Contents

Machine Learning

June 2, 2016

Part I

Week 3: Classification and Logistic Regression

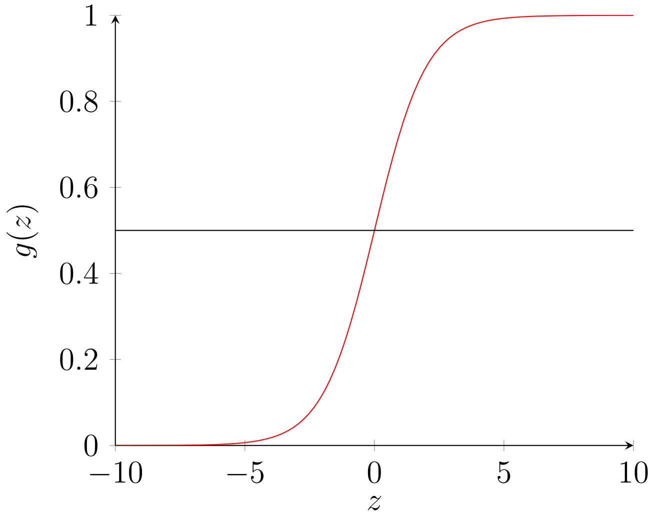


Figure 1: Sigmoid (Logistic) function: $g(z) = \frac{1}{1+e^{-z}}$

0.1 Logistic regression: binary classification

Logistic regression is a **classification algorithm** where the value y take only a small number of discrete values.

- **binary classification** problem: y can only take 2 values: $y \in \{0, 1\}$.('0' is called the negative class and '1' the positive class). Given $x^{(i)}$, the corresponding $y^{(i)}$ is also called **the label** (for the training example).
- **multi-class classification problem**: $y \in \{0, 1, 2, 3\}$ or more.

0.1.1 Hypothesis representation

The classifier must output value between 0 and 1: $0 \leq h_\theta(x) \leq 1$, which can be achieved using *sigmoid* function:

$$h_\theta(x) = g(\theta^{\mathrm{T}} x) = \frac{1}{1 + e^{-\theta^{\mathrm{T}} x}}$$

We interpret $h_\theta(x)$ **as the estimate of the probability of** $y = 1$ on input x.

$$\begin{aligned}
&P(y = 1|x; \theta) = h_\theta(x) \\
&P(y = 0|x; \theta) = 1 - h_\theta(x) \\
&P(y = 1|x; \theta) + P(y = 0|x; \theta) = 1
\end{aligned}$$

$P(y = 1|x; \theta)$ reads: probability that $y = 1$ given x parameterized by θ.

Exple: if for $x = \left[\begin{smallmatrix} x_0 \\ x_1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 \\ \text{tumor size} \end{smallmatrix}\right]$, $h_\theta(x) = 0.7$, it implies that there is 70% chance that the tumor of the patient is malignant ($y = 1$).

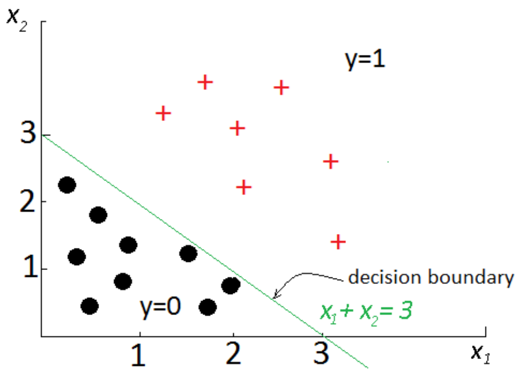
0.1.2 Decision boundary

$$h_\theta(x) = g(\theta^{\mathrm{T}}x) = \frac{1}{1+e^{(\theta^{\mathrm{T}}x)}} = P(y=1|x;\theta)$$

From the sigmoid curve, one can see that :

$$\begin{aligned}
&\text{Predict } y = 1 \text{if } h_\theta(x) = g(\theta^{\mathrm{T}x}) \geq 0.5 \Rightarrow \text{when } \theta^{\mathrm{T}}x \geq 0 \\
&\text{Predict } y = 0 \text{if } h_\theta(x) = g(\theta^{\mathrm{T}x}) < 0.5 \Rightarrow \theta^{\mathrm{T}}x < 0
\end{aligned}$$

Exple: Let's assume a hypothesis function: $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2$, with $\theta = \left[\begin{smallmatrix}-3\\1\\1\end{smallmatrix}\right]$



- Predict $y = 1$ if $\theta^{\mathrm{T}x} = -3 + x_1 + x_2 \geq 0 \Rightarrow (x_1 + x_2) \geq 3$
- Predict $y = 0$ if $\theta^{\mathrm{T}x} = -3 + x_1 + x_2 < 0 \Rightarrow (x_1 + x_2) < 3$

The decision boundary is defined by $h_\theta(x) = 0.5$ $(x_1 + x_2 = 3)$

0.1.3 Cost function for a classification problem

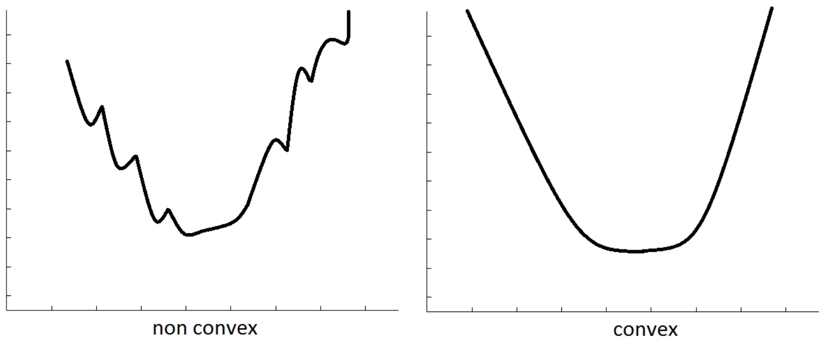
- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), ..., (x^{(m)}, y^{(m)})\}$ with m examples
- feature vector: $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$ with $x_0 = 1$, and $y \in \{0, 1\}$
- The hypothesis: $h_\theta(x) = \frac{1}{1+e^{-\theta^{\mathrm{T}}x}}$

For linear regression, we defined the cost function $J(\theta)$:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\text{Cost}\left[h_\theta(x^{(i)}), y^{(i)}\right]$$

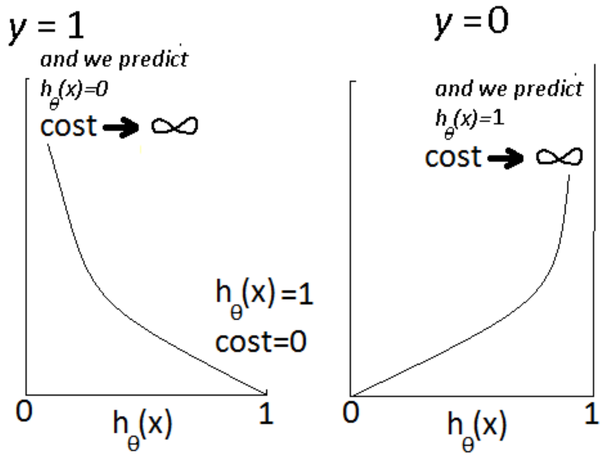
where "Cost":

$$\mathrm{Cost(h_\theta(x), y)} = \frac{1}{2}\left(h_\theta(x) - y\right)^2$$



For logistic regression, we use a different form of the function "Cost" and $J(\theta)$, so that $J(\theta)$ is convex, and gradient descent can converge:

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\mathrm{Log}(h_\theta(x)) & \text{if } y = 1 \\ -\mathrm{Log}(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



The function 'Cost' can take a compact form:

$$\mathrm{Cost}(h_\theta(x), y) = -y\mathrm{Log}\left[h_\theta(x)\right] - (1-y)\mathrm{Log}\left[1 - h_\theta(x)\right]$$

If $y = 1$, $\mathrm{Cost}(h_\theta(x), y) = -\mathrm{Log}(h_\theta(x))$ like in granular form of the equation. Similarly for $y = 0$

Hence, $J(\theta)$ becomes:

$$\begin{aligned} J(\theta) &= \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\ &= -\frac{1}{m}\left(\sum_{i=1}^{m} y^{(i)}\mathrm{Log}(h_\theta(x^{(i)})) + (1-y^{(i)})\mathrm{Log}\left[1 - h_\theta(x^{(i)}))\right]\right) \end{aligned}$$

0.1.4 Gradient Descent

1. Make prediction given $x \Rightarrow$ output: $h_\theta(x) = \frac{1}{1+e^{-\theta^{\mathrm{T}}x}} = h_\theta(x) = P(y=1|x;\theta)$

2. Calculate the cost function $J(\theta)$

3. Similarly to Batch Gradient Descent, we need $\min_\theta J(\theta)$.

Repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

with

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

(update all θ_j simultaneously: $j = 0$ to n)
}

Note that feature scaling can also be applied to logistic regression

0.2 Advanced Optimization

There are several optimization algorithms:

- Gradient Descent

$$\left.\begin{array}{l}\bullet \text{ conjugate gradient} \\ \bullet \text{ BFGS} \\ \bullet \text{ L} - \text{BFGS}\end{array}\right\} \text{No need to manually pick } \alpha\text{. Faster than Gradient descent. But, they are also more complex.}$$

(*advanced Numerical Computing *)

In MatLab, ***fminunc()*** (function minimization unconstrained) is a built-in advanced optimization function (>> help ***fminunc***). In Python, the package ***scipy.optimize*** includes some optimization algorithms:

Here is the procedure for using ***fminunc()***:

1. Step1: Generate a function ***costFunction*** that outputs 2 arguments

   - *jVal*: the value of $J(\theta)$
   - *gradient*: a vector with the value of the gradients

```
function [jVal, gradient] = costFunction(theta)
        jVal = [' code to compute J(theta) '];
        gradient = zeros(n+1, 1) #create a zero vector
        #and fill with gradient values
        gradient(1) = [' code to compute (d J(theta)/d theta_0) '];
        gradient(2) = [' code to compute (d J(theta)/d theta_1) '];
        .....
        .....
        gradient(n+1) = [' code to compute (d J(theta)/d theta_(n))     '];
```

2. Step2: Set the options

```
options = optimset('GradObj', 'on', 'MaxIter', '100');
```

- '100' is the maximum number of iterations
- 'GradObj', 'on' : tells ***fminunc()*** that our function returns both the cost and the gradient

Initial guess for theta

```
initialtheta = zeros(n+1,1)
```

3. Step3: run ***fminunc()***

```
[optTheta, functionVal, exitFlag] = fminunc(@CostFunction, initialTheta, options)
```

'@CostFunction' is a pointer to the function ***CostFunction()***.

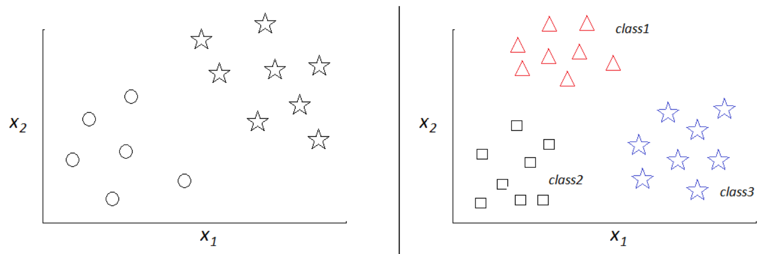
4. Setp 4: ***fminunc()*** outputs 3 parameters values:

- **optTheta** = optimum value of θ as a vector
- **functionVal** (≈ 0) is the costFuntion value at optimum
- **exitFlag** (=1) shows convergence status.

0.3 Multiclass classification

Example of multiclass classification: email foldering/tagging

$$\begin{array}{rcccc} & \text{work} & \text{friends} & \text{family} & \text{hobby} \\ & y=1 & y=2 & y=3 & y=4 \\ \text{or} \rightarrow & y=0 & y=1 & y=2 & y=4 \end{array}$$



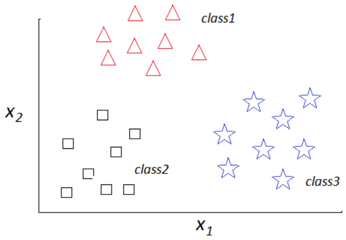
Binary Classification **Multiclass classification**

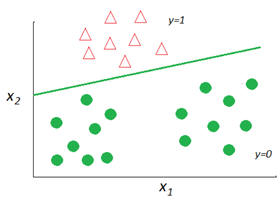
0.3.1 One-vs-all (one-vs-rest) classification

Transform a multiclass (k) problem into several (k) binary classes problems

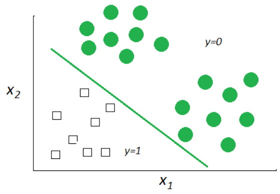
Principle: Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.
On a new input x, to make a prediction, pick the class i that maximizes $h_\theta^{(i)}(x) \Rightarrow \max_i h_\theta^{(i)}(x)$.
Run all three classifier on input x and then choose classifier given the larger value $(max_i h_\theta^{(i)}(x)$.

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \text{with i} = 1, 2, 3$$

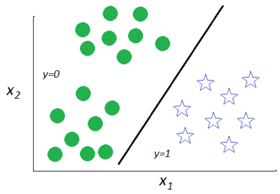




Classifier $h_\theta^{(1)}(x)$- class 1 is positive class

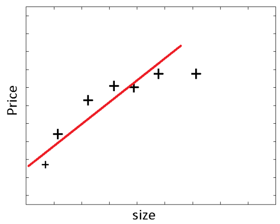


Classifier $h_\theta^{(2)}(x)$- class 2 is positive class



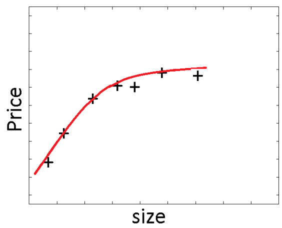
Classifier $h_\theta^{(3)}(x)$- class 3 is positive class

0.4 Underfitting(bias), Overfitting(variance)

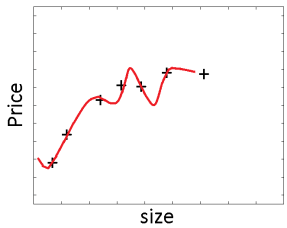


$h_\theta(x) = \theta_0 + \theta_1 x$

Underfitting/High Bias. There is a high bias of the model to fit the data to a linear despite the trend.



$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$



$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting/High Variance. There are too many features. The algorithm makes accurate predictions in the training example ($J(\theta) = 0$), but fails to generalize to new examples

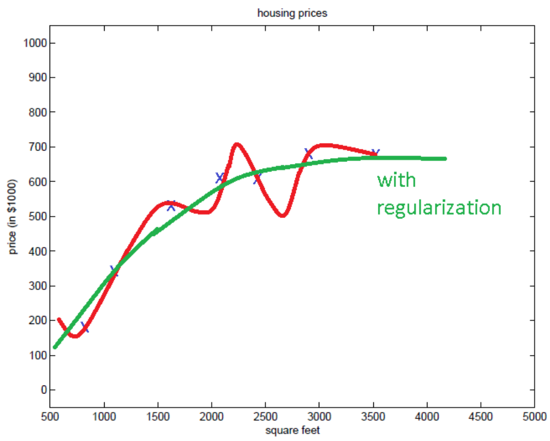
Bias/Variance also apply to logistic regression.

How to adress overfitting :

- Option1:
  - Reduce the number of features (select the more important features manually)
  - Model selection algorithm
- Option2: Regularization $\Rightarrow$ keep all the features but reduce magnitude of θ_j

0.4.1 Regularization

The goal is to reduce the values/amplitudes of the parameters θ_j (with j =1,2...n), to mitigate overfitting (overconfidence). Often overfitting si associated with very large coefficients θ



- There are several options for the regularization term
  - sum of squares (L_2 norm):

$$|| \, theta||_2 = \sum_{j=1}^{n} \theta_j^2 \tag{1}$$

  - sum of absolute values (L_1 norm):

$$||\theta||_1 = \sum_{j=1}^{n} |\theta_j| \tag{2}$$

- Features: $\begin{bmatrix} x_0 \\ x_1 \\ .. \\ .. \\ .. \\ x_n \end{bmatrix}$

- Parameters: $\begin{bmatrix} \theta_0 \\ \theta_1 \\ .. \\ .. \\ .. \\ \theta_n \end{bmatrix}$

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

λ is the regularization parameter. By convention, θ_0 is excluded from the regularization. If λ **is too large** ($\lambda = 10^{10}$), algorithm would result in **underfitting** (fails to fit even the training set): $(\theta_1, \theta_2, \theta_3 ...)$ parameters would be too much penalized $\approx 0 \Rightarrow \theta_1 x^{(1)} \approx \theta_2 x^{(2)} ... \approx 0$ and hence $h_\theta(x) = \theta_0$.

Regularized linear regression

- Cost function

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

  (θ_0 is excluded from the regularization term: $j = 1...n$).

- Gradient descent: $\min_\theta J(\theta)$ with $h_\theta(x) = \theta^{\mathrm{T}} x$

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}$$

......

......

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j$$

for $j = 1, 2, 3...n$

}

The equation of θ_j can be simplified:

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Normal equation

With $X \in \mathbb{R}^{m \times (n+1)}$ and $y \in \mathbb{R}^m$

$$\text{without regularization} : \theta = \left(X^{\mathrm{T}} X \right)^{-1} X^{\mathrm{T}} y$$

$$\text{with regularization} : \theta = \left(X^{\mathrm{T}} X + \lambda \begin{bmatrix} 0 & 0 & 0 & 0... & 0 \\ 0 & 1 & 0 & 0... & 0 \\ 0 & 0 & 1 & 0... & 0 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ 0 & 0 & 0 & 0.. & 1 \end{bmatrix} \right)^{-1} X^{\mathrm{T}} y$$

The matrix of (zeros and ones) is a $(n+1) \times (n+1)$.

Non-invertibility:

Suppose $m \leq n$ (# exples $\leq$ #features), then $(X^{\mathrm{T}} X)$ will be singular/non-invertible/degenerate (although MatLab can still provide a pseudo-inverse with '***pinv()***'.
If $\lambda > 0$, then $(X^{\mathrm{T}} X + \lambda M)$ -where M is the special matrix- is invertible: regularization makes the matrix invertible.

Regularized logistic Regression

- Cost function $J(\theta)$:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} y^{(i)} \mathrm{Log}(h_\theta(x^{(i)})) + (1 - y^{(i)}) \mathrm{Log}(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

- Gradient descent: $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Repeat until convergence {

$$
\begin{aligned}
\theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)} \\
\theta_j &:= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)} + \frac{\lambda}{m} \theta_j \\
&....j = 1, 2, 3...n
\end{aligned}
$$

}

With Advanced Optimization:

```
function [jVal, gradient] = costFunction(theta)
        jVal = [' code to compute J(theta) '];
        gradient = zeros(n+1, 1) #create a zero vector
        #and fill with gradient values
        gradient(1) = [' code to compute (d J(theta)/d theta_0) '];
        gradient(2) = [' code to compute (d J(theta)/d theta_1) '];
        .....
        .....
        gradient(n+1) = [' code to compute (d J(theta)/d theta_(n))     '];
```

where:

$$
J(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \mathrm{Log}\left(h_\theta(x^{(i)}) \right) + (1 - y^{(i)}) \mathrm{Log}\left(1 - h_\theta(x^{(i)}) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2
$$

$$
\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_0^{(i)}
$$

$$
\frac{\partial}{\partial \theta_1} J(\theta) = \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_1^{(i)} \right] + \frac{\lambda}{m} \theta_1
$$

0.5 Assignments

0.5.1 Visualizing the data

```
function plotData(X, y)
        figure;  # Create New Figure
        hold on;
        # X is the array [x0,x1,x2] of the m training examples
        #find indexes of training examples where y=1 (positive)
        pos = find(y==1);
        #find indexes of training examples where y=0 (negative)
```

```
        neg = find(y==0);

        m = length(y); #number of training examples
        # k+ = black plus, ko=black circles
        # MarkerSize= 7
        # Linewidth=2
        #MarkerFaceColor=yellow (filled circles)
        plot(X(pos,1), X(pos,2), 'k+', 'LineWidth', 2, 'MarkerSize', 7);
        plot(X(neg,1), X(neg,2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7);
end
```

0.5.2 Create the sigmoid() function

```
function g = sigmoid(z)
        # return g the sigmoid of z
        # z can be a vector, a matrix or a scalar
        g = zeros(size(z));
        #Take the exponential of z and add 1
        # 1./(...) do an inverse element wise.
        g = 1./(1 + exp(-z))
end
```

0.5.3 cost function and gradient

The cost function in logistic regression is defined by:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\mathrm{Log}\left(h_\theta(x^i)\right) - (1-y^{(i)})\mathrm{Log}\left(1-h_\theta(x^i)\right)\right]$$

and the gradient (a vector of same length as θ where $j = 0, ..., n$.

$$\frac{\partial J(\theta)}{\partial\theta} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^i) - y^{(i)}\right)x_j^{(i)}$$

```
function [J, grad] = costFunction(theta, X, y)
        m = length(y); # number of training examples
        J = 0; #initial value of cost function
        grad = zeros(size(theta)); #initial matrix for gradient
        #co is the item inside the sum of J
        #h(x) = sigmoid(X*theta)
        co = (-y.*log(sigmoid(X*theta)))-(1-y).*log(1-sigmoid(X*theta));
        J = 1/m * sum(co) # J is a scalar
        #grad is a vector matrix with X'=transpose(X)
        grad = 1/m*X'*(sigmoid(X*theta) - y)
end
```

0.5.4 Plot the decision boundary

```
function plotDecisionBoundary(theta, X, y)
%PLOTDECISIONBOUNDARY Plots the data points X and y into a new figure with
%the decision boundary defined by theta
%   PLOTDECISIONBOUNDARY(theta, X,y) plots the data points with + for the
%   positive examples and o for the negative examples. X is assumed to be
%   a either
%   1) Mx3 matrix, where the first column is an all-ones column for the
%      intercept.
%   2) MxN, N>3 matrix, where the first column is all-ones

% Plot Data
plotData(X(:,2:3), y);
hold on

if size(X, 2) <= 3
    % Only need 2 points to define a line, so choose two endpoints
    plot_x = [min(X(:,2))-2,  max(X(:,2))+2];

    % Calculate the decision boundary line
    plot_y = (-1./theta(3)).*(theta(2).*plot_x + theta(1));

    % Plot, and adjust axes for better viewing
    plot(plot_x, plot_y)

    % Legend, specific for the exercise
    legend('Admitted', 'Not admitted', 'Decision Boundary')
    axis([30, 100, 30, 100])
else
    % Here is the grid range
    u = linspace(-1, 1.5, 50);
    v = linspace(-1, 1.5, 50);

    z = zeros(length(u), length(v));
    % Evaluate z = theta*x over the grid
    for i = 1:length(u)
        for j = 1:length(v)
            z(i,j) = mapFeature(u(i), v(j))*theta;
        end
    end
    z = z'; % important to transpose z before calling contour

    % Plot z = 0
    % Notice you need to specify the range [0, 0]
    contour(u, v, z, [0, 0], 'LineWidth', 2)
end
hold off

end
```

0.5.5 Predict label

This function will predict the label (0 or 1) of based on the learned logistic regression parameters theta.

```
function p = predict(theta, X)
m = size(X, 1); # Number of training examples
p = zeros(m, 1);

# calculate sigmoid for all examples p is a vector matrix of size mx1
p = sigmoid(X*theta);
for student = 1:m
    if p(student) >= 0.5
        p(student)=1;
    else
        p(student)=0;
    end
end
```

0.5.6 Learning parameters using *fminunc()*

```
#initialization
#clear the variables, close all figs, clc=clear command window
clear ; close all; clc

# Load Data : col1=exam1, col2=exam2, col3=label(1 or 0)
data = load('ex2data1.txt');
X = data(:, [1, 2]); y = data(:, 3);

#===================== Part 1: Plotting =====================
fprintf(['Plotting data with + indicating (y = 1) examples ...
and o indicating (y = 0) examples.\n']);
plotData(X, y);
#retains plots in the current axes so that
#new plots added to the axes do not delete existing plots
hold on;
# Labels and Legend
xlabel('Exam 1 score')
ylabel('Exam 2 score')
legend('Admitted', 'Not admitted')
hold off;

#============ Part 2: Compute Cost and Gradient ============
# Setup the data matrix appropriately, and add ones for the intercept term
[m, n] = size(X);
#Add intercept term to x and X_test
X = [ones(m, 1) X];

# Initialize fitting parameters
initial_theta = zeros(n + 1, 1);
```

```
# Compute and display initial cost and gradient
[cost, grad] = costFunction(initial_theta, X, y);

fprintf('Cost at initial theta (zeros): \%f\n', cost);
fprintf('Gradient at initial theta (zeros): \n');
fprintf(' \%f \n', grad);

#============= Part 3: Optimizing using fminunc  =============
#  Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 100);

# Run fminunc to obtain the optimal theta
#\@(t) creates a function with argument t which calls the costFunction
[theta, cost, exit_flag] = fminunc(@(t)(costFunction(t, X, y)), ...
initial\_theta, options);

% Print theta to screen
fprintf('Cost at theta found by fminunc: \%f\n', cost);
fprintf('theta: \n');
fprintf(' \%f \n', theta);

# Plot Boundary
plotDecisionBoundary(theta, X, y);
hold on;
xlabel('Exam 1 score')
ylabel('Exam 2 score')
legend('Admitted', 'Not admitted')

#============= Part 4: Predict and Accuracies: predict() =============
#  After learning the parameters, you'll like to use it to predict the outcomes
#  on unseen data. In this part, you will use the logistic regression model
#  to predict the probability that a student with score 45 on exam 1 and
#  score 85 on exam 2 will be admitted.

prob = sigmoid([1 45 85] * theta);
fprintf(['For a student with scores 45 and 85, we predict an admission ' ...
         'probability of %f\n\n'], prob);
# Compute accuracy on our training set
p = predict(theta, X);

fprintf('Train Accuracy: \%f\n', mean(double(p == y)) * 100);
```

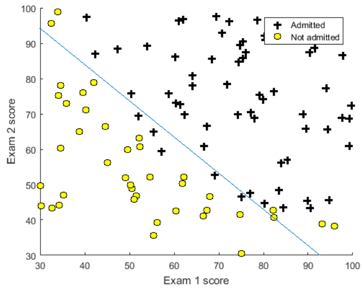


Figure 2: Training data with precision boundary

Appendices

.1 Logistic Regression - University of Washington Coursera

We define $l(\theta)$ as the likelihood of θ: it is a measure of the quality of the fit of the model to the training examples. Maximizing the likelihood over all possible θ will ensure to give the best classifier. For example, for 1 data, we pick a θ value that maximizes $p(y=+1|x,\theta)=\frac{1}{1+e^{-\theta^T x}}$.

- For m examples, the likelihood is the product of the probability (assuming that the examples are independent of each other):

$$l(\theta)=\prod_{i=1}^{m} p(y^{(i)}|x^{(i)},\theta)$$

In order to simplify the mathematics, w use Log$l(\theta)$ (the Log does not change the maxima), hence:

$$\text{Log}(l(\theta))=\sum_{i=1}^{m}\text{Log}(p(y^{(i)}|x^{(i)},\theta))$$

We introduce the indicator function:

$$\mathbb{1}(y^{(1)}=+1)\begin{cases}+1 \text{ if } y^{(1)}\\ -1 \text{ if } y^{(1)}=-1\end{cases}\tag{3}$$

For one example:

$$\begin{aligned}\text{Log}(l(\theta)) &= \mathbb{1}(y^{(1)}=+1)P(y^{(1)}=+1|x^{(i)},\theta)+\not{\mathbb{1}}(y^{(1)}=-1)P(y^{(1)}=-1|x^{(i)},\theta)\\ &= \mathbb{1}(y^{(1)}=+1)P(y^{(1)}=+1|x^{(i)},\theta)+\mathbb{1}(y^{(1)}=-1)(1-P(y^{(1)}=+1|x^{(i)},\theta)\\ &= \mathbb{1}(y^{(1)}=+1)P(y^{(1)}=+1|x^{(i)},\theta)+\mathbb{1}(y^{(1)}=-1)\text{Log}\left(1-\frac{1}{1+e^{-\theta^T x}}\right)\\ &= \mathbb{1}(y^{(1)}=+1)P(y^{(1)}=+1|x^{(i)},\theta)+\mathbb{1}(y^{(1)}=-1)\text{Log}\left(\frac{1+e^{-\theta^T x}-1}{1+e^{-\theta^T x}}\right)\\ &= \mathbb{1}(y^{(1)}=+1)P(y^{(1)}=+1|x^{(i)},\theta)+\mathbb{1}(y^{(1)}=-1)\text{Log}\left(\frac{e^{-\theta^T x}}{1+e^{-\theta^T x}}\right)\end{aligned}\tag{4}$$

- Gradient:

$$\begin{aligned}\frac{\partial\text{Log}}{\partial\theta_j} &= -(1-\mathbb{1}(y^{(1)}=+1)\frac{\partial\theta^T x}{\partial\theta_j}-\frac{\partial}{\partial\theta_j}\text{Log}\left(1-e^{-\theta^T x}\right)\\ &= -(1-\mathbb{1}(y^{(1)}=+1)x_j-\frac{-\theta_j e^{-\theta^T x}}{1-e^{-\theta^T x}})\\ &= -(1-\mathbb{1}(y^{(1)}=+1)x_j+x_j P(y^{(1)}=-1|x,\theta))\\ &= x_j\left(\mathbb{1}(y^{(1)}=+1)-P(y^{(1)}=+1|x,\theta)\right)\end{aligned}\tag{5}$$

For all examples m:

$$\frac{\partial\text{Log}}{\partial\theta_j}=\sum_{i=1}^{N}x_j\left(\mathbb{1}(y^{(1)}=+1)-P(y^{(1)}=+1|x,\theta\right)\tag{6}$$