Homework #2 - A

Spring 2020, CSE 446/546: Machine Learning
Richy Yun
Due: 5/12/20 11:59 PM

Conceptual questions

A.0

- *[2 points]* Suppose that your estimated model for predicting house prices has a large positive weight on 'number of bathrooms'. Does it implies that if we remove the feature "number of bathrooms" and refit the model, the new predictions will be strictly worse than before? Why?

  No it does not. The high weight on 'number of bathrooms' could simply be due to the correlation of that feature with another (ex. square footage, number of bedrooms, etc.) such that removal of it may not change the fit or may even make it fit better.

- *[2 points]* Compared to L2 norm penalty, explain why a L1 norm penalty is more likely to result in a larger number of 0s in the weight vector or not?

  A L1 norm penalty is more likely to result in a larger number of 0s in the weight vector as it provides much more sparse estimates. As visualized in lecture, the point that minimizes loss during regularization is more likely to fall on an axis which will simply choose that index in the weight vector and set the others to 0.

- *[2 points]* In at most one sentence each, state one possible upside and one possible downside of using the following regularizer: $(\sum_i |w_i|^{0.5})$

  A possible upside is that very small weights (¡1) are amplified, thus having a strong likelihood of suppressing those features that may not be as relevant. A possible downside is that much larger weights are not punished as strongly, possibly leading to overfitting and defeating the purpose of regularization.

- *[1 points]* True or False: If the step-size for gradient descent is too large, it may not converge.

  True. If the step size is too large, you may skip over the minima and instead end up "bouncing around" the minima and diverging.

- *[2 points]* In your own words, describe why SGD works.

  Although SGD does not ensure a more accurate model every single iteration, the expected value of the descent for each data point (or batch of data) is the same as performing the descent on the entire data set.Therefore SGD will converge to the correct answer.

- *[2 points]* In at most one sentence each, state one possible advantage of SGD (stochastic gradient descent) over GD (gradient descent) and one possible disadvantage of SGD relative to GD.

  Each step of SGD is potentially faster and uses less memory, allowing for possibly quicker convergence. However, each iteration has jitter that needs to be corrected for, requiring more iterations than GD, meaning it would not be faster if each step is parallelized and not dependent on the size of the data.

Convexity and Norms

A.1

a. *[3 points]* The first two properties clearly hold: i) $f(x)$ is nonnegative as it is a sum of positive values unless the values are 0. ii) A summation is a linear transformation so we can pull out any constant that is multiplied to x, thus giving absolute scalability. The third condition can be proven:

$$\begin{aligned} a &\leq |a| \text{ (by definition)} \\ a + b &\leq |a| + |b| \\ (a+b)^2 &\leq (|a| + |b|)^2 \\ |a+b| &\leq |a| + |b| \end{aligned}$$

The condition holds true through the summation, and thus $f(x)$ is a norm.

b. *[2 points]* Considering two points in $n = 2$: $(0, 1)$ and $(1, 1)$, we have:

$$\begin{aligned} g(0,1) &= 1 \\ g(1,1) &= 1 \\ g(1,2) &= (1+\sqrt{2})^2 = 5.828 \\ g(1,2) &> g(0,1) + g(1,1) = 5 \end{aligned}$$

Thus the triangle inequality does not hold and $g(x)$ is not a norm.

A.2 *[3 points]* II is convex. I is not convex as the line between points b and c lies outside the set. III is not convex as the line between points a and d lies outside the set.

A.3 *[4 points]*

a. It is convex.

b. It is not convex. The line between points a and b lies below the function.

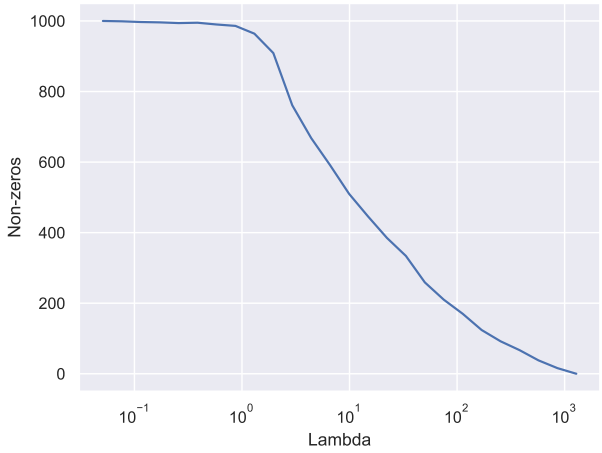
c. It is not convex. The line between points a and c lies below the function.

d. It is convex.

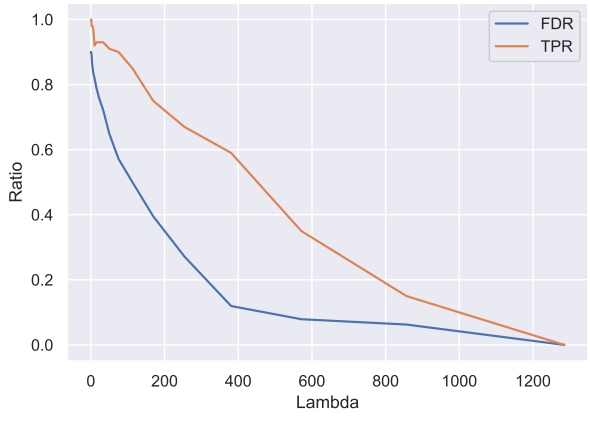
Lasso

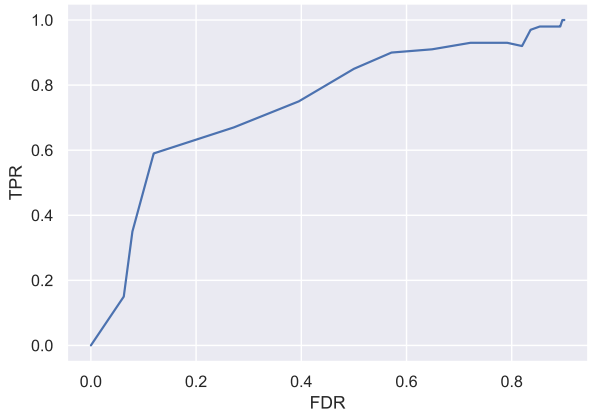
A.4

a. *[10 points]*



b. *[10 points]*





c. *[5 points]* As λ becomes smaller the regularization becomes weaker allowing for more features to be selected. As a result we see an increase in the number of non-zero weights as well as both FDR and TPR. TPR increases much faster than FDR until the "larger" weights are nearly all picked, showing the algorithm does a good job of picking the correct features.

Source Code

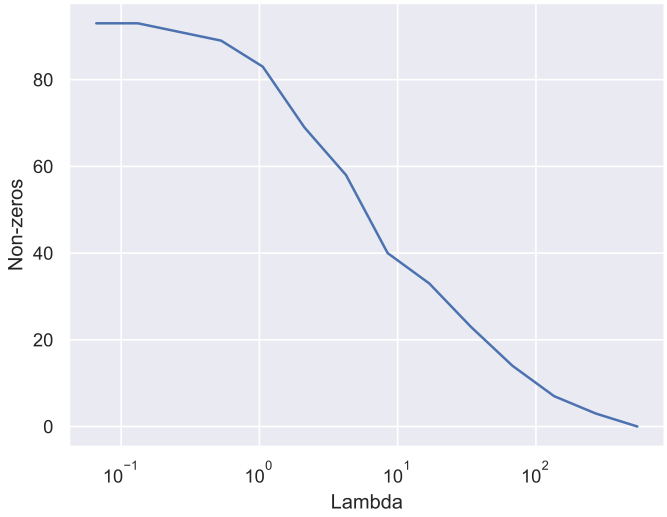
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

# to find the maximum lambda that gives all zero in w
def lambdamax(x, y, n):
    yres = y - np.sum(y) / n
    xres = np.matmul(x, yres)
    xsum = 2 * np.absolute(xres)
    return np.max(xsum)

# initialize variables
n = 500
d = 1000
k = 100
sigma = 1

x = np.random.normal(0, 1, (d, n))
eps = np.random.normal(0, sigma ** 2, n)
w = np.zeros(d)
w[0:k] = np.arange(1, k + 1) / k

# as defined in problem
y = np.matmul(np.transpose(w), x) + eps

# initialize variables
lam = lambdamax(x, y, n)
deltalim = 0.01
maxfeatures = 0.95*d
nonzero = 0
lambdas = []
nonzeros = []
FDR = []
TPR = []

# Keep checking smaller lambdas until at least 900 elements in w are nonzero
a = 2 * np.sum(np.square(x), axis=1)
while nonzero < maxfeatures:
    lambdas.append(lam)
    print('lambda', lam)
    w = np.zeros(d)

    # gradient descent
    maxdelta = float('inf')
    while maxdelta > deltalim:
```

```
        # a and b can be calculated outside of the for loop
        b = np.sum(y - np.matmul(w, x)) / n

        oldw = w.copy()

        for j in range(d):

            # calculate ck
            tempw = np.delete(w, j)
            tempx = np.delete(x, j, 0)
            ck = 2 * np.dot(np.transpose(x[j, :]), y - (b + np.matmul(tempw, tempx)))

            # determine wk
            if ck < -lam:
                w[j] = (ck + lam) / a[j]
            elif ck > lam:
                w[j] = (ck - lam) / a[j]
            else:
                w[j] = 0

        # find difference, set variables for conditions
        delta = np.absolute(oldw - w)
        maxdelta = np.max(delta)

        # sanity check to make sure objective gets smaller each iteration
        tempb = np.sum(y - np.matmul(w, x)) / n
        err = np.sum(np.square(np.matmul(w, x)+tempb-y)) + lam*np.sum(np.absolute(w))
        print('error: ', err)

    # save number of nonzero elements and set new lambda
    nonzero = np.count_nonzero(w)
    print('nonzero:', nonzero)
    nonzeros.append(nonzero)
    lam = lam / 1.5

    # calculate TPR and FDR
    TPR.append(np.count_nonzero(w[0: k]) / k)
    if nonzero == 0:
        FDR.append(0)
    else:
        FDR.append(np.count_nonzero(w[k: d]) / nonzero)

# plot for a
plt.plot(lambdas, nonzeros)
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Non-zeros')

# plot for b
plt.figure()
plt.subplot(2, 1, 1)
plt.plot(lambdas, FDR)
plt.plot(lambdas, TPR)
plt.xlabel('Lambda')
plt.ylabel('Ratio')
```

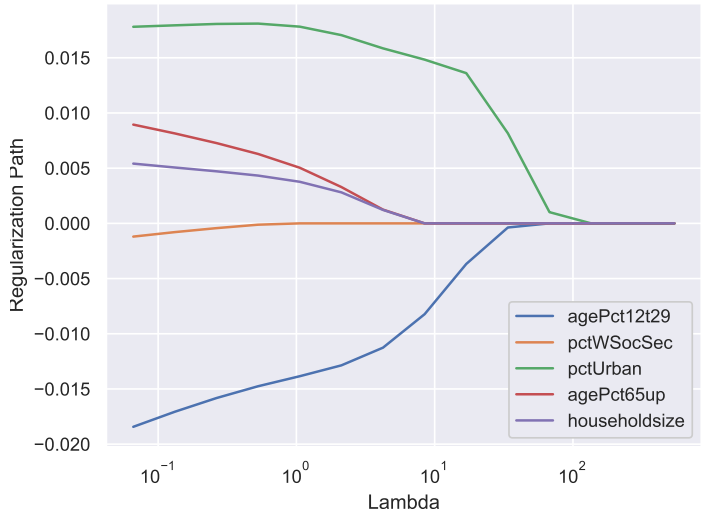
```
plt.legend(('FDR', 'TPR'))
plt.subplot(2, 1, 2)
plt.plot(FDR, TPR)
plt.xlabel('FDR')
plt.ylabel('TPR')
```

A.5

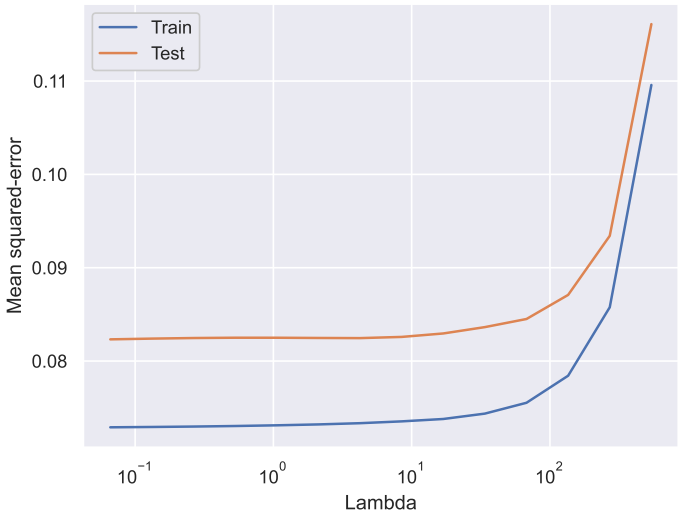
a. *[4 points]*



b. *[4 points]*



c. *[4 points]*



d. *[4 points]* For $\lambda = 30$ PctIlleg has the largest Lasso coefficient and PctFam2Par the lowest. PctIlleg is percentage of kids born to unmarried couples, and PctFam2Par is percentage of families with kids that are headed by two parents. They make intuitive sense as higher crime is often correlated with areas with low socioeconomic status, leading to poor healthcare and access to contraception. More families with both parents present generally indicate stability, thus negatively correlating with crime.

e. *[4 points]* Correlation does not mean causation. There are various factors that correlate with people living to be older, such as income, that could be the true underlying cause. Thus simply moving people over 65 to an area will not decrease the crime rate.

Source Code

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

sns.set()

# to find the maximum lambda that gives all zero in w
def lambdamax(x, y, n):
    yres = y - np.sum(y) / n
    xres = np.matmul(x, yres)
    xsum = 2 * np.absolute(xres)
    return np.max(xsum)

df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")

temp = df_train.values
n = len(df_train)
d = 95
y = temp[:, 0]
x = np.transpose(temp[:, 1:d+1])
temp = df_test.values
ytest = temp[:, 0]
```

```
xtest = np.transpose(temp[:, 1:d+1])
lam = lambdamax(x, y, n)
w = np.zeros(d)

print('hi')

deltalim = 0.01
minlam = 0.1
nonzero = 0
lambdas = []
nonzeros = []
regpath = []
trainmse = []
testmse = []

# Keep checking smaller lambdas until at least 900 elements in w are nonzero
# a can be calculated outside the for loop
a = 2 * np.sum(np.square(x), axis=1)
while lam > 0.1:
    lambdas.append(lam)
    print('lambda', lam)

    # gradient descent
    maxdelta = float('inf')
    while maxdelta > deltalim:

        # b can be calculated outside of the for loop
        b = np.sum(y - np.matmul(w, x)) / n

        oldw = w.copy()

        for j in range(d):

            # calculate ck
            tempw = np.delete(w, j)
            tempx = np.delete(x, j, 0)
            ck = 2 * np.dot(np.transpose(x[j, :]), y - (b + np.matmul(tempw, tempx)))

            # determine wk
            if ck < -lam:
                w[j] = (ck + lam) / a[j]
            elif ck > lam:
                w[j] = (ck - lam) / a[j]
            else:
                w[j] = 0

        # find difference, set variables for conditions
        delta = np.absolute(oldw - w)
        maxdelta = np.max(delta)

        # sanity check to make sure objective gets smaller each iteration
        tempb = np.sum(y - np.matmul(w, x)) / n
        err = np.sum(np.square(np.matmul(w, x)+tempb-y)) + lam*np.sum(np.absolute(w))
        print('error: ', err)
```

```
    # save number of nonzero elements and set new lambda
    nonzero = np.count_nonzero(w)
    print('nonzero:', nonzero)
    nonzeros.append(nonzero)
    lam = lam / 2
    regpath.append(w.copy())
    error = np.sum(np.square(np.matmul(w, x)-y)) / n
    trainmse.append(error)
    error = np.sum(np.square(np.matmul(w, xtest)-ytest)) / np.shape(xtest)[1]
    testmse.append(error)

# plot a
plt.plot(lambdas, nonzeros)
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Non-zeros')

# plot b ind = 4, 13, 8, 6, 2
agePct12t29 = []
pctWSocSec = []
pctUrban = []
agePct65up = []
householdsize = []
for i in range(len(regpath)):
    agePct12t29.append(regpath[i][3])
    pctWSocSec.append(regpath[i][12])
    pctUrban.append(regpath[i][7])
    agePct65up.append(regpath[i][5])
    householdsize.append(regpath[i][1])
plt.plot(lambdas, agePct12t29)
plt.plot(lambdas, pctWSocSec)
plt.plot(lambdas, pctUrban)
plt.plot(lambdas, agePct65up)
plt.plot(lambdas, householdsize)
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Regularization Path')
plt.legend(('agePct12t29', 'pctWSocSec', 'pctUrban', 'agePct65up', 'householdsize'))

# plot c
plt.figure()
plt.plot(lambdas, trainmse)
plt.plot(lambdas, testmse)
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Mean squared-error')
plt.legend(('Train', 'Test'))

# plot d
plt.figure()
plt.plot(w)
MAX = df_train.columns[np.argmax(w)+1]
MIN = df_train.columns[np.argmin(w)+1]
```

Logistic Regression

A.6

a. *[8 points]* To find $\nabla_w J(w,b)$:

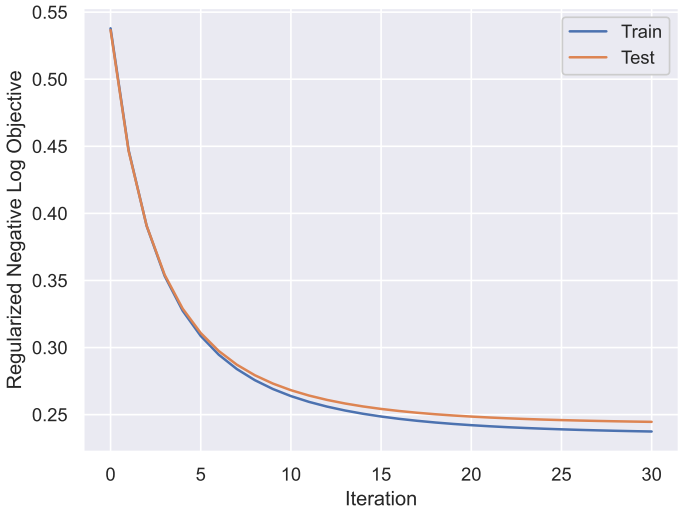
$$\begin{aligned}\nabla_w J(w,b) &= \nabla_w \left[\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^T w))) + \lambda||w||_2^2\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i x_i^T \exp(-y_i(b+x_i^T w))}{1+\exp(-y_i(b+x_i^T w))} + 2\lambda w \\ &= \frac{1}{n}\sum_{i=1}^{n} -y_i x_i^T(1-\mu_i(w,b)) + 2\lambda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

To find $\nabla_b J(w,b)$:

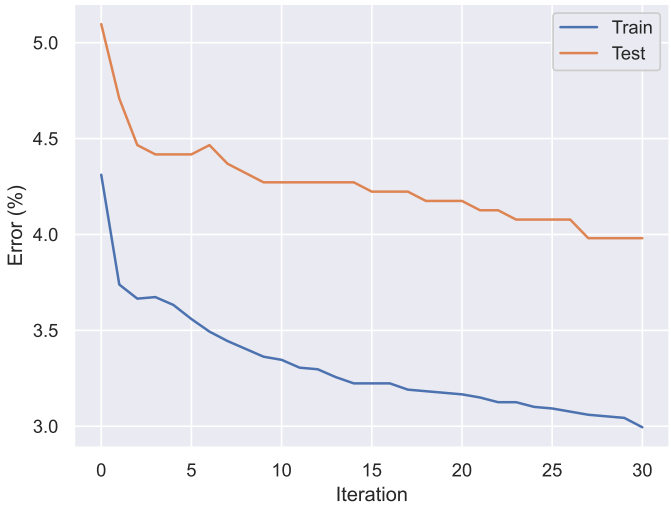
$$\begin{aligned}\nabla_b J(w,b) &= \nabla_b \left[\frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_i(b+x_i^T w))) + \lambda||w||_2^2\right] \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i \exp(-y_i(b+x_i^T w))}{1+\exp(-y_i(b+x_i^T w))} \\ &= \frac{1}{n}\sum_{i=1}^{n} -y_i(1-\mu_i(w,b))\end{aligned}$$

b. *[8 points]*

i)

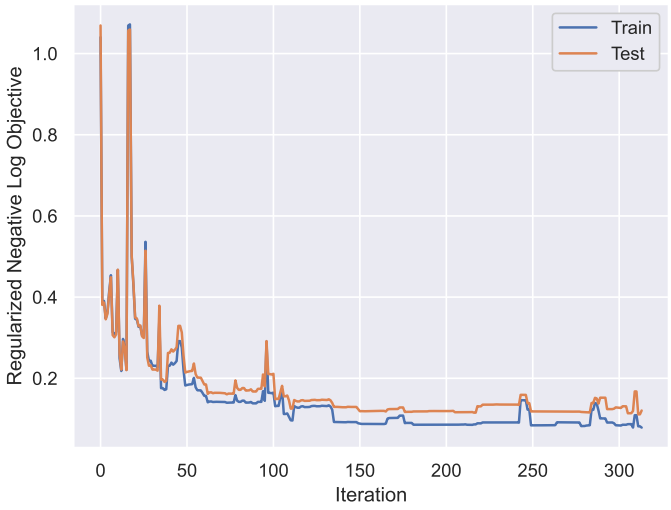


ii)

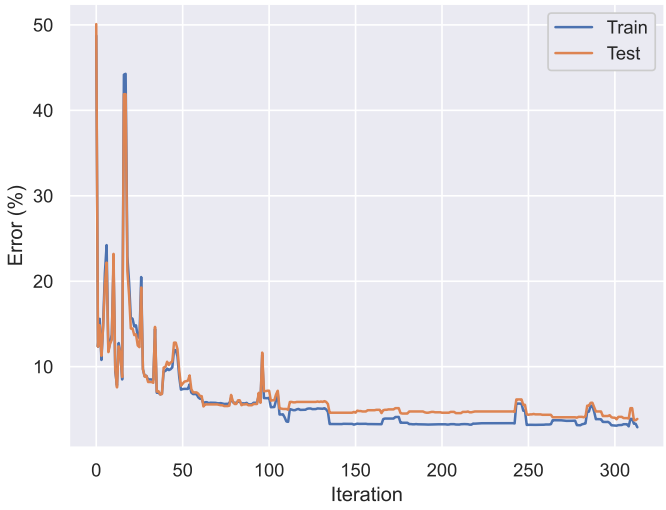


c. *[7 points]*

i)

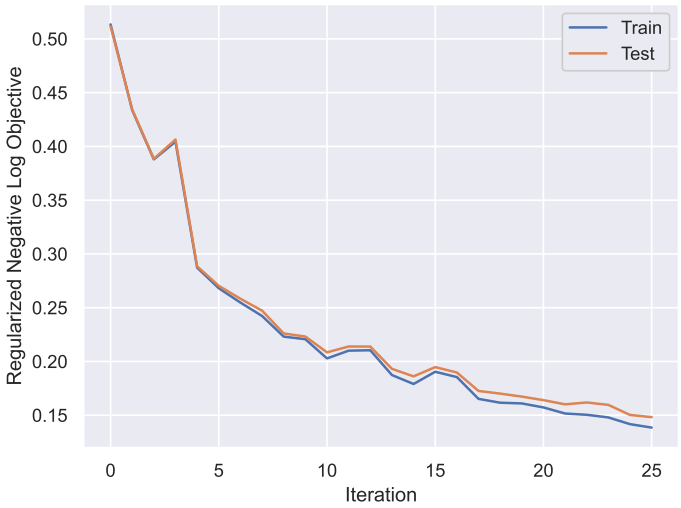


ii)

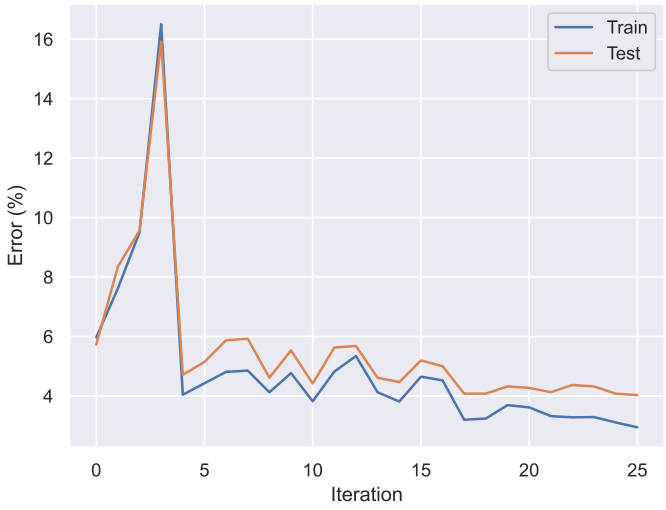


d. *[7 points]*

i)



ii)



Source Code

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

def rearrangeData(d, l):
    ind = np.logical_or(l == 2, l == 7)
    data = d[ind, :]
    label = l[ind]
    label = label.astype('int16')
    label[label == 2] = -1
    label[label == 7] = 1
    return data, label

def muwb(x, y, w, b):
    return 1/(1+np.exp(-y * (b + np.matmul(x, w))))

def objective(x, y, w, b, n, lam):
    return np.sum(np.log(1/muwb(x, y, w, b)))/n + lam*np.linalg.norm(w, 2)**2

# Load data
mndata = MNIST('../MNIST/data/')
X_train, labels_train = map(np.array, mndata.load_training())
X_test, labels_test = map(np.array, mndata.load_testing())

# Rearrange data
X_train = X_train/255.0
X_test = X_test/255.0
X_train, labels_train = rearrangeData(X_train, labels_train)
X_test, labels_test = rearrangeData(X_test, labels_test)

# Initialize variables
```

```
n = np.shape(X_train)[0]
ntest = np.shape(X_test)[0]
d = np.shape(X_train)[1]
w = np.zeros(d)
b = 0
lam = 0.1
eta = 0.1
trainerr = []
testerr = []
obj = []
objtest = []
lasterr = 100
batch = 10

# rescale lambda
lam = lam / (n/batch)

while lasterr > 3:

    # random sampling
    ind = random.sample(range(n), batch)
    x = X_train[ind, :]
    y = labels_train[ind]

    oldw = w.copy()
    gradw = -np.matmul(y * (1-muwb(x, y, oldw, b)), x) / batch + 2*lam*w
    w = w - eta*(gradw)
    gradb = -np.matmul(y, (1-muwb(x, y, oldw, b))) / batch
    b = b - eta*(gradb)

    obj.append(objective(X_train, labels_train, w, b, n, lam))
    objtest.append(objective(X_test, labels_test, w, b, ntest, lam))

    err = np.sign(np.matmul(X_train, w)+b) # probability y=1 (that it is 7)
    trainerr.append((1-np.count_nonzero(err == labels_train)/n)*100)
    err = np.sign(np.matmul(X_test, w)+b)
    testerr.append((1-np.count_nonzero(err == labels_test)/ntest)*100)

    lasterr = trainerr[len(trainerr)-1]
    print(lasterr)

# plot i
plt.plot(obj)
plt.plot(objtest)
plt.xlabel('Iteration')
plt.ylabel('Regularized Negative Log Objective')
plt.legend(('Train', 'Test'))

# plot ii
plt.figure()
plt.plot(trainerr)
plt.plot(testerr)
plt.xlabel('Iteration')
plt.ylabel('Error (%)')
plt.legend(('Train', 'Test'))
```