Problem 2 (5 points)

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt


def plot_data(data, c, title="", xlabel="$x_1$",ylabel="$x_2$",classes=["",""],alpha=1):
    N = len(c)
    colors = ['royalblue','crimson']
    symbols = ['o','s']

    plt.figure(figsize=(5,5),dpi=120)

    for i in range(2):
        x = data[:,0][c==i]
        y = data[:,1][c==i]

        plt.scatter(x,y,color=colors[i],marker=symbols[i],edgecolor="black",linewidths=0

    plt.legend(loc="upper right")
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    ax = plt.gca()
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    plt.xlim([-0.05,1.05])
    plt.ylim([-0.05,1.05])
    plt.title(title)

def plot_contour(predict, mapXY = None):
    res = 500
    vals = np.linspace(-0.05,1.05,res)
    x,y = np.meshgrid(vals,vals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)
    if mapXY is not None:
        XY = mapXY(XY)
    contour = predict(XY).reshape(res, res)
    plt.contour(x, y, contour)
```

Generate Dataset

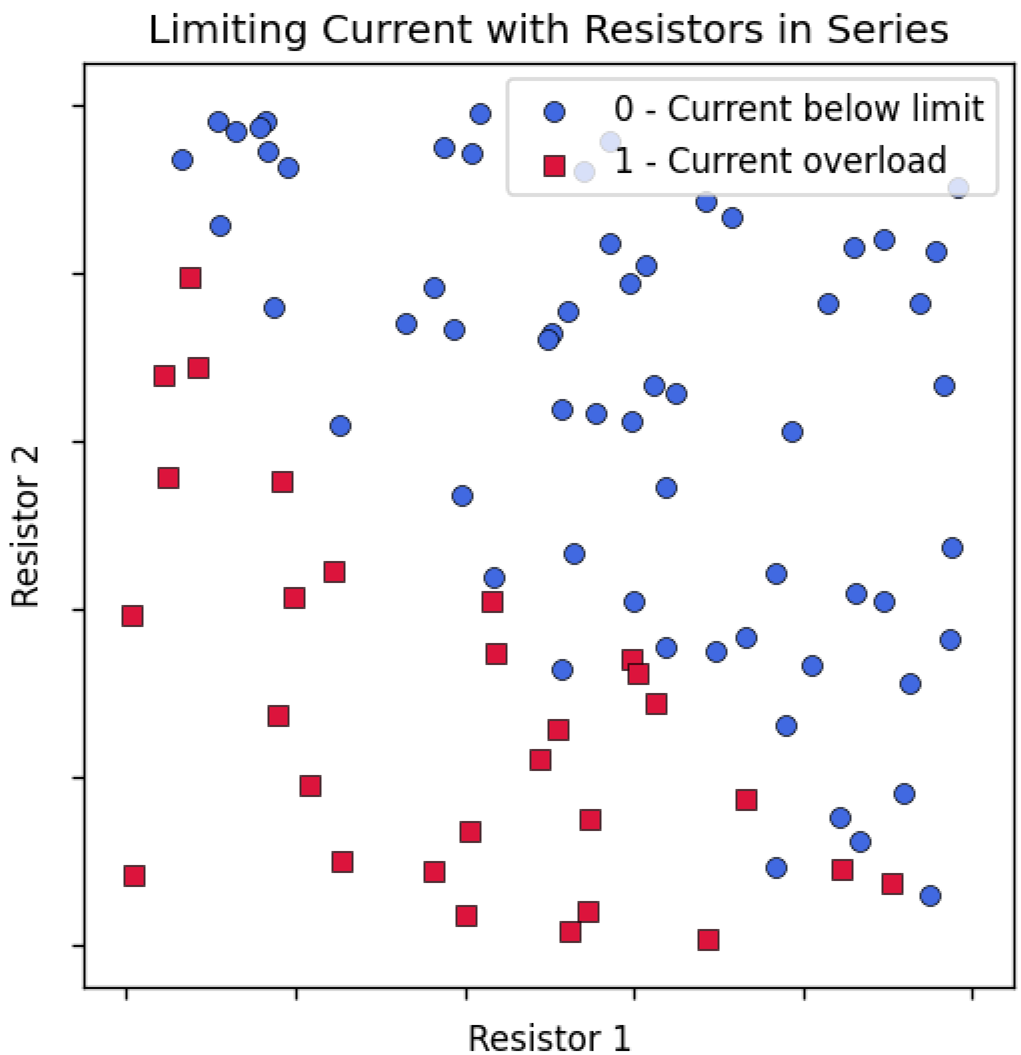
(Don't edit this code.)

In [ ]:
```
def get_line_dataset():
    np.random.seed(4)
    x = np.random.rand(90)
    y = np.random.rand(90)

    h = 1/.9*x + 1/0.9*y - 1

    d = 0.1
    x1, y1 = x[h<-d], y[h<-d]
    x2, y2 = x[np.abs(h)<d], y[np.abs(h)<d]
    x3, y3 = x[h>d], y[h>d]

    c1 = np.ones_like(x1)
```

```
    c2 = (np.random.rand(len(x2)) > 0.5).astype(int)
    c3 = np.zeros_like(x3)
    xs = np.concatenate([x1,x2,x3],0)
    ys = np.concatenate([y1,y2,y3],0)
    c = np.concatenate([c1,c2,c3],0)
    return np.vstack([xs,ys]).T,c
```

In [ ]:

```
data, classes = get_line_dataset()
format = dict(title="Limiting Current with Resistors in Series", xlabel="Resistor 1", yl

plot_data(data, classes, **format)
```



Define helper functions

First, fill in code to complete the following functions. You may use code you wrote in the previous question.

- `sigmoid(h)` to compute the sigmoid of an input `h`
- (Given) `transform(data, w)` to add a column of ones to `data` and then multiply by the 3-element vector `w`
- (Given) `loss(data,y,w)` to compute the logistic regression loss function:

$$L(x,y,w) = \sum_{i=1}^{n} -y^{(i)} \cdot \ln(g(w'x^{(i)})) - (1-y^{(i)}) \cdot \ln(1-g(w'x^{(i)}))$$

- `gradloss(data,y,w)` to compute the gradient of the loss function with respect to w:

$$ \frac{\partial L}{w_j} = \sum_{i=1}^n (g(w'x^{(i)}) - y^{(i)}) x_j^{(i)}

In [ ]:
```
def sigmoid(h):
    return (np.divide(1, (np.add(np.exp(-h), 1))))

def transform(data, w):
    xs = data[:,0]
    ys = data[:,1]
    ones = np.ones_like(xs)
    h = w[0]*ones + w[1]*xs + w[2]*ys
    return h

def loss(data, y, w):
    wt_x = transform(data,w)
    J1 = -np.log(sigmoid(wt_x)) * y
    J2 = -np.log(1-sigmoid(wt_x))*(1-y)
    L = np.sum(J1 + J2)
    return L

def gradloss(data, y, w):
    wt_x = transform(data,w)
    return np.array([np.sum((sigmoid(wt_x) - y) * np.ones(len(data[:,0]))), np.sum((sigm
```

Gradient Descent

Now you'll write a gradient descent loop. Given a number of iterations and a step size, continually update `w` to minimize the loss function. Use the `gradloss` function you wrote to compute a gradient, then move `w` by `stepsize` in the direction opposite the gradient. Return the optimized `w`.

In [ ]:
```
def grad_desc(data, y, w0=np.array([0,0,0]), iterations=100, stepsize=0.1):
    for i in range(iterations):
        w0 = w0 - np.multiply(stepsize,gradloss(data,y,w0))
    return w0
```

Test your classifier

Run these cells to find the optimal `w`, compute the accuracy on the training data, and plot a decision boundary.

In [ ]:
```
w = grad_desc(data,classes)
preds = np.round(sigmoid(transform(data, w))).astype(int)
accuracy = np.sum(preds == classes) / len(classes) * 100
print("           w = ", w)
print("True Classes: ", classes.astype(int))
print(" Predictions: ", preds)
print("    Accuracy: ", accuracy, r"%")
```

```
          w =  [ 7.99449326 -8.54560847 -9.92653181]
True Classes:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1 1 0 1 0 0
0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 Predictions:  [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0
0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
    Accuracy:  91.11111111111111 %
```

In [ ]:

```
predict = lambda data: np.round(sigmoid(transform(data, w)))
plot_data(data, classes, **format)
plot_contour(predict)
plt.show()
```