# Programming Exercise 2: Logistic Regression

```
by Seokkyu Kong
Date: 2016-03-18
Summary:

1) Andrew Ng 교수의 강의: https://www.coursera.org/learn/machine-learning/
2) Coursera machine learning (Prof. Andrew Ng) 강의 내용과 assignment는 octave(matlab)으로 이루어진다.
3) 복습차원에서 해당 코드를 python으로 구현해본다.
4) 모든 내용(도표, 수식, 텍스트)은 강의 내용 및 과제에서 가져왔음을 알린다.
```

python 작업 시 참고한 자료: Numpy 와 MATLAB 비교

- Numpy for Matlab users #1 (https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html)
- NumPy for MATLAB users #2 (http://mathesaurus.sourceforge.net/matlab-numpy.html)

```
http://stackoverflow.com/questions/18801002/fminunc-alternate-in-numpy
http://docs.scipy.org/doc/scipy-0.10.0/reference/tutorial/optimize.html
http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimize

# 아래 stackoverflow 내용 중 소스코드가 많은 도움이 되었다.
http://stackoverflow.com/questions/18801002/fminunc-alternate-in-numpy (*****)
http://www.johnwittenauer.net/machine-learning-exercises-in-python-part-3/ (***)
```

## Introduction

이번 연습문제에서는, logistic regression 을 구현하고 2개의 서로 다른 데이터셋에 응용해본다. 프로그래밍 연습문제를 시작하기 전에, 비디오를 보고 관련된 토픽에 대한 리뷰 문제를 완성할 것을 강력히 권장한다.

연습문제를 시작하면서, 스타터 코드를 다운로드 받고 내용을 디렉토리에 풀어서 연습 문제를 완성해야 한다.

### Files included in this exercise

- plotData.m - Function to plot 2D classification data
- sigmoid.m - Sigmoid Function
- costFunction.m - Logistic Regression Cost Function
- predict.m - Logistic Regression Prediction Function
- costFunctionReg.m - Regularized Logistic Regression Cost

연습문제를 통해서, 스크립트 ex2.m과 ex2_reg.m을 사용할 것이다. 이들 스크립트는 문제에 대한 데이터셋을 설정하고 당신이 작성할 함수를 호출하게 된다. 이들 중 어떤 것도 수정할 필요는 없다. 단지 다른 파일에 있는 함수들만 수정하면 된다. 과제에 있는 지시사항을 따라라.

### Where to get help

# 1. Logistic Regression

연습문제에서, **학생들의 대학입학 승인을 예측하기 위한 logistic regression model 를 구축한다.**

당신이 대학 부서의 관리자라고 가정하자. 그리고 응시자의 승인 가능성을 2개 시험의 결과에 근거해서 결정하고 싶다고 하자. 당신은 이전 응시자들로부터 히스토리 데이터를 가지고 있는데 logistic regression의 training set으로 사용할 수 있다. 각각의 training example에 대해서, 2개 시험의 응시자의 점수와 승인 결정이 포함되어 있다.

당신의 작업은 **2가지 시험점수와 승인결과를 가진 training example을 가지고 응시자의 합격 확률을 측정하는 분류 모델을 구축하는 것이 목표이다.**

이것에 대한 개요와 ex2.m내의 프레임워크 코드는 연습문제를 통해서 당신을 안내해 줄것이다.

## 1.1 Visualizing the data

어떤 학습 알고리즘이라도 구현하기 전에 가능하면 자료를 비쥬얼화 해보는 것은 항상 좋은 일이다. ex2.m의 첫번째 파트에서, 코드는 데이터를 로드하고 plotData 함수를 호출해서 차원 plot상에 그것을 표시한다.

당신은 이제 plotData내의 코드를 완성해서 Figure 1 처럼 그림을 표시한다. 여기서 축은 2개 시험의 점수가 되고 긍정과 부정의 examples는 서로 다른 마커로 보여진다.

Figure 1: Scatter plot of training data

plotting에 좀더 익숙해지는데 도움이 되기 위해서, 우리는 plotData.m을 빈 상태로 두었다. 당신은 스스로 그것을 구현하도록 노력해야 한다. 어쨌든, 이것은 옵션 (평가되지 않는) 연습문제이다. 우리는 또한 우리의 구현을 아래처럼 제공한다. 당신은 그것을 복사하거나 참조할 수 있다. 만약 우리의 예제를 복사한다고 선택한다면, 각 명령어가 무엇을 하는지 Octave/MATLAB의 문서를 참고삼아서 배워야 한다.

```
% Find Indices of Positive and Negative Examples
pos = find(y==1); neg = find(y == 0);
% Plot Examples
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 2, ...
     'MarkerSize', 7);
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', ...
     'MarkerSize', 7);
```

In [1]:
```
%pylab inline

# 패키지를 로딩한다.
import numpy as np
import matplotlib.pyplot as plt
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [2]:
```
## Machine Learning Online Class - Exercise 2: Logistic Regression

# 데이터를 로드한다.
# 처음 2개의 컬럼은 시험 점수를 포함하고 세번째 컬럼은 라벨을 포함한다.
data = np.loadtxt('ex2data1.txt', delimiter=',')
data = np.array(data)
X = data[:, 0:2]
y = data[:, 2]
```

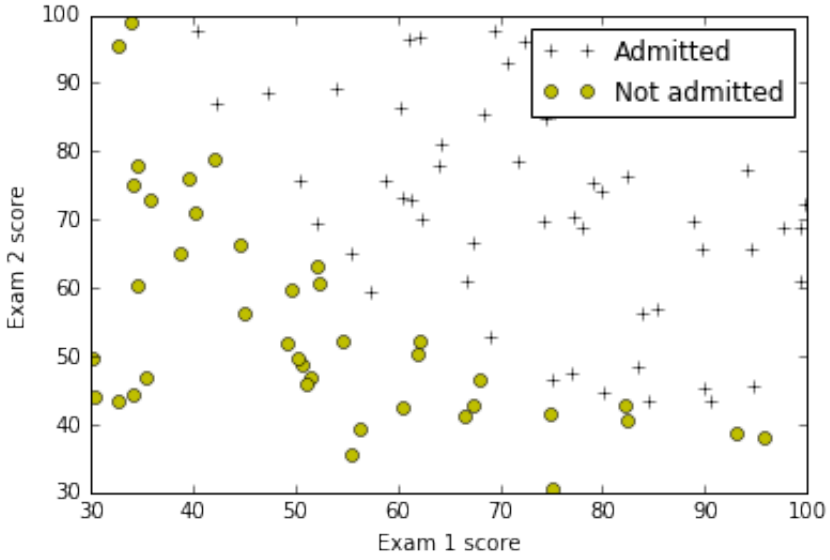
In [3]:
```
## ==================== Part 1: Plotting ====================
# 우리는 데이터를 plotting하면서 연습문제를 시작하는데, 우리가 작업하는
# 문제를 이해하기 위해서이다.

def plotData(X, y):
    """
    데이터 포인트 x와 y를 새로운 figure에 plot한다.
    긍정 examples는 +로, 부정 examples는 o로 표시한다.
    X는 Mx2 행렬이라고 가정한다.
    """

    # ====================== YOUR CODE HERE ======================
    # 지시사항: 긍정과 부정 examples를 2D plot상에 표시하는데, 옵션
    # 'k+'는 긍정 examples에 그리고 'ko' 는 부정 examples에 사용한다.

    # 긍정과 부정 examples의 인덱스를 찾는다.
    # ************************************************************
    # XXX: 인덱스를 찾을 경우, find() 또는 nonzero()를 사용한다.
    # 이렇게 사용하지 말자: pos = (y == 1), neg = (y == 0)
    # ************************************************************
    pos = find(y == 1) # Admitted
    neg = find(y == 0)

    # examples를 plot한다.
```

```
    plt.plot(X[pos, 0], X[pos, 1], 'k+')
    plt.plot(X[neg, 0], X[neg, 1], 'ko', color='y', label='Not admitte
```

```
print('Plotting data with + indicating (y = 1) examples and o \
indicating (y = 0) examples.\n')

plotData(X, y)

# 몇몇 라벨을 표시한다.
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')

# plot 순서로 표시한다.
plt.legend(('Admitted', 'Not admitted'), loc='upper right')
```

```
Plotting data with + indicating (y = 1) examples and o indicating (y
= 0) examples.
```

Out[3]: <matplotlib.legend.Legend at 0x2b062fa8828>




## 1.2 Implementation

### 1.2.1 Warm exercise: sigmoid function

실제 cost function을 가지고 시작하기 전에, logistic regression 의 hypothesis 가설이 다음처럼 정의됨을 기억하라:

$$h_\theta(x) = g(\theta^T x),$$

여기서 함수 g는 sigmoid 함수이다. sigmoid 함수는 다음처럼 정의된다:

$$g(z) = \frac{1}{1+e^{-z}}.$$

첫번째 단계는 sigmoid.m 내의 이 함수를 구현하는 것인데, 프로그램의 나머지 부분들에서 호출된다. 구현을 끝내면, sigmoid(x)를 호출하면서 몇몇 값을 Octave/MATLAB 명령어 라인에서 테스트해 보아라.

**아주 큰 x의 양의 값에 대해서, sigmoid 는 1에 가까워야 하고, 반면에 아주 큰 음의 값에 대해서, sigmoid 는 0에 가까워야 한다. simoid(0)을 평가하면 정확히 0.5의 값을 주어야 한다.**

당신의 코드는 벡터와 행렬에 대해서도 동작해야 한다. **행렬에 대해서, 당신의 함수는 각 요소에 대해서 sigmoid 함수를 수행해야 한다.**

*You should now submit your solutions.*

In [4]:
```
## ============ Part 2: Compute Cost and Gradient ============
# 연습문제의 이 부분에서, logistic regression에 대한 cost와 gradient
# 를 구현할 것이다. costFunction.m 내의 코드를 완성할 필요가 있다.

def sigmoid(z):
    """
    큰 양의 값에 대해서는 결과값이 1에 가깝고,
    큰 음수의 값에 대해서는 0에 가깝다.
    simoid(0) = 0.5
    """
    g = 1 / (1  + np.exp(-z))

    return g

print('sigmoid(10) = %s, sigmoid(-10) = %s, sigmoid(0) = %s'
      % (sigmoid(10), sigmoid(-10), sigmoid(0)))
```

```
sigmoid(10) = 0.999954602131, sigmoid(-10) = 4.53978687024e-05, sigmo
id(0) = 0.5
```

### 1.2.2 Cost function and gradient

이제 logistic regression에 대한 cost function과 gradient를 구현한다. costFunction.m 내의 코드를 완성하고 cost와 gradient를 반환해라.

logistic regression의 cost function은 다음과 같음을 기억하라.

[수식]

그리고 cost의 gradient는 theta와 동일한 길이의 벡터인데, j-번째 요소(for j = 0, 1, ,,, n)은 다음처럼 정의된다:

[수식]

이 gradient가 linear regression gradient와 동일하게 보여짐을 주목하라. 공식은 실제로 다른데 linear와 logistic regression은 서로 다른 $h_\theta(x)$ 를 가지고 있기 때문이다.

일단 완성하면, ex2.m은 costFunction을 호출하는데 초기 파라미터 theta를 사용한다. cost가 약 0.693임을 볼 수 있어야 한다.

In [5]:
```
# You should now submit your solutions.

# 데이터 행렬을 적절히 설정한다. 그리고 절편 항목을 위해서 1을 추가한다.
# data array의 크기를 조회한다.
m, n = np.shape(X)

# 절편 항목을 X와 X_test에 추가한다.
X_1 = hstack((np.ones((m, 1)), X))
#X_1 = np.column_stack((np.ones(m), X))

# 파라미터를 초기화한다. 절편이 추가되니까 n + 1을 해준다.
initial_theta = np.zeros(n+1)
```

In [6]:
```
def costFunction(theta, X, y):
    """
    logistic regression에 대한 cost와 graident를 계산한다.
    cost를 계산하는데 logistic regression에 대한 파라미터로 theta를
    사용한다. 그리고 cost에 대한 gradient를 계산한다.
    """

    # 몇몇 유용한 변수를 초기화 한다.
    m, n = X.shape # m은 training examples 수이다.

    # 다음 변수를 올바르게 반환해야 한다.
    J = 0
    grad = np.zeros(np.size(theta))

    # ====================== YOUR CODE HERE ======================
    # 지시사항: 특정 theta에 대한 cost를 계산한다.
    # J를 cost로 설정한다.
    # 편미분을 계산하고 cost에 대한 편미분을 grad에 설정한다.

    # Note: grad는 theta와 동일한 차원이다.

    """
    XXX: 1) array 에서 * 는 요소곱을 나타낸다. 행렬곱은 .dot() 함수를 사용한다
        2) matrix에서 * 는 행렬곱을 나타낸다. 요소곱은 .multiply() 함수를 사용
        3) 함수 안에서는 matrix 연산으로 통일한다.
    """

    prediction = sigmoid(X.dot(theta))

    # cost J를 구한다.
    J = 1/m * sum(-y * log(prediction) - (1 - y) * log(1 - prediction))

    # gradient를 구한다.
    error = prediction - y

    delta = 1/m * error.T.dot(X)
    grad = delta

    return J, grad
```

In [7]:
```
# 초기 cost와 gradient를 계산하고 표시한다.
cost, grad = costFunction(initial_theta, X_1, y)

print('Cost at initial theta (zeros): %f ' % cost)
print('Gradient at initial theta (zeros): ')
print('%s' % grad)
"""
In Octave/MATLAB:

Cost at initial theta (zeros): 0.693147
Gradient at initial theta (zeros):
 -0.100000
 -12.009217
 -11.262842
"""
print()
```

```
Cost at initial theta (zeros): 0.693147
Gradient at initial theta (zeros):
[ -0.1        -12.00921659 -11.26284221]
```

### 1.2.3 Learning parameters using fminunc

이전 과제에서, gradient descent를 구현해서 linear regression 모델에 대한 최적의 파라미터를 찾았다. 당신은 cost function 과 그것의 gradient를 계산하는 함수를 작성했다. 그리고 난 다음 gradient descent 를 단계적으로 수행했다.

이번에, gradent descent steps을 취하는 대신, Octave/MATLAB의 내장 함수 fminunc를 사용할 것이다.

**Octave/MATLAB의 fminunc는 uncontrained 함수의 최소값을 찾는 optimization solver 이다.**

logistic regression에 대해서, 파라미터 theta를 가진 cost function J(theta)를 최적화 할 것이다.

구체적으로, fminunc를 사용해서 주어진 데이터셋 (X와 y 값)에 대해서 logistic regression cost function에 대한 최적의 파라미터 theta를 찾을 것이다.

- 우리가 최적화하려는 것은 파라미터의 초기 값이다.
- 주어진 training set와 특정 theta가 있을 때 함수는 데이터셋 (X, y)를 위한 theta와 관련된 logistic regression cost와 gradient를 계산한다.

ex2.m 에서, 우리는 이미 올바른 인자를 가지고 fminunc 를 호출하는 코드를 작성했다.

**주석: 최적화에서 Constraints 는 종종 파라미터 상의 constraints를 언급하는데, 예를 들어 theta가 취할 수 있는 (즉, theta <= 1) 가능한 값의 경계가 constraints가 된다. Logistic regression 은 그와 같은 constraints를 가지고 있지 않은데, theta가 어떤 실수 값이라도 가질 수 있기 때문이다.**

```
% Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);

% Run fminunc to obtain the optimal theta
% This function will return theta and the cost
[theta, cost] = ...
    fminunc(@(t)(costFunction(t, X, y)), initial theta, options
);
```

이 코드 snippet에서, 우르는 먼저 fminunc와 함께 사용될 옵션을 정의했다. 특별히, 우리는 **GradObj 옵션을 on으로 설정했는데**, 그것은 **fminunc 에게 우리의 함수가 cost와 gradient 둘 다 반환함을** 알려주는 것이다. 이것은 fminunc 가 함수를 최소화할 때 gradient를 사용하게 해준다. 더우기, 우리는 **MaxIter 옵션을 400으로 설정했기 때문에, fminunc 는 함수가 종료되기 전에 최대 400 단계를 실행할 것이다.**

우리가 최소화하려는 실제 함수를 지정하기 위해, **우리는 "short-hand"를 사용해서 @(t) ( costFunction(t, X, y) )를 가진 함수를** 지정한다. 이것은 **인자 t를 가진 함수를 생성하는데 당신의 costFunction을 호출하게 된다.**

만약 당신이 costFunction을 올바르게 구현했다면, fminunc 는 올바른 최적화 파라미터로 수렴하고 cost 와 theta의 최종 값을 반환하게 된다.

**fminunc를 사용하면서, 당신 스스로 어떤 loops도 작성하지 않고 또는 gradient descent를 위해서 했던것과 같이 learning rate도 설정하지 않음을 주목해라.**

이것은 fminunc에 의해 모두 수행된다: 당신은 cost와 gradient를 계산하는 함수 하나를 제공하기만 하면 되는 것이다.

일단 fminunc 가 완성되면, ex2.m은 당신의 costFunction 함수를 최적의 theta 파라미터로 호출할 것이다. cost가 약 0.203 임을 볼 수 있어야 한다.

**최종 theta 값은 training data 상에서 decision boundary를 plot 하기 위해 사용되는데, Figure 2와 비슷한 결과가 될 것이다. 우리는 또한 plotDecisionBoundary.m 내의 코드를 보고 theta 값을 사용해서 그와 같은 경계선이 어떻게 plot 되는지 살펴보기를 권장한다.**

### 1.2.4 Evaluating logistic regression

파라미터를 학습한 이후에, 당신은 특정 학생이 승인될 것인지 여부를 예측하기 위해서 모델을 사용할 수 있다. Exam 1의 점수가 45점, 그리고 Exam 2의 점수가 85점인 학생에 대해서, 승인될 확률이 0.776임을 기대해야 한다.

**우리가 발견한 파라미터의 quality를 평가하기 위한 다른 방법으로 우리의 training set 상에서 학습된 모델이 얼마나 잘 예측하는지 보는 것이다.**

Figure 2: Training data with decision boundary

이 부분에서, 당신의 작업은 predict.m 내의 코드를 완성하는 것이다. predict 함수는 "1" 또는 "0"의 예측값을 주어진 dataset과 학습된 파라미터 벡터 theta를 이용해서 만들어낸다.

predict.m내의 코드를 완성한 이후에, ex2.m 스크립트는 올바르게 맞춘 examples의 퍼센티지를 계산함으로써 당신의 classifier의 training 정확도를 리포트 할 것이다.

*You should now submit your solutions.*

In [8]:
```
## ============= Part 3: Optimizing using fminunc  =============
# 이번 연습문제에서, 당신은 내장 함수 (fminunc) 를 사용해서 최적의
# theta 파라미터를 찾을 것이다.

# minimize 함수 호출을 사용하기 위해 import 한다.
import scipy.optimize as op

# octave 에서는 costFunction 안에서 cost와 gradient를 한꺼번에 계산했다.
# python 에서는 minimize () 사용을 위해서 cost와 gradient 계산을 분리한다.

def myCost(theta, X, y):
    """
    cost J를 계산한다.
    """

    m, n = X.shape
    # 아래와 같은 방식으로 사용하지 말자.
    #theta1 = theta.reshape(len(theta), 1, order='F').copy()

    predictions = X.dot(theta)
    sig_term = sigmoid(predictions)

    # 원본 수식
    J = 1/m * np.sum(-y * log(sig_term) - (1 - y) * log(1 - sig_term))

    return J

def myGradient(theta, X, y):
    """
    gradient를 계산한다.
    """
    m, n = np.shape(X)

    # errors 의 차원을 확인해야 한다. predictions과 동일한 차원인지 확인한다.
    # dim(predictions) = dim(erros) = dim(y) = m x 1
    predictions = sigmoid(X.dot(theta))
    errors = predictions - y

    delta = 1/m * errors.T.dot(X) # => 1 x n
    grad = delta

    # ***************************************************************
    # XXX: 함수내에서 gradient 반환은 array 타입으로 반환하는 것이 중요하다.
    # return format: array([ -0.1       , -12.00921659, -11.26284221])
    # ***************************************************************
    return grad
```

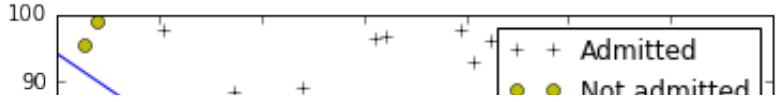
```
# fminunc를 위해서 옵션을 설정한다.
# options = optimset('GradObj', 'on', 'MaxIter', 400); # Octave Code

costFunc = lambda p: costFunction(p, X_1, y)[0]
gradFunc = lambda p: costFunction(p, X_1, y)[1]

# 최적의 theta를 구하기 위해 fminunc를 실행한다.
# 이 함수는 theta와 cost를 반환한다.

# 방법 1) OK
Result = op.minimize(fun = costFunc, x0 = initial_theta, method = 'TNC

# 방법 2) OK
#Result = op.minimize(fun = myCost, x0 = initial_theta, args = (X_1, y
#                     , jac = myGradient);

# 방법 3) OK
#Result = op.fmin_tnc(func = myCost, x0 = initial_theta, fprime = myGra

cost = Result.fun
theta = Result.x

# Cost at theta found by fminunc: 0.203497701589.
# theta: [-25.16131861   0.20623159   0.20147149]
print('Cost at theta found by fminunc: %s. \n' % cost)
print('theta: ')
print('theta: %s. \n' % theta)


# XXX: 아래 방식 fmin_bfgs()는 잘못된 결과가 나오는데,
# 어떤 부분이 잘못되었는지 모르겠다. 추후 검토 필요함
#Result = op.fmin_bfgs(f = myCost, x0 = initial_theta, args = (X_1, y)
#print (myCost(Result[0], X_1, y))
```

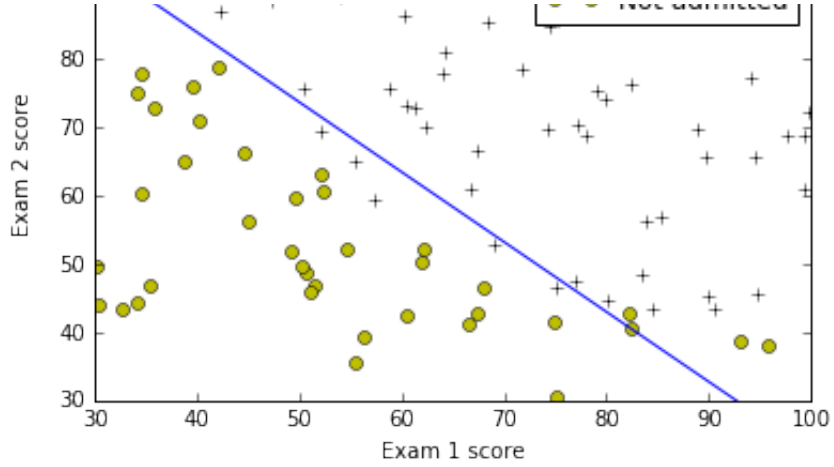
```
Cost at theta found by fminunc: 0.203497701589.

theta:
theta: [-25.1613187    0.20623159   0.20147149].
```

In [9]:
```
def plotDecisionBoundary(theta, X, y):
    """
    데이터 포인트 x와 y를 theta에 의해 정의된 decision boundary를 가진 새로운
    figure 상으로 plot 한다.
    이 함수는 data points를 plot하는데 positive exampes는 +로, negative ex
    표시한다. x는 다음 둘 중 하나로 가정한다.
    1) Mx3 행렬, 첫번째 컬럼은 절편을 위한 1의 벡터로 구성됨
    2) MxN, N > 3 행렬, 첫번째 컬럼은 모두 1로 구성됨
    """

    # 데이터를 plot 한다.
    plotData(X[:, 1:3], y)
```

```
    if np.size(X, 1) <= 3:
        # line을 정의하기 위해서는 단지 2개의 점만 필요하다.
        # 따라서 2개의 endpoints를 선택한다.
        plot_x = np.array([np.min(X[:, 1]) - 2, np.max(X[:, 1] + 2)])

        # ************************************************************
        # XXX: decision boundary line: theta0 + theta1 * x + theta2 * y
        # ************************************************************
        # decision boundary line을 계산한다.
        plot_y = (-1 / theta[2]) * (theta[1] * plot_x + theta[0])

        # plot을 그린다. 그리고 더 좋은 뷰잉을 위해 축을 조정한다.
        plot(plot_x, plot_y)

        # Legent, 연습문제를 위해 지정함
        plt.legend(('Admitted', 'Not admitted'), loc='upper right')
        plt.axis([30, 100, 30, 100])

    # 변환 작업 중
    else:
        # grid range를 설정한다. linspace는 octave, python 동일한 사용법임
        u = linspace(-1, 1.5, 50) # 50,
        v = linspace(-1, 1.5, 50)

        z = np.zeros((np.size(u), np.size(v)))

        # z = theta * x 를 grid 상에서 평가한다.
        for i in np.arange(np.size(u)):
            for j in np.arange(np.size(v)):
                z[i, j] = np.sum(mapFeature(u[i], v[j]) * theta)

        z = z.T # contour를 호출하기 전에 transpose하는 것이 중요하다.

        # Plot z = 0
        # range를 [0, 0]으로 할 필요가 있다.
        # contour(u, v, z, [0, 0], 'LineWidth', 2); # Octave code
        plt.contour(u, v, z)


# 경계선을 plot 한다.
plotDecisionBoundary(theta, X_1, y)
# label과 Legend를 표시한다.
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')

# plot 순서대로 명시한다.
plt.legend(('Admitted', 'Not admitted'), loc='upper right')
```

Out[9]: <matplotlib.legend.Legend at 0x2b063db6cc0>






In [10]:
```
## ============== Part 4: Predict and Accuracies ==============
# 파라미터를 학습한 이후에 여태껏 본적 없는 데이터에 대해서 결과를 예측하기 위해 
# 여기에서는, logtistic regression model을 사용해서, 시험 1에 대해서 45점, 
# 받은 학생이 학교의 승인을 받을 확률을 예측한다.
#
# 더우기, 당신은 우리의 모델에 대해서 training과 test set 의 정확도를 계산할 수
#
# 당신의 작업은 predict.m 내의 코드를 완성하는 것이다.


# predict 함수는 주어진 데이터와 theta를 이용해서 1 또는 0의 예측값을 만들어낸다
# 확률이 50% 를 넘으므로, 합격으로 예측한다.
# array 에서 * 는 요소곱이다. 따라서 np.sum 을 이용해 곱의 합을 구한다.
def predict(theta, X):
    """
    학습된 logistic regression 파라미터 theta를 이용해서 label 이 0 또는 1인
    이 함수는 0.5에서의 threshold를 이용해서 X에 대한 예측을 계산한다.
    (즉, 만약 sigmoid(theta'*x) >= 0.5, 1로 예측함)
    """

    m = X.shape[0] # training exampels의 개수

    # 다음 변수를 올바르게 반환해야 한다.
    p = np.zeros(m)

    # ======================== YOUR CODE HERE ========================
    # 지시사항: 다음 코드를 완성해서 학습된 logistic regression 파라미터를
    # 이용한 예측을 해보아라.
    # p는 0또는 1의 벡터로 설정되어야 한다.

    p = sigmoid(X.dot(theta)) >= 0.5;

    return p

"""
In Octave/MATLAB:

theta =
```

```
  -25.16127
    0.20623
    0.20147
"""
# exam 1의 점수 45점과 exam 2의 점수 85점을 받은 학생에 대해서 예측한다.
prob = sigmoid(np.sum(np.array([1, 45, 85]) * theta))

print('For a student with scores 45 and 85, we predict an admission pro


# training set 에 대한 정확도(accuracy)를 계산한다.
# Train Accuracy: 89.000000
p = predict(theta, X_1)

print('Train Accuracy: %f' % (np.mean(double(p == y)) * 100))

"""
In Octave/MATLAB:

For a student with scores 45 and 85, we predict an admission probabilit

Train Accuracy: 89.000000
"""
print()
```

```
For a student with scores 45 and 85, we predict an admission probabil
ity of 0.776291
Train Accuracy: 89.000000
```

# 2. Regularized logistic regression

연습문제의 이 부분에서는 **regularized logistic regression을 구현해서 제조공정의 마이크로칩이 QA를 통과할지 대한 여부를 예측하게 된다.** QA 동안, 마이크로칩은 기능이 올바르게 동작하는지 확인하기 위해 여러가지 테스트를 거친다.

당신이 공장의 제품 매니저라고 가정하자. 그리고 2개의 서로 다른 테스트로부터 몇몇 마이크로칩에 대한 테스트 결과를 가지고 있다고 하자. 당신은 2개의 서로 다른 테스트 결과로부터 마이크로칩이 통과할지 그렇지 않을지를 결정하고 싶어한다. 의사결정을 돕기 위해 과거 테스트 결과 데이터셋을 가지고 있고 그것을 이용해서 logistic regression model 을 구축할 수 있다.

ex2_reg.m 스크립트를 사용해서 나머지 연습문제를 해결한다.

## 2.1 Visualizing the data

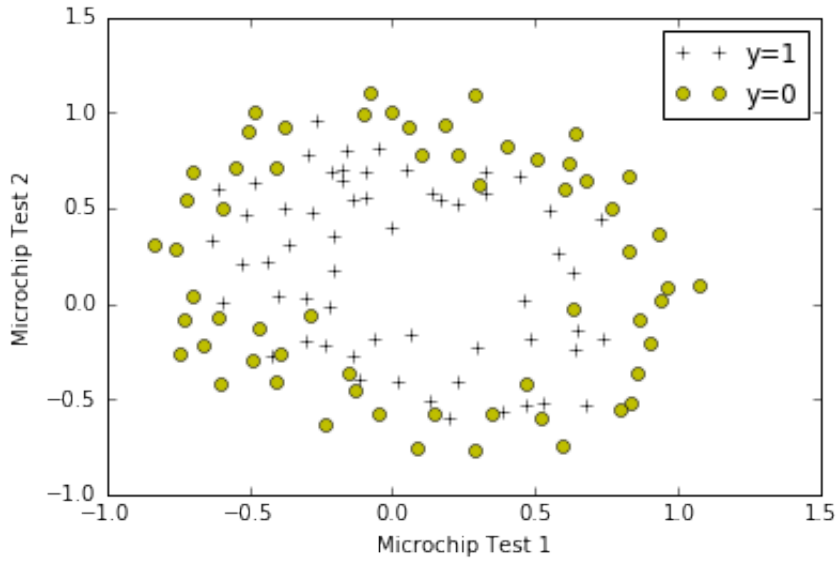
이전 연습문제와 비슷하게 plotData는 Figure 3.과 같은 그림을 만들기 위해 사용된다. 여기서 축은 2개의 테스트 점수이고, positive 는 y=1 이고 accept 된것을 의미한다. negative 는 y=0 이고 reject 된 것을 의미한다. 이들 examples는 서로 다른 마커로 보여진다.

[Figure 3: Plot of training data]

**Figure 3은 우리의 데이터셋이 plot 상에서 직선에 의해 positive와 negative examples가 분리될 수 없음을 보여준다. 따라서, logistic regression의 직선 응용은 이 데이터셋상에 잘 수행될 수 없다. 왜냐하면 logistic regression은 linear decision boundary만 찾을 수 있기 때문이다.**

In [11]:
```
## Machine Learning Online Class - Exercise 2: Logistic Regression

import numpy as np
import matplotlib.pyplot as plt

# 데이터를 로드한다.
# 처음 2개의 컬럼은 X 값을 포함하고 세번째 컬럼은 label (y)를 포함한다.

data = np.loadtxt('ex2data2.txt', delimiter=',')
X = data[:, 0:2]
y = data[:, 2]

plotData(X, y)

# 라벨과 legend 설정
plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')

# plot 순서로 지정한다.
plt.legend(('y=1', 'y=0'), loc='upper right')
```

Out[11]: <matplotlib.legend.Legend at 0x2b063e33320>




## 2.2 Feature mapping

**data를 더 잘 맞게(fit) 하는 방법은 각각의 data point 로 부터 더 많은 feature를 생성하는 것이다.** 제공된 함수 mapFeature.m 에서 **feature들을 다항식 항목인 x1과 x2를 이용해서 6 차까지 매핑할 것이다.**

[수식]

이 매핑의 결과로 2개의 feature로(2개의 QA 테스트의 점수) 이루어진 벡터는 28 dimension vector로 변환된다.

**고차원의 피처 벡터 상에서 훈련된 logistic regression classifier는** 더 복잡한 decision boundary를 갖게 되고 **2차 평면에 그려질 때 nonlinear (직선이 아닌) 로 나타나게 된다.**

**feature mapping 이 더욱 표현이 좋은 분류기를 구축하지만 반면에 overfitting 에 더욱 민감해 진다.** 연습문제의 나머지에서 regularized logistic regression 을 구현해서 data를 fit 하게 되고 **regularization 이 어떻게 overffiting 과 싸우는데 도움이 되는지 직접 확인할 것이다.**

## 2.3 Cost function and gradient

이제 regularized logistic regression 에 대한 cost function 과 gradient를 계산하는 코드를 구현할 것이다. costFunctionReg.m 내의 코드를 완성해서 cost와 gradient를 반환하도록 해라.

logistic regression의 regularized cost function은 다음과 같다.

[수식]

**파라미터 $\theta_0$는 regularize 하지 않음을 주의해라.** Octave/MATLAB에서, 인덱싱은 1부터 시작한다. 따라서 theta(1) (theta0에 일치하는) 는 regularizing 하면 안된다.

cost function의 gradient 는 벡터인데 j-번째 요소는 다음처럼 정의 된다.

[수식]

일단 완성하면, ex2_reg.m은 costFunctionReg 함수를 호출하는데 theta의 초기값(모두 0으로 초기화된)을 사용한다. 당신은 cost가 약 0.693임을 보아야 한다.

*You should now submit your solutions.*

### 2.3.1 Learning parameters using fminunc

이전 파트와 유사하게, 당신은 fminunc를 사용해서 최적의 파라미터 theta를 학습하게 될 것이다. 만약 regularized logistic regression에 대한 cost와 gradient를 올바르게 완성했다면 (costFunctionReg.m), ex2_reg.m의 다음 부분에서 fminunc를 사용해서 파라미터 theta를 학습하는 단계로 진행할 수 있다.

In [31]:
```
## ============ Part 1: Regularized Logistic Regression ============
# 이 파트에서, 선형으로 분리될 수 없는 데이터 포인트를 가진 데이터셋이
# 주어진다. 어쨌든, 당신은 logtistic regression을 사용해서 데이터 포인트를
# 분류하고 싶어할 것이다.
#
```

```
"
# 그렇게 하기 위해서, 당신은 더 많은 feature를 사용해야 하는데 -- 특별히
# 데이터 행렬에 polynomial features를 추가한다. (polynomial regression과 유

# Polynomial features를 추가한다.
# mapFeature는 intercept term(절편 항목)인 1의 컬럼을 추가한다.

def mapFeature(X1, X2):
    """
    polynomial features로 feature를 매핑하는 함수이다.

    이 함수는 2개의 입력 features를 regularization 연습문제에 사용되는
    quadratic features로 매핑한다.

    더 많은 features를 가진 새로운 feature를 반환하는데,
    X1, X2, X1.^2, X2.^2, X1*X2, X1*X2.^2 등으로 구성되어 있다.

    입력 X1, X2는 동일한 사이즈여야 한다.
    """

    degree = 6

    #print(X1.shape)

    m = np.size(X1)
    out = np.ones(m) # 1로 채워진 절편 컬럼 벡터

    # XXX: 벡터 단위로 계산된 내용을 list에 추가하는 방식이 편하다.
    # 계산된 컬럼 벡터를 list에 추가한다.
    # res = [[1,1,1,], [2,2,2,], [3,3,3,]]
    res = []
    res.append(out)

    for i in np.arange(1, degree + 1):
        for j in np.arange(0, i + 1):
            # ***********************************************************
            # XXX: 아래와 같은 수식은 어떻게 만드는걸까?
            # ***********************************************************
            out1 = (X1 ** (i - j)) * (X2 ** j)
            res.append(out1)

    # list에 저장된 것을 2차원 array 벡터로 변환하고 전치 시켜서 반환한다.
    return np.array(res).T

def costFunctionReg(theta, X, y, s_lambda):

    """
    regularization을 가진 logistic regression에 대한 cost와 gradient를 계
    이 함수는 regularized logistic regression에 대한 파라미터로 theta를 사용
    계산한다. 그리고 파라미터에 일치하는 cost의 gradient를 계산한다.
    """

    # 몇몇 유용한 변수를 초기화한다.
    m, n = X.shape
    #m = np.size(y) # training examples 개수
```

```
    #m = np.size(y) # training examples 수

    # 아래 값을 올바르게 반환해야 한다.
    J = 0
    grad = np.zeros(np.size(theta))

    # ====================== YOUR CODE HERE ======================
    # 특정 theta에 대한 cost를 계산한다.
    # 편미분을 계산하고 grad를 theta 내의 각 파라미터에 관한 cost의
    # 편미분 값으로 설정한다.

    """
    cost는 J 값으로 설정한다. 편미분을 계산하고 결과를 grad에 설정한다.

    힌트: cost 함수와 gradient 계산은 효율적으로 벡터화될 수 있다. 예를 들어 다
    계산을 생각해보자.

    sigmoid(X * theta)

    결과 행렬의 각 row는 해당 example의 예측값이 된다.

    힌트: regularized cost 함수의 gradient를 계산할 때, 많은 벡터화 솔루션이
    한가지 가능한 방법은 아래와 같다

    grad = (unregularized gradient for logistic regression)
    temp = theta;
    temp(1) = 0;   % because we don't add anything for j = 0
    grad = grad + YOUR_CODE_HERE (using the temp variable)

    """

    # sum_theta_square는 theta[0]를 포함하지 않는다.
    sum_theta_square = np.sum(theta[1:] ** 2)

    # 예측값을 계산한다.
    X_theta = X.dot(theta)
    prediction = sigmoid(X_theta)

    # 1) cost J를 계산한다.
    J = 1/m * np.sum(-y * log(prediction) - (1 - y) * log(1 - predictio
        + s_lambda / (2 * m) * sum_theta_square

    # 2) 편미분을 계산한다.
    error = prediction - y
    delta = 1/m * error.T.dot(X)

    temp = theta
    temp[0] = 0
    regular = (s_lambda / m) * temp

    grad = delta + regular

    return J, grad

# mapFeature는 1인 컬럼을 추가하고 입력을 조인해라. 따라서 절편 항목을 처리된다
```

```
# mapFeature는 1의 컬럼을 추가하고 있음을 주의해라. 데이터 컬럼 상수도 처리한다.
# X_reg: 118 x 28 dimension, 절편이 포함되어 있다.
X_reg = mapFeature(X[:, 0], X[:, 1])

#print(X_reg[0:2, :])

# fitting 파라미터를 초기화한다.
initial_theta = np.zeros(X_reg.shape[1]) # initial_theta: 28 x 1

# regularization 파라미터 lambda를 1로 설정한다.
# lambda 는 python 키워드이기 때문에 변수명을 s_lambda로 사용한다.
s_lambda = 1

# regularized logistic regression 에 대한 초기 cost 와 gradient 를 계산하고
cost, grad = costFunctionReg(initial_theta, X_reg, y, s_lambda)

# cost is 0.69314718056 and grad is [  8.47457627e-03   1.87880932e-02
print('Cost at initial theta (zeros): %f' % cost)
```

```
Cost at initial theta (zeros): 0.693147
```

## 2.4 Plotting the decision boundary

이 classifier에 의해 학습된 모델을 가시화하는 것을 돕기 위해, 우리는 **plotDecisionBoundary.m 함수를 제공하는데, 그것은 (비선형) decision boundary를 plot 해서 positive와 negative examples를 분리하도록 해준다.**

plotDecisionBoundary.m 내에서, 우리는 공편하게 구분된 그리드 상에서 classifier의 예측을 계산함으로써 비-선형 decision boundary를 plot 한다. 그리고 나서 contour plot을 그리는데 예측의 변화는 y = 0 에서 y = 1 까지이다.

파라미터 theta를 학습한 후에, ex_reg.m의 다음 단계는 Figure 4와 유사한 decision boundary를 plot 할 것이다.

## 2.5 Optional (ungraded) exercises

연습문제 이 부분에서는, 데이터셋에 대한 서로 다른 regularization 파라미터를 시도해보고 regularization 이 overfitting을 어떻게 방지하는지 이해할 수 있을 것이다.

**lambda를 변화함에 따라 decision boundary에서의 변화를 주목하라.** 작은 lambda에서는, classifier가 거의 모든 training example를 올바르게 분류하고 있음을 알 수 있다. 그러나 배우 복잡한 경계선을 그리고 있어서, 데이터를 과적합 overfitting 하고 있다. (Figure 5).

이것은 좋은 decision boundary가 아니다: 예를 들어, x = (-0.25, 1.5) 의 지점에서 예측을 할 경우 (y = 1)로 승인된다. 그것은 주어진 trainig set 에서 올바르지 않게 보인다.

큰 lambda에 대해서, plot이 더 단순한 decision boundary를 보여주는데 그것은 positives와 negatives를 상당히 잘 분리하고 있음을 알 수 있다. 어쨌든, 만약 lambda가 너무 높은 값이라면, 당신은 좋은 fit을 얻지 못하고 decision boundary도 데이터를 잘 따르지 못할 것이다. 이것은 데이터를 underfitting 하게 된다. (Figure 6).

*You do not need to submit any solutions for these optional (ungraded) exercies.*

[Figure 4: Training data with decision boundary (lambda = 1)]

[Figure 5: No regularization (Overfitting) (lambda = 0)]

[Figure 6: Too much regularization (Underfitting) (lambda = 100)]

## Part 2: Regularization and Accuracies

여기서는 lambda값을 서로 다르게 했을 때, regularization 이 decision boundary에 어떤 영향을 미치는지 알아보도록 한다.

lambda를 0, 1, 10, 100으로 각각 설정해본다.

이때 decision boundary와 training set의 accuracy가 어떻게 변화하는가?

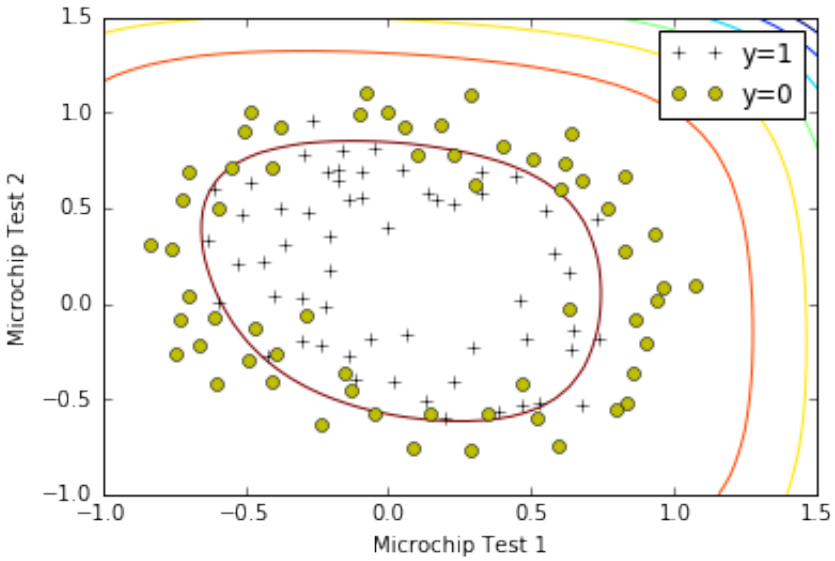
In [37]:
```
# ============== Part 2: Regularization and Accuracies ==============
# Optional Exercise:
# 여기에서, 당신은 서로 다른 lambda 값을 시도해보고 regularization이
# 어떻게 decision coundart에 영향을 주는지 보게 될 것이다.
#
# lambda를 다음 값으로 시도해 보라 (0, 1, 10, 100)
#
# lambda가 변할 때, decision boundary는 어떻게 변화하는가? training set의
# 정확도는 어떻게 변화하는가?

# minimize 함수 호출을 사용하기 위해 import 한다.
import scipy.optimize as op

# octave 에서는 costFunction 안에서 cost와 gradient를 한꺼번에 계산했다.
# python 에서는 minimize () 사용을 위해서 cost와 gradient 계산을 분리한다.


# fitting 파라미터를 초기화한다.
initial_theta = np.zeros(X_reg.shape[1])


# regularization 파라미터 lambda를 1로 설정한다. (이 값을 변화시켜야 한다)
#s_lambda = 0 # cost = 0.257
s_lambda = 1 # cost = 0.529
#s_lambda = 10 # cost = 0.648
#s_lambda = 100 # cost = 0.686

# Options를 설저한다.
# options = optimset('GradObj', 'on', 'MaxIter', 400);
```

```
# Optimize

# costFunctionReg(theta, X, y, s_lambda)
costFunc = lambda p: costFunctionReg(p, X_reg, y, s_lambda)[0]
gradFunc = lambda p: costFunctionReg(p, X_reg, y, s_lambda)[1]

# method: CG => OK / TNC => NOK
Result = op.minimize(fun = costFunc, x0 = initial_theta, \
                     method = 'CG', jac = gradFunc, options = {'maxite

cost = Result.fun
theta = Result.x

# In Octave: Cost at theta found by fminunc: 0.529004
print('Cost at theta found by fminunc: %s. \n' % cost)
print('theta: ')
print(' %s' % theta)

"""
In Octave:
theta:
 1.273005
 0.624876
 1.177376
 -2.020142
 -0.912616
 -1.429907
"""
print()
```

```
Cost at theta found by fminunc: 0.5290027344395168.

theta:
 [ 1.27257437  0.62529974  1.18105434 -2.01959854 -0.91720416 -1.4313
5805
  0.12413601 -0.36557735 -0.35743292 -0.17482593 -1.45831068 -0.05123
77
 -0.61580723 -0.27469253 -1.19281652 -0.24221793 -0.20605881 -0.04496
603
 -0.27780105 -0.29547585 -0.45639359 -1.04351029  0.02759301 -0.29256
359
  0.01542301 -0.32749156 -0.14394421 -0.92493299]
```

In [38]:
```
# 경계선을 그린다.
plotDecisionBoundary(theta, X_reg, y)
plt.xlabel('Microchip Test 1')
plt.ylabel('Microchip Test 2')

plt.legend(('y=1', 'y=0', 'Decision boundary'), loc='upper right')
```

Out[38]: <matplotlib.legend.Legend at 0x2b063f804a8>




In [40]:
```
# training set에 대한 정확도(accuracy)를 계산한다.
p = predict(theta, X_reg)

# In Octave: Train Accuracy: 83.050847
print('Train Accuracy: ', np.mean(double(p == y)) * 100)
```

```
Train Accuracy:  83.0508474576
```

# The End

In [ ]:
```

```