Lab Course - Distributed Data Analytics
Exercise 4

Function name : read_virusshare
Parameter : Filedirectory

This function reads the data in SVM light format returns the data in the format of data frame and the corresponding labels.

```
def read_virusshare(directory):
    data = np.array([])
    label = []
    a = 0
    for files in os.listdir(directory)[:1]:
        x,y = load_svmlight_file(directory+files)
        a+=x.shape[0]
        if x.shape[1] < 479:
            pad = np.zeros((x.shape[0],1))
            x = scipy.sparse.hstack((x,np.array(pad)))
        data = scipy.sparse.vstack((data,x))
        data = data.tocsr()
        label.extend(y)
    return data[1:,].tocsr(),label
```

Function name : read_cup98
Parameter : Dataframe and column names

This function reads the Cup98 dataset data returns the data in the format of data frame. The data is then checked for the categorical columns and are converted to numbers as it is a linear regression problem. In this data, I have considered AVGFIT as the target variable

```
def read_cup98(directory):
    data = pd.read_csv(directory,low_memory=False)
    return data
```

```
cols = cup_data.columns
numCol = cup_data._get_numeric_data().columns
catCol = list(set(cols) - set(numCol))
for each in numCol:
    cup_data[each] = cup_data[each].astype('float')
for each in catCol:
    cup_data[each] = cup_data[each].astype('category')
cat_columns = cup_data.select_dtypes(['category']).columns
cup_data[cat_columns] = cup_data[cat_columns].apply(lambda x: x.cat.codes)
cup_data = cup_data.fillna(cup_data.mean())
Y_data = list(cup_data["AVGGIFT"])
X_data = cup_data.drop(columns = ['AVGGIFT'])
Y_train,Y_test,X_train,X_test = create_Test_Train_data(X_data,Y_data)
```

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(0.7*len(Y_data))]
    Y_test = Y_data[math.ceil(0.7*len(Y_data)):]
    X_train = X_data[:math.ceil(0.7*len(X_data))]
    X_test = X_data[math.ceil(0.7*len(X_data)):]
    return Y_train,Y_test,X_train,X_test
```

Function name : RMSE
Parameter : Actual data and predicted data

This function returns the Root Mean Squared Error (RMSE) between the actual data and predicted data.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

```
def RMSE(y_data,y_pred):
    return np.sqrt(np.sum(pow((y_data - y_pred),2)))/len(y_pred)
```

Function name : linalg_prediction
Parameter : beta values and data

This function returns the dot product between the data and beta values which is the predicted data.

```
def linalg_prediction(beta,data):
    X = np.ones((data.shape[0],data.shape[1]+1))
    X[:,1:] = data
    return np.dot(beta,X.T)
```

Function name : loss
Parameter : beta values , data

This function returns the loss for the training data which is further used for checking the convergence of the model.

```
def loss(beta,data,Y_train):
    pred = linalg_prediction(beta,data)
    return np.sum(pow((Y_train - pred),2))
```

Function name : linreg_PSGD
Parameter : learningrate, beta, training data

This function returns the loss for the training data which is further used for checking the convergence of the model.

```
def linreg_PSGD(alpha,beta,Y_train,X_train):

    X = np.ones((X_train.shape[0],X_train.shape[1]+1))
    X[:,1:] = X_train
    index = np.arange(0,len(X))
    np.random.shuffle(index)
    for i in index:
        xtr = X[i]
        ytr = Y_train[i]
        Y_prediction = np.dot(beta,xtr.T)
        residual = Y_prediction - ytr
        func_gradient = 2 * np.dot(xtr.T,residual)
        beta = beta - (alpha * func_gradient)
    return beta
```

1. Parallel Linear Regression with PSGD:

```
if rank == 0:
    beta_new,beta_split = None,None
    virus_directory = "G:/DA - Hildeshim/DDA Lab/Exercise 4/Virusshare/"
    virus_data, virus_label = read_virusshare(virus_directory)
    X_data = pd.DataFrame(virus_data[:1000].todense())
    Y_data = virus_label[:1000]
    Y_train,Y_test,X_train,X_test = create_Test_Train_data(X_data,Y_data)
    X_train_split = np.array_split(X_train,size-1)
    Y_train_split = np.array_split(Y_train,size-1)
    beta = np.zeros((X_data.shape[1]+1))
    for worker in range(1,size):
        comm.send(X_train_split[worker-1],dest = worker,tag = 1)
        comm.send(Y_train_split[worker-1],dest = worker,tag = 2)
```

While executing the Linear Regression with Stochastic Gradient Descent in parallel, the processes are divided into masters and workers. As we know, the master assigns the task to the workers. In this exercise, the master performs the similar tasks. It performs operations like reading the data from the dataset, converting them to X data that are inputs and Y data that are target values, splitting the data into train and split and initialising the beta values to zero.

Once the tasks are done, the master splits the training data based on the number of workers and sends them in a loop to all the workers assigning a tag such that the input and training data are received properly.

The workers then receive these data of inputs and targets sent by the master and also initialises the none variables for which data is broadcasted by the master. The receiving of data from the master is as shown in the following code snippet.

```
        else:
            beta = None
            X_test = None
            Y_test = None
            X_train = comm.recv(source = 0,tag = 1)
            Y_train = comm.recv(source = 0,tag = 2)
```

Parallel Stochastic Gradient Descent:

```
for i in range(100): #epochs
    beta = comm.bcast(beta,root = 0)
    X_test = comm.bcast(X_test,root = 0)
    Y_test = comm.bcast(Y_test,root = 0)
    if rank != 0:
        beta_split = linreg_PSGD(1.08e-10,beta,Y_train,X_train)

    comm.Barrier()
    beta_new = comm.gather(beta_split,0)
    comm.Barrier()
    if rank == 0:

        losstrain_old = loss(beta,X_train,Y_train)
        beta = np.sum(np.array(beta_new)[1:,])/(size-1)
        losstrain_new= loss(beta,X_train,Y_train)
        if abs(losstrain_old - losstrain_new) < 1e-20:
            print("Converged")
            break
        y_pred_train = linalg_prediction(beta,X_train)
        rmse_train.append(RMSE(Y_train,y_pred_train))
        y_pred_test = linalg_prediction(beta,X_test)
        rmse_test.append(RMSE(Y_test,y_pred_test))
        time.append(MPI.Wtime() - start)
```

This is to perform the Stochastic Gradient descent in parallel. In the first place, the workers receive the data broadcasted by the master and is sent iteratively into PSGD function to find new values of beta. The learning rate is set to 0.5e-14 and 1.08e-10 for KDD and Virus Share dataset respectively. These beta received from the workers are averaged and loss are found. Also, training and testing predictions are found to calculate the RMSE values of the same. The timing is noted down for each epoch. The process time for the P = [2,4,6,8] are tabulated as follows

Process time for different processes on Virus Share dataset

	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8
Process 2	7.73	7.72						
Process 4	5.76	5.78	5.76	5.75				
Process 6	10.66	10.65	10.69	10.65	10.65	10.66		
Process 8	28.08	28.08	28.06	28.1	28.07	28.08	28.07	28.05

Process time for different processes on KDD Cup 1998 dataset

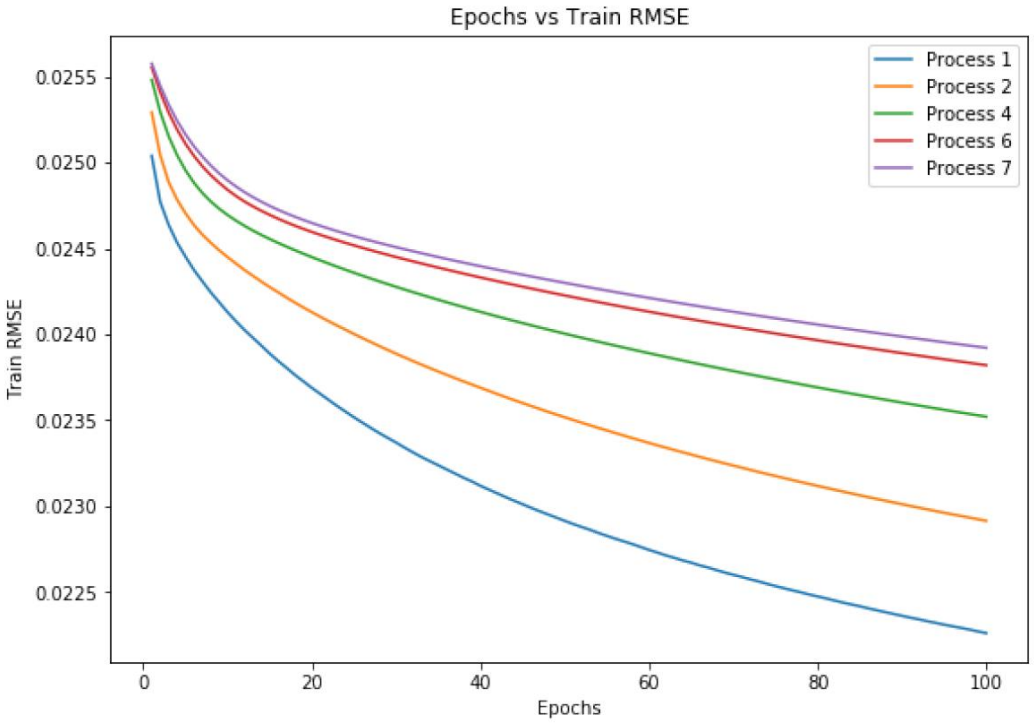
	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8
Process 2	11.36	11.35						
Process 4	5.782	5.781	5.779	5.775				
Process 6	15.287	15.274	15.274	15.255	15.276	15.245		
Process 8	28.20	28.196	28.162	28.187	28.158	28.198	28.161	28.154

2. Performance and convergence of PSGD:

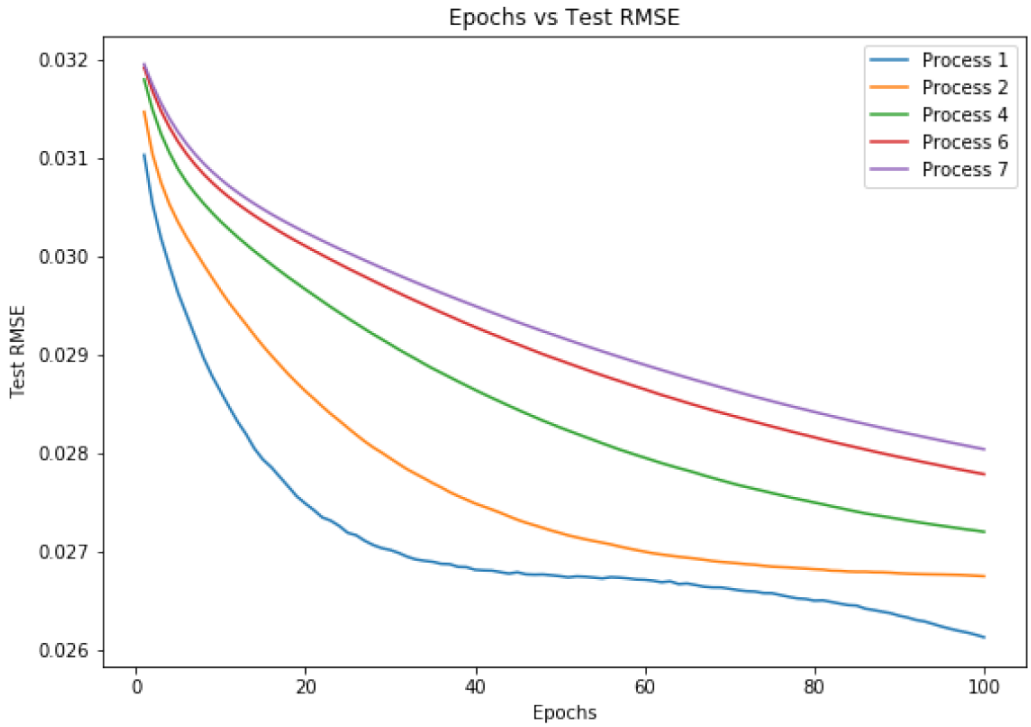
The RMSE of training and testing data are noted for the processes P = [1,2,4,6,7] and are used to plot the graphs using the following code snippet.

```
epochs = [i for i in range(1,len(virus_process_train["Process 1"])+1)]
fig, ((ax1,ax2)) = plt.subplots(2, 2, figsize=(20, 14))
fig.suptitle("Performance Analysis of Cup98 dataset",fontsize = 16)
cols = virus_process_train.columns
odd = [i for i in range(len(cols)) if i%2 == 1]
for train in odd:
    ax1[0].plot(epochs, virus_process_train[cols[train]],label=cols[train])
    ax1[0].set_xlabel("Epochs")
    ax1[0].set_ylabel("Train RMSE")
    ax1[0].set_title("Epochs vs Train RMSE")
    ax1[0].legend()


cols = virus_process_test.columns
plt.figure(figsize=(10,7))
for train in odd:
    ax1[1].plot(epochs, virus_process_test[cols[train]],label=cols[train])
    ax1[1].set_xlabel("Epochs")
    ax1[1].set_ylabel("Test RMSE")
    ax1[1].set_title("Epochs vs Test RMSE")
    ax1[1].legend()


 cols_train= virus_process_train.columns
 plt.figure(figsize=(10,7))
 for train in odd:
     ax2[0].plot(virus_process_train[cols[train-1]], virus_process_train
     ax2[0].set_xlabel("Time")
     ax2[0].set_ylabel("Train RMSE")
     ax2[0].set_title("Time vs Train RMSE")
     ax2[0].legend()


 cols = virus_process_test.columns
 plt.figure(figsize=(10,7))
 for train in odd:
     ax2[1].plot(virus_process_test[cols[train-1]], virus_process_test
     ax2[1].set_xlabel("Time")
     ax2[1].set_ylabel("Test RMSE")
     ax2[1].set_title("Time vs Test RMSE")
     ax2[1].legend()
```

Virus Share Dataset : Epoch vs RMSE

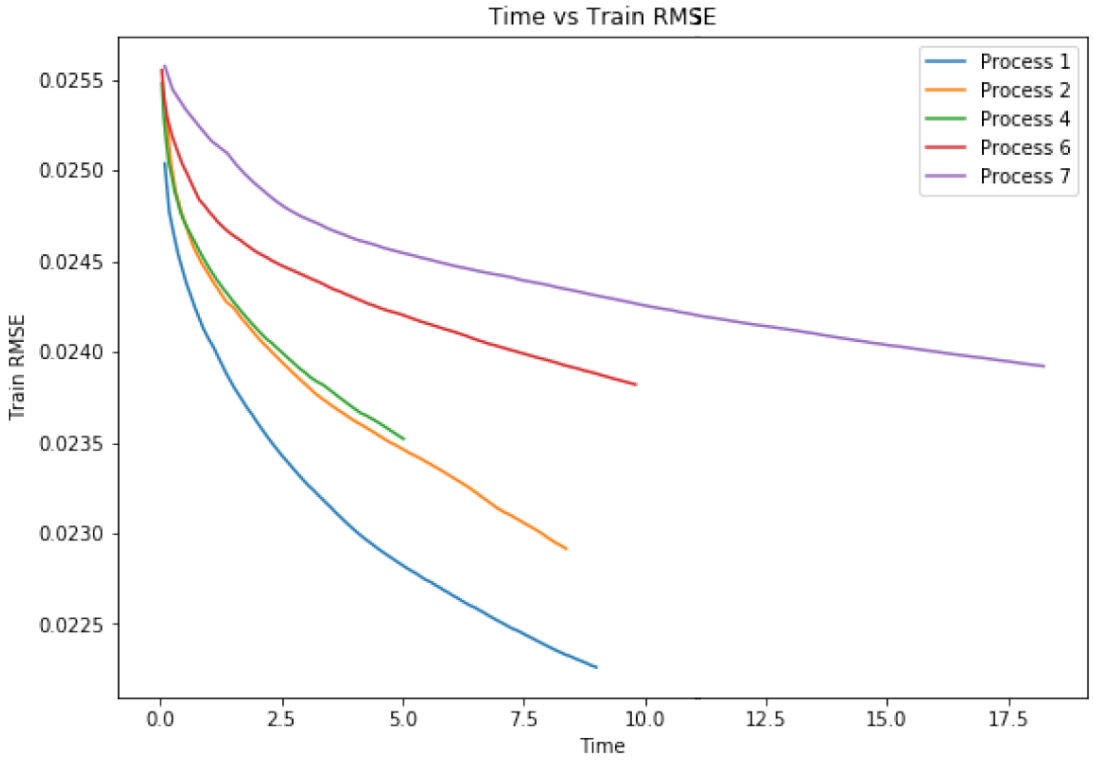


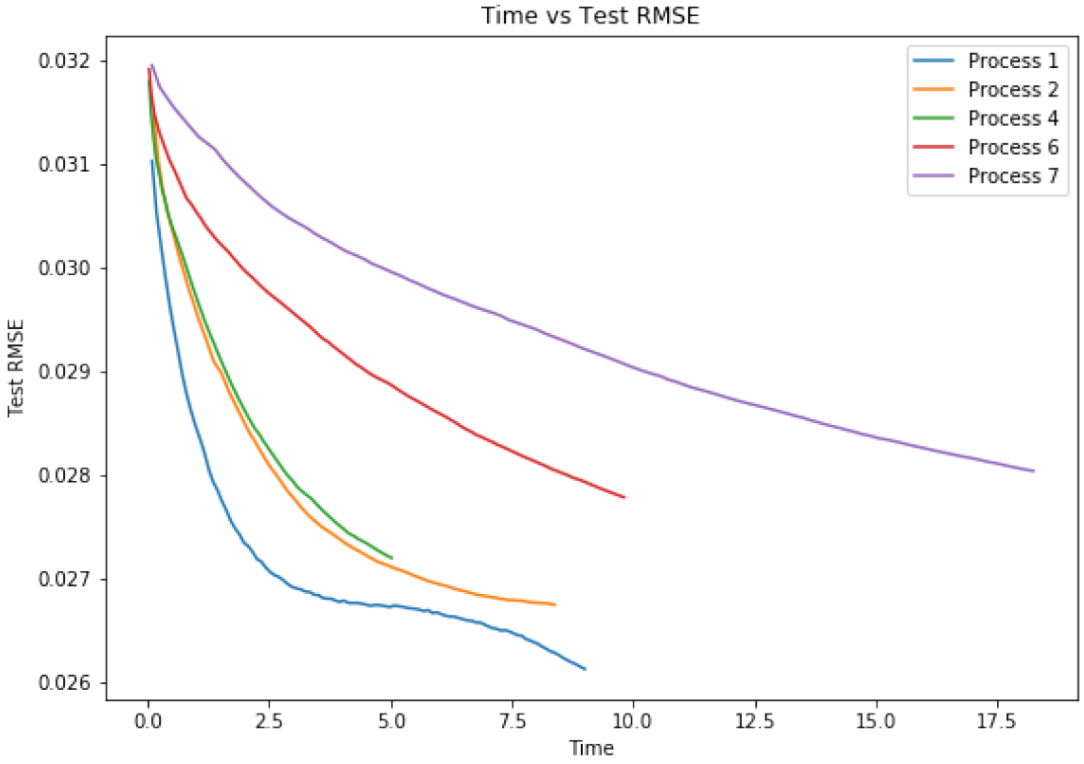
For the Virus Share dataset, from the epochs vs Train RMSE plot, it can be seen that the train RMSE is less for process being 1 and on the other hand it's really high for process 7.



This is similar to the Train RMSE vs Epoch. For the virus share dataset with Test RMSE vs Epoch, the model shows good accuracy with Process 1 and comparitively less for Process 7 compared to all the processes.

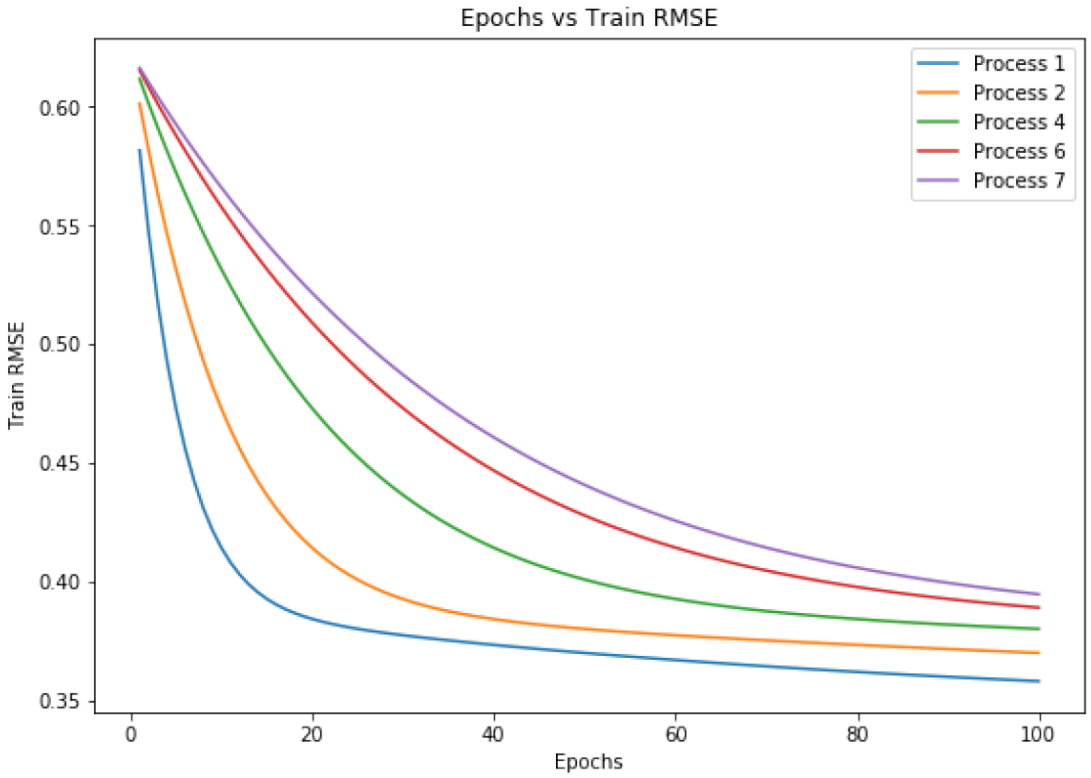
Virus Share Dataset : Time vs RMSE

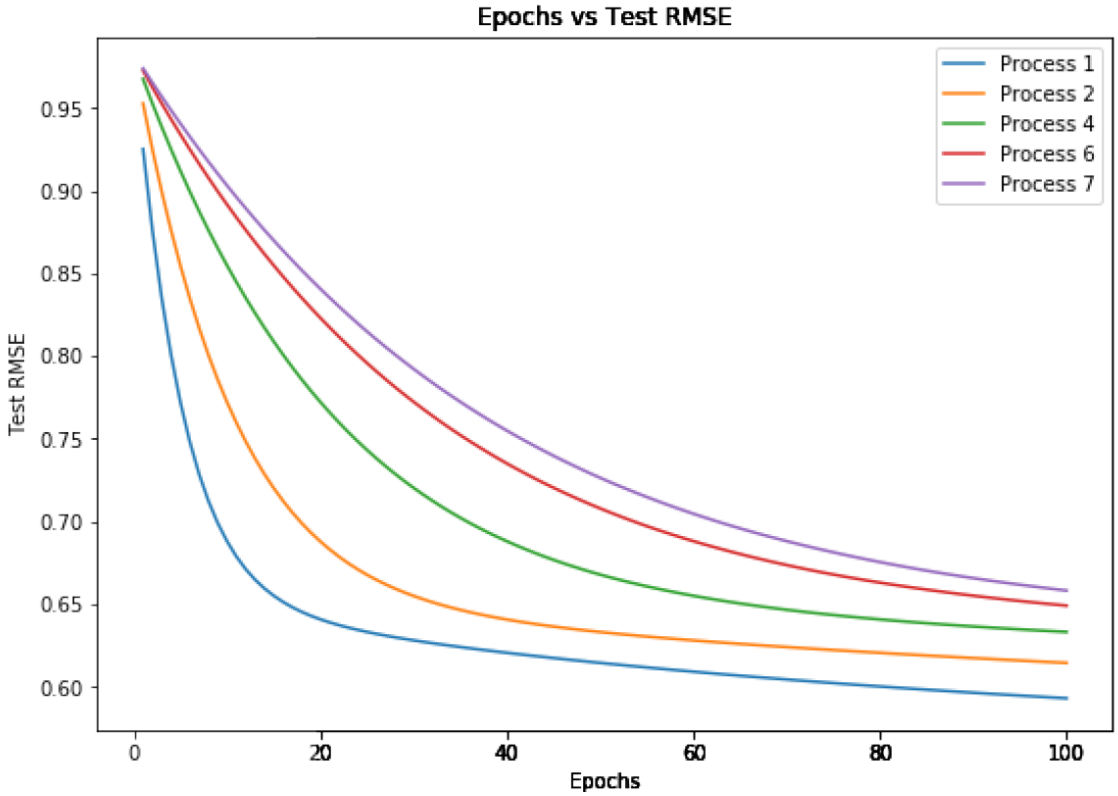




From the graphs, we can conclude that for both the training and testing sets, model with Process 4 converges faster compared to other Processes. Also, the model convergence for 100 iterations take a long time for the Process 7 which is because of the overloading of the process. Hence Process 4 is best suited to perform PSGD on my machine.

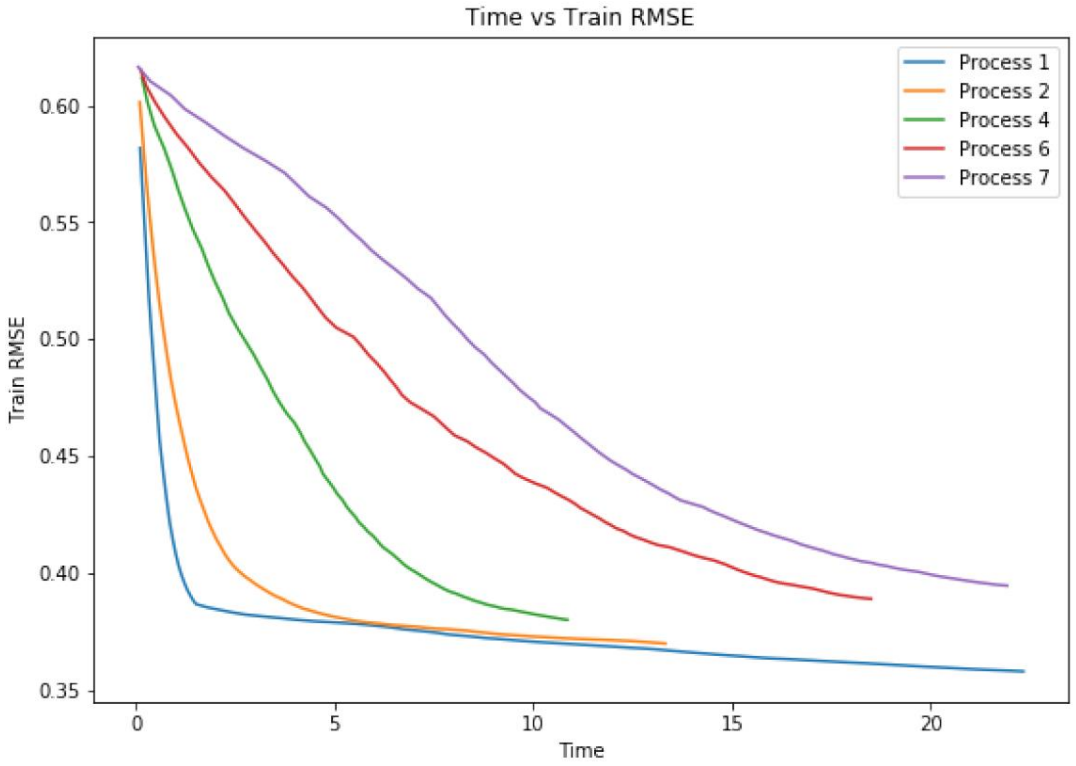
KDD Cup 1998 Dataset : Epoch vs RMSE

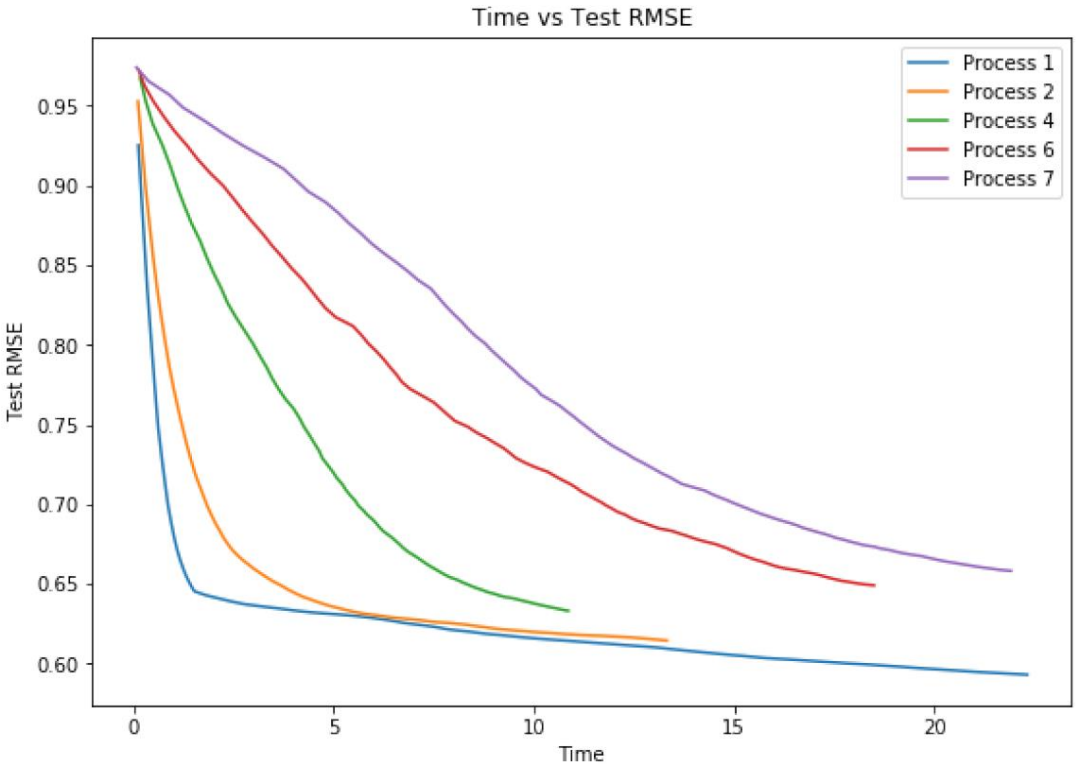




Also for KDD Cup 1998 dataset, the performance of Processes are simialr to the virus share dataset with Process 1 showing better convergence than other processes and Process 7 with poor convergence.

KDD Cup 1998 Dataset : Time vs RMSE





For the training and test dataset of KDD Cup 1998, the process 4 takes less time to complete 100 epochs whereas, the processes 1 and 7 take almost similar time to complete 100 epochs of convergence and the latter is because of the overloading on the process. Hence process 4 is good to do the Parallel Stochastic Gradient Descent on my machine.