Optional Lab: Cost Function for Logistic Regression

Goals

In this lab, you will:

- examine the implementation and utilize the cost function for logistic regression.

In [1]:
```
import numpy as np
%matplotlib widget
import matplotlib.pyplot as plt
from lab_utils_common import  plot_data, sigmoid, dlc
plt.style.use('./deeplearning.mplstyle')
```

Dataset

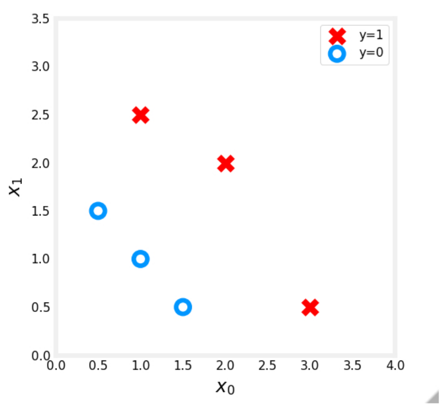
Let's start with the same dataset as was used in the decision boundary lab.

In [2]:
```
X_train = np.array([[0.5, 1.5], [1,1], [1.5, 0.5], [3, 0.5], [2, 2], [1, 2.5]])  #(m,n)
y_train = np.array([0, 0, 0, 1, 1, 1])                                           #(m,)
```

We will use a helper function to plot this data. The data points with label $y = 1$ are shown as red crosses, while the data points with label $y = 0$ are shown as blue circles.

In [3]:
```
fig,ax = plt.subplots(1,1,figsize=(4,4))
plot_data(X_train, y_train, ax)

# Set both axes to be from 0-4
ax.axis([0, 4, 0, 3.5])
ax.set_ylabel('$x_1$', fontsize=12)
ax.set_xlabel('$x_0$', fontsize=12)
plt.show()
```



Cost function

In a previous lab, you developed the *logistic loss* function. Recall, loss is defined to apply to one example. Here you combine the losses to form the **cost**, which includes all the examples.

Recall that for logistic regression, the cost function is of the form

$$J(\mathbf{w},b) = \frac{1}{m} \sum_{i=0}^{m-1} \left[loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) \right] \tag{1}$$

where

- $loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)})$ is the cost for a single data point, which is:

$$loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) = -y^{(i)} \log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)} \right) \right) - \left(1 - y^{(i)}\right) \log \left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)} \right) \right) \tag{2}$$

- where m is the number of training examples in the data set and:

$$\begin{aligned} f_{\mathbf{w},b}(\mathbf{x^{(i)}}) &= g(z^{(i)}) \quad (3) \\ z^{(i)} &= \mathbf{w} \cdot \mathbf{x}^{(i)}+ b \quad (4) \\ g(z^{(i)}) &= \frac{1}{1+e^{-z^{(i)}}} \quad (5) \end{aligned}$$

Code Description

The algorithm for `compute_cost_logistic` loops over all the examples calculating the loss for each example and accumulating the total.

Note that the variables X and y are not scalar values but matrices of shape (m, n) and (m,) respectively, where n is the number of features and m is the number of training examples.

In [4]:
```
def compute_cost_logistic(X, y, w, b):
    """
    Computes cost

    Args:
      X (ndarray (m,n)): Data, m examples with n features
      y (ndarray (m,)) : target values
      w (ndarray (n,)) : model parameters
      b (scalar)       : model parameter

    Returns:
      cost (scalar): cost
    """

    m = X.shape[0]
    cost = 0.0
    for i in range(m):
        z_i = np.dot(X[i],w) + b
        f_wb_i = sigmoid(z_i)
        cost +=  -y[i]*np.log(f_wb_i) - (1-y[i])*np.log(1-f_wb_i)

    cost = cost / m
    return cost
```

Check the implementation of the cost function using the cell below.

In [5]:
```
w_tmp = np.array([1,1])
b_tmp = -3
print(compute_cost_logistic(X_train, y_train, w_tmp, b_tmp))
```

```
0.36686678640551745
```

Expected output: 0.3668667864055175

Example

Now, let's see what the cost function output is for a different value of w.

- In a previous lab, you plotted the decision boundary for $b = -3, w_0 = 1, w_1 = 1$. That is, you had `b = -3, w = np.array([1,1])`.
- Let's say you want to see if $b = -4, w_0 = 1, w_1 = 1$, or `b = -4, w = np.array([1,1])` provides a better model.

Let's first plot the decision boundary for these two different b values to see which one fits the data better.

- For $b = -3, w_0 = 1, w_1 = 1$, we'll plot $-3 + x_0 + x_1 = 0$ (shown in blue)
- For $b = -4, w_0 = 1, w_1 = 1$, we'll plot $-4 + x_0 + x_1 = 0$ (shown in magenta)

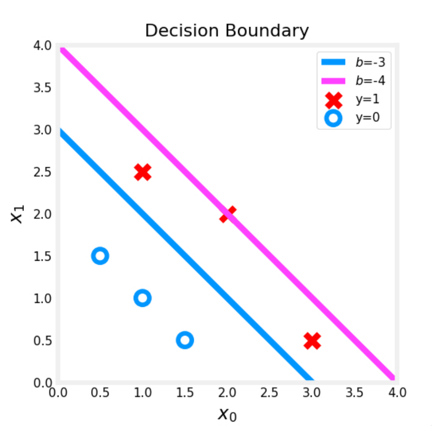
In [6]:
```
import matplotlib.pyplot as plt

# Choose values between 0 and 6
x0 = np.arange(0,6)

# Plot the two decision boundaries
x1 = 3 - x0
x1_other = 4 - x0

fig,ax = plt.subplots(1, 1, figsize=(4,4))
# Plot the decision boundary
ax.plot(x0,x1, c=dlc["dlblue"], label="$b$=-3")
ax.plot(x0,x1_other, c=dlc["dlmagenta"], label="$b$=-4")
ax.axis([0, 4, 0, 4])

# Plot the original data
plot_data(X_train,y_train,ax)
ax.axis([0, 4, 0, 4])
ax.set_ylabel('$x_1$', fontsize=12)
ax.set_xlabel('$x_0$', fontsize=12)
plt.legend(loc="upper right")
plt.title("Decision Boundary")
plt.show()
```



You can see from this plot that `b = -4, w = np.array([1,1])` is a worse model for the training data. Let's see if the cost function implementation reflects this.

In [7]:
```
w_array1 = np.array([1,1])
b_1 = -3
w_array2 = np.array([1,1])
b_2 = -4

print("Cost for b = -3 : ", compute_cost_logistic(X_train, y_train, w_array1, b_1))
print("Cost for b = -4 : ", compute_cost_logistic(X_train, y_train, w_array2, b_2))
```

```
Cost for b = -3 :  0.36686678640551745
Cost for b = -4 :  0.5036808636748461
```

Expected output

Cost for b = -3 : 0.3668667864055175

Cost for b = -4 : 0.5036808636748461

You can see the cost function behaves as expected and the cost for `b = -4, w = np.array([1,1])` is indeed higher than the cost for `b = -3, w = np.array([1,1])`

Congratulations!

In this lab you examined and utilized the cost function for logistic regression.

In [ ]: