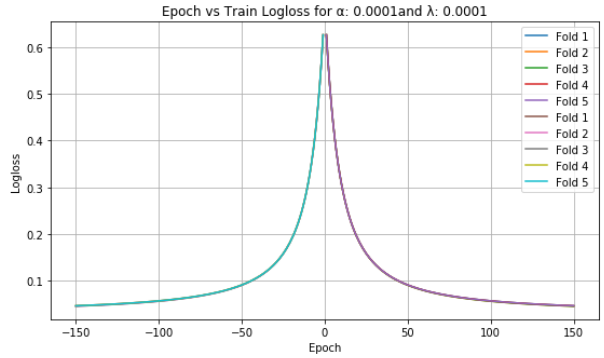


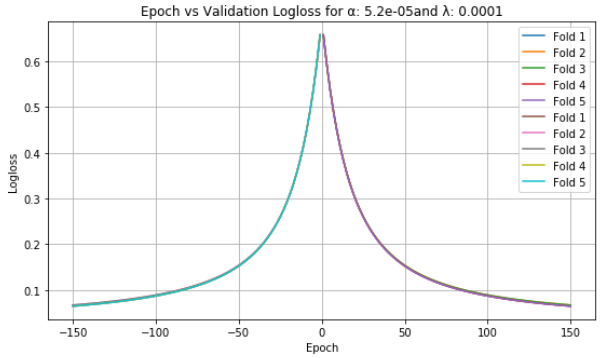


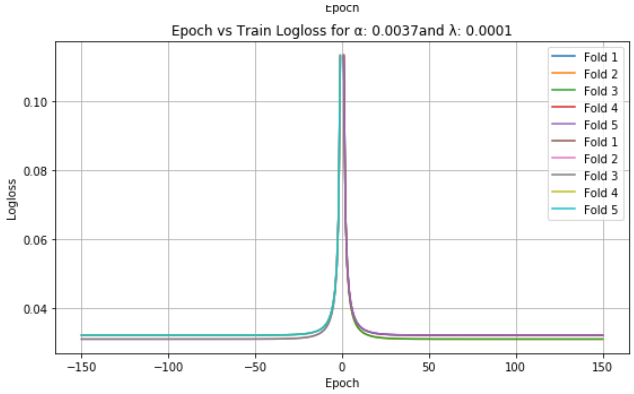


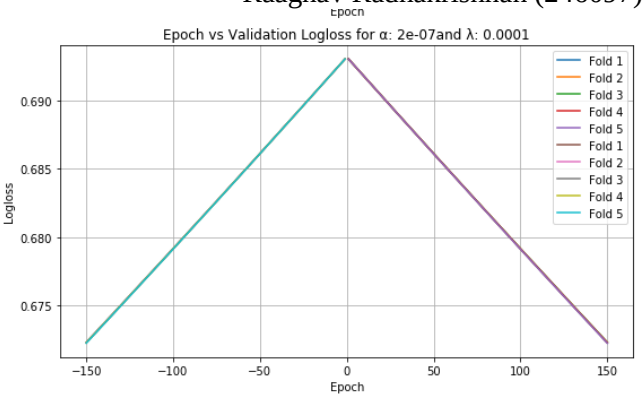


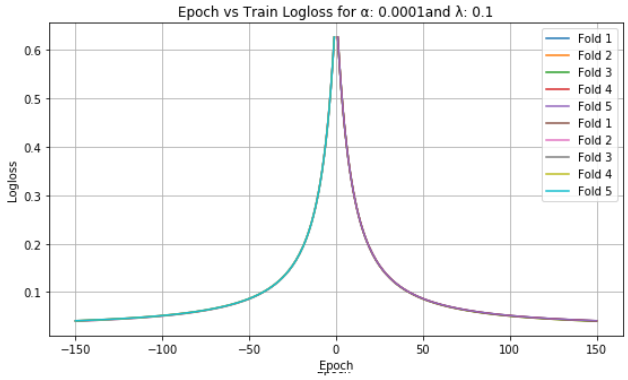


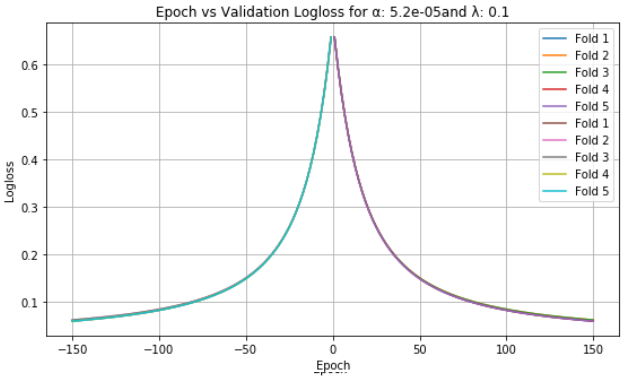


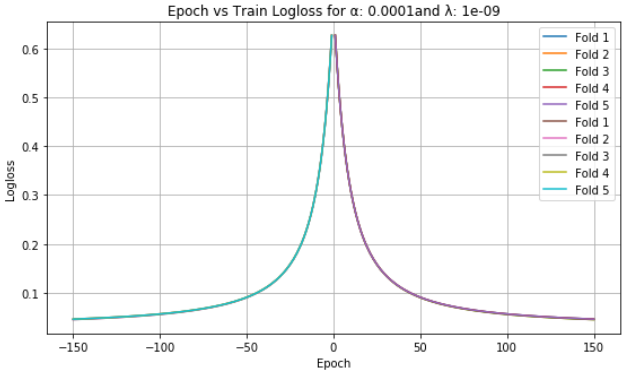


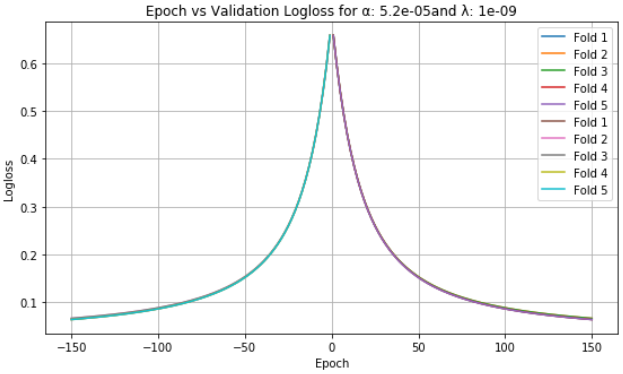

Epoch vs Train Logloss for α: 0.0037and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

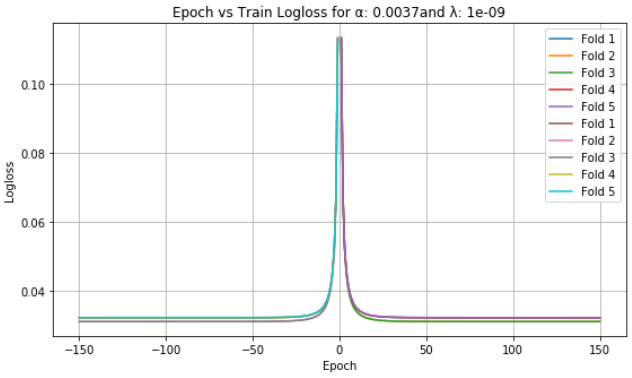

Epoch vs Validation Logloss for α: 2e-07and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

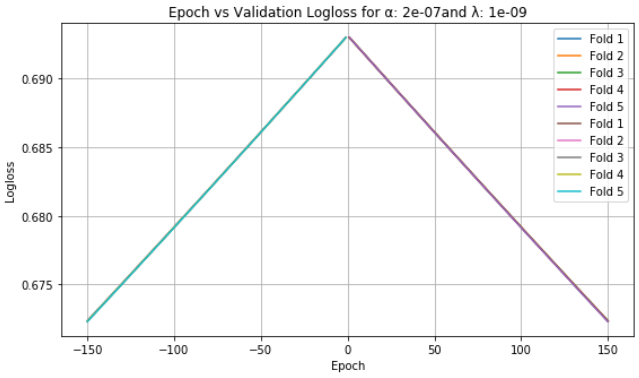

Epoch vs Train Logloss for α: 0.0001and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Validation Logloss for α: 5.2e-05and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Train Logloss for α: 0.0001and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

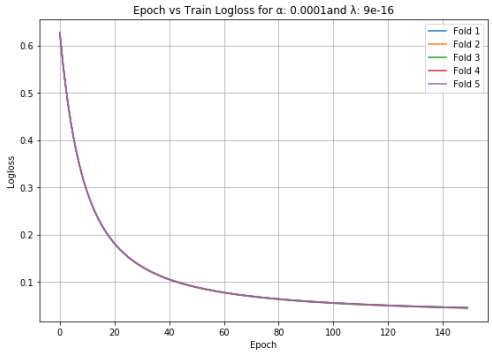

Epoch vs Validation Logloss for α: 5.2e-05and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

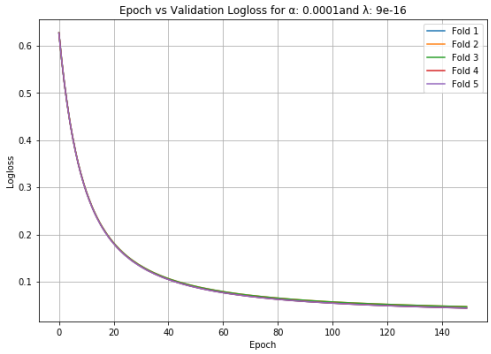

Epoch vs Train Logloss for α: 0.0037and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

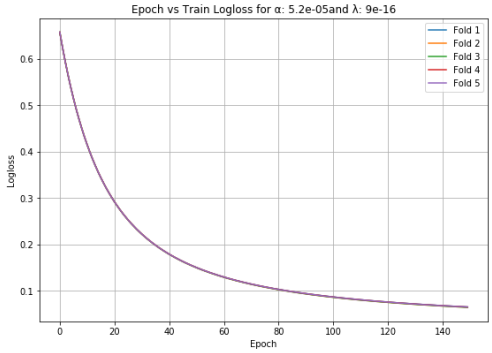

Epoch vs Validation Logloss for α: 2e-07and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

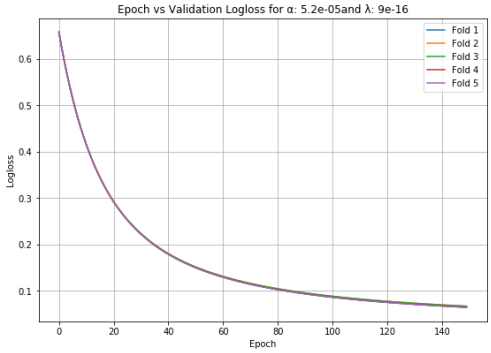

Epoch vs Validation Logloss for α: 2e-07and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

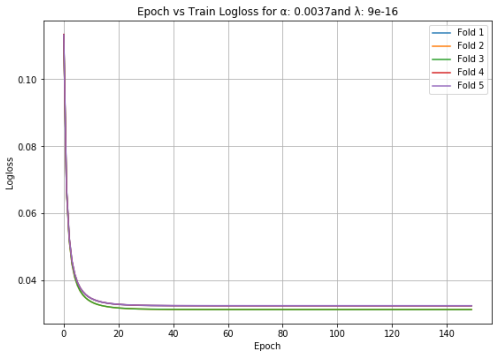
Plot with Epoch vs Train and Epoch vs Test

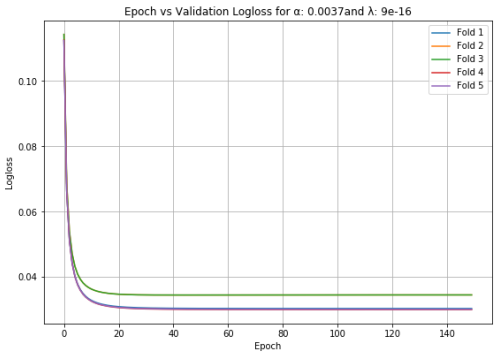


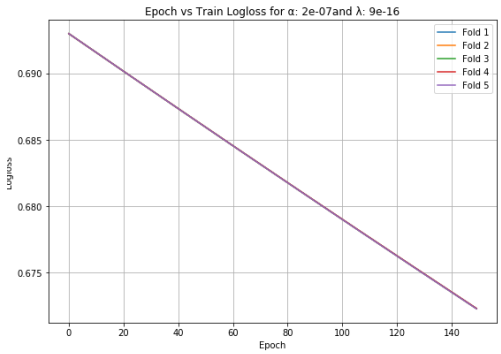


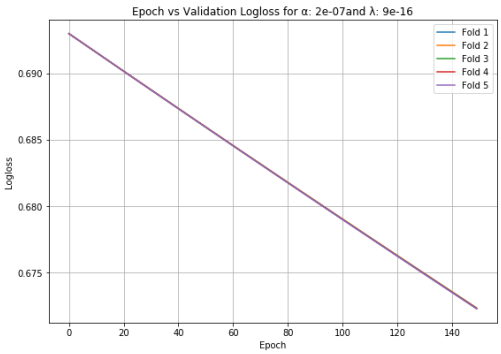


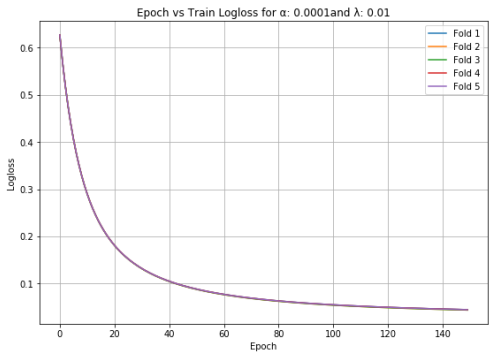


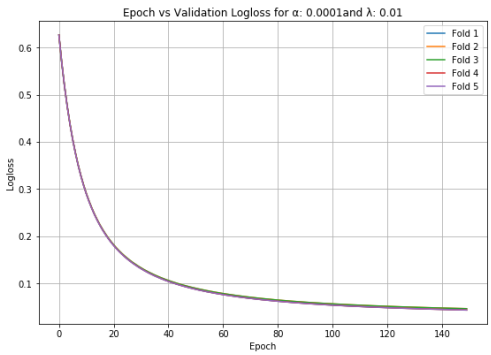


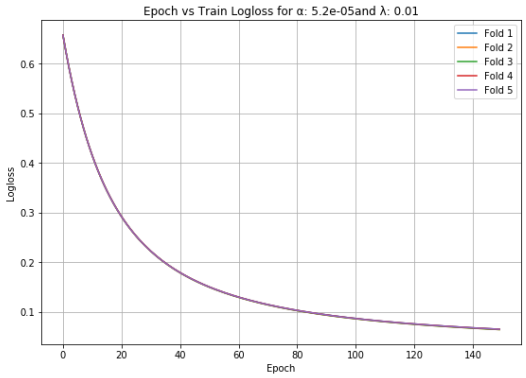


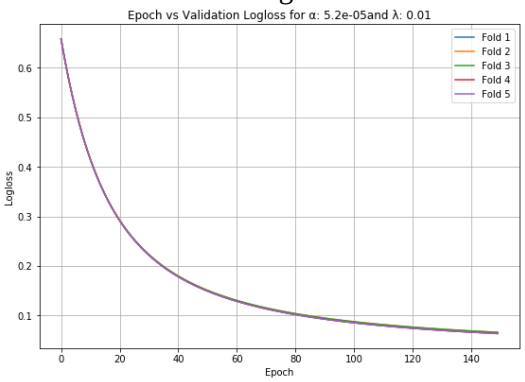


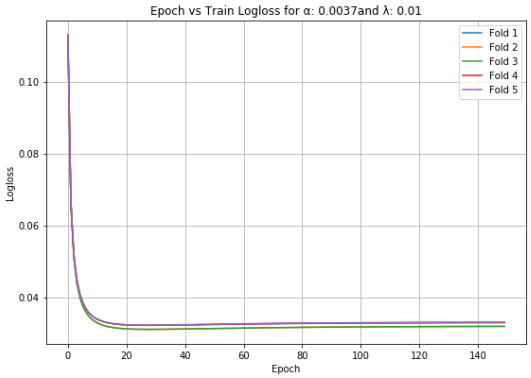


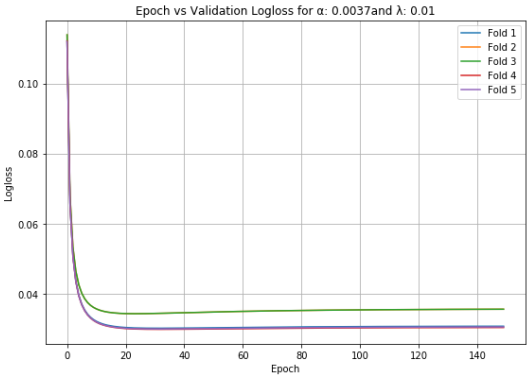


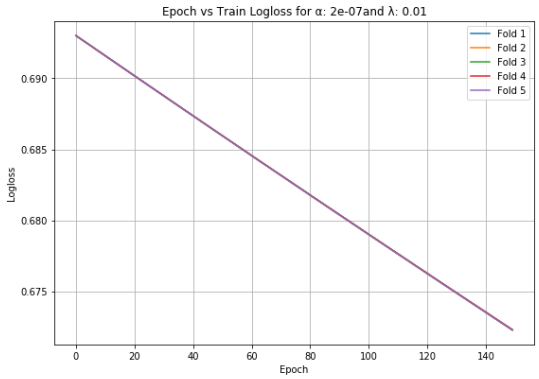


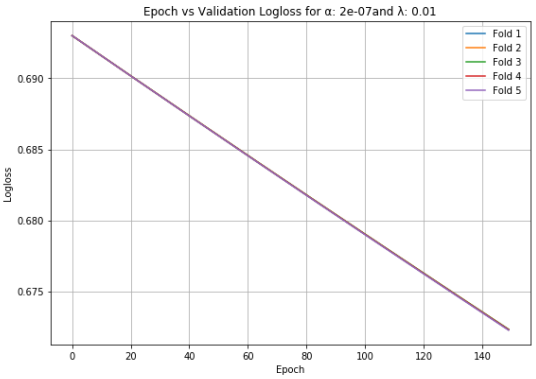

Epoch vs Train Logloss for α: 5.2e-05and λ: 0.01
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

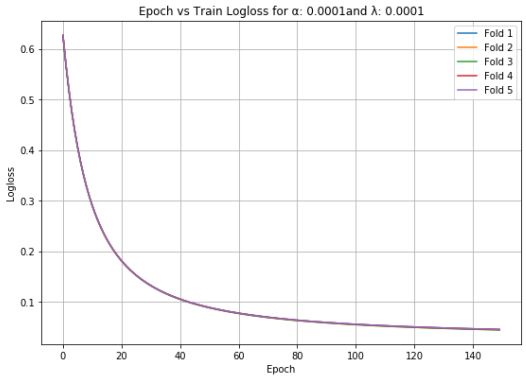

Epoch vs Validation Logloss for α: 5.2e-05and λ: 0.01
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

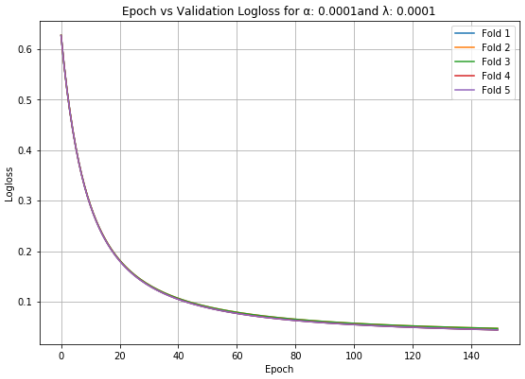

Epoch vs Train Logloss for α: 0.0037and λ: 0.01
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

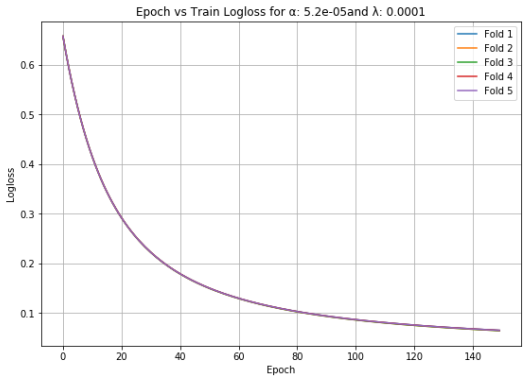

Epoch vs Validation Logloss for α: 0.0037and λ: 0.01
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

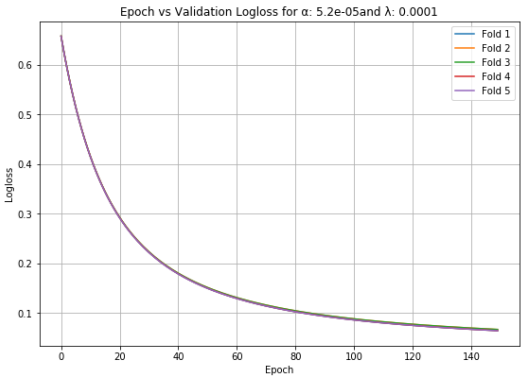

Epoch vs Train Logloss for α: 2e-07and λ: 0.01
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

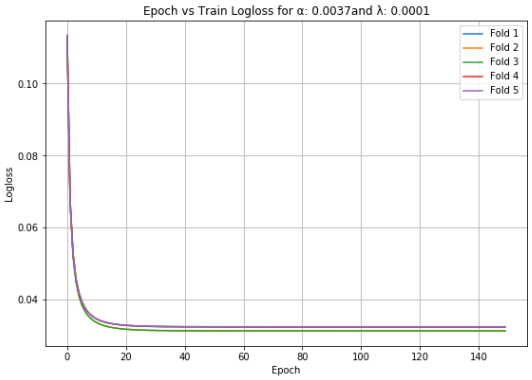

Epoch vs Validation Logloss for α: 2e-07and λ: 0.01
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

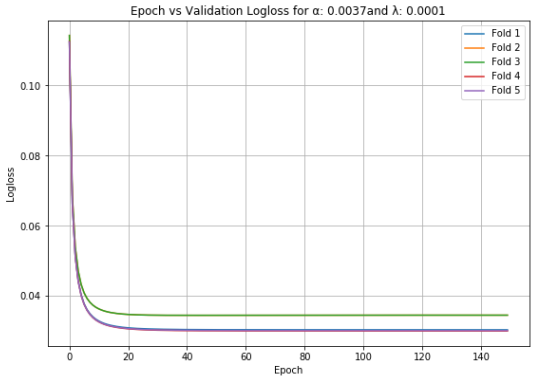

Epoch vs Train Logloss for α: 0.0001and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

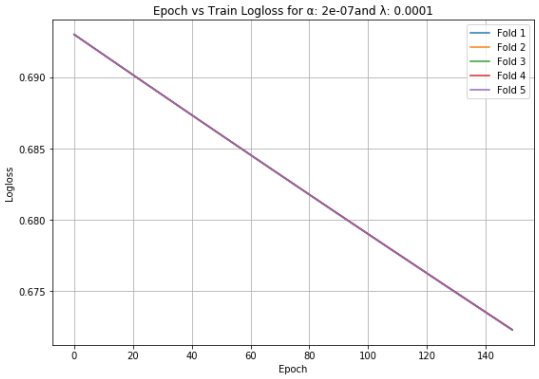

Epoch vs Validation Logloss for α: 0.0001and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

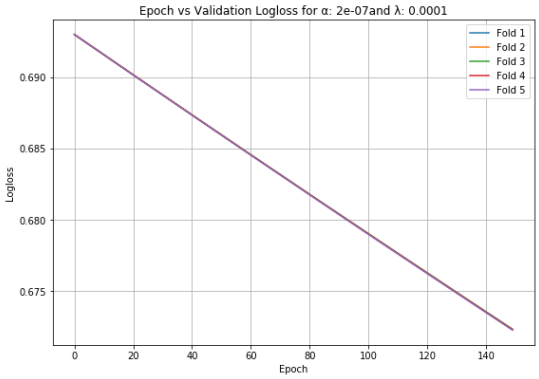

Epoch vs Train Logloss for α: 5.2e-05and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

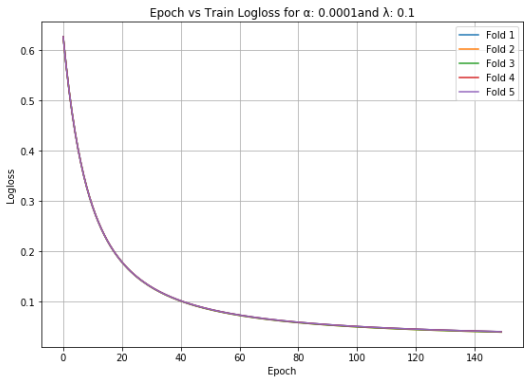

Epoch vs Validation Logloss for α: 5.2e-05and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

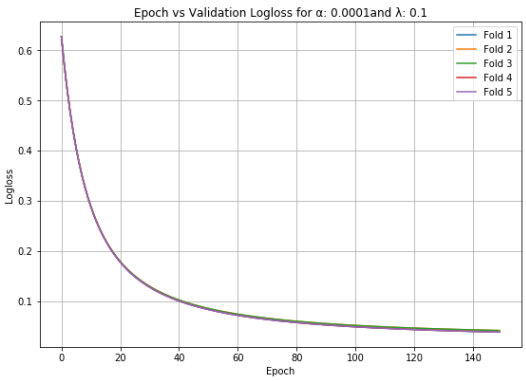

Epoch vs Train Logloss for α: 0.0037and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

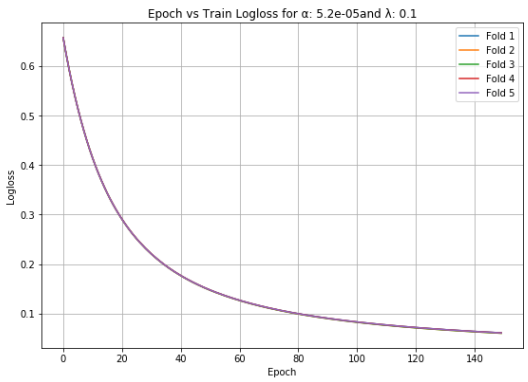

Epoch vs Validation Logloss for α: 0.0037and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

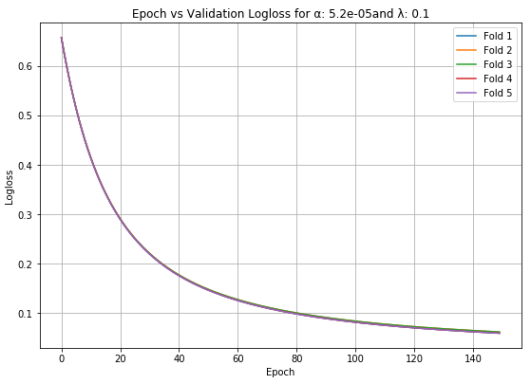

Epoch vs Train Logloss for α: 2e-07and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

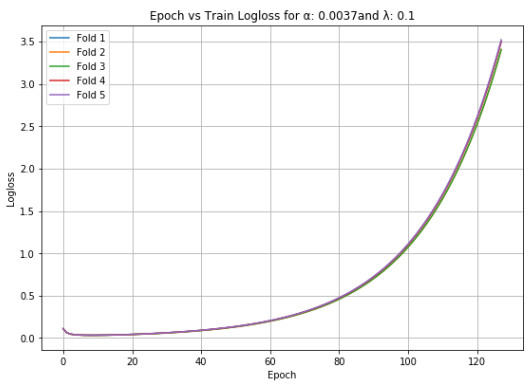

Epoch vs Validation Logloss for α: 2e-07and λ: 0.0001
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

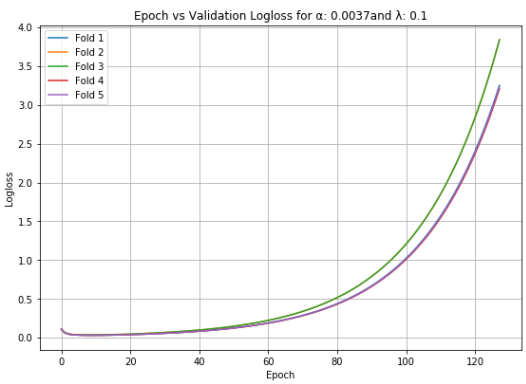

Epoch vs Train Logloss for α: 0.0001and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

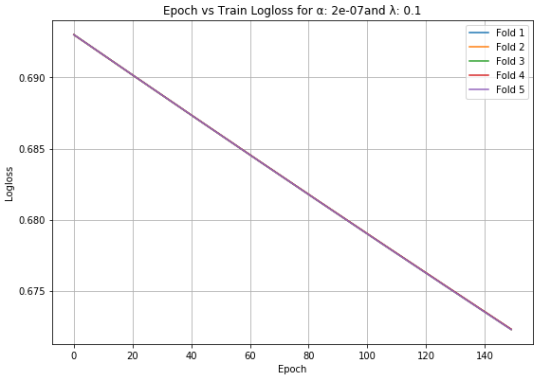

Epoch vs Validation Logloss for α: 0.0001and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

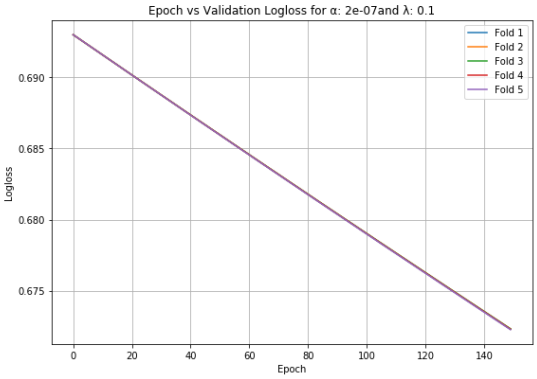

Epoch vs Train Logloss for α: 5.2e-05and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

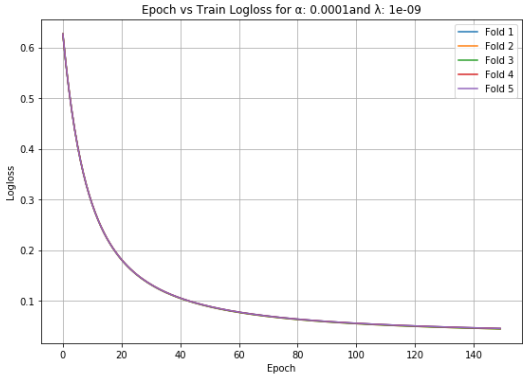

Epoch vs Validation Logloss for α: 5.2e-05and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

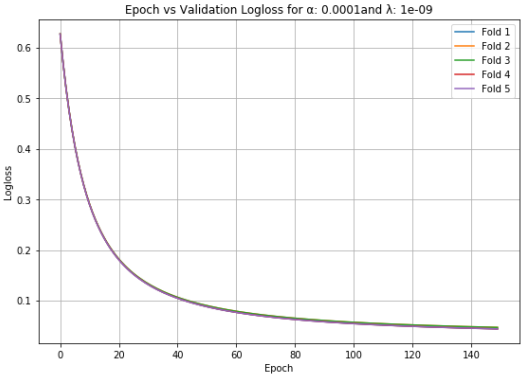

Epoch vs Train Logloss for α: 0.0037and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

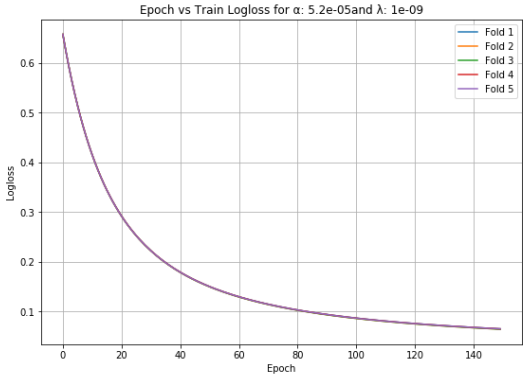

Epoch vs Validation Logloss for α: 0.0037and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

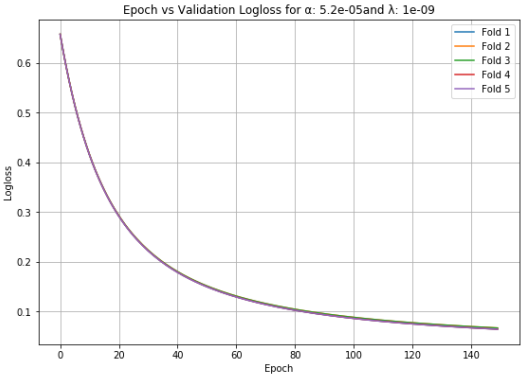

Epoch vs Train Logloss for α: 2e-07and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

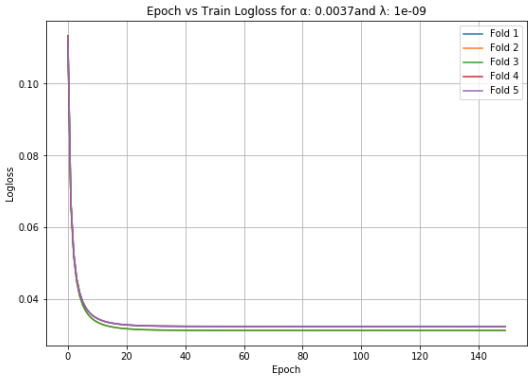

Epoch vs Validation Logloss for α: 2e-07and λ: 0.1
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

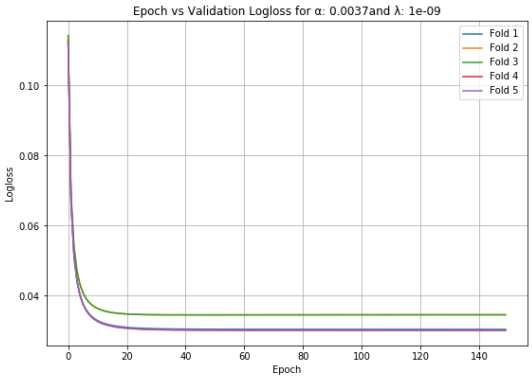

Epoch vs Train Logloss for α: 0.0001and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

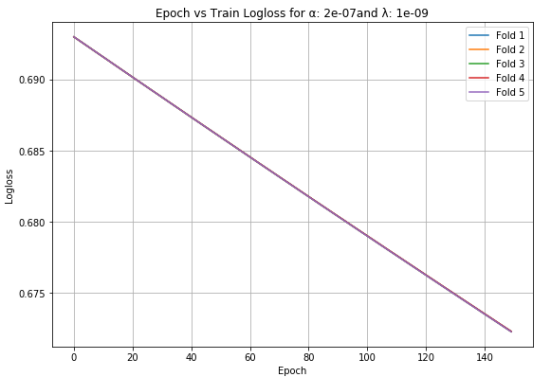

Epoch vs Validation Logloss for α: 0.0001and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

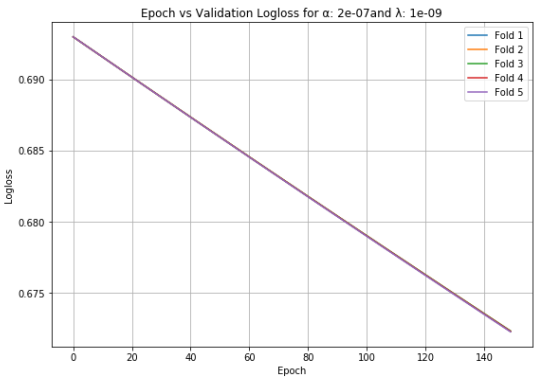

Epoch vs Train Logloss for α: 5.2e-05and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Validation Logloss for α: 5.2e-05and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

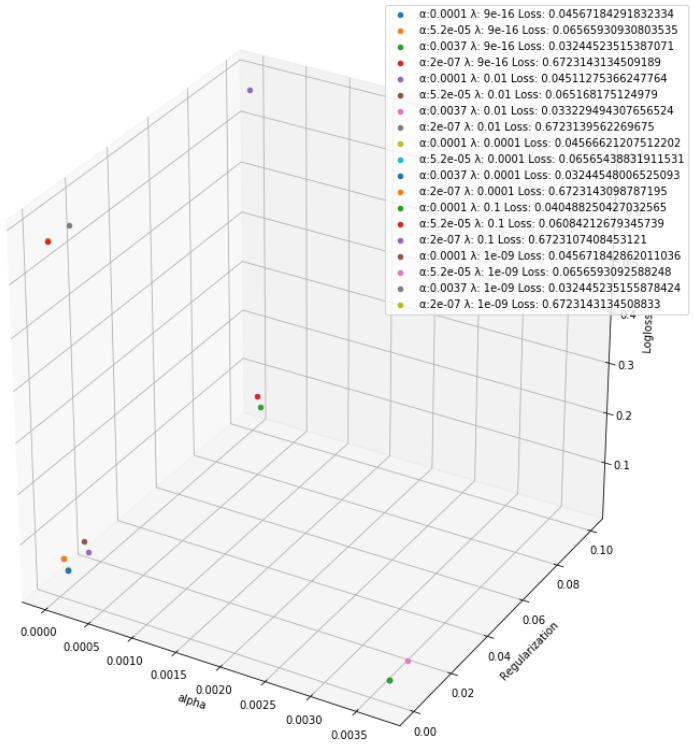

Epoch vs Train Logloss for α: 0.0037and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Validation Logloss for α: 0.0037and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Train Logloss for α: 2e-07and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Validation Logloss for α: 2e-07and λ: 1e-09
Logloss
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

3D plot with axes α,λ,Log loss

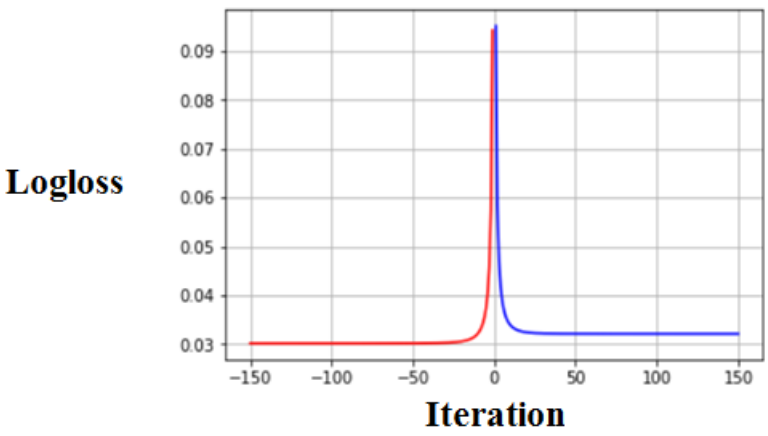


Output:

Optimal Value

```
Best Model using Grid search with k-fold cross validation:
        α: 0.0037
        λ: 9e-16
        Logloss: 0.03244523515387071
```

For the optimal values of alpha and regularization, the following is the logloss on test and train per iteration. Train loss is on the positive range and Test loss is on the negative range.

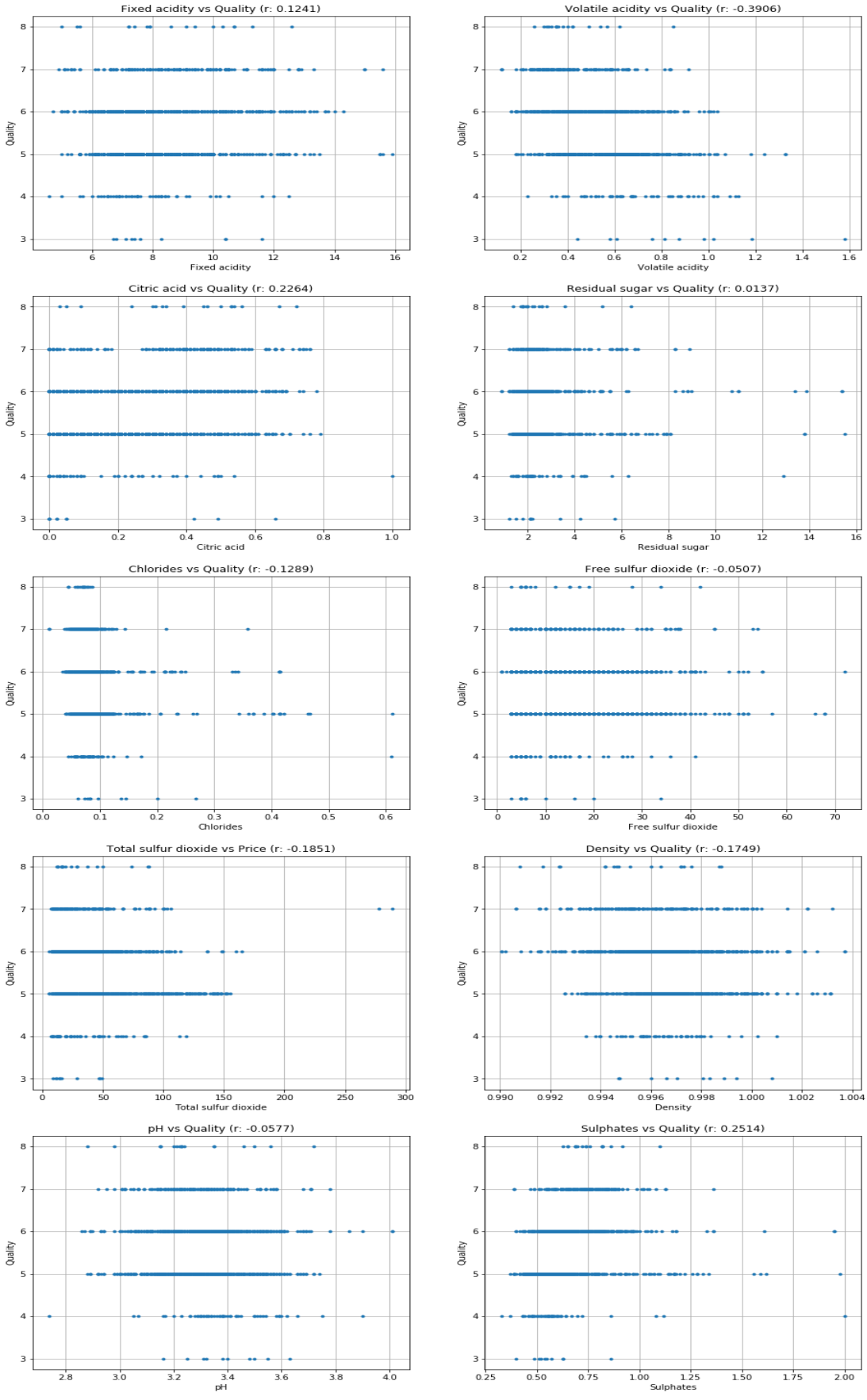


Comparision:

It can be seen that the optimal value of alpha and lambda can be got with grid search using k-fold Validation seem to have the maximum likelihood with minimum logloss. In this, the model not only trains But also validates the training model from which optimum value is chosen for generating the model for Future testing. In mini-BGD, the training model is validated and hence optimimum values might be wrong. It is difficult to calculate hyperparameters with normal mini-BGD, so we go for grid search with k-fold.

Wine dataset:

Correlation between variables and quality



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	0.1241	-0.3906	0.2264	0.0137	-0.1289	-0.0507	-0.1851	-0.1749	-0.0577	0.2514	0.4762	NaN

From the plots between the parameters and price, we can conclude that there is no relation between residual sugar, free sulphur dioxide, pH and quality. Hence, I am dropping out these columns from the dataset to be used for creating the model. After creating the model, as it could be seen that there's poor correlation between fixed acidity, chlorides, total sulphur dioxide and quality. At last, these columns are also dropped down from the dataset.

Data after dropping columns

	volatile acidity	citric acid	density	sulphates	alcohol	quality
0	0.70	0.00	0.9978	0.56	9.4	5
1	0.88	0.00	0.9968	0.68	9.8	5
2	0.76	0.04	0.9970	0.65	9.8	5
3	0.28	0.56	0.9980	0.58	9.8	6
4	0.70	0.00	0.9978	0.56	9.4	5

Normalized wine data

	volatile acidity	citric acid	density	sulphates	alcohol	quality
706	0.034998	0.005995	0.024979	0.017295	0.023869	0.021962
67	0.031633	0.005246	0.024994	0.021342	0.025539	0.021962
1089	0.018396	0.040469	0.025023	0.027966	0.023630	0.030746
382	0.011666	0.031476	0.025024	0.029438	0.022436	0.026354
589	0.013012	0.036722	0.025029	0.027230	0.025062	0.030746

Function name : linridgered_GD
Parameter : training and testing data of X and Y, step length, regularization value, batch size

This function performs the ridge regression using mini Batch Gradient Descent algorithm and calls the function for the same. In mini BGD, the training data is split into batches and is used for training. In our case, we've chosen a batch size of 50. Hence when all the batches are run through an epoch, the process is continued for the maximum epochs. With every batch, the model gets trained and the beta parameter gets updated per iteration and the corresponding log loss for train and test set are recorded for future plotting of the graph.

```
def linridgereg_gd(Y_train,Y_test,X_train,X_test,alpha,reg,minibatchsize,ttype):
    initialbeta = np.zeros(X_data.shape[1] + 1)
    train_graph = list()
    test_graph = list()
    if ttype != 'Tune':
        for i in range(0,len(reg)):
            for j in range(0,len(alpha)):
                rmse_train,rmse_test,beta = linridgereg_mini_bgd(alpha[j],initiall
                train_graph.append(rmse_train)
                test_graph.append(rmse_test)
        return train_graph,test_graph
    else:
        rmse_train,rmse_test,beta = linridgereg_mini_bgd(alpha,initialbeta,miniba
        return rmse_train,rmse_test
```

Function name : ridgered_mini_bgd
Parameter : training and testing data of X and Y, step length, regularization value, batch size

```
def linridgereg_mini_bgd(alpha,beta,minibatchsize,reg,Y_train,Y_test,X_train,X
    X = np.insert(X_train, 0, 1, axis = 1)
    rmse_train = []
    rmse_test = []
    epochs = 150
    batches = math.ceil(len(X)/minibatchsize)
    for iteration in range(0,epochs):
        for i in range(0,batches):
            if i*50+50 < len(X):

                batch_X = X[i*50:i*50+50]
                batch_Y = Y_train[i*50:i*50+50]
            else:

                batch_X = X[i*50:]
                batch_Y = Y_train[i*50]
            Y_prediction = linalg_prediction(beta,batch_X,'Train')
            error = Y_prediction - batch_Y
            func_gradient = 2 * np.dot(batch_X.T,error)
            beta = beta - alpha * (func_gradient)
            Y_prediction = linalg_prediction(beta,batch_X,'Train')
            error = Y_prediction - batch_Y
        if (steplength_type == "Bold"):
            alpha = bold_steplength(alpha,likelihood_new,likelihood)
        rmsetrain = RMSE(batch_Y,Y_prediction)
        rmsetest = RMSE(Y_test,linalg_prediction(beta,X_test,'Test'))
        rmse_train.append(rmsetrain)
        rmse_test.append(rmsetest)
    return rmse_train,rmse_test,beta
```

Function name : linalg_prediction
Parameter : beta and data

```
def linalg_prediction(beta,data,ttype):
    if ttype == 'Test':
        X = np.ones((data.shape[0],data.shape[1]+1))
        X[:,1:] = data
        return np.dot(beta,X.T)
    else:
        return np.dot(beta,data.T)
```

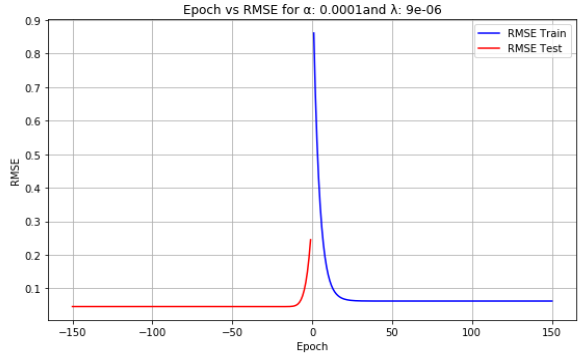
Initializations:

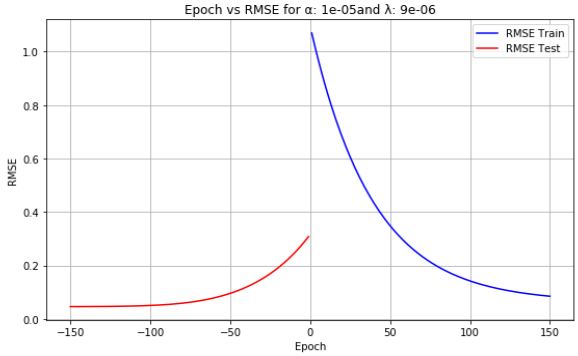
We use the following constant step length, regularization parameters and batch size for training

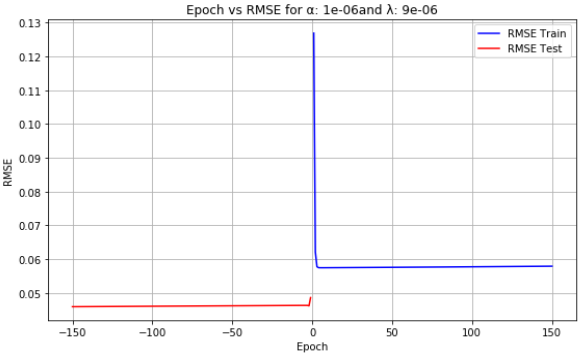
```
alpha = [1e-4,1e-5,1e-3]
reg = [9e-6,1e-10,1e-3]
```

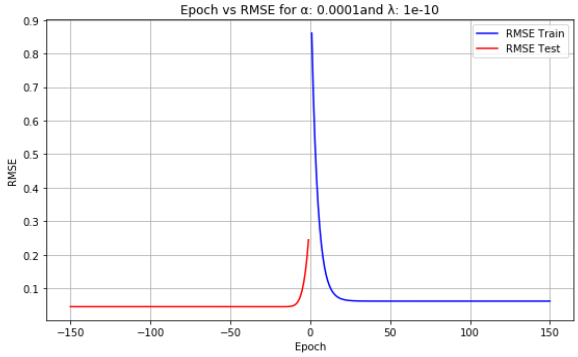
For all the possible combinations, the mini-BGD algorithm is used to train the model.

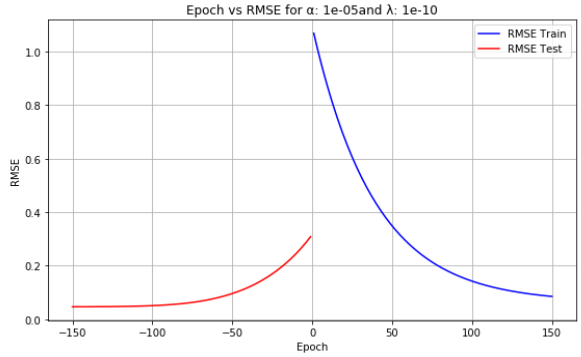
The following are the recorded RMSE train and test data for all the possible combinations.

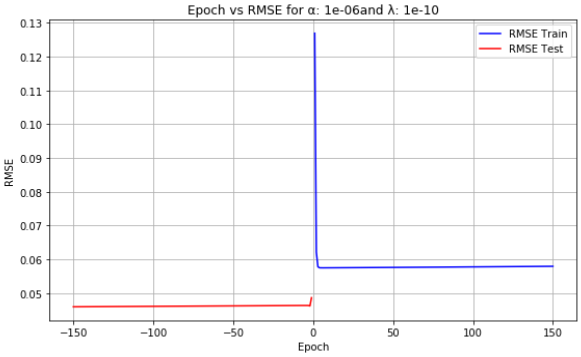


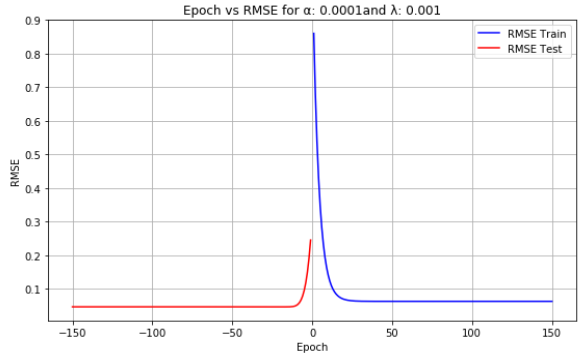


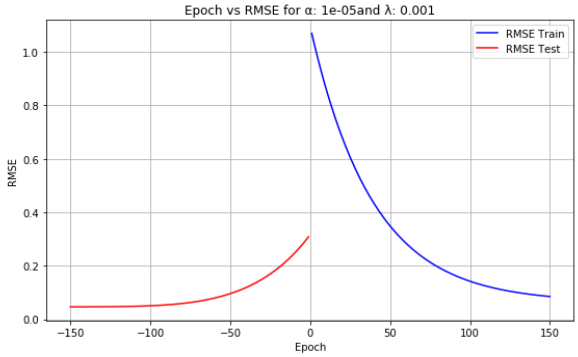


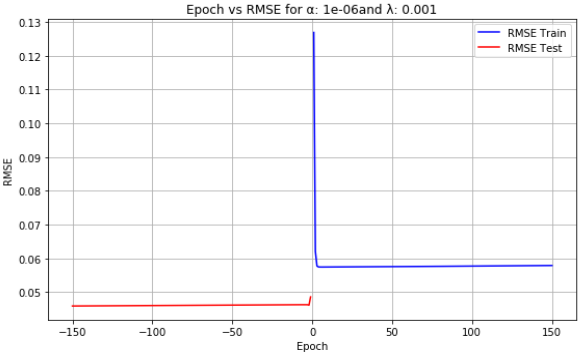


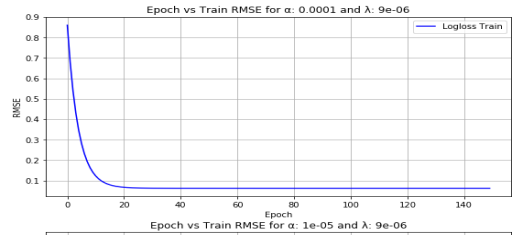


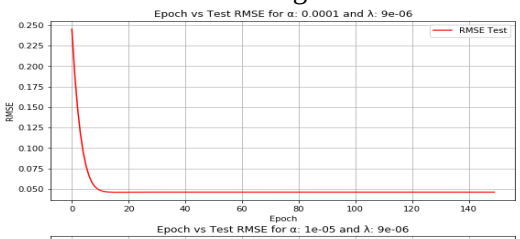


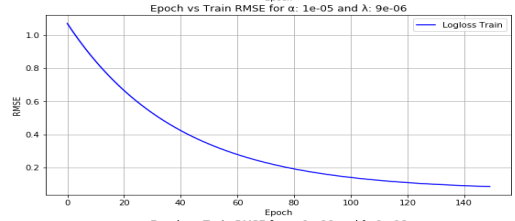


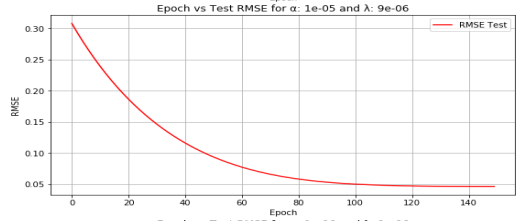


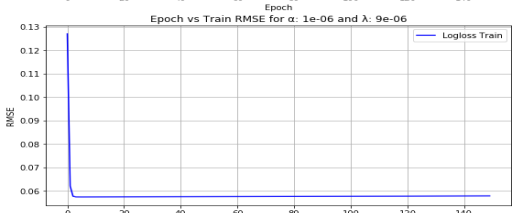


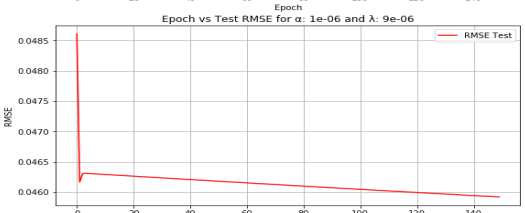

Epoch vs Train RMSE for α: 0.0001 and λ: 9e-06
Logloss Train
RMSE
Epoch

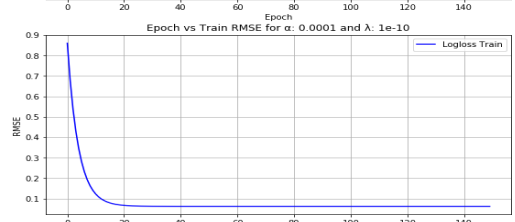

Epoch vs Test RMSE for α: 0.0001 and λ: 9e-06
RMSE Test
RMSE
Epoch

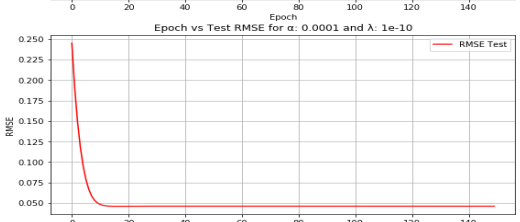

Epoch vs Train RMSE for α: 1e-05 and λ: 9e-06
Logloss Train
RMSE
Epoch

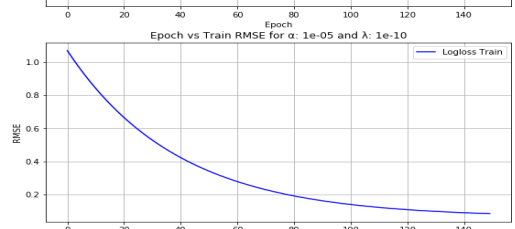

Epoch vs Test RMSE for α: 1e-05 and λ: 9e-06
RMSE Test
RMSE
Epoch

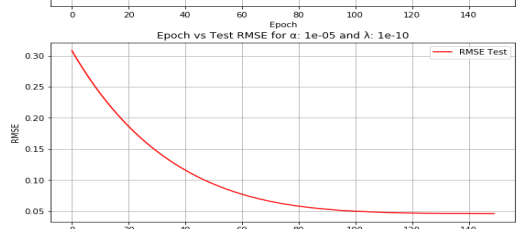

Epoch vs Train RMSE for α: 1e-06 and λ: 9e-06
Logloss Train
RMSE
Epoch

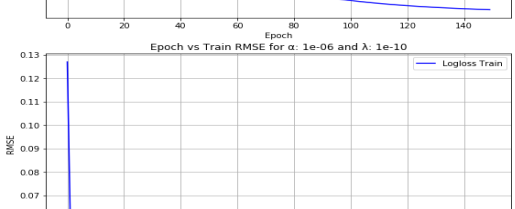

Epoch vs Test RMSE for α: 1e-06 and λ: 9e-06
RMSE Test
RMSE
Epoch

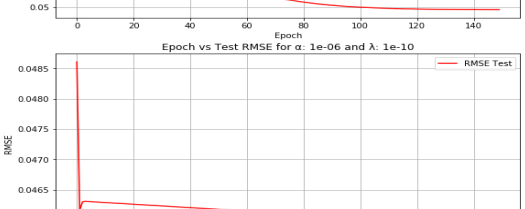

Epoch vs Train RMSE for α: 0.0001 and λ: 1e-10
Logloss Train
RMSE
Epoch

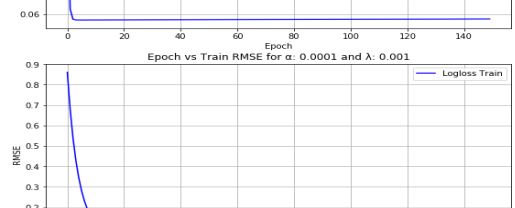

Epoch vs Test RMSE for α: 0.0001 and λ: 1e-10
RMSE Test
RMSE
Epoch

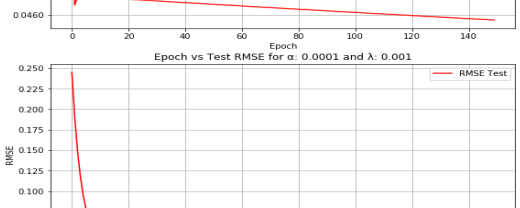

Epoch vs Train RMSE for α: 1e-05 and λ: 1e-10
Logloss Train
RMSE
Epoch

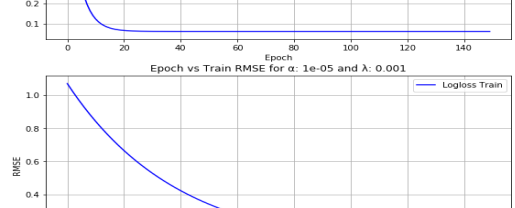

Epoch vs Test RMSE for α: 1e-05 and λ: 1e-10
RMSE Test
RMSE
Epoch

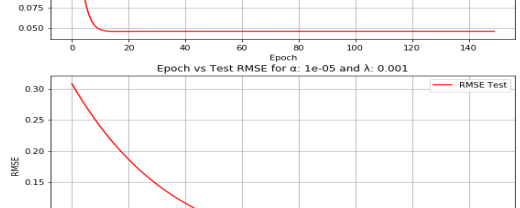

Epoch vs Train RMSE for α: 1e-06 and λ: 1e-10
Logloss Train
RMSE
Epoch

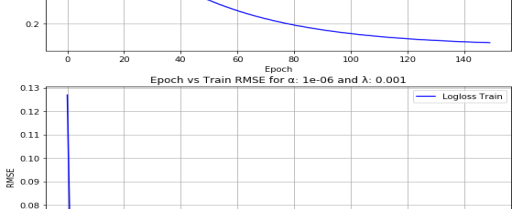

Epoch vs Test RMSE for α: 1e-06 and λ: 1e-10
RMSE Test
RMSE
Epoch

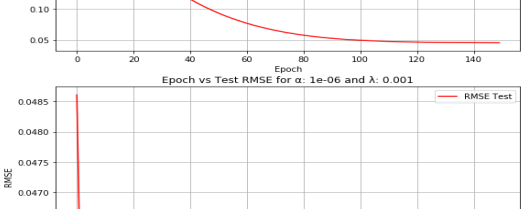

Epoch vs Train RMSE for α: 0.0001 and λ: 0.001
Logloss Train
RMSE
Epoch


Epoch vs Test RMSE for α: 0.0001 and λ: 0.001
RMSE Test
RMSE
Epoch


Epoch vs Train RMSE for α: 1e-05 and λ: 0.001
Logloss Train
RMSE
Epoch


Epoch vs Test RMSE for α: 1e-05 and λ: 0.001
RMSE Test
RMSE
Epoch

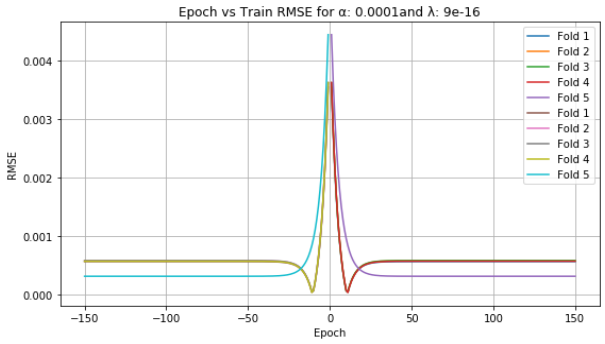

Epoch vs Train RMSE for α: 1e-06 and λ: 0.001
Logloss Train
RMSE
Epoch

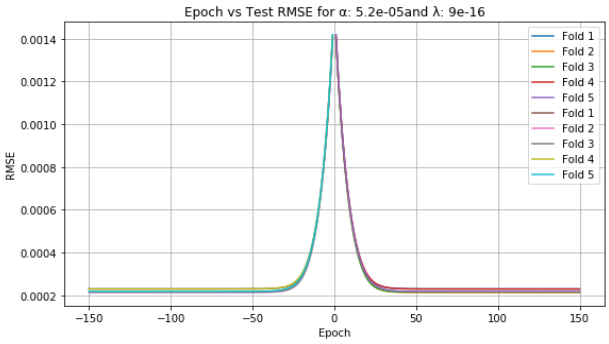

Epoch vs Test RMSE for α: 1e-06 and λ: 0.001
RMSE Test
RMSE
Epoch

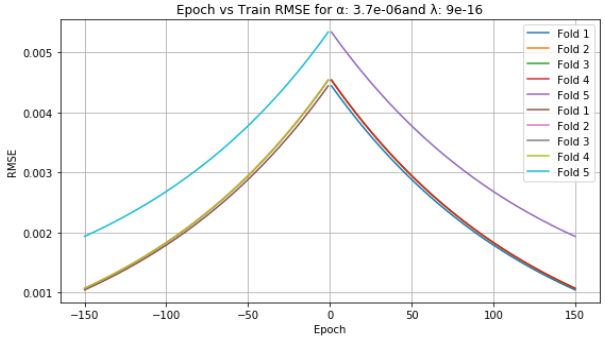
Hyper-parameter tuning:

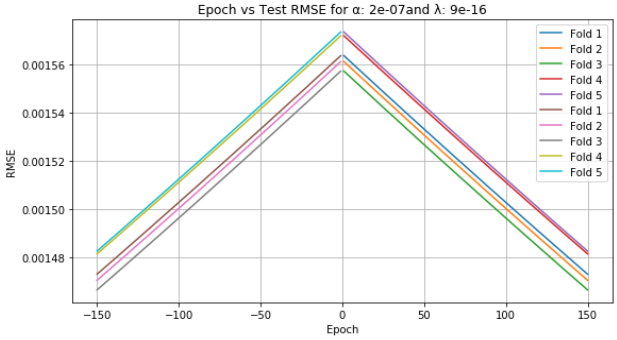
Initializations

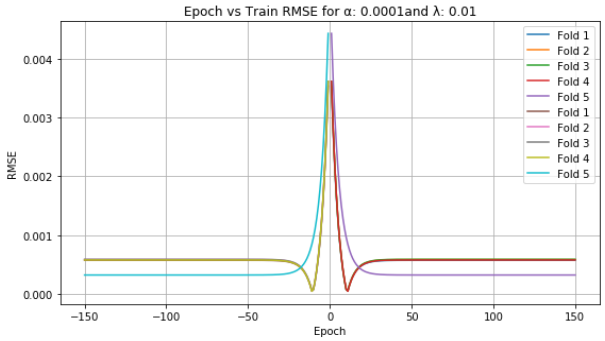
```
alpha = [1e-4,5.2e-5,3.7e-6,2e-7]
reg = [9e-6,0.01,1e-4,0.1,1e-9]
minibatchsize = 50
fold = 5
```

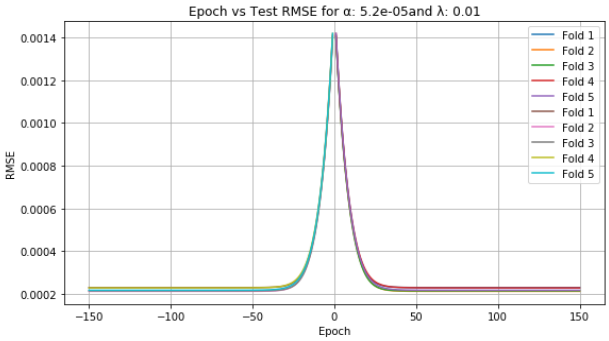


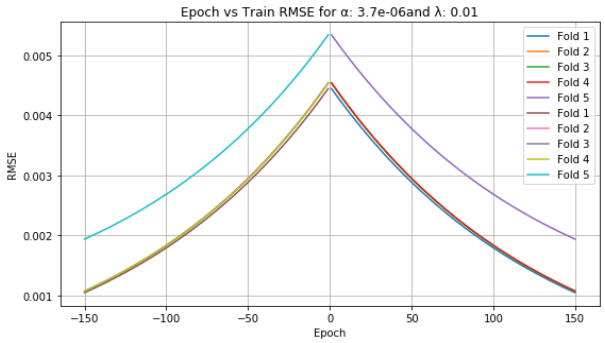


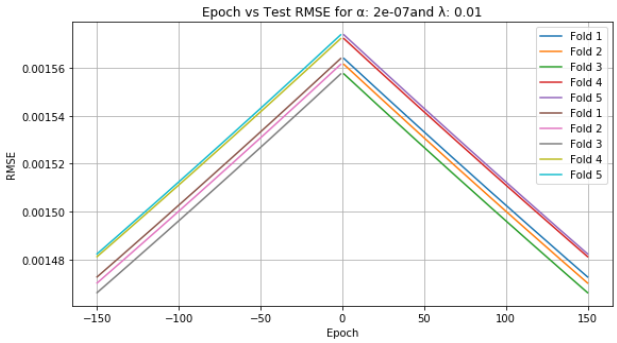


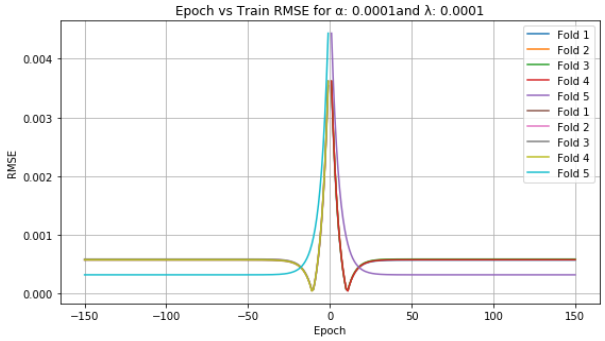


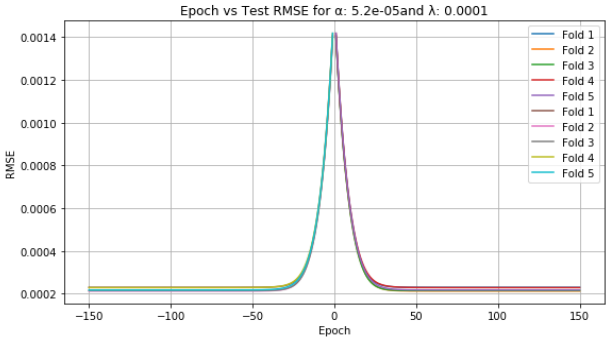






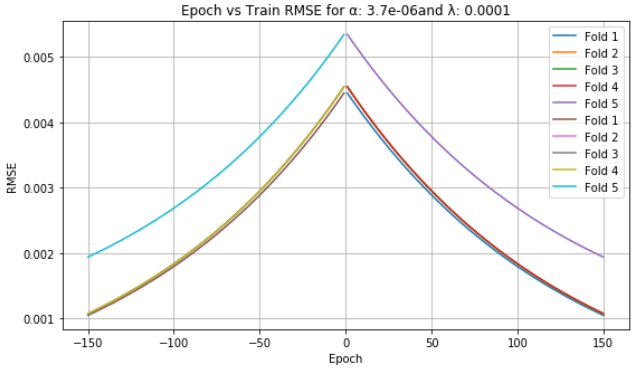


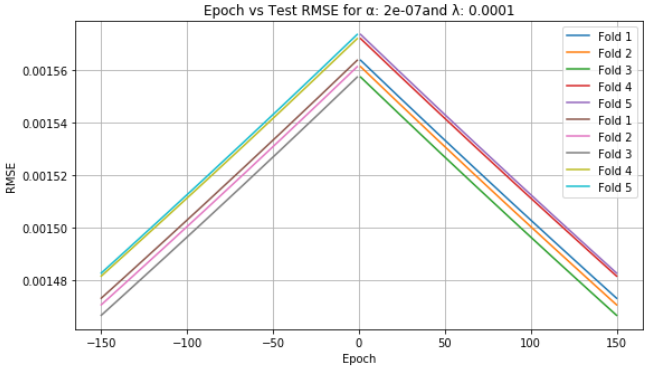


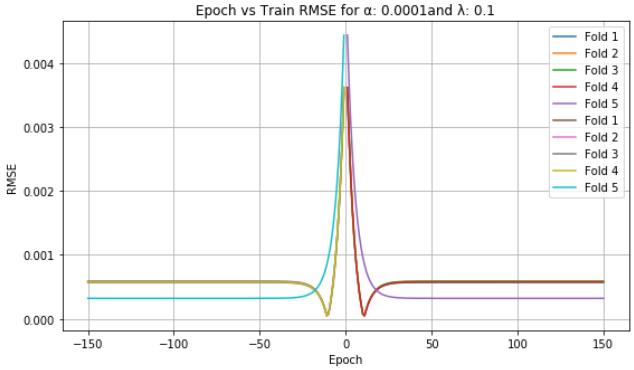


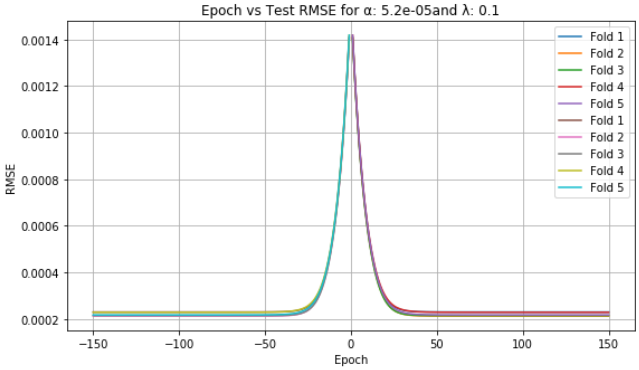
Epoch vs Train RMSE for α: 3.7e-06and λ: 0.0001 Epoch vs Test RMSE for α: 2e-07and λ: 0.0001

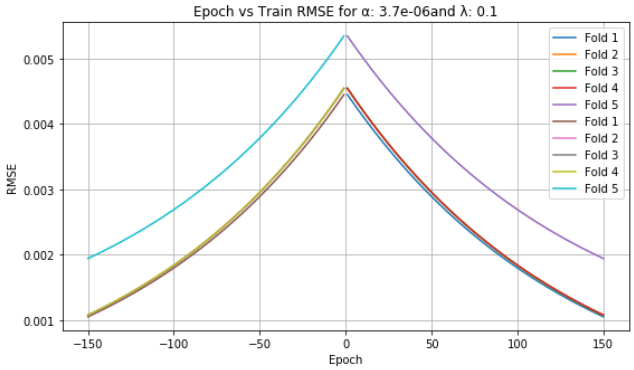
Plot for 5 folds with all possible combinations of steplength and regularization parameters

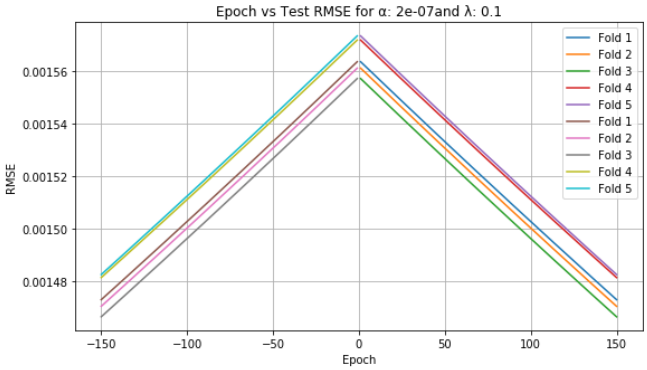


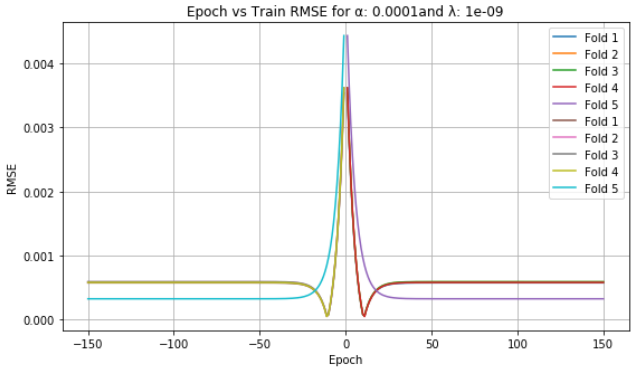


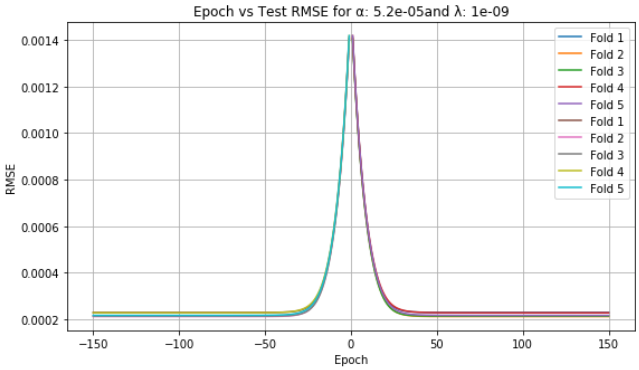


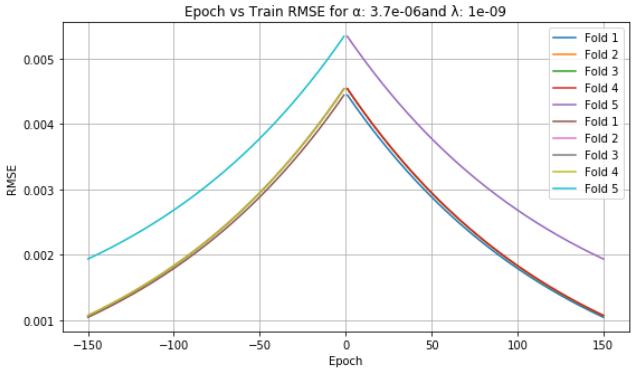


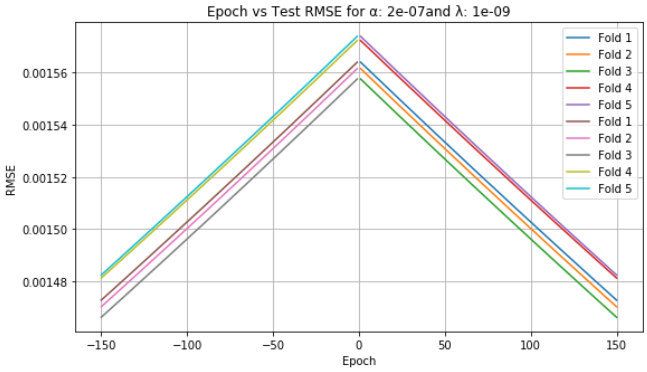




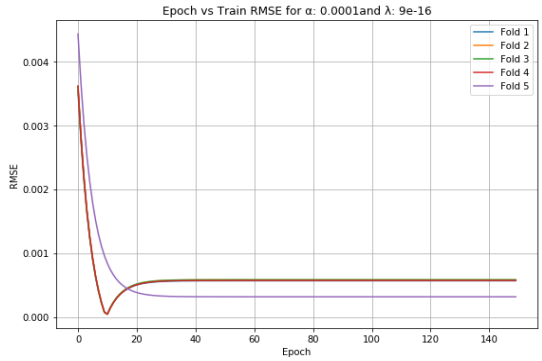


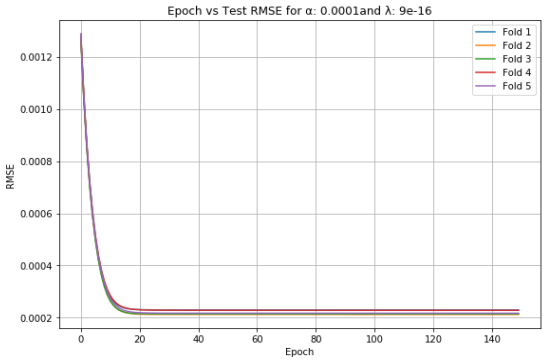


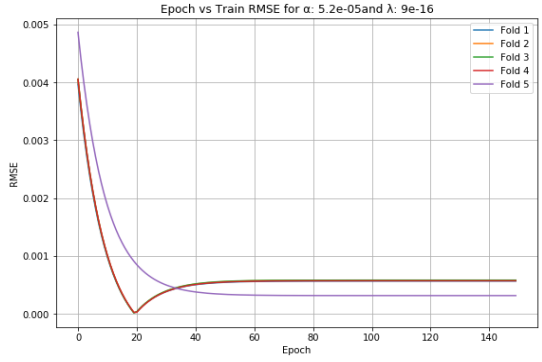


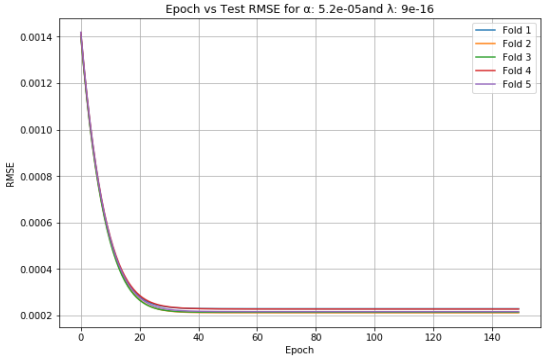


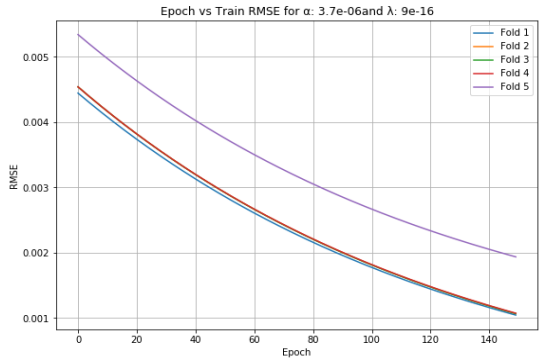
Plot with Epoch vs RMSE Train and Epoch vs RMSE Test

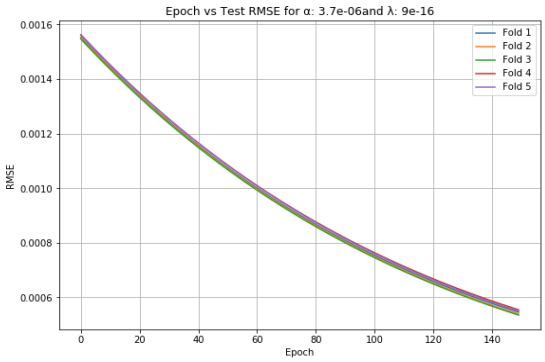


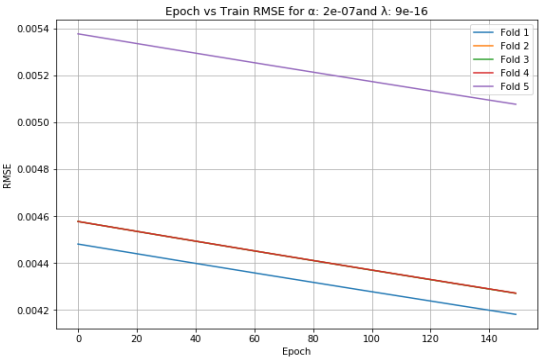


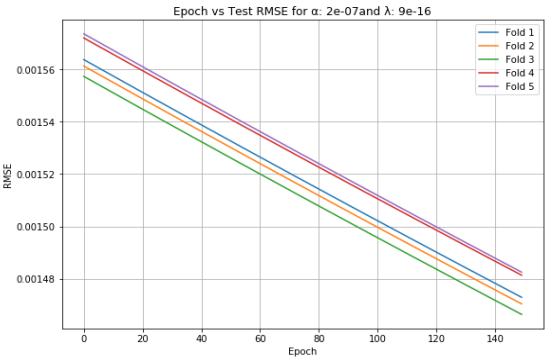


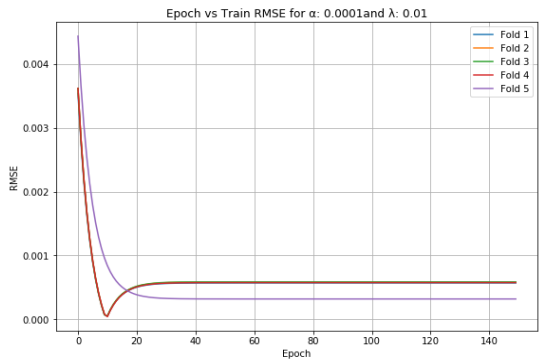


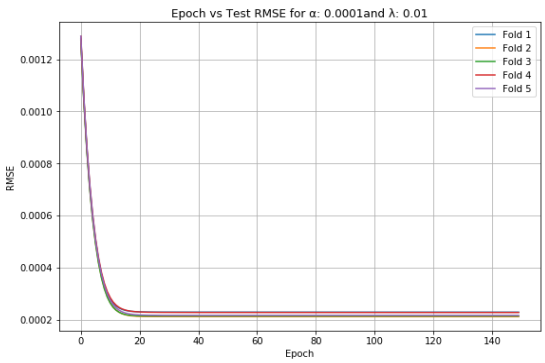


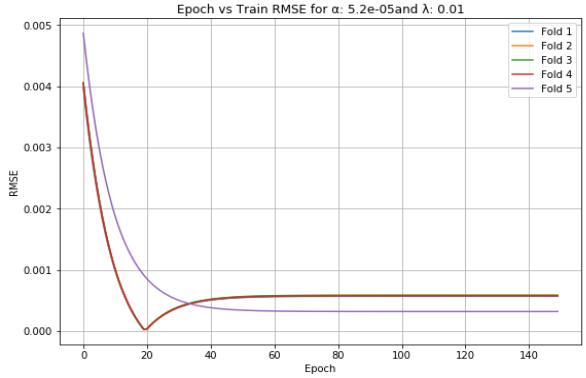


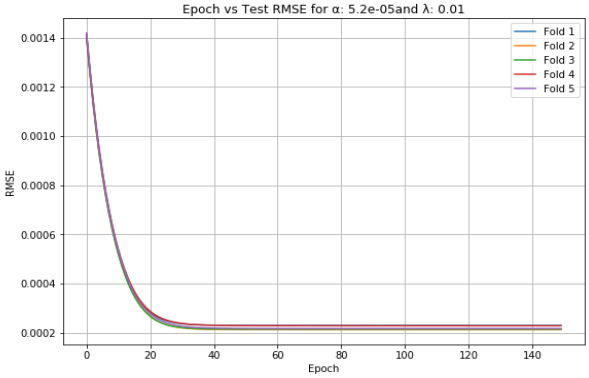


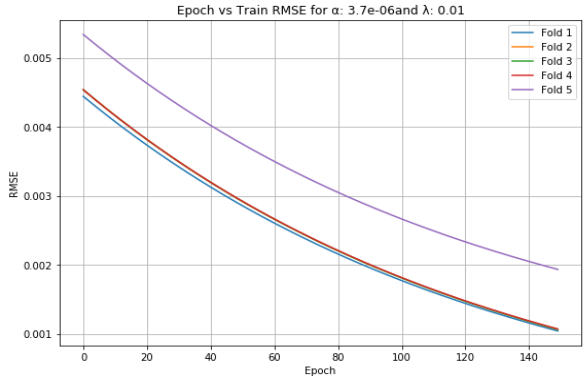


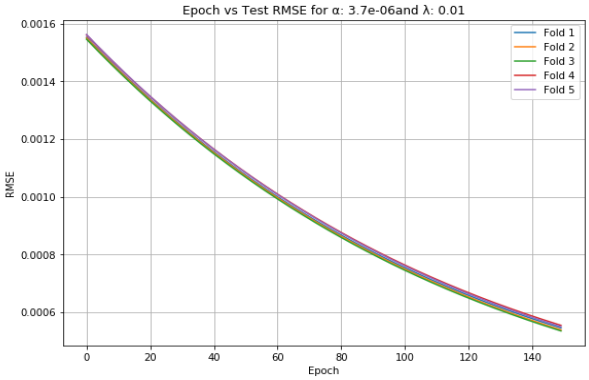


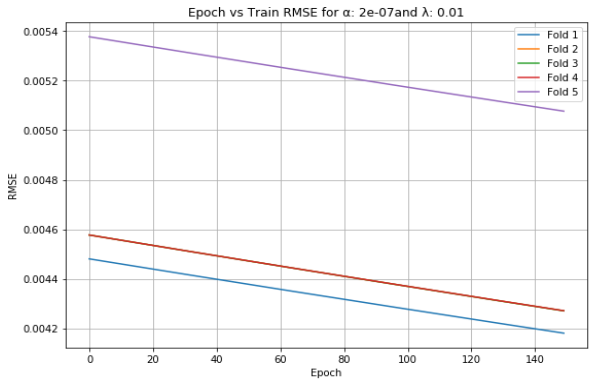


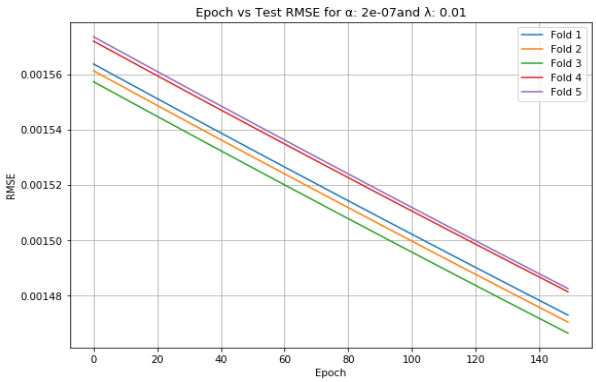

Epoch vs Train RMSE for α: 5.2e-05and λ: 0.01
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

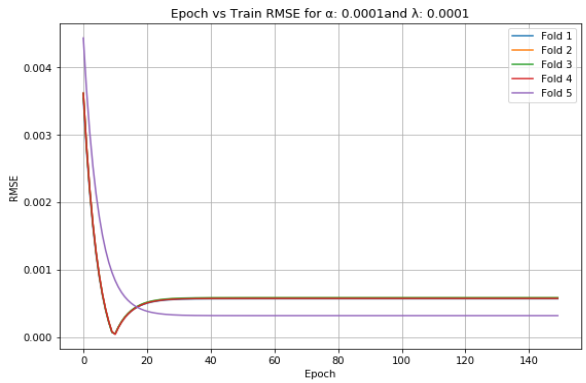

Epoch vs Test RMSE for α: 5.2e-05and λ: 0.01
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

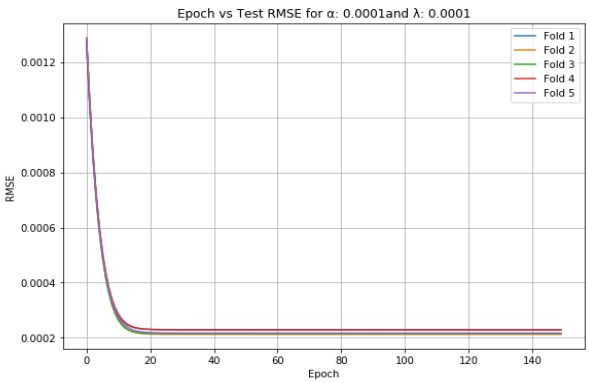

Epoch vs Train RMSE for α: 3.7e-06and λ: 0.01
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

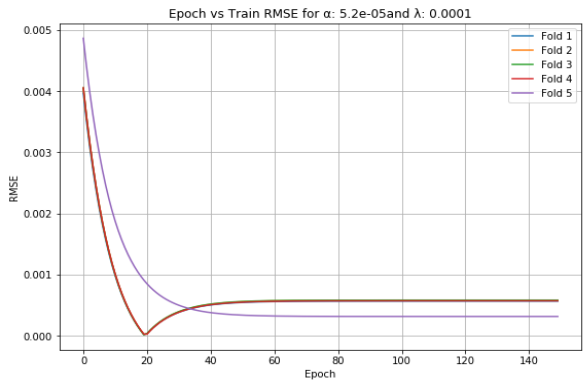

Epoch vs Test RMSE for α: 3.7e-06and λ: 0.01
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

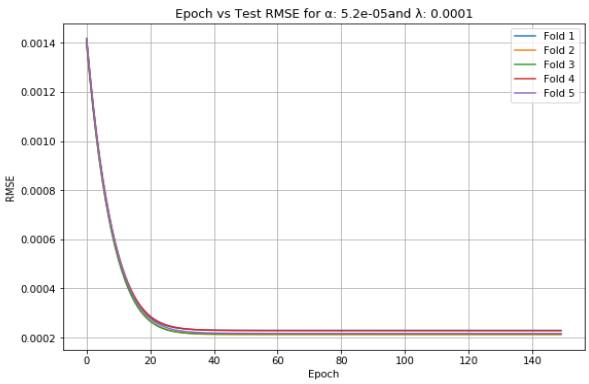

Epoch vs Train RMSE for α: 2e-07and λ: 0.01
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

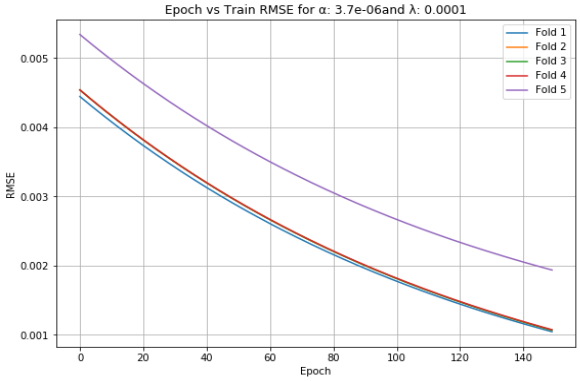

Epoch vs Test RMSE for α: 2e-07and λ: 0.01
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

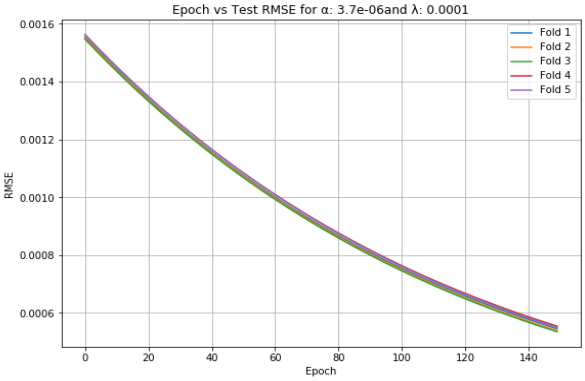

Epoch vs Train RMSE for α: 0.0001and λ: 0.0001
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

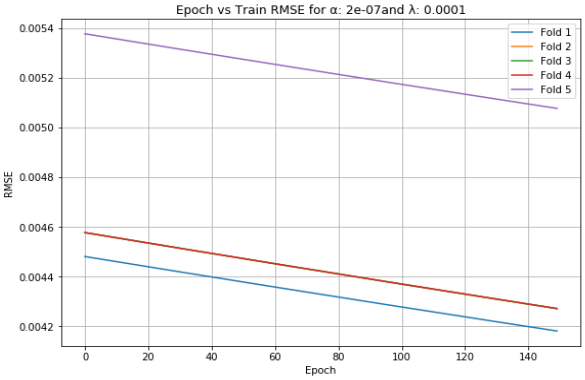

Epoch vs Test RMSE for α: 0.0001and λ: 0.0001
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

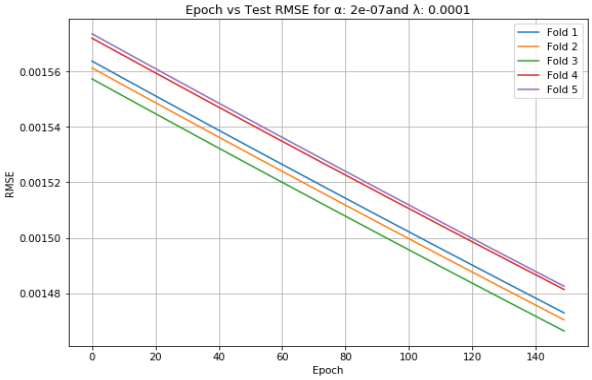

Epoch vs Train RMSE for α: 5.2e-05and λ: 0.0001
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

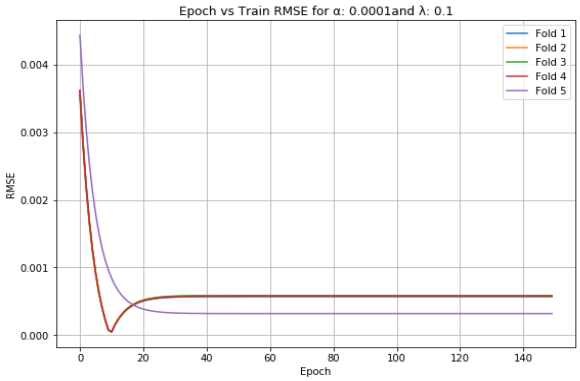

Epoch vs Test RMSE for α: 5.2e-05and λ: 0.0001
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

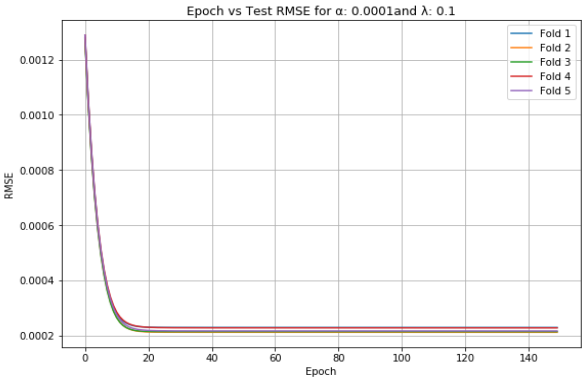

Epoch vs Train RMSE for α: 3.7e-06and λ: 0.0001
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

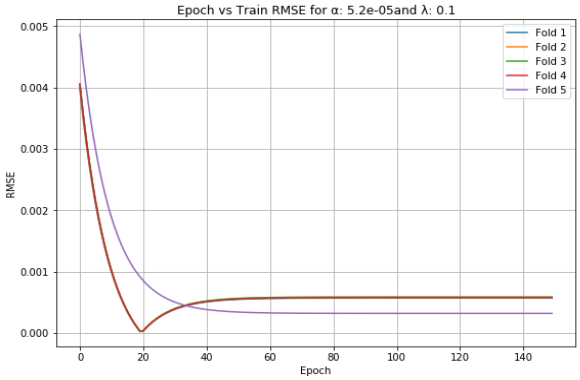

Epoch vs Test RMSE for α: 3.7e-06and λ: 0.0001
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

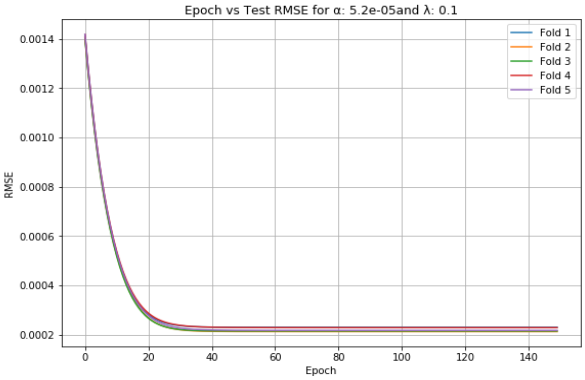

Epoch vs Train RMSE for α: 2e-07and λ: 0.0001
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

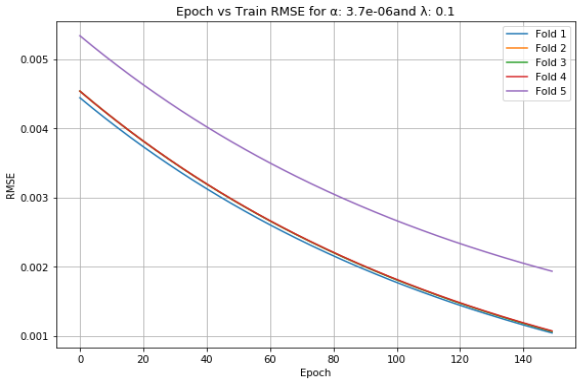

Epoch vs Test RMSE for α: 2e-07and λ: 0.0001
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

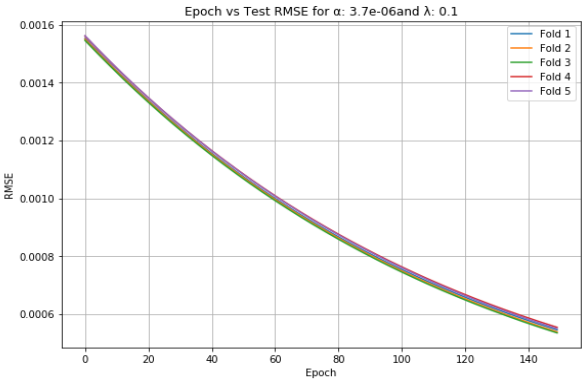

Epoch vs Train RMSE for α: 0.0001and λ: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

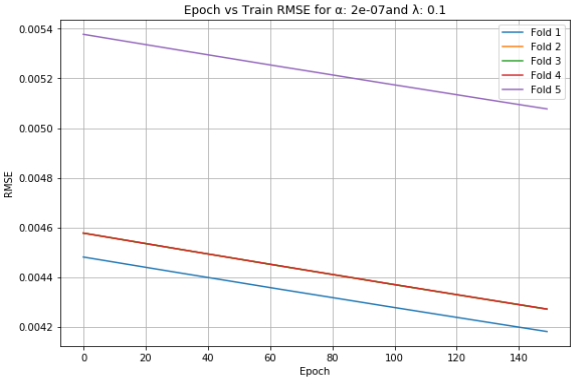

Epoch vs Test RMSE for α: 0.0001and λ: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

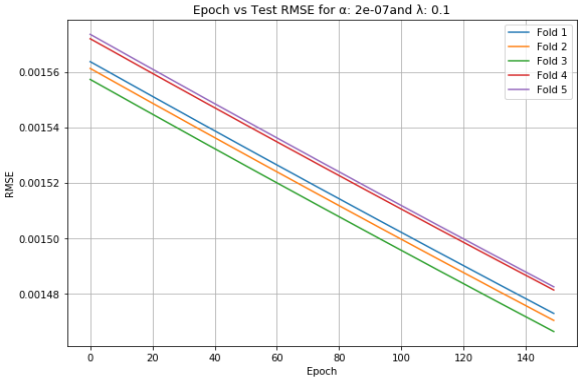

Epoch vs Train RMSE for α: 5.2e-05and λ: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

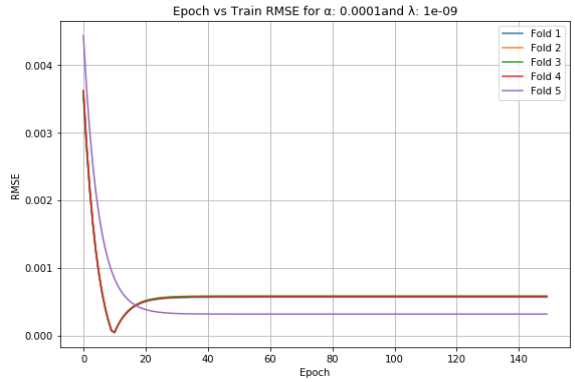

Epoch vs Test RMSE for α: 5.2e-05and λ: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

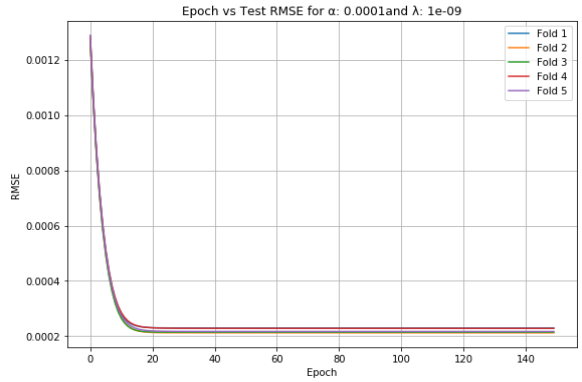

Epoch vs Train RMSE for α: 3.7e-06and λ: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

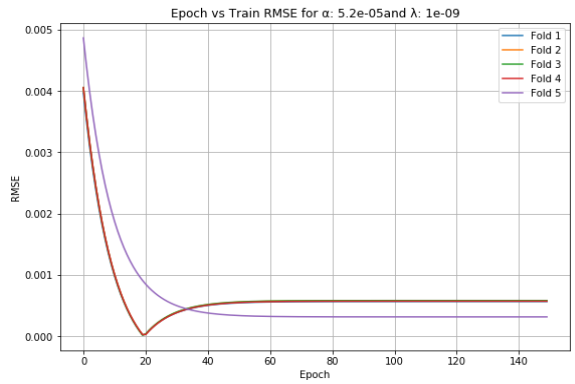

Epoch vs Test RMSE for α: 3.7e-06and λ: 0.1
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
RMSE
Epoch

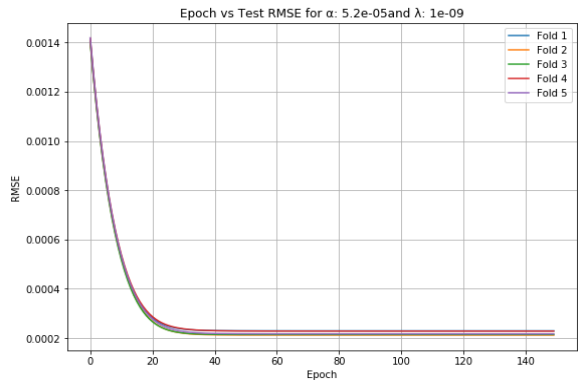

Epoch vs Train RMSE for α: 2e-07and λ: 0.1
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

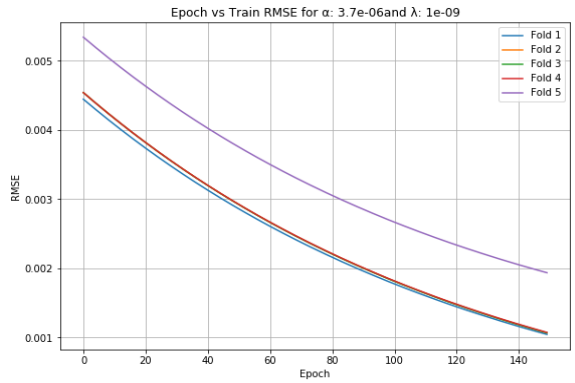

Epoch vs Test RMSE for α: 2e-07and λ: 0.1
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

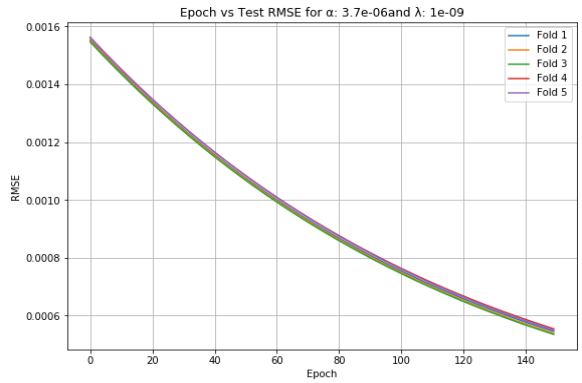

Epoch vs Train RMSE for α: 0.0001and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

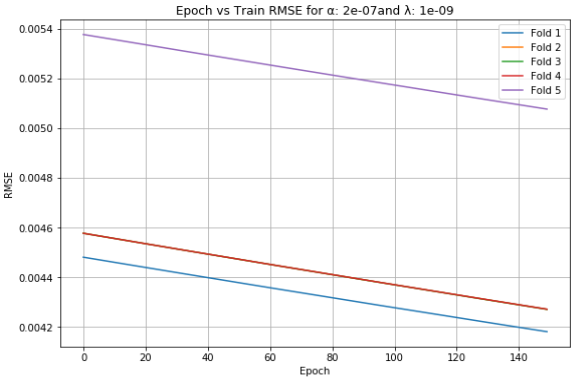

Epoch vs Test RMSE for α: 0.0001and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

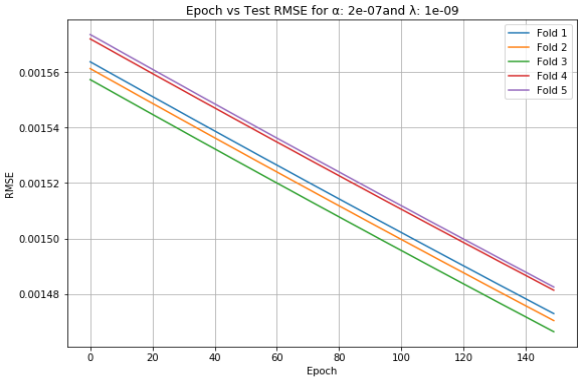

Epoch vs Train RMSE for α: 5.2e-05and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Test RMSE for α: 5.2e-05and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

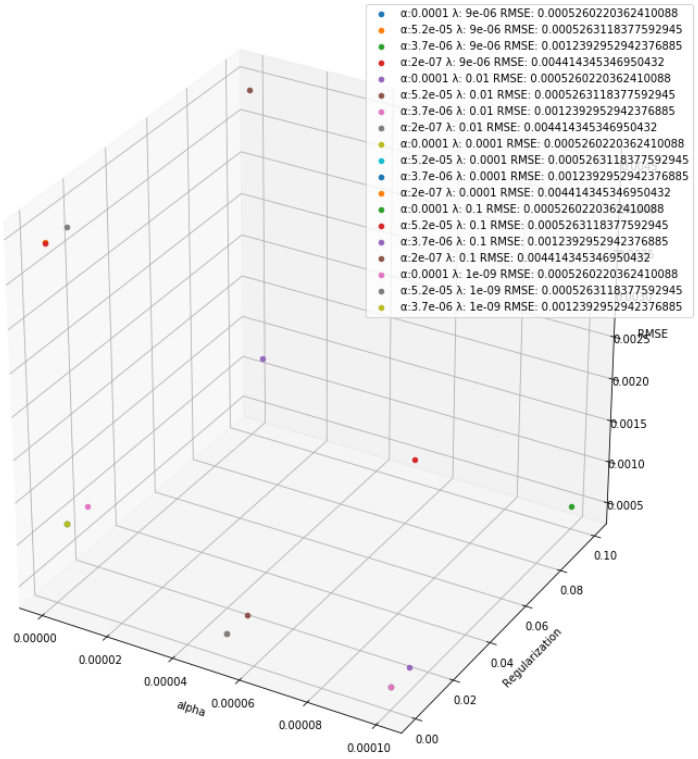

Epoch vs Train RMSE for α: 3.7e-06and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Test RMSE for α: 3.7e-06and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Train RMSE for α: 2e-07and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5


Epoch vs Test RMSE for α: 2e-07and λ: 1e-09
RMSE
Epoch
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5

3D plot with axes α,λ,Log loss

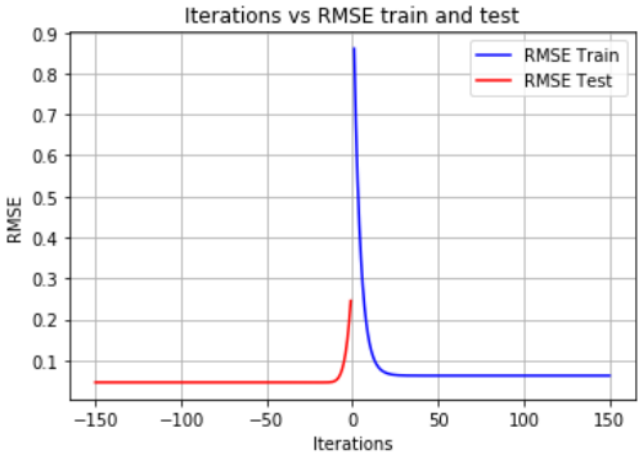


Output:

Optimal Value

```
Best Model using Grid search with k-fold cross validation:
        α: 0.0001
        λ: 9e-16
        RMSE: 0.0005260220362410088
```

For the optimal values of alpha and regularization, the following is the RMSE on test and train per iteration. Train RMSE is on the positive range and Test RMSE is on the negative range.



Comparision:

It can be seen that the optimal value of alpha and lambda got with grid search using k-fold validation seem to have the maximum likelihood with minimum logloss. In this, the model not only trainn but also validates the training model from which optimum value is chosen for generating the model for future testing. In mini-BGD, the training model is validated and hence optimimum values might be wrong. The RMSE error with k-fold validation is as low as compared to the normal mini-BGD. It's always better to validate a model before testing it for unknown instances. It is difficult to calculate hyperparameters with normal mini-BGD, so we go for grid search with k-fold.