import numpy as np

import math

class LinearRegression:

def \_\_init\_\_(self, dim):

self.weights = np.zeros((dim + 1,1))

def X\_new(self,X):

examples = X.shape[0]

X\_new = np.c\_[np.ones(examples), X]

return X\_new

def train(self,X,Y):

X\_new = self.X\_new(X)

pinv\_X = np.linalg.pinv(X\_new) # pseudo inverse

self.weights = np.dot(pinv\_X,Y)

def predict(self,X):

X\_new = self.X\_new(X)

h = np.matmul(X\_new, self.weights)

return h

class NonlinearTrans(LinearRegression):

def \_\_init\_\_(self, dim, lamda, k):

self.dim = (2\*dim + 3) # 1, x1, x2, x1^2, x2^2, x1\*x2, abs(x1-x2), abs(x1+x2)

self.weights = np.zeros((self.dim + 1, 1))

self.lamda = lamda

self.k = k

def change\_lambda(self, lamda):

self.lamda = lamda

def change\_k(self, k):

self.k = k

def X\_new(self,X):

examples = X.shape[0]

X\_multiply = np.prod(X, axis=1) # x1\*x2

X\_subtract = np.c\_[X[:,0],-X[:,1]] # x1 -x2

X\_new = np.c\_[np.ones(examples), X, np.square(X), X\_multiply, np.abs(np.sum(X\_subtract,axis=1)), np.abs(np.sum(X,axis=1))]

return X\_new[:,:(self.k + 1)]

def calc\_error(self, X,Y):

examples = X.shape[0]

predict = np.sign(self.predict(X))

num\_error = np.sum(np.not\_equal(predict, np.sign(Y)))

error = float(num\_error)/float(examples)

return error

def train\_regularization(self, X,Y): # training with regularization: (ZT\*Z + lambda\*I)^-1 \* ZT\*y

X\_new = self.X\_new(X)

xTx = np.dot(X\_new.T, X\_new)

lI = np.multiply(self.lamda, np.identity(xTx.shape[0])) # lambda\*I

inv\_X = np.linalg.inv(np.add(xTx, lI))

self.weights = np.dot(inv\_X, np.dot(X\_new.T, Y))

class Import\_Data:

def \_\_init\_\_(self, trainfile, testfile):

self.dim = 0

self.train\_X, self.train\_Y = self.load\_file(trainfile)

self.test\_X, self.test\_Y = self.load\_file(testfile)

def load\_file(self, filename):

# X = np.array([])

# Y = np.array([])

# with open(filename) as file:

# data = file.readlines()

# self.dim = len(data[0].split()) -1

# for line in data:

# XY = line.split()

# new\_XY = [float(k) for k in XY]

# X = np.append([X], [new\_XY[:-1]], axis = 0)

# Y = np.concatenate((X, [new\_XY[-1]]))

# return X, Y

X = np.array([])

Y = np.array([])

with open(filename) as f:

data = f.readlines()

examples = len(data)

self.dim = len(data[0].split()) - 1

for line in data:

XY = [float(x) for x in line.split()]

X = np.concatenate((X, XY[:-1])) #every X but last elt for Y

Y = np.concatenate((Y, [XY[-1]])) #last elt for Y

X = X.reshape((examples, self.dim))

return X, Y

def main():

data = Import\_Data("in.dta", "out.dta")

lamda = None

k = 7

NLT = NonlinearTrans(data.dim, lamda, k)

print("25 # raining, 10 # validation")

for k in np.arange(3,8):

NLT.change\_k(k)

NLT.train(data.train\_X[:25,:], data.train\_Y[:25])

print("k = %d" % k)

print("Evalidation: %f, Eout: %f" % (NLT.calc\_error(data.train\_X[25:,:], data.train\_Y[25:]), NLT.calc\_error(data.test\_X, data.test\_Y)))

print("10 # raining, 25 # validation")

for k in np.arange(3,8):

NLT.change\_k(k)

NLT.train(data.train\_X[25:,:], data.train\_Y[25:])

print("k = %d" % k)

print("Evalidation: %f, Eout: %f" % (NLT.calc\_error(data.train\_X[:25,:], data.train\_Y[:25]), NLT.calc\_error(data.test\_X, data.test\_Y)))

if \_\_name\_\_== "\_\_main\_\_":

main()

6. [d]

E(e1) = 0.5, E(e2) = 0.5, E(e3) = 0.3333

import numpy as np

N = 100000000

e1 = np.random.uniform(0,1,N)

e2 = np.random.uniform(0,1,N)

min\_e1\_e2 = np.minimum(e1, e2)

print(np.mean(e1))

print(np.mean(e2))

print(np.mean(min\_e1\_e2))

clear ; close all; clc

SVMvsPLA = 0;

TotalSptVecs = 0;

d = 2; % dimension of x vector

a = -1; b = 1; % sample from [a,b]^d

N = 100; % number of points to sample

m = 1000; % number of runs

K = 5000; % number of points in test sample

for i=1:m

[X, Y, fNormVec] = GenSepUniformPts(N, d, a, b);

xTest = [ones(K, 1), a + (b-a).\*rand(K, 2)];

ytest = sign(xTest\*fNormVec);

% PLA

w\_init = zeros(d+1, 1);

[wPLA, NumIter] = Perceptron(X, Y, w\_init);

gPLA = sign(xTest\*wPLA);

PLA\_misClassifiedProb = sum(ytest ~= gPLA)/K;

% SVM

XSVM = X(:,2:end);

[wSVM, bSVM, SptVecs, alpha, indicesSV] = svmLinear(XSVM,Y);

xSVMtest = xTest(:,2:end);

gSVM = sign(xSVMtest\*wSVM + bSVM);

SVM\_misClassifiedProb = sum(ytest ~= gSVM)/K;

SVMvsPLA = SVMvsPLA + (SVM\_misClassifiedProb < PLA\_misClassifiedProb);

TotalSptVecs = TotalSptVecs + SptVecs;

end

PercentSVMvsPLA = SVMvsPLA/m\*100;

AvgNumSptVecs = TotalSptVecs/m;