Information

$Writer : Junhyuck Woo$
$Std.ID : 20145337$
$Assignment06 : Logistic regression for a binary classification with a non-linear classification boundary$
$Deadline : Apr\ 30,\ 2020$

Library

In [0]:
```
import matplotlib.pyplot as plt; import numpy as np; import collections
```

Data

In [0]:
```
path="/content/drive/My Drive/Spring|2020/Machine_Learning/CAU-MachineLearning/Assignment06/data-nonlinear.txt"
data    = np.genfromtxt(path, delimiter=',')

x  = data[:, 0]
y  = data[:, 1]
label   = data[:, 2]

pointX0 = x[label == 0]
pointY0 = y[label == 0]

pointX1 = x[label == 1]
pointY1 = y[label == 1]

m = label.size
```

Logistic regression

$\hat{h} = \sigma(z)$
$z = g(x,\ y;\ \theta)$, where g is a high dimensional function and $\theta \in \mathbb{R}^k$
$\theta = (\theta_0,\ \theta_1,\ \cdots,\ \theta_{k-1})$
$g(x,\ y;\ \theta) = \theta_0 f_0(x,\ y) + \theta_1 f_1(x,\ y) + \cdots + \theta_{k-1} f_{k-1}(x,\ y)$
$f_k(x,\ y)$ be any high dimensional function of x and y
$\sigma(z) = \frac{1}{1+\exp(-z)}$
$\sigma'(z) = \sigma(z)(1-\sigma(z))$

$$z = \theta_0 + \theta_1 x + \theta_2 y + \theta_3 x^2 + \theta_4 y^2$$

$Initial\ Value$
$\theta_0 = 0.5$
$\theta_1 = 0.5$
$\theta_2 = 0.5$
$\theta_3 = 0.5$
$\theta_4 = 0.5$

In [0]:
```
theta0 = 0.5; theta0_old = 0
theta1 = 0.5; theta1_old = 0
theta2 = 0.5; theta2_old = 0
theta3 = 0.5; theta3_old = 0
theta4 = 0.5; theta4_old = 0
theta5 = 0.5; theta5_old = 0

theta0_history = [theta0]
theta1_history = [theta1]
theta2_history = [theta2]
theta3_history = [theta3]
theta4_history = [theta4]
theta5_history = [theta5]

z = theta0 + theta1*x + theta2*y + theta3*x*y + theta4*x*x + theta5*y*y
sigma = 1/(1 + np.exp(-z))
sigma_ = sigma * (1-sigma)

# Check the accuracy
tmp = (np.round(sigma) == label)
accuracy = collections.Counter(tmp)[1] * 100 / m
accuracy_history = [accuracy]
```

Objective Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-l^{(i)}\log(\sigma(g(x^{(i)},\ y^{(i)};\ \theta))) - (1-l^{(i)})\log(1-\sigma(g(x^{(i)},\ y^{(i)};\ \theta))))$$

In [0]:
```
j = np.sum(-label*np.log(sigma) - (1-label)*np.log(1-sigma)) / m
j_old = 0
j_history = [j]
```

Training Process - Gradient Descent

$\alpha = 0.002$
$$\theta_k^{(t+1)} = \theta_0^{(t)} - \alpha\frac{1}{m}\sum_{i=1}^{m}(\sigma(g(x^{(i)},\ y^{(i)};\ \theta)) - l^{(i)})\frac{\partial g(x^{(i)},\ y^{(i)};\ \theta^{(t)})}{\partial \theta_k},\ \ for\ all\ k$$

In [0]:
```
alpha = 0.002

# Check the number of iteration
iteration = 1
```

```
# Set the two condition because it spent too much time for converge
while  (iteration < 4194304):
    # Calculate the theta
    theta0_old = theta0
    theta1_old = theta1
    theta2_old = theta2
    theta3_old = theta3
    theta4_old = theta4
    theta5_old = theta5

    theta0 = theta0 - alpha*np.sum(sigma-label)/m
    theta1 = theta1 - alpha*np.sum((sigma-label)*x)/m
    theta2 = theta2 - alpha*np.sum((sigma-label)*y)/m
    theta3 = theta3 - alpha*np.sum((sigma-label)*x*y)/m
    theta4 = theta4 - alpha*np.sum((sigma-label)*x*x)/m
    theta5 = theta5 - alpha*np.sum((sigma-label)*y*y)/m

    # Update the j
    j_old = j
    z = theta0 + theta1*x + theta2*y + theta3*x*y + theta4*x*x + theta5*y*y
    sigma = 1/(1 + np.exp(-z))
    j = np.sum(-label*np.log(sigma) - (1-label)*np.log(1-sigma)) / m

    # Check the accuracy
    tmp = (np.round(sigma) == label)
    accuracy = collections.Counter(tmp)[1] * 100 / m

    # Record the history of parameter
    theta0_history.append(theta0)
    theta1_history.append(theta1)
    theta2_history.append(theta2)
    theta3_history.append(theta3)
    theta4_history.append(theta4)
    theta5_history.append(theta5)

    j_history.append(j)
    accuracy_history.append(accuracy)

    iteration = iteration +1
```

Check the Convergence

In [6]:
```
# Iteration
print("# Iteration: " + str(iteration) + '\n')

# Theta 0
print("Updated Theta0: "+ str(theta0))
print("Old Theta0: " + str(theta0_old))
print("Diff: " + str(theta0 - theta0_old) + '\n')

# Theta 1
print("Updated Theta1: "+ str(theta1))
print("Old Theta1: " + str(theta1_old))
print("Diff: " + str(theta1 - theta1_old) + '\n')

# Theta 2
print("Updated Theta2: "+ str(theta2))
print("Old Theta2: " + str(theta2_old))
print("Diff: " + str(theta2 - theta2_old) + '\n')

# Theta 3
print("Updated Theta3: "+ str(theta3))
print("Old Theta3: " + str(theta3_old))
print("Diff: " + str(theta3 - theta3_old) + '\n')

# Theta 4
print("Updated Theta4: "+ str(theta4))
print("Old Theta1: " + str(theta4_old))
print("Diff: " + str(theta4 - theta4_old) + '\n')

# Theta 5
print("Updated Theta1: "+ str(theta5))
print("Old Theta1: " + str(theta5_old))
print("Diff: " + str(theta5 - theta5_old) + '\n')

# J, Energy Value
print("Updated J: "+ str(j))
print("Old J: " + str(j_old))
print("Diff: " + str(j - j_old) + '\n')
```

```
# Iteration: 4194304

Updated Theta0: 5.161041681240372
Old Theta0: 5.161041670421705
Diff: 1.0818666851264425e-08

Updated Theta1: 3.2409782729048837
Old Theta1: 3.24097826445854
Diff: 8.44634362451302e-09

Updated Theta2: 4.158459765379226
Old Theta2: 4.158459755316046
Diff: 1.0063180511110659e-08

Updated Theta3: -7.515206625778852
Old Theta3: -7.515206604996127
Diff: -2.078272487437971e-08

Updated Theta4: -12.007238151978141
Old Theta1: -12.00723812656807
Diff: -2.5410070492171144e-08

Updated Theta1: -11.801811310055912
Old Theta1: -11.80181128328923
Diff: -2.6766683092205312e-08

Updated J: 0.3481057148739871
Old J: 0.3481057148750289
Diff: -1.0417777751570156e-12
```

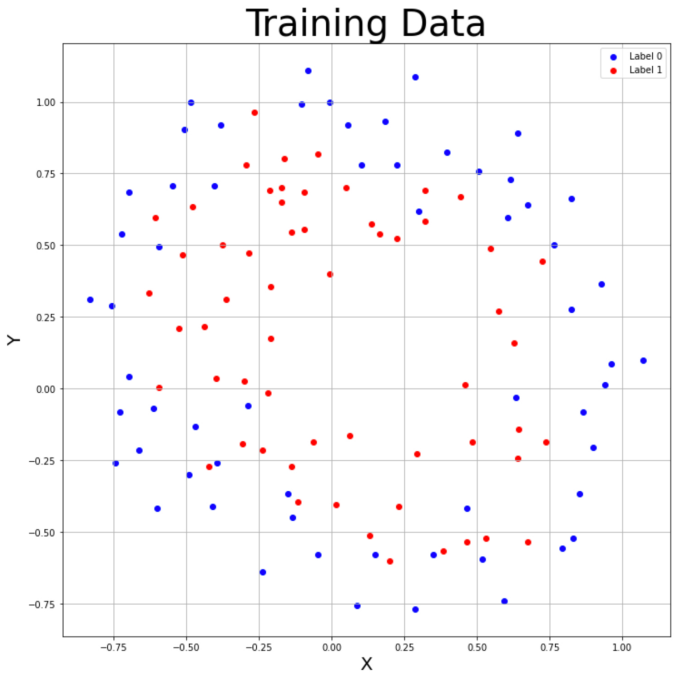
Grid

In [0]:
```
x_range = np.arange(-1, 1.25, 0.01)
y_range = np.arange(-1, 1.25, 0.01)
x_grid, y_grid = np.meshgrid(x_range, y_range)
z_ = theta0 + theta1*x_grid + theta2*y_grid + theta3*x_grid*y_grid + \
theta4*x_grid*x_grid+ theta5*y_grid*y_grid
sig = 1/(1 + np.exp(-z_))
sigma_visual = np.round(sig, 1)
sigma_visual = (sigma_visual==0.5)
sigma_visual = 0.5*np.multiply(sigma_visual, 1)
```

Result

1. Training Data

In [8]:
```
plt.figure(figsize=(12,12))
plt.scatter(pointX0, pointY0, c='b', label='Label 0')
plt.scatter(pointX1, pointY1, c='r', label='Label 1')
plt.grid()
plt.legend()
plt.title('Training Data', fontsize=40)
plt.xlabel('X', fontsize=20)
plt.ylabel('Y', fontsize=20)
plt.show()
```



2. High dimensional function $g(x, y; \theta)$

$g(x, y; \theta) = \theta_0 + \theta_1 x + \theta_2 xy + \theta_3 x^2 + \theta_4 y^2$

3. Training Error

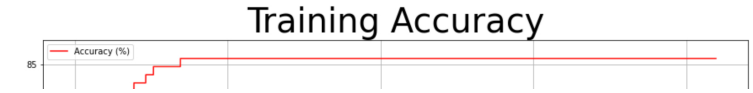
In [9]:
```
plt.figure(figsize=(15,9))
plt.plot(j_history, color='blue', label='Error')
plt.grid()
plt.legend()
plt.title('Training Error', fontsize=40)
plt.xlabel('Iteration', fontsize=20)
plt.ylabel('Error', fontsize=20)
```

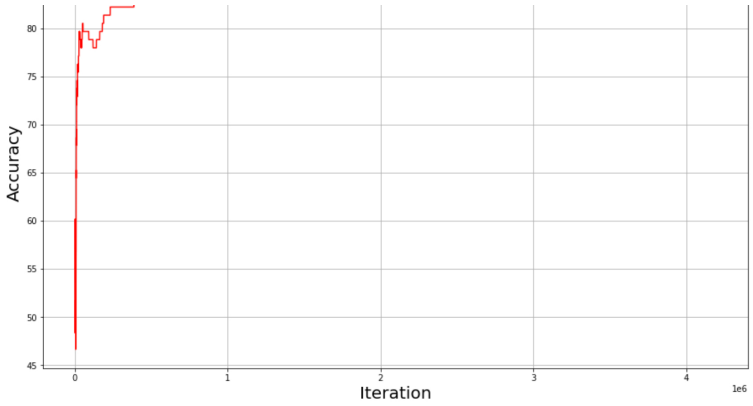
Out[9]: Text(0, 0.5, 'Error')

4. Training Accuracy

In [10]:
```
plt.figure(figsize=(15,9))
plt.plot(accuracy_history, color='red', label='Accuracy (%)')
plt.grid()
plt.legend()
plt.title('Training Accuracy', fontsize=40)
plt.xlabel('Iteration', fontsize=20)
plt.ylabel('Accuracy', fontsize=20)
```

Out[10]: Text(0, 0.5, 'Accuracy')





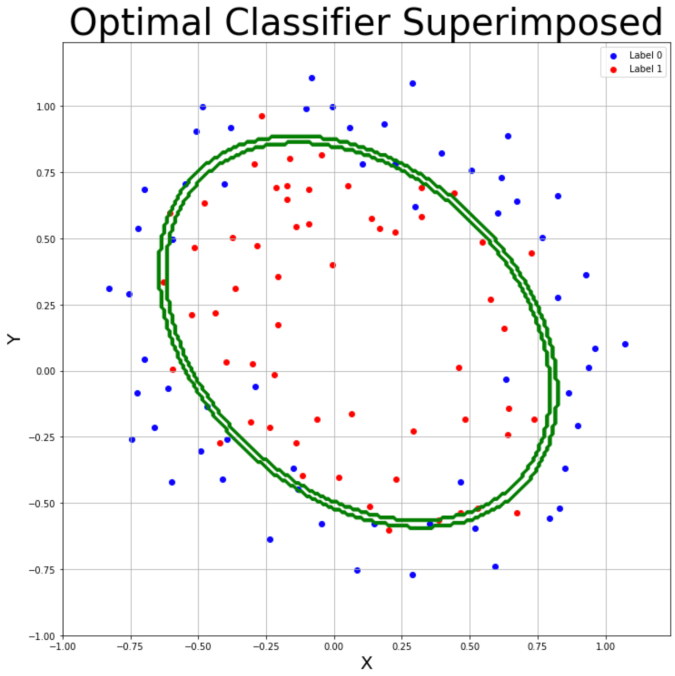
5. Final Training Accuracy

In [11]:
```
print("Accuracy: ", accuracy)
```

```
Accuracy:  85.59322033898304
```

6. Optimal Classifier Superimposed on Training Data

In [12]:
```
plt.figure(figsize=(12,12))
plt.contour(x_grid, y_grid, sigma_visual, colors='g')
plt.scatter(pointX0, pointY0, c='b', label='Label 0')
plt.scatter(pointX1, pointY1, c='r', label='Label 1')
plt.grid()
plt.legend()
plt.title('Optimal Classifier Superimposed', fontsize=40)
plt.xlabel('X', fontsize=20)
plt.ylabel('Y', fontsize=20)
plt.show()
```



In [0]:
```
```