Homework #2A

Spring 2020, CSE 446/546: Machine Learning
Prof. Kevin Jamieson and Prof. Jamie Morgenstern
Due: 5/12/20 11:59 PM

1 Conceptual Questions

A0

a. No, it doesn't. Probably after removing 'number of bathrooms', the weights of some other correlated features increase and maybe they can have a similar prediction as before.

b. An L1 norm penalty doesn't result in more weights to be 0. Because from the figures given in lectures, the diamond (Lasso) has corners on the axes, unlike the disk, and whenever the elliptical region hits such point, one of the features completely vanishes!

c. Upside: fewer weights are 0. Downside: Probably there is no intersection between elliptical region and this regularizer.

d. TRUE

e. SGD is a lot faster since by analyzing only one example at a time and following its slope we can reach a point that is very close to the actual minimum.

f. advantage: faster when we use big datasets.
disadvantage: the convergence path of SGD is noiser.

2 Convexity and Norms

A.1

a. Firstly we show f(x) satisfies triangle inequality:
$f(x+y) = \sum_1^n |x_i + y_i| \leq \sum_1^n |x_i| + |y_i| \leq f(x) + f(y)$
And then:
f(x)≥0 and when all $x_i = 0$, f(x)=0
$f(ax) = \sum_1^n a|x_i| = a\sum_1^n |x_i|$
so f(x) is a norm here.

b. Here we set two points respectively are $x(0,1)$ and $(y(4,3)$ and so
$g(x+y) = 16$ and $g(x) + g(y) = 8 + 4\sqrt{3}$ and we can find that
$g(x+y) < g(x) + g(y)$ so g(x) is not a norm.

A.2
I is not convex since b and c don't satisfy this. II is convex. III is not convex since a and d dpn't satisfy this. A.3

a. It is convex.

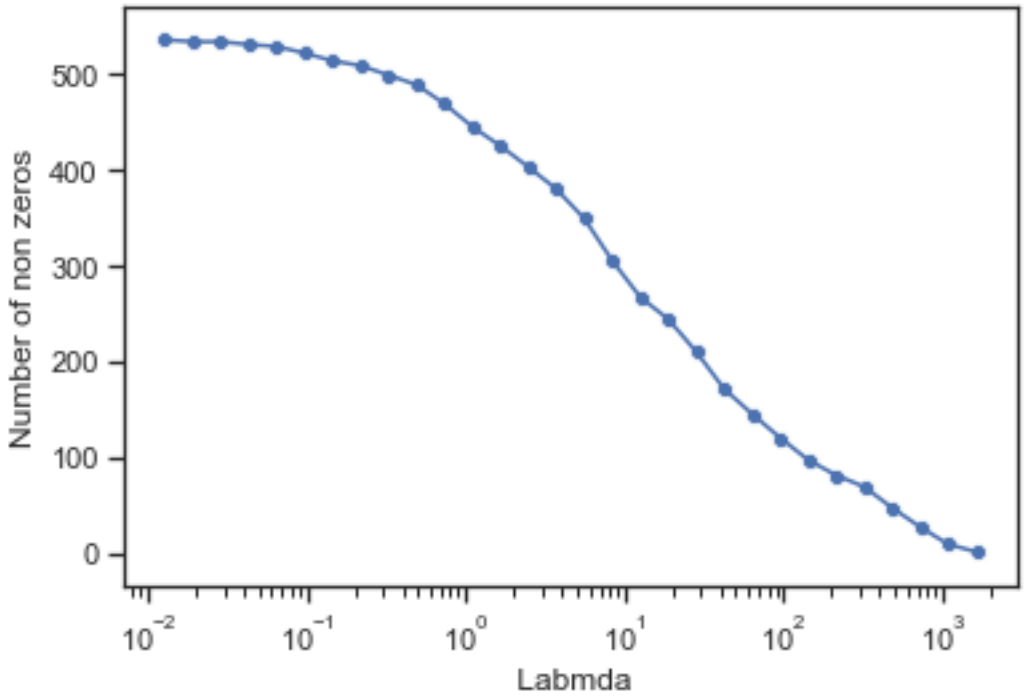
b. It is not convex no matter on the interval (a, b) or (b, c).

c. It is not convex on (a, c).

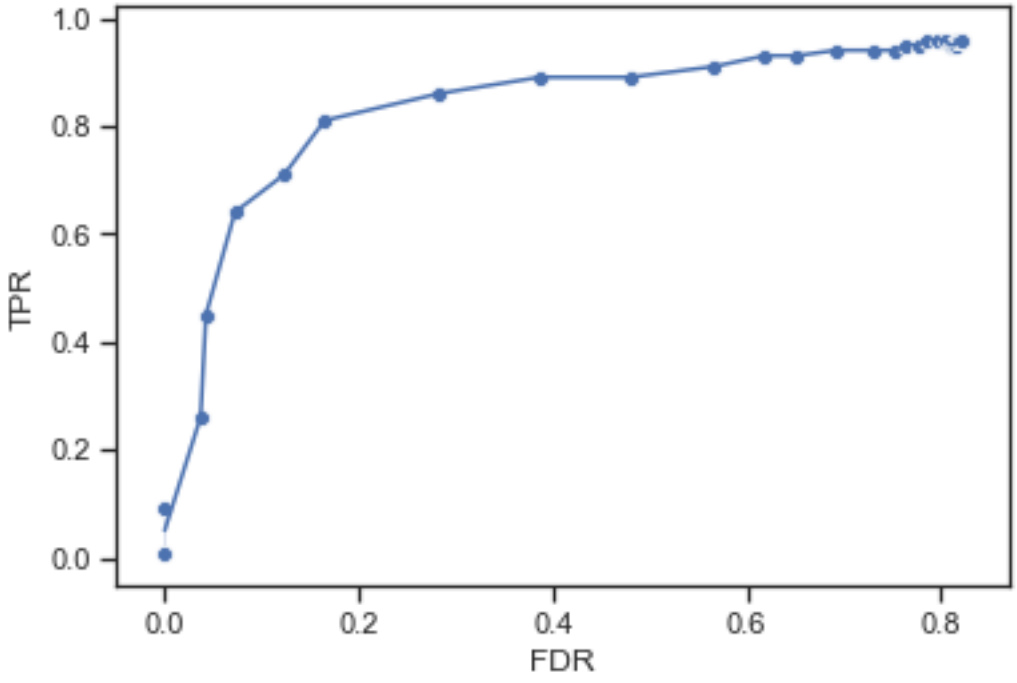
d. It is convex.

3 Lasso

A.4



a.



b.

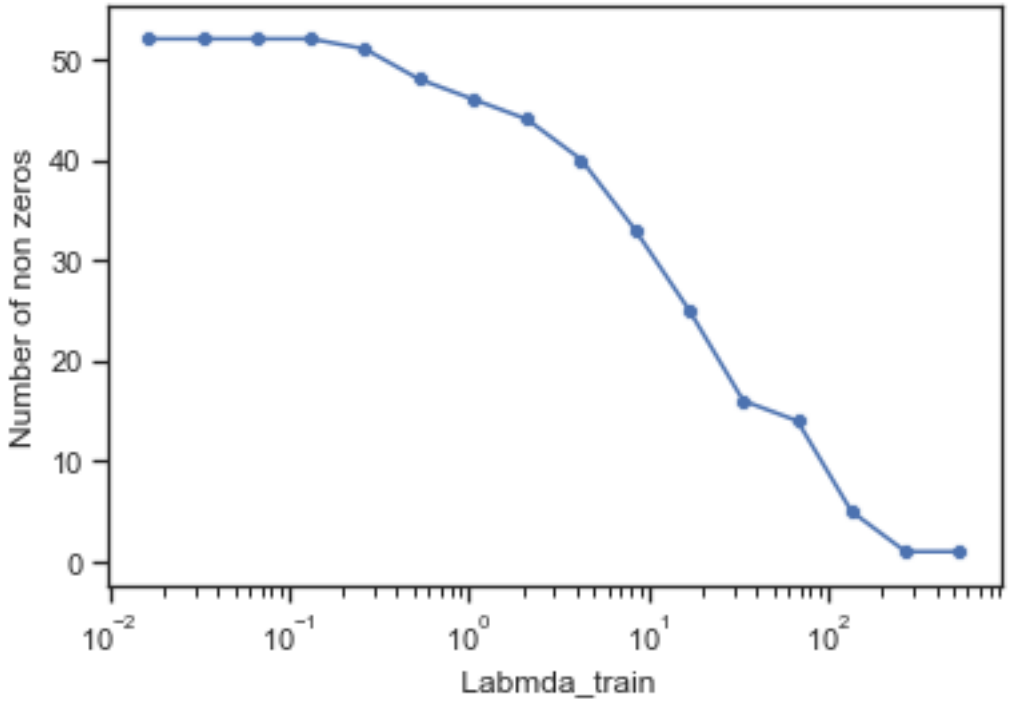
c. In (a), as lambda increases, the number of nonzero features decreases.

Listing 1: A4code

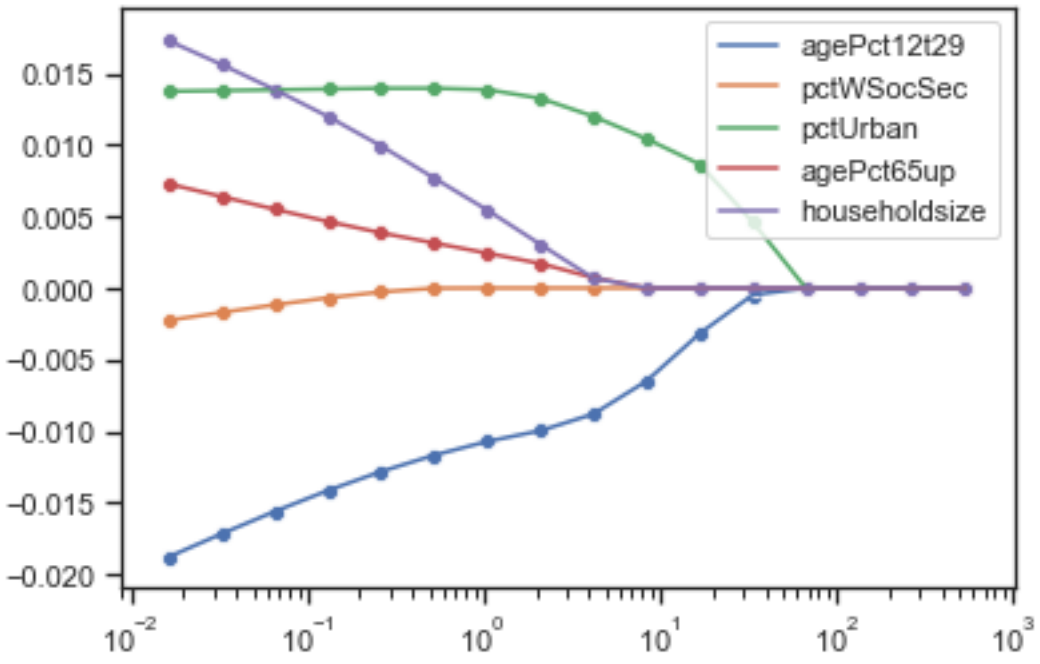
```
import numpy as np
import scipy
import scipy.linalg as la
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import seaborn as sns
from mnist import MNIST
import pickle
import os

sns.set()
sns.set_style("ticks")
np.random.seed(0)

def synData(n=500, d=1000, k =100, sig=1):
    eps = np.random.randn(n)
    x = np.random.randn(n, d)
    w = np.arange(1,d+1)/k
    w[ w > 1] = 0
    Y = (x).dot(w.T) + eps
    return(x, Y, w)


def lambdaMax(X, Y):
        diff = Y - np.mean(Y)
        lmax = 2*np.max(abs(diff.dot(X)))
        return(lmax)


def coord_desc_lasso(X, Y, Lambda, w=None, delta = 0.001):
        diff = delta + 1
        d = X.shape[1]
        if(w is None):
                w = np.zeros(d)
        else:
                w = w.copy()

        aks = 2*np.sum(np.square(X), axis=0)

        while(delta < diff):
                b = 0
                w_original = w.copy()

                for k in np.arange(d):
                        ak = aks[k]
                        xk = X[:,k]
                        jnotk = X.dot(w.T) - w[k]*xk
                        ck = 2*xk.dot(Y - (b + jnotk) )

                        if(ck < -Lambda):
                                w[k] = (ck + Lambda)/ak
```

```
                elif(-Lambda <= ck <= Lambda):
                        w[k] = 0
                else:
                        w[k] = (ck - Lambda)/ak

        diff = np.max(np.abs(w - w_original))


    return(w, b)

n=500;d=1000;k=100
X, Y, realW = synData(n=n, d=d, k=k)
lmax= lambdaMax(X, Y)
print("Lambda max is:{}".format(lmax))

Lambda=lmax
nonZero_list = []
FDR = []
TPR = []
Lambda_list = []
w = np.zeros(d)
while Lambda > 0.01:
    w,b = coord_desc_lasso(X, Y, Lambda, w=w, delta=0.01)
    nzero = (w>0).sum()
    nonZero_list.append( nzero )
    TPR.append( (w[0:k] > 0.0).sum()/k  )
    FDR.append( (w[k:] > 0.0).sum()/nzero  )
    Lambda_list.append(Lambda)
    Lambda = Lambda/1.5



#a
fig, ax = plt.subplots()
sns.lineplot(Lambda_list, nonZero_list)
sns.scatterplot(Lambda_list, nonZero_list)
plt.xscale('log')
ax.set_xlabel(" Labmda")
ax.set_ylabel("Number of non zeros")

#b
fig, ax = plt.subplots()
sns.lineplot(x=FDR, y=TPR, ax=ax)
sns.scatterplot(x=FDR, y=TPR, ax=ax)
ax.set_xlabel("FDR")
ax.set_ylabel("TPR")
```

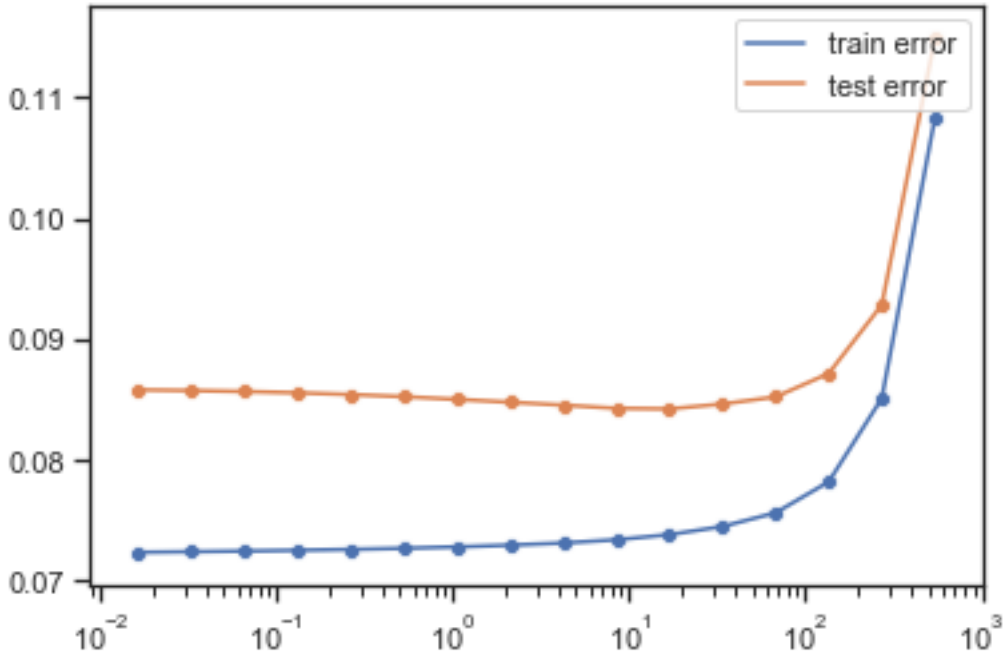
A.5


Number of non zeros
50
40
30
20
10
0
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
Labmda_train

a.


agePct12t29
pctWSocSec
pctUrban
agePct65up
householdsize
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
−0.020
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}

b.



c.

d. PctIlleg has the largest coefficient and PctKids2Par has the smallest one.

e. Safe places have more people above 65 because those are safe and old people themselves stay away from high crime rate regions. So moving old people to high crime rate places can't solve any thing.

Listing 2: A5code

```
import numpy as np
import scipy
import scipy.linalg as la
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import seaborn as sns
from mnist import MNIST
import pickle
import os

sns.set()
sns.set_style("ticks")
np.random.seed(0)

def lambdaMax(X, Y):
        diff = Y - np.mean(Y)
        lmax = 2*np.max(abs(diff.dot(X)))
        return(lmax)


def coord_desc_lasso(X, Y, Lambda, w=None, delta = 0.001):
        diff = delta + 1
        d = X.shape[1]
        if(w is None):
                w = np.zeros(d)
        else:
```

```
            w = w.copy()

        aks = 2*np.sum(np.square(X), axis=0)

        while(delta < diff):
            b = 0
            w_original = w.copy()

            for k in np.arange(d):
                ak = aks[k]
                xk = X[:,k]
                jnotk = X.dot(w.T) - w[k]*xk
                ck = 2*xk.dot(Y - (b + jnotk) )

                if(ck < -Lambda):
                    w[k] = (ck + Lambda)/ak
                elif(-Lambda <= ck <= Lambda):
                    w[k] = 0
                else:
                    w[k] = (ck - Lambda)/ak

            diff = np.max(np.abs(w - w_original))


        return(w, b)

def MSE(Y, X, w, b):
        pred = X.dot(w.T) +b
        se = ((Y - pred) ** 2).mean()
        return(se)

import pandas as pd
df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")
train=df_train.drop("ViolentCrimesPerPop", axis = 1)
X_train = df_train.drop("ViolentCrimesPerPop", axis = 1).values
Y_train = df_train["ViolentCrimesPerPop"].values
X_test = df_test.drop("ViolentCrimesPerPop", axis = 1).values
Y_test = df_test["ViolentCrimesPerPop"].values


lmax_train=lambdaMax(X_train, Y_train)
Lambda_train=lmax_train
print("Lambda_train_max_is:{}".format(lmax_train))
nonZero_list_train = []
Lambda_list_train = []
d_train = X_train.shape[1]
w_train = np.zeros(d_train)
w_train_list=[]
se_train_list=[]
se_test_list=[]
while Lambda_train >= 0.01:
    w_train,b_train = coord_desc_lasso(X_train, Y_train, Lambda_train, w=w_train, delta=0.0
    nzero_train = (w_train>0).sum()
    nonZero_list_train.append( nzero_train )
```

```
    Lambda_list_train.append(Lambda_train)
    w_train_list.append(w_train)
    se_train=MSE(Y_train, X_train, w_train, b_train)
    se_test=MSE(Y_test, X_test, w_train, b_train)
    se_train_list.append(se_train)
    se_test_list.append(se_test)
    Lambda_train = Lambda_train/2.0

#a
fig, ax = plt.subplots()
sns.lineplot(Lambda_list_train, nonZero_list_train)
sns.scatterplot(Lambda_list_train, nonZero_list_train)
plt.xscale('log')
ax.set_xlabel("Labmda_train")
ax.set_ylabel("Number of non zeros")

#b
a=list(train.columns).index("agePct12t29")
b=list(train.columns).index("pctWSocSec")
c=list(train.columns).index("pctUrban")
d=list(train.columns).index("agePct65up")
e=list(train.columns).index("householdsize")
lsta=[item[a] for item in w_train_list]
lstb=[item[b] for item in w_train_list]
lstc=[item[c] for item in w_train_list]
lstd=[item[d] for item in w_train_list]
lste=[item[e] for item in w_train_list]

fig, ax = plt.subplots()
sns.lineplot(Lambda_list_train, lsta)
sns.scatterplot(Lambda_list_train, lsta)
plt.xscale('log')
sns.lineplot(Lambda_list_train, lstb)
sns.scatterplot(Lambda_list_train, lstb)
plt.xscale('log')
sns.lineplot(Lambda_list_train, lstc)
sns.scatterplot(Lambda_list_train, lstc)
plt.xscale('log')
sns.lineplot(Lambda_list_train, lstd)
sns.scatterplot(Lambda_list_train, lstd)
plt.xscale('log')
sns.lineplot(Lambda_list_train, lste)
sns.scatterplot(Lambda_list_train, lste)
plt.xscale('log')
plt.show()
plt.legend(["agePct12t29","pctWSocSec","pctUrban","agePct65up","householdsize"],loc="upper r

#c
fig, ax = plt.subplots()
sns.lineplot(Lambda_list_train, se_train_list)
sns.scatterplot(Lambda_list_train, se_train_list)
plt.xscale('log')
sns.lineplot(Lambda_list_train, se_test_list)
sns.scatterplot(Lambda_list_train, se_test_list)
plt.xscale('log')
```

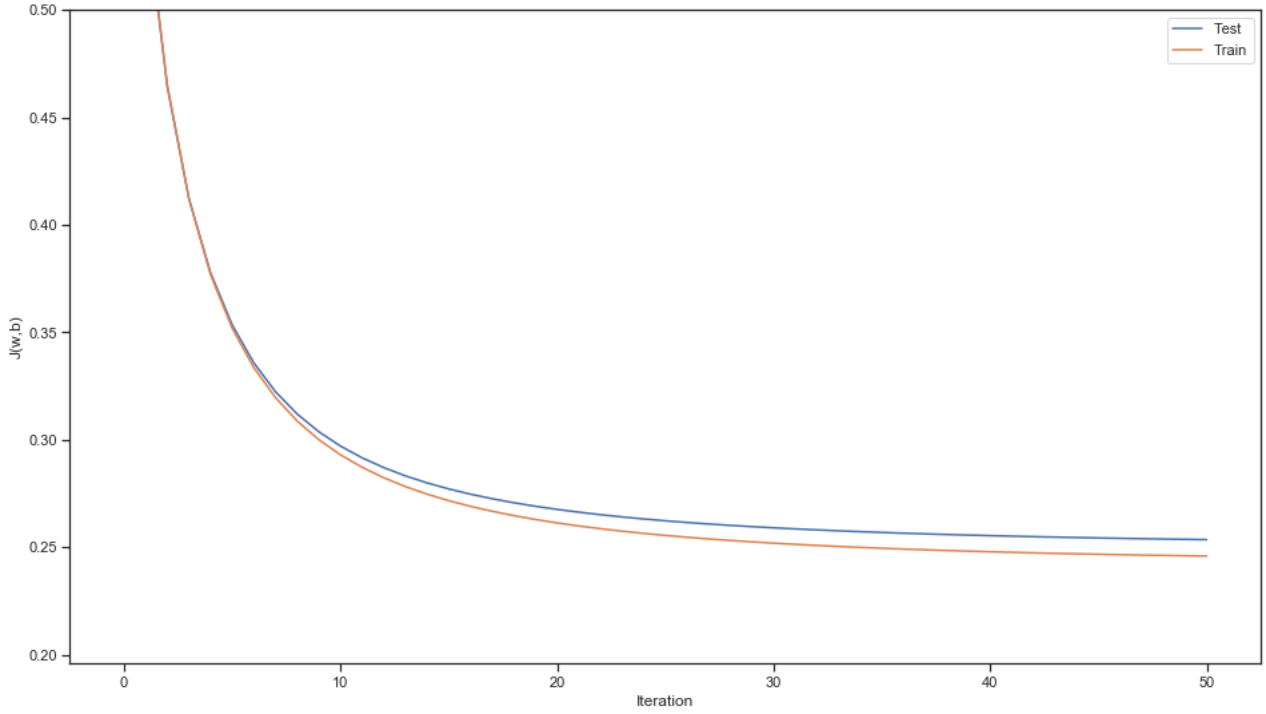
```
plt.show()
plt.legend(["train error","test error"],loc="upper right")

#d
w_4,b_4=coord_desc_lasso(X_train, Y_train, 30, w=w_train, delta=0.001)
largest_loc=list(w_4).index(np.max(w_4))
least_loc=list(w_4).index(np.min(w_4))
print(list(train.columns)[largest_loc])
print(list(train.columns)[least_loc])
```

4 Logistic Regression

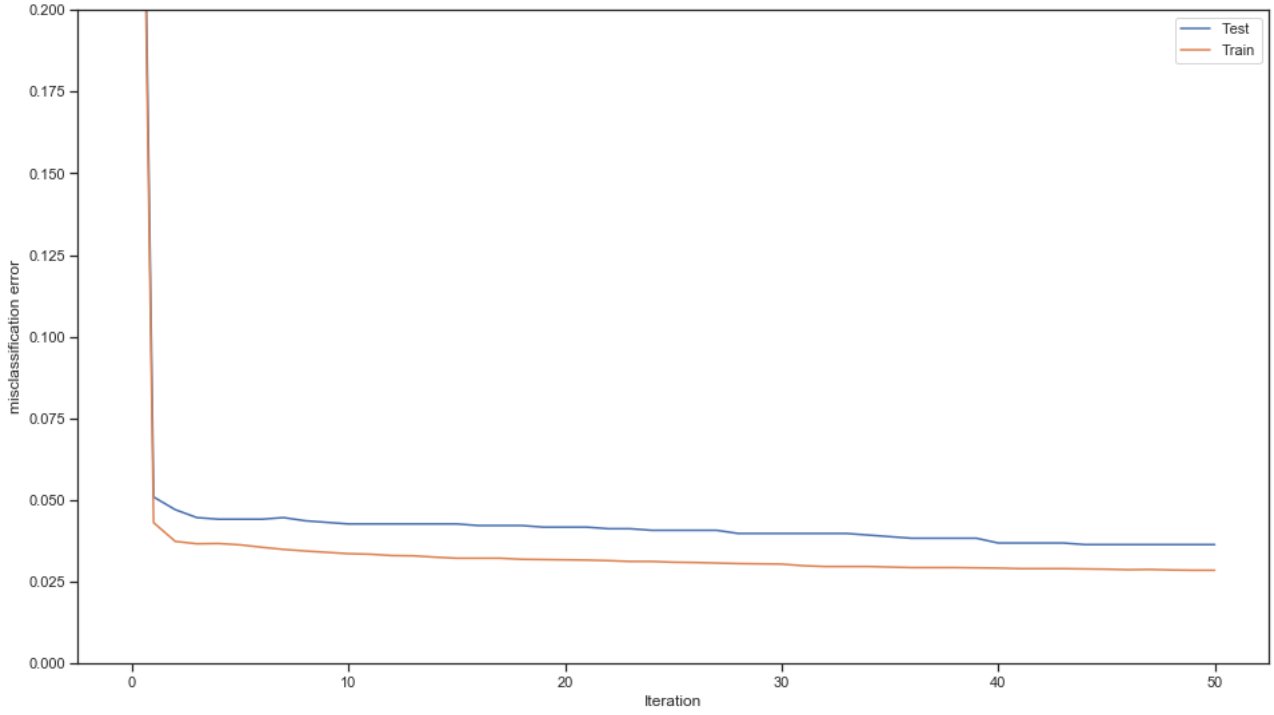
A.6

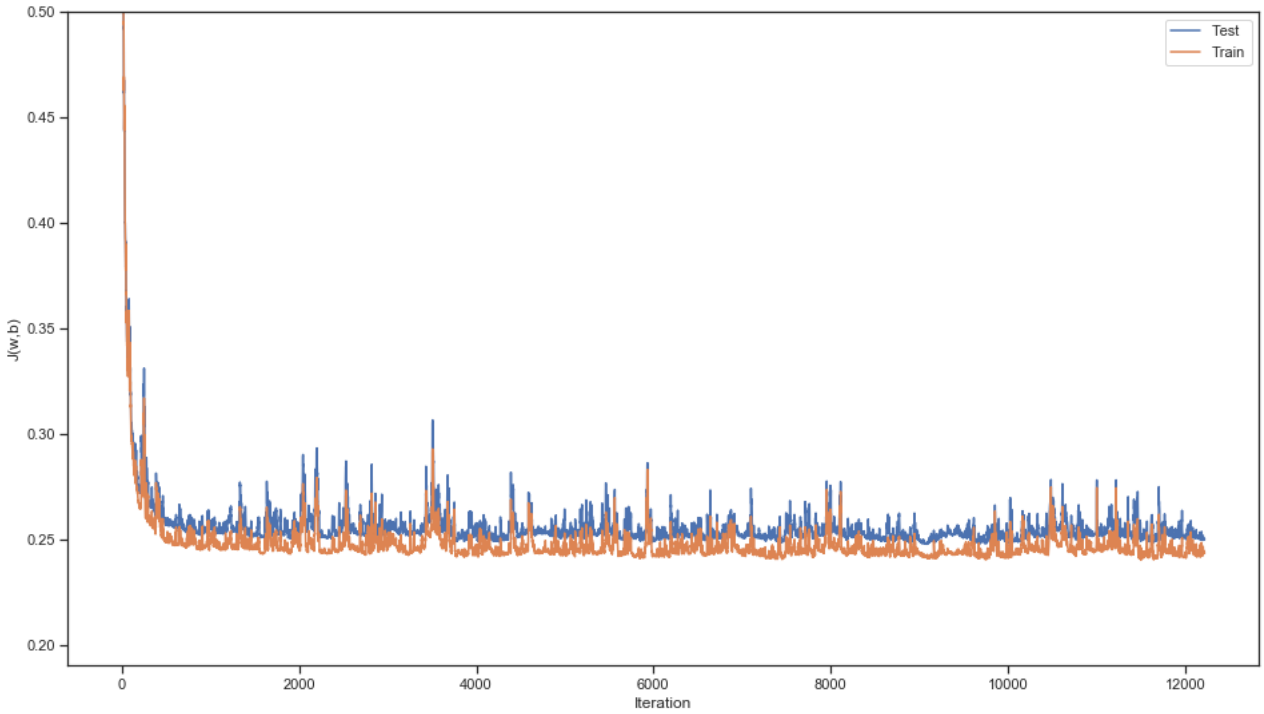
a. Taking the first derivatives with respect to w and then b.

$$
\begin{aligned}
\nabla_w J(w,b) &= \nabla_w \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^T w))) + \lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla_w \log(1+\exp(-y_i(b+x_i^T w))) + \nabla_w \lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i x_i \exp(-y_i(b+x_i^T w))}{1+\exp(-y_i(b+x_i^T w))} + \nabla_w \lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i x_i}{1+\exp(-y_i(b+x_i^T w))}\left(\exp(-y_i(b+x_i^T w))\right) + \nabla_w \lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}-x_i y_i \mu_i(w,b)\left((1-\mu_i(w,b))/(\mu_i(w,b))\right) + \nabla_w \lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}-x_i y_i \left(1-\mu_i(w,b)\right) + 2\lambda w \\
\nabla_b J(w,b) &= \nabla_b \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^T w))) + \lambda||w||_2^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\nabla_b \log(1+\exp(-y_i(b+x_i^T w))) + 0 \\
&= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i \exp(-y_i(b+x_i^T w))}{1+\exp(-y_i(b+x_i^T w))} \\
&= \frac{1}{n}\sum_{i=1}^{n}-y_i\left(1-\mu_i(w,b)\right)
\end{aligned}
$$

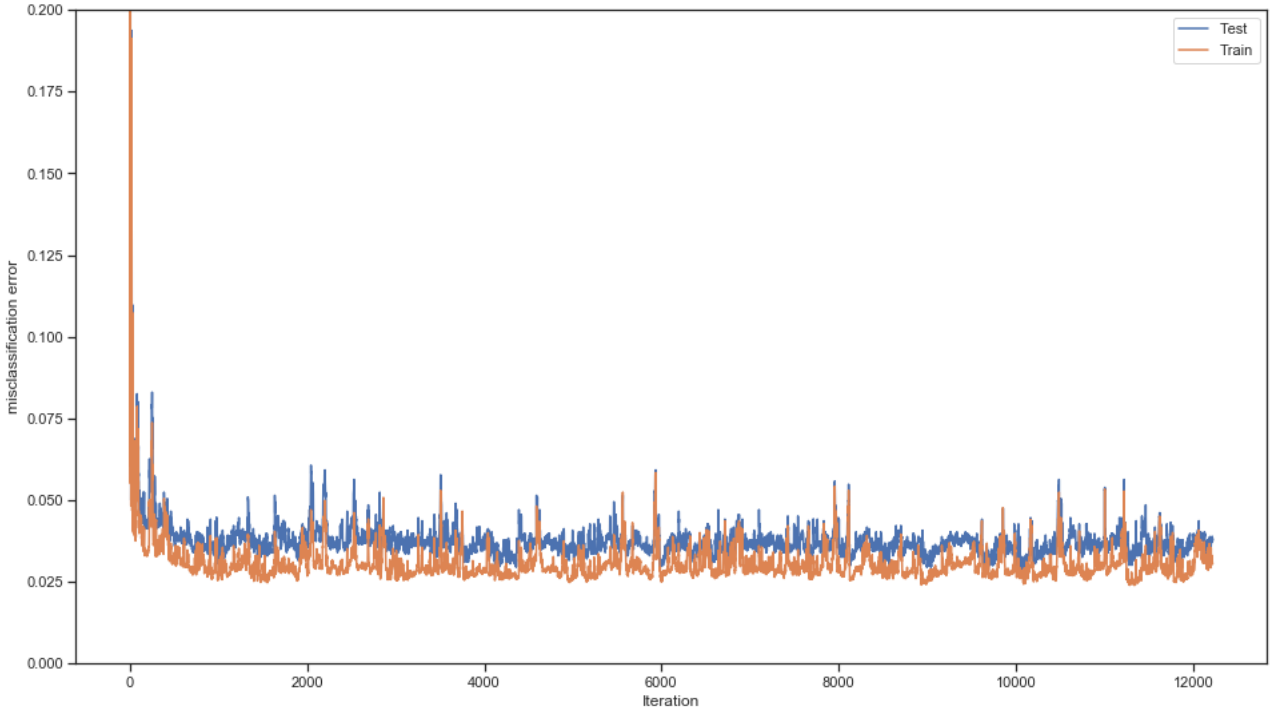

Test
Train
J(w,b)
Iteration
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0
10
20
30
40
50

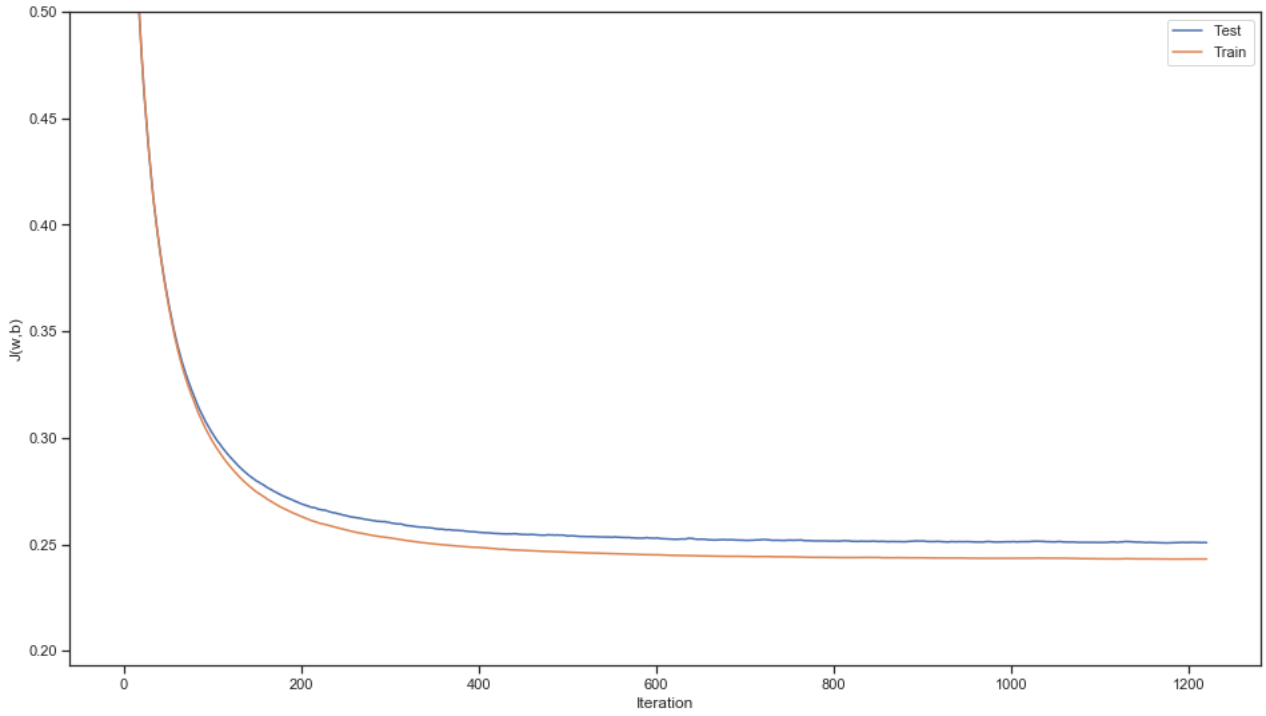
b.


Test
Train
misclassification error
Iteration
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
10
20
30
40
50

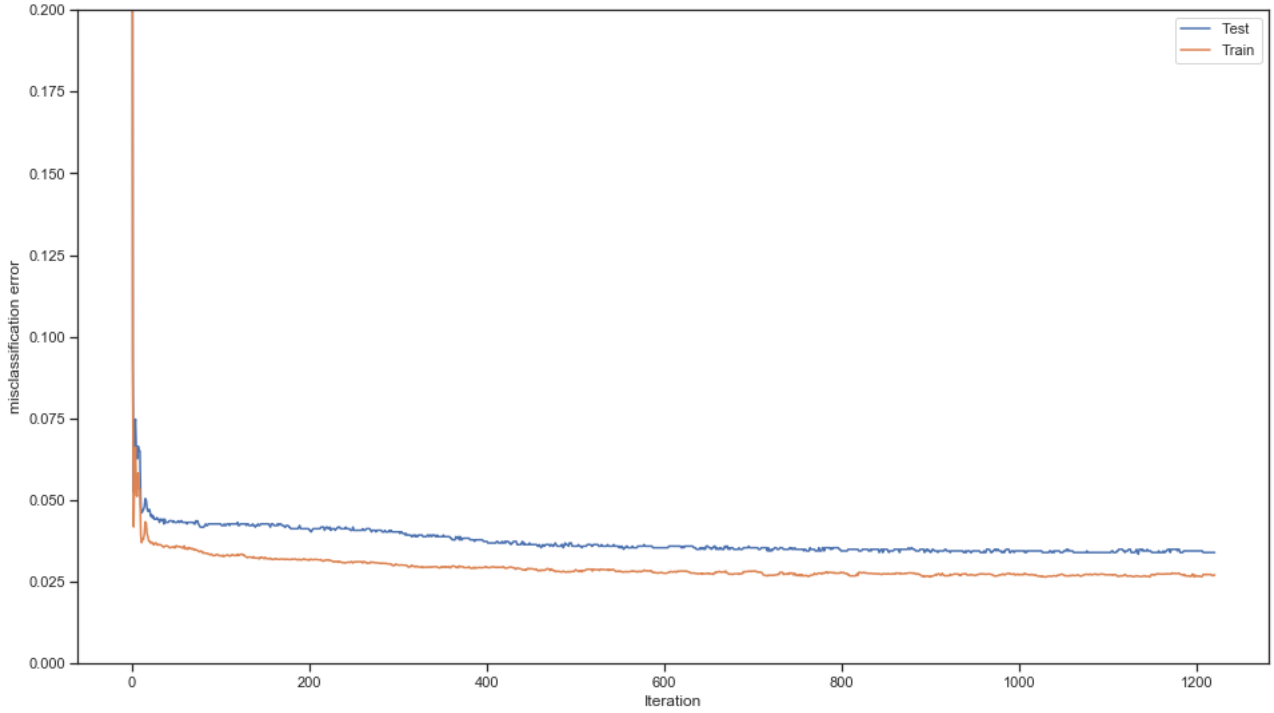

Test
Train
J(w,b)
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0
2000
4000
6000
8000
10000
12000
Iteration

c.


Test
Train
misclassification error
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2000
4000
6000
8000
10000
12000
Iteration



d.



Listing 3: A6code

```
import numpy as np
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
import seaborn as sns
from mnist import MNIST
sns.set()
```

```
sns.set_style("ticks")
np.random.seed(0)
Lambda=0.1

def load_dataset():
    mndata = MNIST('./data/')
    mndata.gz=True
    X_train, labels_train = map(np.array, mndata.load_training())
    X_test, labels_test = map(np.array, mndata.load_testing())
    X_train = X_train/255.0
    X_test = X_test/255.0

    labels_train = labels_train.astype(np.int16)
    labels_test = labels_test.astype(np.int16)
    mask = ( (labels_test == 7) | (labels_test == 2) )
    labels_test = labels_test[mask]
    X_test = X_test[mask,:]
    labels_test[labels_test == 7]  = 1
    labels_test[labels_test == 2]  = -1

    mask = ( (labels_train == 7) | (labels_train == 2) )
    labels_train = labels_train[mask]
    X_train = X_train[mask,:]
    labels_train[labels_train == 7]  = 1
    labels_train[labels_train == 2]  = -1

    return X_train,labels_train, X_test, labels_test




def grad_desc(X, Y, w, b, eta):
    u = 1.0/(1.0+np.exp(-Y*(b + X.dot(w))))
    dtob=(-Y * (1-u)).mean()
    b = b -eta*dtob
    xy = np.multiply(X.T, Y)
    dtow = (- xy * (1-u)).mean(axis=1) + 2 * Lambda * w
    w = w-eta*dtow
    return(w,b)

def err(X, Y, w, b):
    inside = np.log( 1.0 + np.exp( -Y * (b + X.dot(w)) ) )
    j = inside.mean() + Lambda * np.linalg.norm(w,ord=2)
    pred = b + X.dot(w)
    pred[pred < 0] = -1
    pred[pred >= 0 ] = 1
    correct = np.sum(pred == Y)
    error = 1.0 - float(correct) / float(X.shape[0])
    return(j, error)

def makePlots(iters, test_j, train_j, test_e, train_e, name):
        fig, ax = plt.subplots(figsize=(16,9))
        sns.lineplot(x=iters, y=test_j, ax=ax, label="Test")
        sns.lineplot(x=iters, y=train_j, ax=ax, label="Train")
        ax.set_xlabel("Iteration")
```

```
    ax.set_ylabel("J(w,b)")
    plt.ylim(min(train_j)-.05, .5)
    plt.legend()

    fig, ax = plt.subplots(figsize=(16,9))
    sns.lineplot(x=iters, y=test_e, ax=ax, label="Test")
    sns.lineplot(x=iters, y=train_e, ax=ax, label="Train")
    ax.set_xlabel("Iteration")
    ax.set_ylabel("misclassification error")
    plt.ylim(0,.2)
    plt.legend()



def run(X_train, Y_train, X_test, Y_test, name, eta, itersize, batch=0):
    d=X_train.shape[1]
    w = np.zeros(d)
    b = 0
    iters = []; test_j=[]; train_j=[]; test_e = []; train_e = []
    j, error = err(X_train, Y_train, w, b)
    t_j, t_error = err(X_test, Y_test, w, b)
    test_j.append(t_j)
    train_j.append(j)
    test_e.append(t_error)
    train_e.append(error)
    iters.append(0)
    i = 1
    for k in range(0,itersize):
        n = X_train.shape[0]
        idx = np.random.permutation(n)
        X_train = X_train[idx]
        Y_train = Y_train[idx]
        split = n/batch
        Xs = np.array_split(X_train, split)
        Ys = np.array_split(Y_train, split)
        for X_split, Y_split in zip(Xs, Ys):
            w, b = grad_desc(X_split, Y_split, w, b, eta)
            j, error = err(X_train, Y_train, w, b)
            t_j, t_error = err(X_test, Y_test, w, b)
            test_j.append(t_j)
            train_j.append(j)
            test_e.append(t_error)
            train_e.append(error)
            iters.append(i)
            if(i % 100 == 0 or split == 1):
                print(j, error, i, k, X_split.shape)
            i += 1
    makePlots(iters, test_j, train_j, test_e, train_e, name)



X_train, Y_train, X_test, Y_test = load_dataset()
run( X_train, Y_train, X_test, Y_test, "A6b", 0.01, 50, batch=X_train.shape[0])
run( X_train, Y_train, X_test, Y_test, "A6c", 0.01, 1, batch=1 )
run( X_train, Y_train, X_test, Y_test, "A6d", 0.01, 10, batch=100 )
```