Machine Learning – COMS3007

Logistic Regression

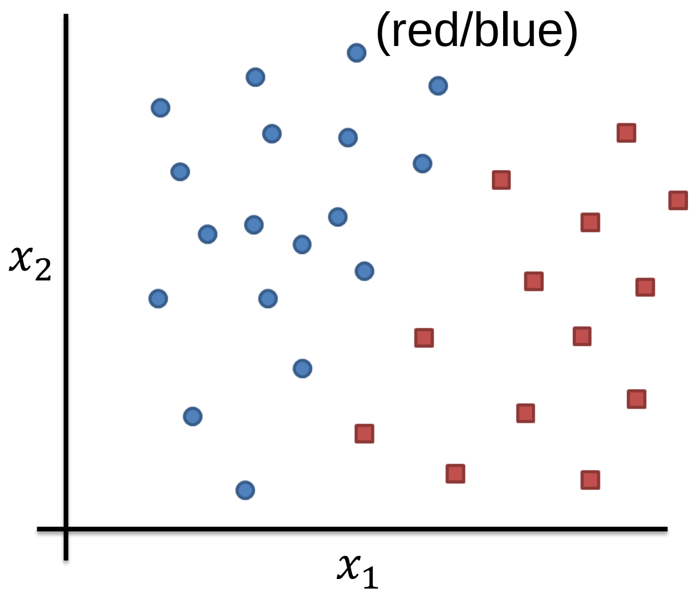
Benjamin Rosman

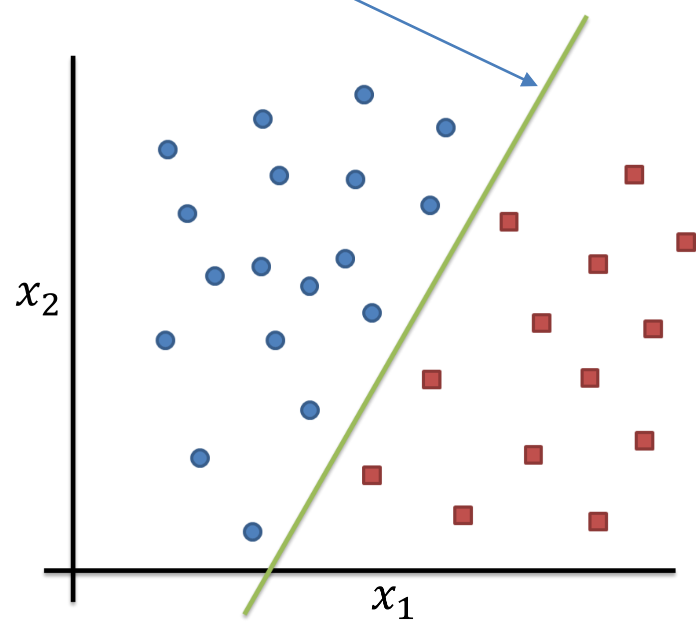
Based heavily on course notes by Chris Williams and Victor Lavrenko, Amos Storkey, Eric Eaton, and Clint van Alten

Classification

- Data $X = \{x^{(0)}, \ldots, x^{(n)}\}$, where $x^{(i)} \in R^d$
- Labels $\mathbf{y} = \{y^{(0)}, \ldots, y^{(n)}\}$, where $y^{(i)} \in \{0,1\}$
- Want to learn function $y = f(x, \theta)$ to predict y for a new x

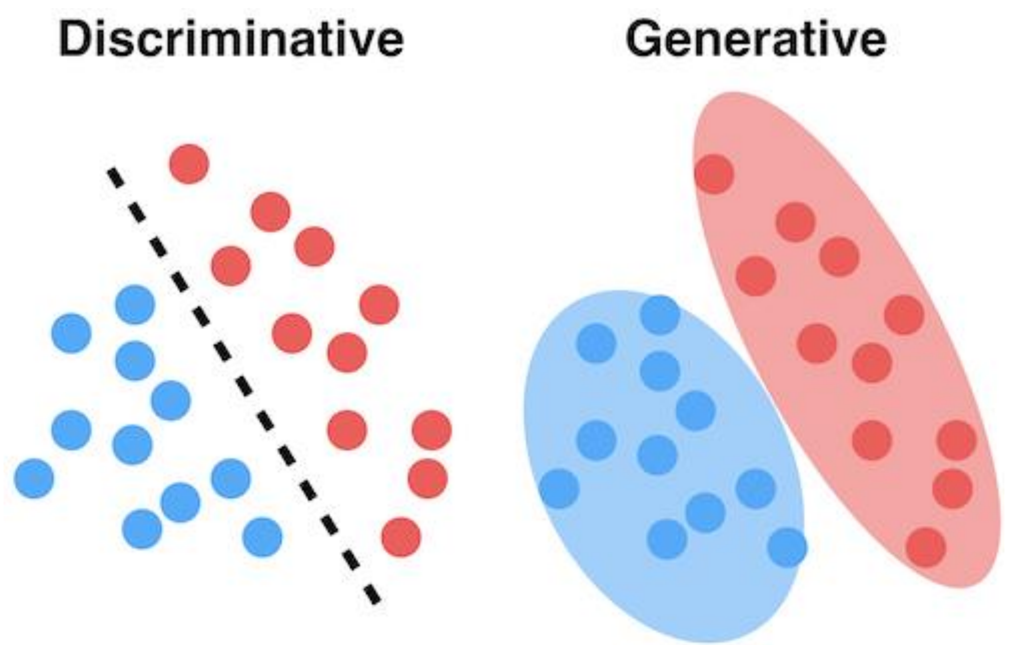
y is the class (red/blue)





Generative vs discriminative

- In Naïve Bayes, we used a generative approach
  - Class conditional modeling
  - $p(y|\boldsymbol{x}) \propto p(\boldsymbol{x}|y)p(y)$
- Now model $p(y|\boldsymbol{x})$ directly: discriminative approach
  - As was the case in decision trees
  - Don’t model $p(x)$
- Discriminative:
  - Can’t generate data
  - Often better
    - Fewer variables
- Both are correct

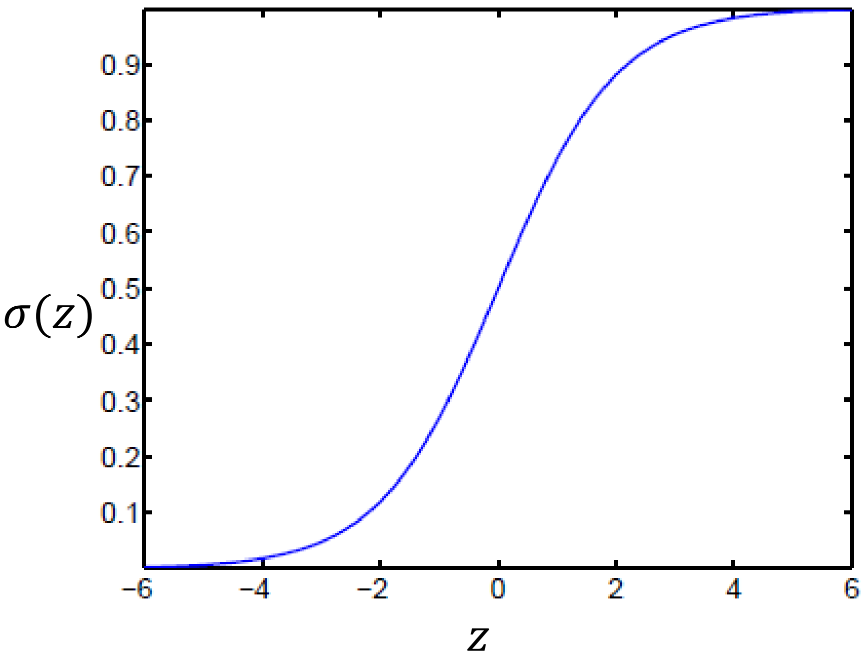


Two class discrimination

- Consider two classes: $y \in \{0,1\}$
- We could use linear regression
  - Doesn't perform well
  - Values < 0 or > 1 don't make sense
- We want a model of the form:
  - $P(y = 1|x) = f(x;\theta)$
- It is a probability, so $0 \leq f \leq 1$
- Also, probabilities sum to 1, so
  - $P(y = 0|x) = 1 - f(x;\theta)$
- What form should we use for f?

The logistic function

- We need a function that gives probabilities: $0 \leq f \leq 1$
- Logistic function
- $f(z) = \sigma(z) = \frac{1}{1+\exp(-z)}$
- "Sigmoid function"
    - S-shape
- "Squashing function"
    - As z goes from $-\infty$ to ∞
    - f goes from 0 to 1



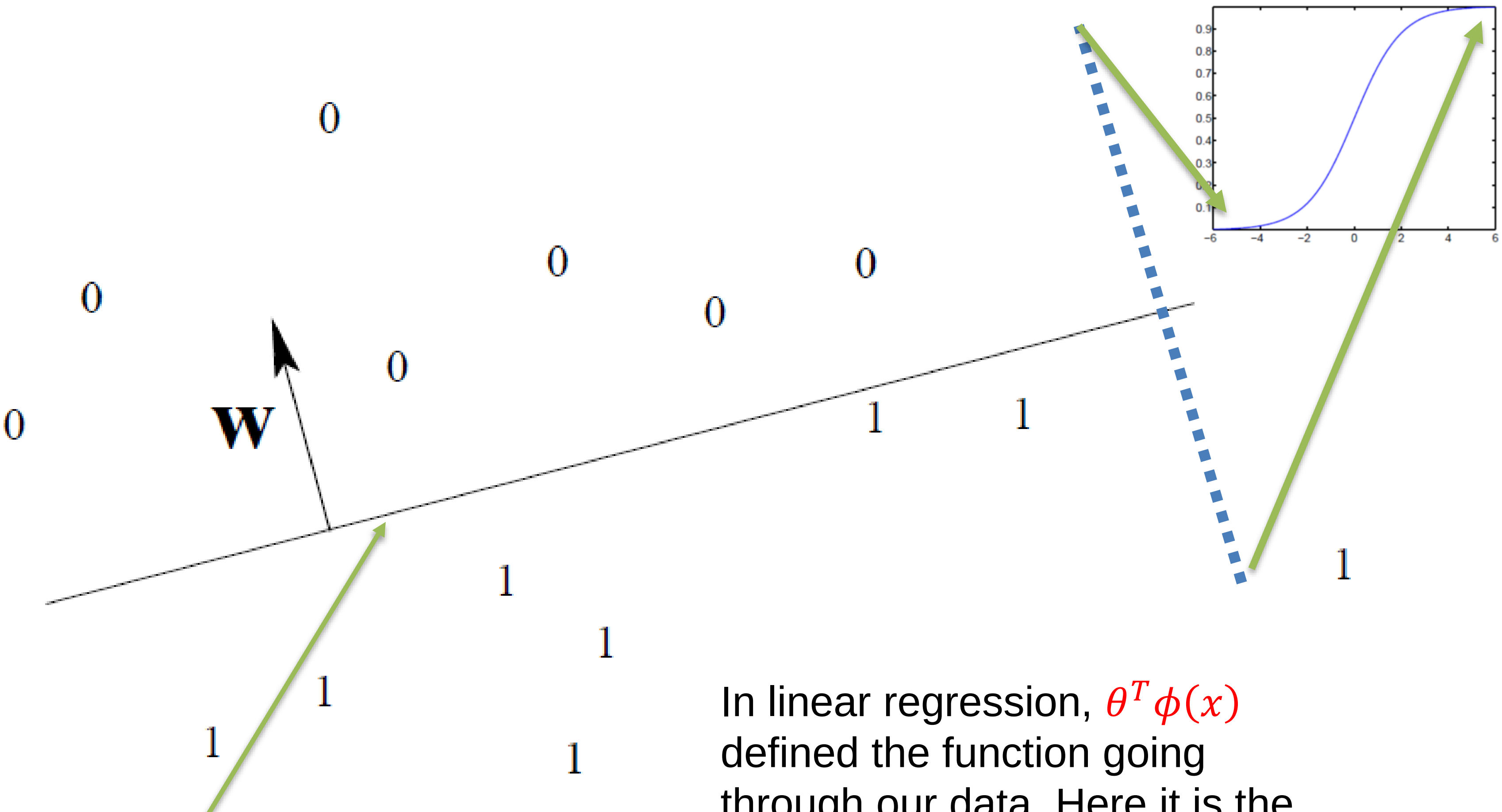
- Notes:
    - $\sigma(0) = 0.5$: "decision boundary"
    - $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

–ve values of z → class 0
+ve values of z → class 1

Linear weights

- Now we need a way of incorporating features x and parameters/weights θ
- Use the same idea of a linear weighting scheme from linear regression
- $p(y = 1|x) = \sigma(\theta^T \phi(x))$
  - θ is a vector of parameters
  - $\phi(x)$ is the vector of features
- Decision boundary: $\sigma(z) = 0.5$ when $z = 0$
  - So: decision boundary = $\theta^T \phi(x) = 0$
  - For an M dimensional problem, boundary is M-1 dimensional hyperplane

Linear decision boundary



decision boundary = $\theta^T \phi(x) = 0$

In linear regression, $\theta^T \phi(x)$ defined the function going through our data. Here it is the function separating our classes.

Cost function

- So:
  - $p(y=1|x;\theta) = \sigma\left(\theta^T \phi(x)\right) = h_\theta(x)$
  - $p(y=0|x;\theta) = 1 - h_\theta(x)$
- Write this more compactly as:

  - $p(y|x;\theta) = \left(h_\theta(x)\right)^y \left(1 - h_\theta(x)\right)^{1-y}$

Why?
What happens
when y=0?
And y=1?

- Likelihood of m data points:
  - $L(\theta) = \prod_{i=1}^{m} p\left(y^{(i)} \middle| x^{(i)};\theta\right)$
    - $= \prod_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}$

Cost function

- Likelihood of m data points:
  - $L(\theta) = \prod_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}$
- Take the log of the likelihood:
  - $l(\theta) = \log L(\theta)$
    - $= \sum_{i=1}^{m} y^{(i)} \log(h_\theta\left(x^{(i)}\right)) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)$
- We need to **maximise** the log likelihood
- Equivalent to **minimising** $E(\theta) = -l(\theta)$
- Cannot use a closed form solution

Regularisation

- Just as in linear regression, regularisation is useful here
  - Penalise the weights for growing too large
  - Note: the higher the weights, the "steeper" the S – so this stops the model becoming over-confident

- $\min_{\theta} E(\theta)$ where
- $E(\theta) =$

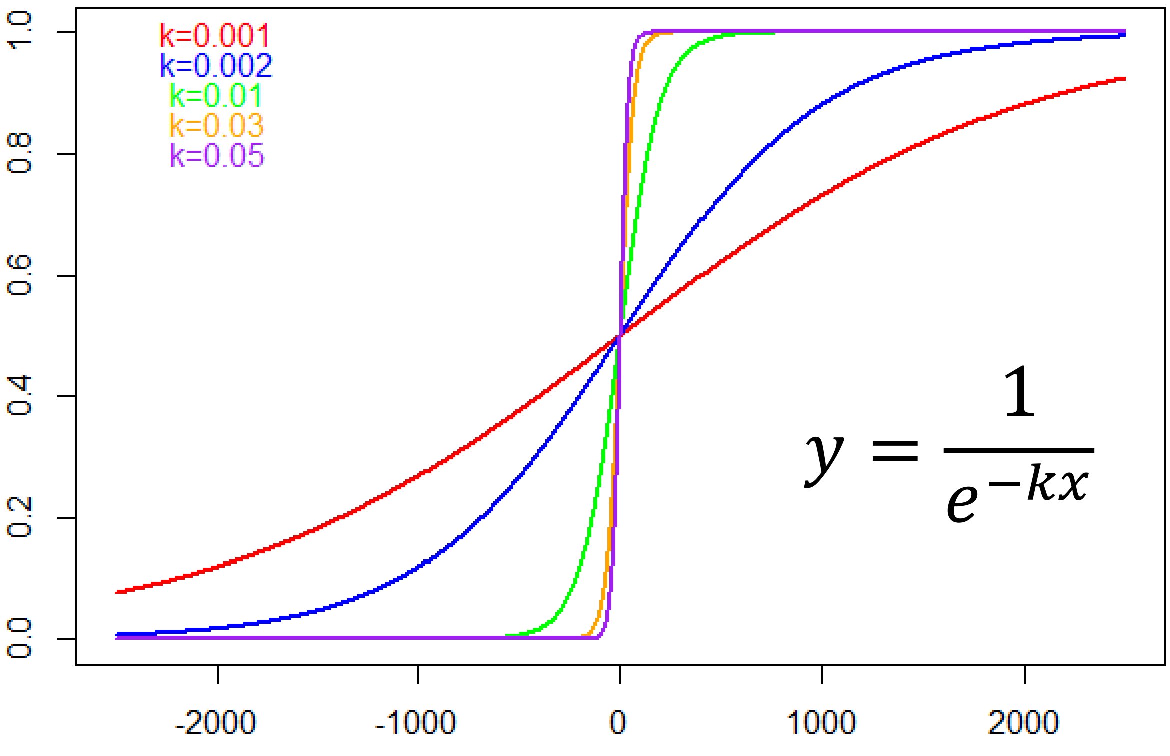
$$-\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)}) + (1-y^{(i)})\log\left(1-h_\theta(x^{(i)})\right)$$

$$+\lambda\sum_{j=1}^{d}\theta_j^2$$

λ = strength of regularisation

Regularisation

- Note: the higher the weights, the "steeper" the S – so regularisation stops the model becoming over-confident

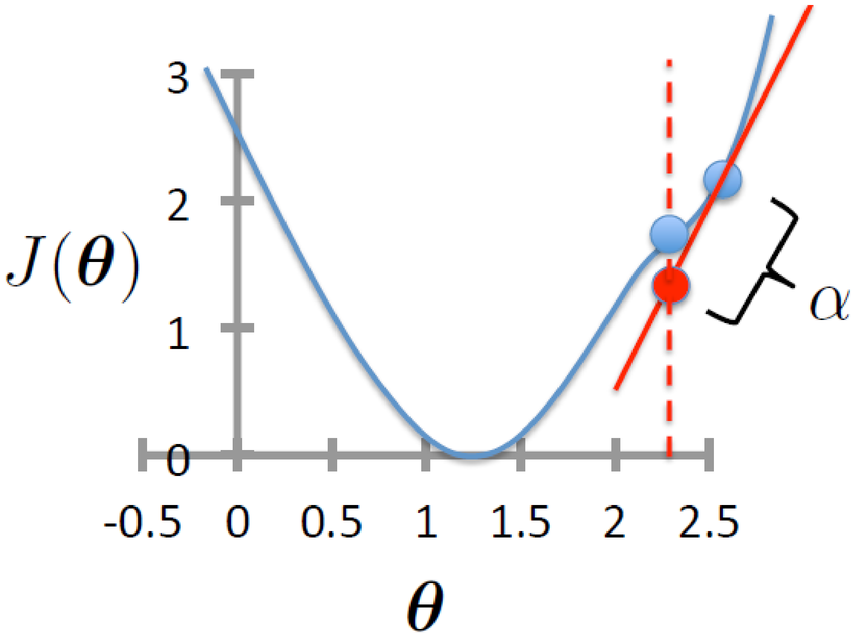


$$y = \frac{1}{e^{-kx}}$$

Gradient descent (again)

- Initialise θ
- Repeat until convergence:
  - $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$
  - Simultaneous update for $j = 0, \ldots, d$

$0 < \alpha \leq 1$ is the learning rate, usually set quite small



Take a step of size α in the "downhill" direction (negative gradient)

GD with regularisation

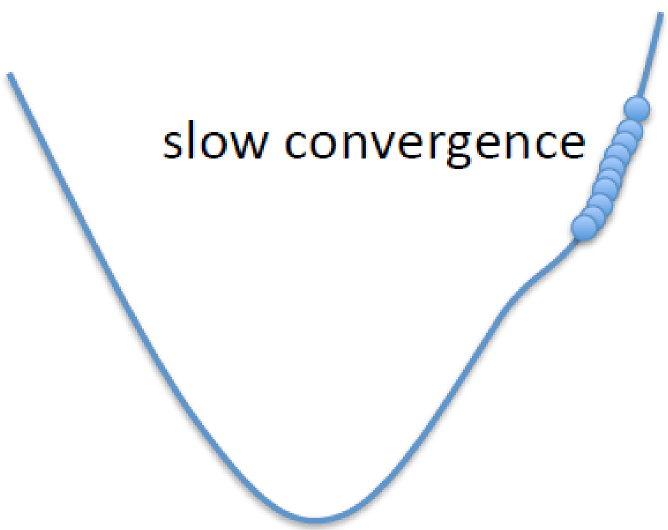
- Initialise θ
- Repeat until convergence:
  - $\theta_0 \leftarrow \theta_0 - \alpha(h_\theta(x^{(i)}) - y^{(i)})$
  - $\theta_j \leftarrow \theta_j - \alpha\left[(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \lambda\theta_j\right]$
  - Simultaneous update for $j = 0, \ldots, d$

No regularisation on θ_0

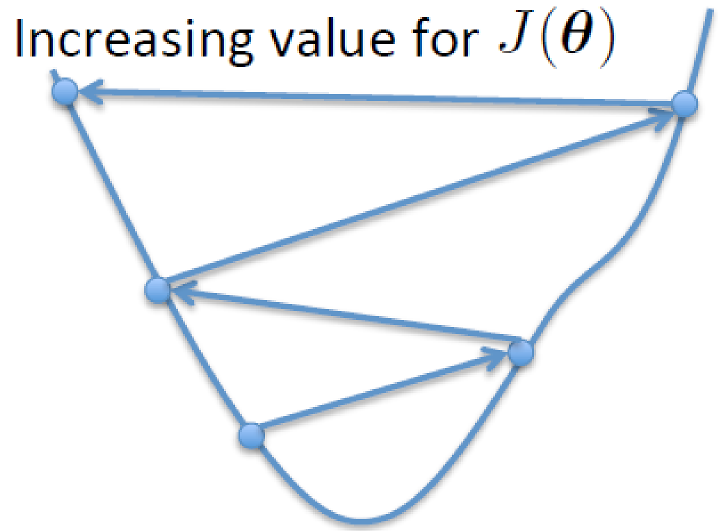
- This is identical to linear regression!
- But the model is completely different:
  - $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

The effect of α

α too small



α too large

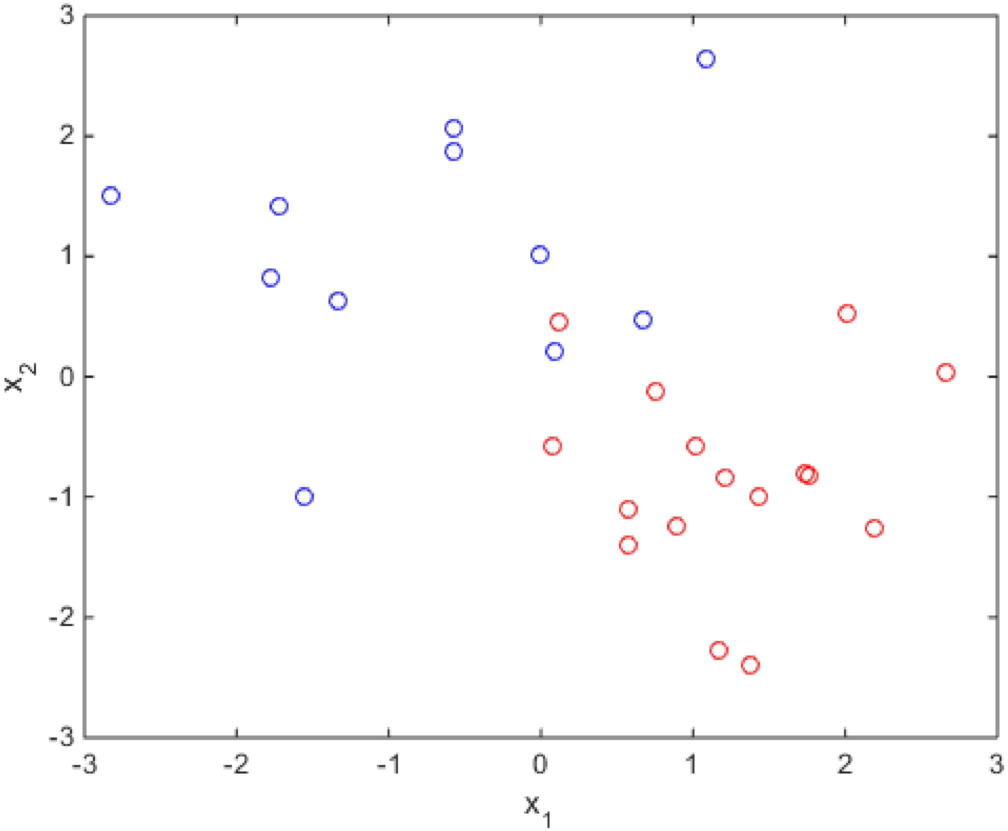


- May overshoot the minimum
- May fail to converge
- May even diverge

Example

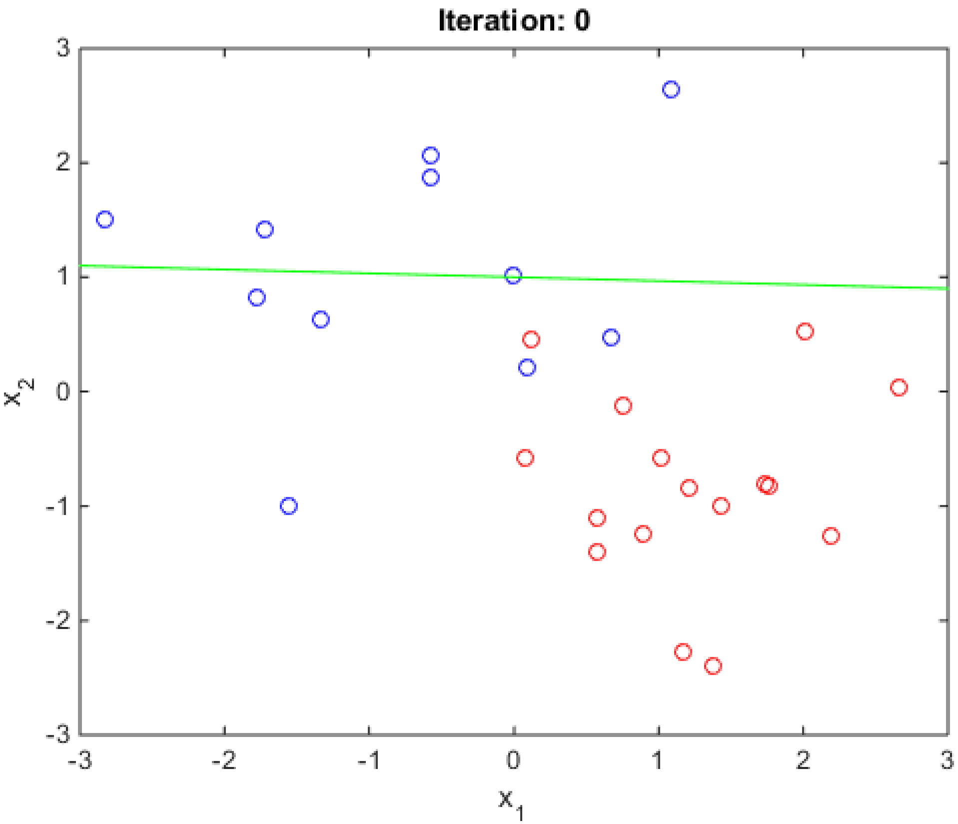
- Generate two random classes of data, from Gaussians centered at (1, -1) and (-1, 1)

$$h_\theta(x) = \sigma\big(\theta^T \phi(x)\big)$$
$$= \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$



Example

- Weights randomly initialized: $\theta = (0.3, -0.01, -0.3)$



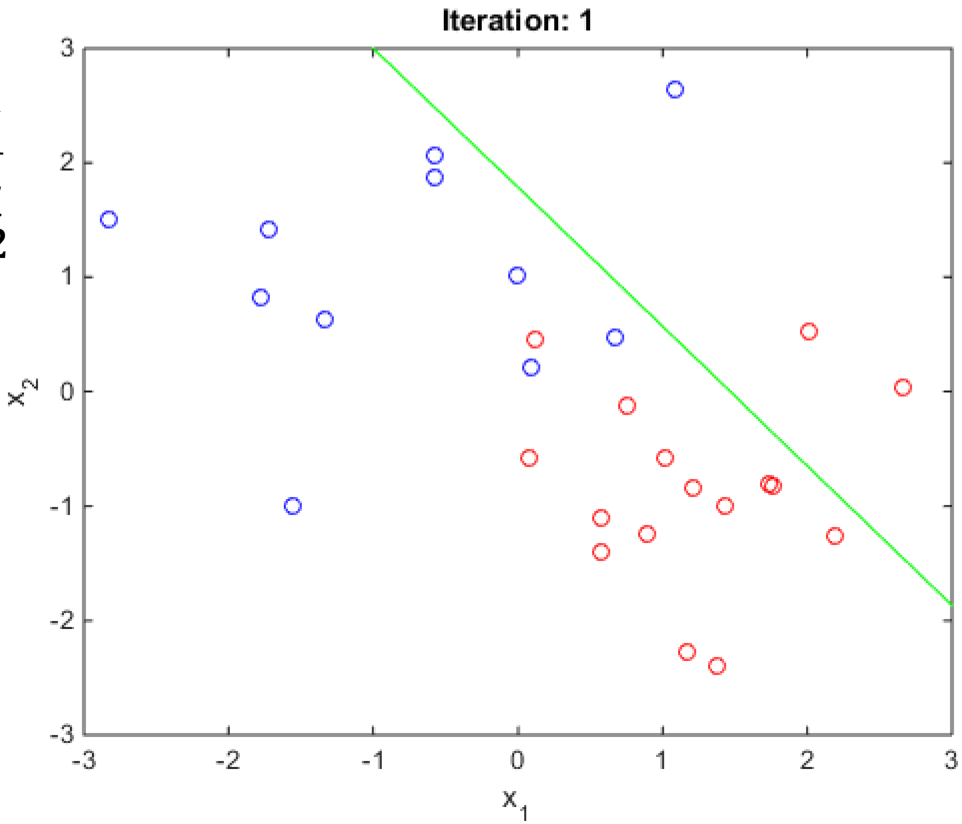
Example

- Cycle through each data point i:
- Compute:

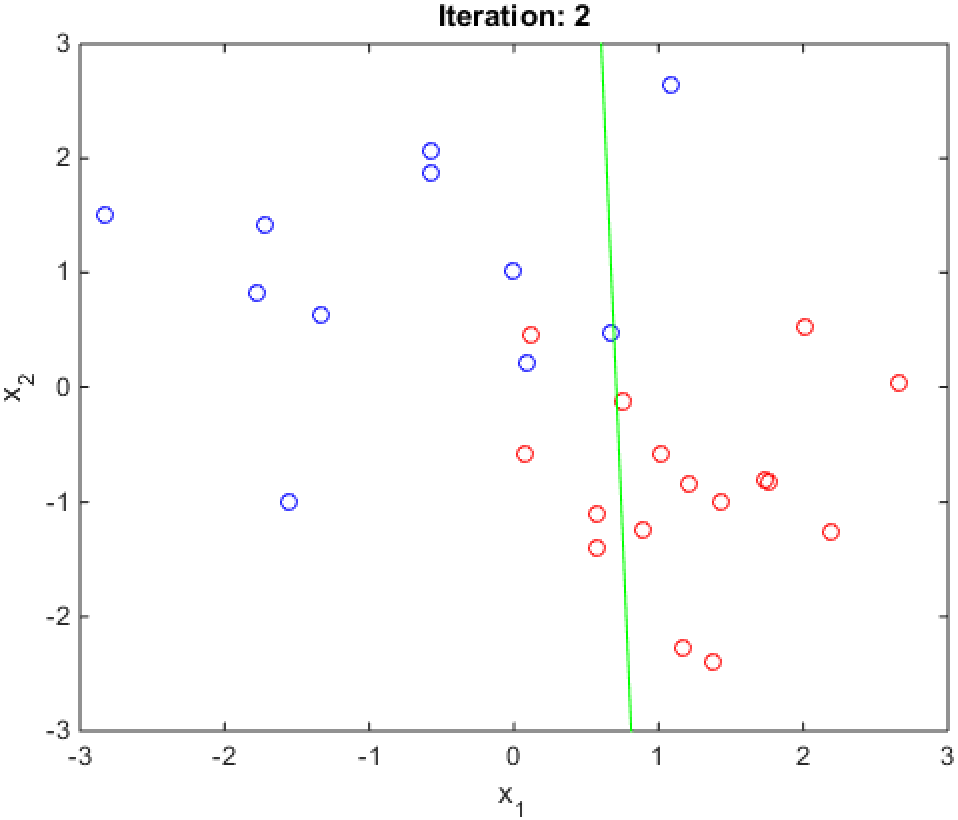
$$\delta\theta_0 = \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right)$$
$$\delta\theta_1 = \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_1^i$$
$$\delta\theta_2 = \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_2^i$$

Update:

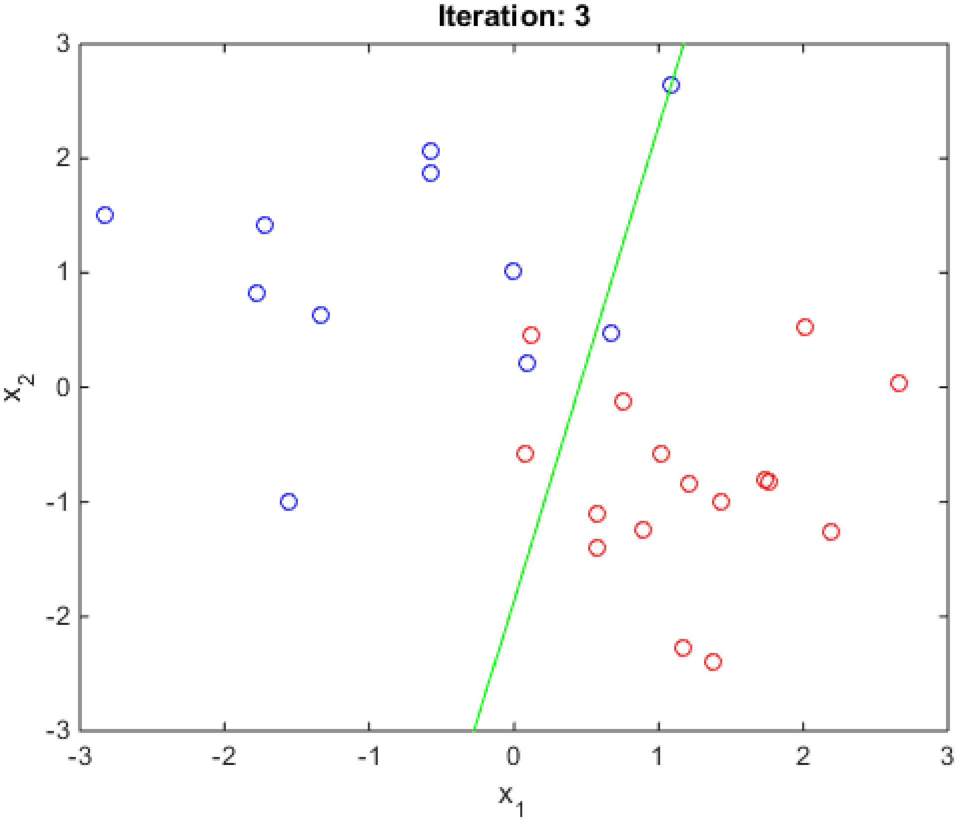
$$\theta_0 \leftarrow \theta_0 + \alpha\delta\theta_0$$
$$\theta_1 \leftarrow \theta_1 + \alpha\delta\theta_1$$
$$\theta_2 \leftarrow \theta_2 + \alpha\delta\theta_2$$



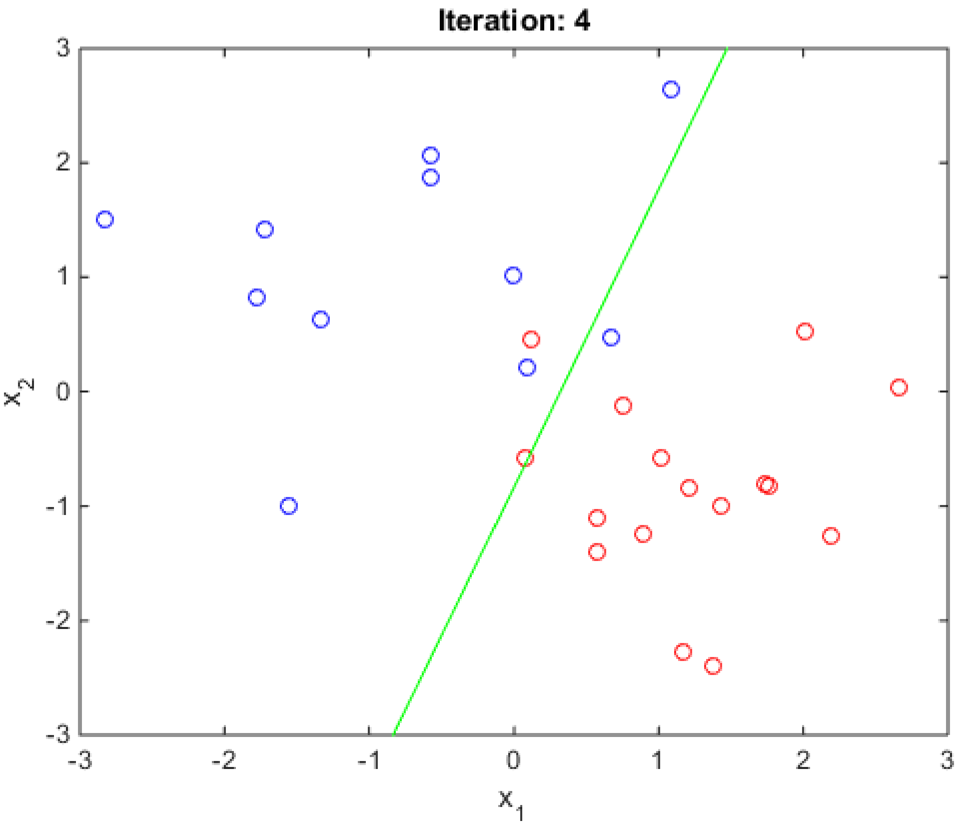
Example



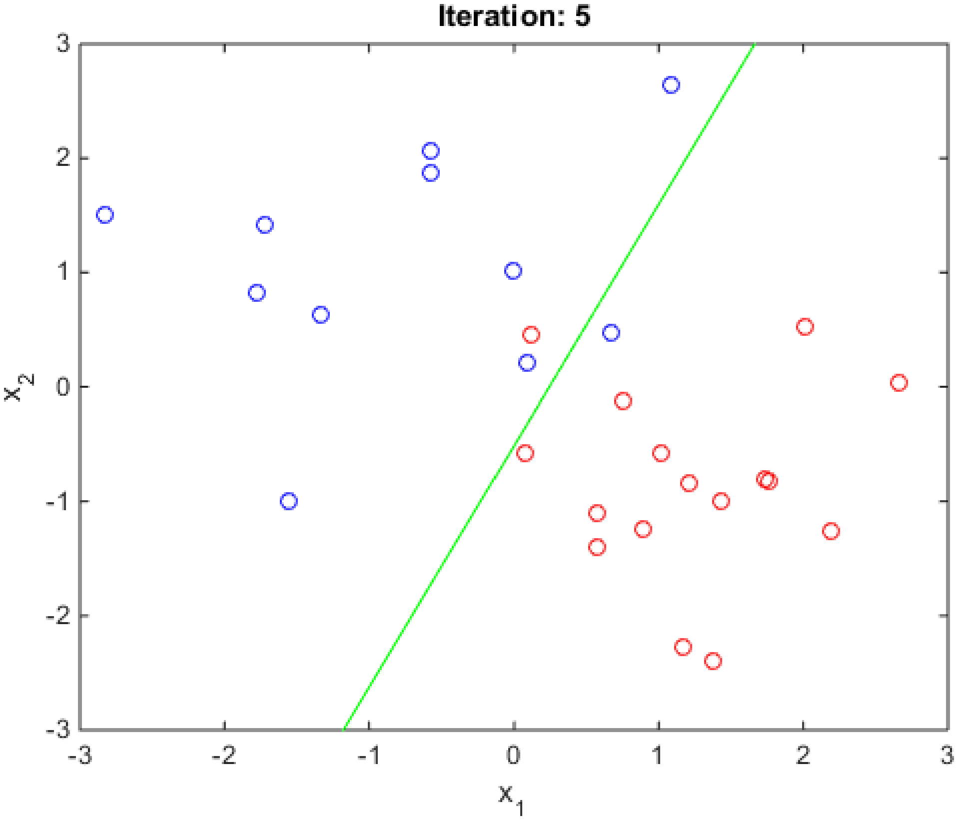
Example



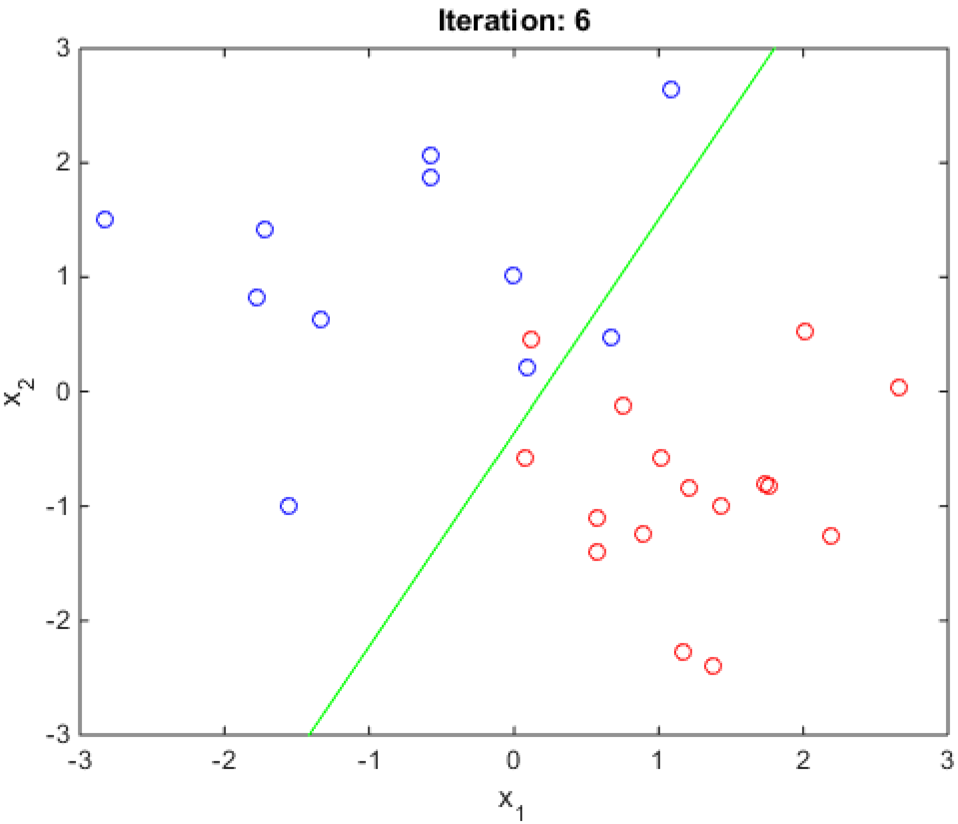
Example



Example



Example



Example

- Run until convergence (threshold on the size of change of θ)

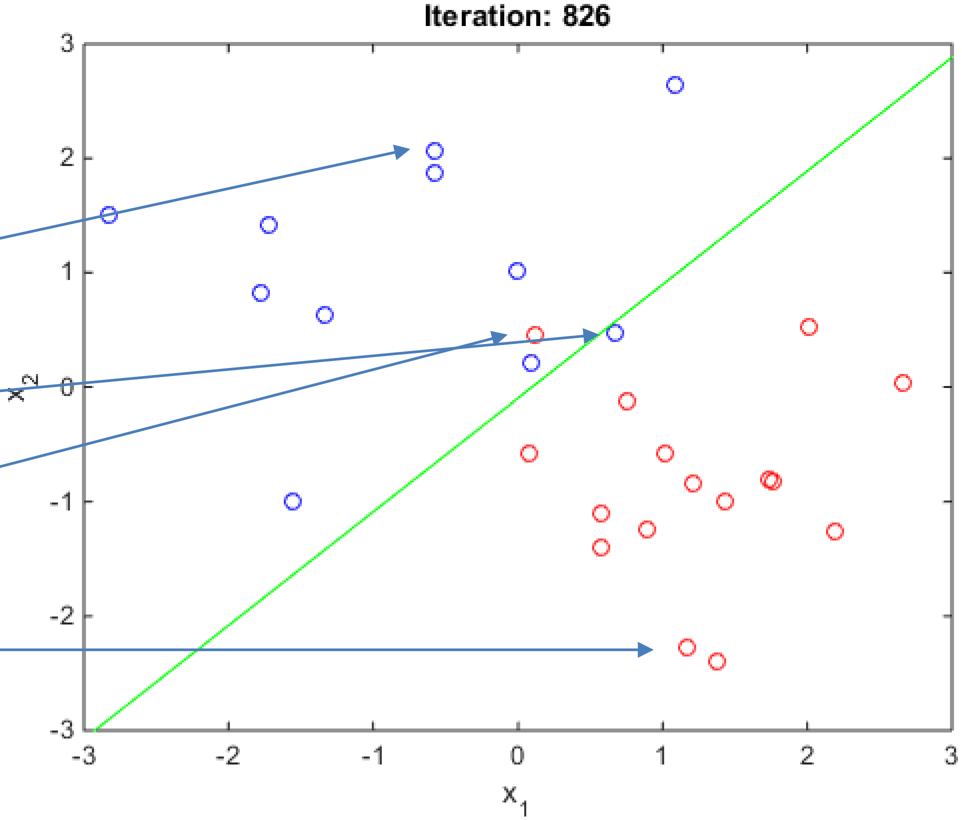
Probabilities (of class 1):

0.9999

0.4386

0.7759

0.0007

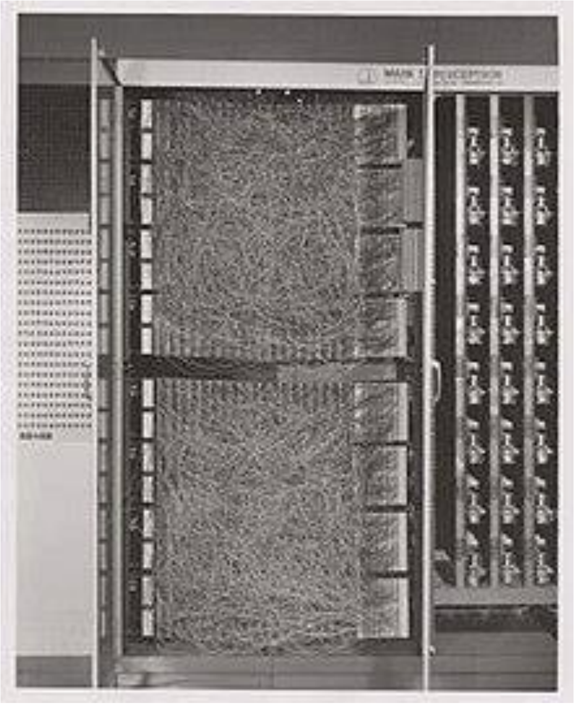


Digression: the perceptron

- The logistic function gives a probabilistic output
  - What if we wanted to instead force it to be {0, 1}?
- Instead of the logistic function, what about a step function?
  - $g(z) = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; if \; z < 0 \end{cases}$
- Use this as before:
  - $p(y = 1|x) = g\left(\theta^T \phi(x)\right) = h_\theta(x)$
- Perceptron learning rule:
  - $\theta_j \leftarrow \theta_j + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$
  - Exactly as before (with a different function)!

The perceptron

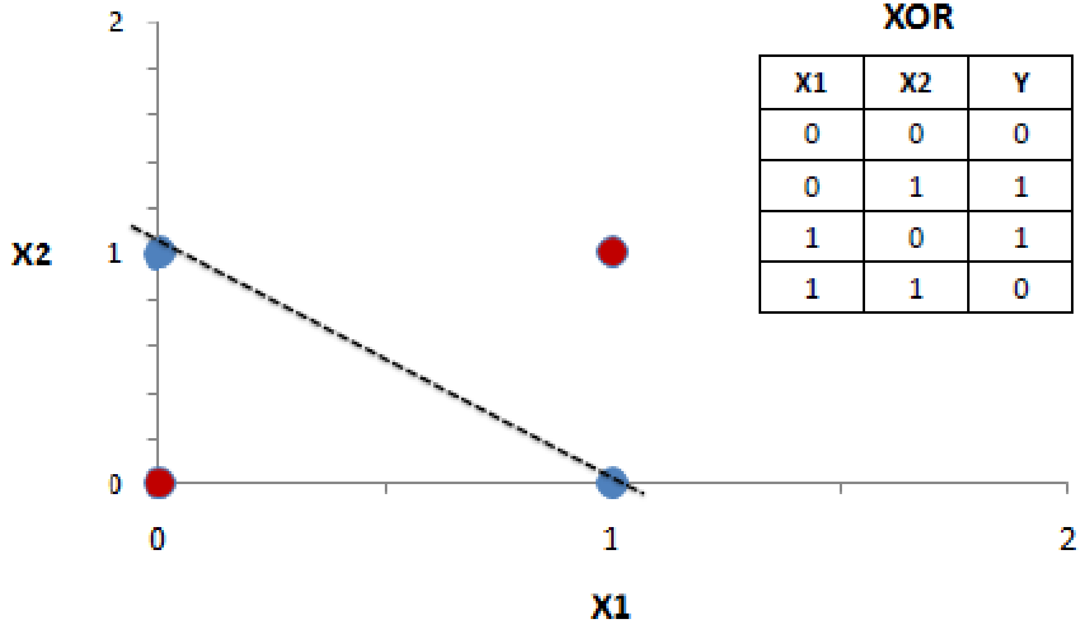
- Historical model
- Invented by Frank Rosenblatt (1957)
- Thought to model neurons in the brain
    - (Crudely)
- Originally a machine!



- Very controversial:
    - Basically claimed they expected to
    - “be able to walk, talk, see, write, reproduce itself and be conscious of its existence”

Linear separability and XOR

- “Perceptrons” by Minsky and Papert (1969)
- Limitation of a perceptron: cannot implement functions such as a XOR function
- Led to decreased research in neural networks, and increased research in symbolic AI



XOR

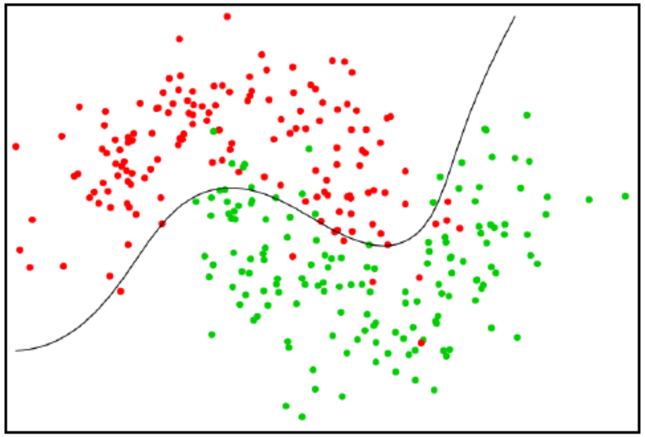
X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Basis functions (again)

- Use basis functions (again) to get round the linear separability
- Still need it to be separable in **some** space

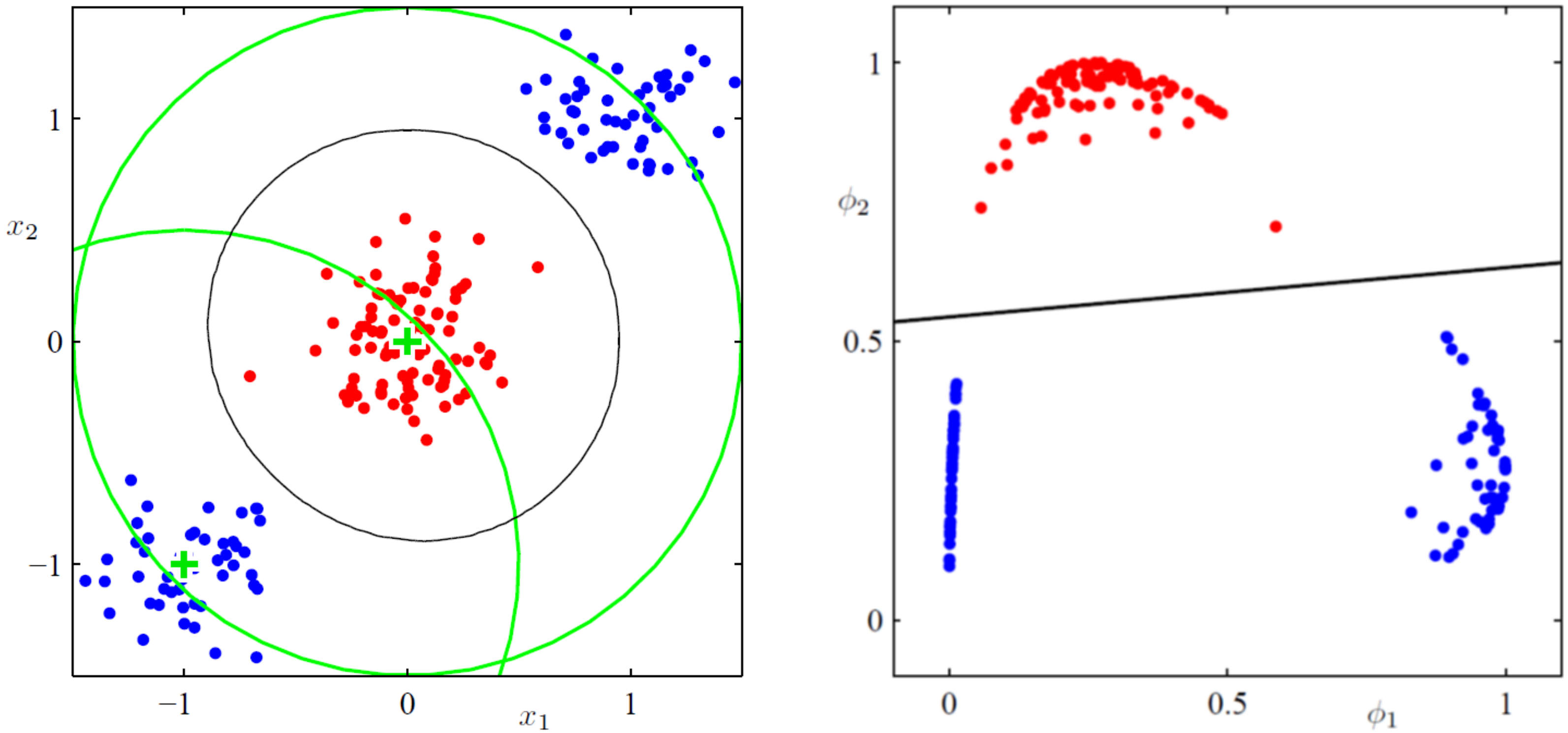
$$\boldsymbol{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \\ x_1^2 x_2 \\ x_1 x_2^2 \\ \vdots \end{bmatrix}$$

Add polynomial basis functions

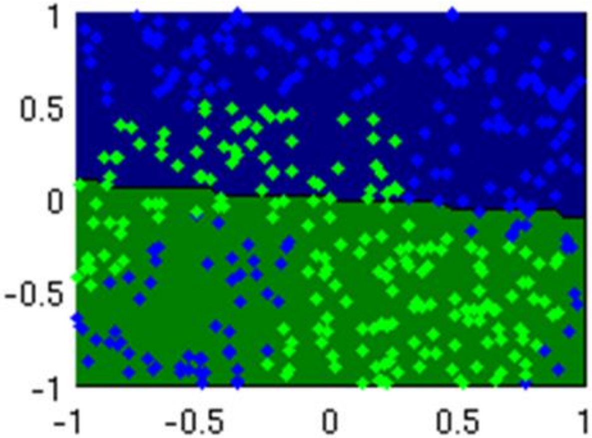


Basis functions (again)

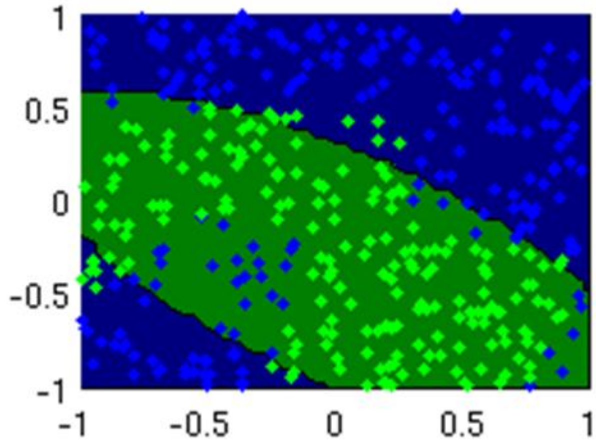
- Two Gaussian basis functions: centered at (-1, -1) and (0, 0)
  - Data is separable under this transformation



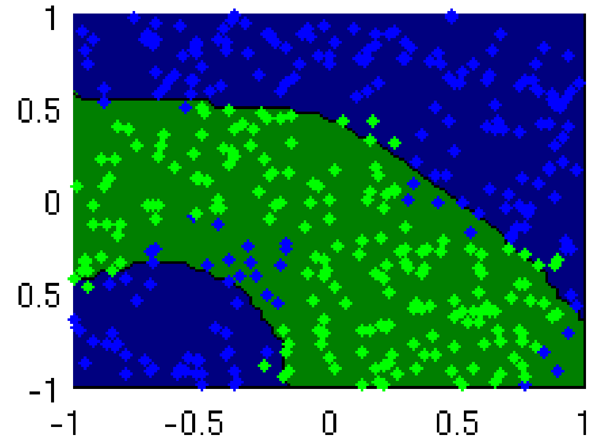
Basis functions (again)



Linear logistic regression



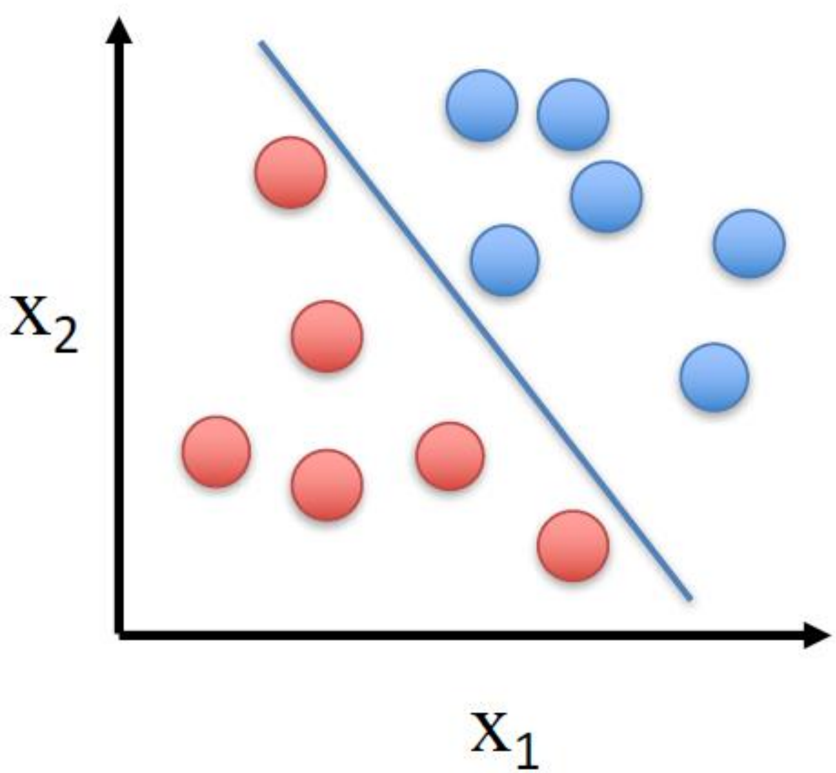
Polynomial basis functions

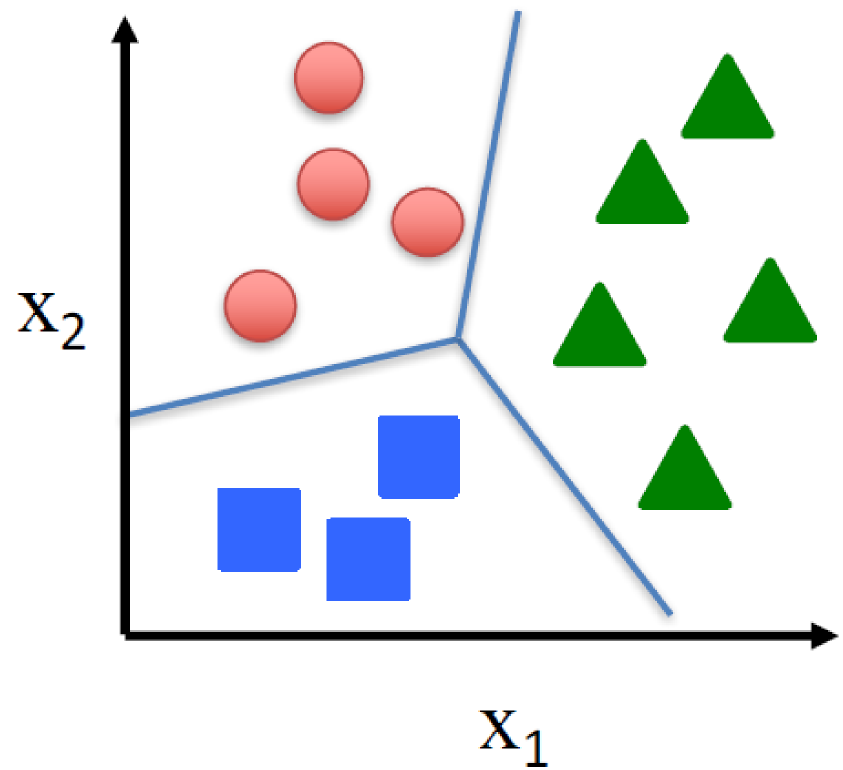


Gaussian basis functions (RBFs)

Multiclass classification

- Instead of classifying between two classes, we may have **more** classes





Multiclass logistic regression

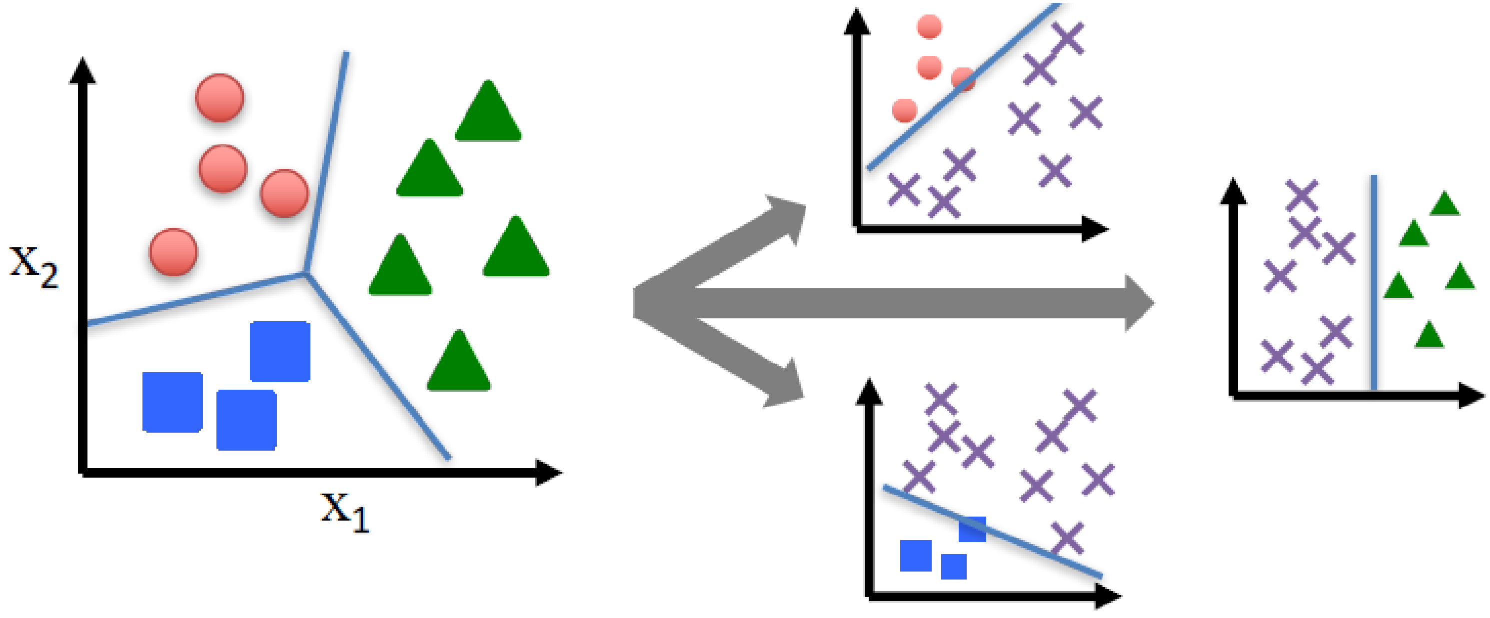
- For two classes:
  - $p(y=1|x;\theta) = h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)}$

$$= \frac{\exp(\theta^T x)}{1+\exp(\theta^T x)}$$

- Given C classes:
  - $p(y=c_k|x;\theta) = \frac{\exp(\theta_k^T x)}{\sum_{j=1}^{C}\exp(\theta_j^T x)}$
  - This is the **softmax** function
  - Note that $0 \le p(c_k|x;\theta) \le 1$, and $\sum_{j=1}^{C} p(c_k|x;\theta) = 1$

Multiclass classification

- Split into one-vs-rest for each of the C classes



- Use gradient descent: update all parameters for all models simultaneously
- Pick most probable class

Recap

- Discriminative vs generative
- Model (logistic function)
- Decision boundaries
- Cost function
- Regularisation
- Gradient descent
- The perceptron
- Basis functions
- Multiclass classification