Table of Contents

1 Exercise 6.1. Implementing sigmoidal Least Squares cost
2 Exercise 6.3. Implementing the Cross Entropy cost
3 Exercise 6.7. Implementing the Softmax cost
4 Exercise 6.8. Implementing the Log Error version of Softmax
5 Exercise 6.9. Using gradient descent to minimize the Perceptron cost
6 Exercise 6.13. Compare the efficacy of two-class cost functions I
7 Exercise 6.14. Compare the efficacy of two-class cost functions II
8 Exercise 6.15. Credit check
9 Exercise 6.16. Weighted classification and balanced accuracy

In [1]:
```
# import autograd-wrapped numpy
import autograd.numpy as np

# datapath to data
datapath = '../mlrefined_datasets/superlearn_datasets/'
```

Exercise 6.1. Implementing sigmoidal Least Squares cost

In [2]:
```
# load in data
csvname = datapath + '2d_classification_data_v1_entropy.csv'
data = np.loadtxt(csvname,delimiter = ',')

# get input/output pairs
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(1, 11)
(1, 11)
```

Exercise 6.3. Implementing the Cross Entropy cost

In [3]:
```
# load in data
csvname = datapath + '2d_classification_data_v1_entropy.csv'
data = np.loadtxt(csvname,delimiter = ',')

# get input/output pairs
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(1, 11)
(1, 11)
```

Exercise 6.7. Implementing the Softmax cost

In [4]:
```
# load in data
csvname = datapath + '2d_classification_data_v1.csv'
data = np.loadtxt(csvname,delimiter = ',')

# take input/output pairs from data
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(1, 11)
(1, 11)
```

Exercise 6.8. Implementing the Log Error version of Softmax

In [5]:
```
# load in dataset
csvname = datapath + '3d_classification_data_v0.csv'
data = np.loadtxt(csvname,delimiter = ',')
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(2, 100)
(1, 100)
```

Exercise 6.9. Using gradient descent to minimize the Perceptron cost

In [6]:
```
# load in dataset
csvname = datapath + '3d_classification_data_v0.csv'
data = np.loadtxt(csvname,delimiter = ',')
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(2, 100)
(1, 100)
```

Exercise 6.13. Compare the efficacy of two-class cost functions I

Below we load in the breast cancer dataset - a description of which you can find here). The input datapoints are stacked *column-wise* in this dataset, with the final row being the label of each point.

In [7]:
```
# data input
csvname = datapath + 'breast_cancer_data.csv'
data = np.loadtxt(csvname,delimiter = ',')

# get input and output of dataset
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(8, 699)
(1, 699)
```

Exercise 6.14. Compare the efficacy of two-class cost functions II

Below we load in a spam email dataset - a description of which you can find here. The input datapoints are stacked *column-wise* in this dataset, with the final row being the label of each point.

In [8]:
```
# data input
csvname = datapath + 'spambase_data.csv'
data = np.loadtxt(csvname,delimiter = ',')

# get input and output of dataset
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(57, 4601)
(1, 4601)
```

Exercise 6.15. Credit check

In [9]:
```
# load in dataset
csvname = datapath + 'credit_dataset.csv'
data = np.loadtxt(csvname,delimiter = ',')
x = data[:-1,:]
y = data[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(20, 1000)
(1, 1000)
```

Exercise 6.16. Weighted classification and balanced accuracy

In [10]:
```
# data input
csvname = datapath + '3d_classification_data_v2_mbalanced.csv'
data1 = np.loadtxt(csvname,delimiter = ',')
```

```
# get input and output of dataset
x = data1[:-1,:]
y = data1[-1:,:]

print(np.shape(x))
print(np.shape(y))
```

```
(2, 55)
(1, 55)
```