Taylor Tanita
A11216947

COGS 118A: Assignment 4

1 Least Square Parabola

a) According to the slides,

$$W^* = \arg min_W = \arg min_W g(W) = (X \cdot W - Y)^T (X \cdot W - Y)$$

$$g(W) = W^T X^T X W - W^T X^T Y - Y^T X W + Y^T Y$$

$$\frac{dg(W)}{dW} = 2X^T X W - 2X^T Y = 0$$

$$W^* = (X^T X)^{-1} X^T Y$$

And so, the optimal $\theta = (X^T X)^{-1} X^T Y$

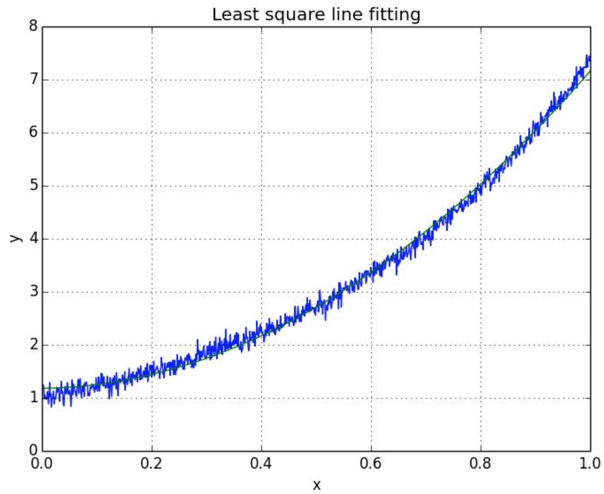
Which is derived in the code:

```
sol = np.dot(np.linalg.inv(np.dot(X_t,X)),np.dot(X_t,y))
```

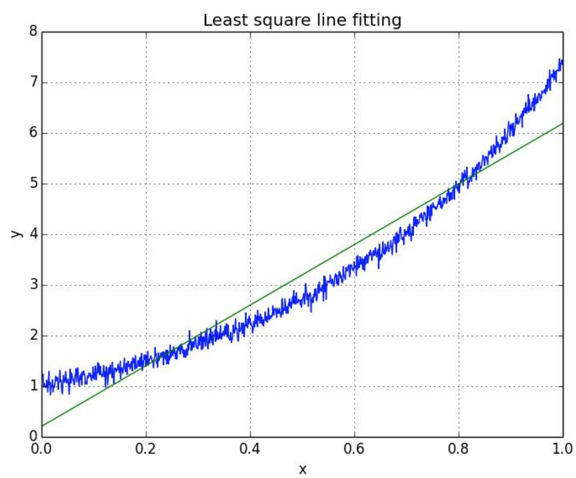
b) **Code: a4_p1.py**

```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
data = sio.loadmat('data.mat')
x = data['x'].reshape([-1, 1])
y = data['y'].reshape([-1, 1])

plt.plot(x,y)
plt.grid()

X = np.hstack((np.ones((len(x),1)),np.power(x,1),np.power(x,2)))
X_t = X.transpose((1,0))
sol = np.dot(np.linalg.inv(np.dot(X_t,X)),np.dot(X_t,y))

plt.hold(True)
plt.plot(x,sol[0]+sol[1]*x+sol[2]*np.power(x,2))

plt.title('Least square line fitting')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('a4_p1.png')
plt.show()
plt.clf()
```

Taylor Tanita
A11216947

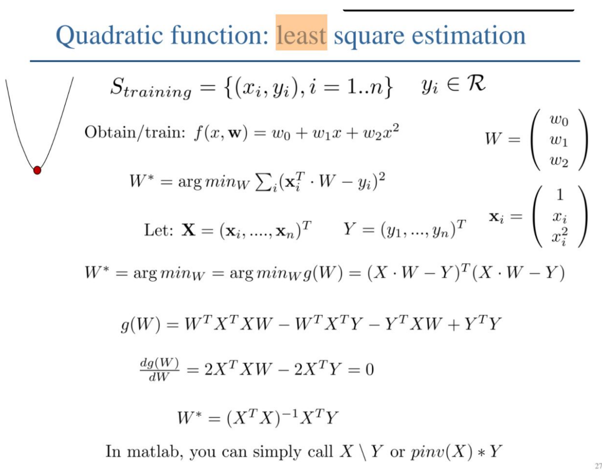


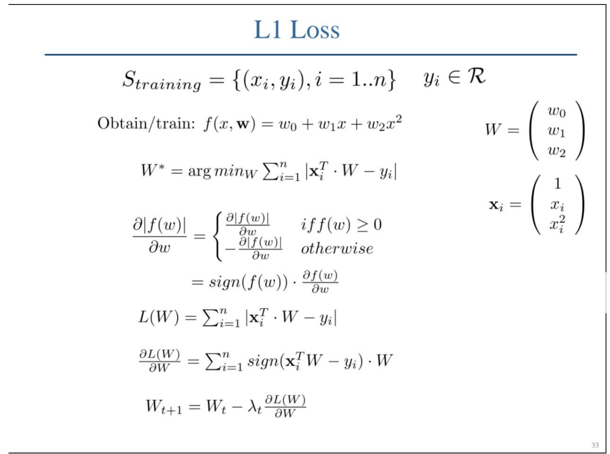
c) A parabola, done in this homework, is more suitable for this data because the data has a curved shape that a line, done in homework 2 cannot account for. The figure outputted in homework 2 is pasted below, and it is clear that the line is way less accurate than the parabola.



Taylor Tanita
A11216947

2 Regression





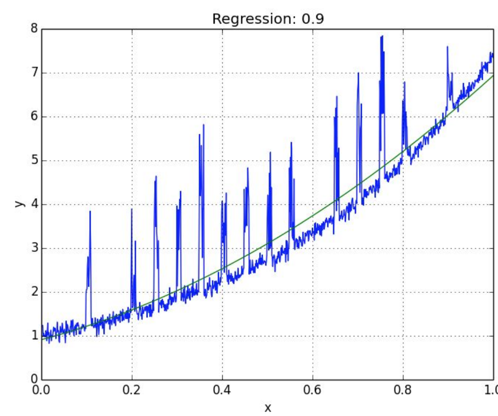
a) Because the loss function is a combination of L2 and L1 loss, we took the derivative as we would individually and the combined them, and applied their respective weights to come up with the derivative

$$\frac{dL(w)}{dL} = \lambda(X^T XW - X^T Y) + (1-\lambda) \sum_{i=1}^{n} sign(x_i^T W - y_i)W$$

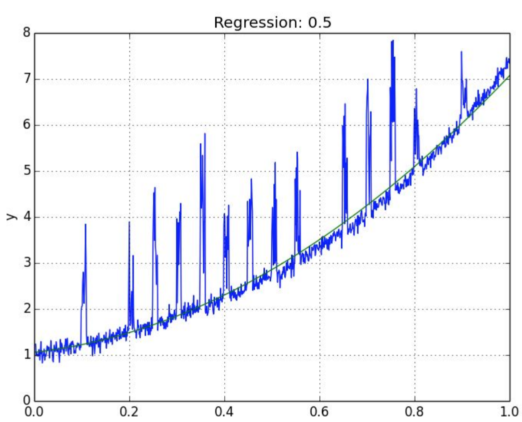
The update formula: $W_{t+1} = W_t - \frac{dL(w)}{dL}$

b) Graphs:

$\lambda = 0.9$

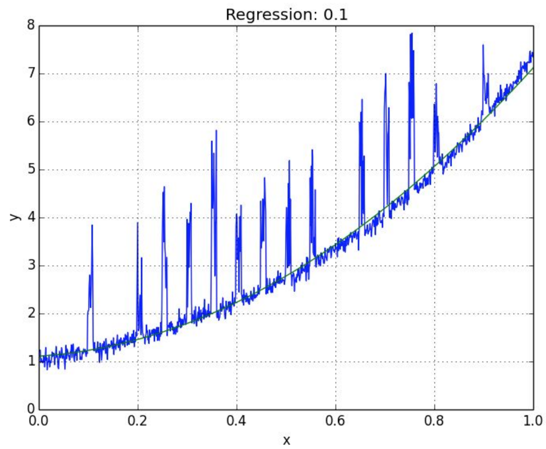


$\lambda = 0.5$



Taylor Tanita
A11216947

$\lambda = 0.1$



Code: a4_p2.py

```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
data = sio.loadmat('modified_data.mat')
x = data['x'].reshape([-1, 1])
y = data['y'].reshape([-1, 1])
X = np.hstack((np.ones((len(x),1)),np.power(x,1),np.power(x,2)))

lamb = 0.0001
converge = 0.001
w = np.array([[0.0],[0.0],[0.0]])
dif = 1

weight = 0.9
while (dif >= converge):
  l2 = np.dot(np.dot(X.transpose(),X),w) - np.dot(X.transpose(), y)
  l2 = l2*weight

  l1 = np.sum(np.sign(np.dot(X,w)-y)*X, axis=0)[np.newaxis].transpose()
  l1 = l1*(1-weight)

  w_old = w
  w = w - lamb*(l2+l1)
  dif = np.sum(np.absolute(w-w_old))

plt.plot(x,y)
plt.grid()
plt.hold(True)
plt.plot(x,w[0]+w[1]*x+w[2]*np.power(x,2))
```

Taylor Tanita
A11216947

```
plt.title('Regression: 0.9')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('a4_p2_0.9.png')
plt.clf()
print('w: (' + str(w[0]) + ', ' + str(w[1]) + ', ' + str(w[2]) + ')')
###
weight = 0.5
w = np.array([[0.0],[0.0],[0.0]])
dif = 1

while (dif >= converge):
  l2 = np.dot(np.dot(X.transpose(),X),w) - np.dot(X.transpose(), y)
  l2 = l2*weight

  l1 = np.sum(np.sign(np.dot(X,w)-y)*X, axis=0)[np.newaxis].transpose()
  l1 = l1*(1-weight)

  w_old = w
  w = w - lamb*(l2+l1)
  dif = np.sum(np.absolute(w-w_old))

plt.plot(x,y)
plt.grid()
plt.hold(True)
plt.plot(x,w[0]+w[1]*x+w[2]*np.power(x,2))
plt.title('Regression: 0.5')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('a4_p2_0.5.png')
plt.clf()
print('w: (' + str(w[0]) + ', ' + str(w[1]) + ', ' + str(w[2]) + ')')
###
weight = 0.1
w = np.array([[0.0],[0.0],[0.0]])
dif = 1

while (dif >= converge):
  l2 = np.dot(np.dot(X.transpose(),X),w) - np.dot(X.transpose(), y)
  l2 = l2*weight

  l1 = np.sum(np.sign(np.dot(X,w)-y)*X, axis=0)[np.newaxis].transpose()
  l1 = l1*(1-weight)
```

```
  w_old = w
  w = w - lamb*(l2+l1)
  dif = np.sum(np.absolute(w-w_old))

plt.plot(x,y)
plt.grid()
plt.hold(True)
plt.plot(x,w[0]+w[1]*x+w[2]*np.power(x,2))
plt.title('Regression: 0.1')
plt.xlabel('x')
plt.ylabel('y')
plt.savefig('a4_p2_0.1.png')
plt.clf()
print('w: (' + str(w[0]) + ', ' + str(w[1]) + ', ' + str(w[2]) + ')')
```

Output:

```
 >>> from a4_p2 import *
w: ([ 0.91179098], [ 2.73098878], [ 3.29051591])
w: ([ 1.05829274], [ 1.18067197], [ 4.83463133])
w: ([ 1.1115787], [ 0.66907575], [ 5.33674609])
```

Taylor Tanita
A11216947

3 Logistic Regression

Decision boundary: $\frac{1}{1+e^{-(0.105689588612-0.729998162069\,x_1+1.24014226544\,x_2)}}$
Optimal parameter: w = (0.105689588612, -0.729998162069, 1.24014226544)
Test error: 3.33333333333%

According to my calculations:

$$\frac{dL(w)}{dL} = [h(x;w)-y]\,(x_1+x_2)$$

$$w_0^{t+1} = w_0^{\,t} - \lambda\sum_i \left(\frac{1}{1+e^{-(w_0^{\,t}+w_1^{\,t}x_1^{\,i}+w_2^{\,t}x_2^{\,i})}}\right)$$

$$w_1^{t+1} = w_1^{\,t} - \lambda\sum_i \left[\left(\frac{1}{1+e^{-(w_0^{\,t}+w_1^{\,t}x_1^{\,i}+w_2^{\,t}x_2^{\,i})}} - y^i\right) x_1^{\,i}\right]$$

$$w_2^{t+1} = w_2^{\,t} - \lambda\sum_i \left[\left(\frac{1}{1+e^{-(w_0^{\,t}+w_1^{\,t}x_1^{\,i}+w_2^{\,t}x_2^{\,i})}} - y^i\right) x_2^{\,i}\right]$$

Code: a4_p3.py

```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import math

train = sio.loadmat('train.mat')
x1 = train['x1'].reshape([-1, 1])
x2 = train['x2'].reshape([-1, 1])
x0 = np.ones((len(x1),1))
y = train['y'].reshape([-1, 1])
X = np.hstack((np.ones((len(x1),1)),x1,x2))

lamb = 0.001
converge = 30
w = np.array([[0.0],[0.0],[0.0]])

def h(x):
  return 1/(1+np.exp(-x))

def classify(x):
  if x >= 0.5:
    return 1.0
  else:
    return 0.0
```

Taylor Tanita
A11216947

```
for i in range(0,100):
  prod = np.dot(X,w)
  h_res = np.apply_along_axis(h,0,prod)
  h_res = h_res - y
  x0_res = np.multiply(h_res,x0)
  x1_res = np.multiply(h_res,x1)
  x2_res = np.multiply(h_res,x2)
  x0_sum = np.sum(x0_res,axis=0)
  x1_sum = np.sum(x1_res,axis=0)
  x2_sum = np.sum(x2_res,axis=0)
  w_old = w
  w[0] = w_old[0] - lamb*x0_sum
  w[1] = w_old[1] - lamb*x1_sum
  w[2] = w_old[2] - lamb*x2_sum
results = np.dot(X,w)
results = np.apply_along_axis(h,0,results)

print('optimal w: ('+str(w[0][0])+', '+str(w[1][0])+', '+str(w[2][0])+')')

#testing
test = sio.loadmat('test.mat')
x1_test = test['x1'].reshape([-1, 1])
x2_test = test['x2'].reshape([-1, 1])
x0_test = np.ones((len(x1),1))
y_test = test['y'].reshape([-1, 1])
X_test = np.hstack((np.ones((len(x1_test),1)),x1_test,x2_test))
results_test = np.dot(X_test,w)
results_test = np.apply_along_axis(h,0,results_test)

s = len(x1_test)
correct = 0.0
total = 0.0
for i in range(0,s):
  if ((results_test[i]>=0.5)&(y_test[i]==1)) | ((results_test[i]<0.5)&(y_test[i]==0)):
    correct = correct + 1.0
  total = total + 1.0

incorrect = total - correct
percent_c = correct/total
percent_i = incorrect/total

print('percent correct: ' + str(percent_c*100) + '%')
print('testing error: ' + str(percent_i*100) + '%')
```

Taylor Tanita
A11216947

Output:

```
>>> from a4_p3 import *
optimal w: (0.105689588612, -0.729998162069, 1.24014226544)
percent correct: 96.6666666667%
testing error: 3.33333333333%
```

Taylor Tanita
A11216947

4 Linear Discriminative Analysis

a. Means of μ_0 and μ_1

μ_0 = (6.00857142857, 2.76857142857)

μ_1 = (5.04571428571, 3.46857142857)

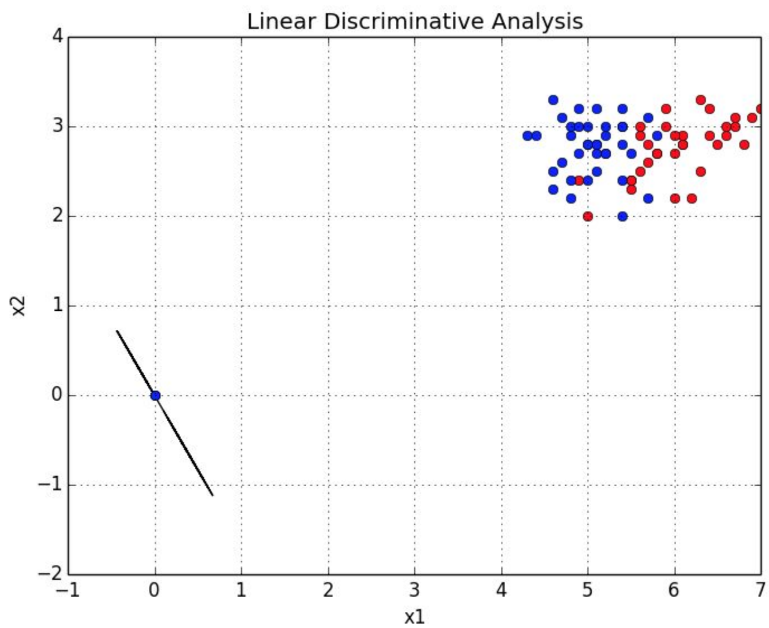
b. Covariance matrix for Σ_0 and Σ_1

```
Σ0 = [[ 1.29771429  0.5721551 ]
      [ 0.5721551   0.31676735]]
Σ1 = [[ 0.85442449  0.59822857]
      [ 0.59822857  0.47630204]]
```

c. Optimal w*

w* = (-0.515174153502, 0.857085521732)

d. Projection



Code: a4_p4.py

```
import scipy.io as sio
import matplotlib.pyplot as plt
import numpy as np
import math

def h(x):
  return 1/(1+np.exp(-x))

train = sio.loadmat('train.mat')
x1 = train['x1'].reshape([-1, 1])
x2 = train['x2'].reshape([-1, 1])
y = train['y'].reshape([-1, 1])
```

Taylor Tanita
A11216947

```
X = np.hstack((np.ones((len(x1),1)),x1,x2))
length = len(x1)
mu0 = np.array([[0.0,0.0]])
mu1 = np.array([[0.0,0.0]])
size0 = 0
size1 = 0
array0 = np.array([[0.0,0.0]])
array1 = np.array([[0.0,0.0]])

for i in range(0,length):
  if (y[i] == 0):
    mu0[0][0] = mu0[0][0] + x1[i]
    mu0[0][1] = mu0[0][1] + x2[i]
    size0 = size0 + 1
    array0 = np.append(array0,np.array([[x1[i][0],x2[i][0]]]),axis=0)
  else:
    mu1[0][0] = mu1[0][0] + x1[i]
    mu1[0][1] = mu1[0][1] + x2[i]
    size1 = size1 + 1
    array1 = np.append(array1,np.array([[x1[i][0],x2[i][0]]]),axis=0)

mu0[0][0] = mu0[0][0]/size0
mu0[0][1] = mu0[0][1]/size0
mu1[0][0] = mu1[0][0]/size1
mu1[0][1] = mu1[0][1]/size1
print('mu0 = (' + str(mu0[0][0]) + ', ' + str(mu0[0][1]) + ')')
print('mu1 = (' + str(mu1[0][0]) + ', ' + str(mu1[0][1]) + ')')

#part b
sigma0 = np.array([[0.0,0.0],[0.0,0.0]])
for i in range(0,size0):
  l = array0[i]-mu0
  prod = np.dot(np.transpose(l),l)
  sigma0 = sigma0 + prod
sigma0 = sigma0/size0

sigma1 = np.array([[0.0,0.0],[0.0,0.0]])
for i in range(0,size1):
  l = array1[i]-mu1
  prod = np.dot(np.transpose(l),l)
  sigma1 = sigma1 + prod
sigma1 = sigma1/size1

print('covariance matrix for sigma0 = ' + str(sigma0))
```

Taylor Tanita
A11216947

```
print('covariance matrix for sigma1 = ' + str(sigma1))

#part c
sw = sigma1 + sigma0
sb = mu1 - mu0
sw_i = np.linalg.inv(sw)
sb_t = sb.transpose()
w_star = np.dot(sw_i,sb_t)
w_den = np.linalg.norm(w_star)
w_star = w_star/w_den

print('w_star: (' + str(w_star[0][0]) + ', ' + str(w_star[1][0]) + ')')

#part d
X = np.hstack((x1,x2))
xx = np.dot(X,w_star)
size = X.size/2
point = np.array([[0.0,0.0]])
for i in range(0,size):
  if i == 0:
    point[0] = w_star.transpose()*xx[i]
  else:
    point = np.append(point, np.array(w_star.transpose()*xx[i]),axis=0)

plt.plot(point[:,0],point[:,1],'k')
plt.plot(array0[:,0],array0[:,1],'ro')
plt.plot(array1[:,0],array0[:,1],'bo')
plt.grid()
plt.hold(True)
plt.title('Linear Discriminative Analysis')
plt.xlabel('x1')
plt.ylabel('x2')
plt.savefig('a4_p4.png')
plt.clf()
```

Output:

```
>>> from a4_p4 import *
mu0 = (6.00857142857, 2.76857142857)
mu1 = (5.04571428571, 3.46857142857)
covariance matrix for sigma0 = [[ 1.29771429  0.5721551 ]
 [ 0.5721551   0.31676735]]
covariance matrix for sigma1 = [[ 0.85442449  0.59822857]
 [ 0.59822857  0.47630204]]
w_star: (-0.515174153502, 0.857085521732)
```

Taylor Tanita
A11216947