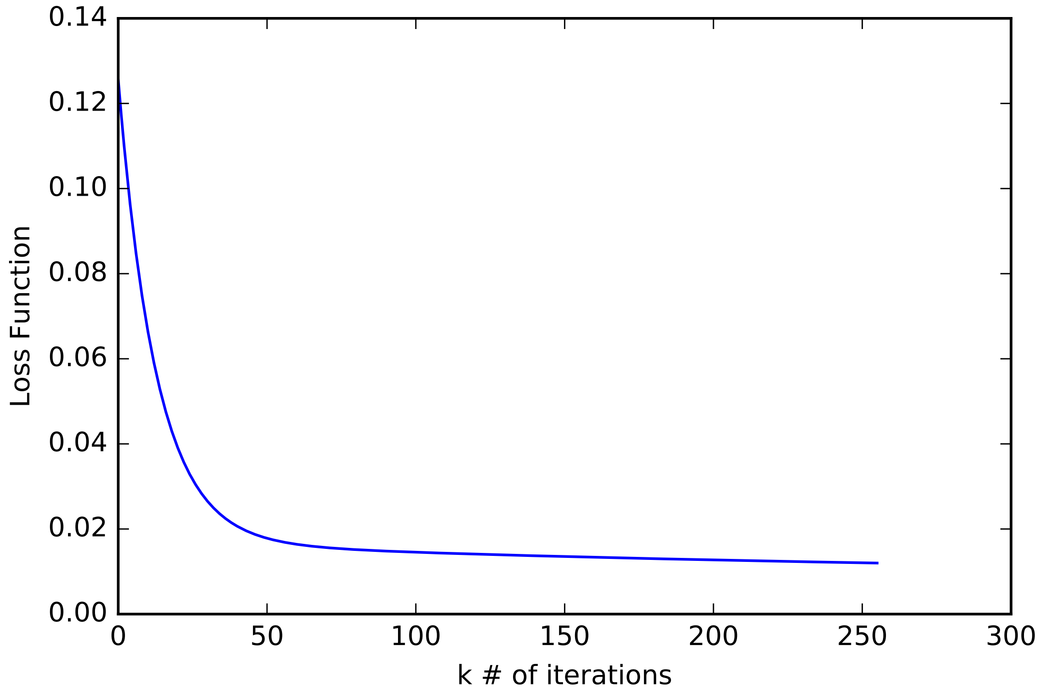
Seyed Hamid Tabari
Spring 2019
University of Arkansas
shtabari@uark.edu

Homework 1

Answer 1.4:

Answer 1:
Plot of loss function $J_k(\theta)$ VS. number of iterations (k)



Answer 2:
Average sum of squared errors:

$$\frac{1}{2m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 = 0.01576348$$

Answer 3:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{2m}g(\theta_j)$$

and,

$$g(\theta_j) = \begin{cases} 1, & \text{if } \theta_j \geq 0 \\ -1, & \theta_j < 0 \end{cases}$$

Answer 4:
The number of non-zero parameters of ridge (1.2) and lasso (1.3) are 14. Both methods have equal number of non-zero parameters.

Answer 5:
CODE 1.2.py

```
import matplotlib
import matplotlib.pyplot as plt
```

```python
import numpy as np

data0=np.loadtxt("data.txt",skiprows=19)
######################################## scaling and centring - 1.1 part
colmean=np.min(data0,axis=0)
datac=data0-colmean
data = datac / (np.ptp(datac,axis=0))
datatr,datats=data[:48,:], data[48:,:]
md=np.size(data,axis=0)
m=np.size(datatr,axis=0)
n=np.size(datatr,axis=1)-1
one=np.ones(m).reshape((m,1))
y=datatr[:,n].reshape((m,1))
XX=datatr[:,1:16].reshape((m,n-1))
X=np.hstack((one,XX))
######################################## data is ready
######################################## defining auxiliary function
def dot2(a,b):
        n=np.size(a,axis=0)
        m=np.size(b,axis=1)
        return np.dot(a,b).reshape(n,m)
def dot3(a,b,c):
        return np.dot(np.dot(a,b),c)
def dot4(a,b,c,d):
        return np.dot( np.dot(a,b), np.dot(c,d))
def inv(a):
        return np.linalg.inv(a)
def inv2(a,b):
        return np.linalg.inv(np.dot(a,b))
def loss_fr(beta,X,y,lam):
        m=len(y)
        PT=dot2(beta.T,X.T)
        eT= y.T - PT
        e2= dot2(eT,eT.T)
        J=(e2/2/m) + ((lam/2/m)*dot2(beta.T,beta))
        return J
def err2r(beta,X,y):
        m=len(y)
        PT=dot2(beta.T,X.T)
        eT= y.T - PT
        e2= dot2(eT,eT.T)
        return e2/(2*m)
def rigid(X,y,beta):
        m=len(y)
        delta=[]
        betah=[] ### betas' history
        J0=[]
        ERR=[]
        k=1
        d=1
        while d >= epsi:
                Jold=loss_fr(beta,X,y,lam)
                beta = beta - ((1/m)*alpha*(dot2(X.T,( (dot2(X,beta)) - y)))) \
                - (alpha*lam/m)*beta
                Jnew=loss_fr(beta,X,y,lam)

                d=(100*np.abs(Jold-Jnew)/Jold)
                delta.append(d)
                betah.append(beta)
                J0.append(loss_fr(beta,X,y,lam))
                ERR.append(err2r(beta,X,y))
```

```
            k += 1
        return beta, betah, J0, ERR, delta, k
######################################## initial condtion and values
lam=1
alpha=0.01
epsi=0.1
init=(0+np.zeros(n)).reshape(n,1)
IP=np.identity(n)
beta_ridge=dot2(inv((lam*IP)+dot2(X.T,X)),dot2(X.T,y))
rigid=rigid(X,y,init)
######################################## plotting
lost=[i[0][0] for i in rigid[2]]
kk=[i for i in range(rigid[-1])]
fig, ax = plt.subplots()
ax.plot(lost)
ax.set(xlabel='k_#_of_iterations', ylabel='Loss_Function')
fig.savefig("1.2.eps",format="eps")
plt.show()
betasr = [x*0.0 if x < 0.005 else x for x in rigid[0]]
print(n-betasr.count(0))
#################################### running result on the test data
one=np.ones(md-m).reshape((md-m,1))
yts=datats[:,n].reshape((md-m,1))
XXts=datats[:,1:16].reshape((md-m,n-1))
Xts=np.hstack((one,XXts))
SL_TS=err2r(np.asarray(rigid[0]),Xts,yts)
print(SL_TS)
```

CODE 1.3.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Feb  5 18:41:00 2019

@author: shtabari
"""


import numpy as np
import matplotlib.pyplot as plt

data0=np.loadtxt("data.txt",skiprows=19)
######################################## scaling and centring 1.1
colmean=np.min(data0, axis=0)
datac=data0-colmean
data = datac / (np.ptp(datac, axis=0))

datatr, datats=data[:48,:], data[48:,:]
md=np.size(data, axis=0)
m=np.size(datatr, axis=0)
n=np.size(datatr, axis=1)-1


one=np.ones(m).reshape((m,1))
y=datatr[:,n].reshape((m,1))
XX=datatr[:,1:16].reshape((m,n-1))

X=np.hstack((one,XX))
```

```
def dot2(a,b):
        n=np.size(a,axis=0)
        m=np.size(b,axis=1)
        return np.dot(a,b).reshape(n,m)

def dot3(a,b,c):
        return np.dot(np.dot(a,b),c)
def dot4(a,b,c,d):
        return np.dot( np.dot(a,b), np.dot(c,d))
def inv(a):
        return np.linalg.inv(a)
def inv2(a,b):
        return np.linalg.inv(np.dot(a,b))

lam=1
alpha=0.01
epsi=0.1

IP=np.identity(n)
init=(0+np.zeros(n)).reshape(n,1)


def loss_fl(beta,X,y,lam):
        m=len(y)
        PT=dot2(beta.T,X.T)
        eT= y.T - PT
        e2= dot2(eT,eT.T)
        J=(e2/2/m) + ((lam/2/m)*np.sum(np.abs(beta)))
        return J

def err2l(beta,X,y):
        m=len(y)
        PT=dot2(beta.T,X.T)
        eT= y.T - PT
        e2= dot2(eT,eT.T)
        return e2/(2*m)

def dl1(beta):
        dl1=(np.asarray([1 if i >= 0 else -1 for i in beta]))
        return dl1.reshape(np.size(beta),1)


def lasso(X,y,beta):
        m=len(y)
        delta=[]
        betah=[] ### betas' history
        J0=[]
        ERR=[]
        k=1
        d=1
        while d > epsi:
                Jold=loss_fl(beta,X,y,lam)
                beta = beta - ((1/m)*alpha*(dot2(X.T,( (dot2(X,beta)) - y)))) \
                - ((alpha*lam/m/2)*dl1(beta))
                Jnew=loss_fl(beta,X,y,lam)
                d=(100*np.abs(Jold-Jnew)/Jold)
                delta.append(d)
                betah.append(beta)
                J0.append(loss_fl(beta,X,y,lam))
                ERR.append(err2l(beta,X,y))
                k += 1
```

```
        return beta, betah, J0, ERR, delta, k


lasso=lasso(X,y,init)

lost=[i[0][0] for i in lasso[2]]
plt.plot(lost)

betasl = [x*0.0 if x < 0.005 else x for x in lasso[0]]
print(n-betasl.count(0))
####################################
one=np.ones(md-m).reshape((md-m,1))
yts=datats[:,1].reshape((md-m,1))
XXts=datats[:,2::].reshape((md-m,n-1))
Xts=np.hstack((one,XXts))
#####################################

SL_TS=err21(np.asarray(betasl),Xts,yts)
print(SL_TS)
```

Answer 2.1:
We consider that we have n samples at root (D) with n_0 observations from class '0' and n_1 observations for class '1'. Assuming, we split the root at two nodes and if there are n_L and n_R observations at left (D_L) and right (D_R) nodes with n_{0L} observations of type '0' and n_{1L} observations of type '1' with $n_L = n_{0L} + n_{1L}$ at node (D_L). Also, $n_R = n_0R + n_1R$ at node D_R then

$$I(D) = n\frac{2n_0n_1}{n}$$

$$I(D_L) = \frac{2n_{0L}n_{1L}}{n_L}$$

$$I(D_R) = \frac{2n_{0R}n_{1R}}{n_R}$$

and the Gini function G is

$$G = I(D) - I(D_L) - I(D_R)$$

Code:

```
import numpy as np
n = 100
n1 = 60
nL = np.array([50, 30, 80])
n1L = np.array([30, 20, 50])

G = []

def Gini(n,n1,nL,n1L):
    n0L = np.subtract(nL,n1L)
    n0 = np.subtract(n,n1)
    nR = 100 - (nL)
    n1R = n1 - (n1L)
    n0R = nR- n1R
    G= (2*n0*n1)/n - (2*n0L*n1L)/nL - (2*n0R*n1R)/nR
    return G

#
for i in range(3):
    G.append(Gini(n,n1,nL[i],n1L[i]))
print(G)
```

Output:

```
G
# Wine, Running, Pizza
[0.0, 0.38, 0.5]
```

Since we have a higher Gini function for pizza (0.5), so the best split is based on 'Pizza'.
