Logistic Regression

Andrew Ng

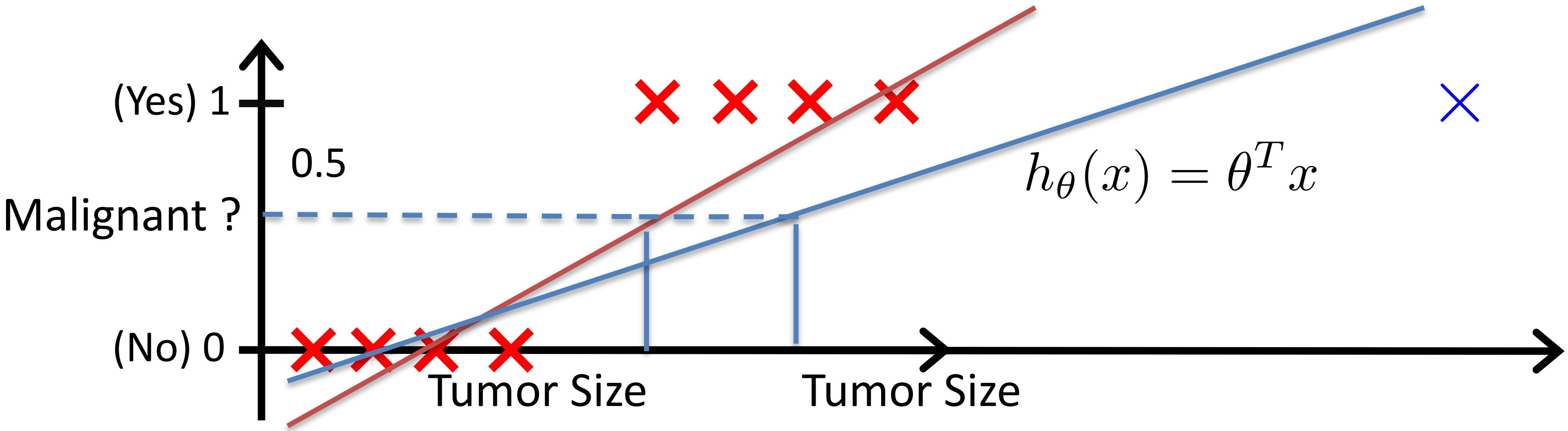
Classification

Email: Spam / Not Spam?
Online Transactions: Fraudulent (Yes / No)?
Tumor: Malignant / Benign ?

$$y \in \{0, 1\}$$

0: “Negative Class” (e.g., benign tumor)
1: “Positive Class” (e.g., malignant tumor)

$$y \in \{0, 1, 2, 3, ..., n\}$$



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

Linear regression does not work even with a threshold output

Classification: y = 0 or 1

In linear regression $h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Logistic Regression

Hypothesis representation

Logistic Regression Model

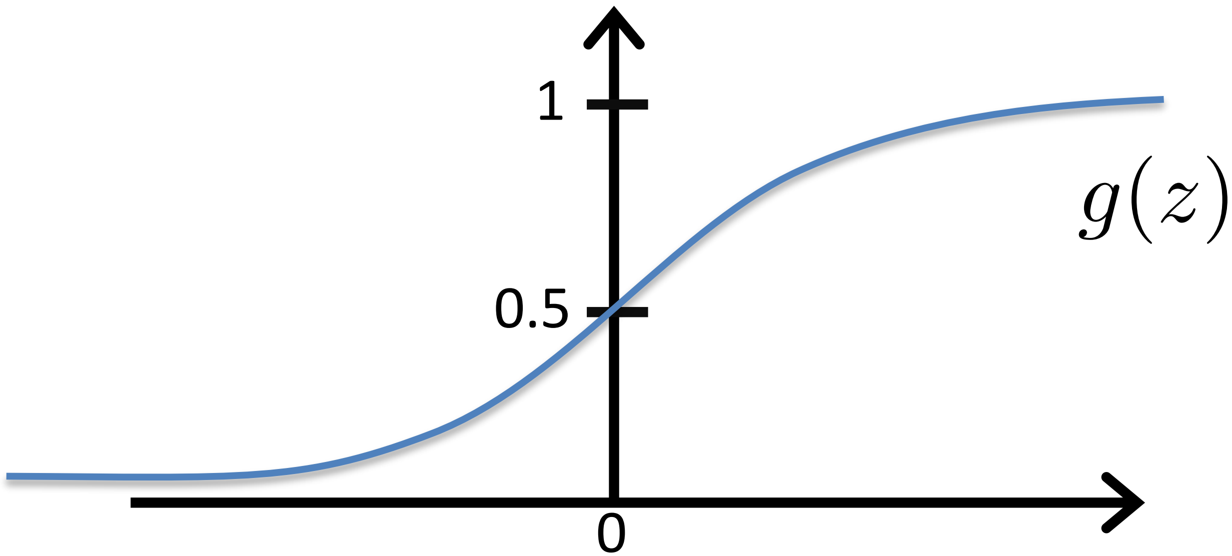
Want $0 \leq h_\theta(x) \leq 1$

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

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

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

Sigmoid function
Logistic function



Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$$h_\theta(x) = 0.7$$

Tell patient that 70% chance of tumor being malignant

$h_\theta(x) = P(y = 1|x;\theta)$ "probability that y = 1, given x, parameterized by θ"

$$P(y = 0|x;\theta) + P(y = 1|x;\theta) = 1$$
$$P(y = 0|x;\theta) = 1 - P(y = 1|x;\theta)$$

Logistic Regression
Decision boundary

Logistic Regression

$$h_\theta(x) = g(\theta^T x) = p(y=1|x;\theta)$$

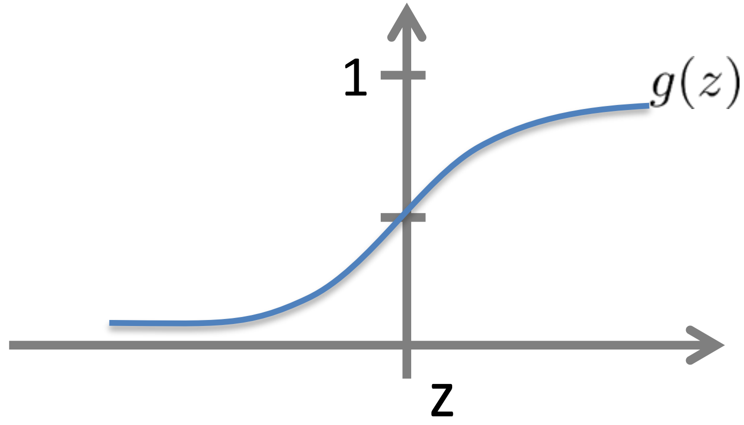
$$g(z) = \frac{1}{1+e^{-z}}$$

Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

$$h_\theta(x) = g(\theta^T x) \geq 0.5 \iff \theta^T x \geq 0$$

predict "$y = 0$" if $h_\theta(x) < 0.5$

$$h_\theta(x) = g(\theta^T x) < 0.5 \iff \theta^T x < 0$$

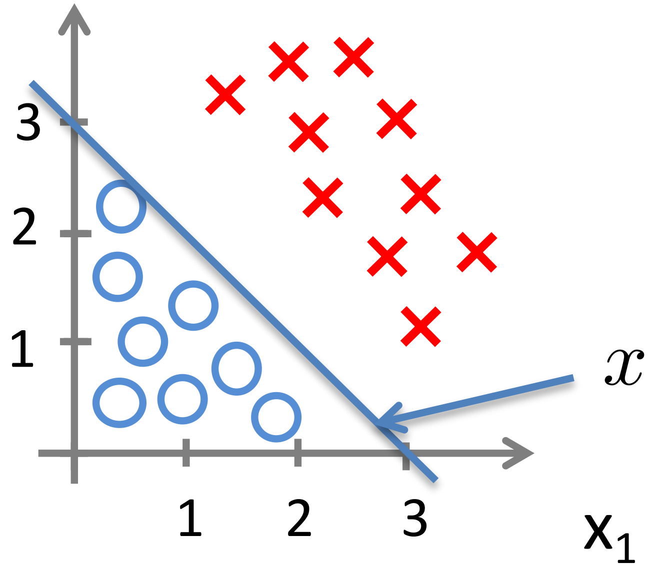


$$g(z) \geq 0.5 \iff z \geq 0$$

$$g(z) < 0.5 \iff z < 0$$

Decision Boundary

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

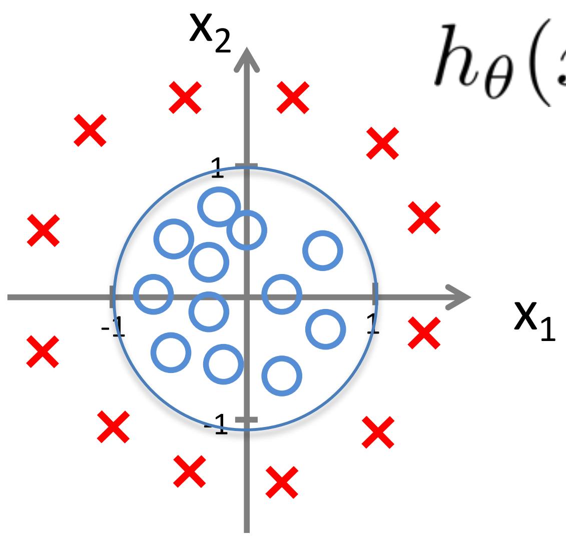


$$x_1 + x_2 = 3$$

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $\underbrace{-3 + x_1 + x_2}_{\theta^T x} \geq 0$

Non-linear decision boundaries

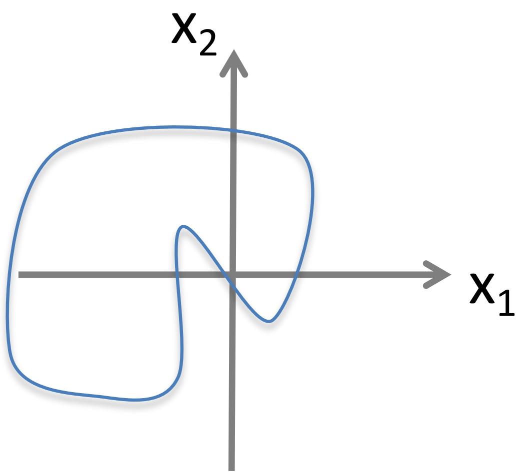


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$$x_1^2 + x_2^2 = 1$$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Logistic Regression

Cost function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

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

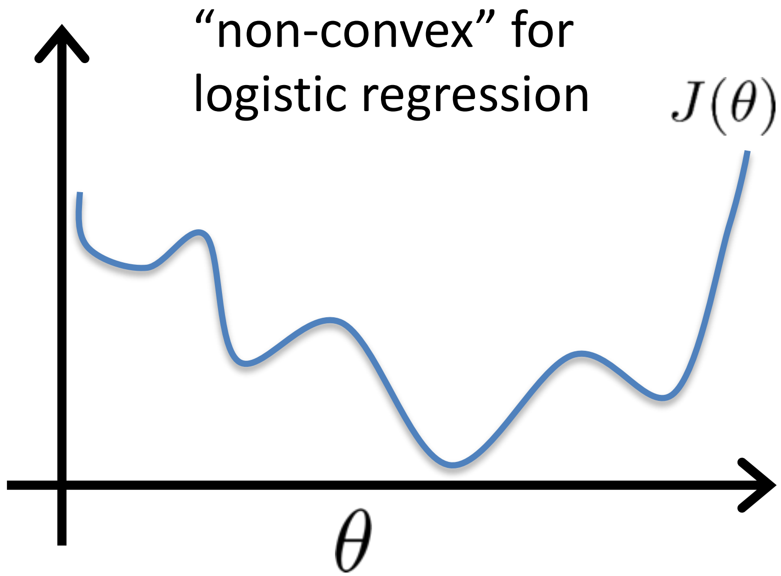
How to choose parameters θ ?

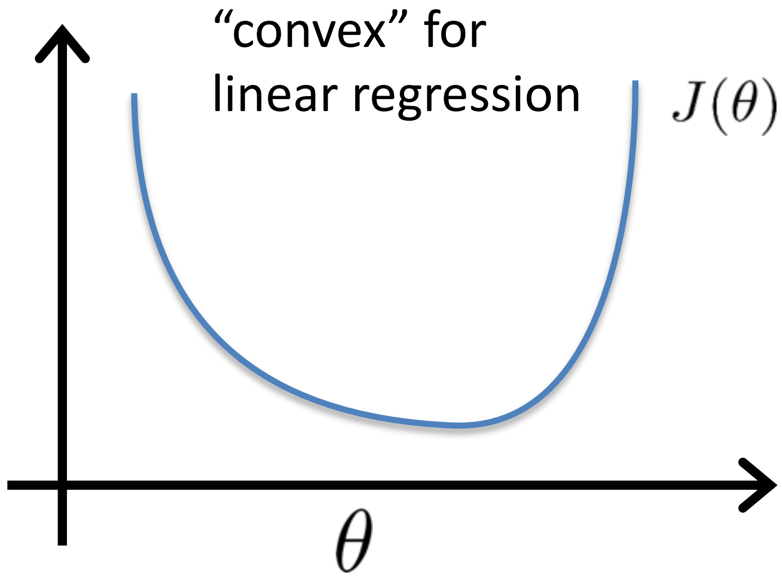
Cost function

$$cost(h_\theta(x^{(i)}), y^{(i)})$$

Linear regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

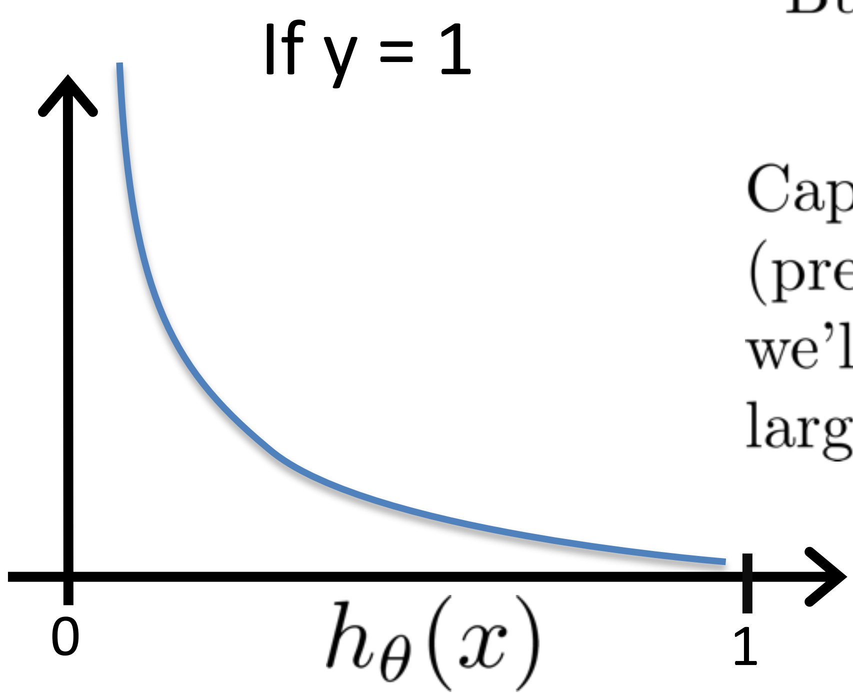
$$\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$





Logistic regression cost function

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



$\mathrm{Cost} = 0$ if $y = 1, h_\theta(x) = 1$

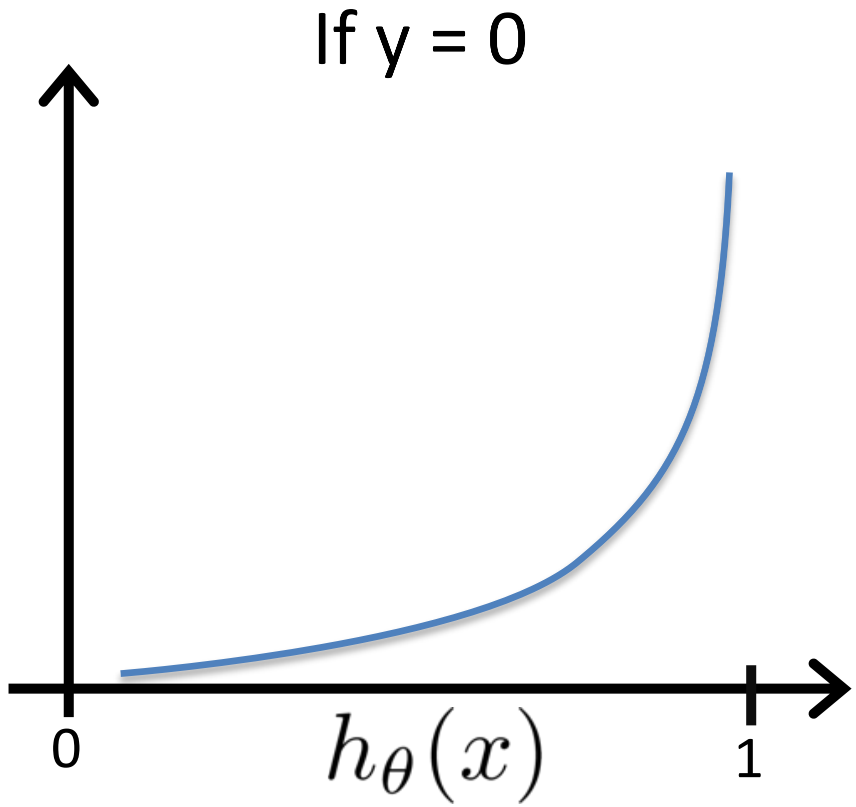
But as $h_\theta(x) \to 0$

$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Logistic regression cost function

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



Logistic Regression

Simplified cost function and gradient descent

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

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

Note: $y = 0$ or 1 always

$$Cost(h_\theta(x), y) = -y \times log(h_\theta(x)) - (1 - y) \times log(1 - h_\theta(x))$$

Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

To fit parameters θ:

$$\min_\theta J(\theta)$$

To make a prediction given new x:

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

Gradient descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

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

Gradient descent

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

Algorithm looks identical to linear regression!

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

vs.

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

Logistic Regression

Advance optimization

Optimization algorithm

Cost function $J(\theta)$. Want $\min_\theta J(\theta)$.

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial\theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$)

Gradient descent:

Repeat $\{$

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

$\}$

Optimization algorithm

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial\theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

scipy.optimize.fmin_tnc

scipy.optimize.fmin_tnc(*func*, *x0*, *fprime=None*, *args=()*, *approx_grad=0*, *bounds=None*, *epsilon=1e-08*, *scale=None*, *offset=None*, *messages=15*, *maxCGit=-1*, *maxfun=None*, *eta=-1*, *stepmx=0*, *accuracy=0*, *fmin=0*, *ftol=-1*, *xtol=-1*, *pgtol=-1*, *rescale=-1*, *disp=None*, *callback=None*) [source]

Minimize a function with variables subject to bounds, using gradient information in a truncated Newton algorithm. This method wraps a C implementation of the algorithm.

Parameters:

func : *callable* `func(x, *args)`

Function to minimize. Must do one of:

1. Return f and g, where f is the value of the function and g its gradient (a list of floats).
2. Return the function value but supply gradient function separately as *fprime*.
3. Return the function value and set `approx_grad=True`.

If the function returns None, the minimization is aborted.

x0 : *array_like*

Initial estimate of minimum.

fprime : *callable* `fprime(x, *args)`

Gradient of *func*. If None, then either *func* must return the function value and the gradient (`f,g = func(x, *args)`) or *approx_grad* must be True.

args : *tuple*

Arguments to pass to function.

Returns:

x : *ndarray*

The solution.

nfeval : *int*

The number of function evaluations.

rc : *int*

Return code as defined in the RCSTRINGS dict.

See also:

minimize Interface to minimization algorithms for multivariate functions. See the 'TNC' *method* in particular.

Notes

The underlying algorithm is truncated Newton, also called Newton Conjugate-Gradient. This method differs from scipy.optimize.fmin_ncg in that

1. It wraps a C implementation of the algorithm
2. It allows each variable to be given an upper and lower bound.

scipy.optimize.minimize(*fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None*) [source]

Minimization of scalar function of one or more variables.

New in version 0.11.0.

Parameters:

fun : *callable*

Objective function.

x0 : *ndarray*

Initial guess.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (Jacobian, Hessian).

method : *str or callable, optional*

Type of solver. Should be one of

- 'Nelder-Mead'
- 'Powell'
- 'CG'
- 'BFGS'
- 'Newton-CG'
- 'Anneal (deprecated as of scipy version 0.14.0)'
- 'L-BFGS-B'
- 'TNC'
- 'COBYLA'
- 'SLSQP'
- 'dogleg'
- 'trust-ncg'
- custom - a callable object (added in version 0.14.0)

If not given, chosen to be one of `BFGS`, `L-BFGS-B`, `SLSQP`, depending if the problem has constraints or bounds.

Example:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$$

$$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
defun cost(theta):
   return (theta[0]-5)**2 +
          (theta[1]-5)**2

defun grad(theta):
  gradient = np.zeros(2);
  gradient[0] = 2*(theta[0]-5)
  gradient[1] = 2*(theta[1]-5)
  return gradient
```

```
initialTheta = np.zeros(2)
result = opt.fmin_tnc(func=cost, x0=initialTheta,
                      fprime=grad)
print(cost(result[0]))
```

$$\texttt{theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

```
def grad(theta, X, Y)
      gradient[0] =   code to compute ∂/∂θ0 J(θ)
      gradient[1] =   code to compute ∂/∂θ1 J(θ)
        ⋮
      gradient[n] =   code to compute ∂/∂θn J(θ)
      return gradient
```

$$\texttt{gradient[0]} = \text{code to compute } \frac{\partial}{\partial \theta_0} J(\theta)$$

$$\texttt{gradient[1]} = \text{code to compute } \frac{\partial}{\partial \theta_1} J(\theta)$$

$$\texttt{gradient[n]} = \text{code to compute } \frac{\partial}{\partial \theta_n} J(\theta)$$

```
result = opt.fmin_tnc(func=cost, x0=theta,
                      fprime=grad, args=(X, Y))
```

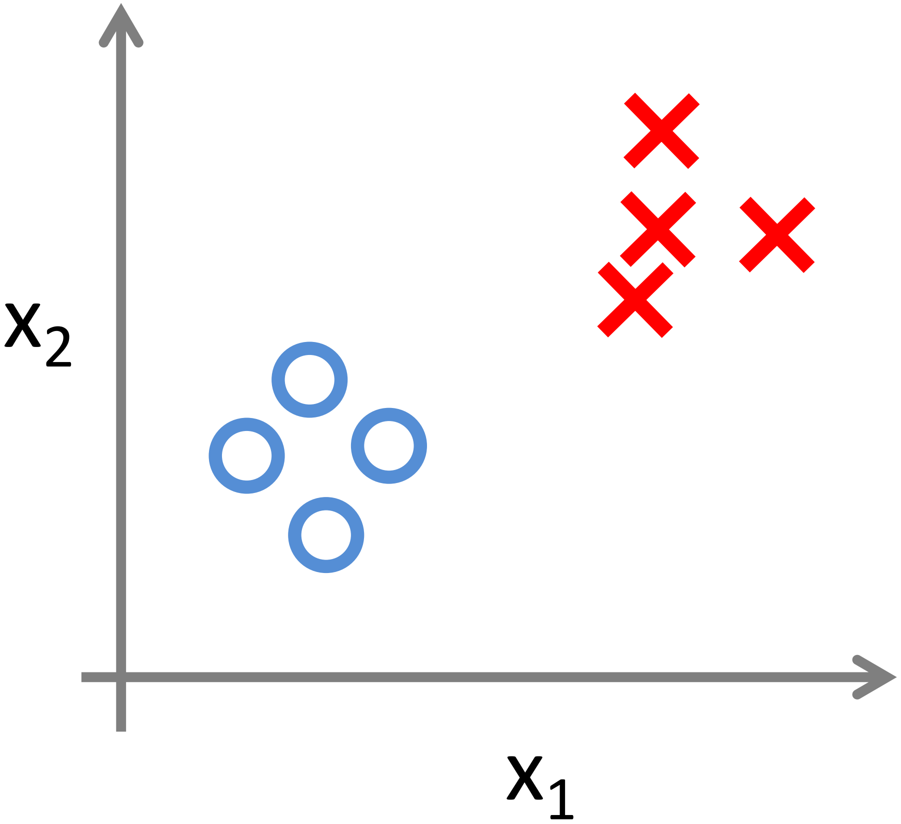
Logistic Regression

Multi-class classification: One-vs-all

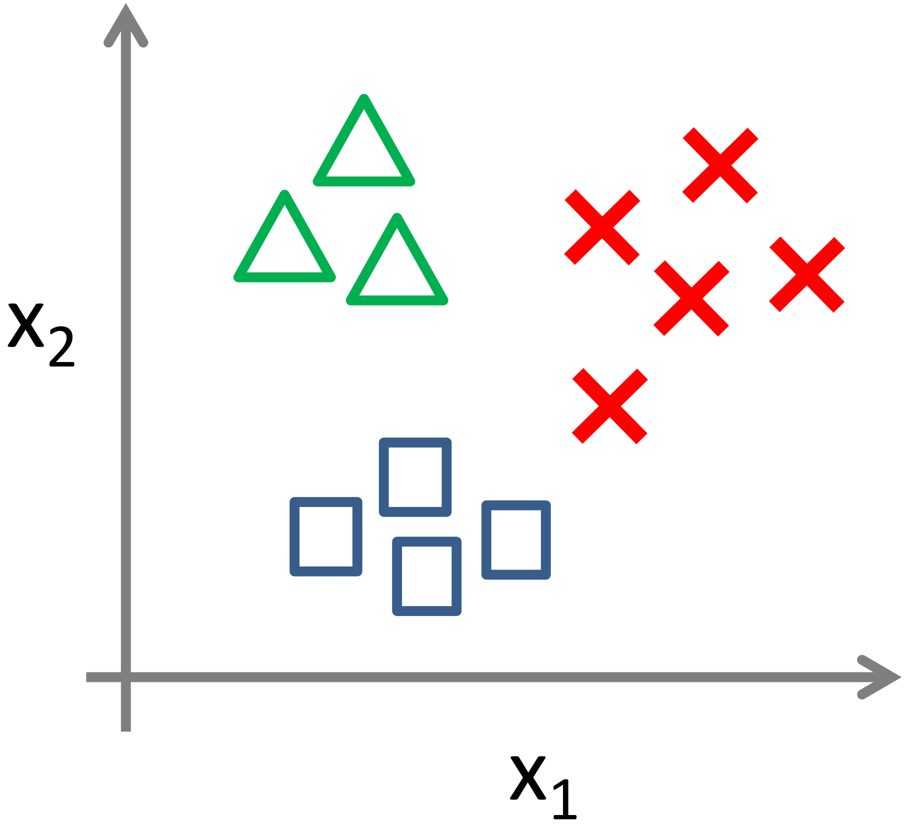
Multiclass classification

- Email foldering/tagging: Work, Friends, Family, Hobby
- Weather: Sunny, Cloudy, Rain, Snow
- Medical diagrams: Not ill, Cold, Flu

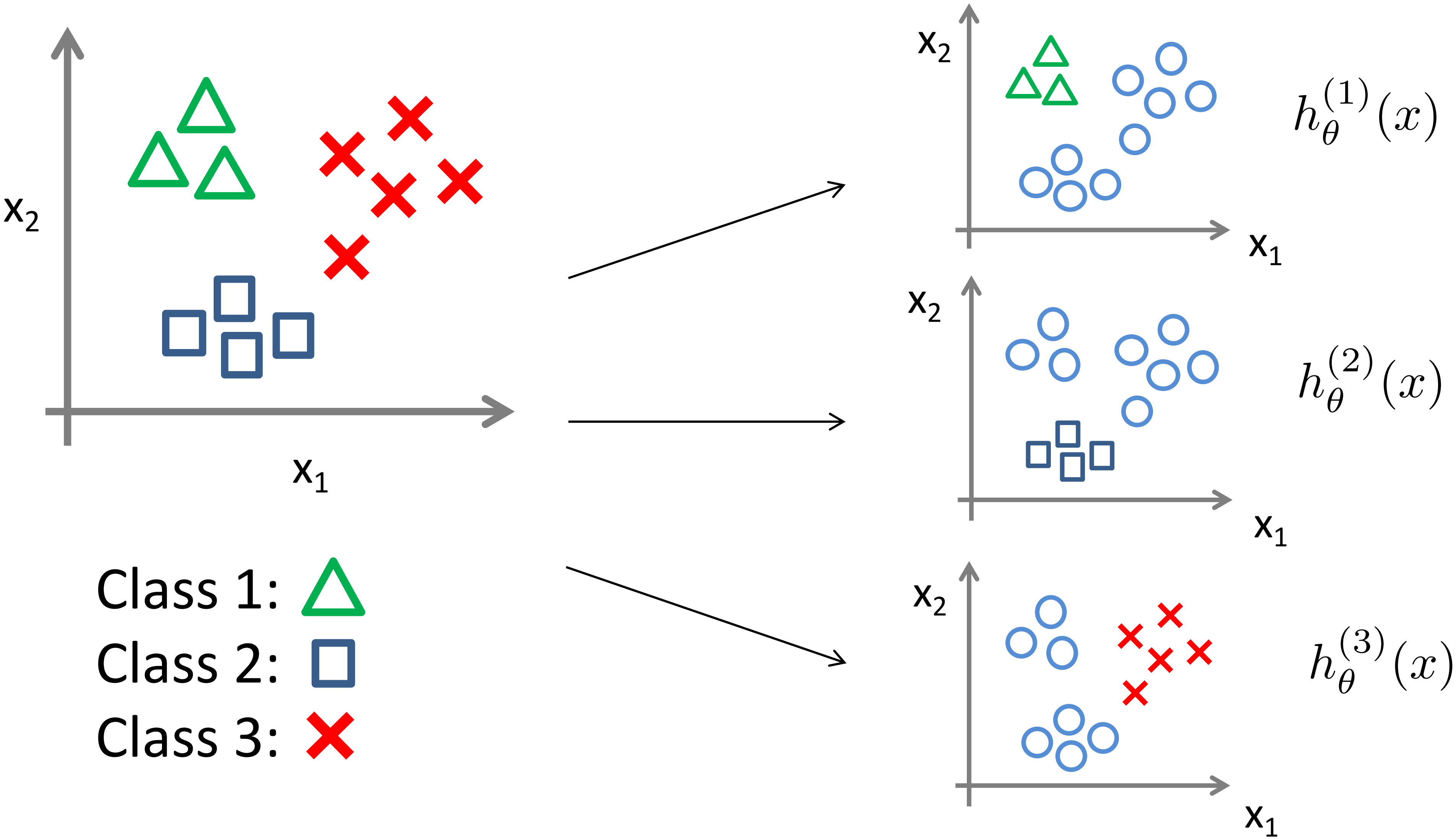
Binary classification:



Multi-class classification:



One-vs-all (one-vs-rest):



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

One-vs-all

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

$$\max_i h_\theta^{(i)}(x)$$