Lecture 5:

Data classification (II)

Outline

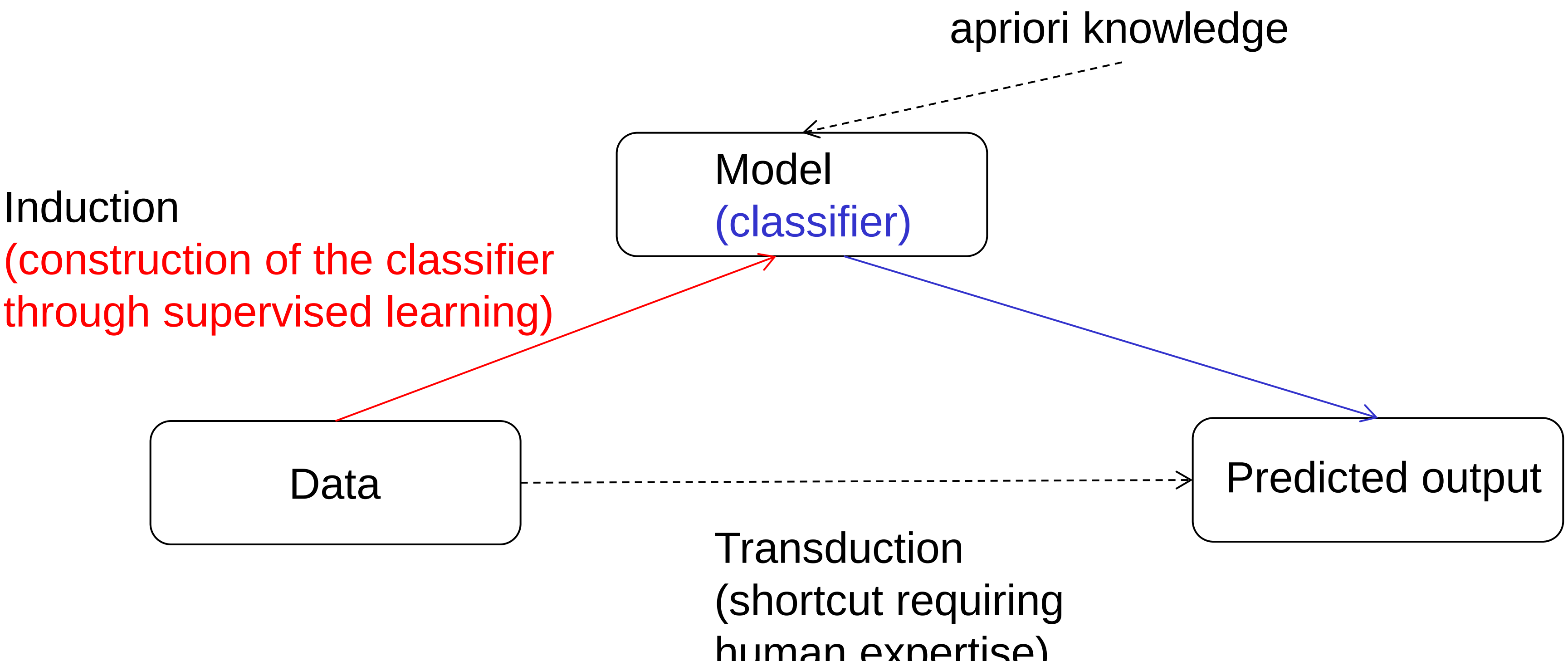
- Instance based classification
- Naïve Bayes classification
- Multilayer perceptrons

Reminder: classification models

Learning/ induction/ inference = construct a model starting from data (and some apriori knowledge specific to the domain)

Different ways of using data, models and knowledge:

induction vs deduction vs transduction



Reminder: classification models

A classification model is a "mapping" between attributes and class labels

Example of classification models:

- Decision trees
- Classification rules
- Prototypes (exemplars)
- Probabilistic models
- Neural networks etc.

The classification model should be:

- Accurate:
  - Identify the right class
- Compact / comprehensible
  - Easy to be understood/ interpreted by the user (it is preferable to not be a black box)
- Efficient in the
  - Learning/training step
  - Classification step

Instance based classifiers

Main idea: similar instances have similar class labels

- The classification model consists of the examples from the training dataset
  - The training process consists only on storing the training examples

- The classification of a new data instance consists of:
  - Compute the similarities (or dissimilarities) between the new data instance and the examples in the training set and find the most similar examples
  - Choose the most frequent class from the set of the similar examples

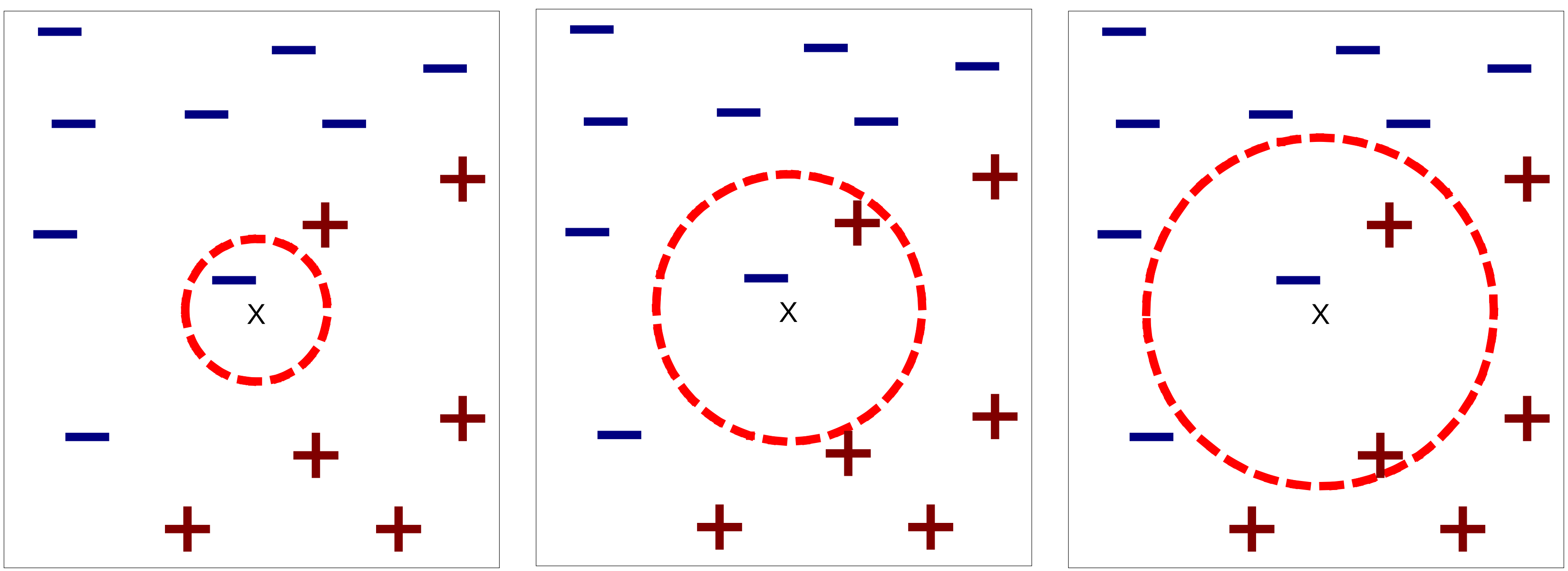
Remarks:

- Such classifiers are usually called “lazy” because the training step does not involve any effort (the entire computational effort is postponed for the classification step)
- The most popular lazy classifiers are those based on the Nearest Neighbors

Instance based classifiers

kNN – k Nearest Neighbour

- For each data to be classified:
  - Find the closest (most similar) k examples from the training dataset
  - Identify the most frequent class



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

[Tan, Steinbach, Kumar; Introduction to Data Mining, slides, 2004]

Instance based classifiers

kNN – k Nearest Neighbour

- For each data to be classified:
    - Find the closest (most similar) k examples from the training dataset
    - Identify the most frequent class

The performance of kNN classifiers depend on:

- The similarity/ dissimilarity (distance) measure
    - It depends on the attribute types and on the particularities of the problem
- The value of k (the number of neighbors)
    - Simplest case: k=1 (not very good in the case of noisy data)

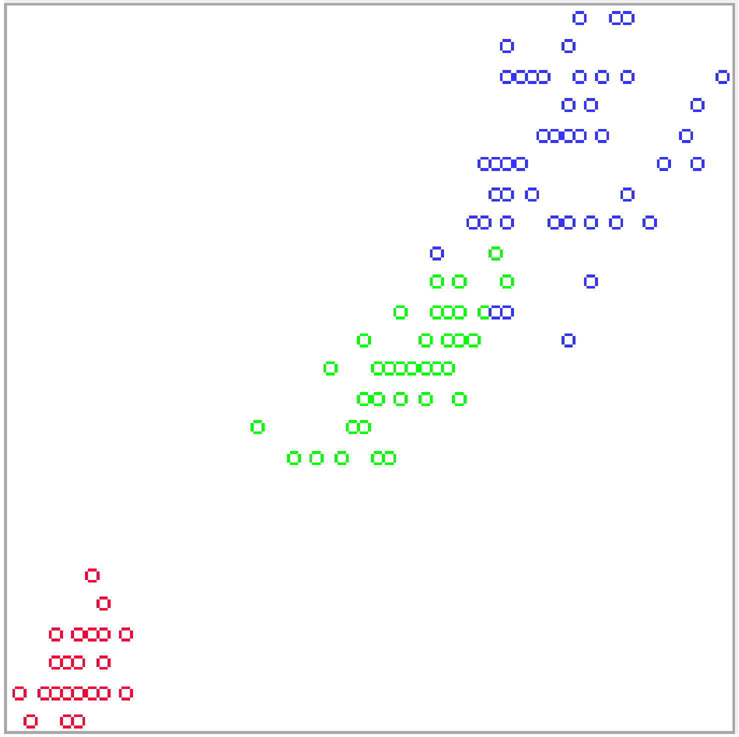
Remark: kNN induces a partition of the data space in regions; the regions are not explicitly computed but are implicitly determined by the similarity measure (and the value of k)

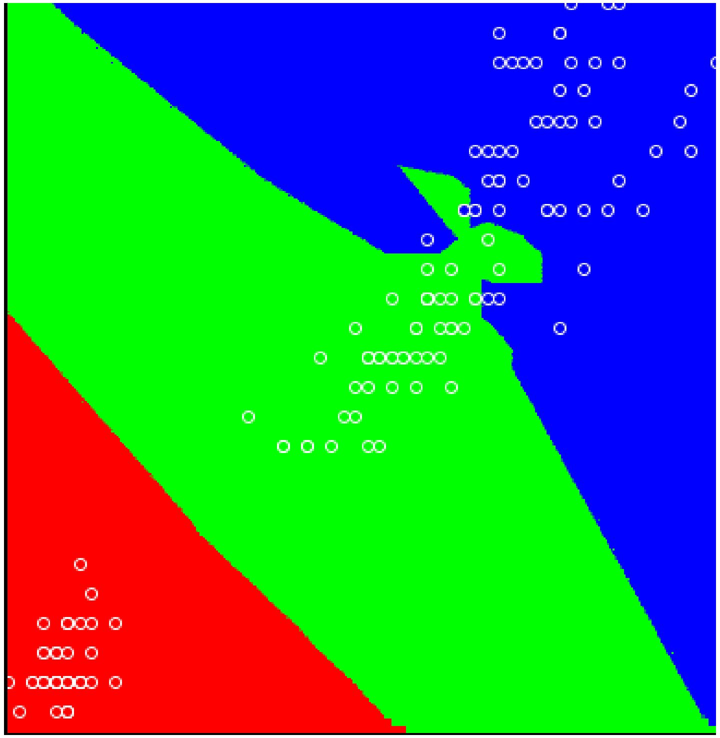
Instance based classifiers

1NN = Nearest Neighbor with one closest neighbor (based on the normalized Euclidean distance)

Illustration of the geometric boundaries. Dataset: iris2D ("petal length" and "petal width").

Plot: Weka->Visualization->BoundaryVisualizer





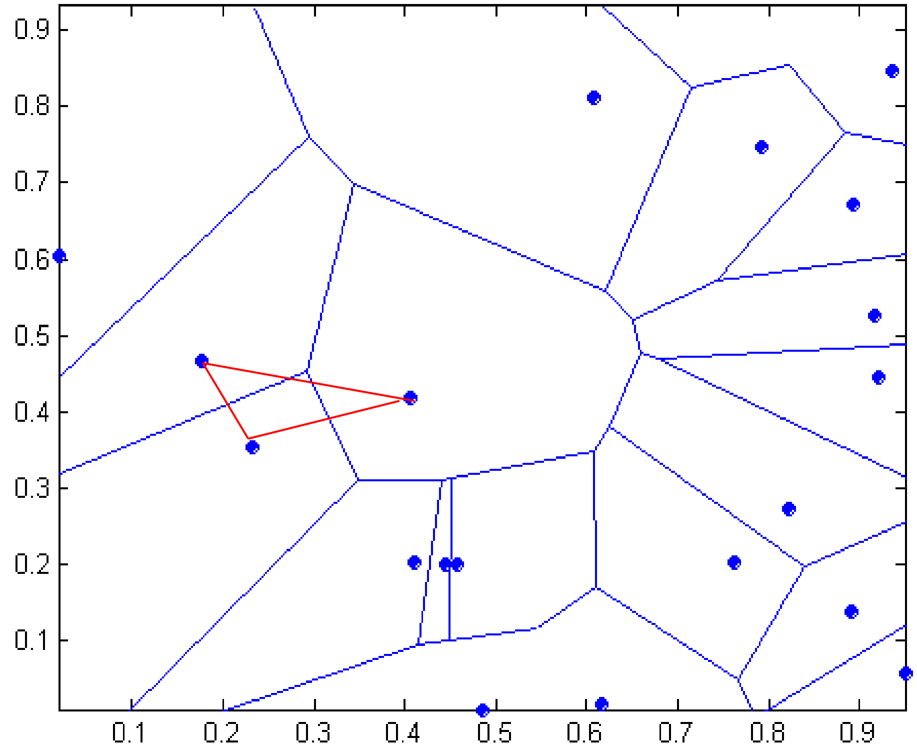
Instance based classifiers

1NN = Nearest Neighbor with one closest neighbor (based on the normalized Euclidean distance)

1NN induces a partitioning of the data space (e.g. In 2D this corresponds to a Voronoi diagram)

Remark:

Each instance in the training dataset has associated a region containing all data which are in the neighborhood of that training instance



[Tan, Steinbach, Kumar: Introduction to Data Mining, slides, 2004]

Similarity/ dissimilarity measures

Let us consider two entities (e.g. data vectors, time series etc) A and B

- A similarity measure S(A,B) is a number which is higher if A and B are more similar
- A dissimilarity measure D(A,B) is a number which is higher if A and B are less similar (or more different)

The choice of an appropriate measure depends on:

- The type of attributes
- The number of attributes
- The distribution of data

Similarity/ dissimilarity measures

Numerical attributes

Most popular dissimilarity measures:

- Euclidean distance
- Manhattan distance

$$d_p(A,B) = \sqrt[p]{\sum_{i=1}^{n}(a_i - b_i)^p} \quad (\text{Minkowski, L}_\text{p})$$

$$d_E(A,B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \quad (\text{Euclidean, p} = 2)$$

$$d_M(A,B) = \sum_{i=1}^{n}| a_i - b_i | \quad (\text{Manhattan, p} = 1)$$

$$d_\infty(A,B) = \max_{i=\overline{1,n}} | a_i - b_i | \quad (\text{p} = \infty)$$

Remarks:

- The Euclidean distance is invariant with respect to rotations
- If not all attributes have the same importance then weights should be included in the distance (e.g. $w_i(a_i-b_i)^2$ instead of $(a_i-b_i)^2$)

The weights can be estimated by using preprocessing methods; instead of using explicit weights the data can be normalized/ standardized

Similarity/ dissimilarity measures

Practical issues – dimensionality curse:

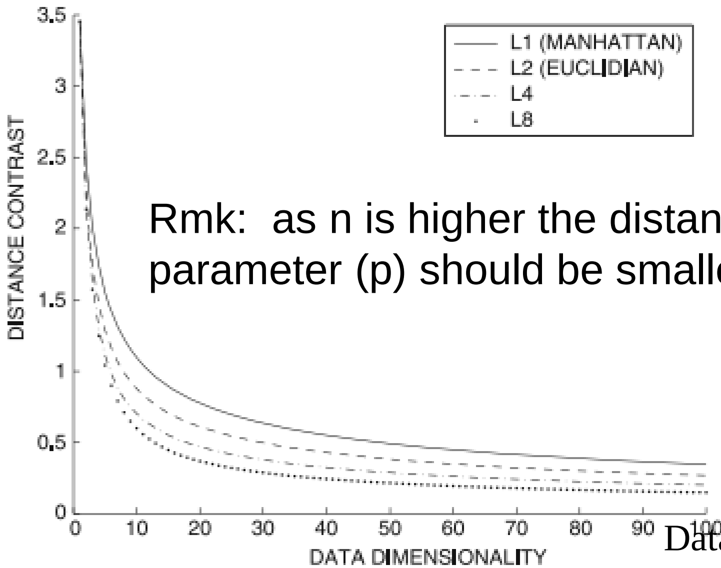
- These distances lose their discriminative power as the size (n) increases => for high-dimensional data the distance-based classifiers may be qualitatively ineffective

$$d_p(A,B)=\sqrt[p]{\sum_{i=1}^{n}(a_i-b_i)^p}\quad (\text{Minkowski, } \mathrm{L_p})$$

$$d_E(A,B)=\sqrt{\sum_{i=1}^{n}(a_i-b_i)^2}\quad (\text{Euclidean, p}=2)$$

$$d_M(A,B)=\sum_{i=1}^{n}|a_i-b_i|\quad (\text{Manhattan, p}=1)$$

$$d_\infty(A,B)=\max_{i=\overline{1,n}}|a_i-b_i|\quad (\mathrm{p}=\infty)$$



Rmk: as n is higher the distance parameter (p) should be smaller

$$\text{Distance contrast}:\ \frac{d_{\max}-d_{\min}}{\sigma}$$

$$d_{\max}, d_{\min} = \text{largest and smallest distance}$$

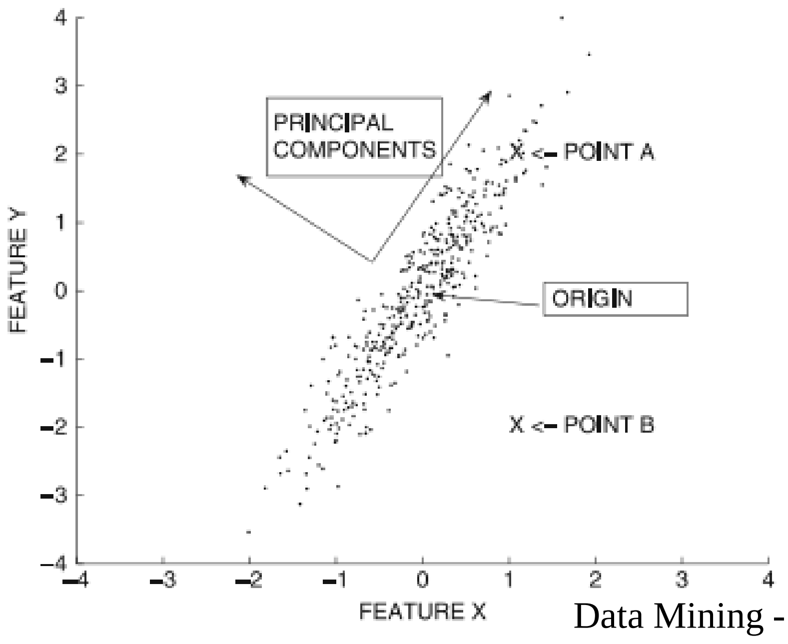
$$\sigma = \text{standard deviation of distances}$$

Aggarwal, Data Mining Textbook, 2015

Similarity/ dissimilarity measures

Practical issues – impact of data distribution

Q: Which is closer to the origin? Point A or point B?



Aggarwal, Data Mining Textbook, 2015

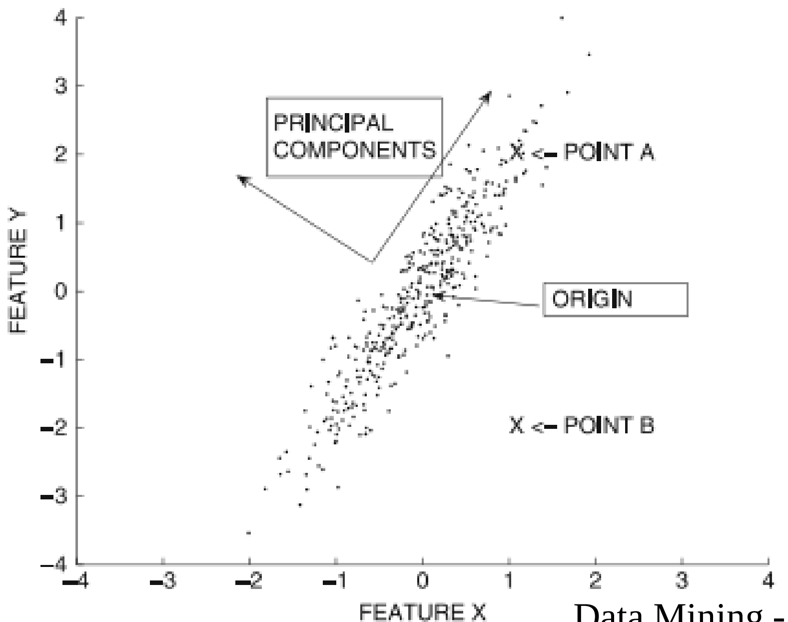
Similarity/ dissimilarity measures

Practical issues – impact of data distribution

Q: Which is closer to the origin? Point A or point B?

A: The Euclidean distances d(O,A) and d(O,B) are equal. However taking into account the data distribution A is more in agreement with the data distribution than B, thus it should be considered that A is "closer" to O than B

New question: how can be quantified the data distribution in the distance?



Mahalanobis distance

$$d_{Mah}(A,B)=\sqrt{(A-B)^T C^{-1}(A-B)}$$

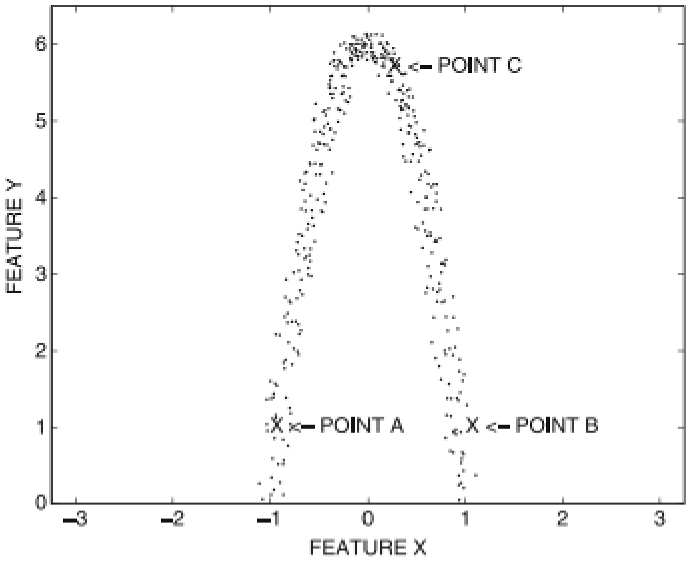
C^{-1} = inverse of the covariance matrix

Aggarwal, Data MiningTextbook, 2015

Similarity/ dissimilarity measures

Practical issues – impact of data distribution

Q: Is the distance between A and B smaller than the distance between B and C?



Aggarwal, Data Mining Textbook, 2015

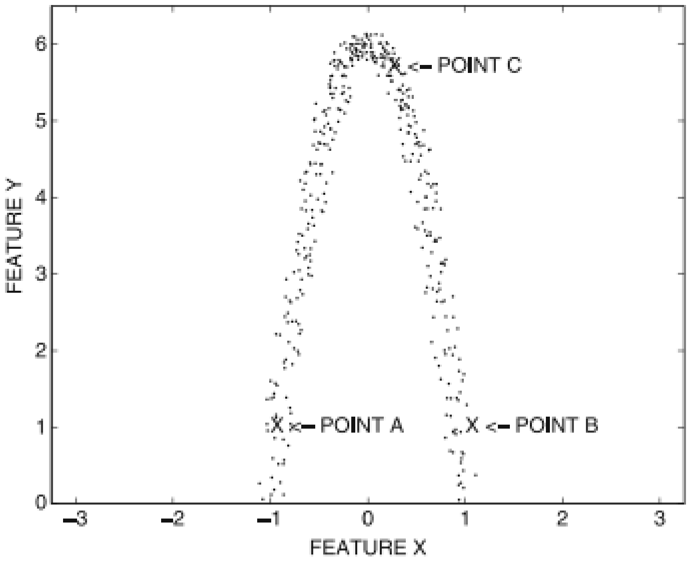
Similarity/ dissimilarity measures

Practical issues – impact of data distribution

Q: Is the distance between A and B smaller than the distance between B and C?

A: Yes, if we ignore the data distribution and we use the Euclidean distance

However, the data distribution provides a context for the problem to be solved and in this context d(A,B)>d(B,C)



Geodesic distance:

- Construct a graph by using the points as nodes and by defining edges between nearest neighbors (each point is connected to its first k nearest neighbors)
- Compute the distance between two points as the shortest path in the graph.

Aggarwal, Data Mining Textbook, 2015

Similarity/ dissimilarity measures

Numerical attributes – similarity measure

- Cosine measure: sim(A,B)=A^TB/(||A|| ||B||) (scalar product between A and B divided by the product of the norms)

Remark:

- In the case of normalized data vectors (||A||=||B||=1) the similarity between the vectors is maximal when the Euclidean distance between them is minimal:

$$d_E^2(A,B) = (A-B)^T(A-B) = A^T A - 2A^T B + B^T B =$$
$$= 2(1\text{-}A^T B) = 2(1\text{-}sim(A,B))$$

Similarity/ dissimilarity measures

Nominal attributes

Approach 1: Transformation of nominal attributes in numerical ones (by binarization) and use similarity/dissimilarity measures for binary vectors:

- Dissimilarity: Hamming distance = Manhattan distance: $d_H(A,B)=d_M(A,B)$
- Similarity: Jaccard coefficient

$$J(A,B)=\frac{\sum_{i=1}^{n} a_i b_i}{\sum_{i=1}^{n}(a_i^2+b_i^2-a_i b_i)}=\frac{card(S_A \cap S_B)}{card(S_A \cup S_B)}$$

Remark: S_A and S_B are the subsets of the global set of n attributes which correspond to the membership vectors A and B, respectively

Similarity/ dissimilarity measures

Nominal attributes

Approach 2: Use local similarity measures (between attribute values)

$$S(A,B)=\sum_{i=1}^{n} S(a_i,b_i)$$

$$S(a_i,b_i)=\begin{cases} 1 & \text{if } a_i=b_i \\ 0 & \text{if } a_i\neq b_i \end{cases}$$

Remark: the similarities which are unusual should be considered more significant than those which are frequent

$$S(a_i,b_i)=\begin{cases} 1/f^2(a_i) & \text{if } a_i=b_i \\ 0 & \text{if } a_i\neq b_i \end{cases}$$

$$f(a_i)=\text{frequency of the value } a_i \text{ in the dataset}$$

$$\text{(for i-th attribute)}$$

Similarity/ dissimilarity measures

Mixed attributes: the measures corresponding to numerical and nominal attributes should be aggregated (using specific weights)

$$S(A,B) = \lambda S_{numerical}(A,B) + (1-\lambda) S_{nominal}(A,B)$$

Other types of data:

- Strings (e.g. text or biological sequences) - use of the edit distance
- Concepts (e.g. nodes in an ontology) – path-based distances in trees/ graphs
- Graphs (e.g. social or biological networks) – amount of similar small structures (patterns)

Nearest Neighbour: choice of k

The performance of kNN is sensitive to the number (k) of neighbours

Extreme cases:

- k=1 - the classifier is not robust (errors in the dataset could mislead the classification result)
- k=N - it is equivalent to ZeroR being based only on the distribution of the data in classes (the sensitivity of the classifier to the underlying data is lost)

How to choose k?

- Trial-and-error approach: try different values and choose the one providing the best performance

Nearest Neighbour: computational cost

In an naïve implementation the classification step requires the computation of N distances (or similarities) for a dataset with N elements (d-dimensional vectors) and the selection of the smallest k distances ➔ O(Nd+kN)

When N is large this could be costly (as it should be done for each new data to classify)

Possible approaches:

- Create indexing data structures allowing to find the k nearest neighbours in an efficient way
- Identify small clusters in the data set and replace each such cluster with a single prototype
- Use instance (prototype) selection = select the most representative instances from the dataset (not easy to decide which ones are representative)

Probabilistic Classification

Idea: construct a model which estimates the probability of a data instance to belong to a given class

Aim: estimate $P(C_k|D_i)$ = probability that the class of data instance D_i is C_k

Remark: P(B|A) is a conditional probability = probability of event B given that (by assumption, presumption, assertion or evidence) event A has occurred

Reminder on probability theory:

$$\text{Conditional probability: } P(B\,|\,A) = \frac{P(A,B)}{P(A)}$$

$$P(A,B) = \text{probability that both events A and B occured}$$

$$\text{Bayes rule: } P(B\,|\,A) = \frac{P(A\,|\,B)P(B)}{P(A)}$$

Bayes rule is useful to estimate the posterior probability P(B|A) when the prior probability P(B) and the other probabilities P(A|B), P(A) can be easy estimated.

Probabilistic Classification

Example: Let us suppose that we are interested to estimate the probability that a patient having some symptom S has the illness T

- We want to estimate P(T|S)

- Let us suppose that we know:

  - P(T) – estimated based on population studies (how frequent is the illness)
  - P(S|T) – estimated based on prior medical knowledge (how often is the symptom present in the case of illness T)
  - P(S) = 1 – the symptom exists (it corresponds to an event which is realized)

- P(T|S)=P(S|T)P(T)/P(S)=P(S|T)P(T)

- What about the case when there is not only one symptom S, but a list of symptoms $S_1,S_2,\ldots,S_n$?

Probabilistic Classification

Example: Let us suppose that we are interested to estimate the probability that a patient having some symptoms $S_1, S_2, \ldots, S_n$ has the illness T

- In this case one have to estimate $P(T | S_1, S_2, \ldots, S_n)$
- Based on the Bayes rule:
  - $P(T | S_1, S_2, \ldots, S_n) = P(S_1, S_2, \ldots, S_n | T) P(T) / P(S_1, S_2, \ldots, S_n)$
- Simplifying assumption: the symptoms $(S_1, S_2, \ldots, S_n)$ are independent events (this is not always true but many practical situations this assumption can be accepted)
- Considering that $P(S_1, S_2, \ldots, S_n) = 1$ (the symptoms are all present)

$$P(T | S_1, S_2, \ldots, S_n) = P(S_1|T) P(S_2|T) \ldots P(S_n|T) P(T)$$

Naïve Bayes Classifier

Classification problem:

- For a data instance $D_i=(a_{i1},a_{i2},\ldots,a_{in})$ find the class to which it belongs

Main idea

- Estimate $P(C_k| D_i)=P(C_k|a_{i1},a_{i2},\ldots,a_{in})\, P(C_k)$ for all k in {1,2,…,K} and select the maximal probability; it will indicate the class to which the data instance most probably belongs
- Simplifying assumption: the attributes are independent (this is why the method is called "naive")
- $P(C_k| D_i)= P(a_{i1}|C_k)\, P(a_{i2}|C_k)\ldots P(a_{in}|C_k)P(C_k)$
- This requires the knowledge of $P(a_{i1}|C_k)$, $P(a_{i2}|C_k)$, …, $P(a_{in}|C_k)$ and $P(C_k)$
- These probabilities can be estimated from the dataset (as relative frequencies) – this is the learning process corresponding to the Naïve Bayes

Naïve Bayes Classifier

Example:

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

P(C1)=P(no)=5/14 P(C2)= P(yes)=9/14

A1: outlook

P(sunny|C1)=P(sunny,C1)/P(C1)
=(3/14)/(5/14)=3/5

P(sunny|C2)=P(sunny,C2)/P(C2)
=(2/14)/(9/14)=2/9

P(overcast|C1)=P(overcast,C1)/P(C1)
=0

P(overcast|C2)=P(overcast,C2)/P(C2)
=(4/14)/(9/14)=4/9

P(rainy|C1)=P(rainy,C1)/P(C1)
=(2/14)/(5/14)=2/5

P(rainy|C2)=P(rainy,C2)/P(C2)
=(3/14)/(9/14)=3/9

Naïve Bayes Classifier

Example:

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	**play** Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

P(C1)=P(no)=5/14   P(C2)= P(yes)=9/14

A2: temperature

P(hot|C1)=P(hot,C1)/P(C1)=2/5

P(hot|C2)=P(hot,C2)/P(C2)=2/9

P(mild|C1)=P(mild,C1)/P(C1)=2/5

P(mild|C2)=P(mild,C2)/P(C2)=4/9

P(cool|C1)=P(cool,C1)/P(C1)
=(2/14)/(5/14)=1/5

P(cool|C2)=P(cool,C2)/P(C2)=2/9

Naïve Bayes Classifier

Example:

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	**play** Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

P(C1)=P(no)=5/14   P(C2)= P(yes)=9/14

A3: humidity

P(high|C1)=P(high,C1)/P(C1)=4/5

P(high|C2)=P(high,C2)/P(C2)=3/9

P(normal|C1)=P(normal,C1)/P(C1)=1/5

P(normal|C2)=P(normal,C2)/P(C2)=6/9

Naïve Bayes Classifier

Example:

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	**play** Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

$P(C1)=P(no)=5/14$ $P(C2)= P(yes)=9/14$

A4: windy

$P(FALSE|C1)=P(FALSE,C1)/P(C1)=2/5$

$P(FALSE|C2)=P(FALSE,C2)/P(C2)=6/9$

$P(TRUE|C1)=P(TRUE,C1)/P(C1)=3/5$

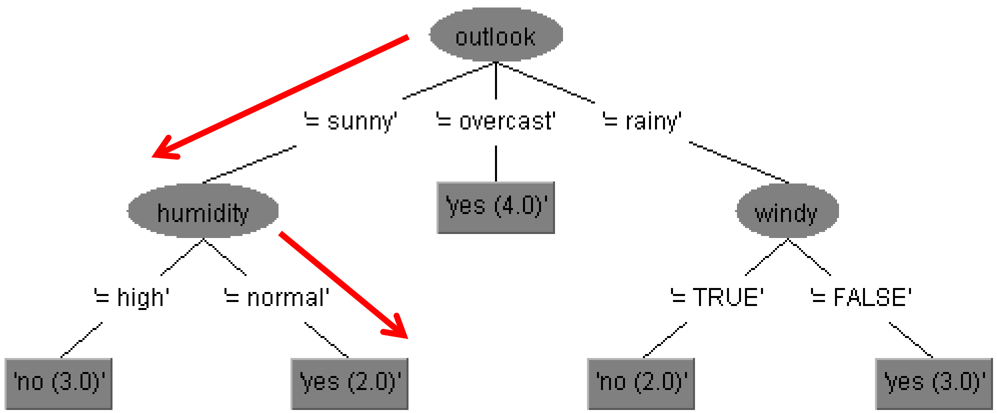
$P(TRUE|C2)=P(TRUE,C2)/P(C2)=3/9$

Naïve Bayes Classifier

Example:

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	**play** Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no



D=(outlook=sunny, temperature=mild, humidity=normal, windy=False)

P(C1|D)=P(sunny|C1)*P(mild|C1)*P(normal|C1)*P(FALSE|C1)*P(C1)/P(D)=
=3/5*2/5*1/5*2/5*5/14/P(D)=6/125

P(C2|D)=P(sunny|C2)*P(mild|C2)*P(normal|C2)*P(FALSE|C2)*P(C2)/P(D)=
=2/9*4/9*6/9*6/9*9/14/P(D)=144/729 ➔ yes

Naïve Bayes Classifier

Example:

Relation: weather.symbolic

No.	outlook Nominal	temperature Nominal	humidity Nominal	windy Nominal	play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

Remark: if for a given attribute value a_{ij} and a given class C_k there is no example in the training set, then $P(a_{ij}| C_k)=0$ and (because of the independency assumption) for any instance having the value a_{ij} for attribute A_i the probability to belong to C_k is 0.

This situation might appear especially in the case of small classes

Laplace smoothing:

$P(a_{ij}| C_k)=(count(a_{ij},C_k)+alpha)/(count(C_k)+m_i*alpha)$
alpha= Laplace smoothing parameter
m_i= number of distinct values of attribute A_i

Naïve Bayes Classifier

Remarks:

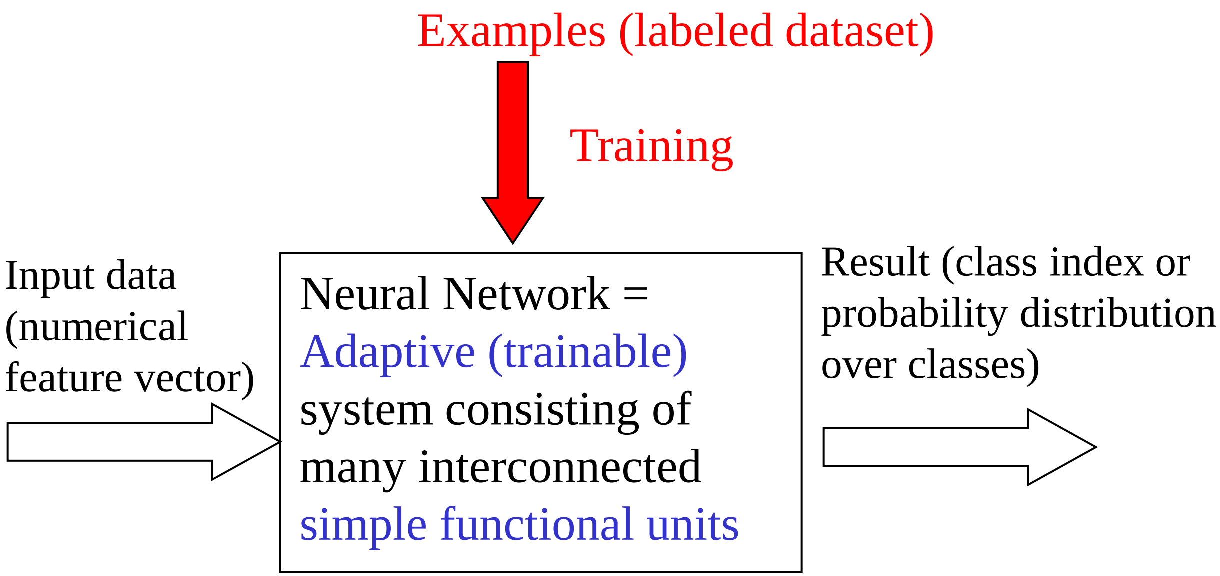
- This classifier can be directly applied for discrete attributes and it is based on the following probabilistic models:
  - Binomial model
  - Multinomial model

- In the case of real attributes there are two main approaches:
  - The attributes are discretized before using the classifier (the classifier performance depends on the discretization process)
  - The attributes are modeled through continuous probabilistic models (e.g. Gaussian) with parameters estimated based on the training data

Artificial Neural Networks

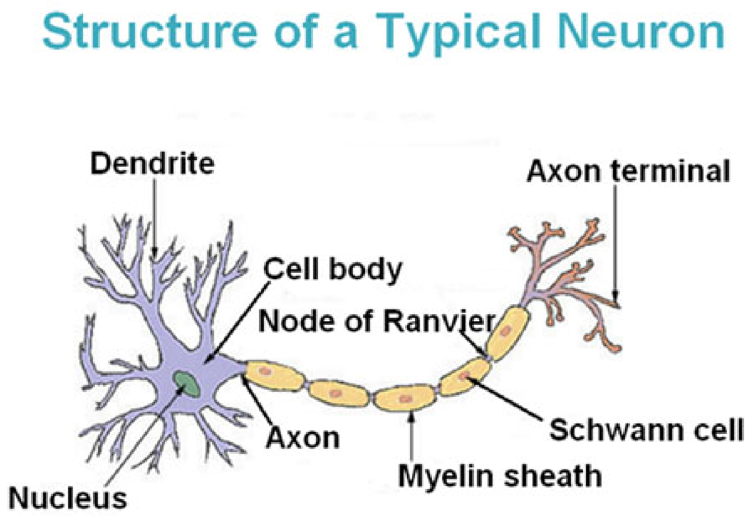
Particularities:

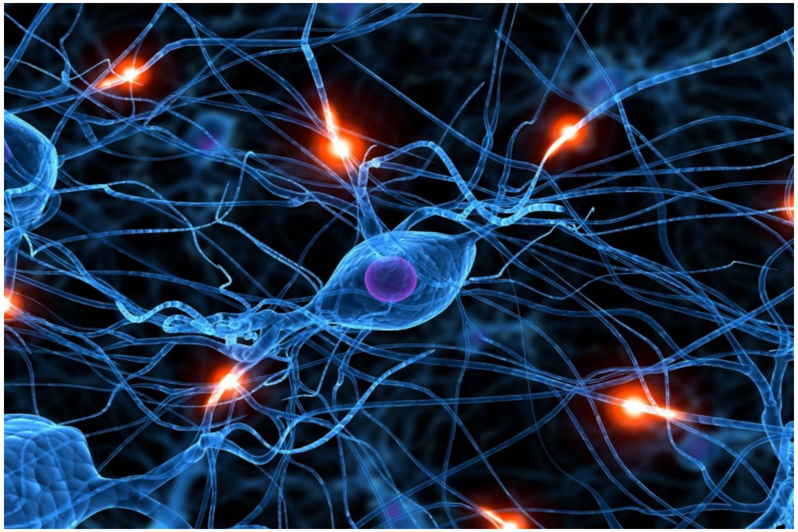
- Artificial neural networks are black-box classifiers, i.e. they just predict the class to which a given data belongs without providing an explicit classification rule



Artificial Neural Networks

- Particularities:
  - They are inspired by the structure and functioning of the brain = system of highly interconnected neurons





Artificial Neural Networks

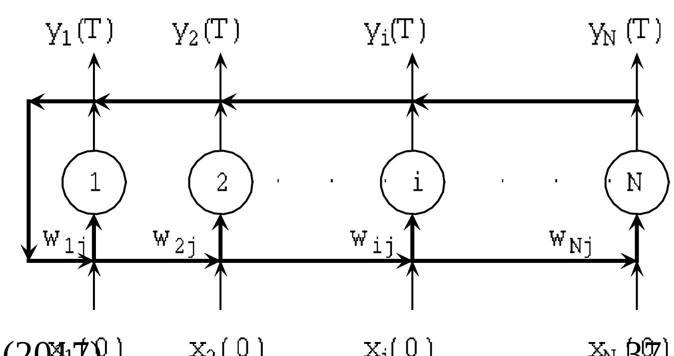
- ANNs are sets of interconnected artificial neurons (functional units)
  - Each neuron receives some input signals and produces an output signal
  - The neural network receives an input vector (through the input neurons) and produces an output vector (through the output neurons)
- The main aspects of an ANN:
  - Architecture = directed weighted graph having artificial neurons as nodes and edges marking the connections; each edge has a numerical weight which models the synaptic permeability
  - Functioning = the process through which the network transforms an input vector in an output vector
  - Training = the process through which are established the values of the synaptic weights (and other parameters of the network)

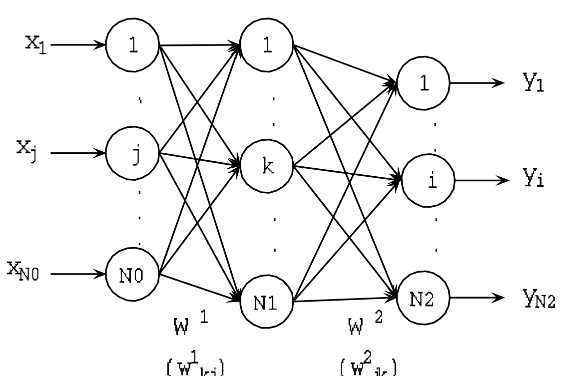
Artificial Neural Networks

Main NN architectures:

- Feed-forward:
  - the support graph does not contain cycles (the neurons are usually placed on several layers)
  - The output signal can be computed by composition of some aggregation and activation (transfer) functions (see next slides)
- Recurrent:
  - The support graph contain cycles
  - The output signal is computed by simulating a dynamical system (iterative process)

Recurrent (fully connected network)





Feed-forward (multilayer perceptron)

ANN Design

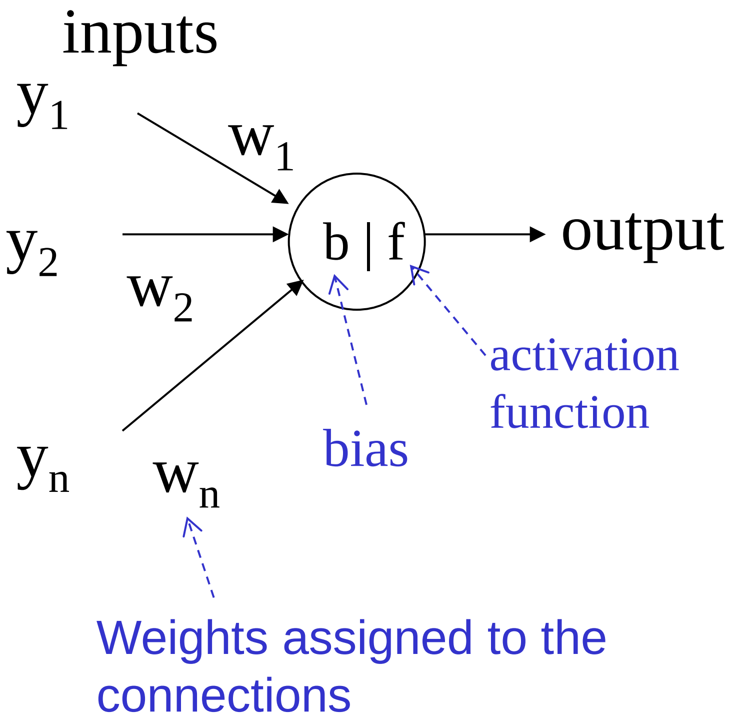
Steps to follow in the design of a neural network:

- Choose the architecture: number of layers, number of units on each layer, activation functions, interconnection style
- Train the network: compute the values of the weights using the training set and a learning algorithm.
- Validate/test the network: analyze the network behavior for data which do not belong to the training set

Remarks:

- in the context of classifying N-dimensional data in M classes the ANN should have:
  - N input units
  - M output units
- the classification model is incorporated in the synaptic weights (attached to the inter-connection edges)

Functional units (artificial neurons)



Functional unit: several inputs, one output

Notations:

- input signals: $y_1, y_2, \ldots, y_n$
- synaptic weights: $w_1, w_2, \ldots, w_n$ (they model the synaptic permeability)
- threshold (bias): b (or theta) - it models the activation threshold of the neuron
- Output: y

Remark: All these values are usually real numbers (there exist also complex networks which have complex numbers as weights)

Functional units (artificial neurons)

Output signal generation:

- The input signals are “combined” by using the connection weights and the threshold (bias)
  - The obtained value corresponds to the local potential of the neuron
  - This “combination” is obtained by applying a so-called aggregation function
- The output signal is constructed by applying an activation (transfer) function
  - It corresponds to the pulse signals propagated along the axon

Input signals ($y_1, \ldots, y_n$) → Neuron’s state (u) → Output signal (y)

Aggregation function

Activation (transfer) function

Functional units (artificial neurons)

Aggregation functions:

Most popular

Weighted sum

$$u = \sum_{j=1}^{n} w_j y_j - w_0$$

Euclidean distance

$$u = \sqrt{\sum_{j=1}^{n} (w_j - y_j)^2}$$

$$u = \prod_{j=1}^{n} y_j^{w_j}$$

Multiplicative neuron

$$u = \sum_{j=1}^{n} w_j y_j + \sum_{i,j=1}^{n} w_{ij} y_i y_j + \ldots$$

High order connections

Remark: in the case of the weighted sum the threshold can be interpreted as a synaptic weight which corresponds to a virtual unit which always produces the value -1

$$u = \sum_{j=0}^{n} w_j y_j$$

Functional units (artificial neurons)

Activation functions:

$$f(u) = \mathrm{sgn}(u) = \begin{cases} -1 & u \le 0 \\ 1 & u > 0 \end{cases}$$ signum

$$f(u) = H(u) = \begin{cases} 0 & u \le 0 \\ 1 & u > 0 \end{cases}$$ Heaviside

$$f(u) = \begin{cases} -1 & u < -1 \\ u & -1 \le u \le 1 \\ 1 & u > 1 \end{cases}$$ Saturated linear

$$f(u) = u$$ linear

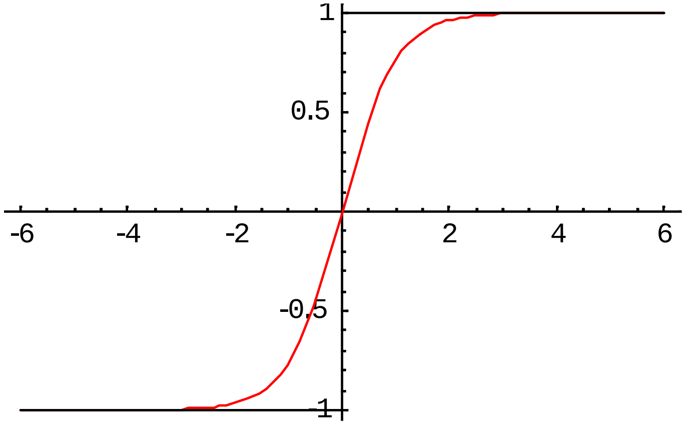
$$f(u) = \max\{0, u\}$$ Rectified linear – used in deep networks

Functional units (artificial neurons)

Sigmoidal activation functions

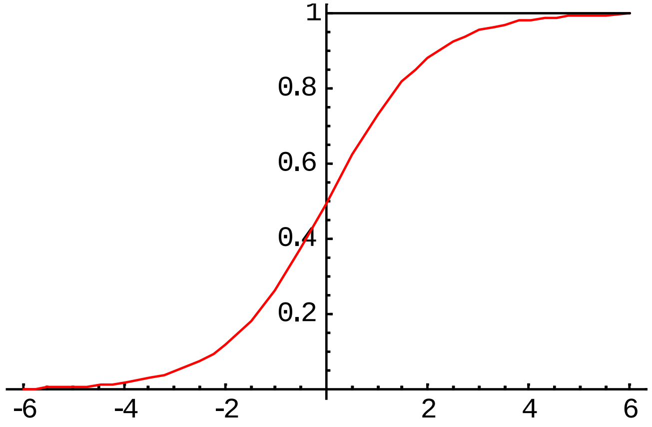
(Hyperbolic tangent)

$$f(u) = \tanh(u) = \frac{\exp(2u) - 1}{\exp(2u) + 1}$$



$$f(u) = \frac{1}{1 + \exp(-u)}$$

(Logistic)

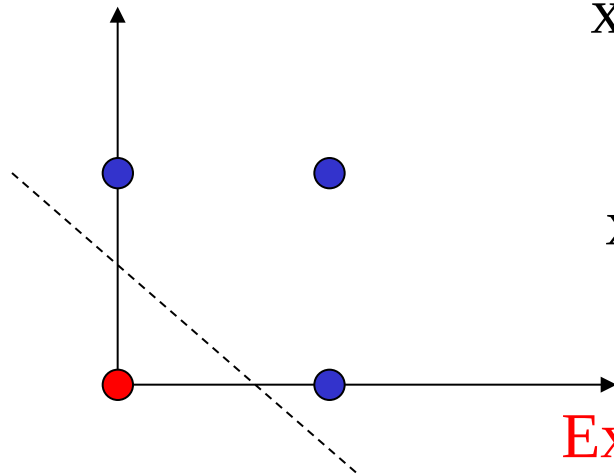


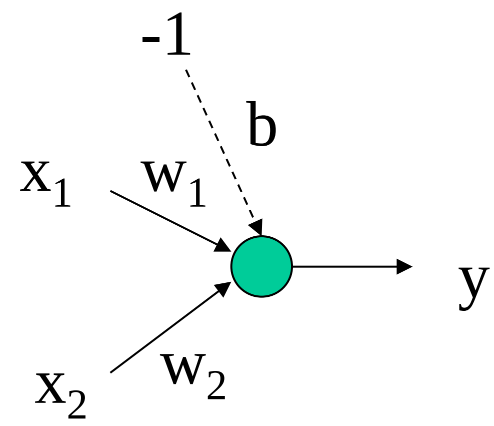
Functional units (artificial neurons)

- What can do a single neuron ?
- It can solve simple problems (linearly separable problems)

OR

	0	1
0	0	1
1	1	1





$y=H(w_1x_1+w_2x_2-b)$

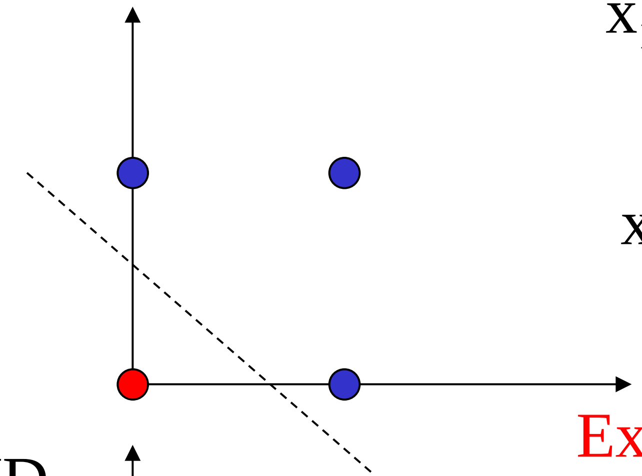
Ex: $w_1=w_2=1, w_0=0.5$

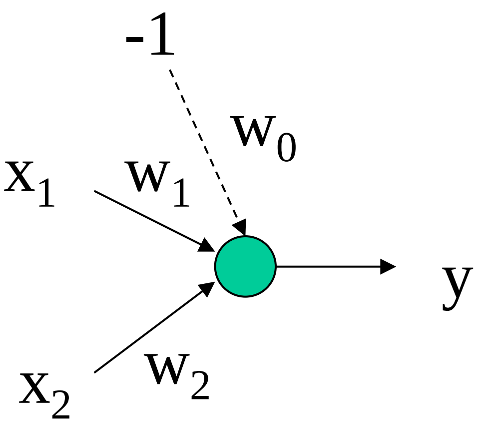
Functional units (artificial neurons)

- What can do a single neuron ?
- It can solve simple problems (linearly separable problems)

OR

	0	1
0	0	1
1	1	1



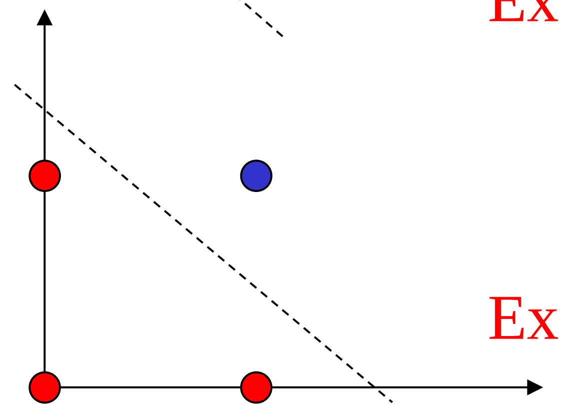


$y=H(w_1x_1+w_2x_2-w_0)$

Ex: $w_1=w_2=1,\ w_0=0.5$

AND

	0	1
0	0	0
1	0	1

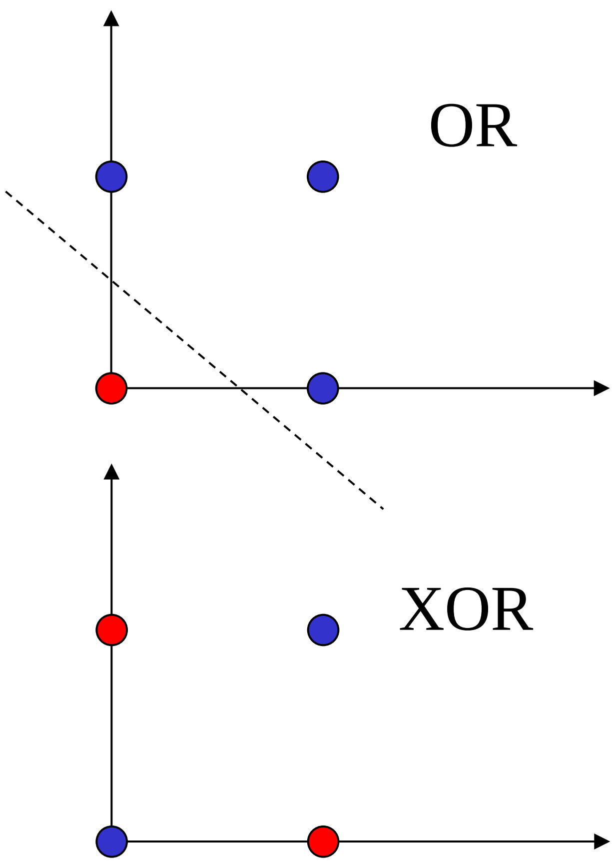


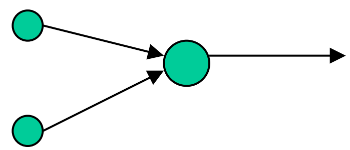
$y=H(w_1x_1+w_2x_2-w_0)$

Ex: $w_1=w_2=1,\ w_0=1.5$

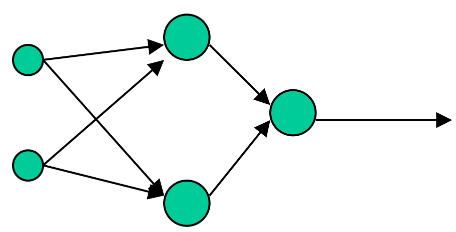
Functional units (artificial neurons)

Representation of boolean functions: $f:\{0,1\}^2 \rightarrow \{0,1\}$





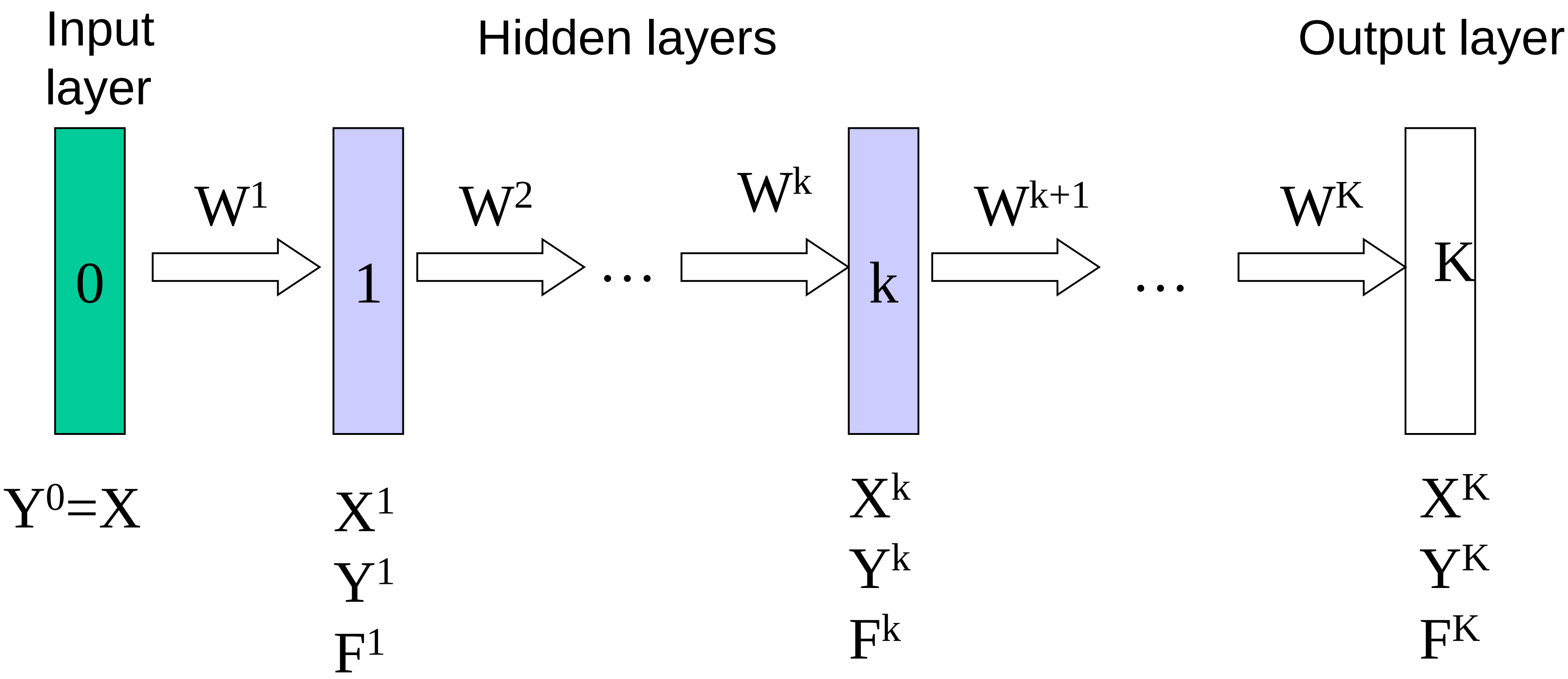
Linearly separable problem: one layer network



Nonlinearly separable problem: multilayer network

Architecture and notations

Feedforward network with K layers



X = input vector, Y= output vector, F=vectorial activation function

Functioning

Computation of the output vector

$$Y^K = F^K(W^K F^{K-1}(W^{K-1} \ldots F^1(W^1 X)))$$

$$Y^k = F^k(X^k) = F(W^k Y^{k-1})$$

FORWARD Algorithm (propagation of the input signal toward the output layer)

Y[0]:=X (X is the input signal)

FOR k:=1,K DO

X[k]:=W[k]Y[k-1]

Y[k]:=F(X[k])

ENDFOR

Rmk:

- Y[K] is the output of the network
- Interpretation of the results: for a given data vector X the index of the functional unit which produces the largest value is the class label

A particular case

One hidden layer

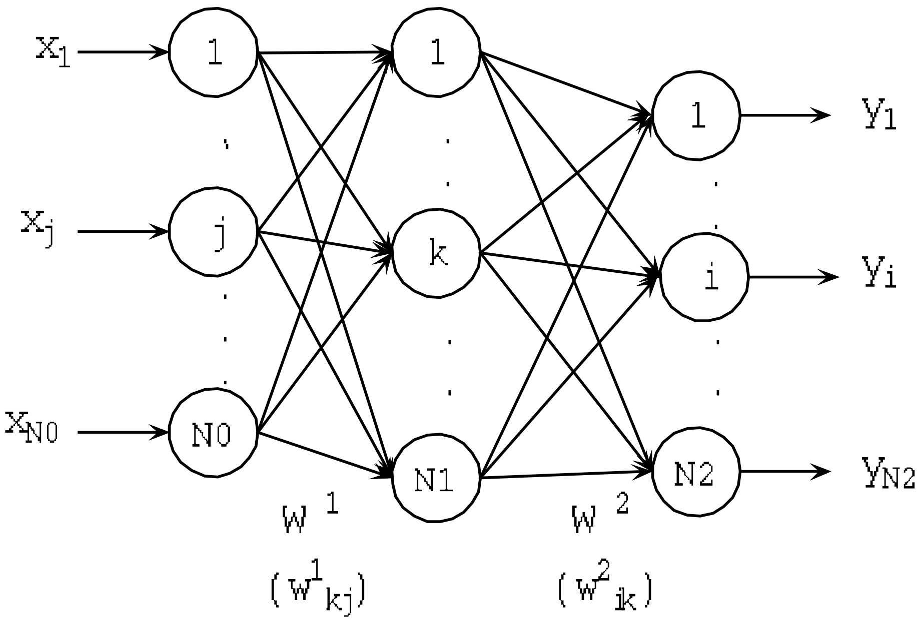
Adaptive parameters: $W^{(1)}$, $W^{(2)}$

$$y_i = f_2\left(\sum_{k=0}^{N1} w^{(2)}{}_{ik}\, f_1\left(\sum_{j=0}^{N0} w^{(1)}{}_{kj}\, x_j\right)\right)$$

A simpler notation :

$$w^{(2)}{}_{ik} = w_{ik};$$

$$w^{(1)}{}_{kj} = w_{kj}$$



Remark:

Traditionally only 1 or 2 hidden layers are used

Lately, architectures involving many hidden layers became more popular (Deep Neural Networks) – they are used mainly for image and language processing (http://deeplearning.net)

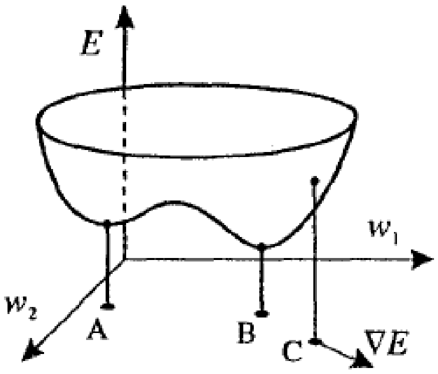
Learning process

Learning based on minimizing an error function

- Training set: $\{(x^1,d^1), \ldots, (x^L,d^L)\}$
- Error function (mean squared error):

$$E(W)=\frac{1}{2L}\sum_{l=1}^{L}\sum_{i=1}^{N2}\left(d_i^l - f_2\left(\sum_{k=0}^{N1} w_{ik} f_1\left(\sum_{j=0}^{N0} w_{kj}x_j\right)\right)\right)^2$$

- Aim of learning process: find W which minimizes the error function
- Minimization method: gradient method



Learning process

Gradient based adjustement

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E(w(t))}{\partial w_{ij}}$$

Learning rate

$$E(W) = \frac{1}{2L}\sum_{l=1}^{L}\underbrace{\sum_{i=1}^{N2}\left(d_i^l - \underbrace{f_2\left(\underbrace{\sum_{k=0}^{N1} w_{ik}\, \underbrace{f_1\left(\underbrace{\sum_{j=0}^{N0} w_{kj} x_j}_{x_k}\right)}_{y_k}}_{x_i}\right)}_{y_i}\right)^2}_{E_l(W)}$$

Learning process

- Partial derivatives computation

$$E(W)=\frac{1}{2L}\sum_{l=1}^{L}\sum_{i=1}^{N2}\left(d_i^l-\underbrace{f_2\left(\underbrace{\sum_{k=0}^{N1} w_{ik}\,\underbrace{f_1\left(\underbrace{\sum_{j=0}^{N0} w_{kj}x_j}_{x_k}\right)}_{y_k}}_{x_i}\right)}_{y_i}\right)^2$$

$$\frac{\partial E_l(W)}{\partial w_{ik}}=-(d_i^l-y_i)\,f_2^{'}(x_i)\,y_k=-\delta_i^l y_k$$

$$\frac{\partial E_l(W)}{\partial w_{kj}}=-\sum_{i=1}^{N2} w_{ik}(d_i^l-y_i)\,f_2^{'}(x_i)\,f_1^{'}(x_k)\,x_j=-\left(f_1^{'}(x_k)\sum_{i=1}^{N2} w_{ik}\delta_i^l\right)x_j=-\delta_k^l x_j$$

$$E_l(W)=\frac{1}{2}\sum_{i=1}^{N2}\left(d_i^l-f_2\left(\sum_{k=0}^{N1} w_{ik}\,f_1\left(\sum_{j=0}^{N0} w_{kj}x_j\right)\right)\right)^2$$

Learning process

- Partial derivatives computation

$$\frac{\partial E_l(W)}{\partial w_{ik}} = -(d_i^l - y_i) f_2^{'}(x_i) y_k = -\delta_i^l y_k$$

$$\frac{\partial E_l(W)}{\partial w_{kj}} = -\sum_{i=1}^{N2} w_{ik} (d_i^l - y_i) f_2^{'}(x_i) f_1^{'}(x_k) x_j = -\left(f_1^{'}(x_k) \sum_{i=1}^{N2} w_{ik} \delta_i^l \right) x_j = -\delta_k^l x_j$$

$$E_l(W) = \frac{1}{2} \sum_{i=1}^{N2} \left(d_i^l - f_2\left(\sum_{k=0}^{N1} w_{ik} f_1\left(\sum_{j=0}^{N0} w_{kj} x_j \right)\right)\right)^2$$

Remark:

The derivatives of sigmoidal activation functions have particular properties:

Logistic: f'(x)=f(x)(1-f(x))=y(1-y)

Tanh: f'(x)=1-f^2(x)=1-y^2

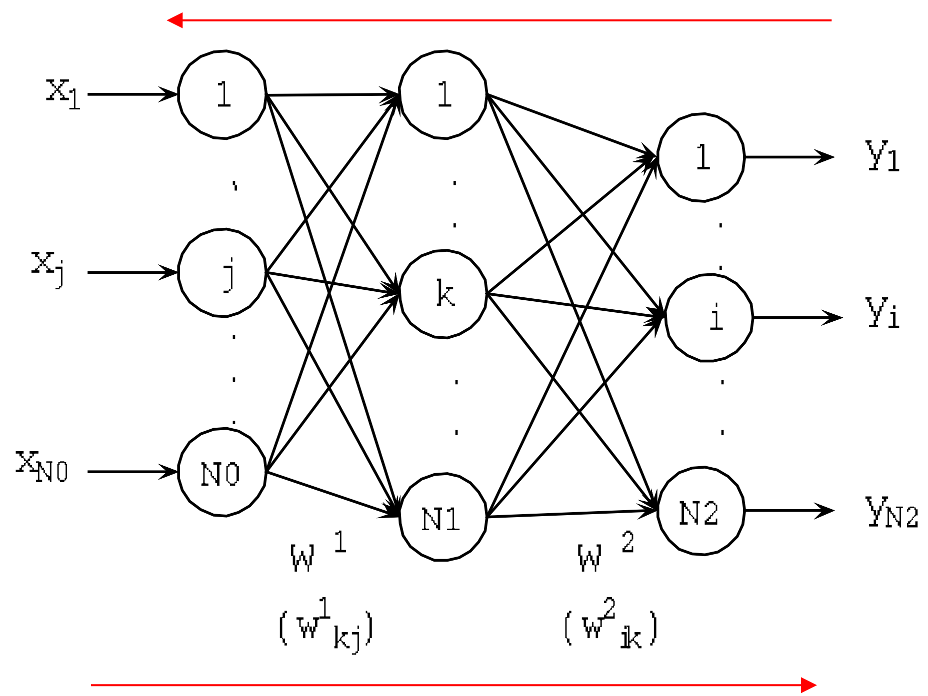
The BackPropagation Algorithm

Main idea:

For each example in the training set:

- compute the output signal
- compute the error corresponding to the output level
- propagate the error back into the network and store the corresponding delta values for each layer
- adjust each weight by using the error signal and input signal for each layer

Computation of the error signal (BACKWARD)



Computation of the output signal (FORWARD)

The BackPropagation Algorithm

General structure

Random initialization of weights

Training epoch

```
REPEAT
   FOR l=1,L  DO
      FORWARD stage
      BACKWARD stage
      weights adjustement
   ENDFOR
   Error (re)computation
UNTIL <stopping condition>
```

Rmk.

- The weights adjustment depends on the learning rate
- The error computation needs the recomputation of the output signal for the new values of the weights
- The stopping condition depends on the value of the error and on the number of epochs
- This is a so-called serial (incremental) variant: the adjustment is applied separately for each example from the training set

The BackPropagation Algorithm

Details (serial variant)

$$w_{kj} = rand(-1,1), w_{ik} = rand(-1,1)$$

$$p = 0$$

REPEAT

FOR $l = 1, L$ DO

/* FORWARD Step */

$$x_k^l = \sum_{j=0}^{N0} w_{kj} x_j^l,\ y_k^l = f_1(x_k^l),\ x_i^l = \sum_{k=0}^{N1} w_{ik} y_k^l,\ y_i^l = f_2(x_i^l)$$

/* BACKWARD Step */

$$\delta_i^l = f_2^{'}(x_i^l)(d_i^l - y_i^l),\ \delta_k^l = f_1^{'}(x_k^l)\sum_{i=1}^{N2} w_{ik}\delta_i^l$$

/* Adjustement Step */

$$w_{kj} = w_{kj} + \eta\delta_k^l x_j^l,\ \ w_{ik} = w_{ik} + \eta\delta_i^l y_k^l$$

ENDFOR

The BackPropagation Algorithm

Details (serial variant)

$$/*\ \text{Error computation}\ */$$

$$E = 0$$

$$\text{FOR } l := 1, L \text{ DO}$$

$$/*\ \text{FORWARD Step}\ */$$

$$x_k^l = \sum_{j=0}^{N0} w_{kj} x_j^l,\ y_k^l = f_1(x_k^l),\ x_i^l = \sum_{k=0}^{N1} w_{ik} y_k^l,\ y_i^l := f_2(x_i^l)$$

$$/*\ \text{Error summation}\ */$$

$$E = E + \sum_{l=1}^{L} (d_i^l - y_i^l)^2$$

$$\text{ENDFOR}$$

$$E = E/(2L)$$

$$p = p + 1$$

$$\text{UNTIL } p > p_{\max} \text{ OR } E < E^*$$

E* denotes the expected training accuracy

p_{max} denotes the maximal number of epochs

The BackPropagation Algorithm

Batch variant

Random initialization of weights

REPEAT

Training epoch

- initialize the variables which will contain the adjustments
- FOR l=1,L DO
  - FORWARD stage
  - BACKWARD stage
  - cumulate the adjustments
- ENDFOR
- Apply the cumulated adjustments
- Error (re)computation

UNTIL <stopping condition>

Rmk.

- The incremental variant can be sensitive to the presentation order of the training examples
- The batch variant is not sensitive to this order and is more robust to the errors in the training examples
- It is the starting algorithm for more elaborated variants, e.g. momentum variant

The BackPropagation Algorithm

Details (batch variant)

$$w_{kj} = rand(-1,1), w_{ik} = rand(-1,1), \ i = 1..N2, k = 0..N1, \ j = 0..N0$$

$$p = 0$$

REPEAT

$$\boxed{\Delta^1_{kj} = 0, \Delta^2_{ik} = 0}$$

FOR $l = 1, L$ DO

/* FORWARD step */

$$x_k^l = \sum_{j=0}^{N0} w_{kj} x_j^l, \ y_k^l = f_1(x_k^l), \ x_i^l = \sum_{k=0}^{N1} w_{ik} y_k^l, \ y_i^l = f_2(x_i^l)$$

/* BACKWARD step */

$$\delta_i^l = f_2^{'}(x_i^l)(d_i^l - y_i^l), \ \delta_k^l = f_1^{'}(x_k^l) \sum_{i=1}^{N2} w_{ik} \delta_i^l$$

/* Adjustment step */

$$\boxed{\Delta^1_{kj} = \Delta^1_{kj} + \eta \delta_k^l x_j^l, \ \ \Delta^2_{ik} = \Delta^2_{ik} + \eta \delta_i^l y_k^l}$$

ENDFOR

$$\boxed{w_{kj} = w_{kj} + \Delta^1_{kj}, \ \ w_{ik} = w_{ik} + \Delta^2_{ik}}$$

The BackPropagation Algorithm

$/ *$ Error computation $* /$

$E = 0$

FOR $l = 1, L$ DO

$/ *$ FORWARD Step $* /$

$$x_k^l = \sum_{j=0}^{N0} w_{kj} x_j^l,\ y_k^l = f_1(x_k^l),\ x_i^l = \sum_{k=0}^{N1} w_{ik} y_k^l,\ y_i^l = f_2(x_i^l)$$

$/ *$ Error summation $* /$

$$E = E + \sum_{l=1}^{L} (d_i^l - y_i^l)^2$$

ENDFOR

$E = E/(2L)$

$p = p + 1$

UNTIL $p > p_{\max}$ OR $E < E*$

Variants

Different variants of BackPropagation can be designed by changing:

- Error function
- Minimization method
- Learning rate choice
- Weights initialization

Variants

Error function:

- MSE (mean squared error function) is appropriate in the case of approximation problems
- For classification problems a better error function is the cross-entropy error:
- Particular case: two classes (one output neuron):
  - d_l is from {0,1} (0 corresponds to class 0 and 1 corresponds to class 1)
  - y_l is from (0,1) and can be interpreted as the probability of class *l*

$$CE(W) = -\sum_{l=1}^{L}(d_l \log y_l + (1-d_l)\log(1-y_l))$$

Rmk: the partial derivatives change, thus the adjustment terms will be different

Variants

Entropy based error:

- Different values of the partial derivatives
- In the case of logistic activation functions the error signal will be:

$$\delta_l = \left(\frac{d_l}{y_l} - \frac{1-d_l}{1-y_l}\right) f_2^{'}(x^{(2)}) = \frac{d_l(1-y_l) - y_l(1-d_l)}{y_l(1-y_l)} \cdot y_l(1-y_l)$$

$$= d_l(1-y_l) - y_l(1-d_l)$$

Variants

Minimization method:

- The gradient method is a simple but not very efficient method

- More sophisticated and faster methods can be used instead:
  - Conjugate gradient methods
  - Newton's method and its variants

- Particularities of these methods:
  - Faster convergence (e.g. the conjugate gradient converges in n steps for a quadratic error function)
  - Needs the computation of the hessian matrix (matrix with second order derivatives) : second order methods

Variants

Example: Newton's method

$E : R^n \to R,\ \ w \in R^n$ is the vector of all weights

By Taylor's expansion in $w(p)$ (estimation corresponding to epoch p)

$$E(w) \cong E(w(p)) + (\nabla E(w(p)))^T (w - w(p)) + \frac{1}{2}(w - w(p))^T H(w(p))(w - w(p))$$

$$H(w(p))_{ij} = \frac{\partial E(w(p))}{\partial w_i \partial w_j}$$

By derivating the Taylor's expansion with respect to w the minimum will be the solution of :

$$H(w(p))w - H(w(p))w(p) + \nabla E(w(p)) = 0$$

Thus the new estimation of w is :

$$w(p+1) = w(p) - H^{-1}(w(p)) \cdot \nabla E(w(p))$$

Variants

Particular case: Levenberg-Marquardt

- This is the Newton method adapted for the case when the objective function is a sum of squares (as MSE is)

$$E(w)=\sum_{l=1}^{L} E_l(w),\; e(w)=(E_1(w),\ldots,E_L(w))^T$$

$$w(p+1)=w(p)-(J^T(w(p))\cdot J(w(p))+\mu_p I)^{-1}J^T(w(p))e(w(p))$$

$$J(w)=\text{ jacobian of } e(w)$$

$$J_{ij}(w)=\frac{\partial E_i(w)}{\partial w_j}$$

Used in order to deal with singular matrices

Advantage:

- Does not need the computation of the hessian

Problems in BackPropagation

- Low convergence rate (the error decreases too slow)
- Oscillations (the error value oscillates instead of continuously decreasing)
- Local minima problem (the learning process is stuck in a local minima of the error function)
- Stagnation (the learning process stagnates even if it is not a local minima)
- Overtraining and limited generalization

Problems in BackPropagation

Problem 1: The error decreases too slow or the error value oscillates instead of continuously decreasing

Causes:

- Inappropriate value of the learning rate (too small values lead to slow convergence while too large values lead to oscillations)

  Solution: adaptive learning rate

- Slow minimization method (the gradient method needs small learning rates in order to converge)

  Solutions:

  - heuristic modification of the standard BP (e.g. momentum)
  - other minimization methods (Newton, conjugate gradient)

Problems in BackPropagation

Adaptive learning rate:

- If the error is increasing then the learning rate should be decreased
- If the error significantly decreases then the learning rate can be increased
- In all other situations the learning rate is kept unchanged

$$E(p) > (1+\gamma)E(p-1) \Rightarrow \eta(p) = a\eta(p-1),\ 0 < a < 1$$
$$E(p) < (1-\gamma)E(p-1) \Rightarrow \eta(p) = b\eta(p-1),\ 1 < b < 2$$
$$(1-\gamma)E(p-1) \le E(p) \le (1+\gamma)E(p-1) \Rightarrow \eta(p) = \eta(p-1)$$

Example: $\gamma=0.05$

Problems in BackPropagation

Momentum variant:

- Increase the convergence speed by introducing some kind of "inertia" in the weights adjustment: the weight changes corresponding to the current epoch includes the adjustments from the previous epoch

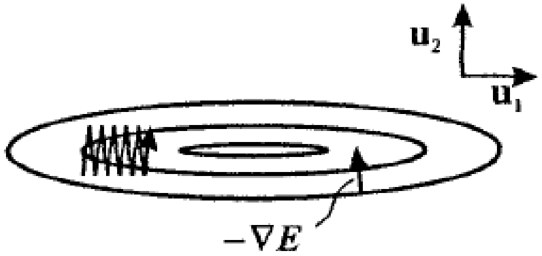
$$\Delta w_{ij}(p+1) = \eta(1-\alpha)\delta_i y_j + \alpha \Delta w_{ij}(p)$$

Momentum coefficient: α in $[0.1, 0.9]$

Problems in BackPropagation

Momentum variant:

- The effect of these enhancements is that flat spots of the error surface are traversed relatively rapidly with a few big steps, while the step size is decreased as the surface gets rougher. This implicit adaptation of the step size increases the learning speed significantly.



Simple gradient descent

Use of inertia term

Problems in BackPropagation

Problem 2: Local minima problem (the learning process is stuck in a local minima of the error function)

Cause: the gradient based methods are local optimization methods

Solutions:

- Restart the training process using other randomly initialized weights
- Introduce random perturbations into the values of weights:

$$w_{ij} := w_{ij} + \xi_{ij}, \quad \xi_{ij} = \text{random variables}$$

- Use a global optimization method

Problems in BackPropagation

Solution:

- Replacing the gradient method with a stochastic optimization method
- This means using a random perturbation instead of an adjustment based on the gradient computation
- Adjustment step:

$$\Delta_{ij} = \text{random values}$$

$$\text{IF } E(W+\Delta) < E(W) \text{ THEN accept the adjustment } (\mathrm{W} := \mathrm{W} + \Delta)$$

Rmk:

- The adjustments are usually based on normally distributed random variables
- If the adjustment does not lead to a decrease of the error then it is not accepted

Problems in BackPropagation

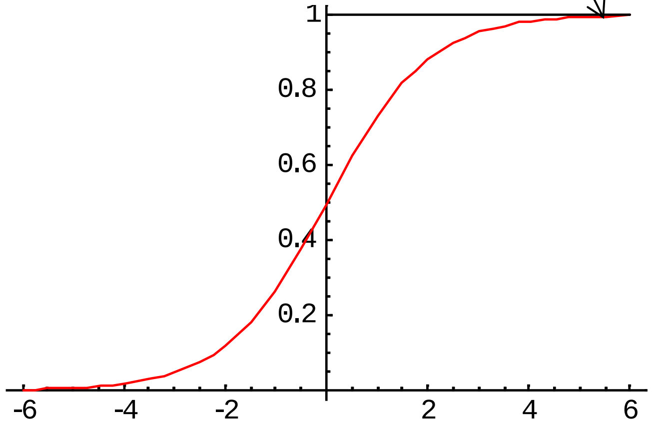
Problem 3: Stagnation (the learning process stagnates even if it is not a local minima)

Cause: the adjustments are too small because the arguments of the sigmoidal functions are too large

Solutions:

- Penalize the large values of the weights (weights-decay)
- Use only the signs of derivatives not

Very small derivates



Problems in BackPropagation

Penalization of large values of the weights: add a regularization term to the error function

$$E_{(r)}(W) = E(W) + \lambda \sum_{i,j} w_{ij}^2$$

The adjustment will be:

$$\Delta_{ij}^{(r)} = \Delta_{ij} - 2\lambda w_{ij}$$

Problems in BackPropagation

Resilient BackPropagation (use only the sign of the derivative not its value)

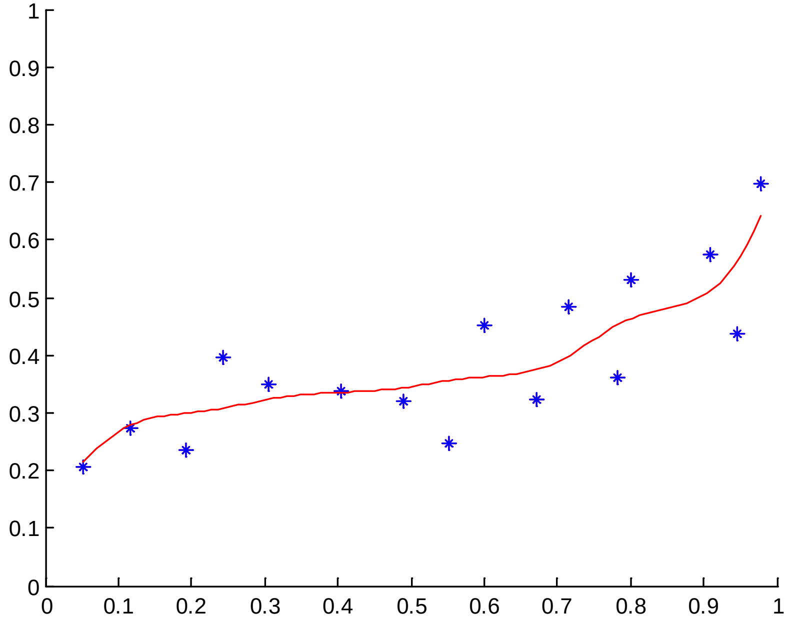
$$\Delta w_{ij}(p) = \begin{cases} -\Delta_{ij}(p) & \text{if } \dfrac{\partial E(W(p-1))}{\partial w_{ij}} > 0 \\ \Delta_{ij}(p) & \text{if } \dfrac{\partial E(W(p-1))}{\partial w_{ij}} < 0 \end{cases}$$

$$\Delta_{ij}(p) = \begin{cases} a\Delta_{ij}(p-1) & \text{if } \dfrac{\partial E(W(p-1))}{\partial w_{ij}} \cdot \dfrac{\partial E(W(p-2))}{\partial w_{ij}} > 0 \\ b\Delta_{ij}(p-1) & \text{if } \dfrac{\partial E(W(p-1))}{\partial w_{ij}} \cdot \dfrac{\partial E(W(p-2))}{\partial w_{ij}} < 0 \end{cases}$$

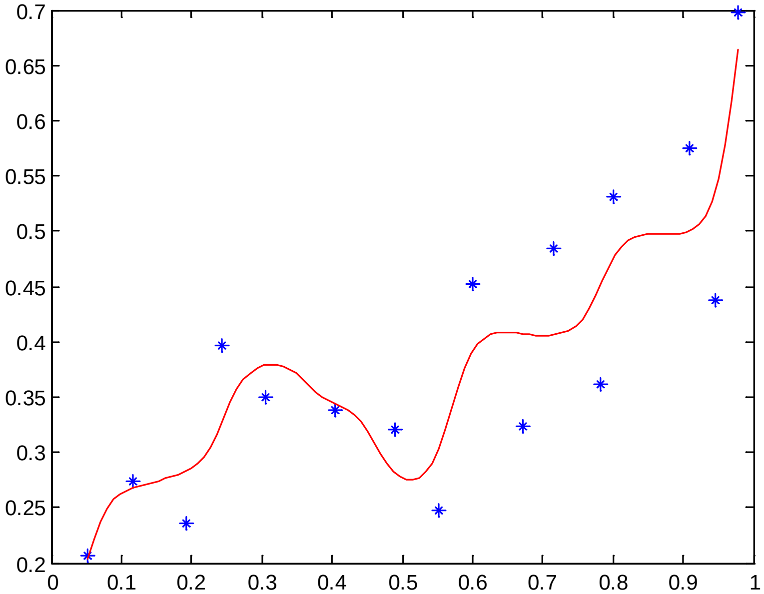
$$0 < b < 1 < a$$

Problems in BackPropagation

Problem 4: Overtraining and limited generalization ability (illustration for an approximation problem)



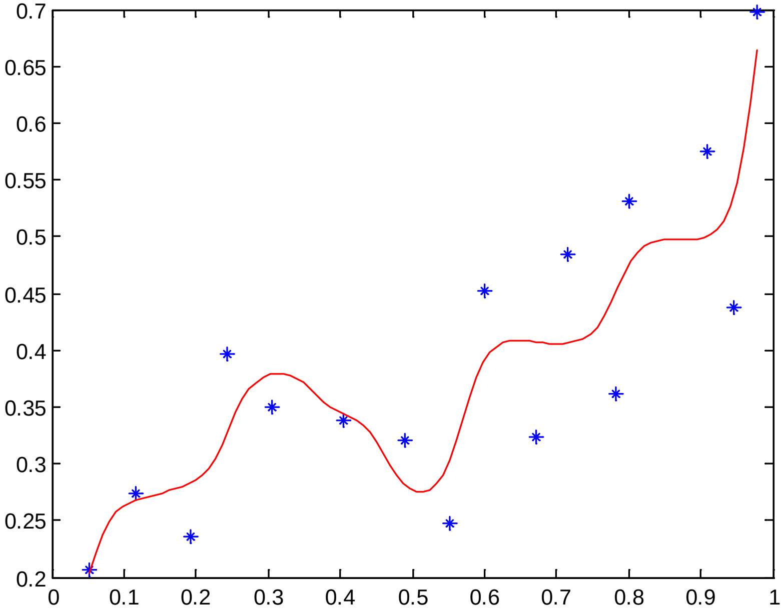
5 hidden units

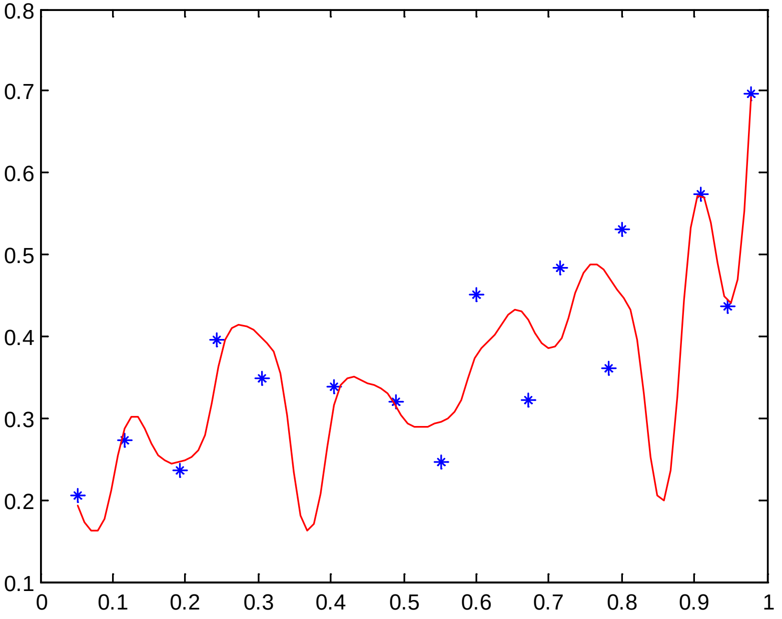


10 hidden units

Problems in BackPropagation

Problem 4: Overtraining and limited generalization ability (illustration for an approximation problem)





10 hidden units

20 hidden units

Problems in BackPropagation

Problem 4: Overtraining and limited generalization ability

Causes:

- Network architecture (e.g. number of hidden units)
  - A large number of hidden units can lead to overtraining (the network extracts not only the useful knowledge but also the noise in data)
- The size of the training set
  - Too few examples are not enough to train the network
- The number of epochs (accuracy on the training set)
  - Too many epochs could lead to overtraining

Solutions:

- Dynamic adaptation of the architecture
- Stopping criterion based on validation error; cross-validation

Problems in BackPropagation

Dynamic adaptation of the architectures:

- Incremental strategy:
  - Start with a small number of hidden neurons
  - If the learning does not progress new neurons are introduced

- Decremental strategy:
  - Start with a large number of hidden neurons
  - If there are neurons with small weights (small contribution to the output signal) they can be eliminated

Problems in BackPropagation

Stopping criterion based on validation error :

- Divide the learning set in m parts: (m-1) are for training and another one for validation
- Repeat the weights adjustment as long as the error on the validation subset is decreasing (the learning is stopped when the error on the validation subset start increasing)

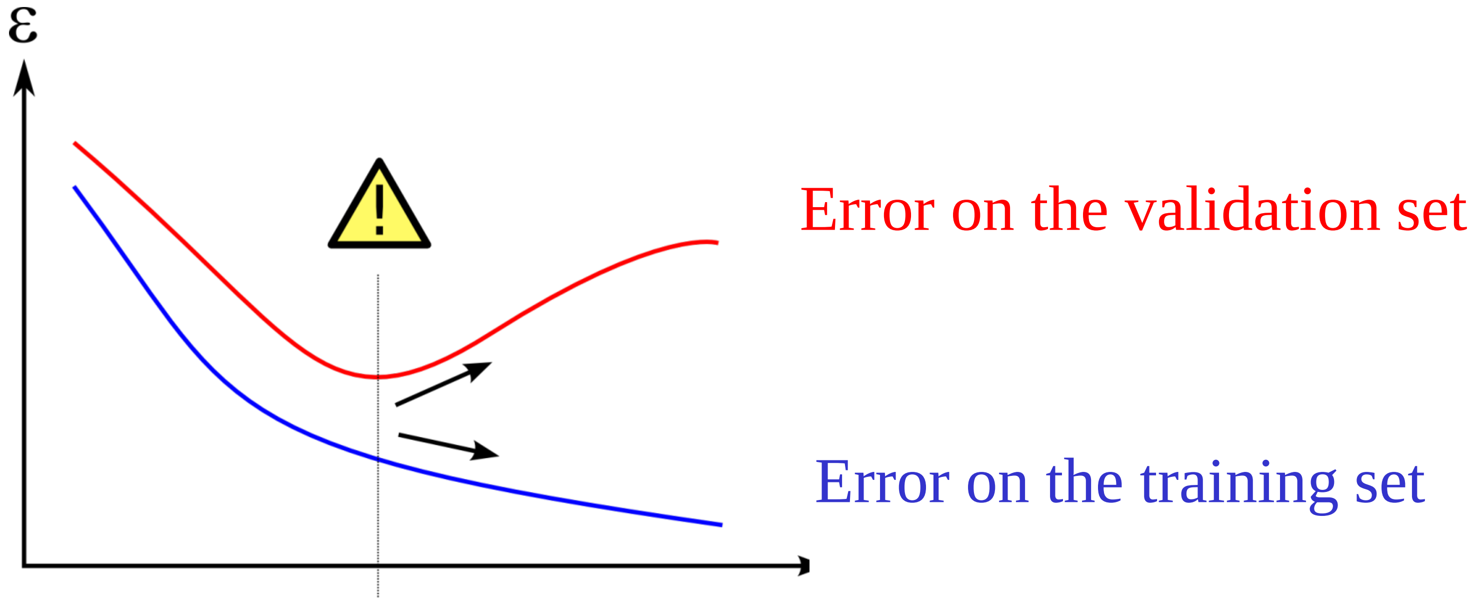
Cross-validation:

- Applies for m times the learning algorithm by successively changing the learning and validation sets

1: S=(S1,S2, ...,Sm)
2: S=(S1,S2, ...,Sm)
....
m: S=(S1,S2, ...,Sm)

Problems in BackPropagation

Stop the learning process when the error on the validation set start to increase (even if the error on the training set is still decreasing) :



Support Vector Machines

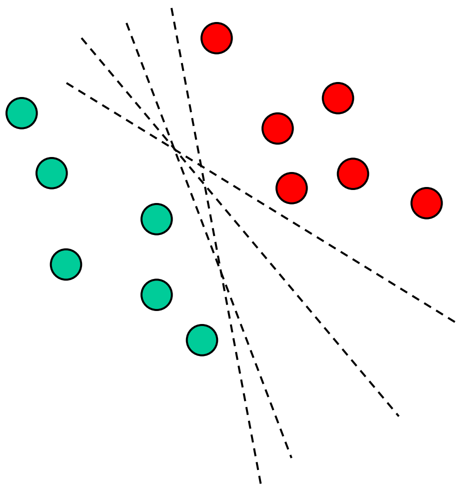
Support Vector Machine (SVM) = a machine learning technique characterized by

- The learning process is based on solving a quadratic optimization problem (avoids the main limits of Backpropagation)
- Ensures a good generalization power
- It relies on the statistical learning theory (main contributors: Vapnik and Chervonenkis)
- Applications: handwritten recognition, speaker identification , object recognition

Biblio: C.Burges – A Tutorial on SVM for Pattern Recognition, Data Mining and Knowledge Discovery, 2, 121–167 (1998)

Support Vector Machines

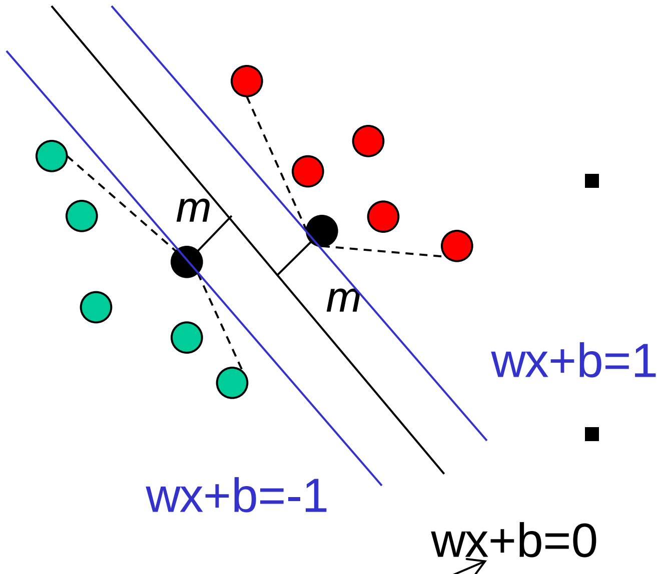
Let us consider a simple linearly separable classification problem



- There exist an infinite number of lines (hyperplanes, in the general case) which ensure the separation in the two classes
- Which separating hyperplane is the best?
- That which leads to the best generalization ability = correct classification for data which do not belong to the training set

Support Vector Machines

Which is the best separating line (hyperplane) ?

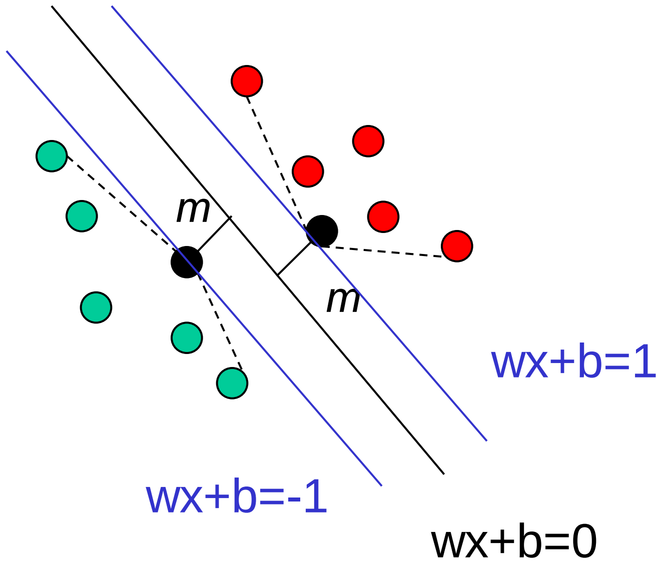


Eq. of the separating hyperplane

- That for which the minimal distance to the convex hulls corresponding to the two classes is maximal
- The lines (hyperplanes) going through the marginal points are called canonical lines (hyperplanes)
- The distance between these lines is 2/||w||, thus maximizing the width of the separating regions means minimizing the norm of w

Support Vector Machines

How can we find the separating hyperplane?



Find w and b which

minimize $||w||^2$

(maximize the separating region)

and satisfy

$(wx_i+b)y_i-1>=0$

For all examples in the training set

$\{(x_1,y_1),(x_2,y_2),\ldots,(x_L,y_L)\}$

$y_i=-1$ for the green class

$y_i=1$ for the red class

(all examples from the training set are classified in the correct class)

Support Vector Machines

The constrained minimization problem can be solved by using the Lagrange multipliers method:

Initial problem:

minimize $||w||^2$ such that $(wx_i+b)y_i-1>=0$ for all i=1..L

By introducing the Lagrange multipliers, the initial optimization problem is transformed in a problem of finding the saddle point of V:

$$V(w,b,\alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{L}\alpha_i(y_i(w\cdot x_i + b) - 1),\ \alpha_i \geq 0$$

$$(w^*, b^*, \alpha^*)\ \text{is saddle point if}:\ V(w^*, b^*, \alpha^*) = \max_{\alpha}\min_{w,b} V(w,b,\alpha)$$

To solve this problem the dual function should be constructed:

$$W(\alpha) = \min_{w,b} V(w,b,\alpha)$$

$$\frac{\partial V(w,b,\alpha)}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{L}\alpha_i y_i x_i \qquad \frac{\partial V(w,b,\alpha)}{\partial b} = 0 \Rightarrow 0 = \sum_{i=1}^{L}\alpha_i y_i$$

Support Vector Machines

Thus we arrived to the problem of maximizing the dual function (with respect to α):

$$W(\alpha) = \sum_{i=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{L} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

such that the following constraints are satisfied:

$$\alpha_i \geq 0, \quad \sum_{i=1}^{L} \alpha_i y_i = 0$$

By solving the above problem (with respect to the multipliers α) the coefficients of the separating hyperplane can be computed as follows:

$$w^* = \sum_{i=1}^{L} \alpha_i y_i x_i, \quad b^* = 1 - w \cdot x_k$$

where k is the index of a non-zero multiplier and x_k is the corresponding training example (belonging to class +1)

Support Vector Machines

Remarks:

- The nonzero multipliers correspond to the examples for which the constraints are active (w x+b=1 or w x+b=-1). These examples are called support vectors and they are the only examples which have an influence on the equation of the separating hyperplane

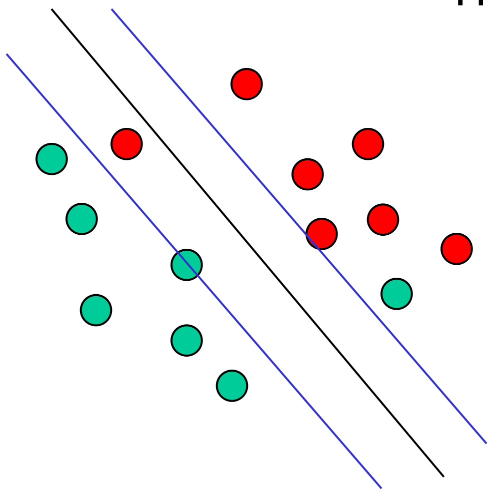
- The other examples from the training set (those corresponding to zero multipliers) can be modified without influencing the separating hyperplane)

- The decision function obtained by solving the quadratic optimizaton problem is:

$$D(z) = \mathrm{sgn}\left(\sum_{i=1}^{L} \alpha_i y_i (x_i \cdot z) + b^*\right)$$

Support Vector Machines

What happens when the data are not very well separated?



The condition corresponding to each class is relaxed:

$$w \cdot x_i + b \geq 1 - \xi_i, \quad \text{if } y_i = 1$$

$$w \cdot x_i + b \leq 1 + \xi_i, \quad \text{if } y_i = -1$$

The function to be minimized becomes:

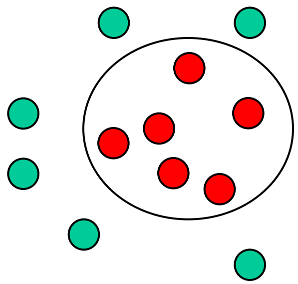
$$V(w, b, \alpha, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{L}\xi_i - \sum_{i=1}^{L}\alpha_i(y_i(w \cdot x_i + b) - 1)$$

Thus the constraints in the dual problem are also changed:

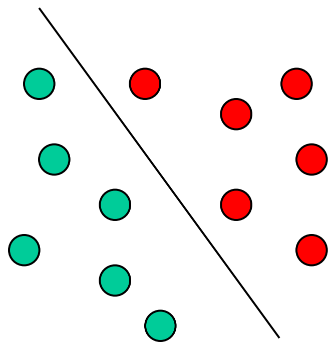
$$\text{instead of } \alpha_i \geq 0 \text{ it is used } 0 \leq \alpha_i \leq C$$

Support Vector Machines

What happens if the problem is nonlineary separable?



$$x_1^2 + x_2^2 - R^2 = 0$$



$$w \cdot z + b = 0,\ \ z_1 = x_1^2, z_2 = x_2^2$$

$$w_1 = w_2 = 1,\ \ b = -R^2$$

$$x_1 \to \theta(x_1) = x_1^2$$

$$x_2 \to \theta(x_2) = x_2^2$$

Support Vector Machines

In the general case a transformation is applied:

$x \rightarrow \theta(x)$ and the scalar product of the transformed vectors becomes:

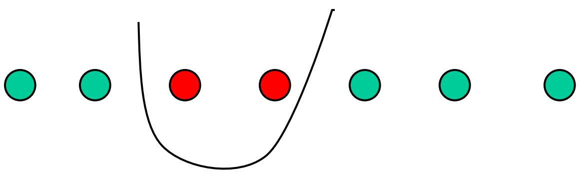
$$\theta(x) \cdot \theta(x') = K(x, x')$$

Since the optimization problem contains only scalar products it is not necessary to know explicitly the transformation θ but it is enough to know the kernel function K

Support Vector Machines

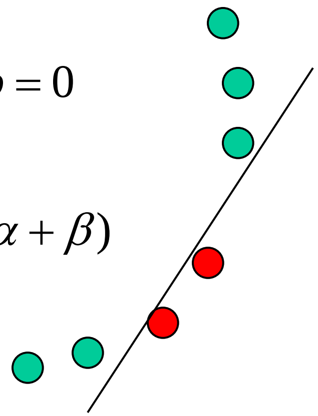
Example 1: Transforming a nonlinearly separable problem in a linearly separable one by going to a higher dimension

$$(x-\alpha)(x-\beta) = x^2 - (\alpha+\beta)x + \alpha\beta$$



1-dimensional nonlinearly separable pb

$$w_1 z_1 + w_2 z_2 + b = 0$$
$$z_1 = x^2, z_2 = x$$
$$w_1 = 1, w_2 = -(\alpha+\beta)$$
$$b = \alpha\beta$$



2-dimensional linearly separable pb

Example 2: Constructing a kernel function when the decision surface corresponds to an arbitrary quadratic function (from dimension 2 the pb.is transferred in dimension 5).

$$\theta(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$$
$$K(x, x') = \theta(x_1, x_2) \cdot \theta(x'_1, x'_2) = (x \cdot x' + 1)^2$$

Support Vector Machines

Examples of kernel functions:

$$K(x, x') = (x \cdot x' + 1)^d$$

$$K(x, x') = \exp\left(-\frac{\left\| x - x' \right\|^2}{2\sigma^2}\right)$$

$$K(x, x') = \tanh(kx \cdot x' + b)$$

The decision function becomes:

$$D(z) = \mathrm{sgn}\left(\sum_{i=1}^{L} \alpha_i y_i K(x_i, z) + b^*\right)$$

Support Vector Machines

Implementations

LibSVM [http://www.csie.ntu.edu.tw/~cjlin/libsvm/]: (+ links to implementations in Java, Matlab, R, C#, Python, Ruby)

SVM-Light [http://www.cs.cornell.edu/People/tj/svm_light/]: implementation in C

Spider [http://www.kyb.tue.mpg.de/bs/people/spider/tutorial.html]: implementation in Matlab

SciLab interface for LibSVM (http://atoms.scilab.org/toolboxes/libsvm

Next lecture

- Data clustering
  - Partitional algorithms
    - Kmeans
    - Fuzzy Cmeans
  - Hierarchical algorithms
    - Agglomerative