

Masinõppimine

An Introduction to Machine Learning

- According to Herbert Simon, learning is, "Any change in a System that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population." [G. F. Luger and W. A. Stubblefield, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, The Benjamin/Cummings Publishing Company, Inc. 1989.]

Machine Learning

- Decision trees
- Instance-based
- Neural Networks
- Kernel Machines

- Probabilistic Models
- Bayesian Learning
- Learning Theory
- Reinforcement Learning
- Genetic Algorithms

Ways humans learn things

- ...talking, walking, running...
- Learning by mimicking, reading or being told facts
- Tutoring
- Being informed when one is correct
- Experience
- Feedback from the environment
- Analogy
- Comparing certain features of existing knowledge to new problems
- Self-reflection
- Thinking things in ones own mind, deduction, discovery

A few achievements

- Programs that can:
 - Recognize spoken words
 - Predict recovery rates of pneumonia patients
 - Detect fraudulent use of credit cards
 - Drive autonomous vehicles
 - Play games like backgammon – approaching the human champion!

What is the Learning problem?

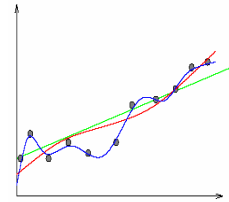
- Learning = improving with experience at some task
- Improve over task T
 - With respect to performance measure P
 - Based on experience E
 - Example: Learning to play checkers
 - T: play checkers
 - P: % of games won in world tournament
 - E: opportunity to play against self

?

- What exactly should be learned?
- How shall it be represented?
- What specific algorithm to learn it?

Why Learning is Difficult?

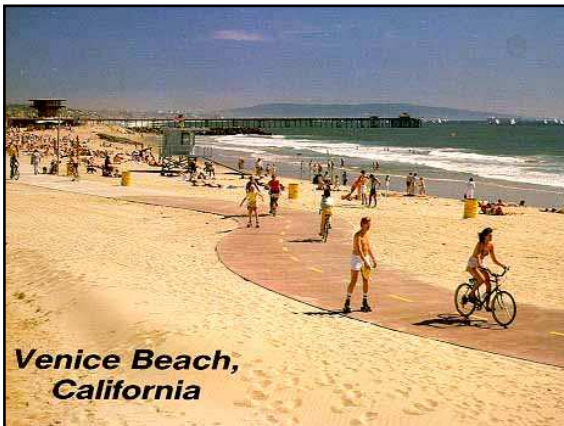
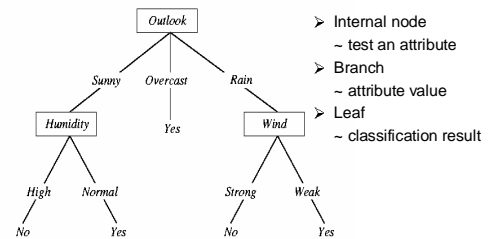
- Given a finite amount of training data, you have to derive a relation for an infinite domain
- In fact, there is an infinite number of such relations
- How should we draw the relation?



Otsustuspuud

- Decision trees

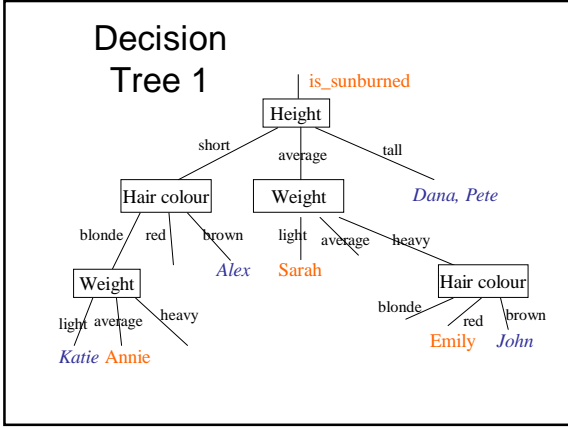
Decision Tree Representation for 'Play Tennis?'



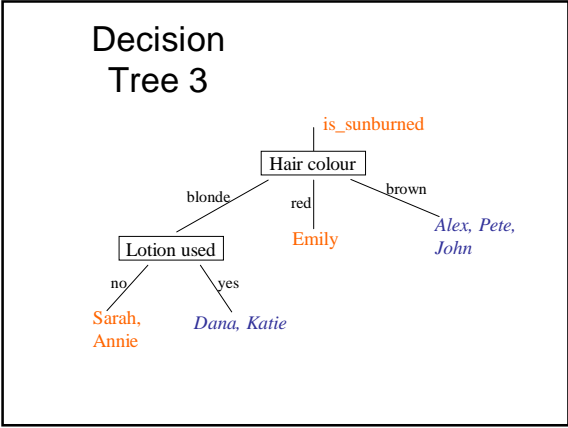
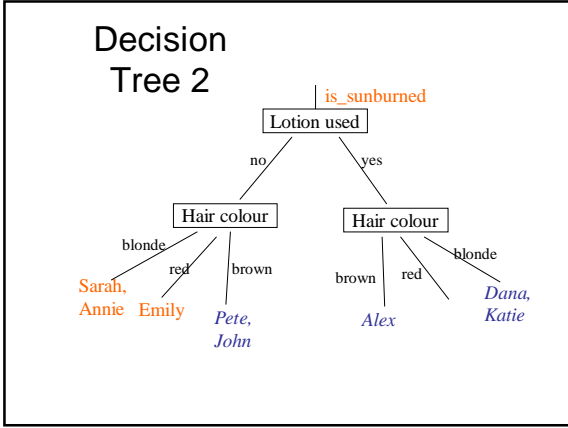
**Venice Beach,
California**

Sunburn Data Collected

Name	Hair	Height	Weight	Lotion	Result
Sarah	Blonde	Average	Light	No	Sunburned
Dana	Blonde	Tall	Average	Yes	None
Alex	Brown	Short	Average	Yes	None
Annie	Blonde	Short	Average	No	Sunburned
Emily	Red	Average	Heavy	No	Sunburned
Pete	Brown	Tall	Heavy	No	None
John	Brown	Average	Heavy	No	None
Kate	Blonde	Short	Light	Yes	None



- ### Sunburn sufferers are ...
- If height="average" then
 - if weight="light" then
 - return(true) ;; Sarah
 - elseif weight="heavy" then
 - if hair_colour="red" then
 - return(true) ;; Emily
 - elseif height="short" then
 - if hair_colour="blonde" then
 - if weight="average" then
 - return(true) ;; Annie
 - else return(false) ;; everyone else



- ### Summing up
- Irrelevant attributes do not classify the data well
 - Using irrelevant attributes thus causes larger decision trees
 - a computer could look for simpler decision trees
 - Q: How?

- ### A: How WE did it?
- Q: Which is the best attribute for splitting up the data?
 - A: The one which is *most informative* for the classification we want to get.
 - Q: What does it mean 'more informative'?
 - A: The attribute which best *reduces the uncertainty* or the disorder
 - Q: How can we measure something like that?
 - A: Simple – just listen ;-)

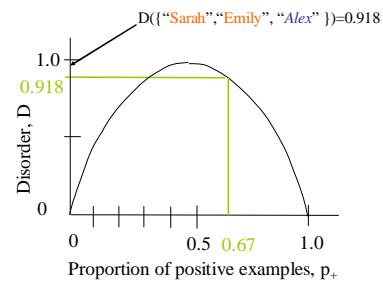
- We need a quantity to measure the *disorder* in a partly classified set of examples
 $S = \{s_1, s_2, s_3, \dots, s_n\}$
 where $s_1 = \text{"Sarah"}$, $s_2 = \text{"Dana"}$, ...
- Then we need a quantity to measure the amount of *reduction of the disorder* level in the instance of knowing the value of a particular attribute

What properties should the *Disorder* (D) have?

- Suppose that $D(S) = 0$ means that all the examples in S have the same class
- Suppose that $D(S) = 1$ means that half the examples in S are of one class and half are the opposite class

Examples

- $D(\{\text{"Dana"}, \text{"Pete"}\}) = 0$
- $D(\{\text{"Sarah"}, \text{"Annie"}, \text{"Emily"}\}) = 0$
- $D(\{\text{"Sarah"}, \text{"Emily"}, \text{"Alex"}, \text{"John"}\}) = 1$
- $D(\{\text{"Sarah"}, \text{"Emily"}, \text{"Alex"}\}) = ?$



Definition of Disorder

The **Entropy** measures the disorder of a set S containing a total of n examples of which n_+ are positive and n_- are negative and it is given by

$$D(n_+, n_-) = -\frac{n_+}{n} \log_2 \frac{n_+}{n} - \frac{n_-}{n} \log_2 \frac{n_-}{n} = \text{Entropy}(S)$$

$$- \sum_i p_i \log p_i$$

Check it!

$$D(0,1) = ? \quad D(1,0) = ? \quad D(0.5,0.5) = ?$$

Back to the beach (or the disorder of sunbathers!)

$$D(\{\text{"Sarah"}, \text{"Dana"}, \text{"Alex"}, \text{"Annie"}, \text{"Emily"}, \text{"Pete"}, \text{"John"}, \text{"Katie"}\})$$

$$= D(3,5) = -\frac{3}{8} \log_2 \frac{3}{8} - \frac{5}{8} \log_2 \frac{5}{8}$$

$$= 0.954$$

Some more useful properties of the Entropy

$$D(n, m) = D(m, n) \quad D(kn, km) = D(km, kn)$$

$$D(0, m) = 0 \quad D(m, m) = 1$$

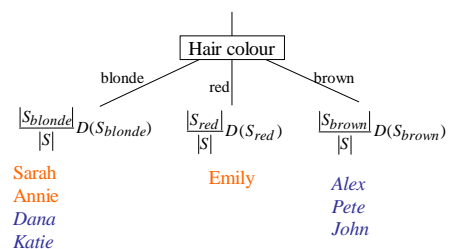
- We can measure the disorder ☺
- What's left:
 - We want to measure how much by knowing the value of a particular attribute this disorder would reduce.

- The **Information Gain** measures the expected reduction in entropy due to sorting on an attribute A

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

the **average disorder** is just the weighted sum of the disorder in the branches associated with the values of A.

Back to the beach: calculate the Average Disorder associated with Hair Colour



...Calculating the Disorder of the "blondes"

- $D(S_{blonde}) = D(\{\text{"Sarah", "Annie", "Dana", "Katie"}\}) = D(2,2) = 1$

$$\frac{|S_{blonde}|}{|S|} D(S_{blonde}) = \frac{|S_{blonde}|}{|S|} = \frac{4}{8} = 0.5$$

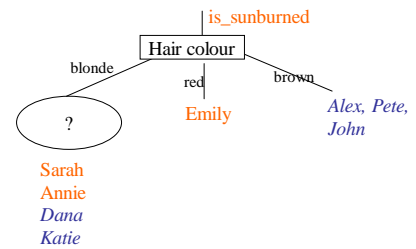
...Calculating the disorder of the others

- We don't have to bother with the "reds" and "browns" because $S_{red} = \{\text{"Emily"}\}$ and $S_{brown} = \{\text{"Alex", "Pete", "John"}\}$.
- Within each set all the examples have the same class

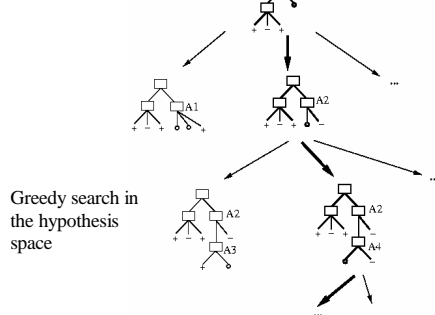
Which decision variable minimises the disorder?

Test	Disorder
Hair	0.5
height	0.69
weight	0.94
lotion	0.61

So what is the best decision tree?



ID3 algorithm

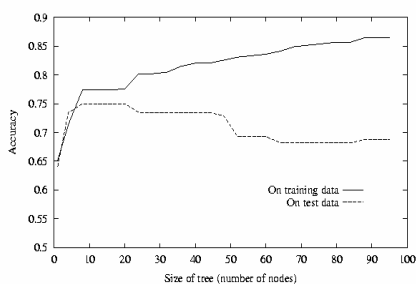


Is this all? So much simple?

Of course not...

- where do we stop growing the tree?
- what if there are noisy (mislabelled) data as well in data set?

Overfitting in Decision Tree Learning



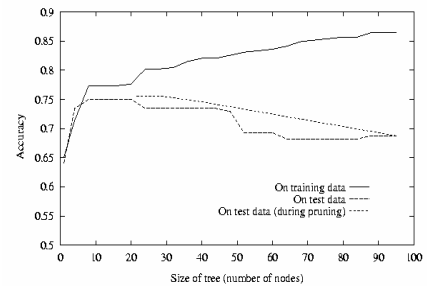
Overfitting

- Consider the error of hypothesis h over:
 - Training data: $\text{error_train}(h)$
 - The whole data set (new data as well): $\text{error_D}(h)$
- If there is another hypothesis h' such that $\text{error_train}(h) < \text{error_train}(h')$ and $\text{error_D}(h) > \text{error_D}(h')$ then we say that hypothesis h **overfits** the training data.

How can we avoid overfitting?

- Split the data into training set & validation set
- Train on the training set and stop growing the tree when further data split deteriorates performance on validation set
- Or: grow the full tree first and then post-prune
- What if data is limited?

...looks a bit better now



Summary

- Decision Tree Representation
- Entropy, Information Gain
- ID3 Learning algorithm
- Overfitting

When to consider Decision Trees?

- If data is described by a finite number of attributes, each having a (finite) number of possible values
- The target function is discrete valued
- Possibly noisy data
- Possibly missing values
- Disjunctive hypothesis
- E.g.:
 - Medical diagnosis
 - Equipment diagnosis
 - Credit risk analysis
 - etc

Lähima naabri meetodid

- k-nearest neighbours kNN

- K-Nearest Neighbour
- Locally weighted regression
- Case-based reasoning
- Lazy and eager learning

Instance-based learning

- One way of solving tasks of approximation discrete or real valued target functions
- Have training examples: $(x_n, f(x_n))$, $n=1..N$.
- Key idea:
 - just store the training examples
 - when a test example is given then find the closest matches

- Nearest neighbour:
 - Given a query instance x_q ,
 - first locate the nearest training example x_n
 - then $f(x_q) := f(x_n)$
- K-Nearest neighbour:
 - Given a query instance x_q ,
 - first locate the k nearest training examples
 - then take vote among its k nearest nbrs (if discrete values target function), or
 - take the mean of the f values of the k nearest nbrs (if real valued target fct.)

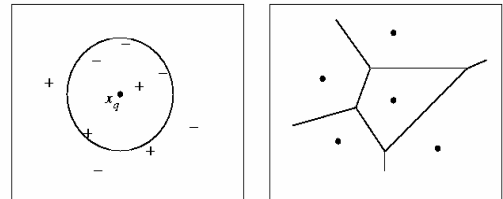
$$f(x_q) := \frac{\sum_{i=1}^k f(x_i)}{k}$$

The distance between examples

- Assume that we have n attributes for the learning problem $a_r(x) \in \mathfrak{R}$
- The distance between two examples x_i, x_j is often defined as the Euclidean distance

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^n [a_r(x_i) - a_r(x_j)]^2}$$

Voronoi Diagram



Characteristics of IBL

- No explicit description of the target concept is learned with training data
- It is capable of producing a *different, local* approximation to the target concept with each test instance
- an instance-based learner is *lazy* and does all the work when the test example is presented

What if the target function is real valued?

- The k-nearest neighbour algorithm would just calculate the mean of the k nearest neighbours

Variant of kNN: Distance-Weighted kNN

- Might want to weight nearer neighbors more heavily

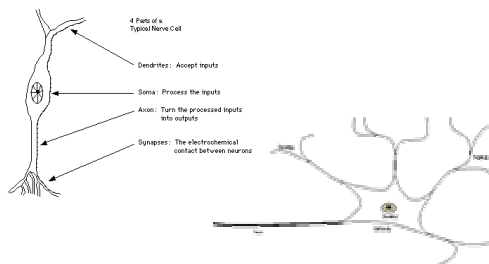
$$f(x_q) := \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i} \text{ where } w_i = \frac{1}{d(x_q, x_i)^2}$$

- Then it makes sense to use *all training examples* instead of just k (Stepard's method)

Närvivörgud

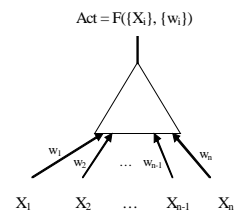
- Neural Networks (NN, ANN)

Natural Neuron



Artificial Neuron

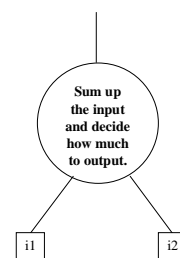
Captures the essence of the natural neuron



Natural vs Artificial

- | | |
|---|--|
| <ul style="list-style-type: none"> • Dendrites • Synapses • Soma's chemical reaction • Axon | <ul style="list-style-type: none"> • Input values X_i from the environment or other neurons • Real-valued weights w_i associated with each input • Function F({X_i}, {w_i}) computing activation as a function of input values and weights • Activation value that may serve as input to other neurons |
|---|--|

A perceptron neuron



The perceptron learning law

Each new weight is
the old weight adjusted by
the error times the input value.

Example: out 0 desired 1 error (0 - 1) = -1
in 1 * weight (e.g. 0.1)

New weight is 0.1 + error = 1.1

A threshold value can be thought of as an input that is always on.
It will be adjustable if it has a weight associated with it.

Summing up

The perceptron basically sums up all its input

$$w_1 * x_1 + w_2 * x_2 + \dots + w_n x_n + (C)$$

Is that sum larger than a threshold value?

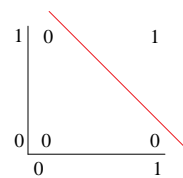
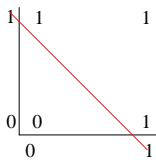
Drawing a line

The decision for on or off is bounded by a linear function:

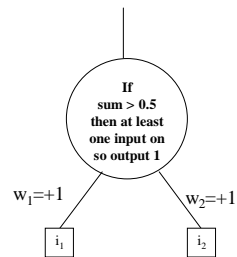
For example, with two inputs,

$$w_1 * i_1 + w_2 * i_2 < \text{threshold value (0 in the previous example)}$$

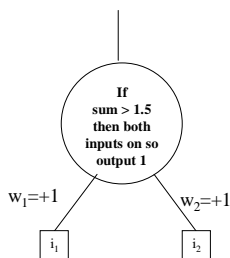
i.e. $ax + by - C = 0$ (this is the equation of a line: $y = (-a/b)x + C$)



A perceptron or

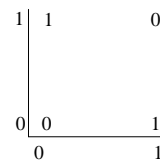


A perceptron and



A problem

Some functions are not 'linearly separable'.



The Dark Ages of NN

Since xor (which is a simple function) could not be separated by a line the perceptron is very limited in what kind of functions it can learn.

This discredited the inventor of the perceptron, Frank Rosenblatt, and made it almost impossible to get funding for neural network research. Funding instead went to Symbolic AI.

Neural networks / perceptrons seemed a dead end.

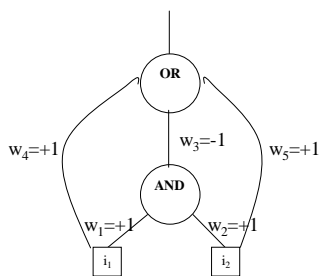
1967 ... 1982 quiet years, though many worked 'underground'.

Combining

xor can be composed of 'simpler' logical functions.

$A \text{ xor } B = (A \text{ or } B) \text{ and not } (A \text{ and } B)$
The last term simply removes the troublesome value.

A combined perceptron



What was the problem?

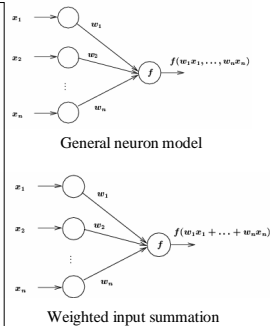
There was no learning law for layered perceptrons.

Nobody bothered to find one!

The main problem was that the perceptron solves a simple linear equation. Any straight combination (adding or multiplying) of linear equations is still a linear equation.

ANN Neuron Models

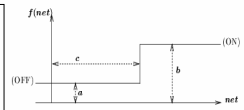
- Each node has one or more inputs from other nodes, and one output to other nodes
- Input/output values can be
 - Binary {0, 1}
 - Bipolar {-1, 1}
 - Continuous
- All inputs to one node come in at the same time and remain activated until the output is produced
- Weights associated with links
- $f(\text{net})$ is the node function
- $\text{net} = \sum_{i=1}^n w_i x_i$ is most popular



Node Function

- Identity function : $f(\text{net}) = \text{net}$.
- Constant function : $f(\text{net}) = c$.
- Step (threshold) function

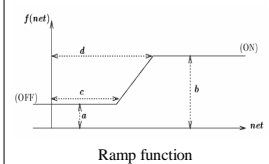
$$f(\text{net}) = \begin{cases} a & \text{if } \text{net} \leq c \\ b & \text{if } \text{net} > c \end{cases}$$



where c is called the threshold

- Ramp function

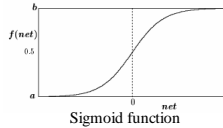
$$f(\text{net}) = \begin{cases} a & \text{if } \text{net} \leq c \\ b & \text{if } \text{net} \geq d \\ a + \frac{(\text{net}-c)(b-a)}{(d-c)} & \text{otherwise} \end{cases}$$



Node Function

- **Sigmoid function**

- S-shaped
- Continuous and everywhere differentiable
- Rotationally symmetric about some point ($net = c$)
- Asymptotically approach



$$\lim_{net \rightarrow -\infty} f(net) = a \quad \lim_{net \rightarrow \infty} f(net) = b$$

- Examples:

$$f(net) = z + \frac{1}{1 + \exp(-x \cdot net + y)}$$

$$f(net) = \tanh(x \cdot net - y) + z,$$

When $y = 0$ and $z = 0$:

$$a = 0, b = 1, c = 0.$$

When $y = 0$ and $z = -0.5$

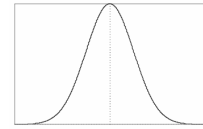
$$a = -0.5, b = 0.5, c = 0.$$

Larger x gives steeper curve

Node Function

- **Gaussian function**

- Bell-shaped (radial basis)
- Continuous
- $f(net)$ asymptotically approaches 0 (or some constant) when $|net|$ is large
- Single maximum (when $net = \mu$)
- Example:

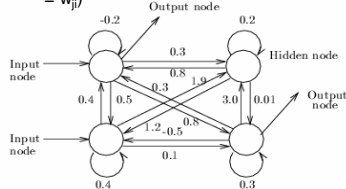


$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{net - \mu}{\sigma}\right)^2\right]$$

Network Architecture

- **(Asymmetric) Fully Connected Networks**

- Every node is connected to every other node
- Connection may be excitatory (positive), inhibitory (negative), or irrelevant (≈ 0).
- Most general
- Symmetric fully connected nets: weights are symmetric ($w_{ij} = w_{ji}$)

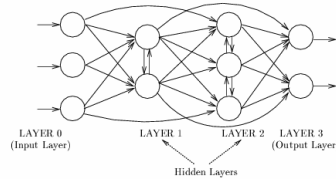


Input nodes: receive input from the environment
Output nodes: send signals to the environment
Hidden nodes: no direct interaction to the environment

Network Architecture

- **Layered Networks**

- Nodes are partitioned into subsets, called layers.
- No connections that lead from nodes in layer j to those in layer k if $j > k$.



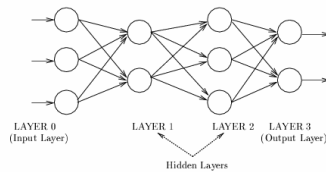
• Inputs from the environment are applied to nodes in layer 0 (**input layer**).

• Nodes in input layer are place holders with no computation occurring (i.e., their node functions are identity function)

Network Architecture

- **Feedforward Networks**

- A connection is allowed from a node in layer i only to nodes in layer $i + 1$.
- Most widely used architecture.



Conceptually, nodes at higher levels successively abstract features from preceding layers

Network Architecture

- **Acyclic Networks**

- Connections do not form directed cycles.
- Multi-layered feedforward nets are acyclic

- **Recurrent Networks**

- Nets with directed cycles.
- Much harder to analyze than acyclic nets.

- **Modular nets**

- Consists of several modules, each of which is itself a neural net for a particular sub-problem
- Sparse connections between modules

NN Characteristics

- (Artificial) neural networks are sets of (highly) interconnected artificial neurons (i.e., simple computational units)
- Characteristics
 - Massive parallelism
 - Distributed knowledge representation (i.e., implicit in patterns of interactions)
 - Opaque (i.e., black box)
 - Graceful degradation (e.g., grandmother cell)
 - Less susceptible to brittleness
 - Noise tolerant

Network Topology

- Pattern of interconnections between neurons: primary source of inductive bias
- Characteristics
 - Interconnectivity (fully connected, mesh, etc.)
 - Number of layers
 - Number of neurons per layer
 - Fixed vs. dynamic

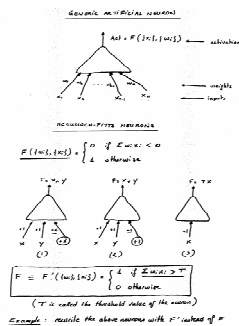
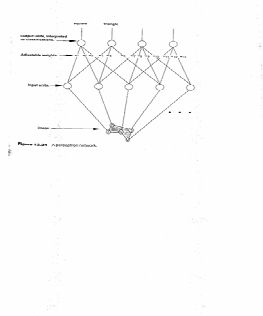
NN Learning

- Learning is effected by one or a combination of several mechanisms:
 - Weight updates
 - Changes in activation functions
 - Changes in overall topology

Perceptrons (1958)

- The simplest class of neural networks
- Single-layer, i.e., only one set of weight connections between inputs and outputs
- Binary inputs
- Boolean activation levels

$$F(x_i, w_i) = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i w_i > t \\ 0 & \text{otherwise} \end{cases}$$



Learning for Perceptrons

- Algorithm devised by Rosenblatt in 1958
- Given an example (i.e., labeled input pattern):
 - Compute output
 - Check output against target
 - Adapt weights when different

Learn-Perceptron

- $d \leftarrow$ some constant (the *learning rate*)
- Initialize weights (typically random)
- For each new training example with target output T
 - For each node (parallel for loop)
 - Compute activation F
 - If ($F=T$) then Done
 - Else if ($F=0$ and $T=1$) then Increment weights on active lines by d
 - Else Decrement weights on active lines by d

Groundhog Day

- The main character gets to relive the same day over and over until he *gets it right*.
- This is the core of nn/backpropagation learning.



Intuition

- Learn-Perceptron slowly (depending on the value of d) moves weights closer to the target output
- Several iterations over the training set may be required


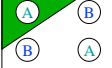










Example

- Consider a 3-input, 1-output perceptron
- Let $d=1$ and $t=0.9$
- Initialize the weights to 0
- Build the perceptron (i.e., learn the weights) for the following training set:
 - 0 0 1 \rightarrow 0
 - 1 1 1 \rightarrow 1
 - 1 0 1 \rightarrow 1
 - 0 1 1 \rightarrow 0

More Functions

- Repeat the previous exercise with the following training set:
 - 0 0 \rightarrow 0
 - 1 0 \rightarrow 1
 - 1 1 \rightarrow 0
 - 0 1 \rightarrow 1
- What do you observe? Why?

Non linearly separable problems

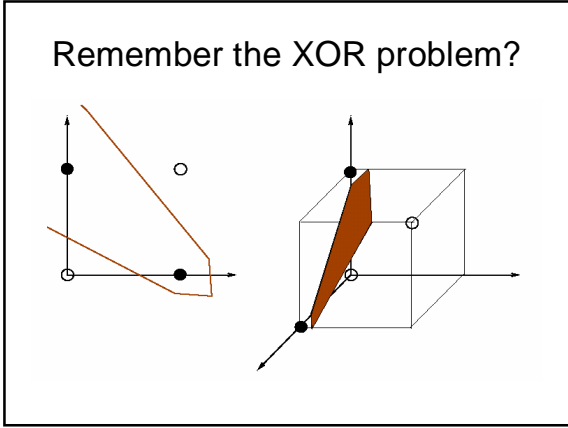
Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Neural Networks – An Introduction Dr. Andrew Hunter

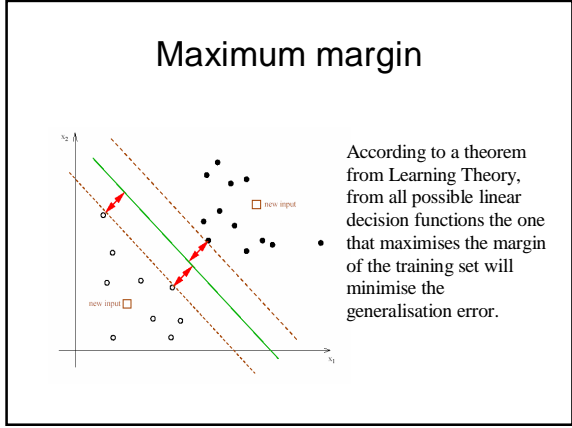
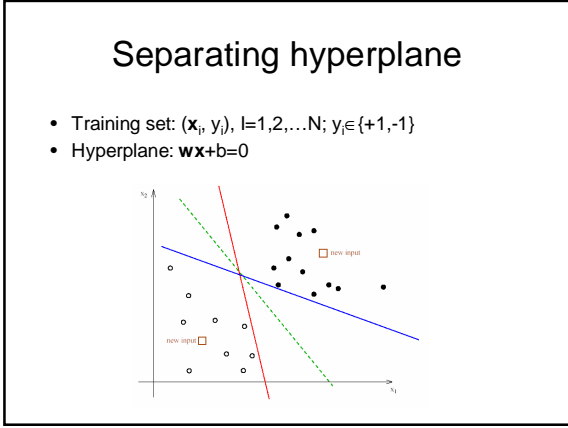
Support Vector Machines Kernel Machines

Tugivector masinad

Borrowed from: Ata Kaban
The University of Birmingham



- ### Support Vector Machines (SVM)
- Supervised learning problems
 - Classification
 - Regression
 - Two key ideas
 - Learn separating hyperplane with maximum margin
 - Expand input into high-dimensional space



Maximum margin

Note1: decision functions (w, b) and (cw, cb) are the same

Note2: but margins as measured by the function outputs are not the same

Def: geometric margin: the margin given by the canonical decision function, i.e. when $c=1/\|w\|$

Strategy:

- 1) we need to maximise the geometric margin! (cf result from learning theory)
- 2) subject to the constraint that training examples are classified correctly

Maximum margin

According to Note1, we can demand the function output for the nearest points to be +1 and -1 on the two sides of the decision function. This removes the scaling freedom.

Denoting a nearest positive example x_+ and a nearest negative example x_- , this is

Computing the geometric margin (that has to be maximised):

$$\frac{1}{2} \left(\frac{w}{\|w\|} x_+ + b - \frac{w}{\|w\|} x_- - b \right) = \frac{1}{2\|w\|} (wx_+ + b - wx_- - b) = \frac{1}{\|w\|}$$

And here are the constraints:

$$\begin{cases} wx_i + b \geq +1 & \text{for } y_i = +1 \\ wx_i + b \leq -1 & \text{for } y_i = -1 \end{cases} \iff y_i(wx_i + b) - 1 \geq 0 \text{ for all } i$$

Maximum margin – summing up

Given a linearly separable training set (x_i, y_i) , $i=1, 2, \dots, N$; $y_i \in \{+1, -1\}$

Minimise $\|w\|^2$

Subject to

$$y_i(wx_i + b) - 1 \geq 0, \quad i = 1, \dots, N$$

→ This is a quadratic programming problem with linear inequality constraints. ☺

Support vectors

The training points that are nearest to the separating function are called support vectors.

What is the output of our decision function for these points?

Solving

- Construct & minimise the Lagrangian

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(wx_i + b) - 1]$$
 wrt. constraint $\alpha_i \geq 0, i = 1, \dots, N$
- Take derivatives wrt. w and b , equate them to 0

$$\frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^N \alpha_i y_i x_i = 0 \quad \rightarrow \text{parameters are expressed as a linear combination of training points}$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = \sum_{i=1}^N \alpha_i y_i = 0$$
 KKT cond: $\alpha_i [y_i(wx_i + b) - 1] = 0 \quad \rightarrow \text{only SVs will have non-zero } \alpha_i$

The Lagrange multipliers α_i are called 'dual variables'
Each training point has an associated dual variable.

Class 2

Class 1

$w^T x + b = 1$

$w^T x + b = 0$

$w^T x + b = -1$

Solving

- So,
- Plug this back into the Lagrangian to obtain the dual formulation (homework)
- The resulting dual that is solved for α by using a QP solver:

$$\text{maximise: } W(\alpha) = -\frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j + \sum_{i=1}^N \alpha_i$$

- The ~~biases~~ b does not appear in the dual so N it is determined separately from the initial constraints (homework)

Data enters only in the form of dot products!

Classifying new data points

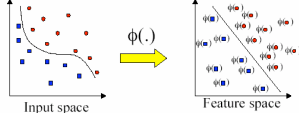
- Once the parameters (α^*, b^*) are found by solving the required quadratic optimisation on the training set of points, the SVM is ready to be used for classifying new points.
- Given new point x , its class membership is $\text{sign}[f(x, \alpha^*, b^*)]$, where

$$f(x, \alpha^*, b^*) = \mathbf{w}^T \mathbf{x} + b^* = \sum_{i=1}^N \alpha_i^* y_i \mathbf{x}_i \mathbf{x} + b^* = \sum_{i \in SV} \alpha_i^* y_i \mathbf{x}_i \mathbf{x} + b^*$$

Data enters only in the form of dot products!

Non-linear SVMs

- Transform $\mathbf{x} \rightarrow \phi(\mathbf{x})$
- The linear algorithm depends only on $\mathbf{x}\mathbf{x}_i$, hence transformed algorithm depends only on $\phi(\mathbf{x})\phi(\mathbf{x}_i)$
- Use kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ such that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)\phi(\mathbf{x}_j)$



Examples of kernels

- Example1: 2D input space, 3D feature space

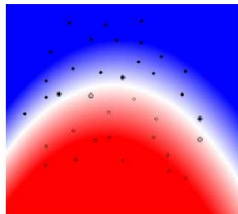
$$\phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \quad K(\mathbf{x}, \mathbf{x}_j) = (\mathbf{x}, \mathbf{x}_j)^2$$

- Example2:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2\}$$

in this case the dimension of ϕ is infinite

- Note: Not every function is a proper kernel. There is a theorem called Mercer Theorem that proper kernels must satisfy
- To test a new input \mathbf{x} when working with kernels



A non-linear SVM decision boundary obtained with the applet at <http://svm.research.bell-labs.com> using the kernel $k(x,y) = (x,y)^3$

Making kernels

New kernels can be made from valid kernels by allowed operations e.g. addition, multiplication and rescaling of kernels gives a proper kernel

$$K(\mathbf{x}_1, \mathbf{x}_2) = K_1(\mathbf{x}_1, \mathbf{x}_2) + K_2(\mathbf{x}_1, \mathbf{x}_2)$$

$$K(\mathbf{x}_1, \mathbf{x}_2) = \lambda K_1(\mathbf{x}_1, \mathbf{x}_2)$$

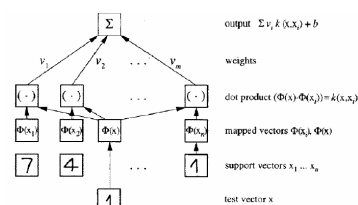
Also, given a real-valued function $f(x)$ over inputs \mathbf{x} , then the following is a valid kernel

$$K(\mathbf{x}_1, \mathbf{x}_2) = f(x_1)f(x_2)$$

Steps for classification

- Prepare the data matrix
- Select the kernel function to use
- Execute the training algorithm using a QP solver to obtain the α_i values
- Unseen data can be classified using the α_i values and the support vectors

Applications



- Handwritten digits recognition
 - Of interest to the US Postal Services
 - 4% error was obtained
 - about 4% of the training data were SVs only
- Text categorisation
- Face detection
- DNA analysis
- ...

Topics not covered

- Dealing with non-separable training sets
- SVM for regression
- Multi-class classification
- ...

Conclusions

- SVMs learn linear decision boundaries (cf perceptrons)
 - Pick hyperplane that maximises margin
 - Optimal hyperplane turns out to be a linear combination of support vectors
- Transform nonlinear problems to higher dimensional space using kernel functions
- Good at avoiding overfitting in high dimensional spaces (cf regularisation)

Resources

- SW & practical guide to SVM for beginners
<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- Kernel machines website:
<http://www.kernel-machines.org/>
- Burges, C.J. C: A tutorial on Support Vector Machines for Pattern Recognition. Data Mining and Knowledge Discovery, Vol.2, nr.2, pp.121—167, 1998. Available from <http://svm.research.bell-labs.com/SVMdoc.html>
- Cristianini & Shawe-Taylor: SVM book (in the School library)

Ajalugu

ML History: 1950s-1960s

- An exploratory period when many general techniques were born. Early research was inspired by biology and psychology.
- Rosenblatt (1957) developed the "perceptron", which was modeled after neurons. It was the precursor to later work in neural networks.
- No knowledge -- Self-organizational, Self-adaptive
- In 1969, Minsky & Papert wrote a seminal paper proving that the perceptron was inherently limited, which discouraged research in this area for almost 20 years!
- Field more closely tied to work in pattern recognition.

ML History : 1970s

- This period was characterized by the development of more practical algorithms, often using symbolic techniques.
- Winston's (1970) important work on learning in blocks-world domain.
- "knowledge acquisition bottleneck"
- META-DENDRAL (Buchanan et al., 1978) learned mass-spectrometry prediction rules.
- The idea of macro-operators was developed (Fikes et al., 1972).
- Discovery
 - Mathematical discovery with AM (Lenat, 1977).
 - Scientific discovery with BACON (Langley, 1978)
- AQ and soybean diagnosis results (Michalski, 1980)

ML History: 1980s

- Explosion of different techniques & increased emphasis on evaluation
- The notion of "version spaces" was developed by Mitchell (1982), and the concept of "inductive bias" (with Utgoff).
- Quinlan (1983) created the ID3 decision tree algorithm.
- Valiant (1984) defined the idea of "probably approximately correct" (PAC) learning, still the dominant theoretical paradigm.
- Connectionist research became active again. The backpropagation algorithm (Rumelhart, 1986) overcame many of the limitations of the perceptron.
- The COBWEB clustering system was developed by Fisher (1987).
- Explanation-based learning, speedup learning, and case-based learning became very popular.

ML History: 1990s & beyond

- What's hot now? Reinforcement learning; Bayesian learning; automatic bias selection; inductive logic programming (mostly in Europe); applications to robotics; adaptive software agents; voting, bagging, boosting, and stacking.
- Data mining has emerged as an important application for many business activities.
- The machine learning (ML) community has become increasingly concerned with evaluation. ML systems must be formally evaluated to ensure that the results are meaningful. Community data sets have been made available for comparing algorithms.
- Future: Complex applications; statistical techniques; combining techniques, . . .

Taxonomy of ML

These differ in their assumptions about the representation, performance methods, and algorithms used in a learning system

- Neural Networks
- Instance/Case-based Learning
- Genetic Algorithms
- Rule Induction
- Analytical Learning
-

Representation, Organization, and Interpretation of Knowledge

- Representation
 - Logical Conjunctions of Features: All or none match
 - Production
 - Frame
- Organization
 - Decision Lists
 - Inference Networks
 - Concept Hierarchies
- Statistical information added to any of the above
- Combinations of the above

Other Taxonomy Methods

- Application Fields
 - Expert System
 - Robot
 - Natural Language Processing
- Methods of Simulation
 - Function (Symbol Learning)
 - Structure (Connectionist)

Input given to an ML System

- Most ML algorithms use a **training set** of examples as the basis for learning.
- Each example is encoded using the instance representation chosen for the problem. The representation is crucial!
- **Supervised Learning:** the learning program is given training examples with the correct classification for each example.
- **Unsupervised Learning:** the learning program is given training examples without classifications.

Instance-based learning

- Instance-based reasoning systems keep track of previously seen instances and apply them directly to new ones.
- In general, the system simply stores each "case" that it experiences in a "case base" which represents its memory of previous episodes.
- To reason about a new instance, the system consults its case base and finds the most similar case that it's seen before. The old case is then applied to the new situation.
- IBL is similar to reasoning by analogy. Many people believe that much of human learning is case-based in nature.

Transformation-based learning

- Transformation-based error-driven learning has been used in corpus-based natural language processing systems.
- Initially, a simple default method is used to classify instances. The classifications are then compared against the correct answers, and rules are learned to fix the mistakes.
- Transformation-based learning methods used general templates that represent the general types of rules that can be learned. The templates are instantiated to create specific rules.

Rule Learning

- Most rule learning systems are covering algorithms that generate classification rules to account for (cover) the data.
- Each rule consists of predicates to be tested and a classification to be assigned if the predicates are satisfied.
- Each rule is constructed by repeatedly adding predicates to it until the rule can reliably classify a subset of the data. The examples "covered" by the rule are then removed from subsequent rule learning.

Genetic Algorithms

- Genetic algorithms are inspired by biological mechanisms related to genetics, such as "recombination" (e.g., cross-over), random mutations, and natural selection.
- Genetic algorithms begin with a population of individuals and repeatedly apply genetic operators to selected individuals. After each iteration, the individuals are evaluated with respect to a "fitness" function and only the most "fit" individuals survive into the next generation.
- Genetic algorithms are often used to solve optimization problems.

Bayesian Learning

- Bayesian learning techniques apply Bayesian statistics to data.
- Statistical techniques can be extremely powerful, but they rely on very large data sets. Fortunately, the wealth of information now available on-line has made statistical learning feasible (with some assumptions).
- Bayesian classifiers can be structured to represent independent features, or hierarchical relationships in a Bayesian network.
- Statistical learning has demonstrated great success on many tasks, including speech recognition.

Connectionist Algorithms

- Connectionist models (also called "neural networks") are inspired by the interconnectivity of the brain.
- Connectionist networks typically consist of many nodes that are highly interconnected. When a node is activated, it sends signals to other nodes so that they are activated in turn.
- Using layers of nodes allows connectionist models to learn complex functions.
- Connectionist models are often amenable to parallelism, but it can sometimes be difficult to understand their behavior.

Reinforcement learning

- Reinforcement learning is often used for control problems. The model produces a series of responses, and then a reinforcement signal is provided by an external source as feedback.
- However, the learning system only gets delayed reinforcement so it is not told whether each individual response is correct or incorrect. It is only told whether the final result is positive or negative!
- One type of reinforcement learning tries to learn from temporally successive predictions (it is sometimes called temporal-difference learning).

Explanation-based Learning

- Explanation-based learning (EBL) systems try to explain why each training instance belongs to the target concept. The resulting "proof" is generalized and saved.
- If a new instance can be explained in the same manner as a previous instance, then it is also assumed to be a member of the target concept.
- One of the strengths of EBL is that the resulting "explanations" are typically easy to understand.
- One of the weaknesses of EBL is that they rely on a domain theory to generate the explanations.

Macro learning

- Macro learning is most often applied in problem-solving domains where a system searches for a sequence of rules to solve a problem.
- Given the same problem multiple times, most systems will rederive the entire sequence of steps each time.
- The idea behind macro learning is to group together sequences of rules that are often applied together. The new "macro" can then be applied as a single operator that fires several rules at the same time.
- In principle, the new system can't solve problems that it couldn't solve before, but it becomes more efficient.

Statistical Machine Learning

- Mixture of Gaussians
- Factor Analysis
- Hidden Markov Model
- Maximum Entropy Principle
- Support Vector Machine
- Expectation Maximization
- Boosting
-