Chapter 9

Unsupervised Learning

9.1 What is Unsupervised Learning?

Consider the various sets of points in a two-dimensional space illustrated in Fig. 9.1. The first set (a) seems naturally partitionable into two classes, while the second (b) seems difficult to partition at all, and the third (c) is problematic. *Unsupervised learning* uses procedures that attempt to find natural partitions of patterns. There are two stages:

- Form an R-way partition of a set Ξ of *unlabeled* training patterns (where the value of R, itself, may need to be induced from the patterns). The partition separates Ξ into R mutually exclusive and exhaustive subsets, $\Xi_1, \ldots, \Xi_R$, called *clusters.*

- Design a classifier based on the labels assigned to the training patterns by the partition.

We will explain shortly various methods for deciding how many clusters there should be and for separating a set of patterns into that many clusters. We can base some of these methods, and their motivation, on minimum-description-length (MDL) principles. In that setting, we assume that we want to encode a description of a set of points, Ξ, into a message of minimal length. One encoding involves a description of each point separately; other, perhaps shorter, encodings might involve a description of clusters of points together with how each point in a cluster can be described given the cluster it belongs to. The specific techniques described in this chapter do not explicitly make use of MDL principles, but the MDL method has

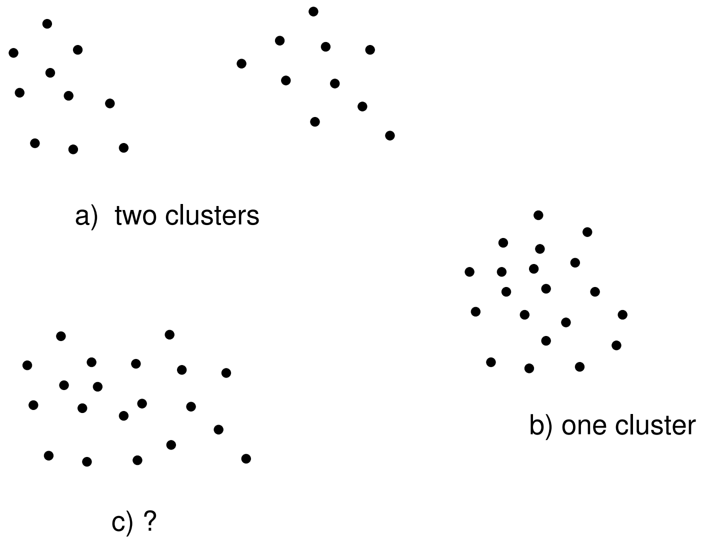

Figure 9.1: Unlabeled Patterns

been applied with success. One of the MDL-based methods, Autoclass II [Cheeseman, *et al.*, 1988] discovered a new classification of stars based on the properties of infrared sources.

Another type of unsupervised learning involves finding hierarchies of partitionings or clusters of clusters. A *hierarchical partition* is one in which Ξ is divided into mutually exclusive and exhaustive subsets, $\Xi_1, \ldots, \Xi_R$; each set, Ξ_i, $(i = 1, \ldots, R)$ is divided into mutually exclusive and exhaustive subsets, and so on. We show an example of such a hierarchical partition in Fig. 9.2. The hierarchical form is best displayed as a tree, as shown in Fig. 9.3. The tip nodes of the tree can further be expanded into their individual pattern elements. One application of such hierarchical partitions is in organizing individuals into taxonomic hierarchies such as those used in botany and zoology.

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

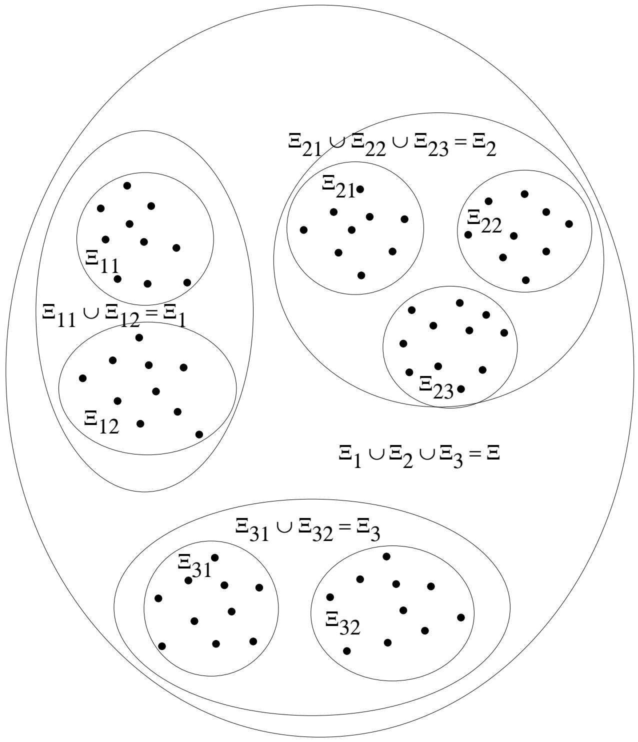

Figure 9.2: A Hierarchy of Clusters

9.2 Clustering Methods

9.2.1 A Method Based on Euclidean Distance

Most of the unsupervised learning methods use a measure of similarity between patterns in order to group them into clusters. The simplest of these involves defining a *distance* between patterns. For patterns whose features are numeric, the distance measure can be ordinary Euclidean distance between two points in an n-dimensional space.

There is a simple, iterative clustering method based on distance. It can be described as follows. Suppose we have R randomly chosen *cluster seekers*, $\mathbf{C}_1, \ldots, \mathbf{C}_R$. These are points in an n-dimensional space that we want to adjust so that they each move toward the center of one of the clusters of patterns. We present the (unlabeled) patterns in the training

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

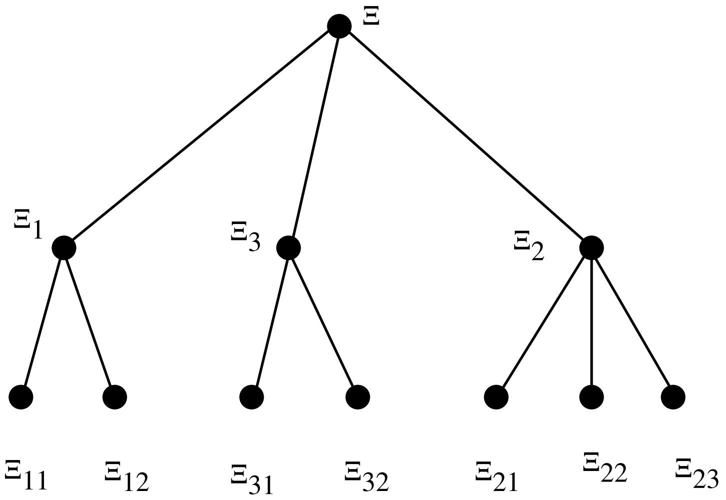

Figure 9.3: Displaying a Hierarchy as a Tree

set, Ξ, to the algorithm one-by-one. For each pattern, $\mathbf{X}_i$, presented, we find that cluster seeker, $\mathbf{C}_j$, that is closest to $\mathbf{X}_i$ and move it closer to $\mathbf{X}_i$:

$$\mathbf{C}_j \longleftarrow (1 - \alpha_j)\mathbf{C}_j + \alpha_j \mathbf{X}_i$$

where α_j is a learning rate parameter for the j-th cluster seeker; it determines how far $\mathbf{C}_j$ is moved toward $\mathbf{X}_i$.

Refinements on this procedure make the cluster seekers move less far as training proceeds. Suppose each cluster seeker, $\mathbf{C}_j$, has a *mass*, m_j, equal to the number of times that it has moved. As a cluster seeker's mass increases it moves less far towards a pattern. For example, we might set $\alpha_j = 1/(1+m_j)$ and use the above rule together with $m_j \longleftarrow m_j + 1$. With this adjustment rule, a cluster seeker is always at the center of gravity (sample mean) of the set of patterns toward which it has so far moved. Intuitively, if a cluster seeker ever gets within some reasonably well clustered set of patterns (and if that cluster seeker is the only one so located), it will converge to the center of gravity of that cluster.

Once the cluster seekers have converged, the classifier implied by the now-labeled patterns in Ξ can be based on a Voronoi partitioning of the

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

space (based on distances to the various cluster seekers). This kind of classification, an example of which is shown in Fig. 9.4, can be implemented by a linear machine.

Georgy Fedoseevich Voronoi, was a Russian mathematician who lived from 1868 to 1909.

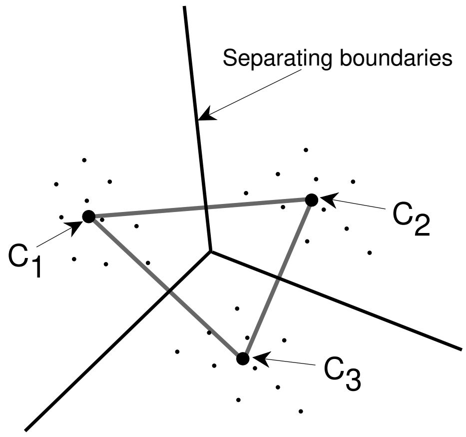

Figure 9.4: Minimum-Distance Classification

When basing partitioning on distance, we seek clusters whose patterns are as close together as possible. We can measure the *badness*, V, of a cluster of patterns, $\{\mathbf{X}_i\}$, by computing its *sample variance* defined by:

$$V = (1/K)\sum_i (\mathbf{X}_i - \mathbf{M})^2$$

where $\mathbf{M}$ is the sample mean of the cluster, which is defined to be:

$$\mathbf{M} = (1/K)\sum_i \mathbf{X}_i$$

and K is the number of points in the cluster.

We would like to partition a set of patterns into clusters such that the sum of the sample variances (badnesses) of these clusters is small. Of course if we have one cluster for each pattern, the sample variances will all be zero, so we must arrange that our measure of the badness of a partition must increase with the number of clusters. In this way, we can seek a trade-off between the variances of the clusters and the number of them in a way somewhat similar to the principle of minimal description length discussed earlier.

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

Elaborations of our basic cluster-seeking procedure allow the number of cluster seekers to vary depending on the distances between them and depending on the sample variances of the clusters. For example, if the distance, d_{ij}, between two cluster seekers, $\mathbf{C}_i$ and $\mathbf{C}_j$, ever falls below some threshold ε, then we can replace them both by a single cluster seeker placed at their center of gravity (taking into account their respective masses). In this way we can decrease the overall badness of a partition by reducing the number of clusters for comparatively little penalty in increased variance.

On the other hand, if any of the cluster seekers, say $\mathbf{C}_i$, defines a cluster whose sample variance is larger than some amount δ, then we can place a new cluster seeker, $\mathbf{C}_j$, at some random location somewhat adjacent to $\mathbf{C}_i$ and reset the masses of both $\mathbf{C}_i$ and $\mathbf{C}_j$ to zero. In this way the badness of the partition might ultimately decrease by decreasing the total sample variance with comparatively little penalty for the additional cluster seeker. The values of the parameters ε and δ are set depending on the relative weights given to sample variances and numbers of clusters.

In distance-based methods, it is important to scale the components of the pattern vectors. The variation of values along some dimensions of the pattern vector may be much different than that of other dimensions. One commonly used technique is to compute the standard deviation (*i.e.*, the square root of the variance) of each of the components over the entire training set and normalize the values of the components so that their adjusted standard deviations are equal.

9.2.2 A Method Based on Probabilities

Suppose we have a partition of the training set, Ξ, into R mutually exclusive and exhaustive clusters, $C_1, \ldots, C_R$. We can decide to which of these clusters some arbitrary pattern, $\mathbf{X}$, should be assigned by selecting the C_i for which the probability, $p(C_i|\mathbf{X})$, is largest, providing $p(C_i|\mathbf{X})$ is larger than some fixed threshold, δ. As we saw earlier, we can use Bayes rule and base our decision on maximizing $p(\mathbf{X}|C_i)p(C_i)$. Assuming conditional independence of the pattern components, x_i, the quantity to be maximized is:

$$S(\mathbf{X}, C_i) = p(x_1|C_i)p(x_2|C_i)\cdots p(x_n|C_i)p(C_i)$$

The $p(x_j|C_i)$ can be estimated from the sample statistics of the patterns in the clusters and then used in the above expression. (Recall the linear form that this formula took in the case of binary-valued components.)

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

We call $S(\mathbf{X}, C_i)$ the *similarity* of $\mathbf{X}$ to a cluster, C_i, of patterns. Thus, we assign $\mathbf{X}$ to the cluster to which it is most similar, providing the similarity is larger than δ.

Just as before, we can define the sample mean of a cluster, C_i, to be:

$$\mathbf{M}_i = (1/K_i) \sum_{\mathbf{X}_j \epsilon\ C_i} \mathbf{X}_j$$

where K_i is the number of patterns in C_i.

We can base an iterative clustering algorithm on this measure of similarity [Mahadevan & Connell, 1992]. It can be described as follows:

1. Begin with a set of unlabeled patterns Ξ and an empty list, L, of clusters.

2. For the next pattern, $\mathbf{X}$, in Ξ, compute $S(\mathbf{X}, C_i)$ for each cluster, C_i. (Initially, these similarities are all zero.) Suppose the largest of these similarities is $S(\mathbf{X}, C_{max})$.

   (a) If $S(\mathbf{X}, C_{max}) > \delta$, assign $\mathbf{X}$ to C_{max}. That is,

   $$C_{max} \longleftarrow C_{max} \cup \{\mathbf{X}\}$$

   Update the sample statistics $p(x_1|C_{max}), p(x_2|C_{max}), \ldots, p(x_n|C_{max})$, and $p(C_{max})$ to take the new pattern into account. Go to 3.

   (b) If $S(\mathbf{X}, C_{max}) \leq \delta$, create a new cluster, $C_{new} = \{\mathbf{X}\}$ and add C_{new} to L. Go to 3.

3. Merge any existing clusters, C_i and C_j if $(\mathbf{M}_i - \mathbf{M}_j)^2 < \varepsilon$. Compute new sample statistics $p(x_1|C_{merge}), p(x_2|C_{merge}), \ldots, p(x_n|C_{merge})$, and $p(C_{merge})$ for the merged cluster, $C_{merge} = C_i \cup C_j$.

4. If the sample statistics of the clusters have not changed during an entire iteration through Ξ, then terminate with the clusters in L; otherwise go to 2.

The value of the parameter δ controls the number of clusters. If δ is high, there will be a large number of clusters with few patterns in each cluster. For small values of δ, there will be a small number of clusters with many patterns in each cluster. Similarly, the larger the value of ε, the smaller the number clusters that will be found.

Designing a classifier based on the patterns labeled by the partitioning is straightforward. We assign any pattern, $\mathbf{X}$, to that category that maximizes $S(\mathbf{X}, C_i)$.

Mention "k-means and "EM" methods.

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

9.3 Hierarchical Clustering Methods

9.3.1 A Method Based on Euclidean Distance

Suppose we have a set, Ξ, of unlabeled training patterns. We can form a hierarchical classification of the patterns in Ξ by a simple *agglomerative* method. (The description of this algorithm is based on an unpublished manuscript by Pat Langley.) Our description here gives the general idea; we leave it to the reader to generate a precise algorithm.

We first compute the Euclidean distance between all pairs of patterns in Ξ. (Again, appropriate scaling of the dimensions is assumed.) Suppose the smallest distance is between patterns $\mathbf{X}_i$ and $\mathbf{X}_j$. We collect $\mathbf{X}_i$ and $\mathbf{X}_j$ into a cluster, C, eliminate $\mathbf{X}_i$ and $\mathbf{X}_j$ from Ξ and replace them by a *cluster vector*, $\mathbf{C}$, equal to the average of $\mathbf{X}_i$ and $\mathbf{X}_j$. Next we compute the Euclidean distance again between all pairs of points in Ξ. If the smallest distance is between pairs of patterns, we form a new cluster, C, as before and replace the pair of patterns in Ξ by their average. If the shortest distance is between a pattern, $\mathbf{X}_i$, and a cluster vector, $\mathbf{C}_j$ (representing a cluster, C_j), we form a new cluster, C, consisting of the union of C_j and $\{\mathbf{X}_i\}$. In this case, we replace $\mathbf{C}_j$ and $\mathbf{X}_i$ in Ξ by their (appropriately weighted) average and continue. If the shortest distance is between two cluster vectors, $\mathbf{C}_i$ and $\mathbf{C}_j$, we form a new cluster, C, consisting of the union of C_i and C_j. In this case, we replace $\mathbf{C}_i$ and $\mathbf{C}_j$ by their (appropriately weighted) average and continue. Since we reduce the number of points in Ξ by one each time, we ultimately terminate with a tree of clusters rooted in the cluster containing all of the points in the original training set.

An example of how this method aggregates a set of two dimensional patterns is shown in Fig. 9.5. The numbers associated with each cluster indicate the order in which they were formed. These clusters can be organized hierarchically in a binary tree with cluster 9 as root, clusters 7 and 8 as the two descendants of the root, and so on. A ternary tree could be formed instead if one searches for the three points in Ξ whose triangle defined by those patterns has minimal area.

9.3.2 A Method Based on Probabilities

A probabilistic quality measure for partitions

We can develop a measure of the goodness of a partitioning based on how accurately we can guess a pattern given only what partition it is in. Suppose we are given a partitioning of Ξ into R classes, $C_1, \ldots, C_R$. As before, we

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

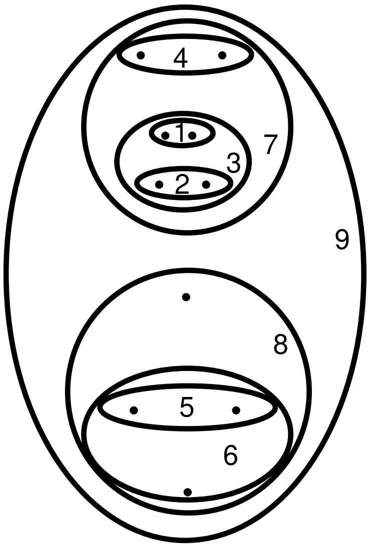

Figure 9.5: Agglommerative Clustering

can compute the sample statistics $p(x_i|C_k)$ which give probability values for each component given the class assigned to it by the partitioning. Suppose each component x_i of $\mathbf{X}$ can take on the values v_{ij}, where the index j steps over the domain of that component. We use the notation $p_i(v_{ij}|C_k) =$ probability$(x_i = v_{ij}|C_k)$.

Suppose we use the following probabilistic guessing rule about the values of the components of a vector $\mathbf{X}$ given only that it is in class k. Guess that $x_i = v_{ij}$ with probability $p_i(v_{ij}|C_k)$. Then, the probability that we guess the i-th component correctly is:

$$\sum_j \text{probability(guess is } v_{ij})p_i(v_{ij}|C_k) = \sum_j [p_i(v_{ij}|C_k)]^2$$

The average number of (the n) components whose values are guessed correctly by this method is then given by the sum of these probabilities over all of the components of $\mathbf{X}$:

$$\sum_i \sum_j [p_i(v_{ij}|C_k)]^2$$

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

Given our partitioning into R classes, the goodness measure, G, of this partitioning is the average of the above expression over all classes:

$$G = \sum_k p(C_k) \sum_i \sum_j [p_i(v_{ij}|C_k)]^2$$

where $p(C_k)$ is the probability that a pattern is in class C_k. In order to penalize this measure for having a large number of classes, we divide it by R to get an overall "quality" measure of a partitioning:

$$Z = (1/R) \sum_k p(C_k) \sum_i \sum_j [p_i(v_{ij}|C_k)]^2$$

We give an example of the use of this measure for a trivially simple clustering of the four three-dimensional patterns shown in Fig. 9.6. There are several different partitionings. Let's evaluate Z values for the following ones: $P_1 = \{a, b, c, d\}$, $P_2 = \{\{a, b\}, \{c, d\}\}$, $P_3 = \{\{a, c\}, \{b, d\}\}$, and $P_4 = \{\{a\}, \{b\}, \{c\}, \{d\}\}$. The first, P_1, puts all of the patterns into a single cluster. The sample probabilities $p_i(v_{i1} = 1)$ and $p_i(v_{i0} = 0)$ are all equal to $1/2$ for each of the three components. Summing over the values of the components (0 and 1) gives $(1/2)^2 + (1/2)^2 = 1/2$. Summing over the three components gives $3/2$. Averaging over all of the clusters (there is just one) also gives $3/2$. Finally, dividing by the number of clusters produces the final Z value of this partition, $Z(P_1) = 3/2$.

The second partition, P_2, gives the following sample probabilities:

$$p_1(v_{11} = 1|C_1) = 1$$

$$p_2(v_{21} = 1|C_1) = 1/2$$

$$p_3(v_{31} = 1|C_1) = 1$$

Summing over the values of the components (0 and 1) gives $(1)^2 + (0)^2 = 1$ for component 1, $(1/2)^2 + (1/2)^2 = 1/2$ for component 2, and $(1)^2 + (0)^2 = 1$ for component 3. Summing over the three components gives 2 1/2 for class 1. A similar calculation also gives 2 1/2 for class 2. Averaging over the two clusters also gives 2 1/2. Finally, dividing by the number of clusters produces the final Z value of this partition, $Z(P_2) = 1\ 1/4$, not quite as high as $Z(P_1)$.

Similar calculations yield $Z(P_3) = 1$ and $Z(P_4) = 3/4$, so this method of evaluating partitions would favor placing all patterns in a single cluster.

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

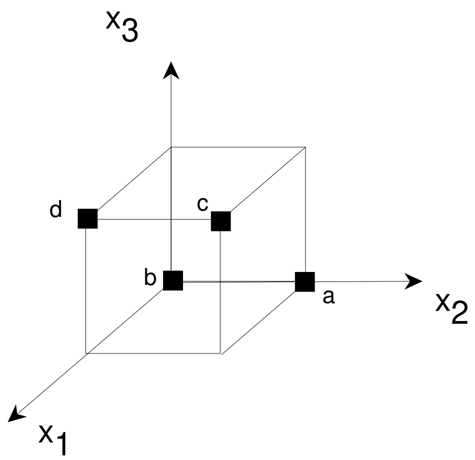

Figure 9.6: Patterns in 3-Dimensional Space

An iterative method for hierarchical clustering

Evaluating all partitionings of m patterns and then selecting the best would be computationally intractable. The following iterative method is based on a hierarchical clustering procedure called COBWEB [Fisher, 1987]. The procedure grows a tree each node of which is labeled by a set of patterns. At the end of the process, the root node contains all of the patterns in Ξ. The successors of the root node will contain mutually exclusive and exhaustive subsets of Ξ. In general, the successors of a node, η, are labeled by mutually exclusive and exhaustive subsets of the pattern set labelling node η. The tips of the tree will contain singleton sets. The method uses Z values to place patterns at the various nodes; sample statistics are used to update the Z values whenever a pattern is placed at a node. The algorithm is as follows:

1. We start with a tree whose root node contains all of the patterns in Ξ and a single empty successor node. We arrange that at all times during the process every non-empty node in the tree has (besides any other successors) exactly one empty successor.

2. Select a pattern $\mathbf{X}_i$ in Ξ (if there are no more patterns to select, terminate).

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

3. Set μ to the root node.

4. For each of the successors of μ (including the empty successor!), calculate the *best host* for $\mathbf{X}_i$. A best host is determined by tentatively placing $\mathbf{X}_i$ in one of the successors and calculating the resulting Z value for each one of these ways of accomodating $\mathbf{X}_i$. The best host corresponds to the assignment with the highest Z value.

5. If the best host is an empty node, η, we place $\mathbf{X}_i$ in η, generate an empty successor node of η, generate an empty sibling node of η, and go to 2.

6. If the best host is a non-empty, singleton (tip) node, η, we place $\mathbf{X}_i$ in η, create one successor node of η containing the singleton pattern that was in η, create another successor node of η containing $\mathbf{X}_i$, create an empty successor node of η, create empty successor nodes of the new non-empty successors of η, and go to 2.

7. If the best host is a non-empty, non-singleton node, η, we place $\mathbf{X}_i$ in η, set μ to η, and go to 4.

This process is rather sensitive to the order in which patterns are presented. To make the final classification tree less order dependent, the COBWEB procedure incorporates node *merging* and *splitting*.

Node merging:

It may happen that two nodes having the same parent could be merged with an overall increase in the quality of the resulting classification performed by the successors of that parent. Rather than try all pairs to merge, a good heuristic is to attempt to merge the two best hosts. When such a merging improves the Z value, a new node containing the union of the patterns in the merged nodes replaces the merged nodes, and the two nodes that were merged are installed as successors of the new node.

Node splitting:

A heuristic for node splitting is to consider replacing the best host among a group of siblings by that host's successors. This operation is performed only if it increases the Z value of the classification performed by a group of siblings.

Example results from COBWEB

We mention two experiments with COBWEB. In the first, the program attempted to find two categories (we will call them *Class 1* and *Class 2*) of

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

United States Senators based on their votes (*yes* or *no*) on six issues. After the clusters were established, the majority vote in each class was computed. These are shown in the table below.

Issue	Class 1	Class 2
Toxic Waste	yes	no
Budget Cuts	yes	no
SDI Reduction	no	yes
Contra Aid	yes	no
Line-Item Veto	yes	no
MX Production	yes	no

In the second experiment, the program attempted to classify soybean diseases based on various characteristics. COBWEB grouped the diseases in the taxonomy shown in Fig. 9.7.

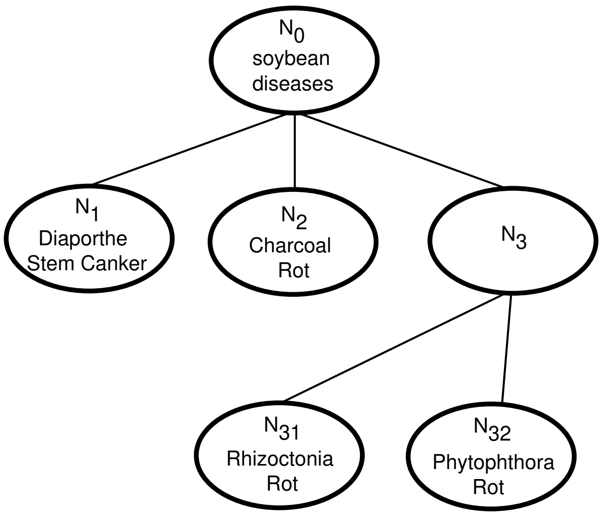

Figure 9.7: Taxonomy Induced for Soybean Diseases

9.4 Bibliographical and Historical Remarks

To be added.

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.

Introduction to Machine Learning ©1996 Nils J. Nilsson. All rights reserved.