# CSE 546 Homework #2 -A

Lu Yu

May 13, 2020

## Conceptual Questions

A.0

a. *[2 points]* No, the larger positive weight variable may be one of several dependent variables. If we remove the attribute, the other dependent variable may has a larger coefficient. The new model will not be strictly worse than before because the collinearity is eliminated.

b. *[2 points]* For lasso, penalty term is absolute. Because of the nature of the constraint($\sum_{j=1}^{p} |\beta_j| \leq t$), making t sufficiently small will cause some of the coefficients to be exactly zero.
While for L2 norm penalty($\sum_{j=1}^{p} \beta_j^2 \leq t$), the penalty term is squared, so squaring a small value will make it smaller. We don't have to make it zero to achieve our aim to get a minimum square error, we will get it before that.

c. *[2 points]* Ad: The shrinkage speed will be faster than lasso penalty, which results in more coefficients turning to zero faster. Therefore, the model is smooth, simple and easy to explain and the computation time is less.
Dis: Low precision and this constraint makes the solutions nonlinear in the $y_i$, and there is no closed form expression.

d. *[1 points]* True

e. *[2 points]* We can see each data point as a piece of feedback, in GD, we review all feedback and make improvements and then review all feedbacks on the improved model again. While in SGD, we randomly pick a piece of feedback to improve and get a new piece of feedback on the improved model. The direction of GD and SGD is the same, i.e. let all feedback satisfied.

f. *[2 points]* Ad: If the sample size is large, SGD's computational complexity has an advantage over GD.
Dis: SGD is more noisy than GD and has less accuracy, making SGD not the overall optimization direction for each iteration.

## Convexity and Norms

A.1

a. *[3 points]* Show that $f(x) = (\sum_{i=1}^{n} |x_i|)$ is a norm.
iii) triangle inequality: $f(x+y) = (\sum_{i=1}^{n} |x_i + y_i|) \leq (\sum_{i=1}^{n}(|x_i| + |y_i|)) = (\sum_{i=1}^{n}(|x_i|) + \sum_{i=1}^{n}(|y_i|)) = f(x) + f(y)$
ii) absolute scalability: $f(ax) = (\sum_{i=1}^{n} |ax_i|) = (\sum_{i=1}^{n} |a||x_i|) = |a|(\sum_{i=1}^{n} |x_i|) = |a|f(x)$
i) non-negative: $f(x) = \sum_{i=1}^{n} |x_i| \geq 0$, with equality if and only if $\forall x_i, x_i = 0$

b. *[2 points]* For point a: $(x_1, x_2) = (0, -4)$, and point b: $(x_1, x_2) = (-4, 0)$

$$g(a) = g(0,-4) = (0 + \sqrt{4})^2 = 4$$
$$g(b) = g(-4,0) = (\sqrt{4} + 0)^2 = 4$$
$$g(a+b) = g(-4,-4) = (\sqrt{4} + \sqrt{4})^2 = 16 > 4 + 4 = g(a) + g(b)$$

This implits the triangle inequality does not hold.

A2. *[3 points]*
I. Not a convex. The middle point in line b-c is not in grey-shaded sets.
II. Convex.
III. Not a convex. The middle point in line a-d is not in grey-shaded sets.

A3.*[4 points]*
a. Convex.
b. Not a convex. $\lambda = 0.5, f(0.5a + 0.5b) > 0.5f(a) + 0.5f(b)$
c. Not a convex. $\lambda = 0.5, f(0.5b + 0.5c) > 0.5f(b) + 0.5f(c)$
d. Convex.

## Lasso

A.4

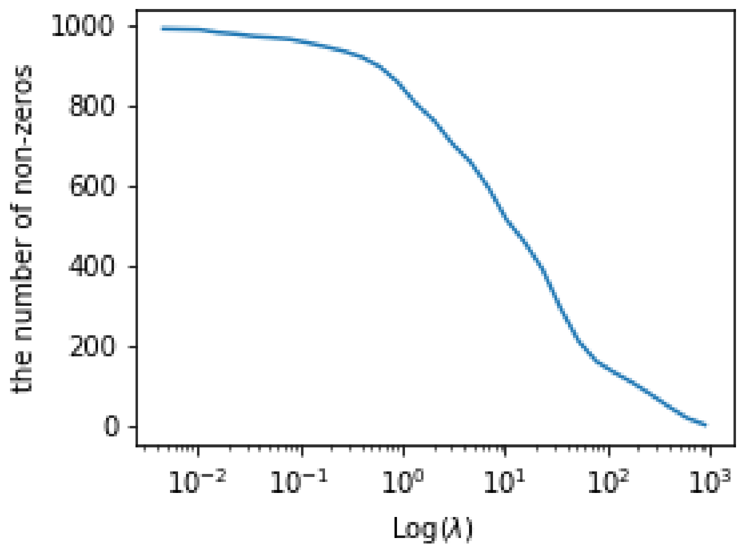
[1]:
```
#HW2 A4 CSE 546
#Lu Yu 2020.5
#Lasso

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

n = 500
d = 1000
k = 100
```

[4]:
```
def createdata():
    """
    Creating some synthetic data
    w   d-by-1
    x   n-by-d
    G   n-by-1
    y   n-by-1
    """
    w=np.zeros(d)
    w[np.arange(k)]=(np.arange(k)+1)/k
    mu, sigma = 0, 1 # mean and standard deviation
    G = np.random.normal(mu, sigma, size = n)
    x = np.random.normal(0, 1, (n, d))
    y = x.dot(w) + G
    return w, x, y

def coorddescent(lambduh, w, x, y, beta = 0.0005):
    a = np.sum(2*(x**2),axis=0)
    diff = beta + 1
    table = []
    while diff > beta:
        table.append((lambduh,int(sum(w!=0)),int(sum(w[:k,]!=0))))
        w_old = np.array(w)
        b = (y - x.dot(w)).mean()
        for j in range(d):
            c = 2*x[:,j].dot(y-(b+x.dot(w)-x[:,j]*w[j]))
            if c < -lambduh:
                w[j] = (c+lambduh)/a[j]
            elif c > lambduh:
                w[j] = (c-lambduh)/a[j]
            else:
                w[j] = 0
        diff = max(abs(w - w_old))
        lambduh = lambduh/1.5
    return table
```

[5]:
```
w, x, y = createdata()
lambda_max = np.max(2*abs(x.T.dot(y-y.mean())))
result = np.array(coorddescent(lambda_max, w, x, y ,beta = 0.0001))
```

[6]:
```
#Plot results
plt.figure(figsize = (4,3))
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('the number of non-zeros')
plt.plot(result[1:,0],result[1:,1])
```

[6]:
```
[<matplotlib.lines.Line2D at 0x11d187c50>]
```

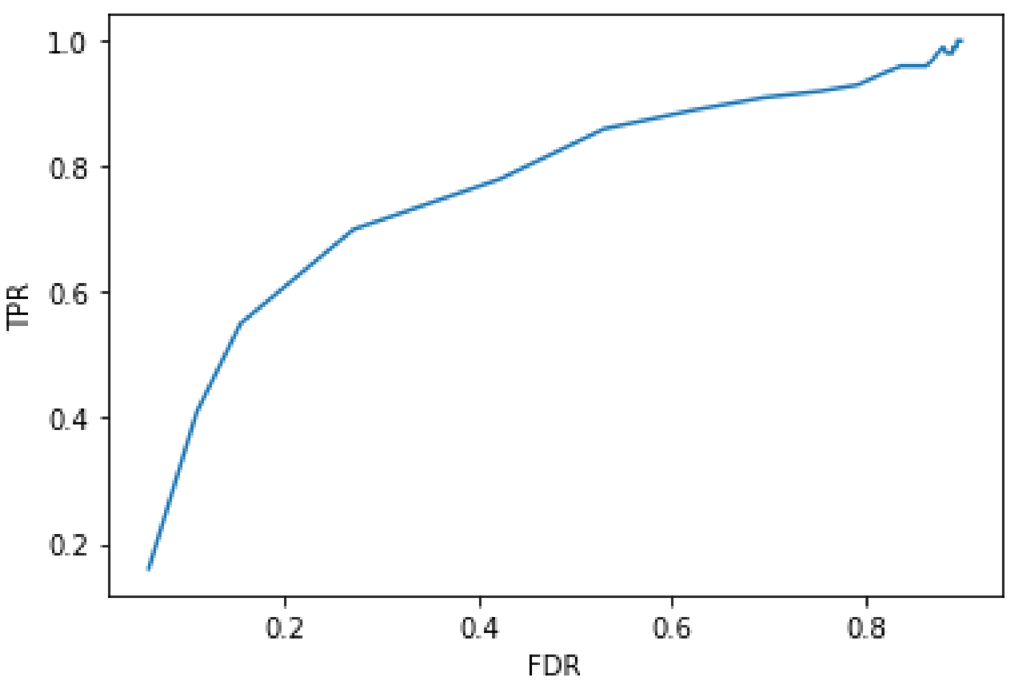


[5]:
```
FDR = (result[1:,1] - result[1:,2])/result[1:,1]
TPR = result[1:,2]/k
plt.xlabel('FDR')
plt.ylabel('TPR')
plt.plot(FDR,TPR)
```

[5]:
```
[<matplotlib.lines.Line2D at 0x121562f28>]
```

A4.c Comment on the effect of $\lambda$ in these two plots.

With the increasement of the $\lambda$, the penalty is increasing which will force the $\lambda$ to be zero to decrease the loss function. Therefore, when the $\lambda$ is large enough, there number of non-zeros will be zero. Thus, in second plot, it will result in a (0,0) pair in FDR and TPR. However, when $\lambda$ is zero, then the penalty disappears. Number of correct nonzeros will be k and total number of nonzeros is d, which will lead to a ($\frac{d-k}{d}$,1) pair in FDR and TPR.



[62]:
```
#HW2 A5 CSE 546

'''
load real data
'''
df_train = pd.read_table("crime-train.txt")
df_test = pd.read_table("crime-test.txt")

y_train = df_train.values[:,0]#.reshape(len(df_train),1)
X_train = df_train.values[:,1:].reshape(len(df_train),df_train.shape[1]-1)
y_test = df_test.values[:,0]#.reshape(len(df_test),1)
X_test = df_test.values[:,1:].reshape(len(df_test),df_test.shape[1]-1)
```

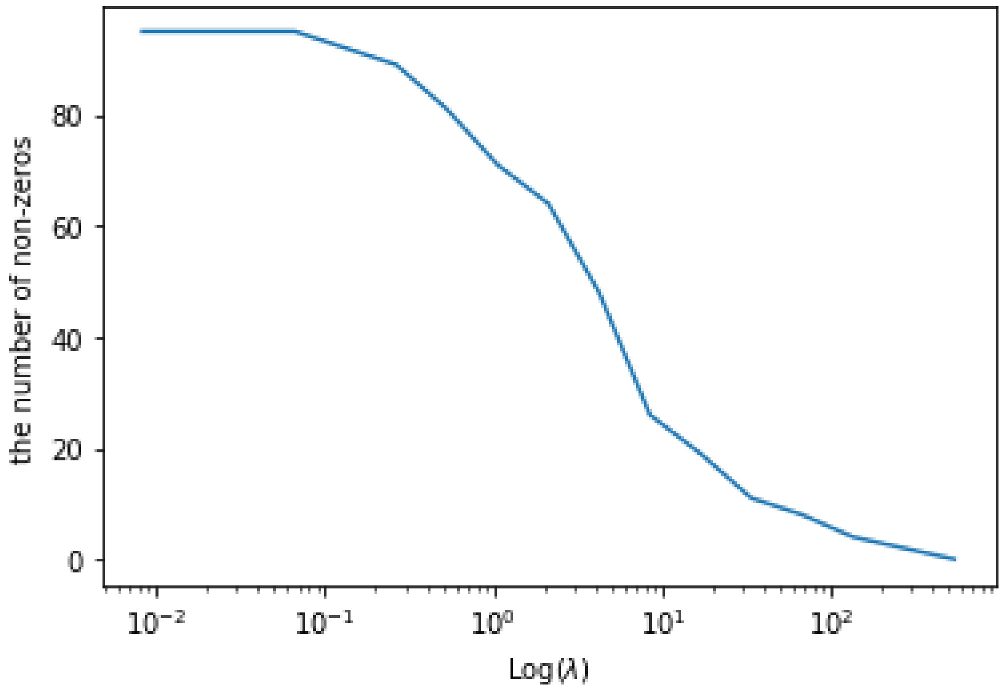
```
/Users/apple/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:6:
FutureWarning: read_table is deprecated, use read_csv instead, passing sep='\t'.

/Users/apple/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7:
FutureWarning: read_table is deprecated, use read_csv instead, passing sep='\t'.
  import sys
```

[64]:
```
def soft_threshold(rho,lamda):
    '''Soft threshold function used for normalized data and lasso regression'''
    if rho < - lamda:
        return (rho + lamda)
    elif rho >  lamda:
        return (rho - lamda)
    else:
        return 0

def coordinate_descent_lasso(theta,X,y,lamda = .01):
    #Initialisation of useful values
    m,n = X.shape
    beta = 0.0005
    diff = beta + 1
    #Looping until max number of iterations
    a = np.sum(2*(X**2),axis=0)
    while diff > beta:
        theta_old = np.array(theta)
        #Looping through each coordinate
        b = (y - X.dot(theta)).mean()
        for j in range(n):
            rho = 2*X[:,j].dot(y-(b+X.dot(theta)-X[:,j]*theta[j]))
            theta[j] =  soft_threshold(rho, lamda)/a[j]
        diff = max(abs(theta - theta_old))
    return theta.flatten()

def lamda(X,y):
    lambda_ = np.max(2*abs(X.T.dot(y-y.mean())))
    lamda_list = []
    lamda_list.append(lambda_)
    while lambda_>0.01:
        lambda_ = lambda_/2
        lamda_list.append(lambda_)
    lamda = np.array(lamda_list) #Range of lambda values
    return lamda

def main_(X,y,X_test,y_test,lamda):
    # Initialize variables
    m,n = X.shape
    theta = np.zeros(n)
    theta_list = list()
    result = []
    #Run lasso regression for each lambda
    for l in lamda:
        theta = coordinate_descent_lasso(theta,X,y,lamda = l)
        error_train = np.mean((y - X.dot(theta))**2)
        error_test = np.mean((y_test - X_test.dot(theta))**2)
        theta_list.append(theta)
        result.append((l,int(sum(theta!=0)),error_train,error_test)) #record␣
 ↪non-zeros theta

    #Stack into numpy array
    theta_lasso = np.stack(theta_list).T
    return theta_lasso,result
```

[65]:
```
lamda = lamda(X_train,y_train)
theta_lasso, result = main_(X_train,y_train,X_test,y_test,lamda)
```

[66]:
```
'''
Plot the number of nonzeros of each solution versus λ.
'''
#Plot results
#plt.figure(figsize = (4,3))
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('the number of non-zeros')
plt.plot(np.array(result)[:,0],np.array(result)[:,1])
```

[66]:
```
[<matplotlib.lines.Line2D at 0x123b090f0>]
```

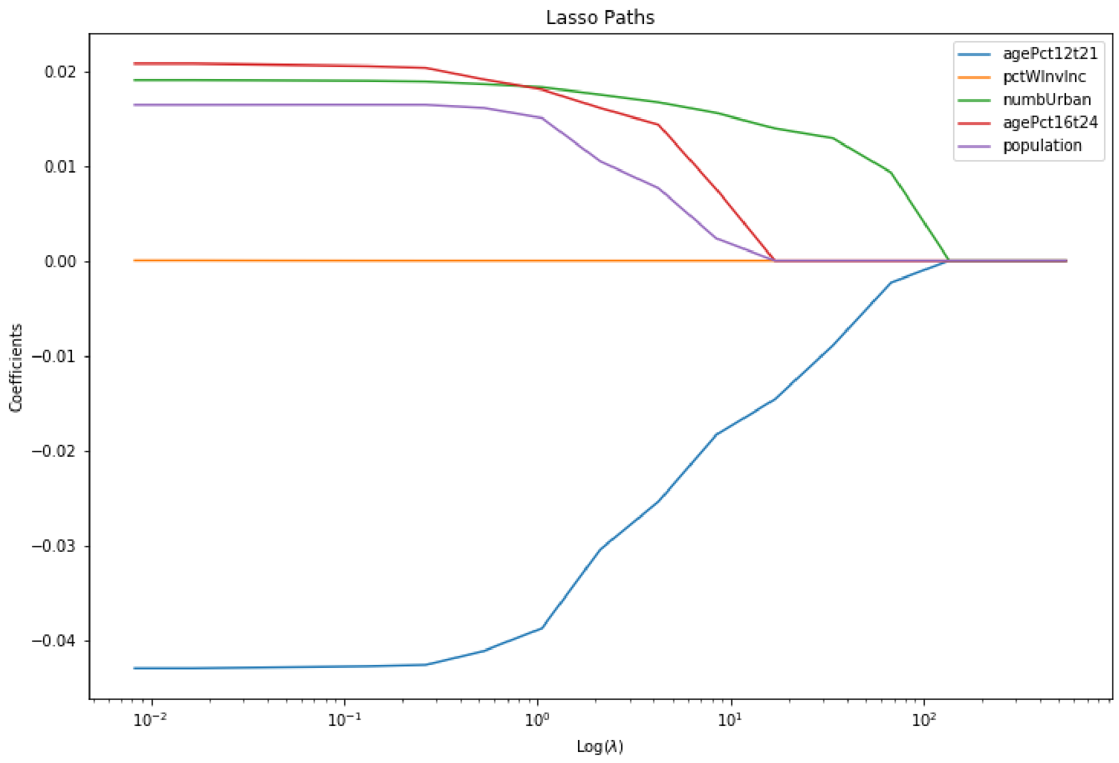


[67]:
```
column=list(df_train.columns)
index = (column.index('agePct12t29')-1,column.index('pctWSocSec')-1, column.
 ↪index('pctUrban')-1, column.index('agePct65up')-1, column.
 ↪index('householdsize')-1)
```

[69]:
```
'''
Plot the regularization paths (in one plot) for the coefficients for input␣
 ↪variables
```

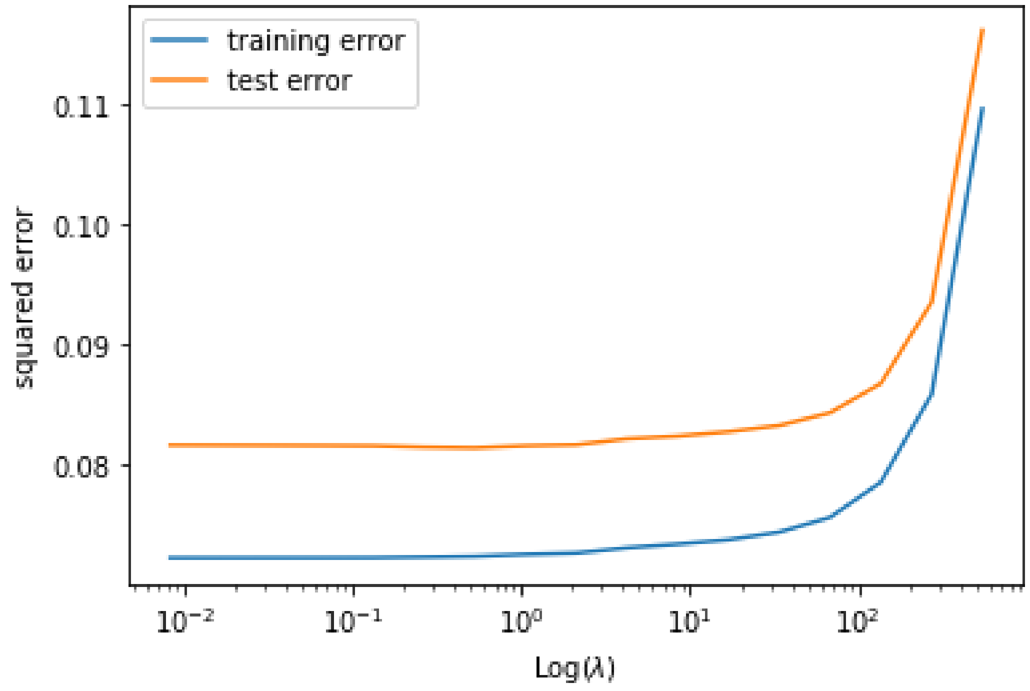
```
'''
#Plot results
plt.figure(figsize = (12,8))

for i in index:
    plt.plot(lamda, theta_lasso[i],label = df_train.columns[i])

df_train.columns[1]
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('Coefficients')
plt.title('Lasso Paths')
plt.legend()
plt.axis('tight')
```

[69]:
```
(0.0047454847328514335,
 942.7759694748172,
 -0.04618787784884825,
 0.023962635093624868)
```



[44]:
```
'''
Plot the squared error on the training and test data versus λ.
'''
plt.xscale('log')
plt.xlabel('Log($\\lambda$)')
plt.ylabel('squared error')
plt.plot(np.array(result)[:,0],np.array(result)[:,2],label = "training error")
plt.plot(np.array(result)[:,0],np.array(result)[:,3],label = "test error")
plt.legend()
```

[44]:
```
<matplotlib.legend.Legend at 0x1237d06a0>
```



[71]:
```
'''
A.5 part d
lamda = 30
'''
lamda = np.array([30])
theta_lasso, result = main_(X_train,y_train,X_test,y_test,lamda)
print('the largest (most positive) Lasso coefficient: ', df_train.columns[np.
 ↪argmax(theta_lasso)+1])
print('the smallest (most negative) Lasso coefficient: ', df_train.columns[np.
 ↪argmin(theta_lasso)+1])
```

```
the largest (most positive) Lasso coefficient:  PctIlleg
the smallest (most negative) Lasso coefficient:  PctKids2Par
```

[ ]:
```
'''A.5 part d (Cont'd)
PctIlleg: percentage of kids born to never married
```

```
    If more young people are gathered at this location, the crime rate may be␣
 ↪higher.
    More young people usually means higher rates of crime.

PctKids2Par: percentage of kids in family housing with two parents
    If children and parents live together in this place, they may be in a␣
 ↪residential
    area. This location is not like the city center, the crime rate is␣
 ↪relatively low.
'''
```

[ ]:
```
'''
HW 2 A.5 part e
This understanding reverses cause and effect. It is precisely because there
are more elderly people over the age of 65 living here, so the crime rate
is low. It's as if the fire truck appeared because the building was on fire.
But if it is reversed, it may not happen.
'''
```

A.6
part.a

$$J(w,b) = \frac{1}{n}\sum_{x,y}\log(1+\exp\left(-y\cdot(w^\top x+b)\right) + \lambda w^\top w$$

$$\nabla_w J(w,b) = \frac{1}{n}\frac{\sum_{x,y} -xy\cdot exp(-y\cdot(w^\top x+b))}{1+exp(-y\cdot(w^\top x+b)} + 2\lambda w = \frac{1}{n}\sum_{i=1}^{n} x_i y_i(\mu_i(w,b)-1) + 2\lambda w$$

$$\nabla_b J(w,b) = \frac{1}{n}\frac{\sum_{x,y} -y\cdot exp(-y\cdot(w^\top x+b))}{1+exp(-y\cdot(w^\top x+b)} = \frac{1}{n}\sum_{i=1}^{n} y_i(\mu_i(w,b)-1)$$

[51]:
```
"""
CSE 546 HW 2 A.6
@author: Leah
"""
import numpy as np
from mnist import MNIST
from matplotlib import pyplot as plt


def load_dataset():
    mndata = MNIST('./data/')
    #mndata = MNIST('./dir_with_mnist_data_files')
    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

    return X_train, labels_train, X_test, labels_test
```

```
def weightInitialization(n_features):
    w = np.zeros(n_features)
    b = 0
    return w,b

def sigmoid(x):
    return (1/(1 + np.exp(-x))) #mu_w

def probability(X, y, w, b):   #\mu
    return sigmoid((np.dot(X,w)+b)*y)

def Model_optimization(w,b,X,y,lamda):
    mu =  probability(X,y,w,b)
    dw = (- np.multiply(X.T, y) * (1 - mu)).mean(axis=1) + 2 * lamda * w
    db = np.mean(( mu -1).T*y)
    return mu, dw, db

def Model_prediction(w, b, X, Y, X_test,Y_test, lamda, learning_rate,␣
 ↪no_iterations):
    cost_history_train = []
    iterations = []
    cost_history_test = []
    misclassified_train_history = []
    misclassified_test_history = []
    for i in range(no_iterations):
        mu, dw, db = Model_optimization(w,b,X,Y,lamda)
        J_train = np.log(1/mu).mean() + lamda * np.linalg.norm(w, ord = 2)
        #weight update
        #Gradient Descent with Ridge Regularization calculation
        w = w - (learning_rate * (dw.T))
        b = b - (learning_rate * db)

        if (i % 20 == 0):
            iterations.append(i)
            cost_history_train.append(J_train)
            #mu,_,_ = Model_optimization(w,b,X_test,Y_test,lamda)
            J_test = np.log(1/probability(X_test,Y_test,w,b)).mean() + lamda *␣
 ↪np.linalg.norm(w, ord = 2)
            cost_history_test.append(J_test)

            X_train_pred = np.dot(X,w)+b
            X_train_pred[X_train_pred <= 0] = -1
            X_train_pred[X_train_pred > 0] = 1
            #misclassified
            misclassified_train  = 1 - (X_train_pred == Y).sum() / float(X.
 ↪shape[0])
            misclassified_train_history.append(misclassified_train)
            X_test_pred = np.dot(X_test,w)+b
            X_test_pred[X_test_pred <= 0] = -1
            X_test_pred[X_test_pred > 0] = 1
            misclassified_test  = 1 - (X_test_pred == Y_test).sum() /␣
↪float(X_test.shape[0])
            misclassified_test_history.append(misclassified_test)

    #final parameters
    coeff = {"w": w, "b": b}
    cost_history = {'train':cost_history_train, 'test':cost_history_test}
    gradient = {"dw": dw, "db": db}
    misclassified = {'train':misclassified_train_history, 'test':
↪misclassified_test_history}

    return coeff, gradient, cost_history,misclassified, iterations

def Model_Prediction_sgd(w, b, X, Y, X_test,Y_test, lamda, learning_rate,␣
↪no_iterations,batch_size = 10):
    cost_history_train = []
    cost_history_test = []
    misclassified_train_history = []
    misclassified_test_history = []
    iterations = []
    m = X.shape[0]

    for i in range(no_iterations):
        # Stochastic Gradient Descent with Ridge Regularization calculation
        idx = np.random.randint(0, m, size=batch_size)
        X_batch = X[idx]
        Y_batch = Y[idx]
        mu, dw, db= Model_optimization(w,b,X_batch,Y_batch,lamda)
        loss_train = np.log(1/probability(X, Y,w,b)).mean() + lamda * np.linalg.
↪norm(w, ord = 2)

        #weight update
        w = w - (learning_rate * (dw.T))
        b = b - (learning_rate * db)

        if (i % 20 == 0):
            iterations.append(i)
            cost_history_train.append(loss_train)
            loss_test = np.log(1/probability(X_test,Y_test,w,b)).mean() + lamda␣
↪* np.linalg.norm(w, ord = 2)
            cost_history_test.append(loss_test)

            X_train_pred = np.dot(X,w)+b
            X_train_pred[X_train_pred <= 0] = -1
            X_train_pred[X_train_pred > 0] = 1
            #misclassified
            misclassified_train  = 1 - (X_train_pred == Y).sum() / float(X.
↪shape[0])
            misclassified_train_history.append(misclassified_train)

            X_test_pred = np.dot(X_test,w)+b
            X_test_pred[X_test_pred <= 0] = -1
            X_test_pred[X_test_pred > 0] = 1
            misclassified_test  = 1 - (X_test_pred == Y_test).sum() /␣
↪float(X_test.shape[0])
            misclassified_test_history.append(misclassified_test)
    #final parameters
    coeff_sgd = {"w": w, "b": b}
    gradient_sgd = {"dw": dw, "db": db}
    cost_history = {'train':cost_history_train, 'test':cost_history_test}
    misclassified = {'train':misclassified_train_history, 'test':
↪misclassified_test_history}

    return coeff_sgd, gradient_sgd, cost_history, misclassified, iterations
```

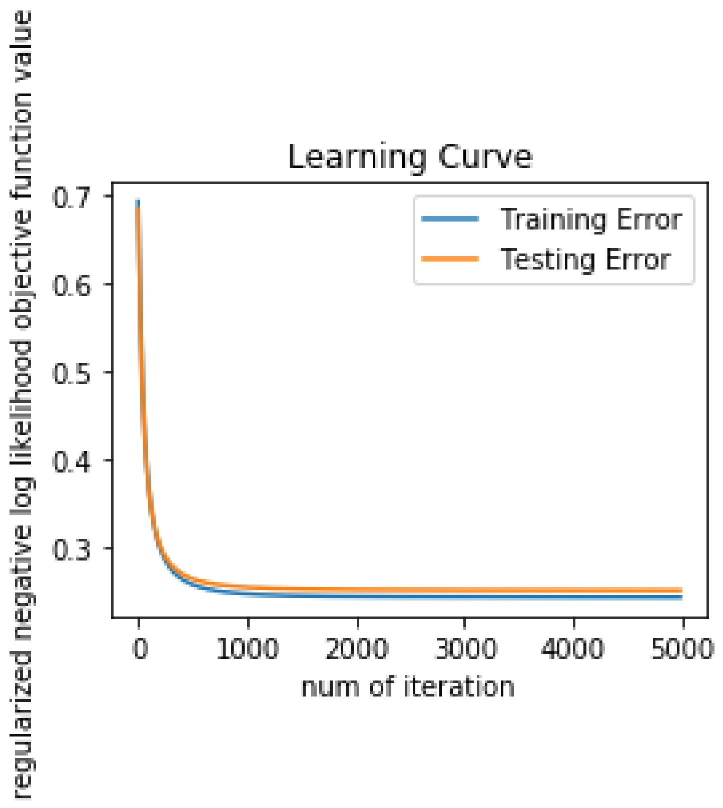
[5]:
```
#-----------------------------------------------------------------
#  Main
#-----------------------------------------------------------------

#LOAD data
X_train, labels_train, X_test, labels_test = load_dataset()

labels_train = np.array(labels_train, dtype = 'int32')
labels_test = np.array(labels_test, dtype = 'int32')

#extract digit 2 and 7
X_train_binary = np.hstack((X_train[np.where(labels_train[:]==2),:],X_train[np.
↪where(labels_train[:]==7),:]))
X_train_binary = X_train_binary[0]
labels_train_binary = np.hstack((labels_train[np.where(labels_train[:
↪]==2)],labels_train[np.where(labels_train[:]==7)]))
labels_train_binary[labels_train_binary[:]==2] = -1
labels_train_binary[labels_train_binary[:]==7] = 1

X_test_binary = np.hstack((X_test[np.where(labels_test[:]==2),:],X_test[np.
↪where(labels_test[:]==7),:]))
X_test_binary = X_test_binary[0]
labels_test_binary = np.hstack((labels_test[np.where(labels_test[:
↪]==2)],labels_test[np.where(labels_test[:]==7)]))
labels_test_binary[labels_test_binary[:]==2] = -1
```

```
labels_test_binary[labels_test_binary[:]==7] = 1
#-----------------------------------------------------------------

n_features = X_train_binary.shape[1]
```

[56]:
```
#GD
n_iter = 5000
eta = 0.005
w, b = weightInitialization(n_features) #Initialization
coeff, gradient, cost_history, misclassified,iterations= Model_prediction(w, b,␣
 ↪X_train_binary, labels_train_binary, X_test_binary, labels_test_binary, lamda␣
 ↪= 0.1, learning_rate = eta, no_iterations=n_iter)
```

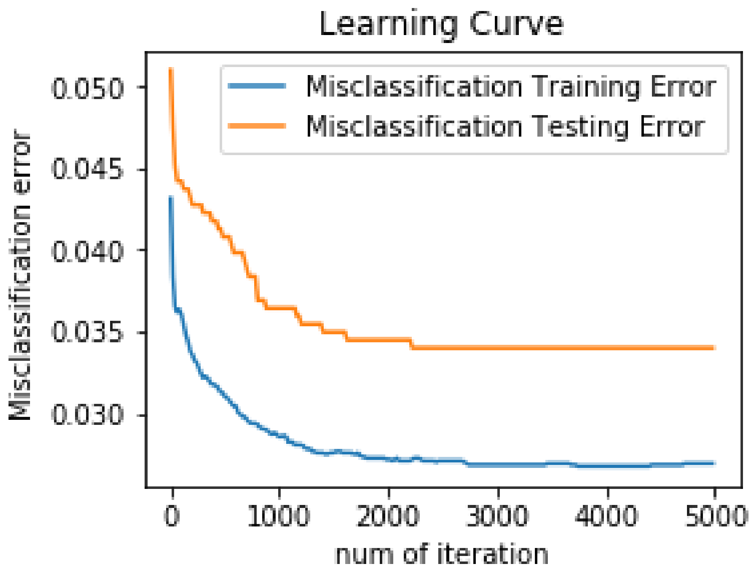
[57]:
```
#Plot results
plt.figure(figsize = (4,3))
plt.xlabel('num of iteration')
plt.ylabel('regularized negative log likelihood objective function value')
plt.plot(iterations[:],cost_history['train'][:])
plt.plot(iterations[:],cost_history['test'][:])
plt.legend(['Training Error', 'Testing Error'], loc='best')
plt.title('Learning Curve')
```

[57]:
```
Text(0.5, 1.0, 'Learning Curve')
```



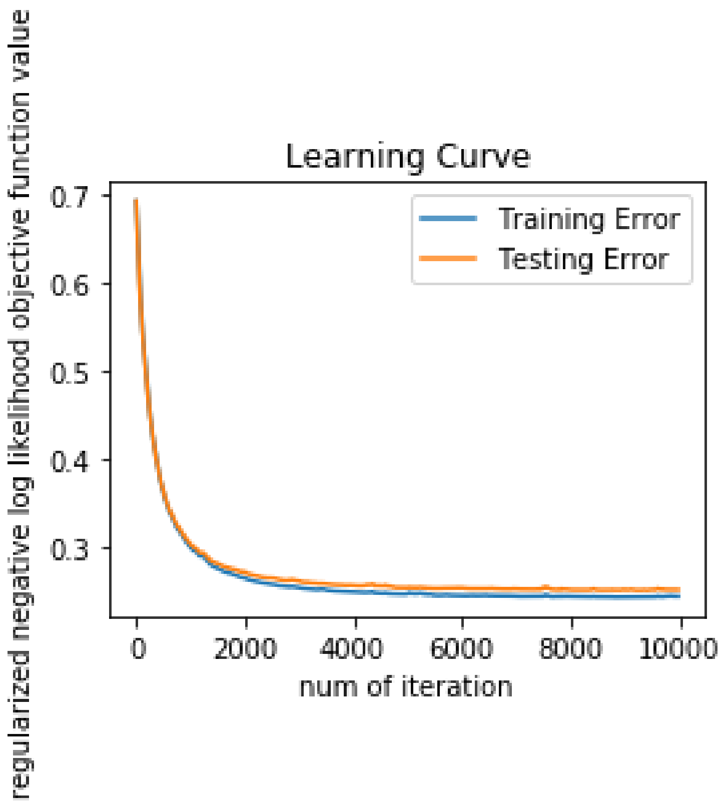
[58]:
```
plt.figure(figsize = (4,3))
plt.xlabel('num of iteration')
plt.ylabel('Misclassification error ')
plt.plot(iterations[:],misclassified['train'][:])
plt.plot(iterations[:],misclassified['test'][:])
plt.legend(['Misclassification Training Error', 'Misclassification Testing␣
 ↪Error'], loc='best')
plt.title('Learning Curve')
```

[58]:
```
Text(0.5, 1.0, 'Learning Curve')
```



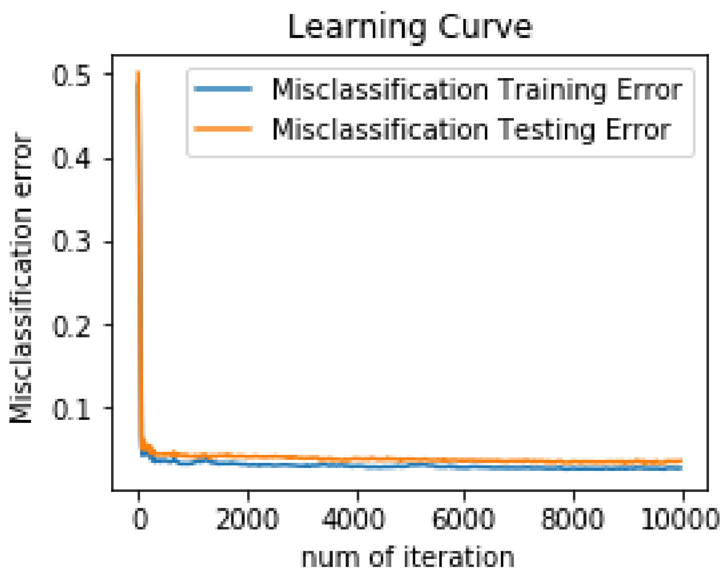
[49]:
```
###SGD
batch_size = 1
n_iter = 10000
eta = 0.001
w, b = weightInitialization(n_features) #Initialization
coeff, gradient,cost_history, misclassified, iterations =␣
 ↪Model_Prediction_sgd(w, b, X_train_binary, labels_train_binary, X_test_binary,␣
 ↪labels_test_binary, lamda = 0.1, learning_rate = eta, no_iterations=n_iter,␣
 ↪batch_size = batch_size)

#Plot results
plt.figure(figsize = (4,3))
plt.xlabel('num of iteration')
plt.ylabel('regularized negative log likelihood objective function value')
plt.plot(iterations[:],cost_history['train'][:])
plt.plot(iterations[:],cost_history['test'][:])
plt.legend(['Training Error', 'Testing Error'], loc='best')
plt.title('Learning Curve')
```

```
[49]: Text(0.5, 1.0, 'Learning Curve')
```



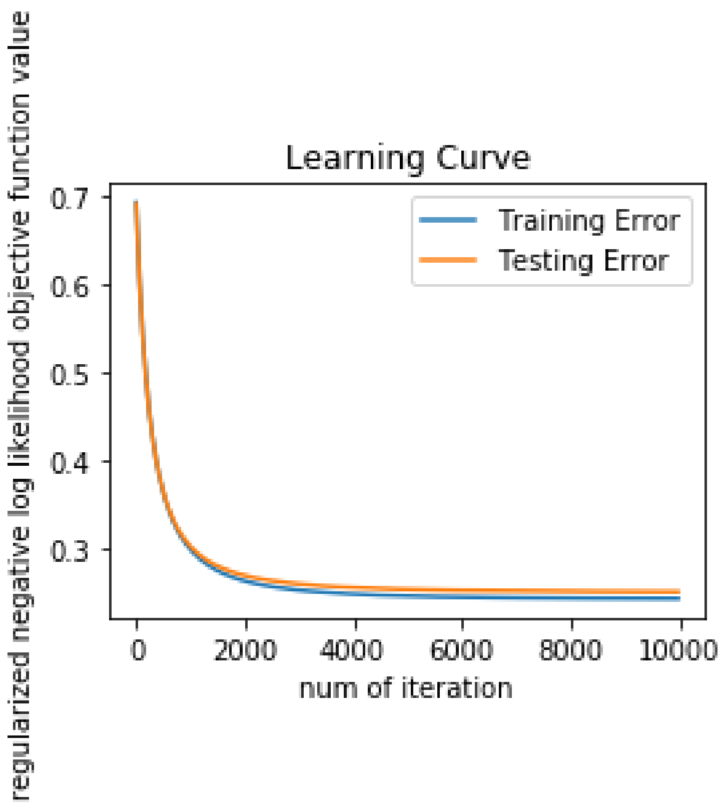
```
[52]: plt.figure(figsize = (4,3))
plt.xlabel('num of iteration')
plt.ylabel('Misclassification error ')
plt.plot(iterations[:],misclassified['train'][:])
plt.plot(iterations[:],misclassified['test'][:])
plt.legend(['Misclassification Training Error', 'Misclassification Testing␣
 ↪Error'], loc='best')
plt.title('Learning Curve')
```

```
[52]: Text(0.5, 1.0, 'Learning Curve')
```



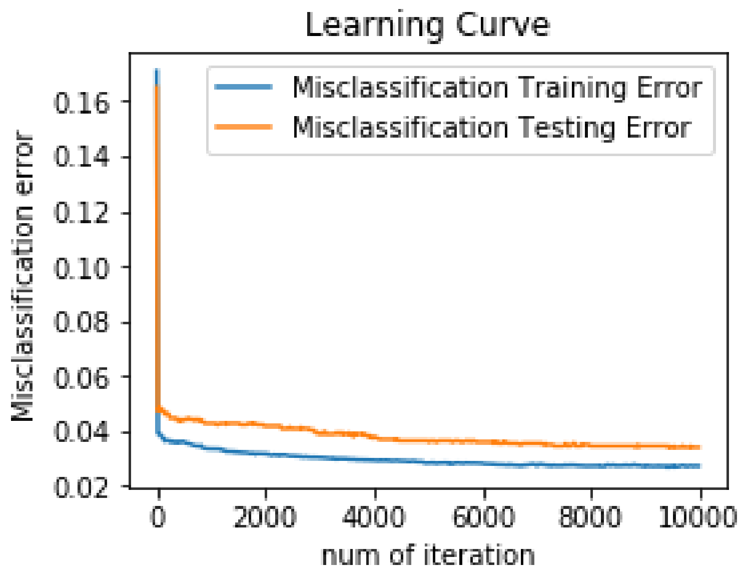
[53]:
```
batch_size = 100
n_iter = 10000
eta = 0.001
w, b = weightInitialization(n_features) #Initialization
coeff, gradient, cost_history, misclassified, iterations = ␣
 ↪Model_Prediction_sgd(w, b, X_train_binary, labels_train_binary, X_test_binary,␣
 ↪labels_test_binary, lamda = 0.1, learning_rate = eta, no_iterations=n_iter,␣
 ↪batch_size = batch_size,)
#Plot results
plt.figure(figsize = (4,3))
plt.xlabel('num of iteration')
plt.ylabel('regularized negative log likelihood objective function value')
plt.plot(iterations[:],cost_history['train'][:])
plt.plot(iterations[:],cost_history['test'][:])
plt.legend(['Training Error', 'Testing Error'], loc='best')
plt.title('Learning Curve')
```

[53]:
```
Text(0.5, 1.0, 'Learning Curve')
```



[54]:
```
plt.figure(figsize = (4,3))
plt.xlabel('num of iteration')
plt.ylabel('Misclassification error ')
plt.plot(iterations[:],misclassified['train'][:])
plt.plot(iterations[:],misclassified['test'][:])
plt.legend(['Misclassification Training Error', 'Misclassification Testing␣
 ↪Error'], loc='best')
plt.title('Learning Curve')
```

[54]:
```
Text(0.5, 1.0, 'Learning Curve')
```


Learning Curve
Misclassification Training Error
Misclassification Testing Error
Misclassification error
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0
2000
4000
6000
8000
10000
num of iteration