Exercise 1: Linear Regression using Parallel SGD (10):

For dataset1: In my first Virus dataset there were 45 files which I have merged together into a single file. My combined text file is in libsvm format. I have used scikit functions to load this file and divided it into X as predictors and Y as targets. I have added bias to my X features and have initialised by beta variables as per the number of features in X and broad casted the beta values to all my workers. I have scattered my X and Y variable to different workers and in each worker I have further split them to 70%training(X-train, Y-train) and 30% testing(X-test, Y-test) respectively. Now I have implemented my Parallel stochastic gradient descent using MPI. My first for loop is used for the number of epochs and at each epoch I am shuffling my training data of each worker using “scikit shuffle” and in my second for loop (it runs for each row till the length of my X-train) , each worker would calculate the hypothesis and compute the respective gradient using which I update the beta values inside each worker. The updated beta values are sent to the root node from each worker using MPI.reduce function where my root node would compute the average of this beta values which I called it as global beta and would broadcast this global beta among all workers for epoch2.At each epoch, The global beta values are used to calculate the training RMSE and testing RMSE score inside each worker. The calculated Training RMSE and Testing RMSE from each worker are sent back to master root node in order to average global RMSE train score and average global RMSE test score. This process continues till my convergence condition is reached. I have checked my convergence condition as after each epoch my master node checks the absolute difference between two consecutive global RMSE Train/Test scores. If the value is less than some threshold ,then the algorithm is said to be converged. The code and results for this Dataset is shown below(Note: My RMSE train and RMSE test scores depends on my learning rate)

In [ ]:

```
"""
Created on Sun May  6 20:32:56 2018

@author: saikiran
"""
from mpi4py import MPI
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import numpy as np
import os
import math

#method to calculate the RMSE of both training and testing score.
def rmse(x,y,theta):
    hypothesis = np.dot(x,theta)
    cost = np.average((y-hypothesis) ** 2)
    rmse = (math.sqrt(cost))
    return rmse




comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.Get_size()
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)
np.random.seed(0)
if rank == 0:
    dir_list = os.listdir('C:\\Users\\saikiran\\Downloads\\dataset')
    #print(len(dir_list))
    with open('C:\\Users\\saikiran\\Downloads\\dataset_combined\\combined.txt', 'w') as out
        for fname in dir_list:
            with open('C:\\Users\\saikiran\\Downloads\\dataset\\'+fname) as infile:
                for line in infile:
                    outfile.write(line)

    data = load_svmlight_file('C:\\Users\\saikiran\\Downloads\\dataset_combined\\combined.t
    X = data[0]
    X = X.toarray()
    bias = np.ones((len(X),1), dtype=int)
    X = np.concatenate((bias,X),axis=1)
    Y = data[1]
    Y = Y.reshape(len(Y),1)
    np.random.seed(0)
    #beta = np.random.rand(X.shape[1],1)
    #beta = np.random.uniform(0.00005,0.0001,X.shape[1])
    beta = np.random.uniform(0.001,0.01,X.shape[1])
    beta = beta.reshape(len(beta),1)
    #beta = np.around(beta, decimals=3)
    a = int(len(X)/size)
    #26964
    X_chunks = [X[x:x+a] for x in range(0, len(X), a)]
    Y_chunks = [Y[x:x+a] for x in range(0, len(Y), a)]
    scat_variable_X = [(X_chunks[i])for i in range(len(X_chunks))]
    scat_variable_Y = [(Y_chunks[i])for i in range(len(Y_chunks))]
    bcast_variable_beta = beta
```

```
else:
    scat_variable_X = None
    scat_variable_Y = None
    bcast_variable_beta = None
receive_X = comm.scatter(scat_variable_X, root=0)
#alpha = 0.00000000035
alpha = 0.0000000001
receive_Y = comm.scatter(scat_variable_Y, root=0)
theta = comm.bcast(bcast_variable_beta,root=0)
X_train, X_test, Y_train, Y_test = train_test_split(receive_X, receive_Y, test_size=0.30, r
Rmse_train_list =[]
Rmse_test_list =[]
for i in range(0,20):
    x_train,y_train = shuffle(X_train,Y_train, random_state=0)
    #xTrans = x_train.transpose()
    #calculating for each row of training set
    for j in range(0,len(x_train)):
        hypothesis = np.dot(x_train[j], theta)
        gradient =  np.multiply(x_train[j].T,(y_train[j]-hypothesis))
        gradient = gradient.reshape(len(gradient),1)
        theta = theta + np.multiply(alpha,gradient)
    root = 0
    local_theta = comm.reduce(theta,root=root,op=MPI.SUM)


    if rank==root:
        global_theta = local_theta/size

    else:
        global_theta=None
        #print("global thetha after first epoch is:",global_theta)
    theta = comm.bcast(global_theta,root=root)
    theta = np.around(theta, decimals=3)
    RMSE_train = rmse(X_train,Y_train,theta)
    RMSE_test = rmse(X_test,Y_test,theta)
    Rmse_train_list = np.append(Rmse_train_list,RMSE_train)
    Rmse_test_list = np.append(Rmse_test_list,RMSE_test)
    end_time = MPI.Wtime()
    execution_time = end_time-start_time
    exe_epoch = comm.reduce(execution_time,root=root,op=MPI.MAX)
    root=0
    train_rmse = comm.reduce(Rmse_train_list,root=root,op=MPI.SUM)
    test_rmse = comm.reduce(Rmse_test_list,root=root,op=MPI.SUM)
    if rank==root:
        print("epoch ={},total execution after each epoch is{}:".format(i,exe_epoch))
        global_train_Rmse = [x / size for x in train_rmse]
        for i in range (len(global_train_Rmse)):
            if i==0:
                print("epoch={} and total train_RMSE={}".format(i,global_train_Rmse[i]))
            else:
                rmse_diff =abs(global_train_Rmse[i]-global_train_Rmse[i-1])
                if rmse_diff<0.0000001:
                    print("converged at epoch={}".format(i))
                    break
                else:
                    print("epoch={} and total train_RMSE={}".format(i,global_train_Rmse[i])

        global_test_Rmse = [y / size for y in test_rmse]
        for i in range (len(global_test_Rmse)):
            if i==0:
                print("epoch={} and total test_RMSE={}".format(i,global_test_Rmse[i]))
```

```
            else:
                rmse_test_diff = abs(global_test_Rmse[i]-global_test_Rmse[i-1])
                if rmse_test_diff<0.0000001:
                    print("converged at epoch={}".format(i))
                    break
                else:
                    print("epoch={} and total test_RMSE={}".format(i,global_test_Rmse[i]))
```

The results of this first dataset are in my result report.pdf

For dataset2: In my second KDDcup dataset i have loaded it using pandas dataframe and converted all the categorical values to numeric values and made all the preprocessing step required by initialising NAN/missiing values to 0 and divided it into X as predictors and Y as targets. I have chosen Target_D feature as my target.I have added bias to my X features and have initialised by beta variables as per the number of features in X and broad casted the beta values to all my workers. I have scattered my X and Y variable to different workers and in each worker I have further split them to 70%training(X-train, Y-train) and 30% testing(X-test, Y-test) respectively. Now I have implemented my Parallel stochastic gradient descent using MPI. My first for loop is used for the number of epochs and at each epoch I am shuffling my training data of each worker using “scikit shuffle” and in my second for loop (it runs for each row till the length of my X-train) , each worker would calculate the hypothesis and compute the respective gradient using which I update the beta values inside each worker. The updated beta values are sent to the root node from each worker using MPI.reduce function where my root node would compute the average of this beta values which I called it as global beta and would broadcast this global beta among all workers for epoch2.At each epoch, The global beta values are used to calculate the training RMSE and testing RMSE score inside each worker. The calculated Training RMSE and Testing RMSE from each worker are sent back to master root node in order to average global RMSE train score and average global RMSE test score. This process continues till my convergence condition is reached. I have checked my convergence condition as after each epoch my master node checks the absolute difference between two consecutive global RMSE Train/Test scores. If the value is less than some threshold ,then the algorithm is said to be converged. The code and results for this Dataset is shown below(Note: My RMSE train and RMSE test scores depends on my learning rate)

In [ ]:

```
# -*- coding: utf-8 -*-
"""
Created on Tue May  8 20:55:20 2018

@author: saikiran
"""

from mpi4py import MPI
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import pandas as pd
import numpy as np
import math

#method to calculate the RMSE of both training and testing score.
def rmse(x,y,theta):
    hypothesis = np.dot(x,theta)
    cost = np.average((y-hypothesis) ** 2)
    rmse = (math.sqrt(cost))
    return rmse




comm = MPI.COMM_WORLD
rank = comm.rank
size = comm.Get_size()
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)
np.random.seed(0)
if rank == 0:
    data = pd.read_csv('C:\\Users\\saikiran\\Downloads\\cup98lrn\\cup98LRN.txt',sep=",",low
    char_cols = data.dtypes.pipe(lambda x: x[x == 'object']).index
    for c in char_cols:
        data[c] = pd.factorize(data[c])[0]

    #fills all NAN values with 0
    preprocessed_data=data.fillna(0)
    Y= preprocessed_data['TARGET_D']
    #X = preprocessed_data.drop(['ODATEDW', 'DOB','TARGET_B', 'TARGET_D','HPHONE_D'], axis=
    X = preprocessed_data.drop(['TARGET_D'], axis=1)
    X = np.array(X)
    Y = np.array(Y)
    #bias = np.random.rand(len(X),1)
    bias = np.ones((len(X),1), dtype=int)
    X = np.concatenate((bias,X),axis=1)
    Y = Y.reshape(len(Y),1)
    np.random.seed(0)
    #beta = np.random.rand(X.shape[1],1)
    beta = np.random.uniform(0.0005,0.001,X.shape[1])
    beta = beta.reshape(len(beta),1)
    #beta = np.around(beta, decimals=3)
    a = int(len(X)/size)
    #23853
    X_chunks = [X[x:x+a] for x in range(0, len(X), a)]
    Y_chunks = [Y[x:x+a] for x in range(0, len(Y), a)]
    scat_variable_X = [(X_chunks[i])for i in range(len(X_chunks))]
    scat_variable_Y = [(Y_chunks[i])for i in range(len(Y_chunks))]
```

```
    bcast_variable_beta = beta
else:
    scat_variable_X = None
    scat_variable_Y = None
    bcast_variable_beta = None
receive_X = comm.scatter(scat_variable_X, root=0)
alpha = 0.00000000003
receive_Y = comm.scatter(scat_variable_Y, root=0)
theta = comm.bcast(bcast_variable_beta,root=0)
X_train, X_test, Y_train, Y_test = train_test_split(receive_X, receive_Y, test_size=0.30, r
Rmse_train_list =[]
Rmse_test_list =[]
for i in range(0,20):
    x_train,y_train = shuffle(X_train,Y_train, random_state=0)
    #xTrans = x_train.transpose()
    #calculating for each row of training set
    for j in range(0,len(x_train)):
        hypothesis = np.dot(x_train[j], theta)
        gradient =  np.multiply(x_train[j].T,(y_train[j]-hypothesis))
        gradient = gradient.reshape(len(gradient),1)
        theta = theta + np.multiply(alpha,gradient)
    root = 0
    local_theta = comm.reduce(theta,root=root,op=MPI.SUM)

    if rank==root:
        global_theta = local_theta/size

    else:
        global_theta=None

        #print("global thetha after first epoch is:",global_theta)
    theta = comm.bcast(global_theta,root=root)
    #theta = np.around(theta, decimals=3)
    RMSE_train = rmse(X_train,Y_train,theta)
    RMSE_test = rmse(X_test,Y_test,theta)
    Rmse_train_list = np.append(Rmse_train_list,RMSE_train)
    Rmse_test_list = np.append(Rmse_test_list,RMSE_test)
    root=0
    train_rmse = comm.reduce(Rmse_train_list,root=root,op=MPI.SUM)
    test_rmse = comm.reduce(Rmse_test_list,root=root,op=MPI.SUM)
    end_time = MPI.Wtime()
    execution_time = end_time-start_time
    exe_epoch = comm.reduce(execution_time,root=root,op=MPI.MAX)
    if rank==root:
        print("epoch ={},total execution after each epoch is{}:".format(i,exe_epoch))
        global_train_Rmse = [x / size for x in train_rmse]
        for i in range (len(global_train_Rmse)):
            if i==0:
                print("epoch={} and total train_RMSE={}".format(i,global_train_Rmse[i]))
            else:
                rmse_diff =abs(global_train_Rmse[i]-global_train_Rmse[i-1])
                if rmse_diff<0.01:
                    print("converged at epoch={}".format(i))
                    break
                else:
                    print("epoch={} and total train_RMSE={}".format(i,global_train_Rmse[i])

        global_test_Rmse = [y / size for y in test_rmse]
        for i in range (len(global_test_Rmse)):
            if i==0:
                print("epoch={} and total test_RMSE={}".format(i,global_test_Rmse[i]))
```

```
            else:
                rmse_test_diff = abs(global_test_Rmse[i]-global_test_Rmse[i-1])
                if rmse_test_diff<0.01:
                    print("converged at epoch={}".format(i))
                    break
                else:
                    print("epoch={} and total test_RMSE={}".format(i,global_test_Rmse[i]))
```

The results of this second dataset are in my result report.pdf

In [ ]:

```

```