# import necessary libraries

import csv

import time

from random import randrange

import pandas as pd

import matplotlib as matplotlib

import numpy as np

import statistics

import math

import operator

import matplotlib.pyplot as plt

artlargePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/artlarge/"

artsmallPath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/artsmall/"

crimePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/crime/"

winePath = "C:/Users/Abhishek Sharma/Downloads/pp2data/pp2data/wine/"

wineTrainFile = "train-wine.csv"

wineTrainRFile = "trainR-wine.csv"

wineTestFile = "test-wine.csv"

wineTestRFile = "testR-wine.csv"

crimeTrainFile = "train-crime.csv"

crimeTrainRFile = "trainR-crime.csv"

crimeTestFile = "test-crime.csv"

crimeTestRFile = "testR-crime.csv"

artsmallTrainFile = "train-artsmall.csv"

artsmallTrainRFile = "trainR-artsmall.csv"

artsmallTestFile = "test-artsmall.csv"

artsmallTestRFile = "testR-artsmall.csv"

artlargeTrainFile = "train-artlarge.csv"

artlargeTrainRFile = "trainR-artlarge.csv"

artlargeTestFile = "test-artlarge.csv"

artlargeTestRFile = "testR-artlarge.csv"

# =========================================PART 1 REGULARISATION====================================================

class regularisation:

# read Data

def readData(self, path, filename):

file = open(path + filename)

csvReader = csv.reader(file)

data = []

for row in csvReader:

data.append(row)

file.close()

return data

# function to calculate MSE

def calculateMSE(self, DataArray, rDataArray):

# Train Data Array (phi)

trainDataArray = np.array(DataArray) # creates a np array of the data set

# print('phi shape: ' + str(trainDataArray.shape))

trainDataArray = trainDataArray.astype(

np.float64) # converting elements into type float for matrix multiplication

# Data Array Transposed (phi transpose)

trainDataArrayTranspose = np.array(trainDataArray.T)

# print('phi transpose shape: ' + str(trainDataArrayTranspose.shape))

trainDataArrayTranspose = trainDataArrayTranspose.astype(

np.float64) # converting elements into type float for matrix multiplication

# TrainR Data Array (t)

trainRDataArray = np.array(rDataArray)

# print('t shape: ' + str(trainRDataArray.shape))

trainRDataArray = trainRDataArray.astype(

np.float64) # converting elements into type float for matrix multiplication

# (phi transpose \* t)

phi\_t = trainDataArrayTranspose @ trainRDataArray # '@' = matrix mult

# print('phi\_t shape: ' + str(phi\_t.shape))

# # matrix Multiplication (phi \* phi transpose)

matrixMultiplyArray = trainDataArrayTranspose @ trainDataArray

# print('matrixMultiplyArray shape: ' + str(matrixMultiplyArray.shape))

eyeSize = len(matrixMultiplyArray) # to calculate the size of the identity matrix

I = np.eye(eyeSize) # Identity Matrix

# print('I shape: ' + str(I.shape))

W = []

for l in range(151): # l =lambda Calculating w for each lambda and appending to List W

wMatrix = I \* l + matrixMultiplyArray

wMatrixInverse = np.linalg.inv(wMatrix)

w = wMatrixInverse @ phi\_t

W.append(w)

# calculate MSE

MSE = []

for w in W:

SE = ((trainRDataArray - trainDataArray @ w) \*\* 2) # SE: square error

MSE.append(np.mean(SE)) # Mean of square Error

return MSE

r = regularisation()

# ========================================DATASET: artLarge===============================================

trainData = r.readData(artlargePath, artlargeTrainFile)

trainRData = r.readData(artlargePath, artlargeTrainRFile)

trainMSE = r.calculateMSE(trainData, trainRData)

testData = r.readData(artlargePath, artlargeTestFile)

testRData = r.readData(artlargePath, artlargeTestRFile)

testMSE = r.calculateMSE(testData, testRData)

lambdaa = range(151)

plt.plot(lambdaa, trainMSE, 'r', label='trainMSE')

plt.plot(lambdaa, testMSE, 'g', label='testMSE')

plt.xlabel("Lambda")

plt.ylabel("MSE")

plt.title("MSE as a function of Lambda: (artlarge)")

plt.legend()

plt.show()

# #================================================DATASET: artSmall======================================

#

# trainData = r.readData(artsmallPath,artsmallTrainFile)

# trainRData = r.readData(artsmallPath,artsmallTrainRFile)

# trainMSE = r.calculateMSE(trainData,trainRData)

#

# testData = r.readData(artsmallPath,artsmallTestFile)

# testRData = r.readData(artsmallPath,artsmallTestRFile)

# testMSE = r.calculateMSE(testData,testRData)

#

# lambdaa = range(151)

# plt.plot(lambdaa,trainMSE,'r',label='trainMSE')

# plt.plot(lambdaa,testMSE,'g',label = 'testMSE')

# plt.xlabel("Lambda")

# plt.ylabel("MSE")

# plt.title("MSE as a function of Lambda: (artsmall)")

# plt.legend()

# plt.show()

#

# # ============================================DATASET: crime============================================

#

# trainData = r.readData(crimePath,crimeTrainFile)

# trainRData = r.readData(crimePath,crimeTrainRFile)

# trainMSE = r.calculateMSE(trainData,trainRData)

#

# testData = r.readData(crimePath,crimeTestFile)

# testRData = r.readData(crimePath,crimeTestRFile)

# testMSE = r.calculateMSE(testData,testRData)

#

# lambdaa = range(151)

# plt.plot(lambdaa,trainMSE,'r',label='trainMSE')

# plt.plot(lambdaa,testMSE,'g',label = 'testMSE')

# plt.xlabel("Lambda")

# plt.ylabel("MSE")

# plt.title("MSE as a function of Lambda: (crime)")

# plt.legend()

# plt.show()

#

#

# # ============================================DATASET: wine======================================

#

# trainData = r.readData(winePath,wineTrainFile)

# trainRData = r.readData(winePath,wineTrainRFile)

# trainMSE = r.calculateMSE(trainData,trainRData)

#

# testData = r.readData(winePath,wineTestFile)

# testRData = r.readData(winePath,wineTestRFile)

# testMSE = r.calculateMSE(testData,testRData)

#

# lambdaa = range(151)

# plt.plot(lambdaa,trainMSE,'r',label='trainMSE')

# plt.plot(lambdaa,testMSE,'g',label = 'testMSE')

# plt.xlabel("Lambda")

# plt.ylabel("MSE")

# plt.title("MSE as a function of Lambda: (wine)")

# plt.legend()

# plt.show()

## ==============================================PART 2 K-fold=========================================================

class kFold:

def cross\_validation\_split(self, dataset, folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / folds)

for i in range(folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

# read Data

def readData(self, path, filename):

file = open(path + filename)

csvReader = csv.reader(file)

data = []

for row in csvReader:

data.append(row)

file.close()

return data

def calculateMSE(self, DataArray, rDataArray, l):

# Train Data Array (phi)

trainDataArray = np.array(DataArray)

# print('phi shape: ' + str(trainDataArray.shape))

trainDataArray = trainDataArray.astype(

np.float64) # converting elements into type float for matrix multiplication

# Data Array Transposed (phi transpose)

trainDataArrayTranspose = np.array(trainDataArray.T)

# print('phi transpose shape: ' + str(trainDataArrayTranspose.shape))

trainDataArrayTranspose = trainDataArrayTranspose.astype(

np.float64) # converting elements into type float for matrix multiplication

# TrainR Data Array (t)

trainRDataArray = np.array(rDataArray)

# print('t shape: ' + str(trainRDataArray.shape))

trainRDataArray = trainRDataArray.astype(

np.float64) # converting elements into type float for matrix multiplication

# (phi transpose \* t)

phi\_t = trainDataArrayTranspose @ trainRDataArray # '@' = matrix mult

# print('phi\_t shape: ' + str(phi\_t.shape))

# # matrix Multiplication (phi \* phi trans)

matrixMultiplyArray = trainDataArrayTranspose @ trainDataArray

# print('matrixMultiplyArray shape: ' + str(matrixMultiplyArray.shape))

eyeSize = len(matrixMultiplyArray)

I = np.eye(eyeSize) # Identity Matrix

# print('I shape: ' + str(I.shape))

W = []

wMatrix = I \* (l + 1) + matrixMultiplyArray

wMatrixInverse = np.linalg.inv(wMatrix)

w = wMatrixInverse @ phi\_t

# calculate SE

SE = ((trainRDataArray - trainDataArray @ w) \*\* 2) # square error

MSE = (np.mean(SE))

return MSE

k = kFold()

start = time.time()

# ============================DATASET: ARTSMALL===========================================

totalTrainData = np.array(k.readData(artsmallPath, artsmallTrainFile))

splittedTrainData = np.array\_split(totalTrainData, 10)

totalTrainRData = np.array(k.readData(artsmallPath, artsmallTrainRFile))

splittedTrainRData = np.array\_split(totalTrainRData, 10)

MSEAverageList = []

lambdaList = []

# loop for splitting the dataset and calculating MSE

for l in range(150):

MSEList = []

for i in range(10):

train = splittedTrainData[i]

test = splittedTrainData[-i]

trainR = splittedTrainRData[i]

testR = splittedTrainRData[-i]

MSE = k.calculateMSE(train, trainR, l)

MSEList.append(MSE)

MSEAverage = statistics.mean(MSEList)

MSEAverageList.append(MSEAverage)

lambdaList.append(l)

# now find the lowest MSE Average and its corresponding lambda

lowestMSEAverage = min(MSEAverageList)

index = MSEAverageList.index(lowestMSEAverage)

lowestLambda = lambdaList[index]

end = time.time()

print("Dataset: ARTSMALL")

print("Lowest MSE Average : " + str(lowestMSEAverage))

print("Corresponding value of Lambda: " + str(lowestLambda))

print("Runtime: " + str(end - start) + " seconds")

# #============================DATASET: ARTLARGE===========================================

#

# totalTrainData = np.array(k.readData(artlargePath, artlargeTrainFile))

# splittedTrainData = np.array\_split(totalTrainData, 10)

#

# totalTrainRData = np.array(k.readData(artlargePath, artlargeTrainRFile))

# splittedTrainRData = np.array\_split(totalTrainRData, 10)

#

# MSEAverageList = []

# lambdaList = []

#

# # loop for splitting the dataset and calculating MSE

# for l in range(150):

# MSEList = []

# for i in range(10):

# train = splittedTrainData[i]

# test = splittedTrainData[-i]

#

# trainR = splittedTrainRData[i]

# testR = splittedTrainRData[-i]

#

# MSE = k.calculateMSE(train, trainR, l)

# MSEList.append(MSE)

#

# MSEAverage = statistics.mean(MSEList)

# MSEAverageList.append(MSEAverage)

# lambdaList.append(l)

#

# # now find the lowest MSE Average and its corresponding lambda

# lowestMSEAverage = min(MSEAverageList)

# index = MSEAverageList.index(lowestMSEAverage)

# lowestLambda = lambdaList[index]

#

# end = time.time()

#

# print("Dataset: ARTLARGE")

# print("Lowest MSE Average : " + str(lowestMSEAverage))

# print("Corresponding value of Lambda: " + str(lowestLambda))

# print("Runtime: "+str(end-start)+" seconds")

# #======================================DATASET: WINE=========================================================

#

# totalTrainData = np.array(k.readData(winePath, wineTrainFile))

# splittedTrainData = np.array\_split(totalTrainData, 10)

#

# totalTrainRData = np.array(k.readData(winePath, wineTrainRFile))

# splittedTrainRData = np.array\_split(totalTrainRData, 10)

#

# MSEAverageList = []

# lambdaList = []

#

# # loop for splitting the dataset and calculating MSE

# for l in range(150):

# MSEList = []

# for i in range(10):

# train = splittedTrainData[i]

# test = splittedTrainData[-i]

#

# trainR = splittedTrainRData[i]

# testR = splittedTrainRData[-i]

#

# MSE = k.calculateMSE(train, trainR, l)

# MSEList.append(MSE)

#

# MSEAverage = statistics.mean(MSEList)

# MSEAverageList.append(MSEAverage)

# lambdaList.append(l)

#

# # now find the lowest MSE Average and its corresponding lambda

# lowestMSEAverage = min(MSEAverageList)

# index = MSEAverageList.index(lowestMSEAverage)

# lowestLambda = lambdaList[index]

#

# end = time.time()

#

# print("Dataset: WINE")

# print("Lowest MSE Average : " + str(lowestMSEAverage))

# print("Corresponding value of Lambda: " + str(lowestLambda))

# print("Runtime: "+str(end-start)+" seconds")

#

# #======================================DATASET: CRIME=========================================================

#

# totalTrainData = np.array(k.readData(crimePath, crimeTrainFile))

# splittedTrainData = np.array\_split(totalTrainData, 10)

#

# totalTrainRData = np.array(k.readData(crimePath, crimeTrainRFile))

# splittedTrainRData = np.array\_split(totalTrainRData, 10)

#

# MSEAverageList = []

# lambdaList = []

#

# # loop for splitting the dataset and calculating MSE

# for l in range(150):

# MSEList = []

# for i in range(10):

# train = splittedTrainData[i]

# test = splittedTrainData[-i]

#

# trainR = splittedTrainRData[i]

# testR = splittedTrainRData[-i]

#

# MSE = k.calculateMSE(train, trainR, l)

# MSEList.append(MSE)

#

# MSEAverage = statistics.mean(MSEList)

# MSEAverageList.append(MSEAverage)

# lambdaList.append(l)

#

# # now find the lowest MSE Average and its corresponding lambda

# lowestMSEAverage = min(MSEAverageList)

# index = MSEAverageList.index(lowestMSEAverage)

# lowestLambda = lambdaList[index]

#

# end = time.time()

#

# print("Dataset: CRIME")

# print("Lowest MSE Average : " + str(lowestMSEAverage))

# print("Corresponding value of Lambda: " + str(lowestLambda))

# print("Runtime: "+str(end-start)+" seconds")

# ===============================================PART 3============================================================

class bayes:

# read Data

def readData(self, path, filename):

file = open(path + filename)

csvReader = csv.reader(file)

data = []

for row in csvReader:

data.append(row)

file.close()

return data

def calculateBphiTphi(self, beta, phiTphi): # Computing Beta \* (PhiTranspose \* Phi)

BphiTphi = beta \* phiTphi

return BphiTphi

def calculateGamma(self, BphiTphi, alpha): # Computing Gamma for calculating alpha and beta

eig = np.linalg.eigvals(BphiTphi)

gamma = sum([e / (e + alpha) for e in eig])

return gamma

def calculateMn(self, B, Sn, phiT\_t): # Computing Mn

Mn = B \* (Sn @ phiT\_t)

return Mn

def calculateSn(self, alpha, beta, phi, phiTphi): # Computing Sn

I = np.identity(np.shape(phi)[1])

Sn\_Inverse = alpha \* I + beta \* phiTphi

Sn = np.linalg.inv(Sn\_Inverse)

return Sn

def calculateAlpha(self, gamma, mn): # Computing Alpha

mnT = mn.T

mnTmn = mnT @ mn

alpha = gamma / mnTmn

return alpha

def calculateBeta(self, phi, t, Mn, gamma): # Computing Beta

beta\_row\_list = []

N = np.shape(phi)[0]

for i in range(0, phi.shape[0]):

phi\_i = phi[i]

t\_i = t[i]

beta\_row\_value = np.square(t\_i - (Mn.T @ phi\_i))

beta\_row\_list.append(beta\_row\_value)

beta = 1 / ((1 / (N - gamma)) \* sum(beta\_row\_list))

return beta

def calculateMSE(self, phi, Mn, t): # Calculating MSE

SE = (phi @ Mn - t) \*\* 2

MSE = np.mean(SE)

return MSE

b = bayes()

start = time.time()

# ============================================DATASET: WINE======================================================

dataarray = b.readData(winePath, wineTrainFile)

t = np.array(b.readData(winePath, wineTrainRFile))

t = t.astype(np.float64)

phi = np.array(dataarray)

phi = phi.astype(np.float64)

phiTranspose = phi.T

phiTranspose = phiTranspose.astype(np.float64)

phiTphi = phiTranspose @ phi

phiT\_t = phiTranspose @ t

alpha = 5.0

beta = 1.0

oldAlpha = 0

while alpha - oldAlpha > 0.0001:

Sn = b.calculateSn(alpha, beta, phi, phiTphi)

Mn = b.calculateMn(beta, Sn, phiT\_t)

BphiTphi = b.calculateBphiTphi(beta, phiTphi)

gamma = b.calculateGamma(BphiTphi, alpha)

oldAlpha = alpha

alpha = b.calculateAlpha(gamma, Mn)

beta = b.calculateBeta(phi, t, Mn, gamma)

lambdaa = alpha / beta

MSE = b.calculateMSE(phi, Mn, t)

end = time.time()

print("Dataset Used: Wine")

print("Value of alpha: " + str(alpha))

print("Value of beta: " + str(beta))

print("Value of Lambda: " + str(lambdaa))

print("Associated MSE:" + str(MSE))

print("Runtime:" + str(end - start))

# #===============================================DATASET: CRIME==========================================

#

# dataarray = b.readData(crimePath, crimeTrainFile)

# t = np.array(b.readData(crimePath, crimeTrainRFile))

# t = t.astype(np.float64)

# phi = np.array(dataarray)

# phi = phi.astype(np.float64)

# phiTranspose = phi.T

# phiTranspose = phiTranspose.astype(np.float64)

# phiTphi = phiTranspose @ phi

# phiT\_t = phiTranspose @ t

#

# alpha = 5.0

# beta = 1.0

#

# oldAlpha = 0

# while alpha-oldAlpha > 0.0001:

# Sn = b.calculateSn(alpha, beta, phi, phiTphi)

# Mn = b.calculateMn(beta, Sn, phiT\_t)

# BphiTphi = b.calculateBphiTphi(beta, phiTphi)

# gamma = b.calculateGamma(BphiTphi, alpha)

# oldAlpha = alpha

# alpha = b.calculateAlpha(gamma, Mn)

# beta = b.calculateBeta(phi, t, Mn, gamma)

#

# lambdaa = alpha/beta

#

# MSE = b.calculateMSE(phi,Mn,t)

#

# end = time.time()

# print("Dataset Used: Crime")

# print("Value of alpha: " + str(alpha))

# print("Value of beta: " + str(beta))

# print("Value of Lambda: " + str(lambdaa))

# print("Associated MSE:" + str(MSE))

# print("Runtime:" + str(end-start))

#

#

# #=================================================DATASET: ARTSMALL=================================================

#

# dataarray = b.readData(artsmallPath, artsmallTrainFile)

# t = np.array(b.readData(artsmallPath, artsmallTrainRFile))

# t = t.astype(np.float64)

# phi = np.array(dataarray)

# phi = phi.astype(np.float64)

# phiTranspose = phi.T

# phiTranspose = phiTranspose.astype(np.float64)

# phiTphi = phiTranspose @ phi

# phiT\_t = phiTranspose @ t

#

# alpha = 5.0

# beta = 1.0

#

# oldAlpha = 0

# while alpha-oldAlpha > 0.0001:

# Sn = b.calculateSn(alpha, beta, phi, phiTphi)

# Mn = b.calculateMn(beta, Sn, phiT\_t)

# BphiTphi = b.calculateBphiTphi(beta, phiTphi)

# gamma = b.calculateGamma(BphiTphi, alpha)

# oldAlpha = alpha

# alpha = b.calculateAlpha(gamma, Mn)

# beta = b.calculateBeta(phi, t, Mn, gamma)

#

# lambdaa = alpha/beta

#

# MSE = b.calculateMSE(phi,Mn,t)

#

# end = time.time()

# print("Dataset Used: ArtSmall")

# print("Value of alpha: " + str(alpha))

# print("Value of beta: " + str(beta))

# print("Value of Lambda: " + str(lambdaa))

# print("Associated MSE:" + str(MSE))

# print("Runtime:" + str(end-start))

#

#

# #========================================================DATASET: ARTLARGE============================================

#

# dataarray = b.readData(artlargePath, artlargeTrainFile)

# t = np.array(b.readData(artlargePath, artlargeTrainRFile))

# t = t.astype(np.float64)

# phi = np.array(dataarray)

# phi = phi.astype(np.float64)

# phiTranspose = phi.T

# phiTranspose = phiTranspose.astype(np.float64)

# phiTphi = phiTranspose @ phi

# phiT\_t = phiTranspose @ t

#

# alpha = 5.0

# beta = 1.0

#

# oldAlpha = 0

# while alpha - oldAlpha > 0.0001:

# Sn = b.calculateSn(alpha, beta, phi, phiTphi)

# Mn = b.calculateMn(beta, Sn, phiT\_t)

# BphiTphi = b.calculateBphiTphi(beta, phiTphi)

# gamma = b.calculateGamma(BphiTphi, alpha)

# oldAlpha = alpha

# alpha = b.calculateAlpha(gamma, Mn)

# beta = b.calculateBeta(phi, t, Mn, gamma)

#

# lambdaa = alpha / beta

#

# MSE = b.calculateMSE(phi, Mn, t)

# end = time.time()

# print("Dataset Used: ArtLarge")

# print("Value of alpha: " + str(alpha))

# print("Value of beta: " + str(beta))

# print("Value of Lambda: " + str(lambdaa))

# print("Associated MSE:" + str(MSE))

# print("Runtime:" + str(end - start))

#