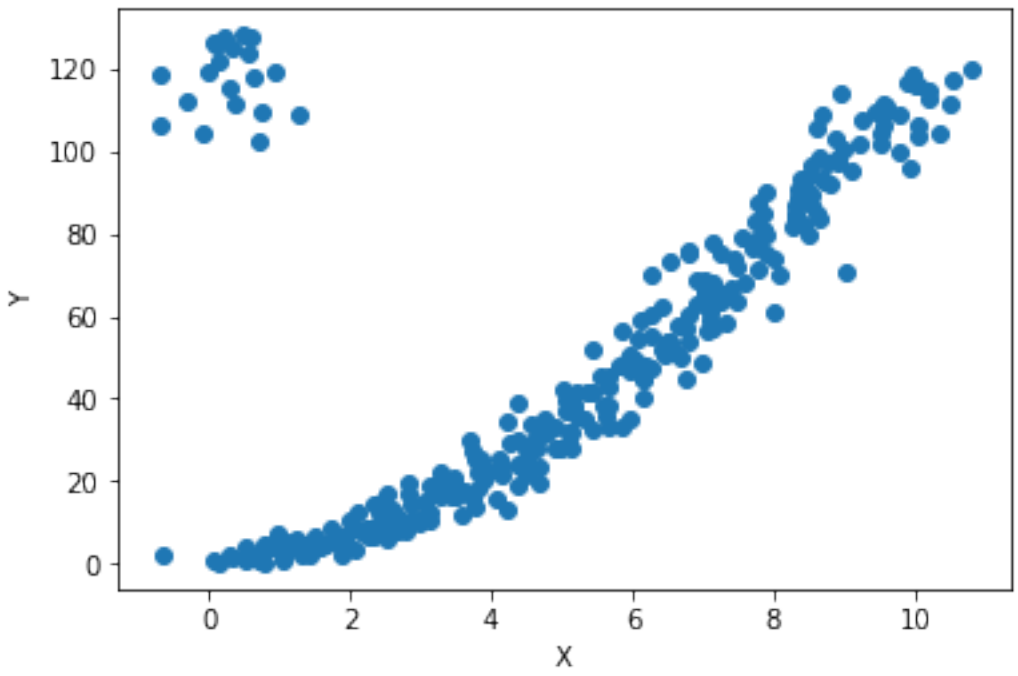
20180213_COGS118a_Hw4

February 14, 2018

1 Parabola Estimation

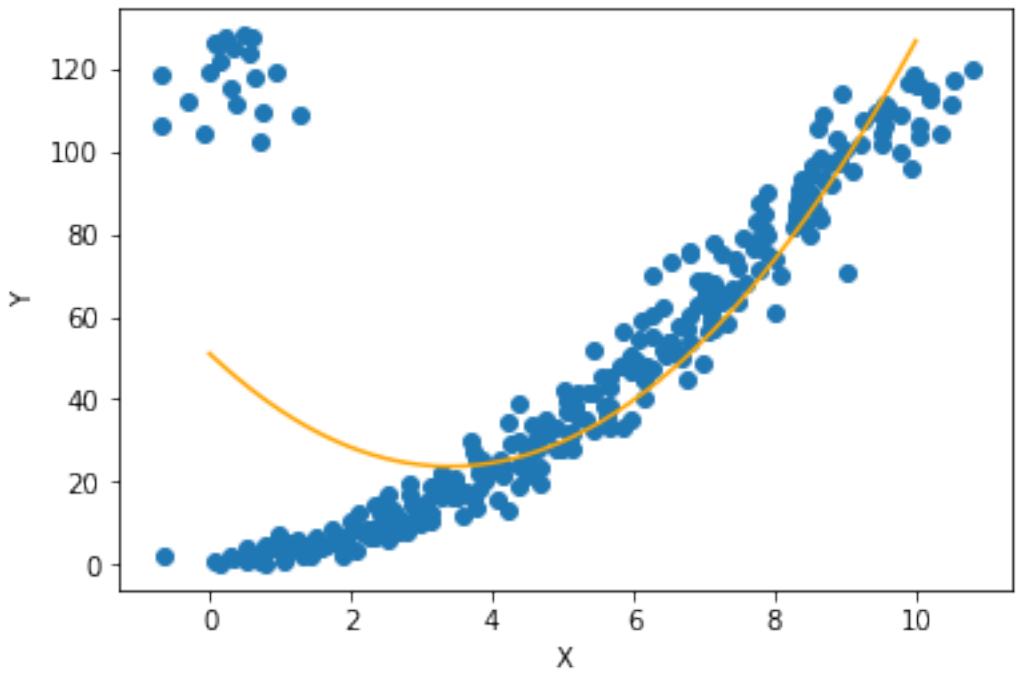
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
X_and_Y = np.load('./hw4-q1-parabola.npy')
X = X_and_Y[:, 0]
Y = X_and_Y[:, 1]

plt.scatter(X, Y)
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



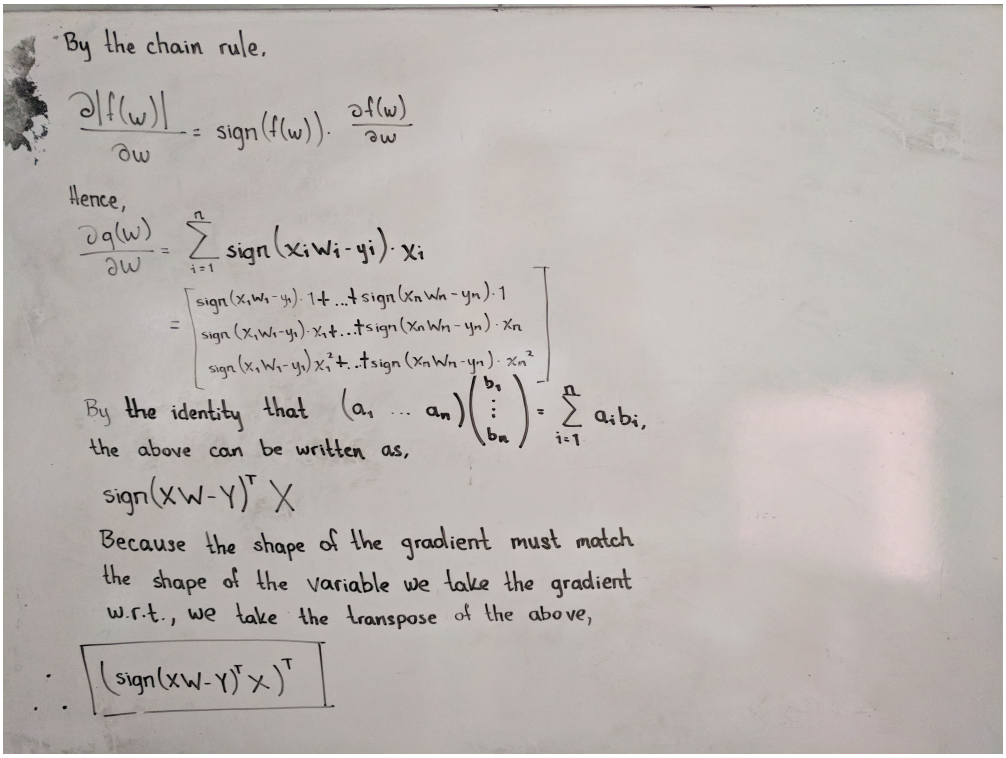
In [12]:
```
X1 = np.matrix(np.hstack((np.ones((len(X),1)),
X.reshape(-1,1))))

X2 = np.matrix(np.hstack((X1, (X**2).reshape(-1,1))))
W = X2.T.dot(X2).I.dot(X2.T).dot(Y)
L2w0, L2w1, L2w2 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X + {:.2f}*X2'.format(w0, w1, w2))
```

```
Y = 50.75 + -15.94*X + 2.35*X2
```

In [13]:
```
X_line = np.linspace(0,10,300)
Y_line = L2w0 + L2w1 * X_line + L2w2 * (X_line**2)
plt.scatter(X, Y)
plt.plot(X_line, Y_line, color='orange')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



In [22]:
```
# g'(W)
def g_prime_W(X, Y, W):
    return (np.sign(X.dot(W) - Y).T.dot(X)).T

W = np.matrix(np.zeros((3,1)))
Y = Y.reshape(-1, 1)
```



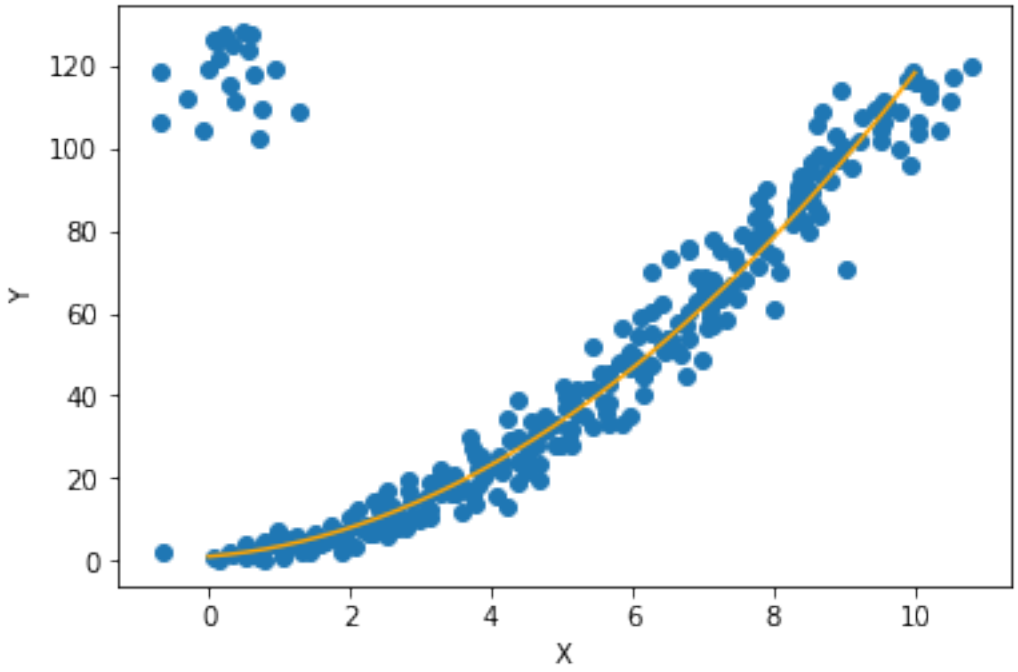
1b

```
# We will keep track of training loss over iterations
iterations = [0]
g_W = [np.absolute(X2.dot(W) - Y)]
for i in range(300000):
    grad = g_prime_W(X2, Y, W)
    W_new = W - 0.000001 * grad
    iterations.append(i+1)
    g_W.append((X2.dot(W_new) - Y).T.dot(X2.dot(W_new) - Y))
    if np.linalg.norm(W_new - W, ord = 1) < 0.00001:
        print("gradient descent terminated after " + str(i) + " iterations")
        break
    W = W_new
L1w0, L1w1, L1w2 = np.array(W).reshape(-1)
print('Y = {:.2f} + {:.2f}*X1 + {:.2f}*X2'.format(L1w0, L1w1, L1w2))
```

```
gradient descent terminated after 41383 iterations
Y = 1.07 + 1.43*X1 + 1.03*X2
```

```
In [23]: X_line = np.linspace(0,10,300)
         Y_line = L1w0 + L1w1 * X_line + L1w2 * (X_line**2)
         plt.scatter(X, Y)
         plt.plot(X_line, Y_line, color='orange')
         plt.xlabel('X')
```

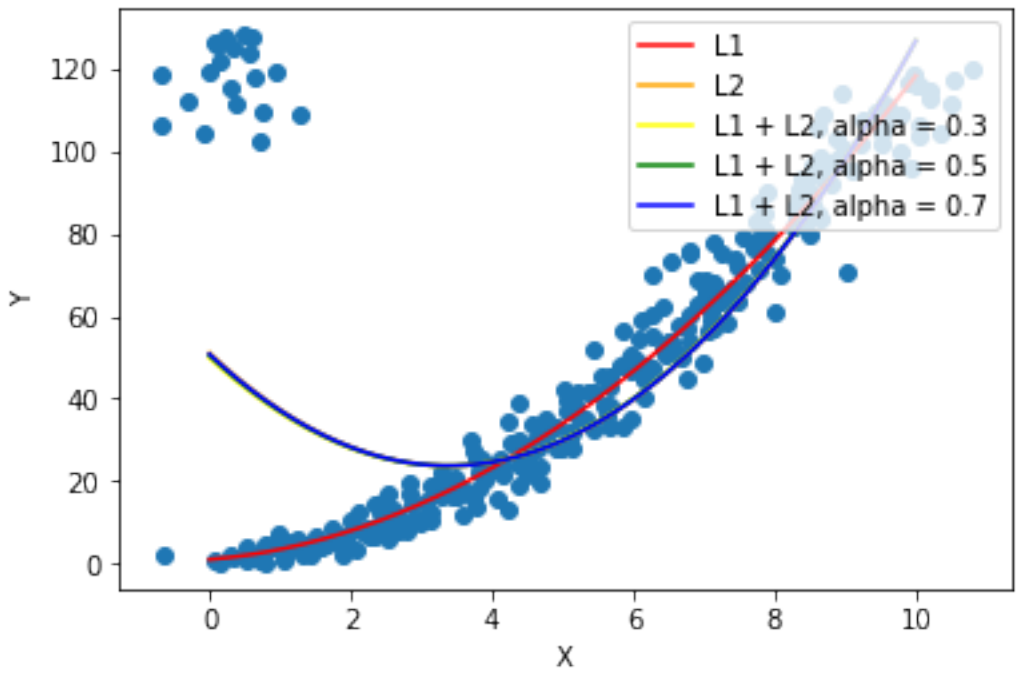
```
plt.ylabel('Y')
plt.show()
```



In [24]:
```
# g'(W)
def g_prime_Walpha(X, Y, W, alpha):
    return alpha * (X.T.dot(2 * ( X.dot(W) - Y))) + (1-alpha) * ((np.sign(X.dot(W) - Y)

alpha_dict = {}
for alpha in [0.3, 0.5, 0.7]:
    W = np.matrix(np.zeros((3,1)))
    Y = Y.reshape(-1, 1)
    # We will keep track of training loss over iterations
    iterations = [0]
    g_W = [alpha * ((X2.dot(W) - Y).T.dot(X2.dot(W) - Y)) + (1 - alpha) *np.absolute(X2
    for i in range(300000):
        grad = g_prime_Walpha(X2, Y, W, alpha)
        W_new = W - 0.000001 * grad
        iterations.append(i+1)
        g_W.append((X2.dot(W_new) - Y).T.dot(X2.dot(W_new) - Y))
        if np.linalg.norm(W_new - W, ord = 1) < 0.00001:
            print("gradient descent terminated after " + str(i) + " iterations")
            break
        W = W_new
    alpha_dict[alpha] = np.array(W).reshape(-1)
    print(alpha_dict)
```

```
gradient descent terminated after 236048 iterations
{0.3: array([ 49.80810639, -15.58382493,   2.32721435])}
gradient descent terminated after 157048 iterations
{0.3: array([ 49.80810639, -15.58382493,   2.32721435]), 0.5: array([ 50.46529152, -15.830045  ,
gradient descent terminated after 119444 iterations
{0.3: array([ 49.80810639, -15.58382493,   2.32721435]), 0.5: array([ 50.46529152, -15.830045  ,
```

In [25]:
```
X_line = np.linspace(0,10,300)
L1Y_line = L1w0 + L1w1 * X_line + L1w2 * (X_line**2)
L2Y_line = L2w0 + L2w1 * X_line + L2w2 * (X_line**2)
alpha3Y_line = alpha_dict[.3][0] + alpha_dict[.3][1] * X_line + alpha_dict[.3][2] * (X_
alpha5Y_line = alpha_dict[.5][0] + alpha_dict[.5][1] * X_line + alpha_dict[.5][2] * (X_
alpha7Y_line = alpha_dict[.7][0] + alpha_dict[.7][1] * X_line + alpha_dict[.7][2] * (X_


plt.scatter(X, Y)

plt.plot(X_line, L1Y_line, color='red', label='L1')
plt.plot(X_line, L2Y_line, color='orange', label='L2')
plt.plot(X_line, alpha3Y_line, color='yellow', label='L1 + L2, alpha = 0.3')
plt.plot(X_line, alpha5Y_line, color='green', label='L1 + L2, alpha = 0.5')
plt.plot(X_line, alpha7Y_line, color='blue', label='L1 + L2, alpha = 0.7')

plt.xlabel('X')
plt.ylabel('Y')
plt.legend(loc = 'upper right')
plt.show()
```



1. The reason that the L1 loss function is a better fit to the data than the L2 loss function is because the L2 loss function is much more sensitive to outliers, which can be seen in the upper left quadrant of the graph.
2. The L2 curve appears to have a sharper curve than the L1 curve because of the outliers.
3. The L2 curve is similiar to the L1 + L2 curves because the square term in the L1 + L2 equations dominate the behavior of the graph, hence they will look similiar to the L2 curve.

hw4-q2-logistic

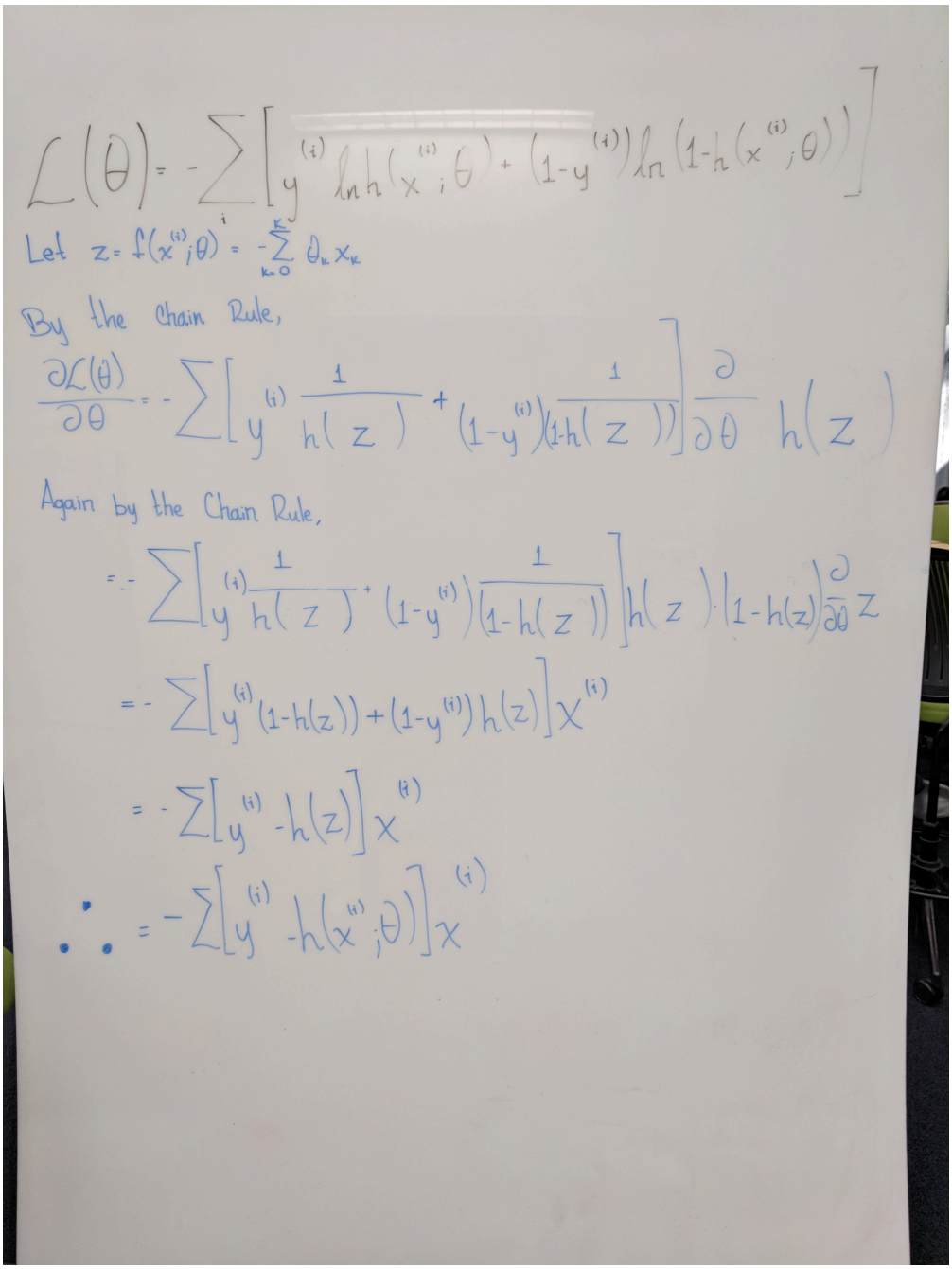
February 14, 2018

0.1 Q2 Logistic Regression

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        from mpl_toolkits.mplot3d import Axes3D
        import math
        %config InlineBackend.figure_format = 'retina'
        %matplotlib inline
        from sklearn.utils import shuffle
        import scipy.io as sio
        plt.rcParams['figure.figsize'] = 8,8
```

0.1.1 Original Data

```
In [2]: X_and_Y_train = np.load('./hw4-q2-logistic-train.npy')
        X_train = X_and_Y_train[:, :2]    # Shape: (70,2)
        X_train = np.matrix(np.hstack((np.ones((len(X_train), 1)), X_train))) # Shape: (70, 3)
        Y_train = X_and_Y_train[:, 2]    # Shape: (70,)

In [3]: X_and_Y_test = np.load('./hw4-q2-logistic-test.npy')
        X_test = X_and_Y_test[:, :2]    # Shape: (30,2)
        X_test = np.matrix(np.hstack((np.ones((len(X_test), 1)), X_test))) # Shape: (30, 3)
        Y_test = X_and_Y_test[:, 2]    # Shape: (70,)

In [4]: mpl.style.use('seaborn')

        fig = plt.figure()
        plt.scatter([X_train[Y_train==0, 1]], [X_train[Y_train==0, 2]], marker='x', color='b', a
        plt.scatter([X_train[Y_train==1, 1]], [X_train[Y_train==1, 2]], marker='o', color='r', a
        plt.xlabel('X1')
        plt.ylabel('X2')
        plt.legend(loc='upper right', fontsize=10)
        plt.title('Training data')
        plt.show()
        #fig.savefig('scatter_1.png', format='png', dpi=400)
```


$L(\theta) = -\sum_i \left[y^{(i)} \ln h(x^{(i)};\theta) + (1-y^{(i)}) \ln (1-h(x^{(i)};\theta)) \right]$
Let $z = f(x^{(i)};\theta) = -\sum_{k=0}^{K} \theta_k x_k$
By the Chain Rule,
$\frac{\partial L(\theta)}{\partial \theta} = -\sum \left[y^{(i)} \frac{1}{h(z)} + (1-y^{(i)}) \frac{1}{(1-h(z))} \right] \frac{\partial}{\partial \theta} h(z)$
Again by the Chain Rule,
$= -\sum \left[y^{(i)} \frac{1}{h(z)} + (1-y^{(i)}) \frac{1}{(1-h(z))} \right] h(z)(1-h(z)) \frac{\partial}{\partial \theta} z$
$= -\sum \left[y^{(i)}(1-h(z)) + (1-y^{(i)}) h(z) \right] x^{(i)}$
$= -\sum \left[y^{(i)} - h(z) \right] x^{(i)}$
$\therefore = -\sum \left[y^{(i)} - h(x^{(i)};\theta) \right] x^{(i)}$

2

0.1.2 Gradient Descent

```
In [5]: def sigmoid(z):
            return 1.0/(1.0+np.exp(-z))

In [6]: # gradient of loss function L(theta)
        def L_prime_theta(X, Y, theta):
            Y = Y.reshape(-1, 1)
            retVal = -((Y - sigmoid(X.dot(theta))).T.dot(X)).T
            return retVal

In [7]: def L_theta(X, Y, theta):
            return -(Y.dot(np.log(sigmoid(X.dot(theta)))) + (1 - Y).dot(np.log(1 - sigmoid(X.dot

In [8]: learning_rate = 0.001
        n_iter = 10000
        theta = np.zeros((X_train.shape[1], 1))
        # We will keep track of training loss over iterations
        iterations = [0]
        L_theta_list = [L_theta(X_train, Y_train, theta)]
        for i in range(n_iter):
            gradient = L_prime_theta(X_train, Y_train, theta)
```

```
            theta_new = theta - learning_rate * gradient
            iterations.append(i+1)
            L_theta_list.append(L_theta(X_train, Y_train, theta_new))

            if np.linalg.norm(theta_new - theta, ord = 1) < 0.001:
                print("gradient descent has converged after " + str(i) + " iterations")
                break
            theta = theta_new

        print ("theta vector: \n" + str(theta))
```

```
gradient descent has converged after 6365 iterations
theta vector:
[[-11.48322099]
 [  0.97466299]
 [  0.88907048]]
```

Equation of decision boundary corresponding to optimal θ^*:

$$y = \begin{cases} 1 & -11.4832 + 0.9747x_1 + 0.8891x_2 \geq 0 \\ 0 & -11.4832 + 0.9747x_1 + 0.8891x_2 < 0 \end{cases}$$

0.1.3 Training curve

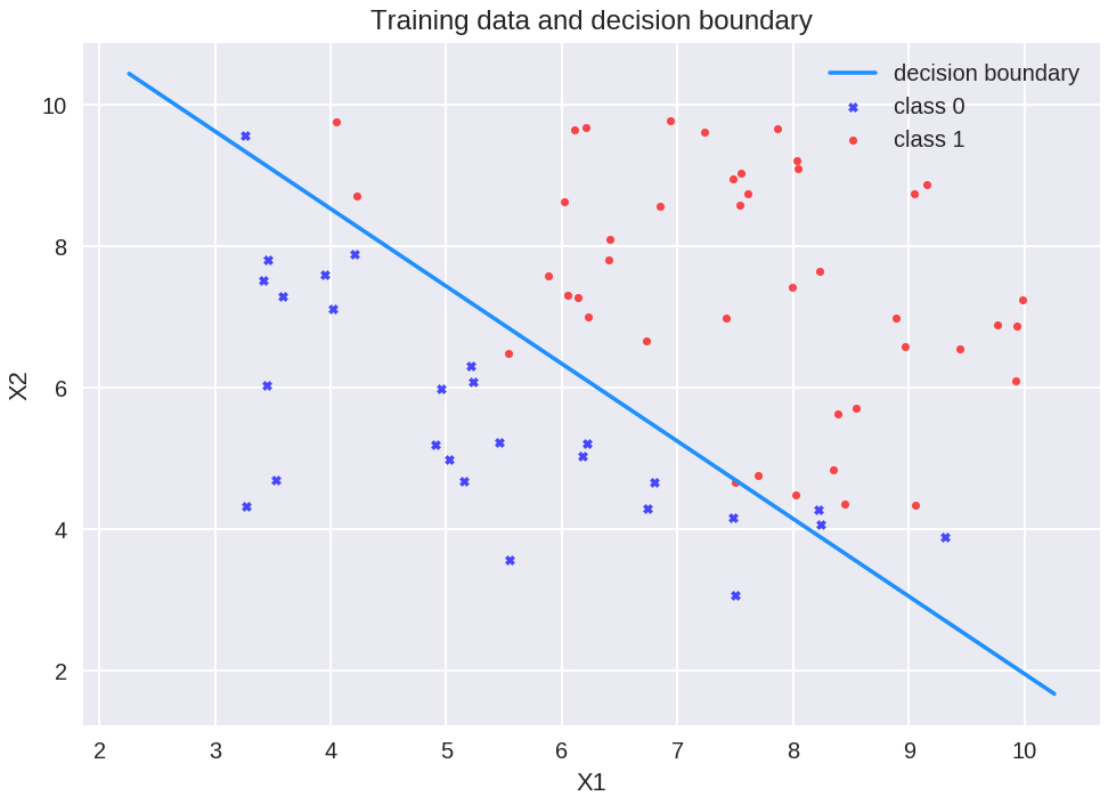
In [9]:
```
plt.title('Training curve')
plt.xlabel('iteration')
plt.ylabel('L(theta)')
plt.semilogy(iterations, np.array(L_theta_list).reshape(-1, 1))
plt.show()
```

0.1.4 Results on Training data

```
In [10]: prediction = sigmoid(np.dot(X_train, theta)) >= 0.5
         testing_accuracy = np.sum(prediction == Y_train.reshape(-1, 1))*1.0/X_train.shape[0]
         print(prediction.shape, Y_test.shape)
         print ("training accuracy: " + str(testing_accuracy))

         x = np.arange(np.min(X_train[:,1])-1,np.max(X_train[:,1])+1,1.0)
         y = (-theta[0][0]-theta[1][0]*x)/theta[2][0]
         plt.scatter([X_train[Y_train==0, 1]], [X_train[Y_train==0, 2]], marker='x', color='b',
         plt.scatter([X_train[Y_train==1, 1]], [X_train[Y_train==1, 2]], marker='o', color='r',


         plt.xlabel('X1')
         plt.ylabel('X2')
         plt.plot(x,y.T, 'dodgerblue', label='decision boundary')
         plt.title('Training data and decision boundary')

         plt.legend(loc='upper right', fontsize=10)
```

```
(70, 1) (30,)
training accuracy: 0.914285714286
```

Out[10]: <matplotlib.legend.Legend at 0x7f512a5ed9e8>

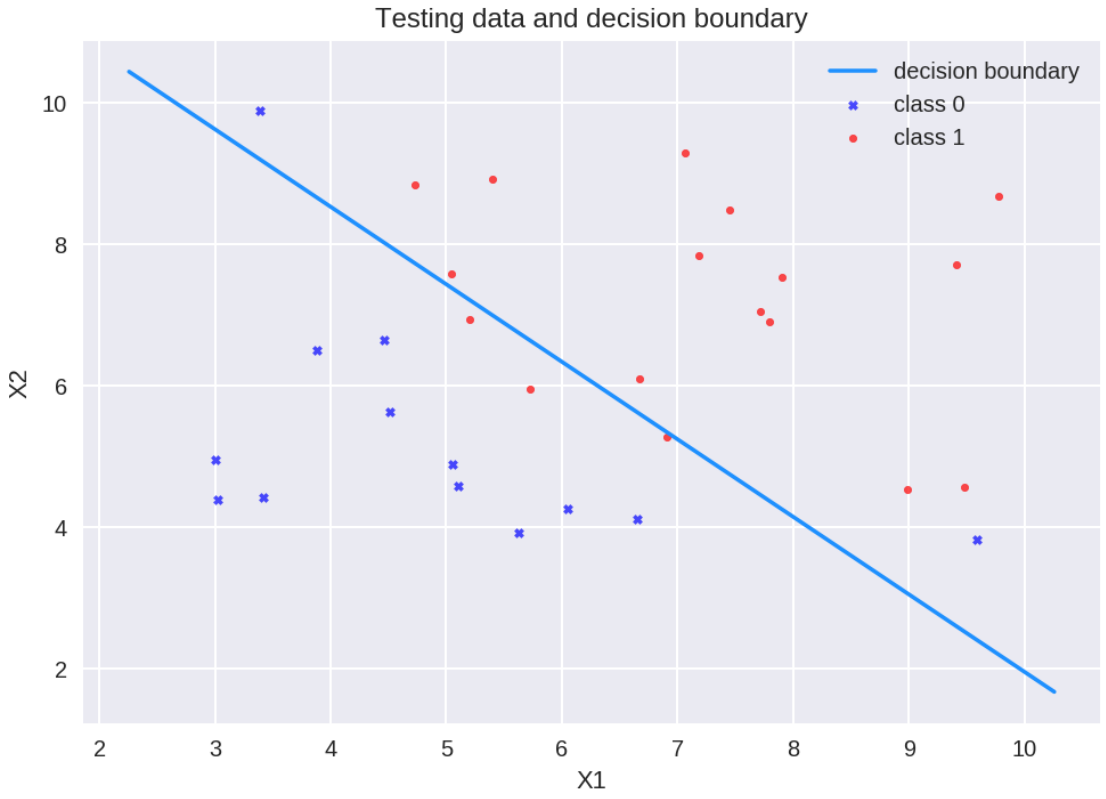


0.1.5 Results on Testing data

```
In [11]: prediction = sigmoid(np.dot(X_test, theta)) >= 0.5
         testing_accuracy = np.sum(prediction == Y_test.reshape(-1, 1))*1.0/X_test.shape[0]
         print ("testing accuracy: " + str(testing_accuracy))

         x = np.arange(np.min(X_train[:,1])-1,np.max(X_train[:,1])+1,1.0)
         y = (-theta[0][0]-theta[1][0]*x)/theta[2][0]
         plt.scatter([X_test[Y_test==0, 1]], [X_test[Y_test==0, 2]], marker='x', color='b', alph
         plt.scatter([X_test[Y_test==1, 1]], [X_test[Y_test==1, 2]], marker='o', color='r', alph


         plt.xlabel('X1')
         plt.ylabel('X2')
         plt.plot(x,y.T, 'dodgerblue', label='decision boundary')
         plt.title('Testing data and decision boundary')

         plt.legend(loc='upper right', fontsize=10)
```

```
testing accuracy: 0.833333333333
```

Out[11]: <matplotlib.legend.Legend at 0x7f5128d59898>



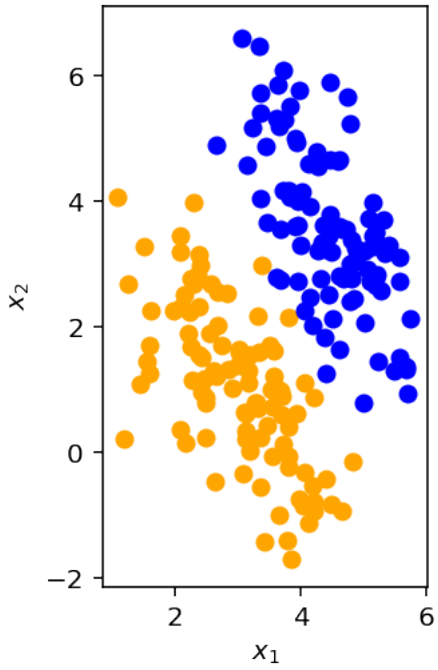
hw4-q3-lda

February 14, 2018

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
```

In [3]:
```
# Load the data and visualize.
Xs = np.load('hw4-q3-lda.npy')

X_0 = np.matrix(Xs[:, 0:2]).T  # Shape: (2, 100).
X_1 = np.matrix(Xs[:, 2:4]).T  # Shape: (2, 100).

print(X_0.shape, X_1.shape)
plt.scatter(X_0[0].tolist(), X_0[1].tolist(), color='orange')
plt.scatter(X_1[0].tolist(), X_1[1].tolist(), color='blue')
plt.axis('scaled')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.show()
```

```
(2, 100) (2, 100)
```




```
In [12]: # (a) Compute mean of each class.
         mu_0 = np.matrix([X_0[0].mean(), X_0[1].mean()]).T # Shape: (2, 1).
         mu_1 = np.matrix([X_1[0].mean(), X_1[1].mean()]).T # Shape: (2, 1).

         print(mu_0.shape, mu_1.shape)
         print('mu_0=\n{},\nmu_1=\n{}'.format(mu_0, mu_1))
```

```
(2, 1) (2, 1)
mu_0=
[[ 2.98351552]
 [ 1.06453902]],
mu_1=
[[ 4.46952033]
 [ 3.52885988]]
```

```
In [14]: # (b) Compute the covariance matrix for each class, Sigma_0 and Sigma_1.
         Sigma_0 = np.cov(X_0) # Shape: (2, 2).
         Sigma_1 = np.cov(X_1) # Shape: (2, 2).

         print(Sigma_0.shape, Sigma_1.shape)
         print('Sigma_0=\n{},\nSigma_1=\n{}'.format(Sigma_0, Sigma_1))
```

```
(2, 2) (2, 2)
Sigma_0=
```

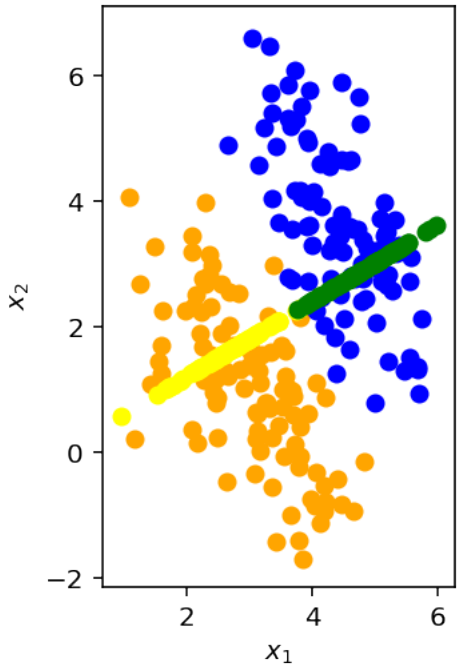
```
[[ 0.70628859 -0.6905174 ]
 [-0.6905174   1.61474336]],
Sigma_1=
[[ 0.48981843 -0.57477756]
 [-0.57477756  1.67666167]]
```

```
In [16]: from numpy import linalg as LA

         # (c) Find the optimal w_star and w_tilde_star with unit length.
         w_star       = LA.inv(Sigma_0 + Sigma_1).dot(mu_0 - mu_1) # Shape: (2, 1).
         w_tilde_star = w_star/LA.norm(w_star, 2) # Shape: (2, 1).

         print(w_star.shape, w_tilde_star.shape)
         print('w_star=\n{},\nw_tilde_star=\n{}'.format(w_star, w_tilde_star))
```

```
(2, 1) (2, 1)
w_star=
[[-3.42871346]
 [-2.06679356]],
w_tilde_star=
[[-0.85643702]
 [-0.51625152]]
```

```
In [19]: # (d) Compute the projection and plot the figure.
         Xproj_0 = w_tilde_star.T.dot(X_0).T.dot(w_tilde_star.T).T # Shape: (2, 100).
         Xproj_1 = w_tilde_star.T.dot(X_1).T.dot(w_tilde_star.T).T # Shape: (2, 100).

         print(Xproj_0.shape, Xproj_1.shape)
         plt.scatter(X_0[0].tolist(), X_0[1].tolist(), color='orange')
         plt.scatter(X_1[0].tolist(), X_1[1].tolist(), color='blue')
         plt.scatter(Xproj_0[0].tolist(), Xproj_0[1].tolist(), color='yellow')
         plt.scatter(Xproj_1[0].tolist(), Xproj_1[1].tolist(), color='green')
         plt.axis('scaled')
         plt.xlabel('$x_1$')
         plt.ylabel('$x_2$')
         plt.show()
```

```
(2, 100) (2, 100)
```


6
4
2
0
−2
2
4
6
x_2
x_1