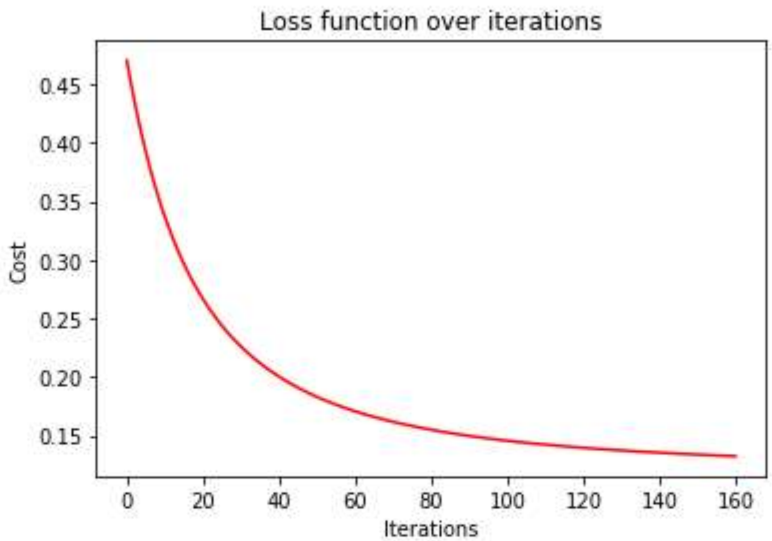
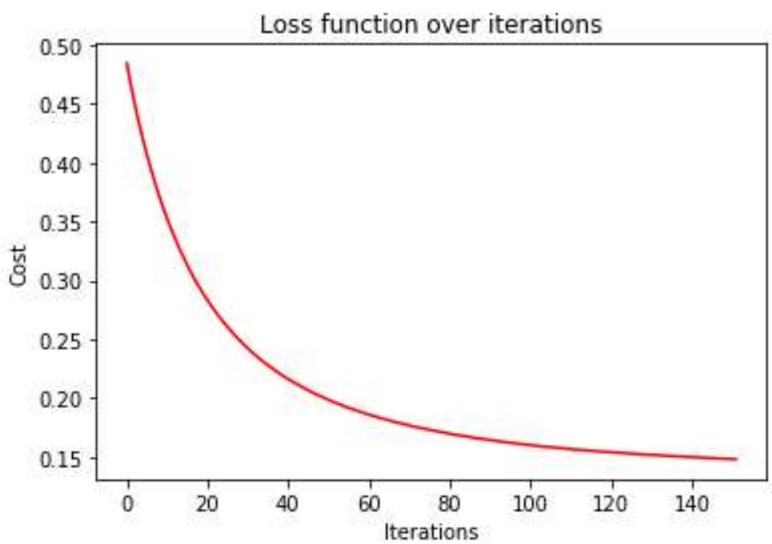
Suhail Basalama - Machine Learning - D. Lu Zhang - HW1

Problem 1

1.
    a. Linear Regression with Quadratic Regularization Plot



    b. Linear Regression with Lasso Regularization Plot



2. The squared loss on the test data for Section 1.2
   Cost/loss function of testing data is 0.22056690771403786

3. Equation for the gradient of Eq. (2)

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_0(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda \theta_j}{2m|\theta_j|}$$

4. Numbers of non-zero parameters of the models obtained in Sections 1.2 and 1.3

15 non-zero parameters for each model

Q Regularization	Lasso Regularization
0	0
0.1609328	0.15220242
-0.08465155	-0.06936921
-0.00978376	-0.01154991
-0.06235717	-0.06260193
0.07976758	0.06784284
-0.15386822	-0.16141882
-0.13316892	-0.13422905
0.25478985	0.24372173
0.28989582	0.28527884
-0.0096616	-0.01133199
0.07824452	0.06831061
-0.03059351	-0.02587364
0.02080199	0.00701408
0.23081007	0.22344558
0.08101115	0.067435

5. Source Code
    a. Linear Regression with Quadratic Regularization Source Code:

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

my_data =
pd.read_csv('raw_training_data.txt',names=["f1","f2","f3","f4","f5","f6","f7"
,"f8","f9","f10","f11","f12","f13","f14","f15","l"]) #read the data
my_data = (my_data - my_data.mean())/my_data.std()

#prepare X matrix
X = my_data.iloc[:,0:15]
ones = np.ones([X.shape[0],1])
X = np.concatenate((ones,X),axis=1)

#prepare y matrix
y = my_data.iloc[:,15:16].values #.values converts it from
pandas.core.frame.DataFrame to numpy.ndarray

#prepare theta matrix
theta = np.zeros([1,16])

#set parameters
alpha = 0.01
epsilon = 0.1
lambd = 1

#cost/loss function
def Cost(X,y,theta, lambd):
  m = len(X)
  #regular format expression
  #sum1 = np.power(((X @ theta.T)-y),2)
  #sum2 = np.power(theta,2)
##############################################################################
  #matrix expression
  J = ((y.T-theta@X.T)@(y.T-theta@X.T).T)/(2*m)
  sum2 = np.power(theta,2)
  #return for regular format
  #return np.sum(sum1)/(2*len(X))+lambd*np.sum(sum2)/(2*len(X))
  #return for matrix format
  return  np.asscalar(J+lambd*np.sum(sum2)/(2*len(X)))

#print the loss function value before training
initialCost = Cost(X,y,theta,lambd)
print("Cost function before training",initialCost)

#gradient function Quadratic Regularization (partial derivative with respect
to thetaj)
def Gradient(X,y,theta,lambd,j):
```

```
  m = len(X)
  Xj = X[:,j]
  Xj = Xj.reshape(len(X),1)
  sum = ((X @ theta.T)-y)*Xj
  return (np.sum(sum)/m)+(lambd*theta[0][j]/m)

#linear Regression with Quadratic Regularization
def LinearRegression(X,y,theta,alpha,epsilon,lambd):
    cost = []
    k = 0
    tempCost = 10
    while(tempCost>epsilon):

        for j in range(len(theta[0])):
          theta[0][j] = theta[0][j] - alpha*Gradient(X,y,theta,lambd,j)

        cost.append(Cost(X, y, theta, lambd))

        if(k != 0):
          tempCost = (abs(cost[k-1]-cost[k])*100)/cost[k-1]

        k += 1

    return theta,cost, k

#running the gd and cost function
g,cost, count = LinearRegression(X,y,theta,alpha,epsilon,lambd)

#print loss/cost function after training
finalCost = Cost(X,y,g,lambd)
print("Cost function after training",finalCost)


def TestingCost(X,y,theta):

    sum1 = np.power(((X @ theta.T)-y),2)

    return np.sum(sum1)/(2 * len(X))

test_data =
pd.read_csv('raw_testing_data.txt',names=["f1","f2","f3","f4","f5","f6","f7",
"f8","f9","f10","f11","f12","f13","f14","f15","l"]) #read the data
test_data = (test_data - test_data.mean())/test_data.std()

X = test_data.iloc[:,0:15]

ones = np.ones([X.shape[0],1])
X = np.concatenate((ones,X),axis=1)

y = test_data.iloc[:,15:16].values #.values converts it from
pandas.core.frame.DataFrame to numpy.ndarray
theta = np.zeros([1,16])


print("Cost function of testing data",TestingCost(X,y,g))

#print trained parameters of theta
```

```
def Rounding(theta):
  for i in range(len(theta[0])):
      if(abs(g[0][i])<0.005): theta[0][i] = 0

#print trained parameters of theta
print(g[0])
Rounding(g)
print(g[0])
#plot the cost
fig, ax = plt.subplots()
ax.plot(np.arange(count), cost, 'r')
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Loss function over iterations')
```

b. Linear Regression with Lasso Regularization Source Code:

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

my_data =
pd.read_csv('raw_training_data.txt',names=["f1","f2","f3","f4","f5","f6","f7"
,"f8","f9","f10","f11","f12","f13","f14","f15","l"]) #read the data
my_data = (my_data - my_data.mean())/my_data.std()

#prepare X matrix
X = my_data.iloc[:,0:15]
ones = np.ones([X.shape[0],1])
X = np.concatenate((ones,X),axis=1)

#prepare y matrix
y = my_data.iloc[:,15:16].values #.values converts it from
pandas.core.frame.DataFrame to numpy.ndarray

#prepare theta matrix
theta = np.zeros([1,16])

#set parameters
alpha = 0.01
epsilon = 0.1
lambd = 1

#cost/loss function
def Cost(X,y,theta, lambd):
  #regular format expression
  m = len(X)
  #sum1 = np.power(((X @ theta.T)-y),2)
  #sum2 = abs(theta)

#############################################################################
  #matrix expression
  J = ((y.T-theta@X.T)@(y.T-theta@X.T).T)/(2*m)
  sum2 = abs(theta)
```

```python
  #return for regular format
  #return np.sum(sum1)/(2*len(X))+lambd*np.sum(sum2)/(2*len(X))
  #return for matrix format
  return  np.asscalar(J+lambd*np.sum(sum2)/(2*len(X)))

#print the loss function value before training
initialCost = Cost(X,y,theta,lambd)
print("Cost function before training",initialCost)

#gradient function Lasso Regularization (partial derivative with respect to
thetaj)
def Gradient(X,y,theta,lambd,j):
  m = len(X)
  Xj = X[:,j]
  Xj = Xj.reshape(len(X),1)
  sum = ((X @ theta.T)-y)*Xj
  if(theta[0][j]==0):
    lasso = 1
  else:
    lasso = (lambd*theta[0][j])/(2*m*abs(theta[0][j]))

  return (np.sum(sum)/m)+lasso
#linear Regression with Quadratic Regularization
def LinearRegression(X,y,theta,alpha,epsilon,lambd):
    cost = []
    k = 0
    tempCost = 10
    while(tempCost>epsilon):

        for j in range(len(theta[0])):
          theta[0][j] = theta[0][j] - alpha*Gradient(X,y,theta,lambd,j)

        cost.append(Cost(X, y, theta, lambd))

        if(k != 0):
          tempCost = (abs(cost[k-1]-cost[k])*100)/cost[k-1]

        k += 1

    return theta,cost, k

#running the gd and cost function
g,cost, count = LinearRegression(X,y,theta,alpha,epsilon,lambd)

#print loss/cost function after training
finalCost = Cost(X,y,g,lambd)
print("Cost function after training",finalCost)


def TestingCost(X,y,theta):

    sum1 = np.power(((X @ theta.T)-y),2)

    return np.sum(sum1)/(2 * len(X))
```

```python
test_data = 
pd.read_csv('raw_testing_data.txt',names=["f1","f2","f3","f4","f5","f6","f7",
"f8","f9","f10","f11","f12","f13","f14","f15","l"]) #read the data
test_data = (test_data - test_data.mean())/test_data.std()

X = test_data.iloc[:,0:15]

ones = np.ones([X.shape[0],1])
X = np.concatenate((ones,X),axis=1)

y = test_data.iloc[:,15:16].values #.values converts it from 
pandas.core.frame.DataFrame to numpy.ndarray
theta = np.zeros([1,16])


print("Cost function of testing data",TestingCost(X,y,g))

def Rounding(theta):
  for i in range(len(theta[0])):
      if(abs(g[0][i])<0.005): theta[0][i] = 0

#print trained parameters of theta
print(g[0])
Rounding(g)
print(g[0])
#plot the cost
fig, ax = plt.subplots()
ax.plot(np.arange(count), cost, 'r')
ax.set_xlabel('Iterations')
ax.set_ylabel('Cost')
ax.set_title('Loss function over iterations')
```

raw_training_data.txt

```
36,27,71,8.1,3.34,11.4,81.5,3243,8.8,42.6,11.7,21,15,59,59,921.870
35,23,72,11.1,3.14,11.0,78.8,4281,3.6,50.7,14.4,8,10,39,57,997.875
44,29,74,10.4,3.21,9.8,81.6,4260,0.8,39.4,12.4,6,6,33,54,962.354
47,45,79,6.5,3.41,11.1,77.5,3125,27.1,50.2,20.6,18,8,24,56,982.291
43,35,77,7.6,3.44,9.6,84.6,6441,24.4,43.7,14.3,43,38,206,55,1071.289
53,45,80,7.7,3.45,10.2,66.8,3325,38.5,43.1,25.5,30,32,72,54,1030.380
43,30,74,10.9,3.23,12.1,83.9,4679,3.5,49.2,11.3,21,32,62,56,934.700
45,30,73,9.3,3.29,10.6,86.0,2140,5.3,40.4,10.5,6,4,4,56,899.529
36,24,70,9.0,3.31,10.5,83.2,6582,8.1,42.5,12.6,18,12,37,61,1001.902
36,27,72,9.5,3.36,10.7,79.3,4213,6.7,41.0,13.2,12,7,20,59,912.347
52,42,79,7.7,3.39,9.6,69.2,2302,22.2,41.3,24.2,18,8,27,56,1017.613
33,26,76,8.6,3.20,10.9,83.4,6122,16.3,44.9,10.7,88,63,278,58,1024.885
40,34,77,9.2,3.21,10.2,77.0,4101,13.0,45.7,15.1,26,26,146,57,970.467
35,28,71,8.8,3.29,11.1,86.3,3042,14.7,44.6,11.4,31,21,64,60,985.950
37,31,75,8.0,3.26,11.9,78.4,4259,13.1,49.6,13.9,23,9,15,58,958.839
35,46,85,7.1,3.22,11.8,79.9,1441,14.8,51.2,16.1,1,1,1,54,860.101
36,30,75,7.5,3.35,11.4,81.9,4029,12.4,44.0,12.0,6,4,16,58,936.234
15,30,73,8.2,3.15,12.2,84.2,4824,4.7,53.1,12.7,17,8,28,38,871.766
31,27,74,7.2,3.44,10.8,87.0,4834,15.8,43.5,13.6,52,35,124,59,959.221
30,24,72,6.5,3.53,10.8,79.5,3694,13.1,33.8,12.4,11,4,11,61,941.181
31,45,85,7.3,3.22,11.4,80.7,1844,11.5,48.1,18.5,1,1,1,53,891.708
31,24,72,9.0,3.37,10.9,82.8,3226,5.1,45.2,12.3,5,3,10,61,871.338
42,40,77,6.1,3.45,10.4,71.8,2269,22.7,41.4,19.5,8,3,5,53,971.122
43,27,72,9.0,3.25,11.5,87.1,2909,7.2,51.6,9.5,7,3,10,56,887.466
```

```
46,55,84,5.6,3.35,11.4,79.7,2647,21.0,46.9,17.9,6,5,1,59,952.529
39,29,76,8.7,3.23,11.4,78.6,4412,15.6,46.6,13.2,13,7,33,60,968.665
35,31,81,9.2,3.10,12.0,78.3,3262,12.6,48.6,13.9,7,4,4,55,919.729
43,32,74,10.1,3.38,9.5,79.2,3214,2.9,43.7,12.0,11,7,32,54,844.053
11,53,68,9.2,2.99,12.1,90.6,4700,7.8,48.9,12.3,648,319,130,47,861.833
30,35,71,8.3,3.37,9.9,77.4,4474,13.1,42.6,17.7,38,37,193,57,989.265
50,42,82,7.3,3.49,10.4,72.5,3497,36.7,43.3,26.4,15,10,34,59,1006.490
60,67,82,10.0,2.98,11.5,88.6,4657,13.6,47.3,22.4,3,1,1,60,861.439
30,20,69,8.8,3.26,11.1,85.4,2934,5.8,44.0,9.4,33,23,125,64,929.150
25,12,73,9.2,3.28,12.1,83.1,2095,2.0,51.9,9.8,20,11,26,50,857.622
45,40,80,8.3,3.32,10.1,70.3,2682,21.0,46.1,24.1,17,14,78,56,961.009
46,30,72,10.2,3.16,11.3,83.2,3327,8.8,45.3,12.2,4,3,8,58,923.234
54,54,81,7.4,3.36,9.7,72.8,3172,31.4,45.5,24.2,20,17,1,62,1113.156
42,33,77,9.7,3.03,10.7,83.5,7462,11.3,48.7,12.4,41,26,108,58,994.648
42,32,76,9.1,3.32,10.5,87.5,6092,17.5,45.3,13.2,29,32,161,54,1015.023
36,29,72,9.5,3.32,10.6,77.6,3437,8.1,45.5,13.8,45,59,263,56,991.290
37,38,67,11.3,2.99,12.0,81.5,3387,3.6,50.3,13.5,56,21,44,73,893.991
42,29,72,10.7,3.19,10.1,79.5,3508,2.2,38.3,15.7,6,4,18,56,938.500
41,33,77,11.2,3.08,9.6,79.9,4843,2.7,38.6,14.1,11,11,89,54,946.185
44,39,78,8.2,3.32,11.0,79.9,3768,28.6,49.5,17.5,12,9,48,53,1025.502
32,25,72,10.9,3.21,11.1,82.5,4355,5.0,46.4,10.8,7,4,18,60,874.281
34,32,79,9.3,3.23,9.7,76.8,5160,17.2,45.1,15.3,31,15,68,57,953.560
10,55,70,7.3,3.11,12.1,88.9,3033,5.9,51.0,14.0,144,66,20,61,839.709
18,48,63,9.2,2.92,12.2,87.7,4253,13.7,51.2,12.0,311,171,86,71,911.701
```

raw_testing_data.txt

```
13,49,68,7.0,3.36,12.2,90.7,2702,3.0,51.9,9.7,105,32,3,71,790.733
35,40,64,9.6,3.02,12.2,82.5,3626,5.7,54.3,10.1,20,7,20,72,899.264
45,28,74,10.6,3.21,11.1,82.6,1883,3.4,41.9,12.3,5,4,20,56,904.155
38,24,72,9.8,3.34,11.4,78.0,4923,3.8,50.5,11.1,8,5,25,61,950.672
31,26,73,9.3,3.22,10.7,81.3,3249,9.5,43.9,13.6,11,7,25,59,972.464
40,23,71,11.3,3.28,10.3,73.8,1671,2.5,47.4,13.5,5,2,11,60,912.202
41,37,78,6.2,3.25,12.3,89.5,5308,25.9,59.7,10.3,65,28,102,52,967.803
28,32,81,7.0,3.27,12.1,81.0,3665,7.5,51.6,13.2,4,2,1,54,823.764
45,33,76,7.7,3.39,11.3,82.2,3152,12.1,47.3,10.9,14,11,42,56,1003.502
45,24,70,11.8,3.25,11.1,79.8,3678,1.0,44.8,14.0,7,3,8,56,895.696
42,83,76,9.7,3.22,9.0,76.2,9699,4.8,42.2,14.5,8,8,49,54,911.817
38,28,72,8.9,3.48,10.7,79.8,3451,11.7,37.5,13.0,14,13,39,58,954.442
```

Problem 2

1.

   a. Splitting on Wine Feature:

$$I(D_R) = I(likeWine) = 50\left(1 - \left(\frac{30}{50}\right)^2 - \left(\frac{20}{50}\right)^2\right) = 24$$

$$I(D_L) = I(dislikeWine) = 50\left(1 - \left(\frac{30}{50}\right)^2 - \left(\frac{20}{50}\right)^2\right) = 24$$

$$I(D) = I(likeBeer) = 100\left(1 - \left(\frac{60}{100}\right)^2 - \left(\frac{40}{100}\right)^2\right) = 48$$

$$G = I(D) - \big(I(D_R) + I(D_L)\big)$$

$$wine\ G = 48 - (24 + 24) = 0$$

   b. Splitting on Running Feature:

$$I(D_R) = I(likeRunning) = 30\left(1 - \left(\frac{20}{30}\right)^2 - \left(\frac{10}{30}\right)^2\right) = 13.33$$

$$I(D_L) = I(dislikeRunning) = 70\left(1 - \left(\frac{40}{70}\right)^2 - \left(\frac{30}{70}\right)^2\right) = 34.28$$

$$I(D) = I(likeBeer) = 100\left(1 - \left(\frac{60}{100}\right)^2 - \left(\frac{40}{100}\right)^2\right) = 48$$

$$G = I(D) - \big(I(D_R) + I(D_L)\big)$$

$$running\ G = 48 - (13.33 + 34.28) = 0.39$$

   c. Splitting on Pizza Feature:

$$I(D_R) = I(likePizza) = 80\left(1 - \left(\frac{50}{80}\right)^2 - \left(\frac{30}{80}\right)^2\right) = 37.5$$

$$I(D_L) = I(dislikePizza) = 20\left(1 - \left(\frac{10}{20}\right)^2 - \left(\frac{10}{20}\right)^2\right) = 10$$

$$I(D) = I(likeBeer) = 100\left(1 - \left(\frac{60}{100}\right)^2 - \left(\frac{40}{100}\right)^2\right) = 48$$

$$G = I(D) - \big(I(D_R) + I(D_L)\big)$$

$$running\ G = 48 - (37.5 + 10) = 0.5$$

2. Since the Pizza Attribute gives the highest gain, I should split my data on the pizza feature.