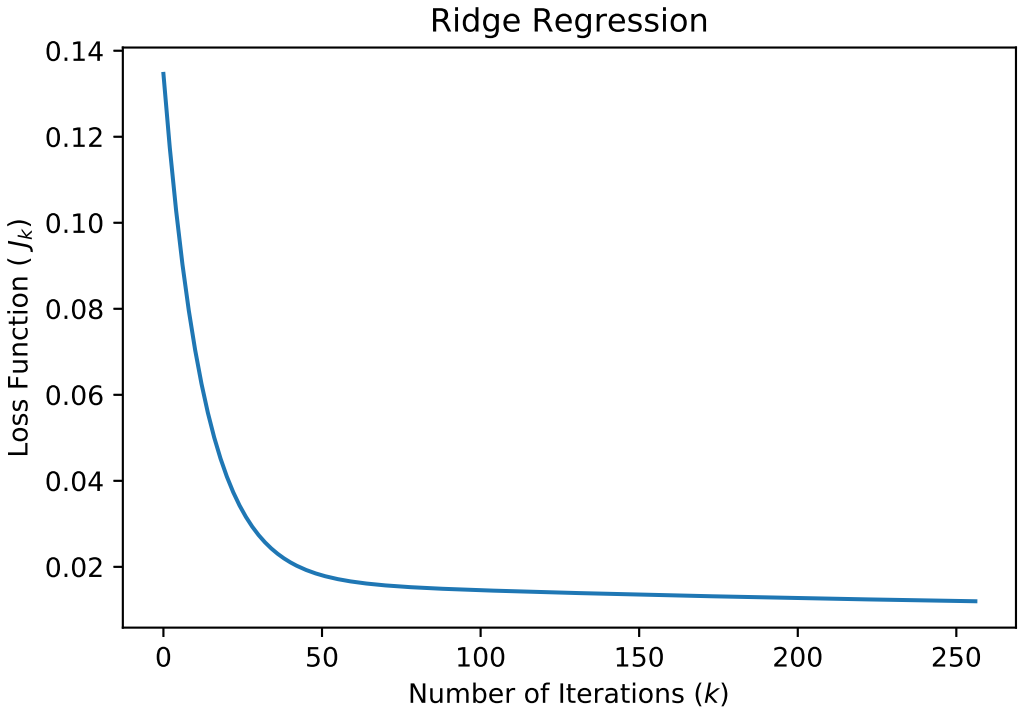
Md Abul Hayat
University of Arkansas

spring 2019
mahayat@uark.edu

Homework # 1 : CSCE 5063

Answer 1.2:
Notes: I have used a `data.txt` file that does not have the comments at the beginning, it only has the dataset required for the regression. I am attaching that file with this report. Please use that file for simulation. I have normalized the data on max-min criterion and have used first 80% of the data as training data and the remaining 20% as testing data. Also, used 5×10^{-3} as a threshold to set θ_i zero for both Ridge and Lasso.
(1) Plot of loss function $J_k(\theta)$ vs. number of iterations (k)



(2) Average sum of squared errors for testing data (Ridge Regression)

$$\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - h_\theta(x^{(i)}))^2 = 0.0315260$$

(3)

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

where,

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

(4) In both cases, (Ridge and Lasso) I have 14 non-zero parameters. Just for comparison, θ for Ridge is

```
[0.12355992, 0.09724431, 0.03219468, 0.0784728 , 0.04689273, 0.10405103, 0.01671621, 0.03515272,
0.059146  , 0.0902391 , 0.03568201, 0.06983985, 0.        , 0.        , 0.06764307, 0.07125605]
```

and the θ for Lasso is

```
[0.12621829, 0.07896546, 0.03132892, 0.0710426 , 0.05078255, 0.08205439, 0.04449737, 0.05242172,
0.03917229, 0.05748263, 0.04313021, 0.04940145, 0.        , 0.        , 0.03641074, 0.06592151]
```

(5) **Code 1.2: (Ridge Regression)**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Take Data As Pandas
df = pd.read_csv('data.txt', sep = '\s+')
D = df.drop('Index', 1).as_matrix() # 60 x 16
n = D.shape[0]      # n = 60
p = D.shape[1]-1    # p = 15

# Normalize
nD = (D - D.min(0))/(D.max(0)-D.min(0))

# Create X and y
X = np.append(np.ones((n,1)),nD[:,0:p], axis = 1)
y = nD[:,p]

# Training Testing Split
Xtr = X[0:48,:]
Xts = X[48:60,:]


ytr = y[0:48]
yts = y[48:60]

ntr = ytr.shape[0]
nts = yts.shape[0]

# Initiate : Gradient Descent
theta = np.zeros(p+1,)
alp = 0.01
lam = 1
eps = 0.1

# Gradient Descent
itr = 0
J = []
J.append(np.dot(ytr,ytr)/(2*ntr))
#
while(1):
    ytrh = np.matmul(Xtr,theta)
    T = ytrh - ytr
    update = np.zeros(p+1)
    for i in range(0,p+1):
        update[i] = np.dot(T,Xtr[:,i]) + lam*theta[i]
    theta = theta - (alp/ntr)*update
    T2 = np.matmul(Xtr,theta)-ytr
    J.append((np.dot(T2,T2)+lam*np.dot(theta,theta))/(2*ntr))
    itr = itr + 1
    dcost = 100*abs((J[itr-1]-J[itr])/J[itr-1])
    if dcost < eps:
        break;

for i in range(0,p+1):
    if abs(theta[i]) < 5e-3:
        theta[i] = 0

# Plot Loss Function
```

```
plt.plot(np.array(J))
plt.xlabel('Number of Iterations ($k$)')
plt.ylabel('Loss Function ( $J_k$)')
plt.title('Ridge Regression')
plt.savefig('F1.eps', format='eps')

# Testing Data
Ts = np.matmul(Xts,theta) - yts
tse = np.dot(Ts,Ts)/nts
np.sum(theta > 0)
```

Code 1.3: (Lasso)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Take Data As Pandas
df = pd.read_csv('data.txt', sep = '\s+')
D = df.drop('Index', 1).as_matrix() # 60 x 16
n = D.shape[0]      # n = 60
p = D.shape[1]-1    # p = 15

# Normalize
nD = (D - D.min(0))/(D.max(0)-D.min(0))

# Create X and y
X = np.append(np.ones((n,1)),nD[:,0:p], axis = 1)
y = nD[:,p]

# Training Testing Split
Xtr = X[0:48,:]
Xts = X[48:60,:]


ytr = y[0:48]
yts = y[48:60]

ntr = ytr.shape[0]
nts = yts.shape[0]

# Initiate : Gradient Descent
theta = np.zeros(p+1,)
alp = 0.01
lam = 1
eps = 0.1

# Gradient Descent
itr = 0
J = []

J.append(np.dot(ytr,ytr)/(2*ntr))
#
while(1):
    ytrh = np.matmul(Xtr,theta)
    T = ytrh - ytr
    update = np.zeros(p+1)
    for i in range(0,p+1):
        if theta[i] >= 0:
            s = 1
        else:
```

```
            s = -1
        update[i] = np.dot(T,Xtr[:,i]) + lam*s/2
    theta = theta - (alp/ntr)*update
    T2 = np.matmul(Xtr,theta)-ytr
    J.append((np.dot(T2,T2)+lam*sum(abs(theta)))/(2*ntr))
    itr = itr + 1
    dcost = 100*abs((J[itr-1]-J[itr])/J[itr-1])
    if dcost < eps:
        break;

for i in range(0,p+1):
    if abs(theta[i]) < 5e-3:
        theta[i] = 0
#
# Plot Loss Function
plt.plot(np.array(J))

# Testing Data
Ts = np.matmul(Xts,theta) - yts
tse = np.dot(Ts,Ts)/nts
np.sum(theta > 0)
```

Answer 2.1:
Assuming, we have n samples at root (D) with n_0 observations from class '0' and n_1 observations for class '1'. If we split the root at two branches 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). Similarly, $n_R = n_{0R} + n_{1R}$ at node D_R then

$$I(D) = \frac{2n_0 n_1}{n}$$

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

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

and the Gini index G is

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

Code:

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

n0L = np.subtract(nL,n1L)
n0 = n - np.asarray(n1)
nR = 100 - np.asarray(nL)
n1R = n1-np.asarray(n1L)
n0R = nR-np.asarray(n1R)

G = []
for i in range(3):
    G.append((2*n0*n1)/n - (2*n0L[i]*n1L[i])/nL[i] - (2*n0R[i]*n1R[i])/nR[i])
```

Output:

```
G
# Wine, Running, Pizza
[0.0, 0.3809523809523796, 0.5]
```

As we have a higher Gini index for pizza (0.5), so the best criteria to split is based on 'Pizza'.
