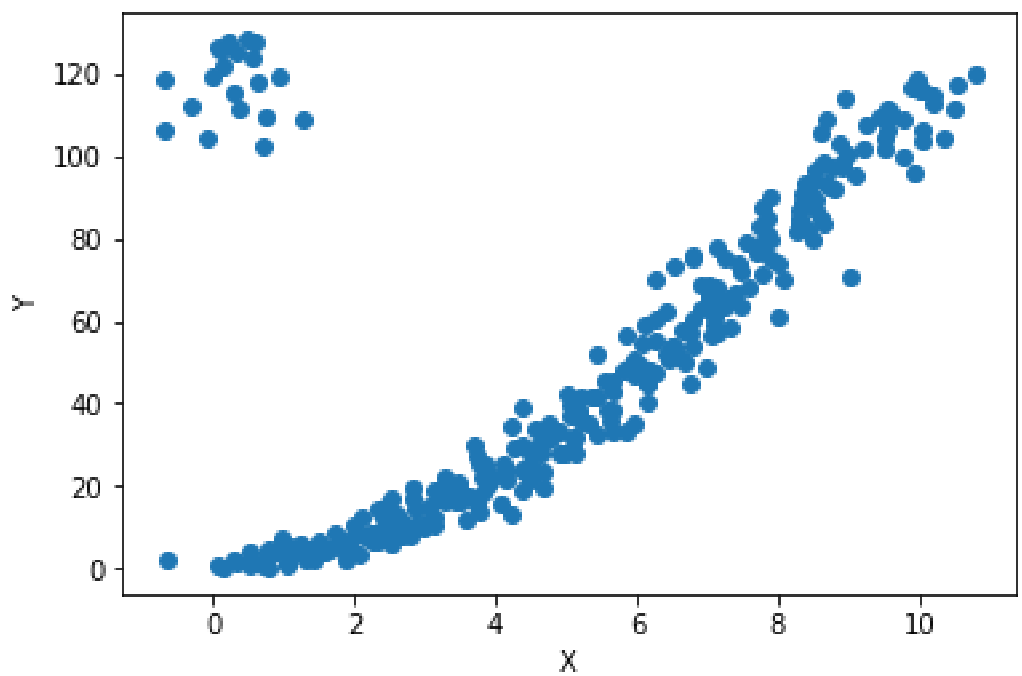
20180213_COGS118a_Hw4

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()
```

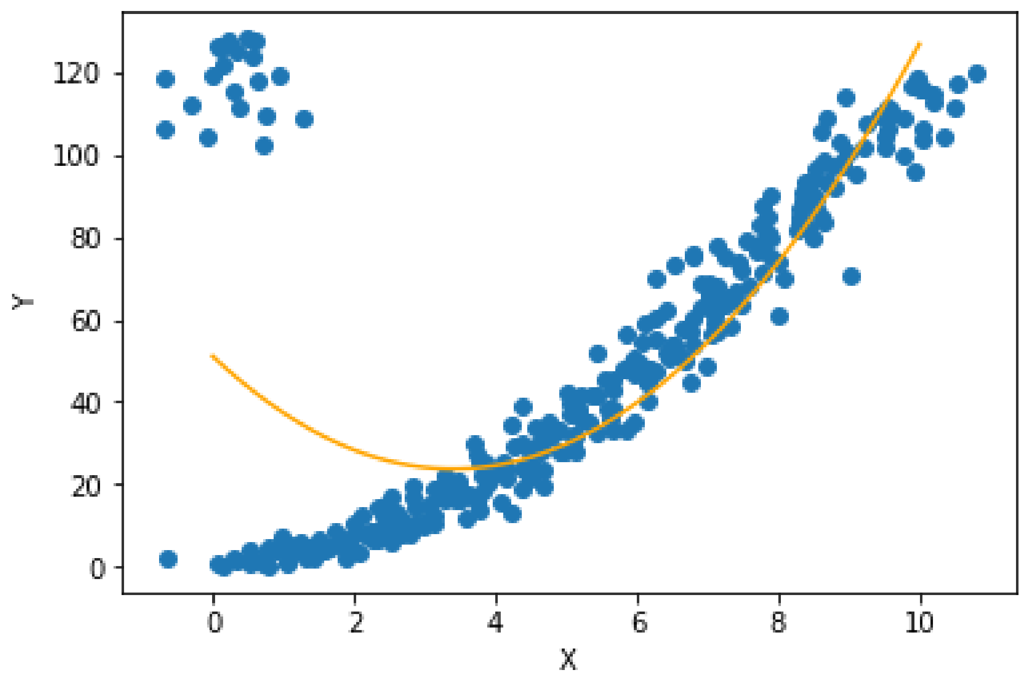


In [12]:
```
X1 = np.matrix(np.hstack((np.ones((len(X),1)),
X.reshape(-1,1))))

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)
```

By the chain rule,

$$\frac{\partial |f(w)|}{\partial w} = \operatorname{sign}(f(w)) \cdot \frac{\partial f(w)}{\partial w}$$

Hence,

$$\frac{\partial g(w)}{\partial w} = \sum_{i=1}^{n} \operatorname{sign}(x_i w_i - y_i) \cdot x_i$$

$$= \begin{bmatrix} \operatorname{sign}(x_1 w_1 - y_1) \cdot 1 + \dots + \operatorname{sign}(x_n w_n - y_n) \cdot 1 \\ \operatorname{sign}(x_1 w_1 - y_1) \cdot x_1 + \dots + \operatorname{sign}(x_n w_n - y_n) \cdot x_n \\ \operatorname{sign}(x_1 w_1 - y_1) x_1^2 + \dots + \operatorname{sign}(x_n w_n - y_n) \cdot x_n^2 \end{bmatrix}$$

By the identity that $(a_1 \ \dots \ a_n) \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \sum_{i=1}^{n} a_i b_i$,

the above can be written as,

$$\operatorname{sign}(XW - Y)^T X$$

Because the shape of the gradient must match the shape of the variable we take the gradient w.r.t., we take the transpose of the above,

$$\therefore \boxed{\left(\operatorname{sign}(XW - Y)^T X\right)^T}$$

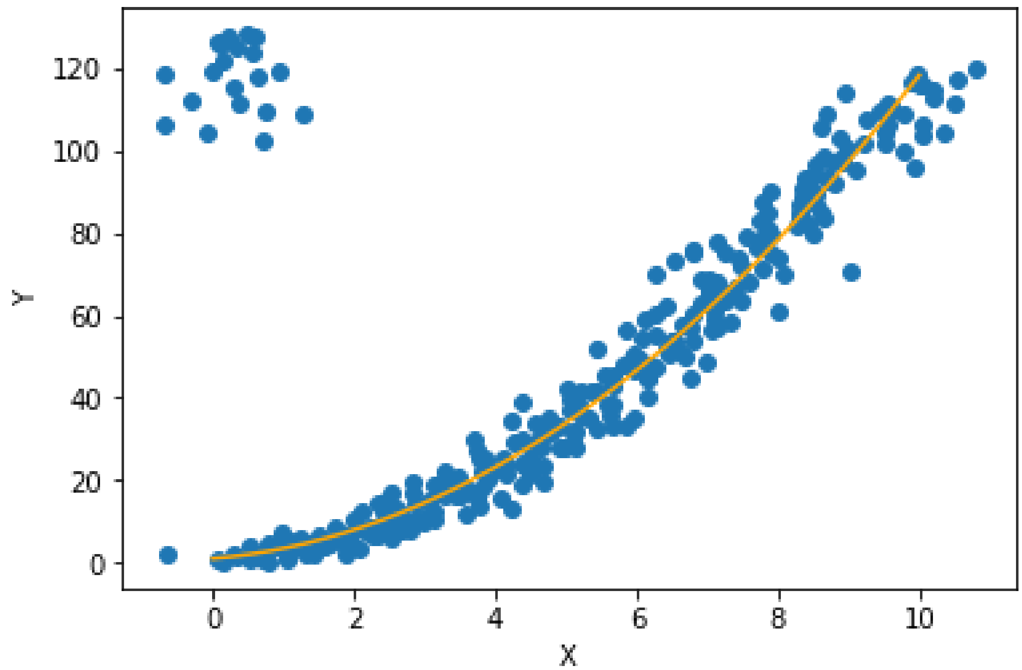
1b

```
# 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)
```

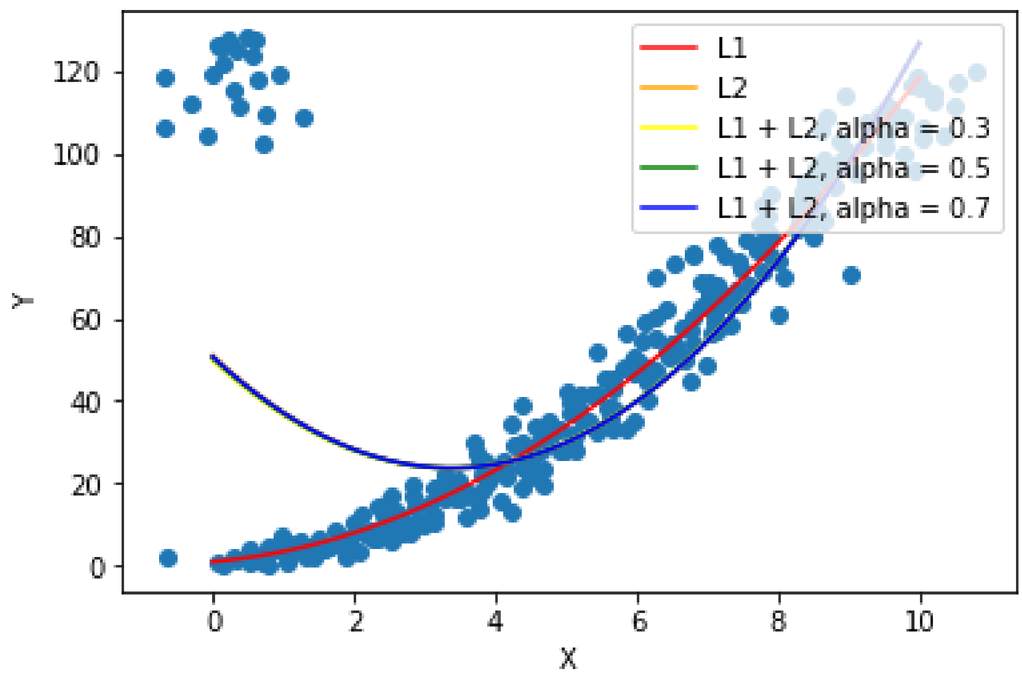
```
gradient descent terminated after 236048 iterations
{0.3: array([ 49.80810639, -15.58382493,   2.32721435])}
gradient descent terminated after 157048 iterations
{0.3: array([ 49.80810639, -15.58382493,   2.32721435]), 0.5: array([ 50.46529152, -15.830045  ,
gradient descent terminated after 119444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