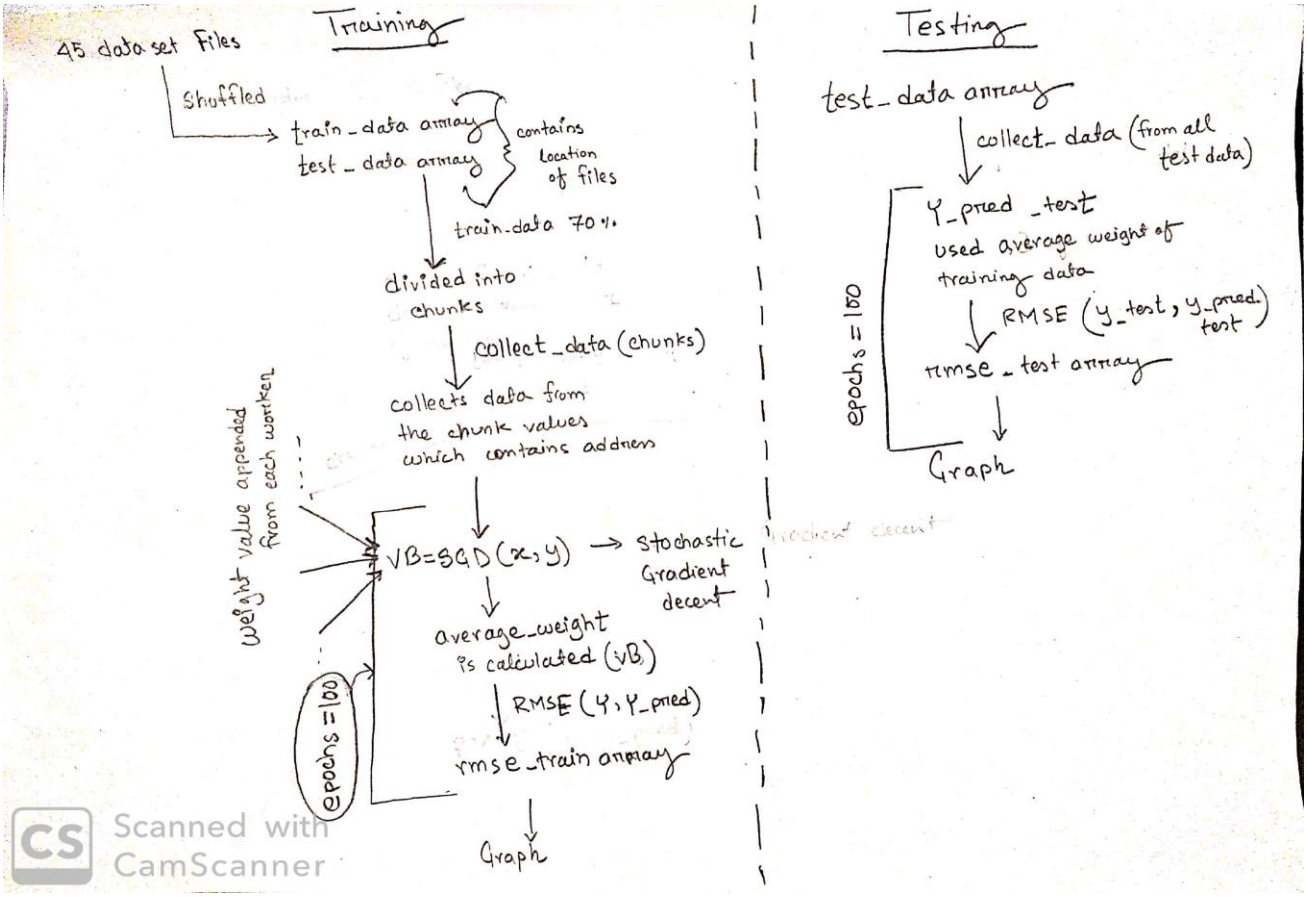
NAME-ISTIAQUE MANNAFEE SHAIKAT
MATRICULATION NUMBER-303527

1. Explaining the approach in diagram for Dynamic Features of Virus Share Executable Data Set:



2. Explaining the python code:

Functions:

A. collect_data(chunks)

```
def collect_data(chunks):
    for i in range((len(chunks))-1):
        #decoding sparse data

#         print(chunks[i])
        tup= load_svmlight_file(chunks[i])
        y=tup[1]
        x=tup[0].todense()
        if x.shape[1]<479:
            n,m = x.shape # for generality
            x0= np.zeros((n,1))
            x = np.hstack((x,x0))

    return x,y
```

This function is used to preprocess the sparse dataset using the library load_svmlight_file where chunks array contains address of the files (diagram is below).In few files the shape of features differed hence had to reshape according to weight size which is initialized

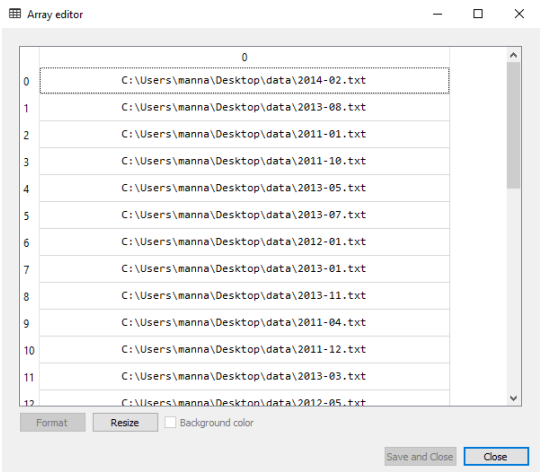


Fig: chunk array of training datasets

B. SGD(x,y) stochastic gradient decent

```
def SGD(x,y):

    w=np.zeros(x.shape[1])
    l=0.0000000001 #learning rate

    for k in range(epochs):

        print('train','number of epoch is:',k)

        indexes = np.arange(0,x.shape[0])
        shuffle(indexes)
        for i in indexes:

            y_pred_train=w.dot((np.transpose(x[i])))
            d_m=-(y[i]-y_pred_train[0,0])*x[i]
            w=w-l*d_m
        y_pred_train=w.dot(np.transpose(x))
        rmse_train.append(rmse(y,y_pred_train))
        weight_epoch.append(w)

#         print('epoch:',k,w)
    return weight_epoch


def rmse(y,y_pred):
```

This function is used to calculate the weight using differentiation function of RMSE to predict y which is label. Then, RMSE is calculated for training dataset for each chunk (code snippet given below). Afterwards, the weight is updated for each chunk and the final weight_epoch is returned.

```
def rmse(y,y_pred):
    y_pred = np.squeeze(np.asarray(y_pred))
    rmse= Math.sqrt(np.sum(pow((y-y_pred),2)))/len(y)
#     print(rmse)
    return rmse
```

Parallel coding explanation:

In the master worker the train dataset and test data set are separated then the train data sets are separated into chunks according to the number of workers.

Afterwards using collective communication the chunks are scattered to different workers to get the average weight and rmse values for each epochs.(code snippet given below)

```
if worker==0:
    files=[]
    new_weight=[]
    for file in os.listdir(file_path):
        if file.endswith(".txt"):
            files.append(os.path.join(file_path, file))
    random.shuffle(files)
    train_data=int(len(files)*0.7)
    test_data=len(files)
    chunks=np.array_split(files[0:train_data],num_workers)
#     print('chunk',chunks)
else:
    chunks=None
    files=None
    new_weight=None

vA=comm.scatter(chunks,root=0)

vB=comm.bcast(new_weight,root=0)
#vC=comm.bcast(chunks,root=0)

start = MPI.Wtime()
x,y=collect_data(vA)
vB.append(SGD(x,y))
```

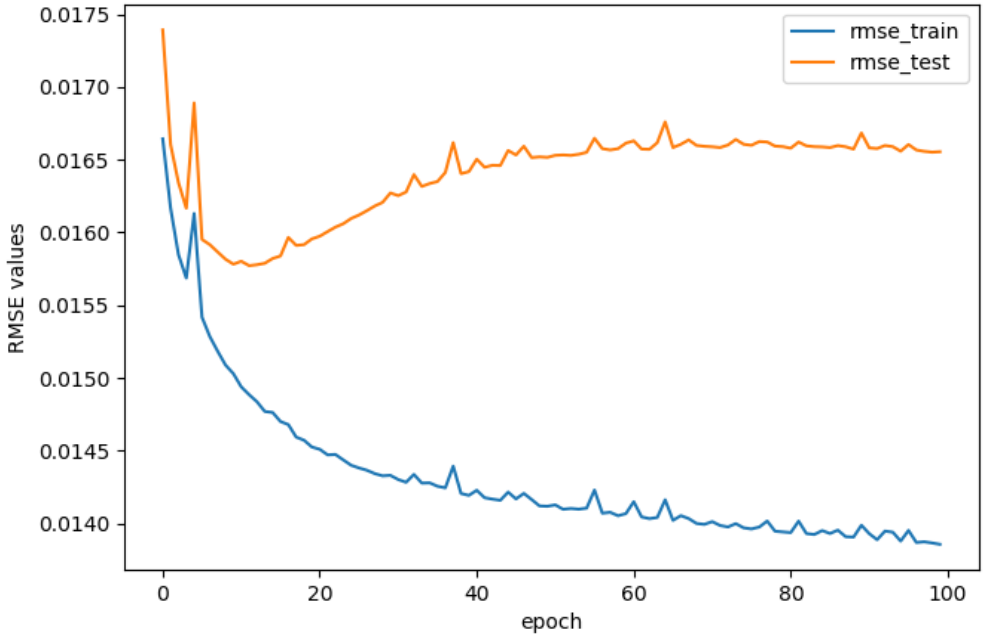
In the testing period the weight is used to predict the value of y for test data and also calculate the rmse for the test data using the original y value and predicted y values. (Code snippet given below)

```
if worker==0:
    average_weight=np.mean(vB)
    print("average weight of training data =",average_weight)
    x,y=collect_data(files[train_data:test_data])
    for k in range(epochs):
        print('test','number of epoch is:',k)

        y_pred_test=vB[0][k].dot(np.transpose(x))
        rmse_test.append(rmse(y,y_pred_test))
        end = MPI.Wtime()
    print("Runtime", end-start)
```

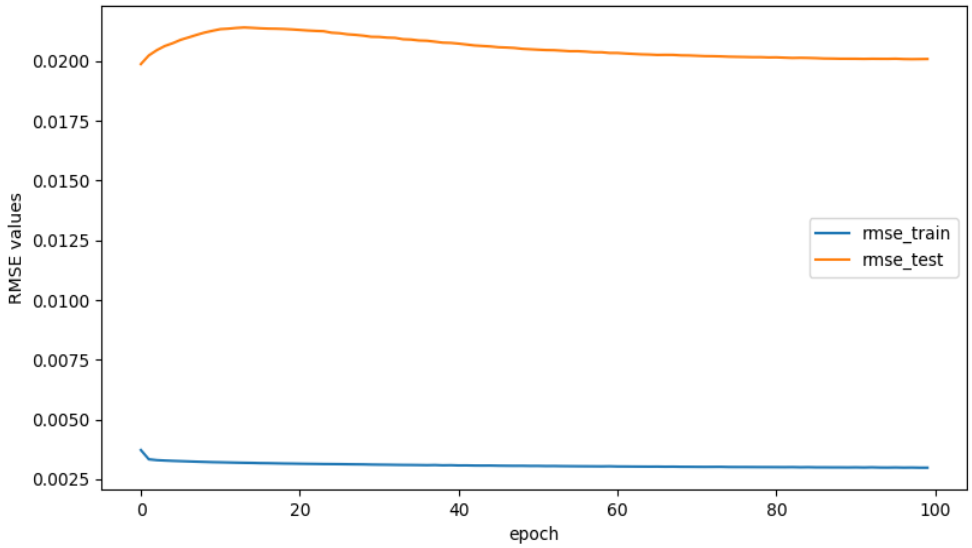
3. RESULTS:

Convergence curve for P=1 AND epochs=100



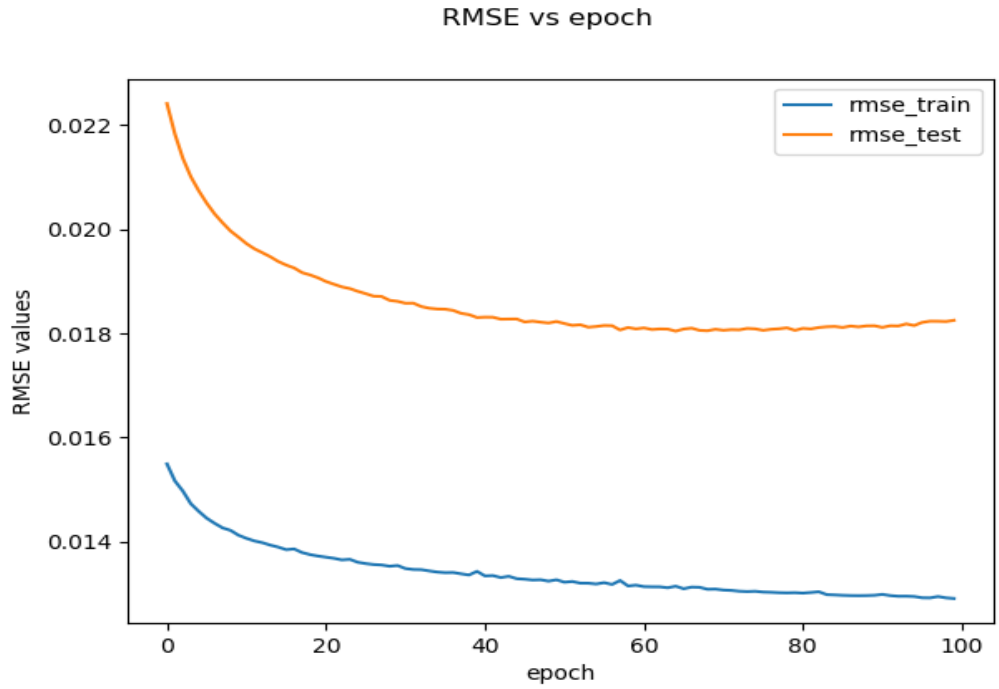
Time=28.712002599990228s

Convergence curve for P=2 AND epochs=100



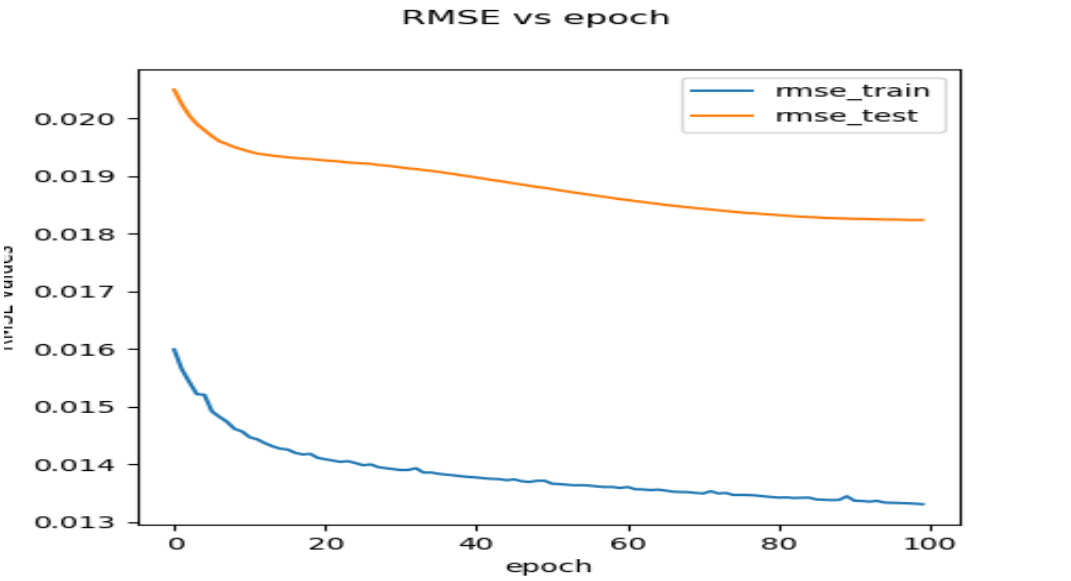
Time= 26.835568099981174s

Convergence curve for P=4 AND epochs=100



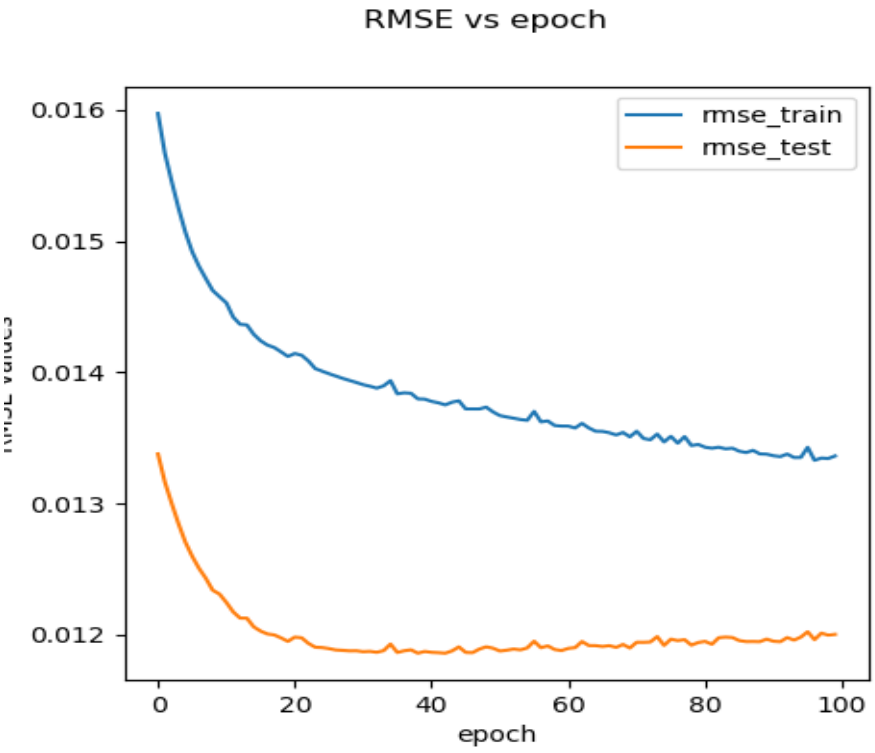
Time=14.8474599999899951s

Convergence curve for P=6 AND epochs=100

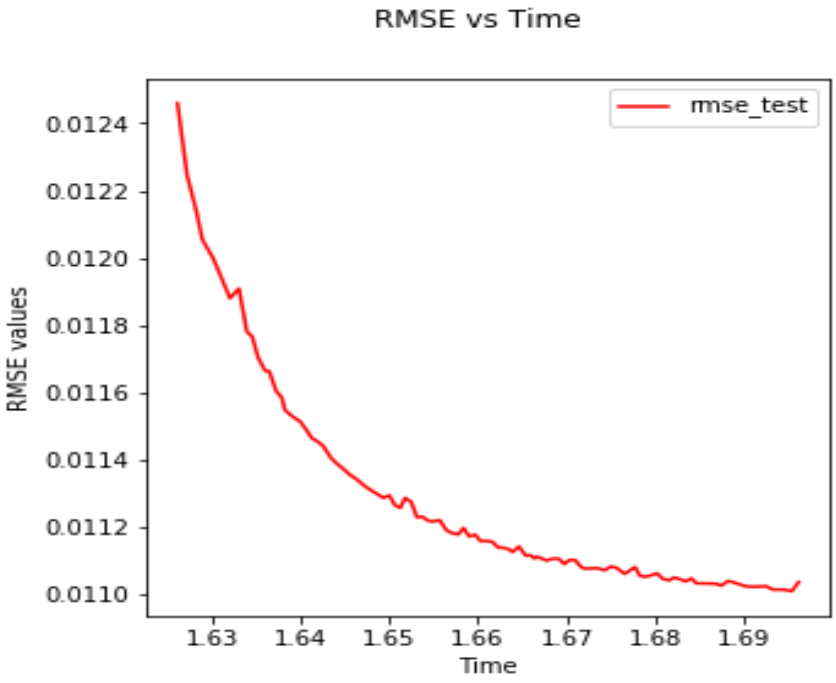


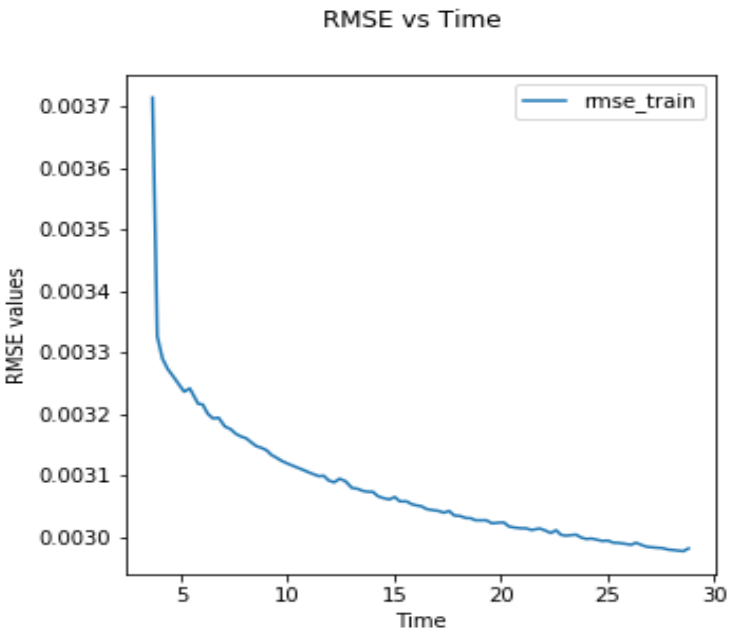
Time= 12.351702599989949s

Convergence curve for P=7 AND epochs=100



Time= 15.926856200007023





Time table for processes(100 Epoch):

P	TIME/second
1	28.712002599990228
2	26.835568099981174
4	14.847459999989951
6	12.351702599989949
7	15.926856200007023

4. **Explaining the approach for KDD Cup 1998 DataSet:**

Three steps are followed to preprocess the data

a) Selection of features-few features were selected for the regression model like 'STATE', 'PVASTATE', 'MDMAUD', 'CLUSTER', 'GENDER', 'HIT', 'DATASRCE', 'MALEMILI', 'MALEMILI', 'VIETVETS', 'WWIIVETS', 'LOCALGOV', 'STATEGOV', 'FEDGOV','CARDPROM', 'NUMPROM', 'RAMNTALL', 'NGIFTALL', 'CARDGIFT', 'AVGGIFT', 'TARGET_D'

b) Conversion of data-

df= pd.get_dummies(df)

This code is used to convert categorical features to numerical features

c) Data split-The data is spilt into train and test data using the code

X_train, X_test, y_train, y_test = train_test_split(x, y, train_size=0.7)

Where x is the dataset features and y is the label.

5. Explaining the code:

```
def collect_data():
    df= pd.read_csv('cup98LRN.txt', sep = ',')
    featureList = ['STATE', 'PVASTATE', 'MDMAUD', 'CLUSTER', 'GENDER', 'HIT',
    'DATASRCE', 'MALEMILI', 'MALEMILI', 'VIETVETS', 'WWIIVETS', 'LOCALGOV', 'STATEGOV',
    'FEDGOV','CARDPROM', 'NUMPROM', 'RAMNTALL', 'NGIFTALL', 'CARDGIFT', 'AVGGIFT', 'TARGET_D']
    df = df[featureList]
    df= pd.get_dummies(df)
    x = np.array(df.loc[:, df.columns != 'TARGET_D'])
    y = np.array(df['TARGET_D']).reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(x, y, train_size=0.7)
    chunks_x_train=np.array_split(X_train,num_workers)
    chunks_y_train=np.array_split(y_train,num_workers)

    return chunks_x_train,chunks_y_train,X_test,y_test
```

The function collect_data() is used to preprocess the data and spilt the data to training and testing data then the training data is broken in to chunks. These chunks are then scattered to different workers to build a model using SDG function which is explained in my previous dataset.

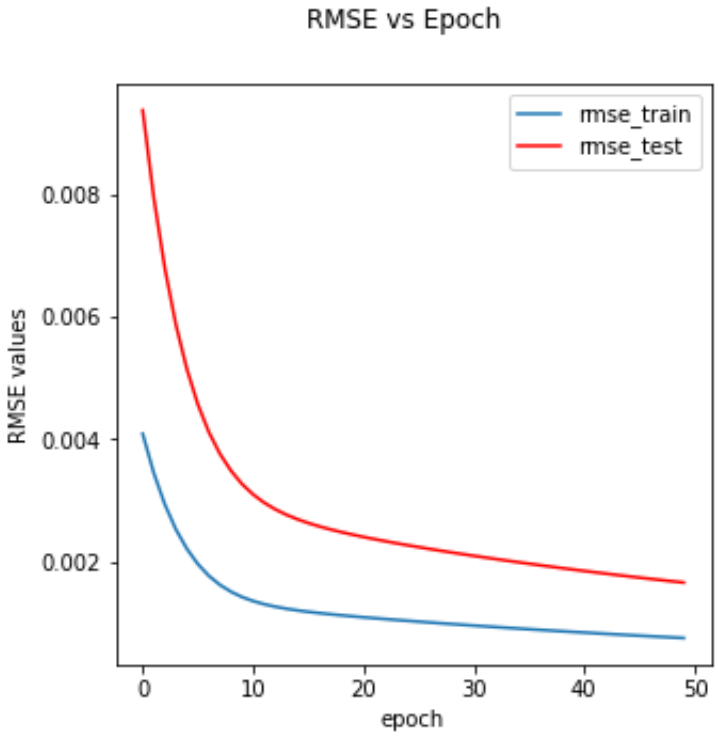
```
if worker==0:
#    average_weight=np.mean(vB)
#    print("average weight of training data =",average_weight)
#
#
    start_test=MPI.Wtime()

    for k in range(epochs):
        print('train','number of epoch is:',k)
        y_pred_test=vB[0][k].dot(np.transpose(X_test))
        rmse_test.append(rmse(y_test,y_pred_test))
        end_test = MPI.Wtime()
        time_train.append(end_test-start_test)
```

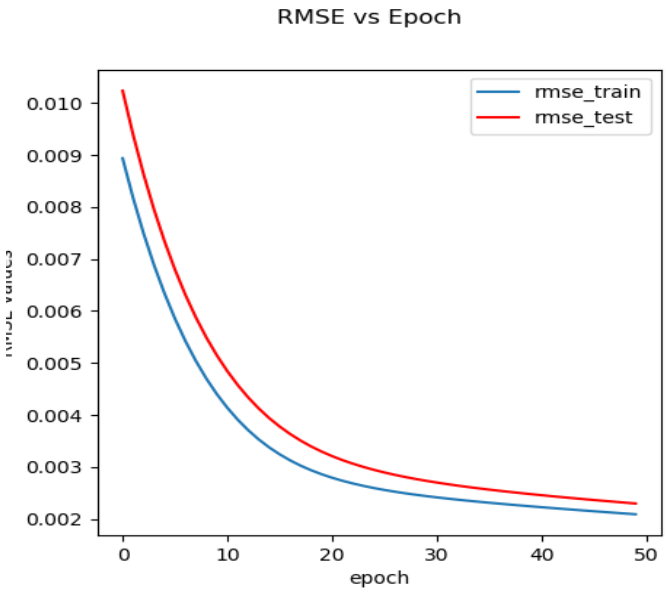
In master worker the test data is used to predict the y using the updated wait from the model in SDG. The rmse for the test data is calculated using the original y value and predicted y values.

6. RESULTS

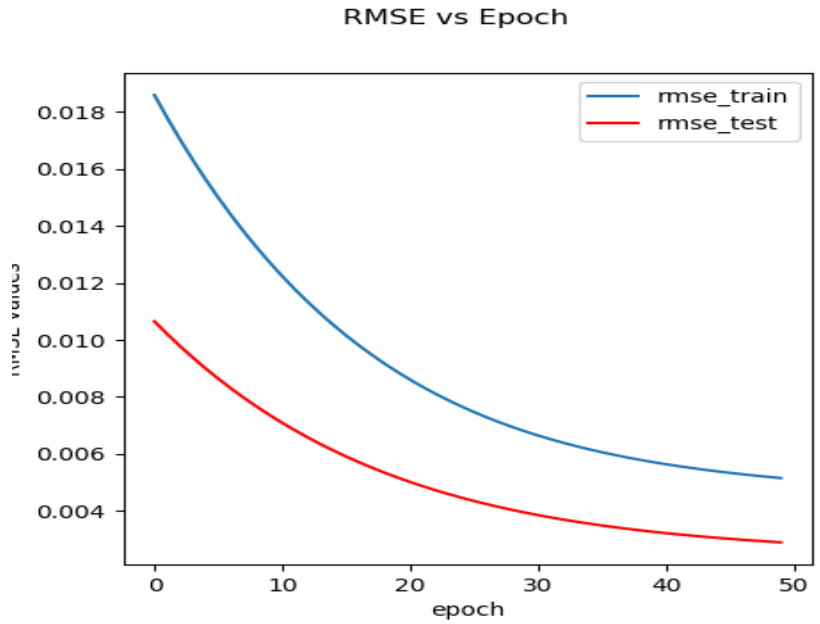
Convergence curve for P=1 AND epochs=50



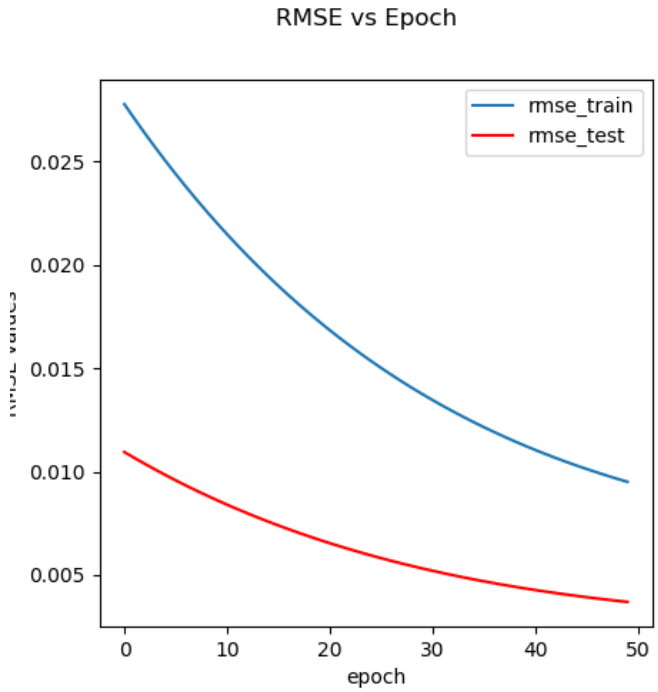
Convergence curve for P=2 AND epochs=50



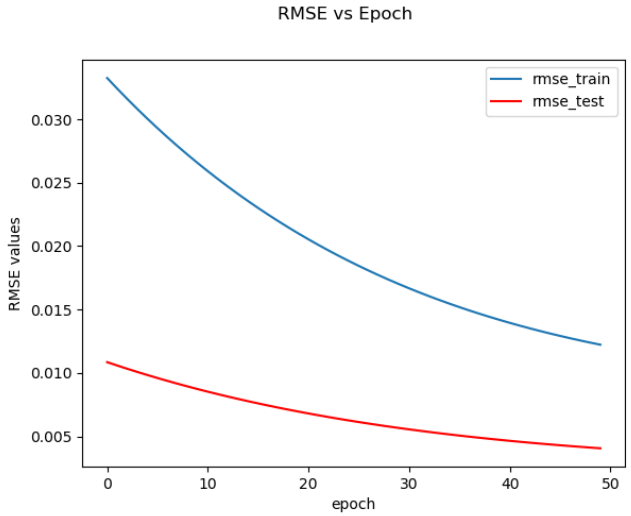
Convergence curve for P=4 AND epochs=50

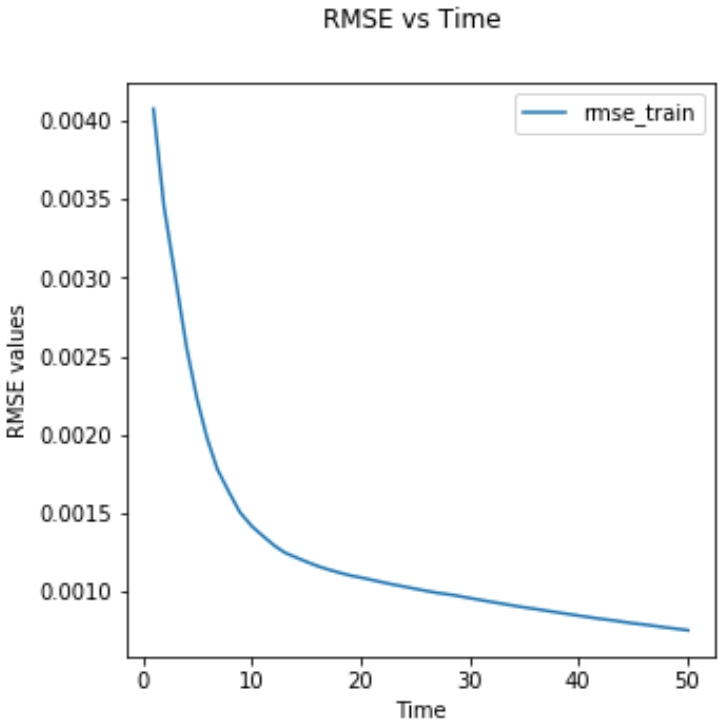


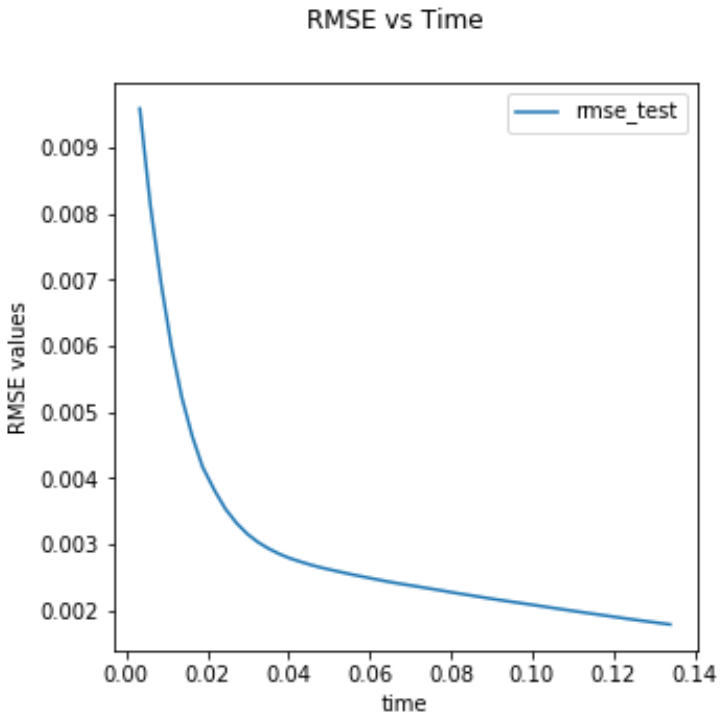
Convergence curve for P=6 AND epochs=50



Convergence curve for P=7 AND epochs=50







Time table for processes(50 EPOCHS):

P	**TIME/second**
1	56.18791069998406
2	29.60754460003227
4	19.097829699981958
6	14.111229899979662
7	12.874634000007063