Sharon Laurance

Distributed Machine Learning (Supervised)

1. Parallel Linear Regression

Input : cup98LRN dataset is read from csv file. 10000 rows are considered and all columns are considered for the parallel linear regression. But the algorithm works on the entire dataset.

Output: The train/test RMSE scores are seen for each epoch. Along with this time is noted.

Parallelization Strategy:

Collective communication is used for sending data to workers. scatter() method is used for sending the data to the workers and allreduce() method is used for averaging beta.

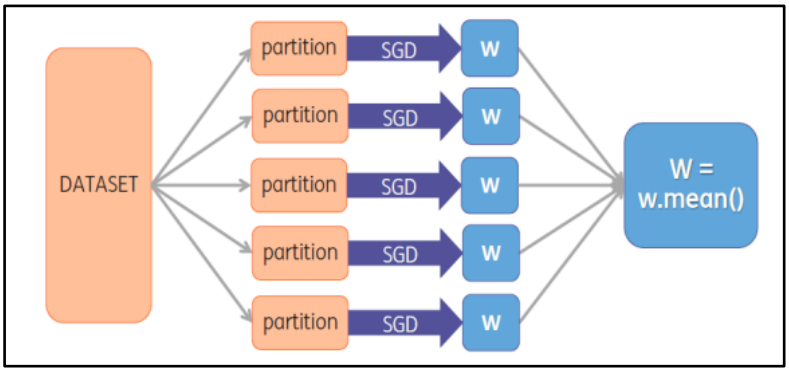


Fig1: Architecture of Parallel SGD Algorithm

Methods used for SGD are:

1. Read_data_kdd(): Used to read the data and perform the following preprocessing
    - Read the Data from the .txt file using pandas
    - Perform Label encoding
    - Normalize the Data
    - Split the data into train and test based on 70%:30% split
    - Return test and train

```python
def read_data_kdd():
    #Read the Data
    #Perform Label  encoding
    #Normalize the Data
    #Split the data into train and test based on 70%:30% split
    #Return test and train
    kdd_df=pd.read_csv("D:\\OneDrive\\Desktop\\Data Analytics\\DDA LAB\\Lab 5\\cup98LRN.txt",delimiter=',',low_memory=False)
    #Label Encoding
    for col in kdd_df.columns:
        if kdd_df[col].dtype=='object':
            kdd_df[col] = kdd_df[col].astype('category')
            kdd_df[col] = kdd_df[col].cat.codes

    # normalized_df=(kdd_df-kdd_df.mean())/kdd_df.std()
    df1=kdd_df.fillna(0)

    df1=df1.iloc[0:10000,:]
    kdd_df_train=df1.sample(frac=0.7,random_state=200) #random state is a seed value
    kdd_df_test=df1.drop(kdd_df_train.index)
    return kdd_df_train,kdd_df_test
```

```python
def find_X_y(kdd_df_train,kdd_df_test):
    X_train=kdd_df_train.drop(columns=['TARGET_D','TARGET_B','CONTROLN','RFA_2R'],inplace=False)
    y_train=kdd_df_train["TARGET_D"]
    X_test=kdd_df_test.drop(columns=['TARGET_D','TARGET_B','CONTROLN','RFA_2R'],inplace=False)
    y_test=kdd_df_test["TARGET_D"]

    X_train=np.array(X_train)
    y_train= np.array(y_train).reshape(-1,1)

    X_train=(X_train-X_train.mean())/X_train.std()
    y_train=(y_train-y_train.mean())/y_train.std()

    X_test=np.array(X_test)
    y_test= np.array(y_test).reshape(-1,1)

    X_test=(X_test-X_test.mean())/X_test.std()
    y_test=(y_test-y_test.mean())/y_test.std()

    return X_train,y_train,X_test,y_test
```

2. stochasticgradientdescent() – Here learning rate is initialized. Also in stochastic gradient descent parameter update for beta by updating one value at a time.

```python
def stochasticgradientdescent(X_train,y_train,beta):
    #Define X and y

    X=X_train
    y=y_train
    learning_rate = 1e-5
    index = np.arange(0,len(X))
    np.random.shuffle(index)
    for i in index:
        X_ele=X[i].reshape(-1,1)
        temp=np.dot(X_ele.T,beta)-y[i]
        temp=np.dot(X_ele,temp)
        gradient = temp * (2)
        beta = beta - (learning_rate * gradient)
    return beta
```

3. find_least_square () – This returns the RMSE value for the function

```
def find_least_square_test(X_test,y_test,global_beta):

    Y_predicted=np.dot(X_test,global_beta)
    mse=np.square(np.subtract(y_test,Y_predicted)).mean()
    rmse=sqrt(mse)
    return rmse

def find_least_square(X_train,y_train,global_beta):


    yhat = np.dot(X_train,global_beta)
    costtrain = np.sqrt(np.mean(pow((y_train-yhat),2)))
    return costtrain
```

4. In the following code snippet we can see data is being split and scatter operation is used to send them to the worker.

```
if rank==0:
    #After test and train is returned, split the train based on the number of workers
    start = MPI.Wtime()
    kdd_df_train,kdd_df_test=read_data_kdd() ##Reads data into a dataframe
    X_train,y_train,X_test,y_test=find_X_y(kdd_df_train,kdd_df_test)
    x_split=np.array_split(X_train,size)
    y_split=np.array_split(y_train,size)
    # splits = np.array_split(range(kdd_df_train.shape[0]),size)
    # kdd_df_train_split = [kdd_df_train.iloc[split,:] for split in splits]

else:
    x_split=None
    y_split=None


x_split=comm.scatter(x_split,0)
y_split=comm.scatter(y_split,0)
print()
beta=np.zeros([x_split.shape[1],1])
flag_check_convergence = False
cost_train = []
cost_test = []
```

5. Finally convergence criteria is checked and if it satisfies the criteria. The train and test RMSE scores are written to a dictionary and are stored in the csv file.

```
while not flag_check_convergence:

    local_beta= stochasticgradientdescent(x_split,y_split,beta)
    old_beta=beta.copy()

    global_beta=comm.allreduce(local_beta,op=MPI.SUM)
    beta = global_beta / size
    if rank==0:
        #Make two array for train rmse and test rmse
        costtrain=find_least_square(X_train,y_train,beta)
        cost_train.append(costtrain)

        costtest=find_least_square_test(X_test,y_test,beta)
        cost_test.append(costtest)

        diff1=abs(find_least_square(X_train,y_train,old_beta)-find_least_square(X_train,y_train,beta))
        if diff1<0.000005:  #Checking for Convergence criteria
            print("Inside Final Loop")
            flag_check_convergence=True
            path="D:\\OneDrive\\Desktop\\"
            df = pd.DataFrame({'TrainRMSE':cost_train,'TestRMSE': cost_test,'Time' :round(MPI.Wtime() - start,4)})
            df.to_csv(path+'KDDCup'+str(size)+'.csv')
            print("No of Workers are",size,"Time taken for execution is",round(MPI.Wtime() - start,4))
    else:
        flag_check_convergence=False

    flag_check_convergence = comm.bcast(flag_check_convergence,root=0) #Final value of flag is broadcasted to all workers.
```

2. Performance and convergence of PSGD

Below figure shows all 5 processes graph in one.

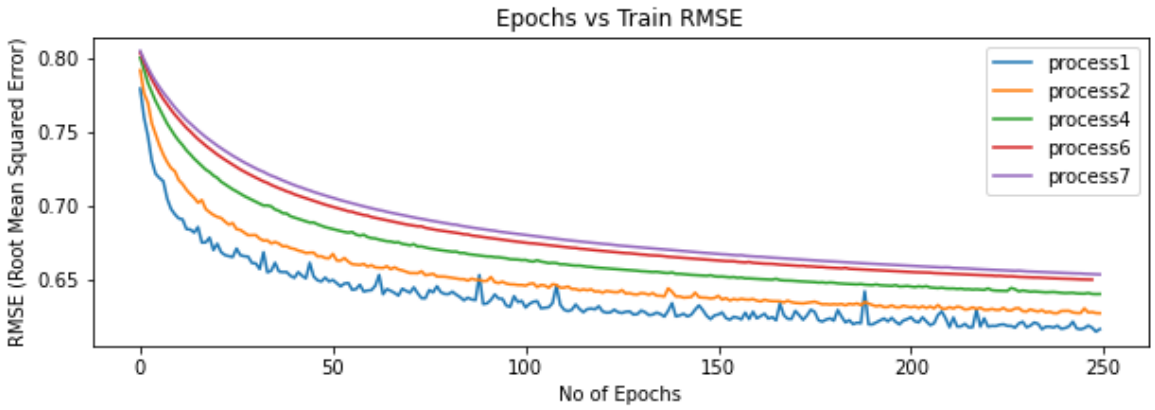


Fig2: Graph of Epoch vs Train RMSE scores

Fig 2 shows the as the epochs increase the RMSE score decreases. As this is SGD Algorithm the line is not completely straight, it has some peaks. Overall error is seen to decrease over a period of time. P={1,2,4,6,7} are shown together. Below graph shows individual Train/TEST RMSE for each process

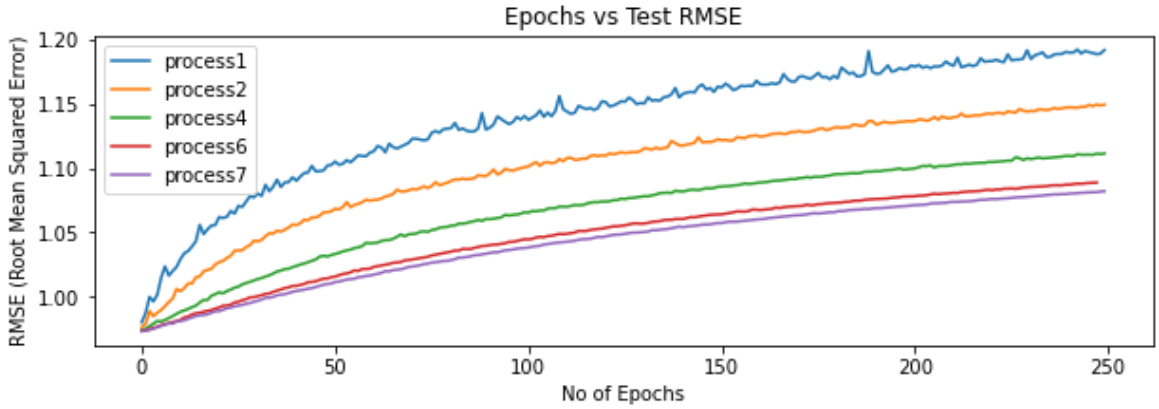


Fig3: Graph of Epoch vs Test RMSE scores

Fig 3 shows the as the epochs increase the RMSE score of Test Dataset increases. But the increase is very less. It is only 0.2. The data can be seen to overfit on training data, to modify this we can have some L2 regularization. Overall the increase is not much. P={1,2,4,6,7} are shown together. Below graph shows individual Train/TEST RMSE for each process.

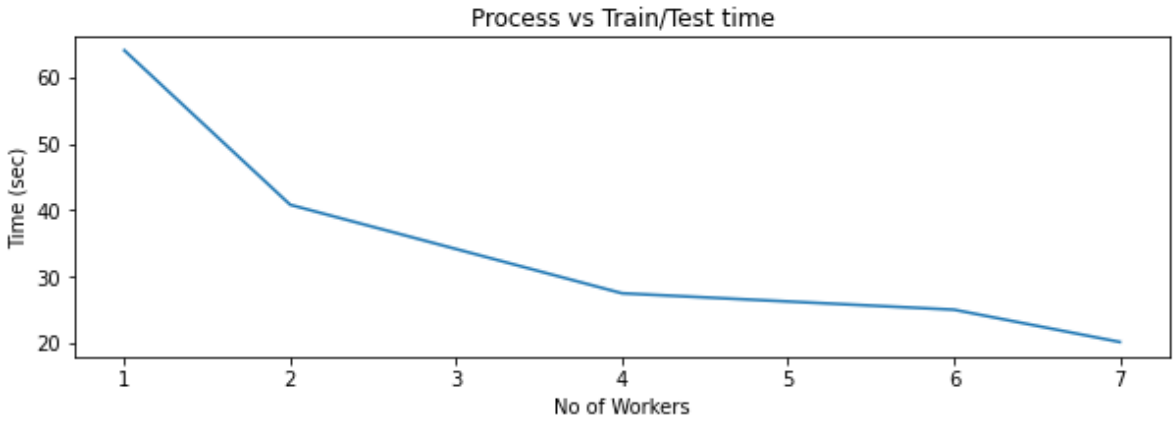
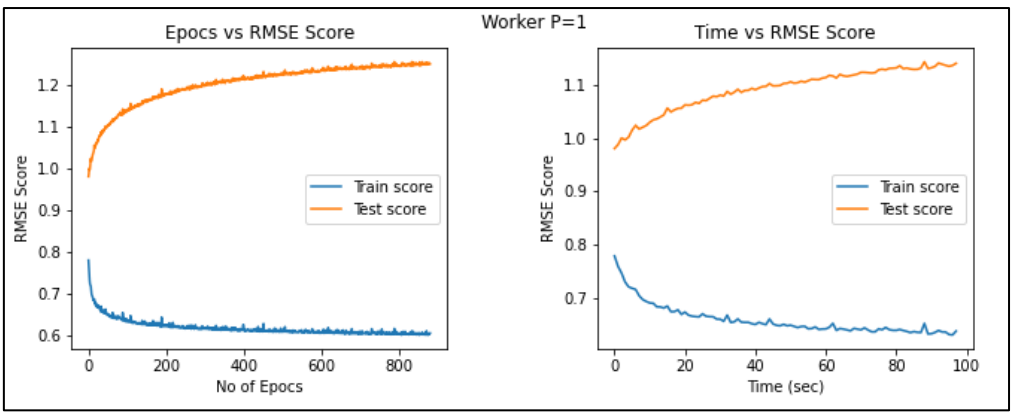


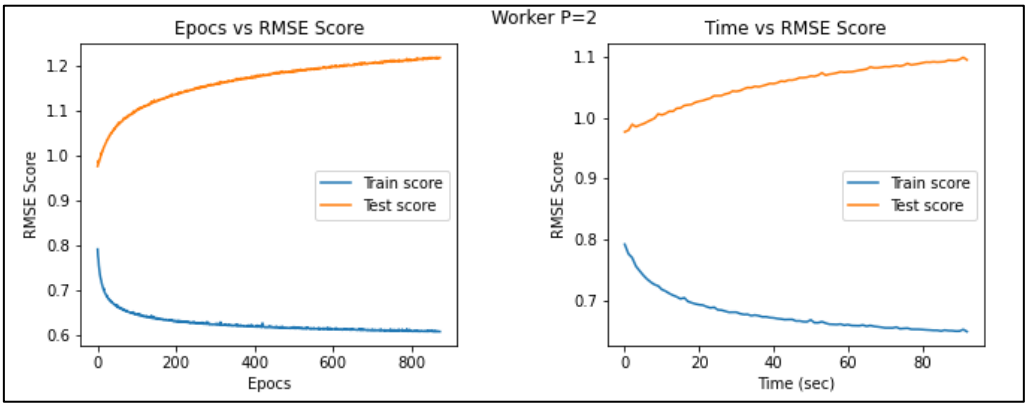
Fig4: No of Workers vs Train/Test time

Fig 4 shows the as the number of workers increase the time taken by the algorithm decreases decreases. P={1,2,4,6,7} are shown together. This shows good parallelization strategy and work being divided among workers.

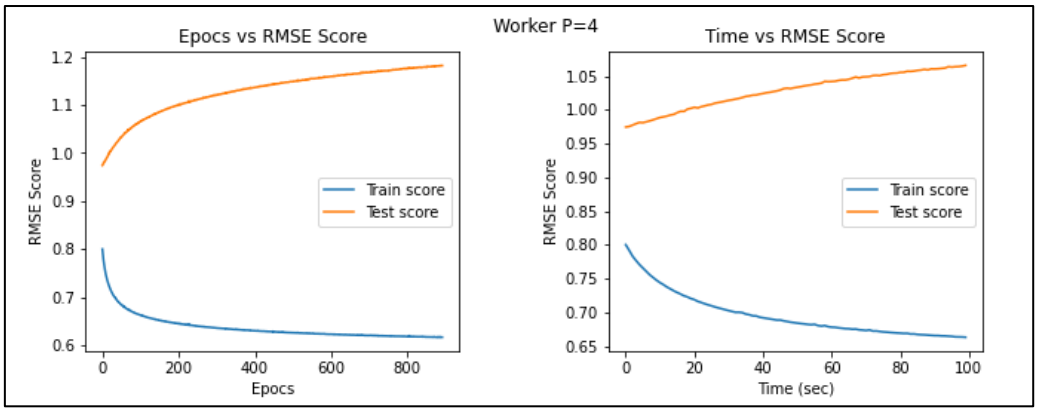
Now we see graph for each process.



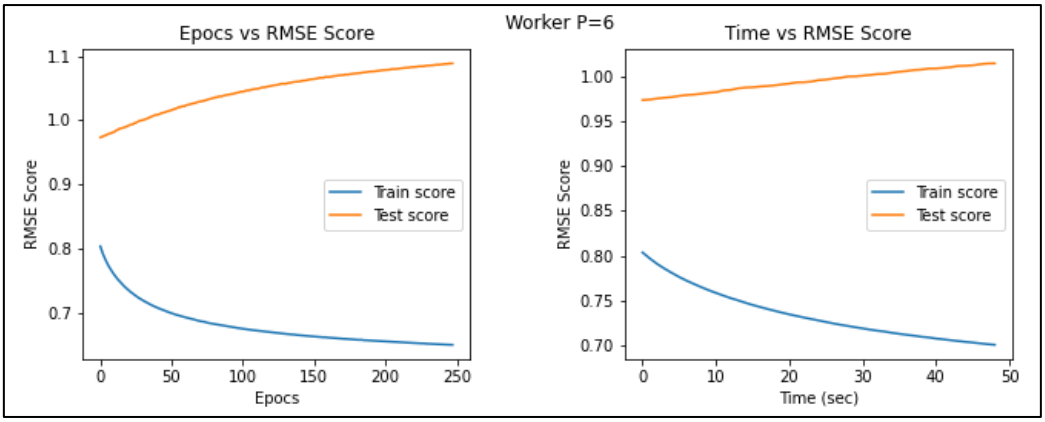
For P=1 , we see the Train RMSE decrease over the no of epoch. Also, as time increases the rmse score decreases. For the Test RMSE, the score over test dataset increases, this could mean the model overfits on test data. But the increase is 0.2 which is very less. If we employ strategies like regularisation the test error also would reduce. The graph has small peaks as this is Stochastic Gradient Descent algorithm



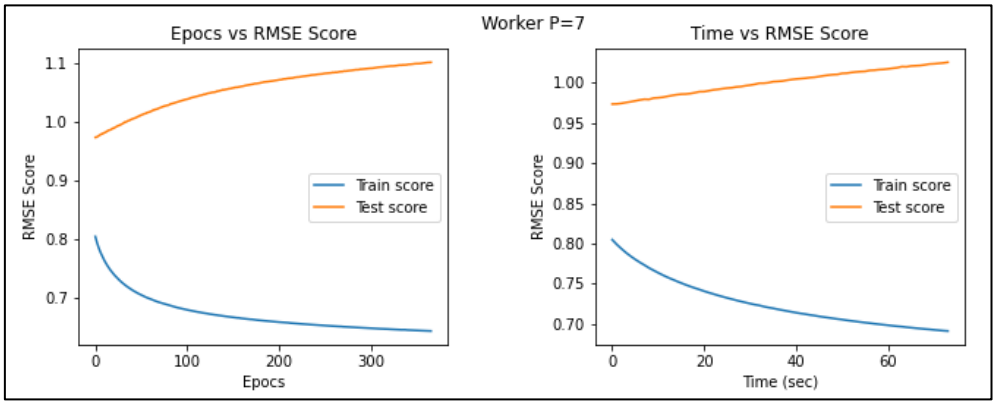
For P=2 , we see the Train RMSE decrease over the no of epoch. Also, as time increases the train rmse score decreases. For the Test RMSE, the score over test dataset increases, this could mean the model overfits on test data. But the increase is 0.1 which is very less. If we employ strategies like regularisation the test error also would reduce .



For P=4 , we see the Train RMSE decrease over the no of epoch. Also, as time increases the rmse score decreases. For the Test RMSE, the score over test dataset increases, this could mean the model overfits on test data. But the increase is 0.1 which is very less. If we employ strategies like regularisation the test error also would reduce.



For P=6 , we see the Train RMSE decrease over the no of epoch. Also, as time increases the rmse score decreases. For the Test RMSE, the score over test dataset increases, this could mean the model overfits on test data. But the increase is 0.05 which is very less. If we employ strategies like regularisation the test error also would reduce. It converges in nearly 50 sec.



For P=7 , we see the Train RMSE decrease over the no of epoch. Also, as time increases the rmse score decreases. For the Test RMSE, the score over test dataset increases, this could mean the model overfits on test data. But the increase is 0.5 which is very less. If we employ strategies like regularisation the test error also would reduce.

Reference:

- https://pbpython.com/categorical-encoding.html