CSCE 970: Pattern Recognition	**Spring Semester, 1999**

Topic 6: Sizing the network, Generalized linear classifiers, Decision Trees

Lecturer: Stephen Scott *Scribe: Cătălin Ioan Tomai*

1 Introduction

The object of study of the Pattern Recognition discipline is the problem of classifying objects described by their features into a number of classes by the means of a classifier. Linear classifiers were designed in this goal, but their applicability is limited to linearly-separable classes. The issues that I will discuss in the following pages belong to the larger problem of designing nonlinear classifiers that attempt to classify non-linearly separable classes.

2 Review of previous material

The topics discussed until now dealt with the design of a two layer perceptron that transforms a nonlinearly separable problem into a linearly separable one. The classification capabilities of the perceptron were examined and a three-layer perceptron was proposed to solve the inability of the two-layer perceptron to separate any union of polyhedral regions. The *backpropagation algorithm* was proposed for the training of the perceptrons and some related issues, like the question of when to stop the iterations through the training set? or how to set the learning rate? were discussed. Two variations of the backpropagation scheme, the batch and the pattern modes were presented.

In the following I will present other important issues related to the backpropagation algorithm: the choice of the network size and techniques of eliminating some of the weights in the goal of reducing the complexity of the training process and to avoid overfitting. Also, I will discuss Generalized Linear Classifiers, that is, classifiers that attempt to classify vectors in the k-dimensional space. I will present different kinds of generalized linear classifiers, like the polynomial classifiers or the radial-basis networks. In the end I will present another class of nonlinear classifiers, widely used in this research area, namely, the decision trees.

3 Sizing the Network

This section discusses the choice of the number of layers and neurons of the neural network. For a certain number N of training pairs, the number of weights should be:

- large enough to learn the similarities between the feature vectors in each class and what makes classes different from each other
- small enough so that it does not recognize the differences between the data of the same class.

The most used approaches to train the size of a multi-layer network are:

- Analytical methods - use some algebraic or statistical methods to determine the number of free parameters. These methods are not very realistic.

- Pruning techniques - start with a large network, find the relevant weights and get rid off the non-relevant ones according to a preselected rule.
- Constructive techniques - neurons and weights are successively added to a small initial network, following a certain learning rule.

In the following sections we will discuss the Pruning and Constructive techniques.

3.1 Pruning techniques

The idea here is that if we vary a weight a lot and we do not get much result then the weight must be not very important. Let J be the cost function and the cost change given by:

$$\delta J = \sum_i g_i \delta w_i + \frac{1}{2} \sum_i h_{ii} \delta w_i^2 + \frac{1}{2} \sum_{\substack{i,j \\ i \neq j}} h_{ii} \delta w_i \delta w_j$$

where $g_i = \frac{\partial J}{\partial w_i}$ and $h_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$, as i and j run over the weights. After some simplifications we have that

$$\delta J = \frac{1}{2} \sum_i h_{ii} \delta w_i^2$$

The contribution of each parameter to the above cost is given by the *saliency* value s_i which is given by the expression: $s_i = \frac{h_{ii} w_i^2}{2}$. If the *saliency* factor is small then the weight w_i doesn't affect too much the overall cost. The pruning algorithm has the following steps:

- train the network using the backpropagation algorithm
- compute the salience for each weight w_i, Remove the weight if its saliency is below a certain threshold.
- continue the process with the remaining weights. The process stops according to a preestablished criterion.

Figure 1 shows the decision surfaces separating two classes "+" and "o" before and after the pruning. In the final MLP(Multi Layer Perceptron), only 25 of the initial 480 weights were left. This method gives us savings in terms of complexity and overhead.

Another pruning technique imposes a penalty on small weights. The cost function has the form $J' = J + \alpha \varepsilon_p(w)$. The Penalty (the second term) depends on the weight vector and is chosen to favor small values for the weights. The weights that do not affect much the output of the network do not affect much the first term of the function. The penalty term drives them down and they are eliminated when they are smaller than a certain threshold. In this way we prune the set of weights by eliminating the insignificant ones.

3.2 Constructive techniques

The first technique, named Cascade correlation starts only with the input and output units. Hidden layers are added gradually if the residual error on the outputs is too big. The hidden layers are connected to the network by two kinds of weights. The first kind links the neurons of the hidden layer with the input nodes and the outputs of the previously added hidden neurons. The weights are computed so that they correlate well the new unit's output to the residual error prior to the adding of the new unit, that is, we set the weights such that the output

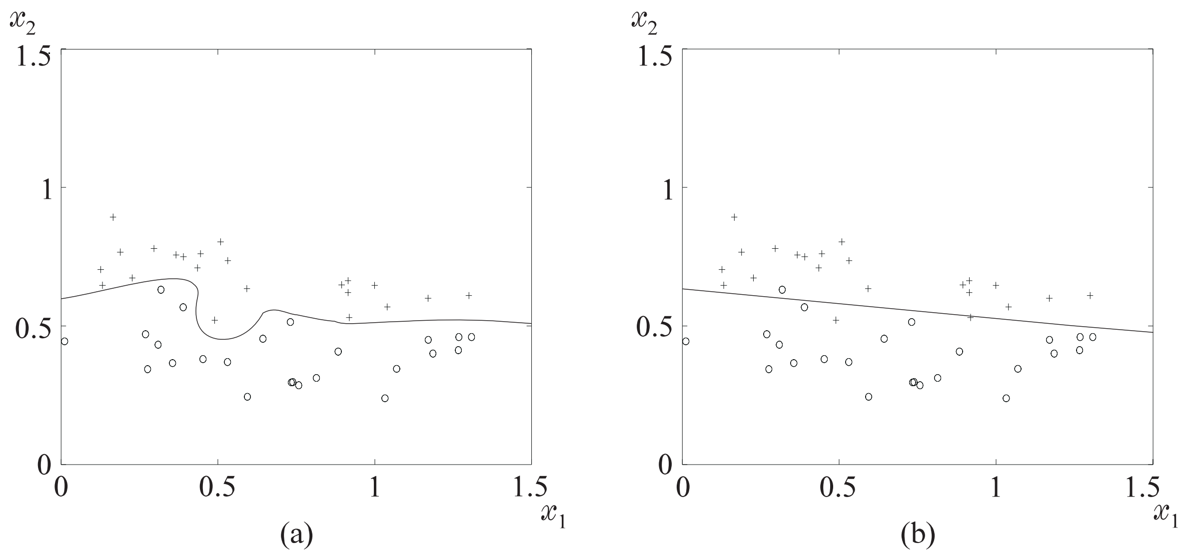

Figure 1: Decision curve before and after pruning

of the hidden unit reflects the output of the output units. Once they are computed, these weights remain fixed. The second kind of weights connect the added neurons with the output nodes. These weights are trained afterwards each time a neuron is added to minimize the cost function. Figure 2 shows the process of adding a hidden unit to the network. The process

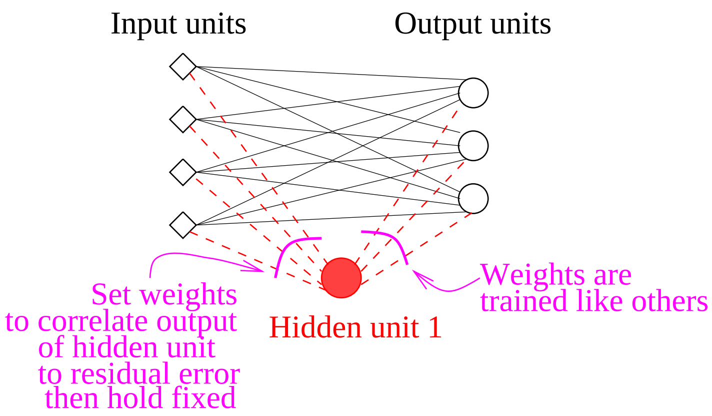

Figure 2: Addition of a hidden unit to the network

stops when the performance of the network is right. The number of hidden layers has to be limited to avoid overfitting.

4 Generalized Linear Classifiers

In this section the goal is to linearly classify the vectors that are not classifiable in the current space by re-mapping them into a higher dimensional space. In the XOR problem we were using a two-layer perceptron's hidden layer to map the regions of the input space to the vertices of a hypercube. In this way the problem becomes linearly-separable. We consider our feature vectors to be in the l-dimensional space $\mathcal{R}^l$. The vectors belong to one of the two classes A

and B which are nonlinearly separable. We will be looking at a specific family of nonlinear functions f_i

$$f_i : \mathcal{R}^l \to \mathcal{R}, i = 1, 2, ..., k$$

that define the mapping:

$$x \in \mathcal{R}^l \to y \in \mathcal{R}^k, y \equiv [f_1(x), f_2(x), ..., f_k(x)]^T$$

We want to find the value of k and the functions f_i so that A and B are linearly separable in the k-dimensional space. That is, we look for the separating hypersurface $g(x) = 0$ that would separate our mapped vectors. The hypersurface is estimated by the output layer and computed by a linear combination of the $f_i(x)$ functions.

$$g(x) = w_0 + \sum_{i=1}^{k} w_i f_i(x)$$

where w_0 is the threshold. Figure 3 shows a Generalized linear classifier. The f_i functions are

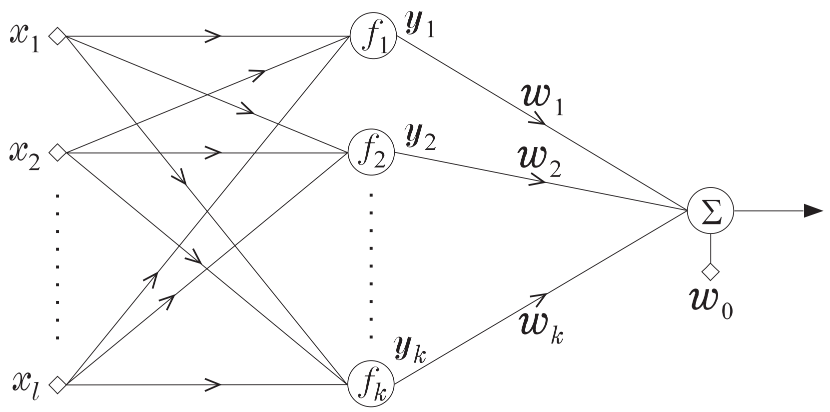

Figure 3: Generalized linear classifier

interpolation functions. We will use different classes of functions to do the approximation.

In the following we want to give a justification of the above assumption that if we increase the dimensionality of the space we also increase the probability of finding a separating hyperplane in the output layer. Let's say that we have N points in the l-dimensional space. We do not consider the case in which the points are not well distributed, that is, when there is a subset of $l+1$ of them that lie on an $(l-1)$-dimensional plane. E.g, in 2D no three points are colinear. Let's say that we have two classes, w_1 and w_2. The number of classifications formed by the $(l-1)$ hyperplanes to separate the N points in w_1 and w_2 is given by the formula:

$$O(n, l) = 2 \sum_{i=0}^{l} \binom{N-1}{i}$$

The factor of 2 in the above expression comes from the fact that we can have the cases: w_1 at the top, w_2 at the bottom and vice-versa. Each one of these arbitrary classifications is called a *dichotomy*. If $N \leq l+1$ then $O(N, l) = 2^N$, otherwise $O(N, l) < 2^N$. In the latter case we cannot obtain all the dichotomies. By example, when $N = 4$ and $l = 2$, the number of dichotomies

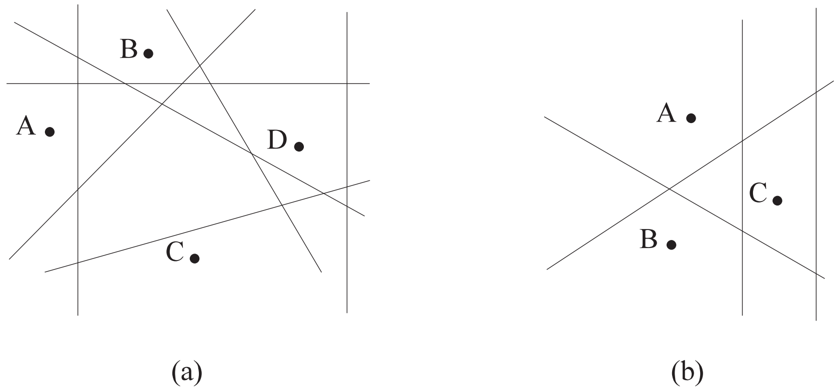

Figure 4: Linear dichotomies for four and three points

$= 14$, which is less than the total possible number of classifications $= 16$. Also, for $N = 3$ and $l = 2$, no. of classifications $= 8$. Figure 4 displays the linear dichotomies for the two cases. We will enumerate them here, where A,B,C are the three points. The first four dichotomies are:

$$
\begin{gathered}
A, B, C \in w_1, \text{ nothing in } w_2 \\
A, B \in w_1, C \in w_2 \\
A, C \in w_1, B \in w_2 \\
B, C \in w_1, A \in w_2.
\end{gathered}
$$

The other four are obtained by changing w_1 with w_2 (the factor of 2 in the above formula). If $l \geq N - 1$ we can guarantee that a perfect separating hyperplane exists, otherwise we cannot make this guarantee. The probability of classifying N points in the l-dimensional space in two linearly separable classes is given by the expression:

$$
P_N^l = \frac{O(N,l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^{l} \begin{pmatrix} N-1 \\ i \end{pmatrix} & N > l+1 \\ 1 & N \leq l+1 \end{cases}
$$

In order to study the dependence of P_N^l on N and l we consider $r = N/(l+1)$ and plot the probabilities for variable r. From what we see in Figure 5, as we increase the dimensions (the l), the curve becomes sharper. That is, when we increase the dimensionality, and $N < 2(l+1)$ the probability of having two groups linearly separable approaches unity. In conclusion, if we have N points, mapping them into a high-dimensional space increases the probability of linearly classifying them.

4.1 Polynomial classifiers

As we said in the introductory part, we are interested in different classes of functions f_i when approximating the $g(x)$ function. Here we will discuss the case of polynomial functions. That is, $g(x)$ is approximated by polynomials of grade at most r of the x components. E.g, for $r = 2$ we have:

$$
\widehat{g}(x) = w_0 + \overbrace{\sum_{i=1}^{l} w_i x_i}^{w_1 f_1 + \ldots + w_l f_l} + \overbrace{\sum_{i=1}^{l=1} \sum_{m=i+1}^{l} w_{im} x_i x_m}^{w_{l+1} f_{l+1} + \ldots + w_{k-l} f_{k-l}} + \overbrace{\sum_{i=1}^{l} w_{ii} x_i^2}^{w_{k-l} f_{k-l} + \ldots + w_k f_k} \quad , k = l(l+3)/2
$$

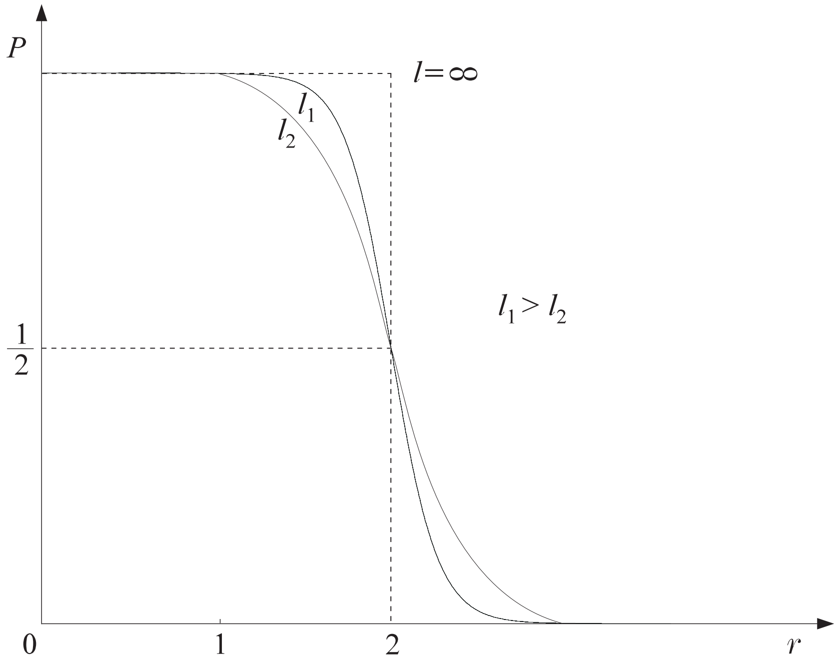

Figure 5: Probability of linearly separable groupings of N = r(l+1) points in the l-dimensional space

where k is the number of nodes in the hidden layer. By example, for $l = 2$, $x = [x_1, x_2]^T$ and $y = [x_1, x_2, x_1x_2, x_1^2, x_2^2]^T$ ($x_1 = f(1), x_2 = f(2), x_1x_2 = f(3), x_1^2 = f(4), x_2^2 = f(5)$) the function $\widehat{g}(x) = x^T y + w_0$ and $w^T = [w_1, w_2, w_{12}, w_{11}, w_{22}]$.

When we generalize the above formula for $r > 2$, $g(x)$ will contain products of the form $x_1^{p1} x_2^{p2} ... x_l^{pl}$ where $p_1 + p_2 + ... + p_l \leq r$. It can be shown that $k = \frac{(l+r)!}{r!l!}$. The quantity k can become very big, even for small values of l and r, so the time to classify and update becomes exponential in $(l + r)$. E.g. for $l = r = 10$, k = 184,756!. Because the number of hidden layers blows up exponentially we have to keep them pretty reasonable.

Let's take the XOR problem as example. We consider $y = [x_1, x_2, x_1x_2]^T$, by omitting the x_1^2 and x_2^2 terms. The input vectors are mapped onto the vertices of a three-dimensional hypercube as we can see in Figure 6. The points are separated by the plane defined by the relation:

$$g(y) = y_1 + y_2 - 2y_3 - \frac{1}{4}$$

or, when written in terms of x:

$$g(x) = -\tfrac{1}{4} + x_1 + x_2 - 2x_1x_2.$$

$g(x) > 0$ when $x \in A$ and $g(x) < 0$ when $x \in B$

4.2 Radial basis function networks

The interpolation functions f_i considered in this section are of the general form: $f(\| x - \| c_i)$. The argument of the function f_i is the Euclidean distance between x and the designated center c_i. Function f can take different forms, like:

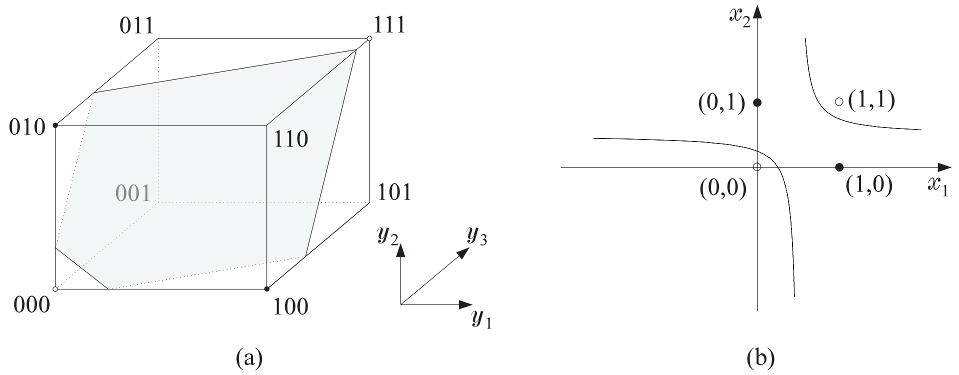

Figure 6: Decision plane in the three-dimensional space and in the two-dimensional space

$$f(x) = exp(\frac{1}{2\sigma_i^2} \parallel x - x_i \parallel)$$

or

$$f(x) = \frac{\sigma^2}{\sigma^2 + \parallel x - c_i \parallel^2}$$

with the Gaussian form the preferred one. For large values of k it can be shown that the function $g(x)$ is approximated by:

$$\widehat{g}(x) = w_0 + \sum_{i=1}^{k} w_i exp\Big(- \frac{(x - c_i)^T (x - c_i)}{2\sigma_i^2}\Big)$$

The method is close to the *nearest neighbour* approaches because only the f_i's with centers near x will have significant output, otherwise it will be negligible. The output of node i will drop exponentially as we increase the distance to the center. Let's take the same example of the XOR problem. Now we will use a RBF (Radial Basis Function) to do the classification. We choose k = 2, the centers $c_1 = [1,1]^T$, $c_2 = [0,0]^T$, and $f(x) = exp(-\parallel x - c_i \parallel^2)$ The corresponding y is given by:

$$y = y(x) = \begin{bmatrix} exp(-\parallel x - c_1 \parallel^2) \\ exp(-\parallel x - c_2 \parallel^2) \end{bmatrix}$$

Hence, for our case we obtain:

Class	$[x_1, x_2]^T$	$[y_1, y_2]^T$
$w_1(A)$	$[0,1]^T$	$[0.368, 0.368]^T$
$w_1(A)$	$[1,0]^T$	$[0.368, 0.368]^T$
$w_2(B)$	$[0,0]^T$	$[0, 135, 1]^T$
$w_2(B)$	$[1,1]^T$	$[1, 0.135]^T$

In the left part of Figure 7 we can see the class position after the mapping in the y space. The two classes are now separated by the hyperplane:

$$g(y) = y_1 + y_2 + 1$$

In the right part of Figure 7 we have the equivalent decision curve given by:

$$g(y) = -1 + e^{-\|x-c_1\|^2} + e^{-\|x-c_2\|^2}$$

As we can see, if x is close to the centers then we have a good response. The natural question

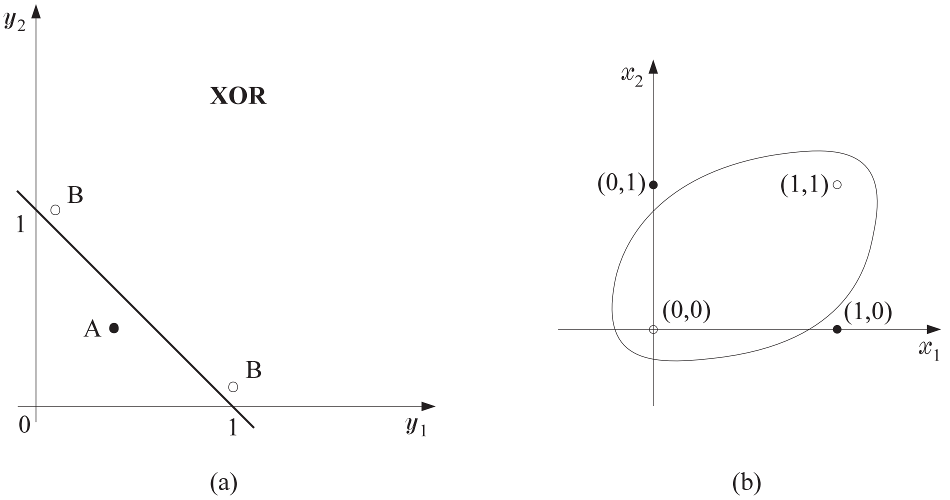

Figure 7: Decision curves in the transformed space and in the original space

is: How do we choose the centers (c_1 and c_2 in the data)? Response: we try to infer them from the data!

The first proposed method randomly selects them from the training set. This could be a reasonable method if we assume that the training set is distributed in a representative manner over all the feature vector space. Unfortunately, using this method we can miss even an entire class!

The second method tries to find them in the training phase, along with the weights w_i. This method is often computationally complex.

A third method would use clustering to choose the centers. That is, we take the data, separate it in groups without looking at the labels (to which class they belong). After that, we choose a representative from each cluster. A variation would be to allow the representative to not be one of the input points. Another way would be to start with a single node and train the network. By considering more and more input vectors that we use to update the parameters of the network according to an algorithm (like the gradient descent scheme) we come out with one that no matter how much we try does not belong to the cluster. That is, the input vector is very far from the existing centers (according to a preestablished threshold) and the output error of the network is greater than a certain threshold. That means that it is time to choose a new center which will be the new vector.

5 Decision trees

This section discusses a different kind of nonlinear classifiers, namely the *decision trees*. In this method classes are sequentially rejected until we reach a finally accepted class. In order to do this we divide the feature space into unique regions, corresponding to the classes, in a sequential manner. For a certain feature vector we look for the class to which to assign this

vector by making a sequence of decisions along the nodes of the *decision tree*. We start from the root and when we reach a leaf we have the answer. Some of these decision trees split the space into rectangles with sides parallel to the axis. An example of such a space is shown in Figure 8, together with the corresponding decision tree. By example, the vector $x = [1/2, 1/4]^T$ is classified as w_3.
The immediate question that arises is: How do we choose these trees? Here is the algorithm:

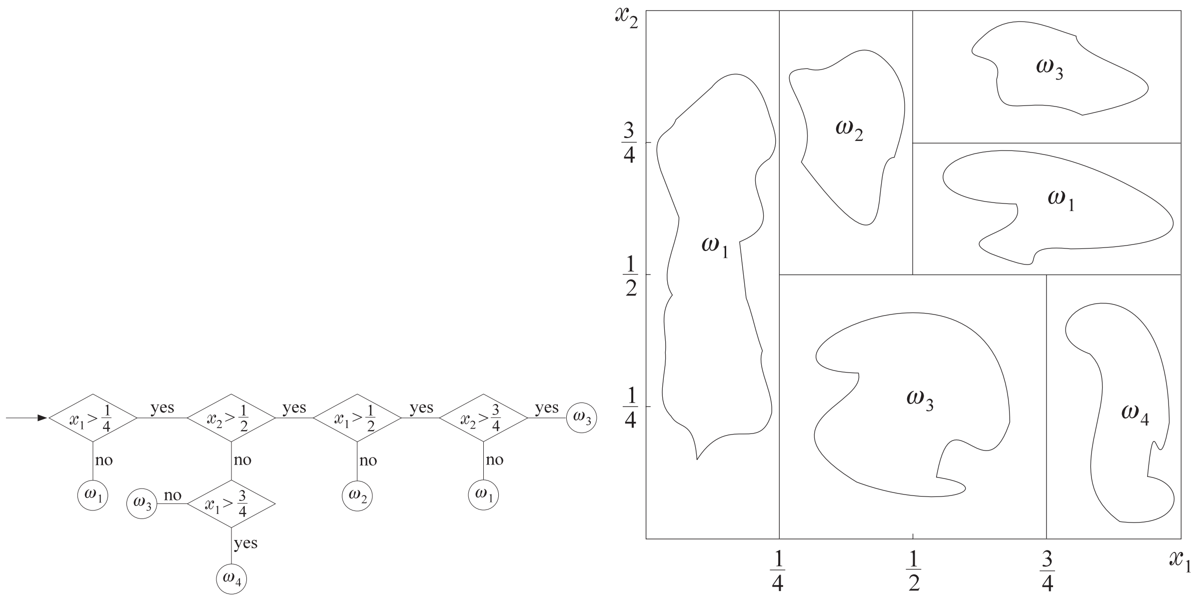

Figure 8: Decision tree and the corresponding divided space

At the root of the tree we consider the feature that yields the highest information gain. First the space is partitioned along dimensions into a set A of pairs (feature,positions), where the positions indicate where the classes change:

$$A = \{(x_1, 0), (x_1, 1/4), (x_1, 3/4), (x_2, 0), (x_2, 1/2), (x2, 3/4)\}$$

For $a = (x_i, b) \in A$ we define: $S_a = \{x \in S : x_i > b\}$ and $S_a^{'} = \{x \in S : x_i \leq b\}$
We also define:

$$Gain(S, a) = Ent(S) - \left(\frac{|S_a|}{S} Ent(|S_a|) + \frac{|S_a^{'}|}{|S|} Ent(S_a^{'})\right)$$

where the entropy (which measuers the impurity of the data) is defined as follows:

$$Ent(S) = \sum_{i=1}^{M} -p_i log_2(p_i)$$

S represents the set of N feature vectors. By example, for the pair (x_1,1/4) the second term of the previous expression has value 0. We choose the a from A that maximizes *Gain*, we place it at the root and recursively compute S_a and $S_a^{'}$. This algorithm is the basis of other widely-used algorithms ID3 and C4.5. We can prevent over-fitting by pruning.