Homework 6

Jarod Klion

February 16th, 2022

1. A picture containing shape

   Description automatically generatedGisette Dataset

|  |  |  |
| --- | --- | --- |
| K | Train Error | Test Error |
| 10 | 0.13 | 0.142 |
| 30 | 0.128 | 0.141 |
| 100 | 0.128 | 0.14 |
| 300 | 0.127 | 0.14 |
| 500 | 0.127667 | 0.14 |

1. A picture containing chart

   Description automatically generatedDexter Dataset

|  |  |  |
| --- | --- | --- |
| K | Train Error | Test Error |
| 10 | 0.243333 | 0.25 |
| 30 | 0.21 | 0.223333 |
| 100 | 0.21 | 0.213333 |
| 300 | 0.193333 | 0.203333 |
| 500 | 0.183333 | 0.2 |

1. A picture containing icon

   Description automatically generatedMadelon dataset

|  |  |  |
| --- | --- | --- |
| K | Train Error | Test Error |
| 10 | 0.402 | 0.403333 |
| 30 | 0.3995 | 0.405 |
| 100 | 0.398 | 0.403333 |
| 300 | 0.395 | 0.398333 |
| 500 | 0.395 | 0.4 |

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

# # Problem 4.a: Gisette dataset

# In[309]:

# Define the FSA class with usual functions

class FSAClassifier:

def \_\_init\_\_(self, s = 0.0001, mu = 100, eta = 0.1, iterations = 300):

#initialize parameters

self.iterations = iterations

self.s = s

self.mu = mu

self.eta = eta/5000

self.desired\_features = [10, 30, 100, 300, 500] #number of feature selections wanted

self.train\_errs = []

self.test\_errs = []

self.k30\_train\_loss = []

def sigmoid(self, z):

'''

Sigmoid function

Parameters:

x : int, float or numpy array

Returns:

the sigmoid function applied to each element in the array

'''

return 1 / (1 + np.exp(-z))

def log\_likelihood(self, x, y):

'''

Log likelihood of penalized logistic regression FSA for classification

Parameters:

x : regressor matrix, X

beta: sparsity constraints (array)

y : labels {0, 1}

'''

z = np.dot(x, self.betas)

return (1 / x.shape[0])\*sum(np.log(1 + np.exp(-y \* z)))

def logistic\_loss(self, x, y):

z = np.dot(x, self.betas)

return (1 / x.shape[0]) \* sum(np.log(1 + np.exp(-(y \* z)))) + self.s \* np.linalg.norm(self.betas)

def fit(self, X, y, X\_test, y\_test):

'''

Fit data to the FSA model

Parameters:

X\_ : 2D Regressor matrix (num\_obs, num\_feats)

y\_ : 1D array of labels {0, 1}

Returns:

training loss, training errors, test errors

'''

num\_obs = X.shape[0]

num\_feats = X.shape[1]

#start with b0 = 0

self.betas = np.zeros(num\_feats)

#loop through each desired number of features

for k in self.desired\_features:

#loop for num of iterations

for j in range(self.iterations):

z = np.dot(X, self.betas)

y\_pred = self.sigmoid(z)

#calculate gradient

gradient = np.dot(X.T, y - y\_pred)

#update betas

self.betas = self.betas + self.eta \* gradient

#do feature selection

Mi = k + (num\_feats - k) \* int(max(0, (self.iterations - 2 \* j)/(2 \* j \* self.mu + self.iterations)))

sorted\_betas = np.argsort(np.abs(self.betas)) #argsort to get sliced lists

sorted\_betas = sorted\_betas[-Mi:]

#get the last Mi betas

self.betas = self.betas[sorted\_betas]

X = X[:, sorted\_betas]

X\_test = X\_test[:, sorted\_betas]

#Calculate training loss for k=30 specifically

if k == 30:

self.k30\_train\_loss.append(self.logistic\_loss(X, y))

#Capture the iteration loop's last value

if (j == self.iterations - 1):

self.train\_errs.append(1 - self.score(X,y))

self.test\_errs.append(1 - self.score(X\_test, y\_test))

def predict\_proba(self, X):

z = np.dot(X, self.betas)

probabilities = self.sigmoid(z)

return probabilities

def predict(self, X, threshold=0.5):

# Thresholding probability to predict binary values

binary\_predictions = np.array(list(map(lambda x: 1 if x > threshold else 0, self.predict\_proba(X))))

return binary\_predictions

def score(self, X, y\_true):

#classification accuracy

y\_pred = self.predict(X)

acc = accuracy\_score(y\_true, y\_pred)

return acc

# In[304]:

gis\_train = pd.read\_csv("../datasets/Gisette/gisette\_train.data", sep = ' ', header=None).dropna(axis=1)

gis\_train\_labels = np.where(np.ravel(pd.read\_csv("../datasets/Gisette/gisette\_train.labels", sep = ' ', header=None).values) == -1, 0, 1)

gis\_test = pd.read\_csv("../datasets/Gisette/gisette\_valid.data", sep = ' ', header=None).dropna(axis=1)

gis\_test\_labels = np.where(np.ravel(pd.read\_csv("../datasets/Gisette/gisette\_valid.labels", sep = ' ', header=None).values) == -1, 0, 1)

# In[305]:

#Normalize training data in gisette to have mean 0 and standard deviation 1

sc\_gis = StandardScaler()

sc\_gis.fit(gis\_train)

gis\_train\_norm = sc\_gis.transform(gis\_train)

#apply the same transformation to testing data

gis\_test\_norm = sc\_gis.transform(gis\_test)

# In[310]:

gis\_model = FSAClassifier()

gis\_model.fit(gis\_train\_norm, gis\_train\_labels, gis\_test\_norm, gis\_test\_labels)

# In[349]:

plt.figure(figsize=(13,4))

plt.subplot(121)

plt.plot(range(300), gis\_model.k30\_train\_loss)

plt.xlabel("Iteration number")

plt.ylabel("Training loss")

plt.title("Training Loss vs Iteration Number\nGisette Dataset")

plt.subplot(122)

plt.plot(gis\_model.desired\_features, gis\_model.train\_errs, '-o', label = "training error")

plt.plot(gis\_model.desired\_features, gis\_model.test\_errs, '-o', label = "test error")

plt.xlabel("Number of features")

plt.ylabel("Misclassification error")

plt.title("Misclassification Error vs Number of Features\nGisette Dataset")

plt.legend()

plt.savefig("Gisette.png")

plt.show()

# In[354]:

gis\_info = {"K": gis\_model.desired\_features,

"Train Error": gis\_model.train\_errs,

"Test Error": gis\_model.test\_errs,

}

gis\_df = pd.DataFrame(gis\_info)

gis\_df.sort\_values(['K'], ascending=True)

# # Problem 4.b: Dexter dataset

# In[314]:

#Read in the data for dexter dataset

dex\_train = pd.read\_csv("../datasets/dexter/dexter\_train.csv", header=None).dropna(axis=1)

dex\_train\_labels = np.where(np.ravel(pd.read\_csv("../datasets/dexter/dexter\_train.labels", sep = ' ', header=None).values) == -1, 0, 1)

dex\_test = pd.read\_csv("../datasets/dexter/dexter\_valid.csv", header=None).dropna(axis=1)

dex\_test\_labels = np.where(np.ravel(pd.read\_csv("../datasets/dexter/dexter\_valid.labels", sep = ' ', header=None).values) == -1, 0, 1)

# In[315]:

#Normalize training data in dexter to have mean 0 and standard deviation 1

sc\_dex = StandardScaler()

sc\_dex.fit(dex\_train)

dex\_train\_norm = sc\_dex.transform(dex\_train)

#apply the same transformation to testing data

dex\_test\_norm = sc\_dex.transform(dex\_test)

# In[316]:

dex\_model = FSAClassifier()

dex\_model.fit(dex\_train\_norm, dex\_train\_labels, dex\_test\_norm, dex\_test\_labels)

# In[350]:

plt.figure(figsize=(13,4))

plt.subplot(121)

plt.plot(range(300), dex\_model.k30\_train\_loss)

plt.xlabel("Iteration number")

plt.ylabel("Training loss")

plt.title("Training Loss vs Iteration Number\nDexter Dataset")

plt.subplot(122)

plt.plot(dex\_model.desired\_features, dex\_model.train\_errs, '-o', label = "training error")

plt.plot(dex\_model.desired\_features, dex\_model.test\_errs, '-o', label = "test error")

plt.xlabel("Number of features")

plt.ylabel("Misclassification error")

plt.title("Misclassification Error vs Number of Features\nDexter Dataset")

plt.legend()

plt.savefig("Dexter.png")

plt.show()

# In[353]:

dex\_info = {"K": dex\_model.desired\_features,

"Train Error": dex\_model.train\_errs,

"Test Error": dex\_model.test\_errs,

}

dex\_df = pd.DataFrame(dex\_info)

dex\_df.sort\_values(['K'], ascending=True)

# # Problem 4.c: Madelon dataset

# In[330]:

mad\_train = pd.read\_csv("../datasets/madelon/madelon\_train.data", sep=' ', header=None).dropna(axis=1)

mad\_train\_labels = np.where(np.ravel(pd.read\_csv("../datasets/madelon/madelon\_train.labels", sep=' ', header=None).dropna(axis=1).values) == -1, 0, 1)

mad\_test = pd.read\_csv("../datasets/madelon/madelon\_valid.data", sep = ' ', header=None).dropna(axis=1)

mad\_test\_labels = np.where(np.ravel(pd.read\_csv("../datasets/madelon/madelon\_valid.labels", sep=' ', header=None).dropna(axis=1).values) == -1, 0, 1)

# In[332]:

#Normalize training data in madelon to have mean 0 and standard deviation 1

sc\_mad = StandardScaler()

sc\_mad.fit(mad\_train)

mad\_train\_norm = sc\_mad.transform(mad\_train)

#apply the same transformation to testing data

mad\_test\_norm = sc\_mad.transform(mad\_test)

# In[339]:

mad\_model = FSAClassifier(eta = 0.01)

mad\_model.fit(mad\_train\_norm, mad\_train\_labels, mad\_test\_norm, mad\_test\_labels)

# In[351]:

plt.figure(figsize=(13,4))

plt.subplot(121)

plt.plot(range(300), mad\_model.k30\_train\_loss)

plt.xlabel("Iteration number")

plt.ylabel("Training loss")

plt.title("Training Loss vs Iteration Number\nMadelon Dataset")

plt.subplot(122)

plt.plot(mad\_model.desired\_features, mad\_model.train\_errs, '-o', label = "training error")

plt.plot(mad\_model.desired\_features, mad\_model.test\_errs, '-o', label = "test error")

plt.xlabel("Number of features")

plt.ylabel("Misclassification error")

plt.title("Misclassification Error vs Number of Features\nMadelon Dataset")

plt.legend()

plt.savefig("Madelon.png")

plt.show()

# In[352]:

mad\_info = {"K": mad\_model.desired\_features,

"Train Error": mad\_model.train\_errs,

"Test Error": mad\_model.test\_errs,

}

mad\_df = pd.DataFrame(mad\_info)

mad\_df.sort\_values(['K'], ascending=True)

# In[ ]: