Computer Vision – TP8
Statistical Classifiers

Miguel Coimbra, Hélder Oliveira

U. PORTO FC

Outline

- Statistical Classifiers
- Support Vector Machines
- Neural Networks

Topic: Statistical Classifiers

- Statistical Classifiers
- Support Vector Machines
- Neural Networks

Statistical PR

- I use **statistics** to make a decision
  - I can make **decisions** even when I don’t have full a priori knowledge of the whole process
  - I can make **mistakes**
- How did I **recognize** this pattern?
  - I **learn** from previous observations where I know the classification result
  - I **classify** a new observation

Features

- Feature ***F_i*** $F_i = [f_i]$

- Feature ***F_i*** with ***N*** values.

  $$F_i = [f_{i1}, f_{i2}, \ldots, f_{iN}]$$

- Feature vector F with M features.

  $$F = [F_1 | F_2 | \ldots | F_M]$$

- Naming conventions:
  - Elements of a **feature vector** are called **coefficients**
  - **Features** may have one or more **coefficients**
  - **Feature vectors** may have one or more **features**

Classifiers

- A **Classifier C** maps a class into the feature space

$$C_{\text{Spain}}(x,y) = \begin{cases} true & ,y > K \\ false & ,otherwise \end{cases}$$

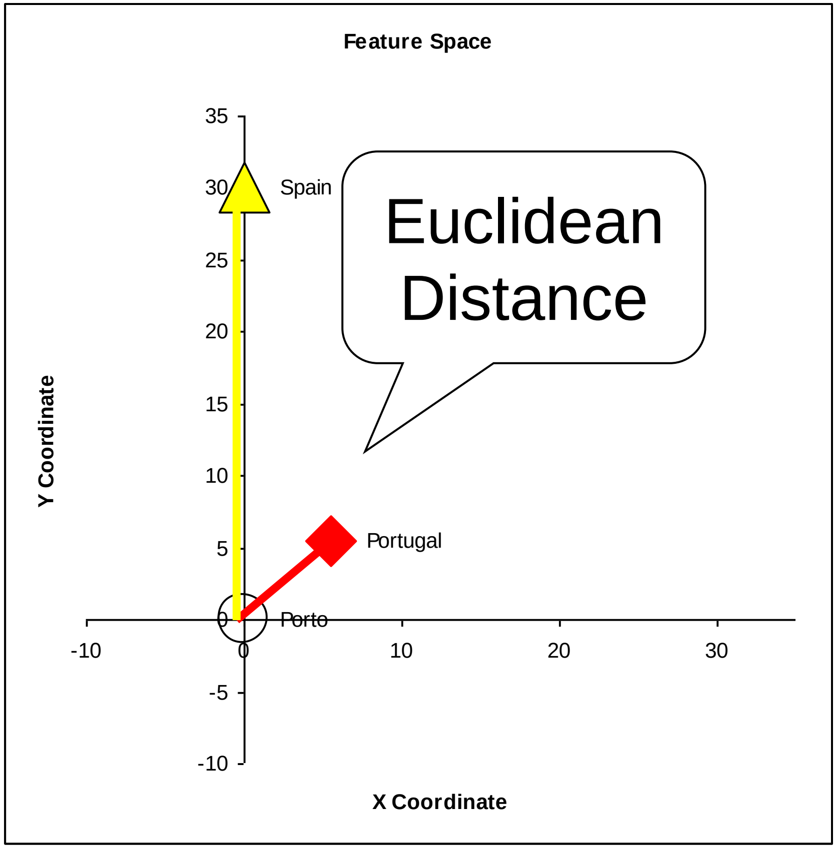
- Various types of classifiers
  - Nearest-Neighbours
  - Support Vector Machines
  - Neural Networks
  - Etc...

Distance to Mean

- I can represent a class by its mean feature vector

$$C = \overline{F}$$

- To classify a new object, I choose the class with the closest mean feature vector
- Different distance measures!



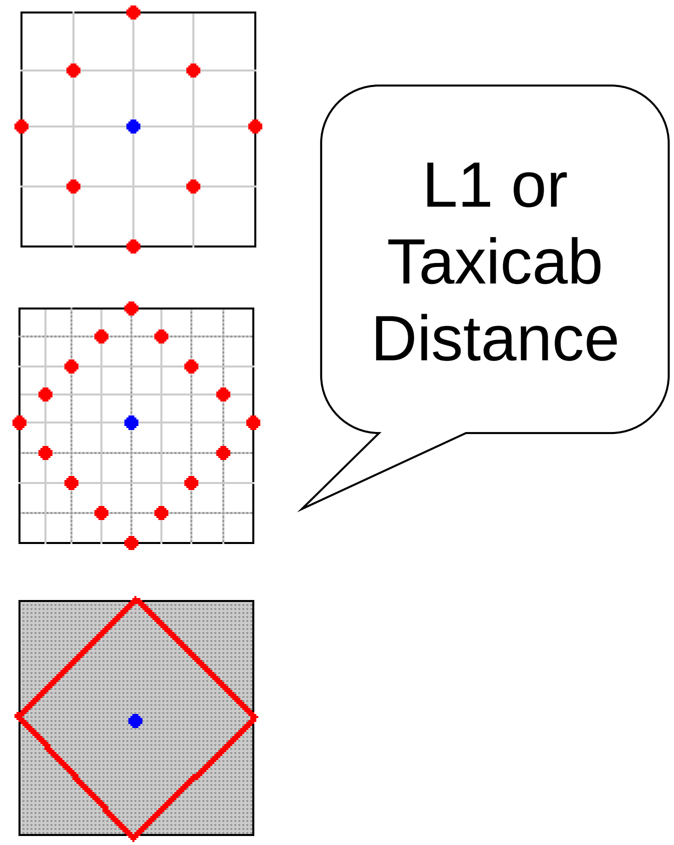
Possible Distance Measures

- L1 Distance

$$L1(x,y) = \sum_{i=1}^{N} |x_i - y_i|$$

- Euclidean Distance (L2 Distance)

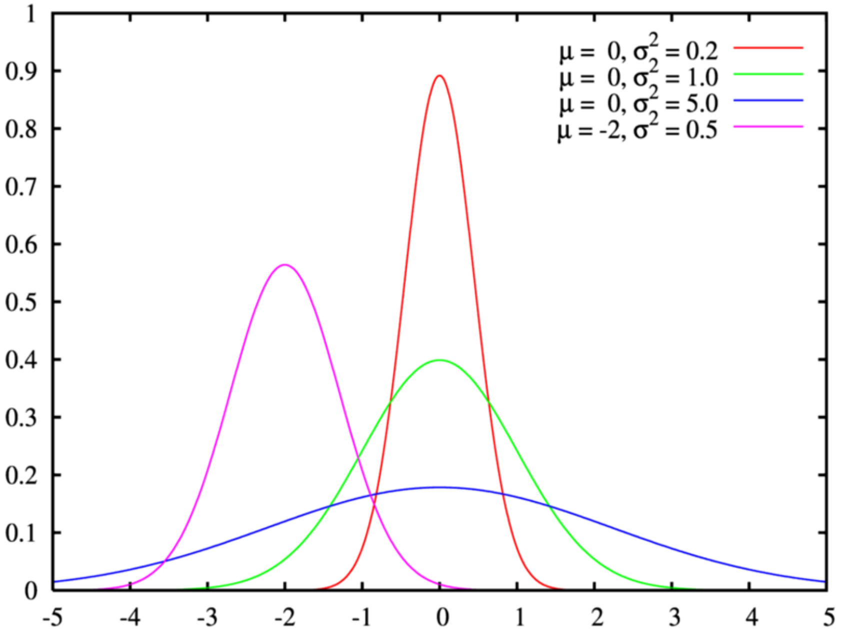
$$L2(x,y) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2}$$



Gaussian Distribution

- Defined by two parameters:
  – Mean: μ
  – Variance: σ^2
- Great approximation to the distribution of many phenomena.
  – *Central Limit Theorem*

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-u)^2}{2\sigma^2}\right)$$



Multivariate Distribution

- For N dimensions:

$$f_X(x_1,\ldots,x_N) = \frac{1}{(2\pi)^{N/2}\left|\Sigma\right|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right)$$

- Mean feature vector:

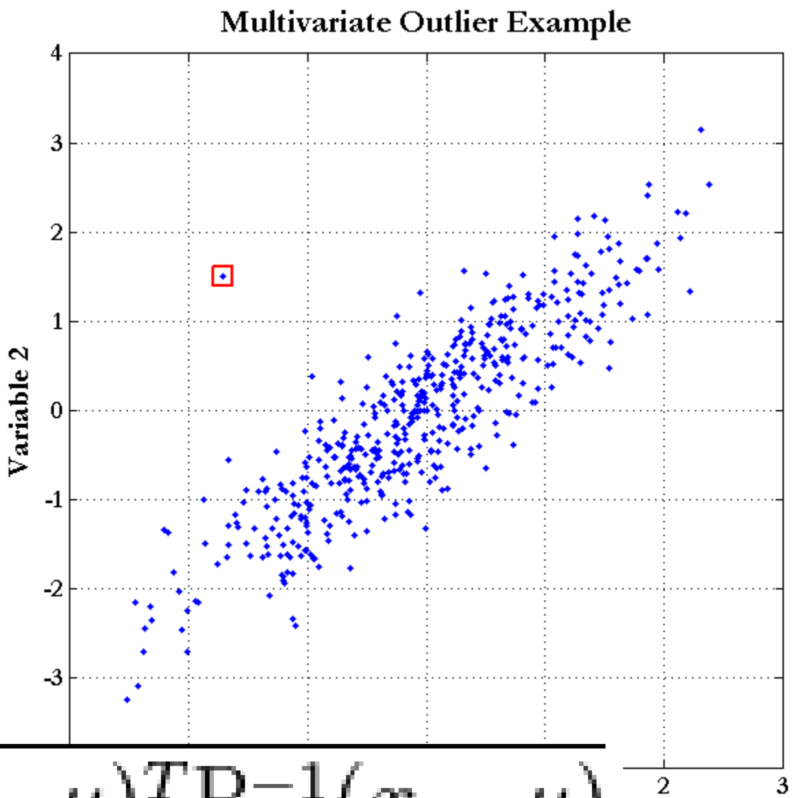
$$\mu = \overline{F}$$

- Covariance Matrix:

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_n \end{bmatrix} \qquad \mu_i = \mathrm{E}(X_i) \qquad \Sigma_{ij} = \mathrm{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right]$$

Mahalanobis Distance

- Based on the covariance of coefficients
- Superior to the Euclidean distance



$$D_M(x) = \sqrt{(x-\mu)^T P^{-1}(x-\mu)}.$$

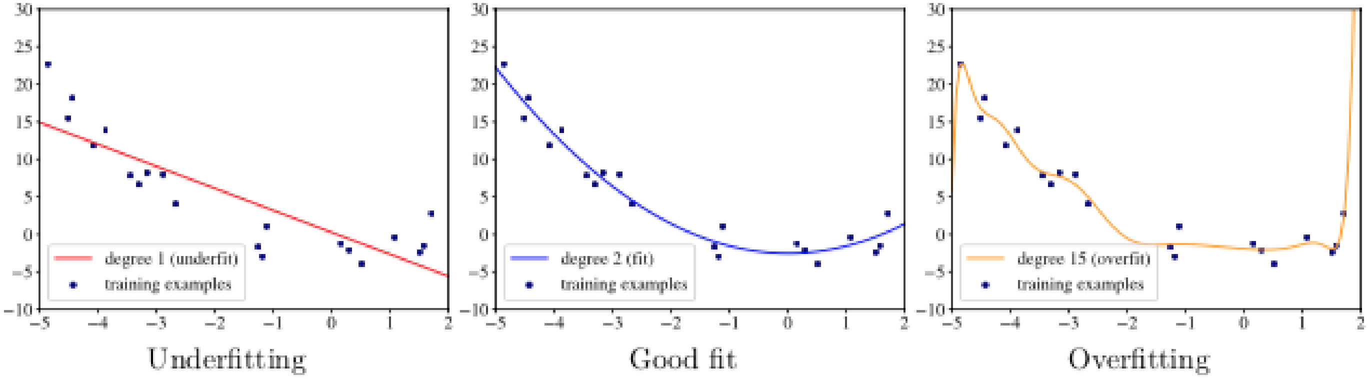
Generalization

- Classifiers are optimized to reduce training errors
  - (supervised learning): we have access to a set of training data for which we know the correct class/answer

- What if test data is different from training data?

Underfitting and Overfitting

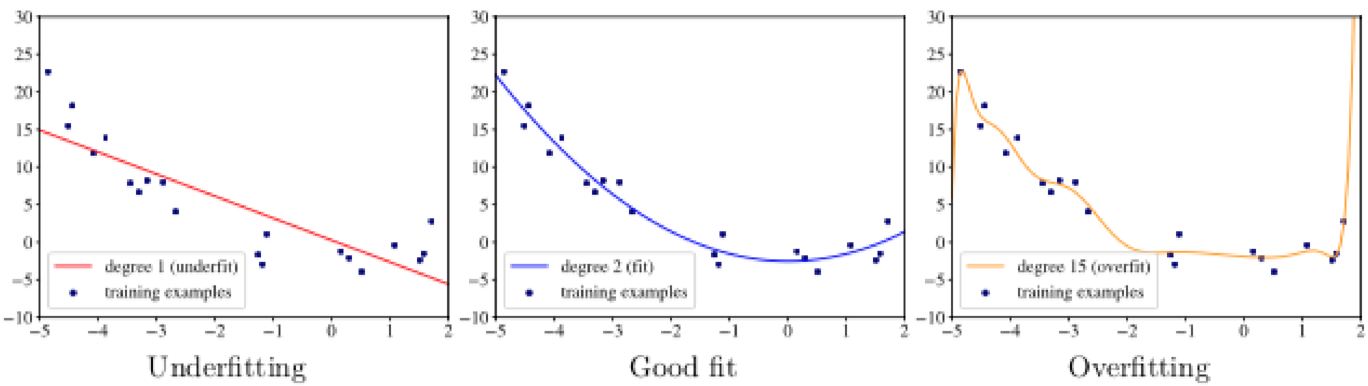
- Is the model too simple for the data?
  - Underfitting: cannot capture data behavior
- Is the model too complex for the data?
  - Overfitting: fit perfectly training data, but will not generalize well on unseen data



Bias and variance

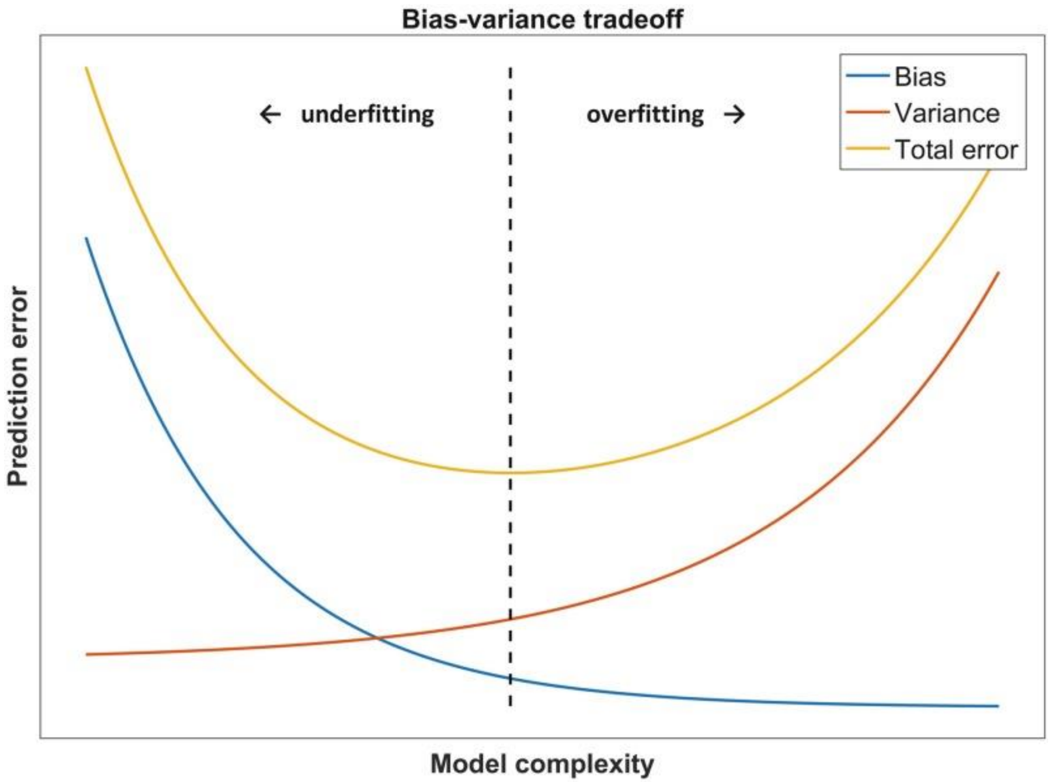
- Bias
  – Average error in predicting correct value
- Variance
  – Variability of model prediction

High bias | Low bias, low variance | High variance



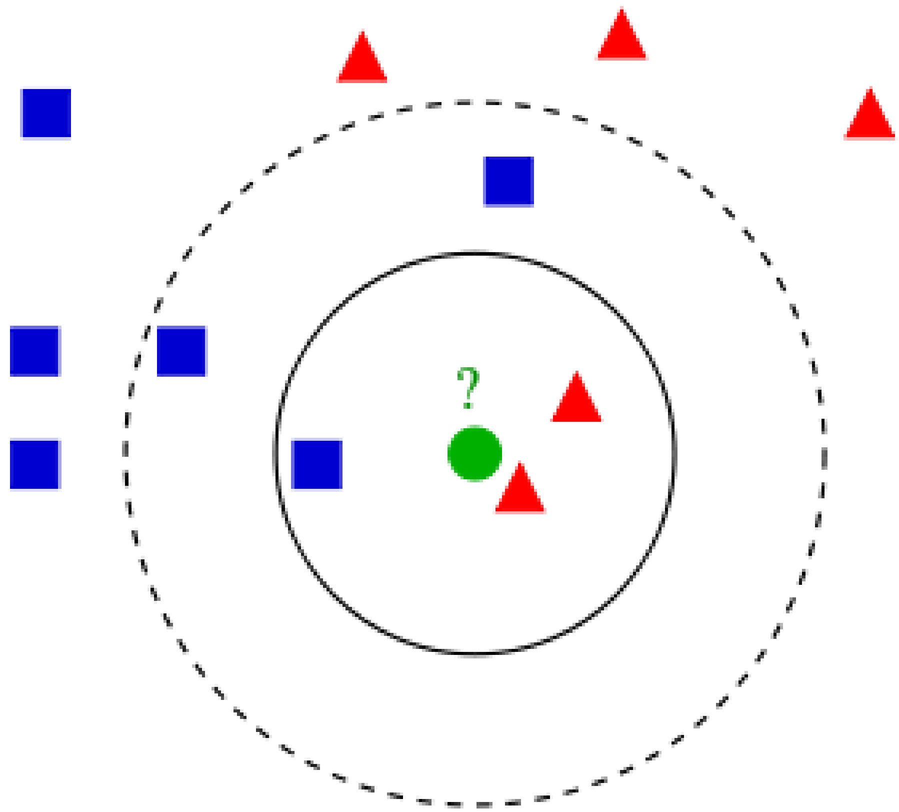
Bias-variance tradeoff

- total err = bias^2 + variance + irreducible err



K-Nearest Neighbours

- Algorithm
  - Choose the closest K neighbours to a new observation
  - Classify the new object based on the **class** of these K objects
- Characteristics
  - Assumes no model
  - Does not scale very well...



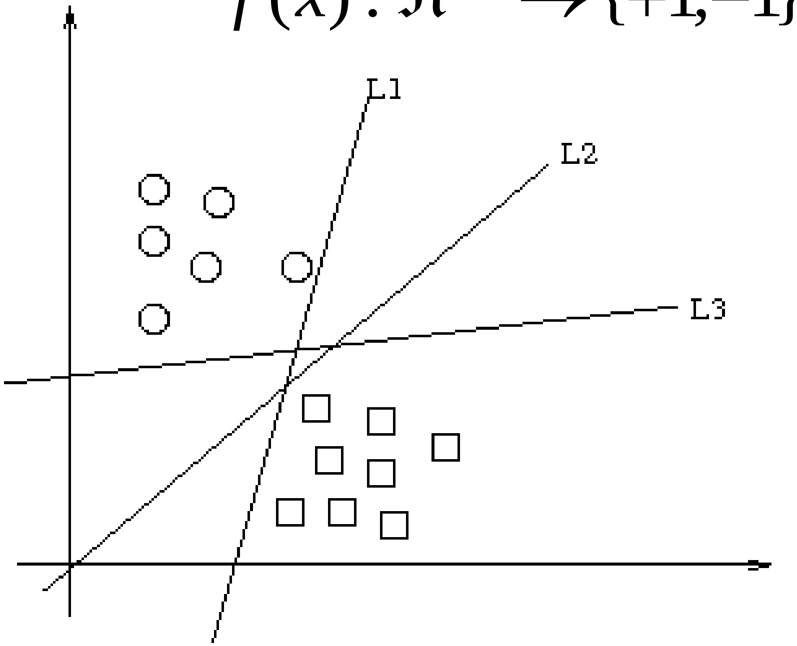
Topic: Support Vector Machines

- Statistical Classifiers
- Support Vector Machines
- Neural Networks

Maximum-margin hyperplane

- There are many planes that can separate our **classes** in **feature space**
- Only one **maximizes the separation margin**
- Of course that classes need to be separable in the first place...

$$f(x): \Re^M \rightarrow \{+1, -1\}$$

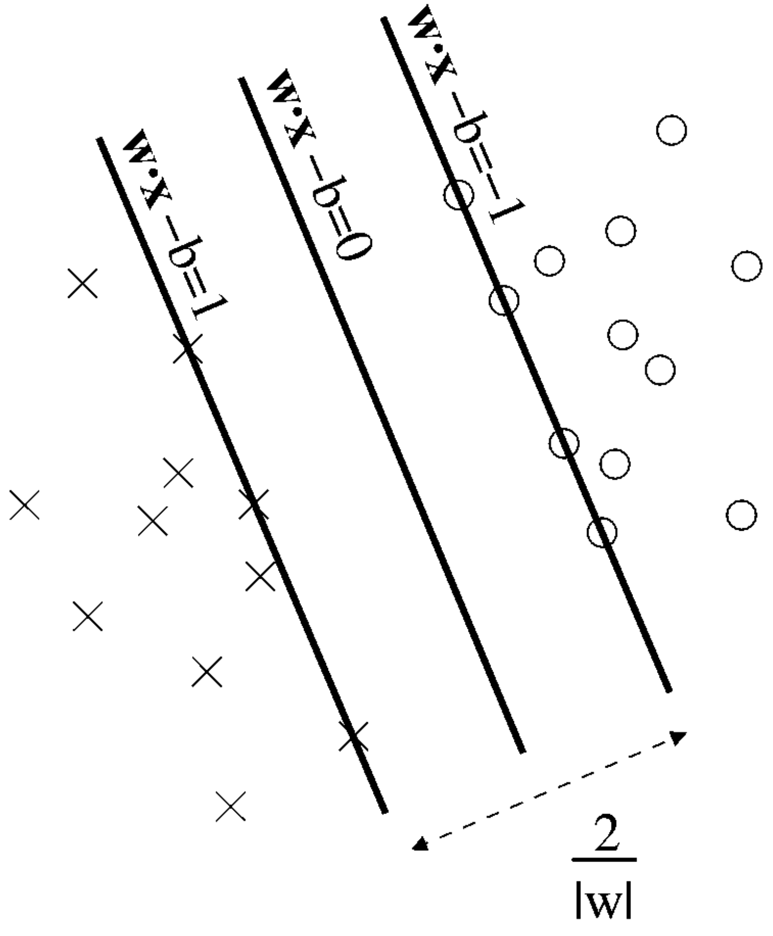


$$C \in \{+1, -1\}$$

$$V = \{v_1, v_2, \ldots, v_M\}, v_i \in \Re^M$$

Support vectors

- The **maximum-margin hyperplane** is limited by some vectors
- These are called **support vectors**
- Other vectors are irrelevant for my decision



Decision

- I map a new **observation** into my **feature space**
- Decision hyperplane:

$$(w.x) + b = 0, w \in \Re^N, b \in \Re$$

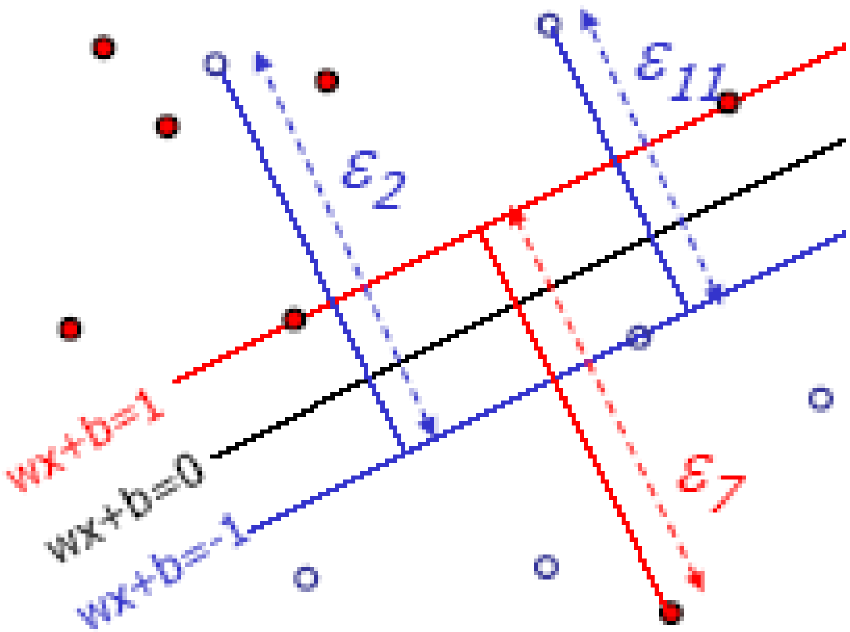
- Decision function:

$$f(x) = sign((w.x) + b)$$

A vector is either **above** or **below** the hyperplane

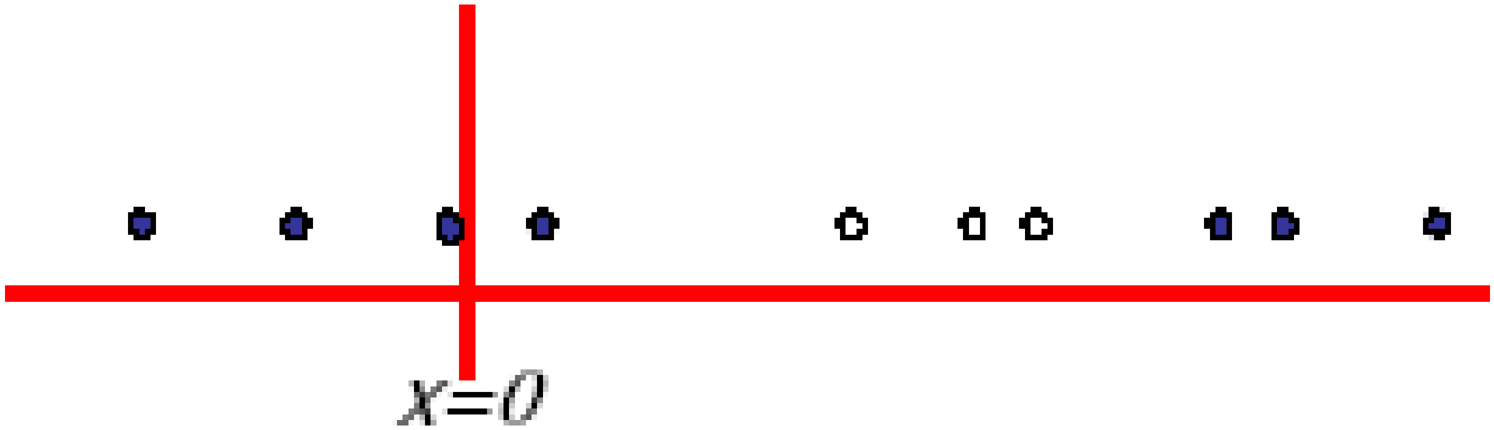
Slack variables

- Most **feature spaces** cannot be segmented so easily by a hyperplane
- Solution:
  - Use slack variables
  - ‘Wrong’ points ‘pull’ the margin in their direction
  - Classification errors!



But this doesn't work in most situations...

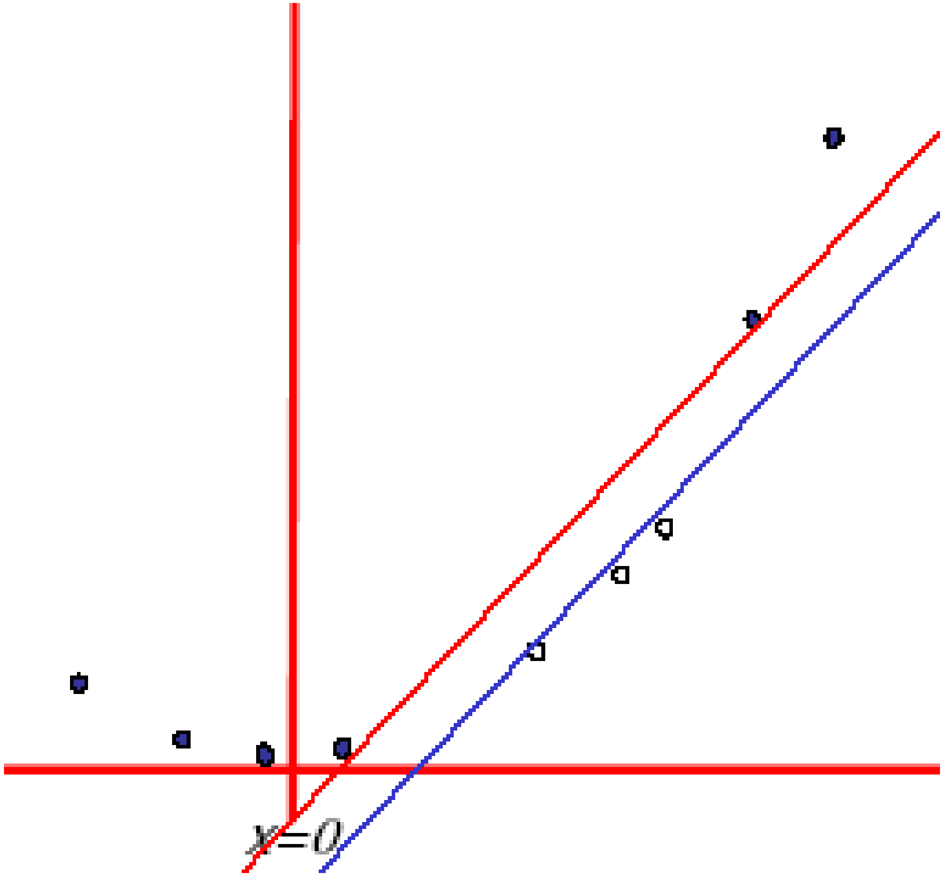
- Still, how do I find a **Maximum-margin hyperplane** for some situations?



- Most real situations face this problem...

Solution: Send it to hyperspace!

- Take the previous case: **f(x) = x**
- Create a new higher-dimensional function: **g(x) = (x, x²)**
- A **kernel function** is responsible for this transformation



https://www.youtube.com/watch?v=3liCbRZPrZA

Typical kernel functions

Linear	$K(x,y) = x.y + 1$
Polynomial	$K(x,y) = (x.y + 1)^p$
Radial-Base Functions	$K(x,y) = e^{-\|x-y\|^2 / 2\sigma^2}$
Sigmoid	$K(x,y) = \tanh(kx.y - \delta)$

Classification

- Training stage:
  - Obtain kernel parameters
  - Obtain maximum-margin hyperplane
- Given a new **observation**:
  - Transform it using the kernel
  - Compare it to the hyperspace

Topic: Neural Networks

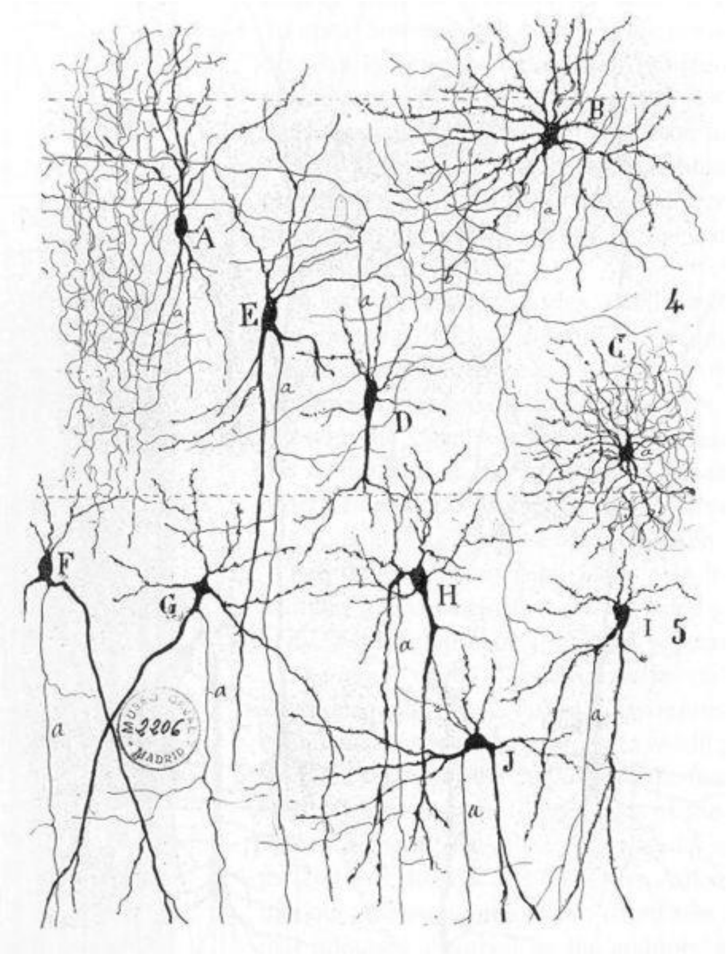
- Statistical Classifiers
- Support Vector Machines
- Neural Networks

If you can't beat it.... Copy it!

http://managementcraft.typepad.com/photos/uncategorized/brain.jpg

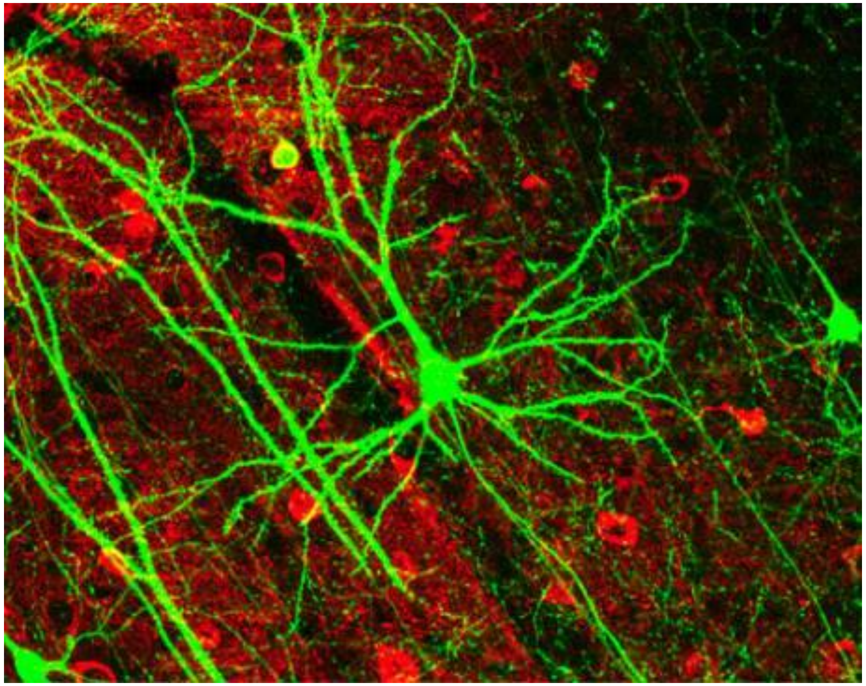
Biological Neural Networks

- Neuroscience:
  - Population of physically inter-connected neurons
- Includes:
  - Biological **Neurons**
  - Connecting **Synapses**
- The human brain:
  - 100 billion neurons
  - 100 trillion synapses



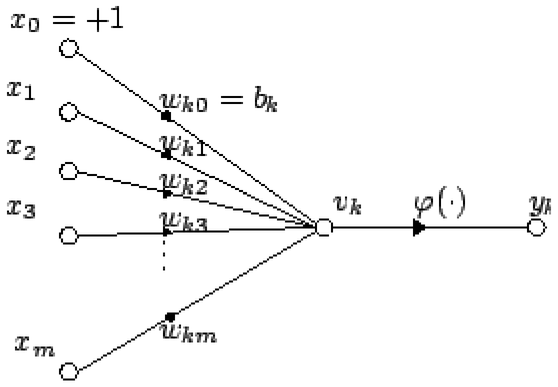
Biological Neuron

- Neurons:
  - Have K inputs (*dendrites*)
  - Have 1 output (*axon*)
  - If the sum of the input signals surpasses a *threshold*, sends an *action potential* to the axon
- Synapses
  - Transmit electrical signals between neurons



Artificial Neuron

- Also called the **McCulloch-Pitts neuron**
- Passes a **weighted sum of inputs**, to an **activation function**, which produces an **output** value



$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 7:115 - 133.

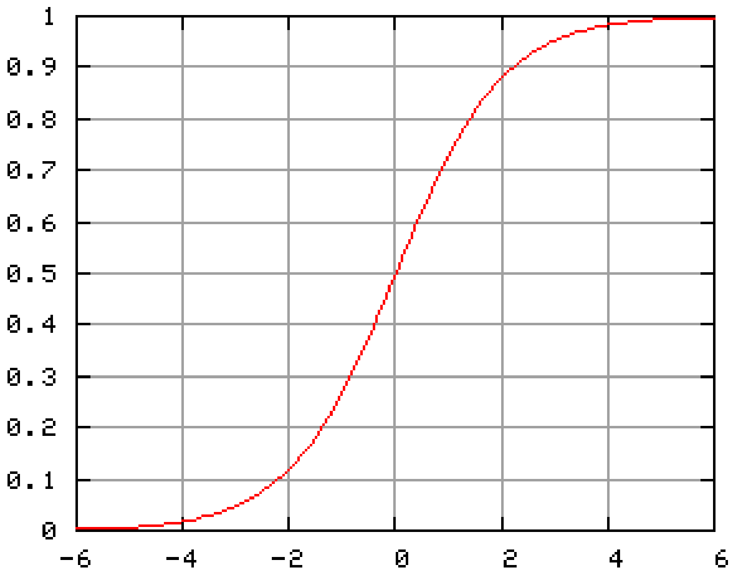
Sample activation functions

- Rectified Linear Unit (ReLU)

$$y = \begin{cases} u, & \text{if } u \geq 0 \\ 0, & \text{if } u < 0 \end{cases} \quad , \quad u = \sum_{i=1}^{n} w_i x_i$$

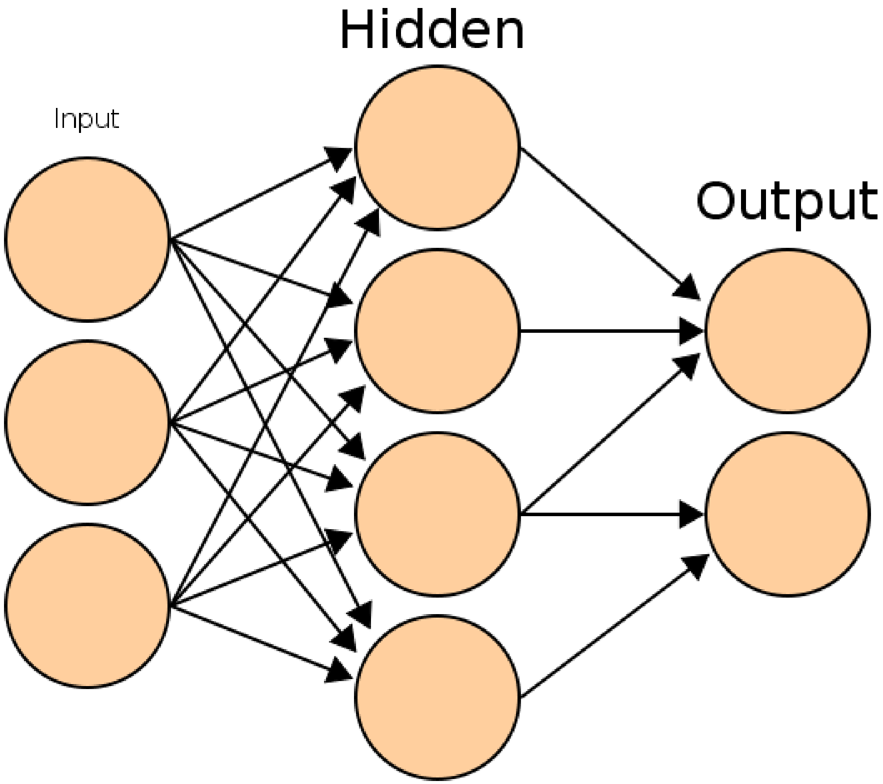
- Sigmoid function

$$y = \frac{1}{1+e^{-u}}$$



Artificial Neural Network

- Commonly refered as **Neural Network**
- Basic principles:
  - One neuron can perform a simple decision
  - Many **connected** neurons can make more **complex decisions**



Characteristics of a NN

- Network configuration
  - How are the neurons inter-connected?
  - We typically use *layers* of neurons (input, output, hidden)
- Individual Neuron parameters
  - Weights associated with inputs
  - Activation function
  - Decision *thresholds*

How do we find these values?

Learning paradigms

- We can define the network configuration
- How do we define neuron ***weights*** and ***decision thresholds***?
  - **Learning** step
  - We **train** the NN to classify what we want
- Different learning paradigms
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning

Appropriate for **Pattern Recognition**.

Learning

- We want to obtain an **optimal solution** given a set of **observations**
- A **cost function** measures how close our solution is to the **optimal solution**
- Objective of our learning step:
  - Minimize the **cost function**

Backpropagation Algorithm

In formulas

Network output: $\text{Out}(x) = \varphi(\sum_m w_{nm}^{(L)} \varphi(\ldots \varphi(\sum_j w_{\ell j}^{(2)} \varphi(\sum_k w_{jk}^{(1)} x_k)))$

Training set: $\{(x_i, y_i)\}_{i=1,\ldots,N}$ (input: x_i; label: y_i)

Optimization: find $[w_{jk}^{(1)}, w_{\ell j}^{(2)}, \ldots, w_{nm}^{(L)}]$ such that

$$\text{minimize} \sum_{i=1}^{N} \text{Loss}(\text{Out}(x_i), y_i)$$

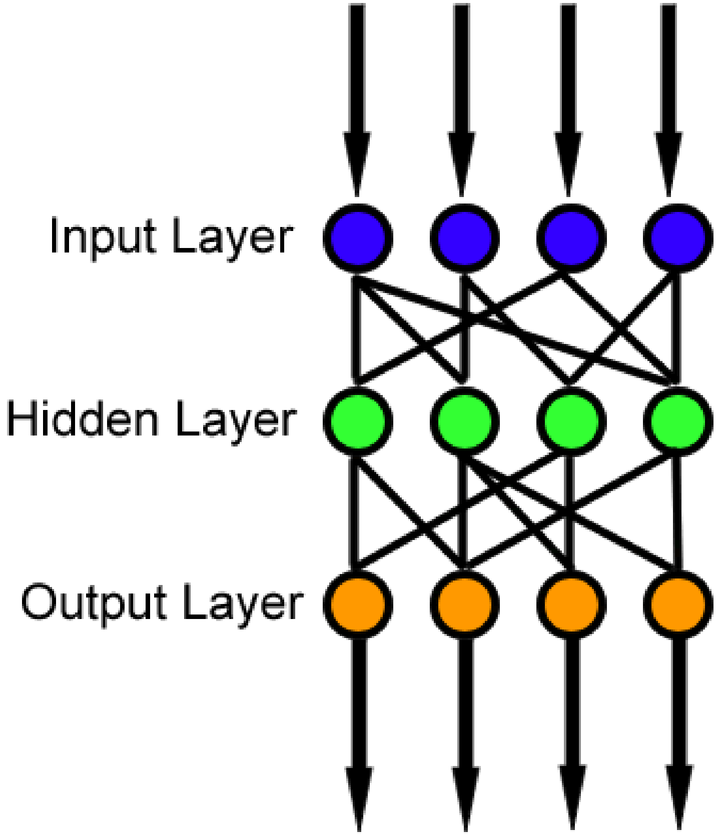
It is solved with (variants of) the gradient descent, where gradients are computed via the backpropagation algorithm

Losses

- They quantify the distance between the output of the network and the true label, i.e., the correct answer
- Classification problems:
  - The output (obtained usually with softmax) is a probability distribution
  - Loss-function: cross-entropy. It is defined in terms of the Kullback-Leibler distance between probability distributions
- Regression problems:
  - The output is a scalar or a vector of continuous values (real or complex)
  - Loss-function: mean-squared error. It is the distance associated with the L2-norm

Feedforward neural network

- Simplest type of NN
- Has no *cycles*
- Input layer
  - Need as many neurons as coefficients of my *feature vector*
- Hidden layers
- Output layer
  - Classification results



Resources

- Andrew Moore, “Statistical Data Mining Tutorials”, http://www.cs.cmu.edu/~awm/tutorials.html
- C.J. Burges, “A tutorial on support vector machines for pattern recognition”, in Knowledge Discovery Data Mining, vol.2, no.2, 1998, pp.1-43.