An Introduction to Machine Learning

Spencer Hanson, Samuel Reed

HackCU III, University Of Colorado Boulder

April 2017

Integration - Iris Classification

Figure 1: Iris Classification

Load in the Data

```python
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.special import expit
from scipy import optimize

m  = 0;

def loadData(filename):
    content = [];
    #Load Data into list
    with open(filename) as f:
        content = f.readlines()
    content = [x.strip() for x in content]

    num_columns = len(content[1].split(","));
    num_rows = len(content);

    x_data = np.zeros(shape=(num_rows, num_columns-1));
    y_data = np.zeros(shape=(num_rows, 1));


    #Load Data into numpy array
    for i in range(0, len(content)):
        line = content[i].split(",");
        flower = line[num_columns-1];
        if flower == "Iris-setosa":
            y_data[i] = 1;
        elif flower == "Iris-versicolor" or flower == "Iris-virginica":
            y_data[i] = 0;
        for j in range(0, num_columns-1):
            x_val = line[j];
            x_data[i,j] = x_val;

    #Normalize The Data
    for i in range(0, num_columns-1):
        x_min = min(x_data[:,i]);
        x_max = max(x_data[:,i]);
        x_diff = x_max - x_min;
        for j in range(0, num_rows):
            x_data[j,i] = (x_data[j,i]-x_min)/(x_diff);

    return [x_data, y_data];
```

Figure 2: Framework for loading in data - Provided on github

Hypothesis Function

```python
def hypothesisFunc(theta, x):
    hx = expit(np.dot(x, theta));
    return hx;
```

Figure 3: Hypothesis Function

- "expit" is a function from the math library that is identical to the sigmoid function - we will use it for simplicities sake.

Cost Function

```python
def costFunc(theta, x, y):
    hx = x.dot(theta);
    term1 = np.dot(-np.array(y).T, np.log(hypothesisFunc(theta, x)));
    term2 = np.dot((1-np.array(y)).T, np.log(1-hypothesisFunc(theta, x)));
    J = (1./m) * (np.sum(term1 - term2));
    return J;
```

Figure 4: Cost Function

- ".dot" from the math library allows us to do matrix multiplication.

Gradient Descent

```python
def gradDescent(theta, x, y):
    result = optimize.fmin(costFunc, x0=theta, args=(x, y), maxiter=1000, full_output=True);
    return result[0], result[1];
```

Figure 5: Gradient Descent

- "optimize" is a python library function that will minimize our cost function with gradient descent.

Plot Data

```
def plotData(x, y):
    plt.figure(figsize=(10,6));

    pos = np.array([x[i] for i in xrange(m) if y[i][0] == 1]);
    neg = np.array([x[i] for i in xrange(m) if y[i][0] == 0]);


    plt.plot(pos[:,0], pos[:,1], 'g+', label="Iris Setosa");
    plt.plot(neg[:,0], neg[:,1], 'mo', label="Iris Versicolor");
    plt.ylabel("Sepal Width (cm)");
    plt.xlabel("Sepal Length (cm)");
    plt.legend();
    plt.grid(True);
```

Figure 6: Plotting the Data

Decision Boundary

```
def graphBoundary(theta, x, y):

    dim_1 = 1;
    dim_2 = 2;

    bound_x = np.array([np.min(x[:, dim_1]), np.max(x[:,dim_1])]);


    bound_y = (-1./theta[dim_2])*(theta[0] + theta[dim_1]*bound_x);
    plotData(x, y);
    plt.plot(bound_x, bound_y, 'b-', label="Decision Boundary");
    plt.legend();
    plt.show();
```

Figure 7: Decision Boundary

- Plot the line of the decision boundary on our chart.

Main

```python
if __name__ == "__main__":
    filename = "iris_data.csv";
    [x_data, y_data] = loadData(filename);
    (m, n) = x_data.shape;

    theta = np.zeros(shape=(n,1));
    theta, mincost = gradDescent(theta, x_data, y_data);
    graphBoundary(theta, x_data, y_data);
```

Figure 8: Main

- Main framework for calling our functions