Lecture 1
What is soft computing
Techniques used in soft computing

What is Soft Computing ? (adapted from L.A. Zadeh)

- Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect, the role model for soft computing is the human mind.

What is Hard Computing?

- Hard computing, i.e., conventional computing, requires a precisely stated analytical model and often a lot of computation time.
- Many analytical models are valid for ideal cases.
- Real world problems exist in a non-ideal environment.

What is Soft Computing? (Continued)

- The principal constituents, i.e., tools, techniques, of Soft Computing (SC) are
  - Fuzzy Logic (FL), Neural Networks (NN), Support Vector Machines (SVM), Evolutionary Computation (EC), and
  - Machine Learning (ML) and Probabilistic Reasoning (PR)

Premises of Soft Computing

- The real world problems are pervasively imprecise and uncertain
- Precision and certainty carry a cost

Guiding Principles of Soft Computing

- The guiding principle of soft computing is:
  - Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost.

Hard Computing

- Premises and guiding principles of Hard Computing are
  - Precision, Certainty, and rigor.
- Many contemporary problems do not lend themselves to precise solutions such as
  - Recognition problems (handwriting, speech, objects, images
  - Mobile robot coordination, forecasting, combinatorial problems etc.

Implications of Soft Computing

- Soft computing employs NN, SVM, FL etc, in a complementary rather than a competitive way.
- One example of a particularly effective combination is what has come to be known as "neurofuzzy systems.”
- Such systems are becoming increasingly visible as consumer products ranging from air conditioners and washing machines to photocopiers, camcorders and many industrial applications.

Unique Property of Soft computing

- Learning from experimental data
- Soft computing techniques derive their power of generalization from approximating or interpolating to produce outputs from previously unseen inputs by using outputs from previous learned inputs
- Generalization is usually done in a high-dimensional space.

Current Applications using Soft Computing

- Application of soft computing to handwriting recognition
- Application of soft computing to automotive systems and manufacturing
- Application of soft computing to image processing and data compression
- Application of soft computing to architecture
- Application of soft computing to decision-support systems
- Application of soft computing to power systems
- Neurofuzzy systems
- Fuzzy logic control

Future of Soft Computing (adapted from L.A. Zadeh)

- Soft computing is likely to play an especially important role in science and engineering, but eventually its influence may extend much farther.
- Soft computing represents a significant paradigm shift in the aims of computing
  - a shift which reflects the fact that the human mind, unlike present day computers, possesses a remarkable ability to store and process information which is pervasively imprecise, uncertain and lacking in categoricity.

Overview of Techniques in Soft Computing

- Neural networks
- Support Vector Machines
- Fuzzy Logic
- Genetic Algorithms in Evolutionary Computation

Neural Networks Overview

- Neural Network Definition
- Some Examples of Neural Network Algorithms and Architectures
- Successful Applications

Definitions of Neural Networks

- According to the *DARPA Neural Network Study* (1988, AFCEA International Press, p. 60):
- ... a neural network is a system composed of many simple processing elements operating in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.

Definitions of Neural Networks

- According to Haykin (1994), p. 2:
- A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
  - Knowledge is acquired by the network through a learning process.
  - Interneuron connection strengths known as synaptic weights are used to store the knowledge

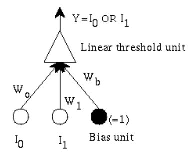
Definitions of Neural Networks

- According to Nigrin (1993), p. 11:
- A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information. Furthermore each element operates asynchronously; thus there is no overall system clock.

Definitions of Neural Networks

- According to Zurada (1992), p. xv:
- Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store, and utilize experiential knowledge.

A Simple Neural Network



The network has 2 inputs, and one output. All are binary.
The output is
1 if W0 *I0 + W1 * I1 + Wb > 0
0 if W0 *I0 + W1 * I1 + Wb <= 0
We want it to learn simple OR: output a 1 if either I0 or I1 is 1.

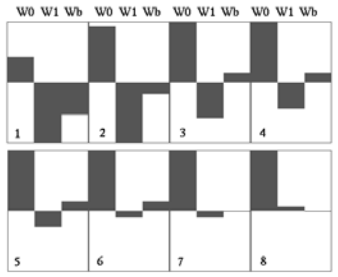
A Simple Neural Network

- **The simple Perceptron:**
- The network adapts as follows: *change the weight by an amount proportional to the difference between the desired output and the actual output.* As an equation:
- Δ Wi = η * (D-Y).Ii
- where η is the learning rate, D is the desired output, and Y is the actual output.
- This is called the *Perceptron Learning Rule*, and goes back to the early 1960's.

A Simple Neural Network

We expose the net to the patterns:

I_0	I_1	**Desired output**
0	0	0
0	1	1
1	0	1
1	1	1



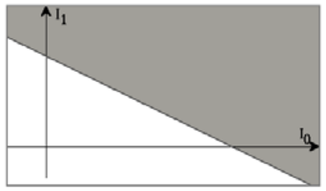
A Simple Neural Network

- We *train* the network on these examples. Weights after each epoch (exposure to complete set of patterns)
- At this point (8) the network has finished learning. Since (D-Y)=0 for all patterns, the weights cease adapting. Single perceptrons are limited in what they can learn:
- If we have two inputs, the decision surface is a line and its equation is I1 = (W0/W1).I0 + (Wb/W1)
- In general, they implement a simple hyperplane decision surface
- This restricts the possible mappings available.

A Simple Perceptron Network

The decision surface is a line in this case.

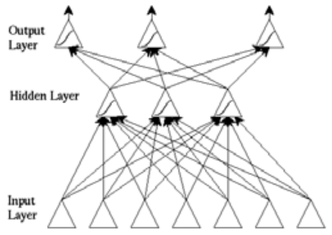
This restricts the possible mappings available.



Developments from the simple perceptron

- Multi-layer perceptrons (MLPs) and Radial Basis Function Networks (RBF) are both well-known developments of the Perceptron Learning Rule.
- Both can learn arbitrary mappings or classifications. Further, the inputs (and outputs) can have real values

Multilayer Perceptron (MLP)

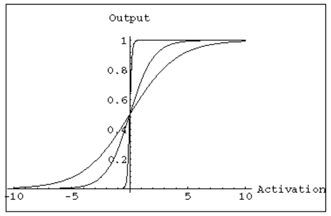


- The network topology is constrained to be *feedforward*: i.e. loop-free - generally connections are allowed from the input layer to the first (and possibly only) hidden layer; from the first hidden layer to the second,..., and from the last hidden layer to the output layer.

Multilayer Perceptron

- The hidden layer learns to *recode* (or to *provide a representation* for) the inputs. More than one hidden layer can be used.
- The architecture is more powerful than single-layer networks: it can be shown that any mapping can be learned, given two hidden layers (of units).

Multilayer Perceptron



- The units are a little more complex than those in the original perceptron: their input/output graph is as shown in the left.
- Output Y as a function:
  Y = 1 / (1+ exp(-k.(Σ Win * Xin))
- The graph shows the output for *k=0.5, 1, and 10,* as the activation varies from -10 to 10.

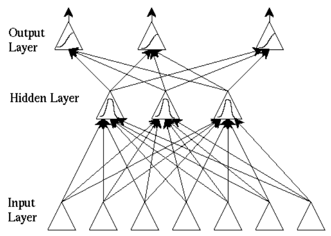
Training MLP Networks

- The weight change rule is a development of the perceptron learning rule. Weights are changed by an amount proportional to *the error at that unit* times *the output of the unit feeding into the weight*.

Training the MLP

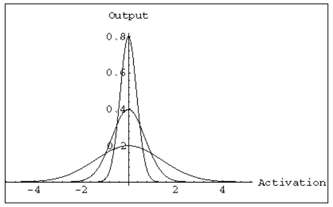
- Running the network consists of
- Forward pass:
  - the outputs are calculated and the error at the output units calculated.
- Backward pass:
  - The output unit error is used to alter weights on the output units. Then the error at the hidden nodes is calculated (by *back-propagating* the error at the output units through the weights), and the weights on the hidden nodes altered using these values.
- For each data pair to be learned a forward pass and backwards pass is performed. This is repeated over and over again until the error is at a low enough level (or we give up).

Radial Basis function Networks



- Radial basis function networks are also feedforward, but have only *one* hidden layer.
- Like MLP, RBF nets can learn arbitrary mappings: the primary difference is in the hidden layer.
- RBF hidden layer units have a *receptive field* which has a *centre:* that is, a particular input value at which they have a maximal output. Their output tails off as the input moves away from this point.

Training RBF Networks



- Generally, the hidden unit function is a Gaussian:
- Gaussians with three different standard deviations as shown in the left figure.

Training RBF Networks.

- RBF networks are trained by
- deciding on how many hidden units there should be
- deciding on their centers and the sharpness (standard deviation) of their Gaussians

Training up the output layer of RBF Networks

- Generally, the centers and SDs are decided on first by examining the vectors in the training data. The output layer weights are then trained using the Delta rule. Back Propagation Network, i.e., MLP, is the most widely applied neural network technique. RBFs are gaining in popularity.

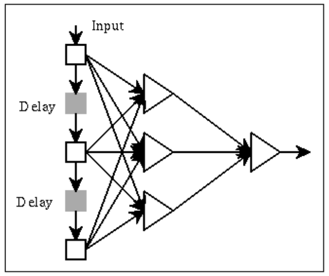
Nets can be

- trained on classification data (each output represents one class), and then used directly as classifiers of new data.
- trained on (x,f(x)) points of an unknown function f, and then used to interpolate.

RBF Networks

- RBFs have the advantage that one can add extra units with centers near parts of the input which are difficult to classify
- Both MLPs and RBFs can also be used for processing time-varying data: one can consider a *window* on the data:
- Networks of this form (finite-impulse response) have been used in many applications.
- There are also networks whose architectures are specialised for processing time-series

Architectures for Processing Time-series



- Simple Perceptrons, MLP, and RBF networks need a teacher to tell the network what the desired output should be. These are supervised networks.

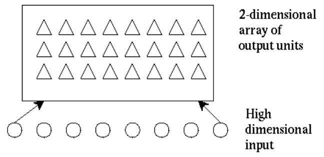
Unsupervised networks

- In an unsupervised net, the network adapts purely in response to its inputs. Such networks can learn to pick out structure in their input.

Applications for unsupervised nets

- clustering data:
  - exactly one of a small number of output units comes on in response to an input.
- reducing the dimensionality of data:
  - data with high dimension (a large number of input units) is compressed into a lower dimension (small number of output units).
- Although learning in these nets can be slow, running the trained net is very fast - even on a computer simulation of a neural net.

Kohonen clustering Algorithm



- takes a high-dimensional input, and clusters it, but retaining some topological ordering of the output.
- After training, an input will cause *some* the output units in some area to become active.
- Such clustering (and dimensionality reduction) is very useful as a preprocessing stage, whether for further neural network data processing, or for more traditional techniques.

Where are Neural Networks applicable?

- in signature analysis:
  - as a mechanism for comparing signatures made (e.g. in a bank) with those stored. This is one of the first large-scale applications of neural networks in the USA, and is also one of the first to use a neural network chip.
- in process control:
  - there are clearly applications to be made here: most processes cannot be determined as computable algorithms.
- in monitoring:
  - networks have been used to monitor
    - the state of aircraft engines. By monitoring vibration levels and sound, early warning of engine problems can be given.
    - British Rail have also been testing a similar application monitoring diesel engines.

Neural Network Applications

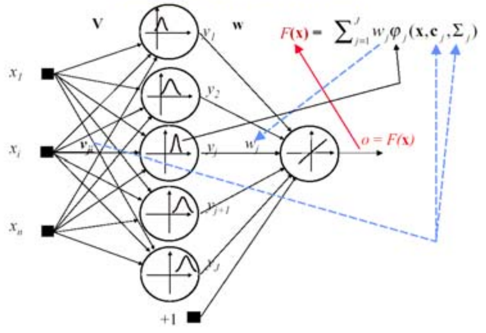
- Pen PC's
  - PC's where one can write on a tablet, and the writing will be recognised and translated into (ASCII) text.
- Speech and Vision recognition systems
  - Not new, but Neural Networks are becoming increasingly part of such systems. They are used as a system component, in conjunction with traditional computers.

Support Vector Machines and Neural Networks

The learning machine that uses data to find the APPROXIMATING FUNCTION (in regression problems) or the SEPARATION BOUNDARY (in classification, pattern recognition problems), is the same in high-dimensional situations.
It is either the so-called **SVM** or the **NN**.

SVMs and NNs

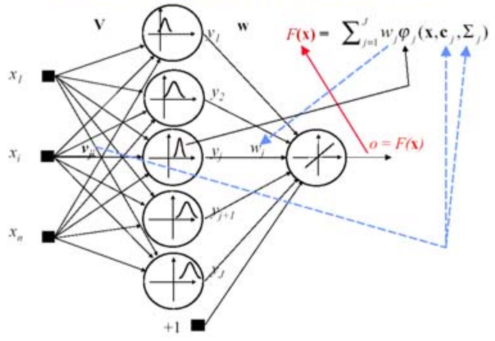
This is a Neural Network,



V
w
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
x_1
x_i
x_n
v_{ji}
y_1
y_2
y_j
y_{j+1}
y_J
w_j
$o = F(\mathbf{x})$
+1

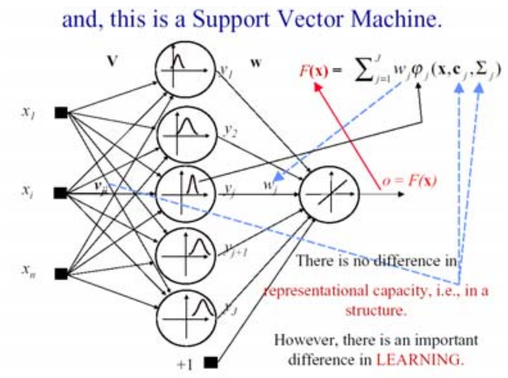
SVMs and NNs

and, this is a Support Vector Machine.



V
w
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
x_1
x_i
x_n
v_{ji}
y_1
y_2
y_j
y_{j+1}
y_J
w_j
$o = F(\mathbf{x})$
+1

SVMs and NNs


and, this is a Support Vector Machine.

V
w
x_1
x_i
x_n
y_1
y_2
y_j
y_{j+1}
y_J
v_{ji}
w_j
+1
$F(\mathbf{x}) = \sum_{j=1}^{J} w_j \varphi_j(\mathbf{x}, \mathbf{c}_j, \Sigma_j)$
$o = F(\mathbf{x})$

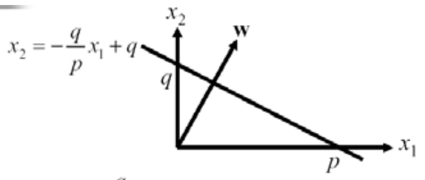
There is no difference in representational capacity, i.e., in a structure.

However, there is an important difference in LEARNING.

Introduction of SVM

- SVM is a classifier derived from statistical learning theory by Vapnik and Chervonenkis (1992)
- One of the newest feedforward multi-layer techniques
- SVM can be used to determine the number of hidden neurons of both radial-basis function networks and 2-layer perceptions
- Suitable for pattern classification or nonlinear regression problems

Background Knowledge



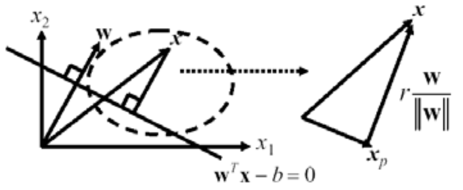
$$x_2 = -\frac{q}{p}x_1 + q \Rightarrow qx_1 + px_2 - pq = 0$$

$$\Rightarrow \begin{bmatrix} q & p \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - pq = 0$$

$$\Rightarrow \mathbf{w}^T\mathbf{x} + b = 0,$$

where $\mathbf{w} = \begin{bmatrix} q & p \end{bmatrix}^T$, $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$, and $b = -pq$.

Background Knowledge



x_2
$\mathbf{w}$
x
x_1
$\mathbf{w}^T\mathbf{x} - b = 0$
x
$r\frac{\mathbf{w}}{\|\mathbf{w}\|}$
x_p

$$x = x_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

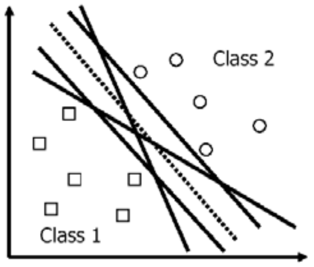
Let the discriminant function $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.

We have $g(\mathbf{x}_p) = 0$ and

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = \mathbf{w}^T(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + b = r\|\mathbf{w}\| \text{or } r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}.$$

Optimal Hyperplane

- Many decision boundaries can separate these two classes
- Which one should we choose?



Optimal Hyperplane (Cont'd)

- Given training samples $\{(\boldsymbol{x}_i, y_i)\}$, $i = 1,\ldots, N$, where $\boldsymbol{x}_i$ is the input pattern and y_i is the corresponding target output.
- An optimal hyperplane is defined as when that is likely to produce the most discriminating power between the 2 classes ($y_i = +1$ or $y_i = -1$)
- An optimal hyperplane can easily be computed in truly linearly separable classes

Optimal Hyperplane (Cont'd)

- Any separating hyperplane is described using the equation:

  $\mathbf{w}^T\mathbf{x} + b = 0$

  $\mathbf{w}^T\mathbf{x} + b \geq 0$ for one of the classes

  $\mathbf{w}^T\mathbf{x} + b < 0$ for the other class
- For an optimal hyperplane:

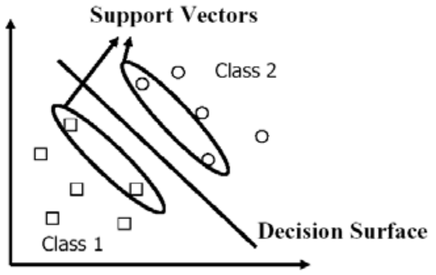
  $\mathbf{w}_o{}^T\mathbf{x} + b_o = 0$

  $\mathbf{w}_o{}^T\mathbf{x} + b_o \geq 1$ for $y_i = +1$

  $\mathbf{w}_o{}^T\mathbf{x} + b_o < -1$ for $y_i = -1$

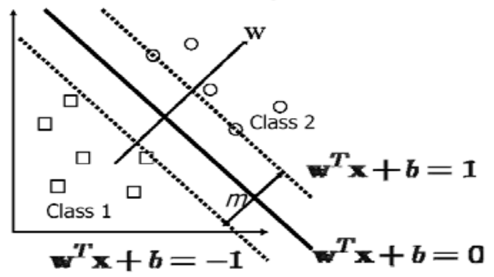
Support Vectors

The support vectors are those data points that lie closest to the decision surface and are therefore the most difficult to classify.



Optimal Hyperplane (cont'd)

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, m


w
Class 2
Class 1
m
$\mathbf{w}^T\mathbf{x}+b=1$
$\mathbf{w}^T\mathbf{x}+b=0$
$\mathbf{w}^T\mathbf{x}+b=-1$

Optimal Hyperplane (cont'd)

- Consider a support vector $\mathbf{x}^{(s)}$ for which $y^{(s)}=+1$, then by definition we have
  $g(\mathbf{x}^{(s)})=\mathbf{w}_0^T\mathbf{x}^{(s)}\mp b_0=\mp 1 \quad \text{for } y^{(s)}=\mp 1.$
  The algebraic distance from the support vector $\mathbf{x}^{(s)}$ to the optimal hyperplane is

$$r=\frac{g(\mathbf{x}^{(s)})}{\|\mathbf{w}_0\|}=\begin{cases}\dfrac{1}{\|w_0\|} & \text{if } y^{(s)}=+1.\\ -\dfrac{1}{\|w_0\|} & \text{if } y^{(s)}=-1.\end{cases} \quad \text{and } m=2r=\frac{2}{\|\mathbf{w}_0\|}.$$

Optimization Problem

- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly $\Rightarrow$ $y_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1, \quad \forall i$
- A constrained optimization problem

$$\text{Minimize } \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1 \qquad \forall i$$

Optimization Problem

- The constrained optimization problem can be solved using the method of Lagrange multipliers. That is, we construct the Lagrangian function:

$$J(\mathbf{w},b,\alpha)=\frac{1}{2}\mathbf{w}^T\mathbf{w}-\sum_{i=1}^{N}\alpha_i\left[y_i(\mathbf{w}^T\mathbf{x}_i+b)-1\right]. \qquad (1)$$

- Minimize the function with respect to $\mathbf{w}$ and b and we get the following two conditions of optimality:

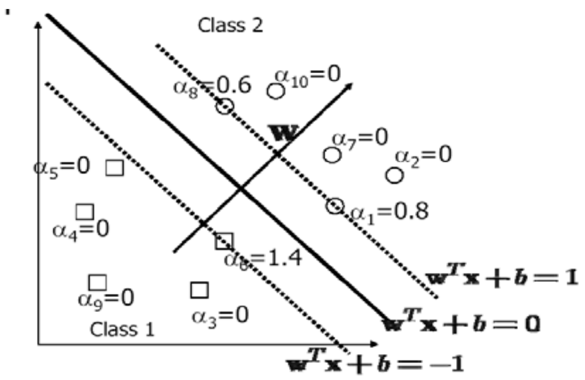
$$\text{Condition 1: } \frac{\partial J(\mathbf{w},b,\alpha)}{\partial \mathbf{w}}=0 \;\Rightarrow\; \mathbf{w}=\sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \qquad (2)$$

$$\text{Condition 2: } \frac{\partial J(\mathbf{w},b,\alpha)}{\partial b}=0 \;\Rightarrow\; \sum_{i=1}^{N}\alpha_i y_i=0 \qquad (3)$$

Characteristics of Solutiions

- Many of the α_i are zero
  - **w** is a linear combination of a small number of data
- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j =1, ..., s) be the indices of the s support vectors. We can write $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class 2 otherwise

Geometrical Interpretation



Some Properties of SVM

- There are theoretical upper bounds on the error on unseen data for SVM
  - The larger the margin, the smaller the bound
  - The smaller the number of SV, the smaller the bound
- Note that in both training and testing, the data are referenced only as inner product, $\mathbf{x}^T\mathbf{y}$
  - This is important for generalizing to the non-linear case

Linearly Nonseparable Patterns

Two cases to consider:

- The data point ($\mathbf{x}_i$, y_i) falls inside the region of separation but on the right side of the decision surface, as illustrated in Fig. 1.
- The data point ($\mathbf{x}_i$, y_i) falls on the wrong side of the decision surface, as illustrated in Fig. 2.

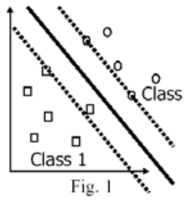


Fig. 1

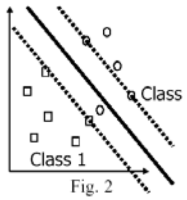
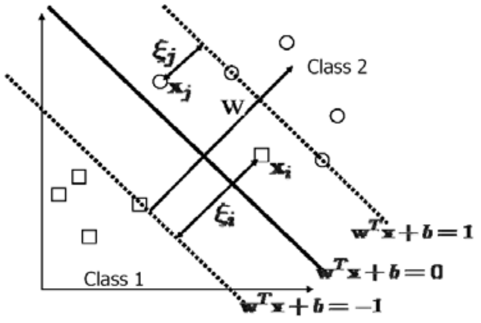


Fig. 2

18

Nonlinear Separation

- We allow "error" ξ_i in classification



57

SVM: A Hyperplane Classifier

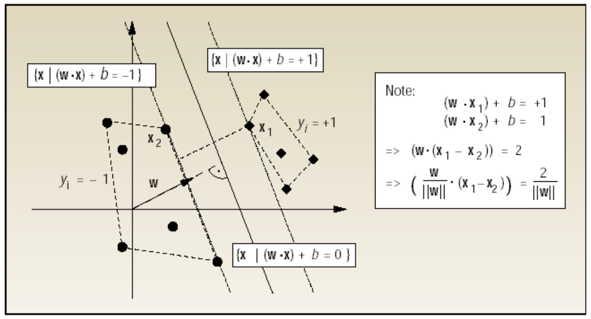

$\{x \mid (w \cdot x) + b = -1\}$
$\{x \mid (w \cdot x) + b = +1\}$
$\{x \mid (w \cdot x) + b = 0\}$
x_2
x_1
$y_i = +1$
$y_i = -1$
w
Note:
$(w \cdot x_1) + b = +1$
$(w \cdot x_2) + b = 1$
$\Rightarrow (w \cdot (x_1 - x_2)) = 2$
$\Rightarrow \left(\frac{w}{\|w\|} \cdot (x_1 - x_2)\right) = \frac{2}{\|w\|}$

Figure 1. A separable classification toy problem: separate balls from diamonds. The optimal hyperplane is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half way. There is a weight vector $\mathbf{w}$ and a threshold b such that $y_i \cdot ((\mathbf{w} \cdot \mathbf{x}_i) + b) > 0$. Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1$, we obtain a form $(\mathbf{w}, b)$ of the hyperplane with $y_i((\mathbf{w} \cdot x_i) + \mathrm{b}) \geq 1$. Note that the margin, measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$. To maximize the margin, we thus have to minimize $|\mathbf{w}|$ subject to $y_i((\mathbf{w} \cdot \mathbf{x}_i) + \mathrm{b}) \geq 1$.

58

SVM: A Hyperplane Classifier

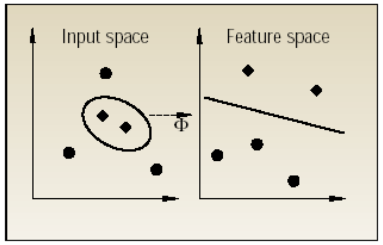


Figure 2. The idea of SV machines: map the training data nonlinearly into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function, it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

59

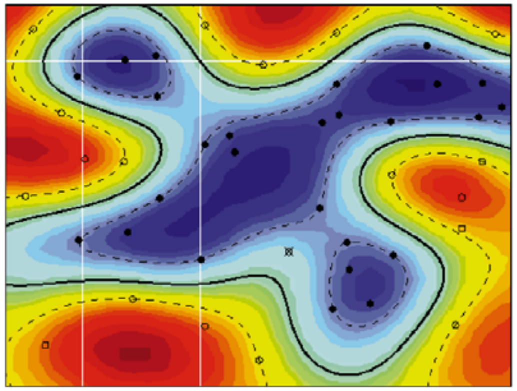


Figure 3. Example of an SV classifier found by using a radial basis function kernel (Equation 8). Circles and disks are two classes of training examples; the solid line is the decision surface; the support vectors found by the algorithm lie on, or between, the dashed lines. Colors code the modulus of the argument $\sum_i v_i \cdot k(\mathbf{x}, \mathbf{x}_i) + b$ of the decision function in Equation 10.

60

Some SVM Features

SVMs combine three important ideas

- Applying optimization algorithms from Operations Research (Linear Programming and Quadratic Programming)
- Implicit feature transformation using kernels
- Control of overfiting by maximizing the margin