CHAPTER 3

The Induction of Threshold Concepts

The logical formalism we considered in the previous chapter is not the only approach to representing conceptual knowledge. In this chapter, we consider an alternative framework that relies on *threshold* concepts. The representational bias of such descriptions differs from that of logical descriptions, making them suitable for different domains.

Much of the work on threshold concepts has occurred within the 'connectionist' or 'neural network' paradigm, which typically uses a network notation to describe acquired knowledge, based on an analogy to structures in the brain. For the sake of unity, we will instead use notations similar to those from earlier chapters. Despite the differences between threshold and logical concepts, we will find that there are also some important similarities.

In the following sections we consider methods for acquiring three broad classes of threshold descriptions from training data – criteria tables, linear threshold units, and spherical threshold units. First we consider these descriptions themselves, along with some generic issues that arise in their use and acquisition. We then examine algorithms for inducing each type of concept from supervised training data.

3.1 General issues for threshold concepts

As we explained in Chapter 1, threshold concepts provide a more flexible representation of knowledge than logical formalisms. The key lies in partial matching, an approach to classification that assigns instances to a class even when only some of the features in the class description are present. For this to occur, the degree of match must exceed some *thresh-*

old, from which derives our name for this scheme. However, there exist many variations on this basic approach to representation, performance, and learning.

3.1.1 Representation and use of criteria tables

The simplest form of threshold description is known as a *criteria table* or an *m-of-n concept*. Such a description includes a set of n Boolean features[1] and an integral threshold m that falls between 1 and n. The conjunction of n features is often referred to as a *prototype*, suggesting a point in the instance space, although it can extend over an entire region of the space when irrelevant features are present. This region serves as the central tendency of the m-of-n concept.

For example, consider the description `2 of {¬one-nucleus, ¬one-tail, thick-wall}` for the dreaded disease `lycanthropy` from the cell domain. This description states that, if any two of the three Boolean features `¬one-nucleus`, `¬one-tail`, or `thick-wall` are present in a particular cell, then it constitutes a positive instance of the concept. Such descriptions can vary in either the features included in their prototype or in their threshold value. Criteria tables have found wide use within the medical community in diagnosis of disease.

One can rewrite an m-of-n description as a simple linear combination rule. In this example, we have

`If ¬one-nucleus + ¬one-tail + thick-wall ≥ 2, then lycanthropy.`

This notation clarifies the performance component associated with such summaries. Briefly, one counts the number of features in the prototype that are present in the instance and, if the sum equals or exceeds the threshold, one labels it as a member of the concept. In this case, a patient having two or more of the specified symptoms would be classified as lycanthropic, whereas those with one or zero symptoms would not.

Note that criteria tables can represent conjunction and disjunction as special cases, with n-of-n indicating a conjunction of n features and 1-of-n specifying a disjunction of n features. However, the scheme is not limited to these extremes in that it can represent intermediate concepts

1. One can extend the method to incorporate internal disjunctions of nominal attributes and ranges of numeric attributes, but one can then treat these as Boolean features in turn.

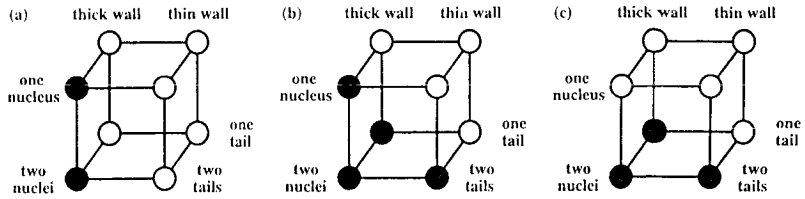

Figure 3-1. Extensional definitions of (a) a conjunctive concept, (b) the criteria table for the concept `lycanthropy`, and (c) a linear threshold unit. Note that (a) can be represented as a 2-of-2 concept but that (b) cannot be described as a conjunction. Similarly, (b) can be described as a linear threshold unit with integral weights, but (c) cannot be handled by such a criteria table.

as well. Figures 3-1 (a) and (b) depict this relationship, with the former showing the extension for the conjunction `¬one-tail ∧ thick-wall`, which can be stated as a 2-of-2 concept, and with the latter showing the extension for `lycanthropy`, which cannot be stated in conjunctive terms.

3.1.2 Representation and use of linear threshold units

Figure 3-1 also makes it clear that there exist some simple concepts that neither conjunctions nor criteria tables can represent. One natural way to handle this larger class is to allow weights on each feature, representing degrees of relevance. For example, the combination

`If −1 one-nucleus + −1 one-tail + −2 thick-wall ≥ −1.5, then lycanthropy ,`

produces the extension shown in Figure 3-1 (c), which one cannot express as any m-of-n description. To classify an instance, one multiplies each observed attribute by its weight, sums the products, and sees if the result exceeds the threshold. Such weighted combinations are often called *linear threshold units* or LTUs, but other names include *perceptrons* and *threshold logic units*.

Clearly, one can view any criteria table as a special case of a linear threshold unit in which all weights are either 1 or −1. But an arbitrary LTU can characterize *any* extensional definition that can be separated by a single hyperplane drawn through the instance space, with the weights specifying the orientation of the hyperplane and the thresh-

old giving its location along a perpendicular. For this reason, target concepts that can be represented by linear units are often referred to as being *linearly separable*.

The representational ability of linear threshold units becomes even more clear in numeric domains. Figure 3-2 (a) shows the extension of an LTU for the height/girth domain from the previous chapter. In this case, the combination rule can be stated as

$$\texttt{If } 1.0\ \texttt{height} + -2.2\ \texttt{girth} \geq 0.5\texttt{, then thin} \ .$$

In words, this description specifies that a person is `thin` if her `height` minus 2.2 times her `girth` is greater than or equal to 0.5. Algebra lets one transform this statement into the relation $\texttt{height} \geq 2.2\,\texttt{girth} + 0.5$, which clarifies the label `thin`.

The simplest linear units assume a 'hard' threshold that acts as a step function to predict 1 (for positive cases) and either 0 or -1 (for negatives). However, they can also predict continuous functions of the observed attributes. Sigmoid functions, which give a smooth transition rather than a step function, are often used to obtain this effect. One commonly used technique calculates the weighted sum $s = \sum_k w_k v_k$ of the observed attribute values v_k, then combines this term with the threshold θ in the logistic function

$$\frac{1}{1 + e^{\theta - s}}$$

to determine a predicted value anywhere between 0 and 1. However, because the classification task still requires an all-or-none decision, LTU-related performance methods typically label an instance as positive if the predicted value is 0.5 or greater.

3.1.3 Representation and use of spherical threshold units

Criteria tables and linear threshold units produce very similar decision boundaries in Boolean domains, although we have seen that the latter subsume the former. However, LTUs do not produce clear prototypes when used in numeric domains, since their extension is unbounded. To generate such an effect, one must move to a different type of threshold description. Figure 3-2 (b) shows the extension of a sample spherical unit that can be stated as

$$\texttt{If } \sqrt{(\texttt{height} - 5.6)^2 + (\texttt{girth} - 2.5)^2} \leq 1.5\texttt{, then normal} \ ,$$

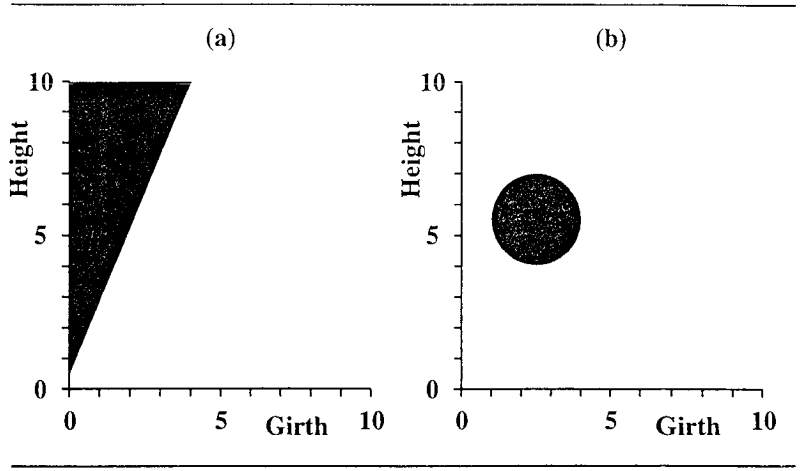

Figure 3-2. Extensional definitions of (a) a linear threshold unit and (b) a spherical threshold unit in the numeric domain of girth and height. The former covers an open region, bounded on one side by a linear equation and on the other only by the limits of the instance space; the latter covers a local region bounded by a spherical equation.

which produces a hypersphere in the instance space. Here the constant associated with each attribute specifies the location of the hypersphere's center along that dimension, whereas the threshold indicates its radius. In this case, any person whose height and girth fall within the given circle would be classified as normal in size.

As with LTUs, one can adapt spherical units to incorporate a sigmoid transformation, so that they predict continuous functions of the input attributes rather than a step function. In this case, we compute the Euclidean distance between an instance and the unit's center as $d = \sqrt{\sum_k (v_k - c_k)^2}$, where v_k is the observed value of attribute k and c_k is the center on that dimension. If we let r be the radius of the hypersphere, then the output of the unit is

$$\frac{1}{1 + e^{r - d}} \ ,$$

which varies between 0, for an instance that falls at the unit's center, and 1, for an instance that lies infinitely far away. Again, we can transform such a prediction into a classification decision by labeling an instance

as positive if the predicted value is 0.5 or less, if we want positives to fall within the sphere, or the opposite, if want them to fall outside.

An alternative approach to achieving a smooth spherical bias involves the use of a Gaussian expression

$$e^{-d/\sigma^2} ,$$

where d is again the distance between an instance and the sphere's center but where σ corresponds to the standard deviation for the Gaussian. Such a description is sometimes called a *radial basis function* or a *localized receptive field*. Gaussian units are often used to predict a numeric output, but they can also be used for classification by predicting a positive instance if the above expression is less than σ.

One can extend the representational power of spherical units by allowing additional terms in the combination rule. Hyperspheres give equal weight to each attribute, which may cause the `normal` rule above to include some thin and plump people that one would like to exclude. One response is to use combinations that produce hyperellipses with axes of different lengths. This requires one additional parameter for each attribute, provided that one is willing to assume that each axis parallels some axis in the instance space. More general hyperellipses that violate this assumption require even more parameters to specify the higher-order interactions among attributes. Similarly, one can extend LTUs to produce nonlinear boundaries on open extensions by introducing quadratic, cubic, and higher-order terms. We will not focus on such extensions in this chapter, but they do provide added representational power to the threshold framework.

3.1.4 Induction of threshold concepts as search

The learning task for threshold concepts is analogous to that for logical ones: given training cases and their associated classes, generate an intensional threshold description that correctly assigns class names to new test cases. As before, one can view this task in terms of search through the space of concept descriptions, constrained both by the training data and by any bias embedded in the inductive algorithm. However, the nature of threshold concepts leads to some interesting differences from the logical framework on this dimension.

One distinction lies in the nature of the operators for moving through the concept space. As we will see in later sections, the use of partial matching means that, even for simple threshold concepts, the addition of features produces more general descriptions rather than more specific ones, since it provides more options for exceeding the threshold. Moreover, the reliance on a threshold introduces new types of learning operators that are concerned with determining its setting. In general, increasing a threshold leads to a more specific concept, whereas decreasing it produces a more general one. In schemes that use linear threshold units, increasing (decreasing) the weight on a single attribute gives greater (less) generality.[2]

Thus, for many threshold concepts there exists a partial ordering by generality much like that for logical concepts, although the larger size of the hypothesis space makes exhaustive search even less plausible than for conjunctive schemes. The partial ordering is most easily seen with criteria tables, since their threshold and weights are effectively discrete. In this case, the partial ordering is a proper superset of that for logical conjunctions, in that each such conjunction can be cast as an m-of-n concept but not vice versa. In Boolean domains, there is a clear partial ordering even for linear threshold units. The situation is more complex for numeric domains, in that arbitrarily small weight changes are possible, but this does not keep logical methods from taking advantage of the partial ordering idea to organize search.

Nevertheless, the vast majority of research on induction of threshold concepts ignores this partial ordering. Instead, it relies primarily on *gradient descent* methods to search the description space. Gradient descent is a special form of incremental hill climbing that uses a difference equation to compute a revised description from the current hypothesis H and H's behavior on a training instance. Typically, the difference equation is closely linked to an explicit criterion that one is trying to minimize, such as the number of errors in classification or the mean squared error in prediction. In most situations, one determines the difference equation by taking the derivative of the criterion expression with respect to the quantity one wants to minimize. We will see examples of specific difference equations in Sections 3.3 and 3.4.

For now, the important point is that most work in this framework incrementally applies such an equation after each training case, letting

2. Altering the weights on spherical units does not map onto the generality spectrum, since they indicate the central tendency of the unit rather than degrees of relevance.

it gradually converge on some global or local optimum in the space of threshold descriptions. Although such gradient descent methods are inherently incremental, algorithms for inducing threshold concepts often invoke them repeatedly on the training set to mimic the effects of nonincremental processing. Each such pass through the training set is often referred to as an *epoch*. Most such techniques also include a *gain factor*, which influences the size of the step taken through the weight space. We will see other approaches to learning in this framework, but gradient descent of the above sort predominates.

3.2 Induction of criteria tables

Criteria tables are the simplest form of threshold unit, making them a natural place to start our examination of the induction process. In this section, we consider algorithms for inducing such m-of-n concepts from training data. We will focus on nominal attributes and Boolean features, although the basic approach can be extended to numeric domains.

3.2.1 Simple threshold modification

We will begin by focusing on domains that involve only r relevant features and no irrelevant ones. In such cases, we can partition the task of inducing criteria tables into two components: determining the prototype and selecting an appropriate threshold between 1 and r (inclusive). The *threshold revision* (TR) algorithm takes advantage of this division. Given all the training instances, TR first finds the modal (most frequent) value for each attribute, then uses the resulting set P as the prototype. Earlier we noted that the space of criteria tables is ordered by generality. Thus, the algorithm can start with either the most specific possible m-of-n concept, r of P, or the most general one, 1 of P.

In the first scheme, which we will call TRG (for *threshold revision/general*), the only learning operator involves decreasing the threshold by 1, producing a more general hypothesis. In the second method, TRS (for *threshold revision/specific*), the only operator increases the threshold by 1, generating a more specific hypothesis. Although the overall space is only partially ordered, the subset of hypotheses that differ only in their thresholds is totally ordered; as a result, neither strategy requires any real search. TRG and TRS simply continue generalizing or specializing until the quality of the hypothesis fails to improve on some criterion.

Incremental versions of both schemes are also possible. Here one initializes the prototype to the first positive instance, but revises it as new instances alter the modal values for the class. After each new training case, one also considers both increasing and decreasing the threshold, moving to the resulting hypothesis only if such an action provides some improvement on the training data on hand (e.g., the most recent k instances). This algorithm continues to revise its prototype and threshold as long as new training cases suggest improved descriptions.

3.2.2 Feature selection and threshold modification

The above methods are only sufficient to induce criteria tables when all attributes are relevant. In more realistic situations, only r out of n features are relevant, and search is needed to find the target concept. In such cases, simple threshold modifications do not produce minimal changes in concept descriptions. Thus, one must replace the learning operator with more conservative ones. First let us consider a nonincremental, specific-to-general algorithm, HCT, for the heuristic induction of criteria tables, as summarized by the pseudocode in Table 3-1.

There exist two operators that generate minimally more general criteria tables from an existing hypothesis. We will assume that `¬one-tail` and `thick-wall` are part of the prototype, which HCT determines using modal values as in the TR method. The first operator simultaneously removes a feature and lowers the threshold by one. For example, given the current hypothesis `2 of {¬one-nucleus, ¬one-tail, thick-wall}`, this operator generates three alternatives: `1 of {¬one-nucleus, ¬one-tail}`, `1 of {¬one-nucleus, thick-wall}`, and `1 of {¬one-tail, thick-wall}`. The second operator holds the threshold constant but introduces a new feature. For example, given the current hypothesis `1 of {¬one-nucleus}`, it generates the two successors `1 of {¬one-nucleus, ¬one-tail}` and `1 of {¬one-nucleus, thick-wall}`.

At any given stage in the search process, the HCT algorithm must evaluate the competitors according to some criterion. In small, noise-free domains, one might simply eliminate hypotheses that cover any negative training instances, as in the ESG method from Chapter 2. If the hypothesis space is large or if noise may be present, then one must use some heuristic search method, directed by an evaluation function, as described in the previous chapter. As the table shows, HCT uses a form of beam search, retaining only the best hypotheses at each level.

Table 3-1. The HCT algorithm: Heuristic induction of criteria tables using beam search, starting with the most specific possible hypothesis and moving toward more general ones. Replacing the initial hypothesis and the operators produces general-to-specific search.

```
Inputs: PSET is the set of positive instances.
        NSET is the set of negative instances.
        A set of nominal attributes ATTS.
Output: A criteria table for classifying new instances.

Procedure HCT(PSET, NSET, ATTS).

Let the prototype P be the set of most common values
    in PSET for each of the attributes in ATTS.
Let the initial threshold T be the size of ATTS.
Let the initial hypothesis set HSET be {[T of P]}.
Return HCT-Aux(PSET, NSET, P, { }, HSET).

Procedure HCT-Aux(PSET, NSET, P, CLOSED-SET, HSET).

Let OPEN-SET be the empty set.
For each description H in HSET,
    Let SPECS be CT-Specialize(H, P).
    Let NEW-SET be the empty set.
    For each specialized description S in SPECS,
        If Score(S, PSET, NSET) > Score(H, PSET, NSET),
        Then add the description S to NEW-SET.
    If NEW-SET is empty,
    Then add the description H to CLOSED-SET.
    Else for each description S in NEW-SET,
         Add S to OPEN-SET.
If OPEN-SET is empty,
Then return the highest-scoring member of CLOSED-SET.
Else let BEST-SET be the Beam-Size highest-scoring
         members of the union of OPEN-SET and CLOSED-SET.
     Let CLOSED-SET be the members of CLOSED-SET in BEST-SET.
     Let OPEN-SET be the members of OPEN-SET in BEST-SET.
     Return HCT-Aux(PSET, NSET, P, CLOSED-SET, OPEN-SET).
```


A similar scheme organizes search from general to specific hypotheses. In this framework, one learning operator simultaneously adds a feature and increases the threshold by 1, whereas the other retains the current threshold and removes an existing feature. Again, this assumes that the method has computed a plausible prototype at the outset, from which the first operator draws features. In small, noise-free domains, one can eliminate any alternative that fails to cover a positive training case, but again, more realistic problems require heuristic search directed by an evaluation function.

A third viable approach searches from the simplest criteria tables, which occur in the middle of the generality ordering, toward the most complex ones, which occur at the extremes of specificity and generality. Thus, this method's initial hypothesis has the form `1 of {F}`, where the term F may be any of the known features or their negation. For example, for the cell domain, this approach might initialize the hypothesis to be `1 of {¬one-nucleus}` or `1 of {thick-wall}`. One can select the initial feature from a prototype computed at the outset, as in the other techniques, or one can use an evaluation function to select the feature that best discriminates between classes.

This simple-to-complex approach requires some form of bidirectional search using a combination of operators from the above methods. For example, the operators 'add feature/maintain threshold' and 'add feature/increase threshold' are sufficient to generate all criteria tables, starting from any one of the simplest descriptions. The converse pair, 'remove feature/ lower threshold' and 'remove feature/maintain threshold', can also reach any hypothesis within the space. Using all four operators in conjunction may also help with some evaluation functions. Bidirectional use of operators is also necessary in incremental hill-climbing variants that process only a subset of the training data at a time.

Although the simplest methods for inducing criteria tables assume that attributes are nominal or Boolean, one can extend them to handle numeric attributes. One approach, which we consider in Section 3.4, relies on restating the criteria table in terms of spherical threshold units, then searching the resulting weight space. Another scheme, based on analogy with logical induction methods, introduces > and < tests into the concept representation, then searches through the set of possible values to determine the best such test for each attribute.

3.3 Induction of linear threshold units

As noted in Section 3.1, linear threshold units move beyond criteria tables in that they allow nonintegral weights on attributes, thus supporting degrees of relevance. Determination of these weights requires some different learning algorithms from those we have yet considered.

3.3.1 The perceptron revision method

Now let us consider the more common incremental approaches to inducing linear threshold units. The earliest algorithm of this sort, which we will call the *perceptron revision method* (PRM), retains a single LTU in memory and processes one instance at a time. As shown in Table 3-2, the method retains the current hypothesis if it correctly classifies a training case, but classification errors lead to changes in weights. In particular, we have

$$\Delta w_i = \eta v_i$$

as the change in the weight on attribute i when the misclassified instance is positive and

$$\Delta w_i = -\eta v_i$$

when the instance is negative, where v_i is the value of attribute i in the training case and $\eta > 0$ is a gain factor. Thus, each weight w_i is changed in proportion to the attribute value v_i, with the type of error determining the direction of change. This includes the weight corresponding to the threshold, which is treated as a special attribute that always has the value 1.

Figure 3-3 traces the behavior of this induction algorithm on four training cases from the height/girth domain. The PRM method can start with any combination of weights, but these are typically randomly set to small, nonzero values. Here we assume the initial hypothesis

$$1.0\ \texttt{height} + -2.0\ \texttt{girth} \geq 1.5\ ,$$

which is near the target concept (to keep things simple). The line in Figure 3-3 (a) shows the decision boundary produced by this description, with the region for positive instances on the left. We will also use $\eta = 0.04$ as our gain factor, which leads to rapid learning for this example, and we will assume that the algorithm processes training cases in the order (2.0, 5.0), (2.5, 5.0), (1.75, 6.0), and (3.0, 6.25).

Table 3-2. The perceptron revision method (PRM): Incremental induction of linear threshold units using gradient descent search.

```
Inputs:  A hypothesized linear threshold unit H.
         A set of classified training instances ISET.
         A set of attributes ATTS used in the instances.
Output:  A revised linear threshold unit.
Params:  A gain term η that determines revision rate.

Procedure PRM(H, ISET, ATTS).

For each training instance I in ISET,
    Let C be the observed class of instance I.
    Let P be the class hypothesis H predicts for I.
    If P is negative and C is positive,
    Then let the sign of the change S be 1.
    Else if P is positive and C is negative,
         Then let the sign of the change S be -1.
    If predicted class P differs from observed class C,
    Then for each attribute A in ATTS,
              Let W be the weight for A in H.
              Let V be the value of A in instance I.
              Add the product S × η × V to W in H.
         Add the product S × η to the threshold in H.
Return the revised hypothesis H.
```

Figure 3-3 (b) shows that the initial LTU misclassifies the first training instance (marked with a rectangle), which in turn leads to weight modifications. Using the expression above, we obtain $\Delta w = 0.04 \times 1.0 = 0.04$ as the weight change for the threshold, $\Delta w = 0.04 \times 2.0 = 0.08$ for `girth`, and $\Delta w = 0.04 \times 5.0 = 0.2$ for `height`. Adding these quantities to the original LTU gives the revised hypothesis

$$1.2\ \texttt{height} + -1.92\ \texttt{girth} \geq 1.54\ .$$

The solid line in Figure 3-3 (b) shows the decision boundary produced by this threshold unit. From this one can also see that it correctly classifies the second and third training cases, (2.5, 5.0) and (1.75, 6.0), as negative and positive instances, respectively. Thus, no learning takes place when the algorithm processes them.

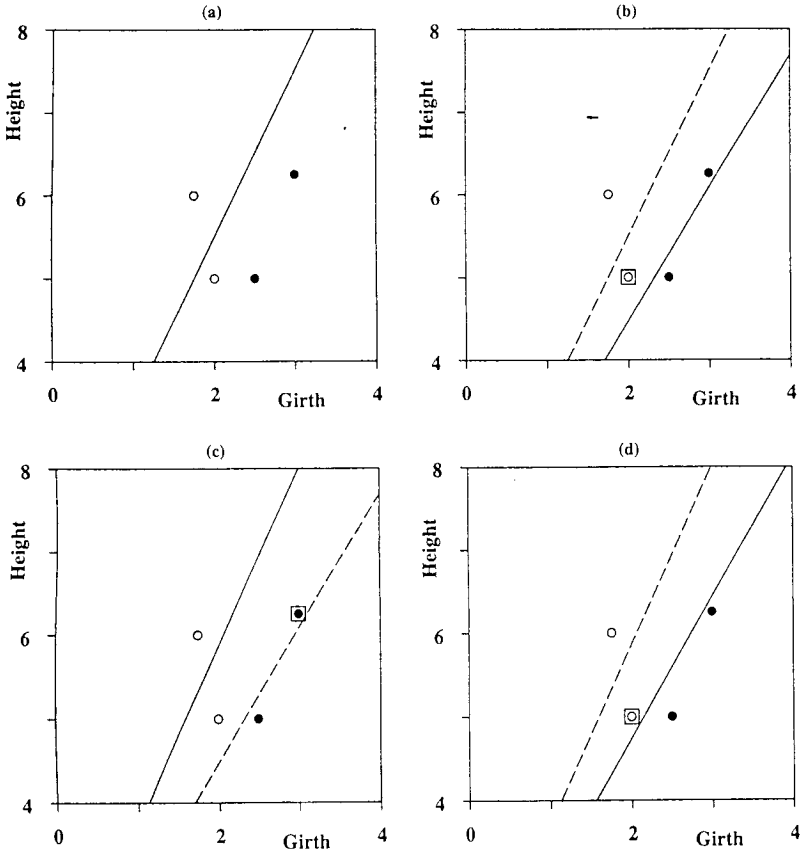

Figure 3-3. Behavioral trace of the perceptron learning algorithm on two positive (white) and two negative (black) training cases from the height/girth domain. The initial LTU in (a) misclassifies the first instance, (2.0, 5.0), as negative, producing the revised decision boundary in (b). This correctly labels the instances processed next, (2.5, 5.0) and (1.75, 6.0), but it mislabels (3.0, 6.25) as positive, giving the revised regions in (c). Finally, a second error on (2.0, 5.0) gives the LTU in (d), which correctly classifies all the training data.

However, as Figure 3-3 (c) shows, the final instance, (3.0, 6.25), is classified as positive rather than negative, initiating further changes. This time, the modifications are $0.04 \times 1.0 = 0.04$ for the threshold, $0.04 \times 3.0 = 0.12$ for `girth`, and $0.04 \times 6.25 = 0.25$ for `height`, but they are subtracted from the current weights rather than added. The result is the revised threshold unit

$$0.95\ \texttt{height} + -2.04\ \texttt{girth} \geq 1.5\ ,$$

which gives the solid decision boundary shown in Figure 3-3 (c). Clearly, even when the algorithm has processed all of the training cases, the resulting threshold unit may not correctly classify them, since the changes introduced may have been small or since later alterations may have offset earlier ones.

For this reason, although the PRM algorithm for perceptron learning is inherently incremental, it is typically run through the entire training set multiple times to mimic a nonincremental method. In this example, the LTU from Figure 3-3 (c) has overcompensated and again misclassifies the first training instance, (2.0, 5.0), labeling it as negative rather than positive. This time the revised hypothesis is

$$1.15\ \texttt{height} + -1.96\ \texttt{girth} \geq 1.54\ ,$$

as shown by the solid line in Figure 3-3 (d). Fortunately, this description correctly classifies the other three instances. Thus, if one runs the algorithm through the data a third time, it will make no errors, having arrived at an LTU that completely separates the classes.

Table 3-3 presents pseudocode for a higher-level algorithm that calls on PRM repeatedly until no errors occur on the training data or until it exceeds a user-specified number of iterations. This method is sometimes called the *perceptron convergence procedure* (PCP) because, if the training set is linearly separable, it is guaranteed to converge in a finite number of iterations on an LTU that makes no errors on these data.

Of course, linear threshold units may provide useful approximations even when the target concepts (and probably the training data) are not linearly separable. Also, the presence of noise can make the training set nonseparable even when this assumption holds for the target concept. In such cases, one can still run the PCP algorithm, but it needs a halting criterion other than the absence of errors on the training data. One method halts PCP when it detects repeated sets of weights, which

Table 3-3. The perceptron convergence procedure (PCP): Nonincremental induction of linear threshold units by applying PRM iteratively to the training set until it produces an LTU that makes no errors.

```
Inputs: A set of classified training instances ISET.
        A set of attributes ATTS used in the instances.
Output: A linear threshold unit for classifying new instances.
Params: A gain term η that determines revision rate.
        The maximum number of iterations Maximum-Iterations.

Procedure PCP(ISET, ATTS).

Let H be an LTU with random weights between −1 and 1.
Let COUNT be Maximum-Iterations.
Repeat until COUNT is zero:
    Let NO-ERRORS be True.
    For each instance I in ISET,
        If H incorrectly classifies I,
        Then let NO-ERRORS be False.
    If NO-ERRORS is True,
    Then return the hypothesis H.
    Else decrement COUNT by one.
         Let H be PRM(H, ISET, ATTS).
Return the hypothesis H.
```

are guaranteed to arise on nonseparable training data. However, this requires that one retain multiple descriptions in memory, and an easier scheme simply halts training after a specified number of iterations, as shown in Table 3-3.

The rate of convergence is significantly affected by the gain term η, which controls the size of steps through the hypothesis space. Small values lead to overly conservative behavior, with each learning event producing only a fraction of the weight change needed to remove the offending error. On the other hand, large values lead to overshooting, with weight changes tending to undo corrections made by earlier ones. One response is to start with a high setting for η and gradually reduce it as learning proceeds. There also exist schemes that determine η adaptively for each training case, selecting a setting that is just sufficient to produce a correct classification on the current instance.

3.3.2 The LMS procedure

The perceptron learning method assumes a hard threshold that predicts a positive instance if exceeded and a negative one otherwise. However, one can adapt the method to handle soft boundaries like those produced by the logistic function we discussed in Section 3.2.1. Here the goal is to learn an LTU that predicts a numeric value between 0 and 1, which corresponds to the negative and positive class, respectively.

One widely used modification method, known as the *least mean square* or *LMS* algorithm, computes

$$\Delta w_i = \eta \delta v_i$$

as the change in the weight on attribute i, where v_i is the value of attribute i, $\eta > 0$ is a gain factor, and

$$\delta = p(1-p)(o-p)$$

is an error term that incorporates the observed value o and the predicted value p for the training case. One can view this scheme as a generalization of the perceptron updating rule, in which δ is either 1 or -1. Thus, the amount of weight change depends not only on the attribute values but also on the amount of error.

The name of the LMS algorithm derives from the fact that it converges on a linear threshold unit that minimizes the mean squared error between the desired and generated output, even when the training data are not linearly separable. In fact, one can obtain the δ term by taking the derivative of this error quantity with respect to the weights. As a result, asymptotically it produces LTUs that tend to split the training instances more evenly than the perceptron method. Thus, the two learning methods have different inductive biases that, in some cases, produce quite different predictions on novel test cases.

Although one can iteratively invoke the LMS difference equation from within an algorithm like PCP, individual instances can produce changes in different directions, which can lead to large oscillations in the weights and slow learning. Another approach, shown in Table 3-4, computes the error and proposed weight changes to the current LTU for each instance, then sums these changes and uses the result to modify the LTU. Another scheme for reducing oscillation, also represented in the table, introduces a 'momentum' term into the update rule. The new equation,

$$\Delta w_i(n) = \eta \delta v_i + \alpha \Delta w_i(n-1) \quad ,$$

Table 3-4. The least mean square (LMS) algorithm: Induction of linear threshold units using gradient descent, with the aim of minimizing the mean squared error between predictions and observations. LMS assumes that the value for the threshold term is always 1.

```
Inputs: A set of classified training instances ISET.
        A set of attributes ATTS used in the instances.
Output: A linear threshold unit for classifying new instances.
Params: A gain term η that determines revision rate.
        A momentum term α that reduces oscillation.
        The maximum number of iterations Maximum-Iterations.
        The minimum acceptable error Minimum-Error.

Procedure LMS(ISET, ATTS).

Let H be an LTU with random weights between −0.5 and 0.5.
For each attribute A in ATTS, let Δ_A be zero.
Let COUNT be Maximum-Iterations.
Repeat until COUNT is zero:
    Let TOTAL-ERROR be zero.
    For each instance I in ISET,
        Let O_I be the observed quantity for instance I.
        Let P_I be the quantity that H predicts for I.
        Add the error (O_I − P_I)^2 to TOTAL-ERROR.
    If TOTAL-ERROR is less than Minimum-Error,
    Then return the hypothesis H.
    Else for each attribute A in ATTS,
            Let GRADIENT be zero.
            For each instance I in ISET,
                Let δ be P_I · (1 − P_I) · (O_I − P_I).
                Let V be the value of A in instance I.
                Add η · δ · V to GRADIENT.
            Let Δ_A be GRADIENT + (α · Δ_A).
            Let W be the weight for A in H.
            Add Δ_A to W in H.
        Decrement COUNT by one.
Return the hypothesis H.
```


takes into account previous weight changes, although they are multiplied by a factor, $0 < \alpha < 1$, that gives them successively less influence. Also note that, because LMS focuses on reducing quantitative errors rather than eliminating misclassifications, the perceptron halting criterion does not apply. Instead, one typically iterates through the training set either a specified number of times or until the error reaches a low enough level.

Let us examine the behavior of the LMS algorithm on the same training data as in Section 3.3.1. We assume that the gain term ν is 0.4, the momentum term α is 0.9, and the initial LTU is

$$0.1 \text{ height} + -0.2 \text{ girth} \geq 0.15 \ ,$$

which produces a decision boundary identical to that in the perceptron example. The mean squared error is 0.922 for this linear threshold unit.

After its first pass through the training set, LMS produces the revised hypothesis

$$0.119 \text{ height} + -0.27 \text{ girth} \geq 0.156 \ ,$$

which gives an error of 0.898 and has moved in roughly the right direction. The next iteration gives the LTU

$$0.193 \text{ height} + -0.384 \text{ girth} \geq 0.174 \ ,$$

which has a further reduced error 0.853 and which correctly classifies the entire training set.

However, the halting criterion for LMS differs from that for the PCP method, and the current algorithm cycles through the data again, producing

$$0.235 \text{ height} + -0.57 \text{ girth} \geq 0.189 \ .$$

Although the mean squared error has decreased again to 0.796, the new LTU actually misclassifies one of the training cases. However, this misclassification is soon remedied on the next round, which gives

$$0.382 \text{ height} + -0.76 \text{ girth} \geq 0.225 \ ,$$

a description that has 0.736 as its mean squared error. Nor does the algorithm halt at this point, but future iterations produce LTUs that correctly classify the training data as they gradually converge on one that minimizes the error.

3.3.3 Iterative weight perturbation

Although most work on the induction of linear threshold units has relied on gradient descent methods like PRM and LMS, other approaches are possible. For example, one can discretize the weight space and use any search method, such as the genetic algorithm from Chapter 2, to explore alternative LTUs. Here we consider one alternative scheme that uses the entire training set to revise weights one at a time.

Recall that the decision boundary generated by an LTU takes the form of a hyperplane,

$$\sum_{i=1}^{d}(w_i x_i) - w_0 = 0 \quad ,$$

where w_0 is the threshold, w_i is the weight for attribute a_i, and x_i is the value of an instance on that attribute. However, few instances actually lie on this hyperplane, and for a given instance j, we can define

$$\sum_{i=0}^{d} w_i x_{ij} = V_j$$

as the perpendicular distance of point j from the hyperplane, where x_{ij} is the value of j on attribute i and where the value x_{0j} is always -1. If V_j is greater than 0, the LTU will classify j as positive, whereas a negative score will result in a negative label. We would like to find a set of weights that produces the greatest number of correct signs for these V scores.

We can also define an analogous term for a particular attribute a_k,

$$(\sum_{i \neq k} w_i x_{ij})/x_{kj} = U_{kj} \quad ,$$

which also assigns a scalar to the instance j. In this case, the LTU will classify j as positive if the weight w_k, which we specifically omitted from the expression, is less than U_{kj} and label it as negative otherwise. Effectively, this lets us determine the effect on accuracy of varying w_k while holding the other weights constant. Given n training instances, we can arrange the U_{kj} values in order of decreasing size, consider settings for w_k that fall between adjacent U values, and select the setting that gives the best classification results on the training data.

Table 3-5. The IWP algorithm: Induction of linear threshold units through iterative perturbation of each attribute's weight, which continues until no further improvements occur in fit to the training data.

```
Inputs: A set of classified training instances ISET.
        A set of attributes ATTS used in the instances.
Output: A linear threshold unit for classifying new instances.
Params: The maximum number of iterations Maximum-Iterations.

Procedure IWP(ISET, ATTS).

Let H be an LTU with random weights between -1 and 1.
Let BEST be the hypothesis H.
Let COUNT be Maximum-Iterations.
Repeat until COUNT is zero:
    For each attribute K in ATTS,
        For each instance J in ISET,
            Compute U_KJ using H and J.
        Place the U_KJ values in decreasing order.
        For each adjacent pair of U values,
            Let W_k' be the average of the pair.
            Let H' be H with W_k replaced by W_k'.
            Compute Score(H', ISET).
        Let H be the LTU with the highest score.
    If Score(H, ISET) is one,
    Then return the hypothesis H.
    Else if Score(H, ISET) ≤ Score(BEST, ISET),
         Then let BEST be the hypothesis H.
         Decrement COUNT by one.
Return the hypothesis BEST.
```

Table 3-5 presents pseudocode for IWP (iterative weight perturbation), a method that iteratively perturbs the weights for each attribute in turn, in each case using the U_{kj} values to determine the best weight for attribute a_k. If the resulting weight vector correctly classifies all training cases, then IWP halts; otherwise it runs through the attribute set a second time, a third, and so on until it exceeds a user-specified number of cycles. Because a revised LTU may have a lower score than its predecessor, IWP retains the best description generated so far, which it returns if it cannot find one that perfectly splits the two classes.

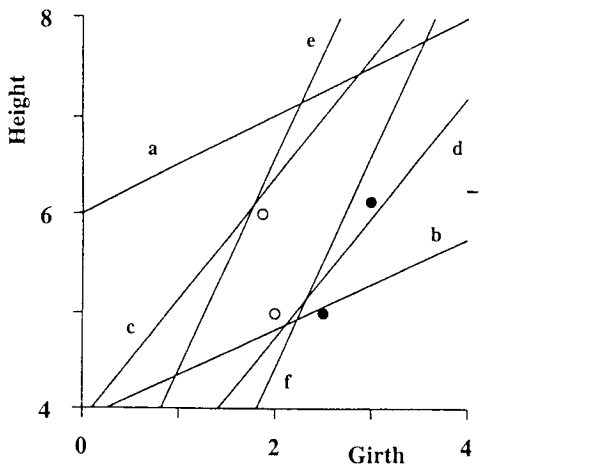

Figure 3-4. Decision boundaries generated by the IWP algorithm, starting with (a) the initial LTU, `1.0 height + −0.5 girth` $\geq$ 6.0, and ending with (f) a description that correctly classifies the training data, `1.0 height +` -2.156 `girth` $\geq$ 2.36. White circles depict positive instances and black circles indicate negative ones.

Figure 3-4 shows the behavior of this algorithm in the height/girth domain, starting with the threshold description

$$1.0 \texttt{ height} + -0.5 \texttt{ girth} \geq 6.0$$

(labeled 'a' in the figure) on the same training data as in Section 3.3.1. We assume that the algorithm first revises the threshold and then the weight for `girth`, leaving the weight for `height` unchanged, since we have one degree of freedom.

As can be seen, the first modification moves the intercept of the decision boundary but leaves the slope unmodified. Two of the five choices give the highest score 0.75, one of which produces the LTU

$$1.0 \texttt{ height} + -0.5 \texttt{ girth} \geq 3.875 \ ,$$

shown as 'b' in the figure. The second revision alters the decision boundary's slope but not its intercept. Again, two choices give the highest score 0.5 (lower than the previous round), one of which is

$$1.0 \texttt{ height} + -1.224 \texttt{ girth} \geq 3.875 \ ,$$

shown as 'c' in the figure. Future iterations alter the threshold to 2.245 and the `girth` weight to -2.156, with accuracies 0.75 and 0.5, respectively. Finally, the next threshold revision leads to

$$1.0 \texttt{ height} + -2.156 \texttt{ girth} \geq 0.236 \ ,$$

shown as 'f' in the figure. This LTU correctly classifies the entire training set, which leads the algorithm to halt.

Like many hill-climbing methods, IWP can get caught in local optima. For this reason, some versions include a strategy of randomly perturbing the entire weight vector when the fit ceases to improve, then resuming the iterative search process from this new hypothesis. Another response to this problem involves starting from many different, randomly generated starting vectors, running IWP or a related method for each, and selecting the best result.

3.4 Induction of spherical threshold units

Now let us consider the induction of spherical threshold units. The main approaches to this learning task are closely related to those for the other representational schemes we have considered in this chapter already, and we will focus on these relations in our discussion.

3.4.1 Spherical induction through state-space search

As we have seen, spherical threshold units have much in common with criteria tables, in that they extend the notion of prototypes to numeric domains. The parameters c_i in such units specify the center of a sphere, and thus constitute a prototypical (even if unobserved) instance. Similarly, one can use the threshold to specify the radius of the sphere, the continuous version of m in an m-of-n concept.

Thus, one obvious method for inducing spherical units is nearly identical to the TR algorithm for simple threshold modification that we described in Section 3.2.1. First one computes the mean value (rather than the mode) for each attribute over the set of positive training cases. The resulting means become the w_i parameters for the spherical unit, which defines the prototype of the concept. Next one gradually alters the radius of the hypersphere, moving either from the prototype outward

(specific to general) or from the largest possible radius inward (general to specific). In the first case, threshold modification halts when the hypersphere covers all positive cases; in the latter, it stops when the unit ceases to cover any negative instances.

Again, this approach assumes that all attributes are relevant, but the HCT algorithm from Section 3.2.2 also readily adapts to spherical units. The partial ordering on this hypothesis space is more complex, but there clearly exist operators for altering the radius (described in the previous paragraph) and for adding attributes to the unit or removing them.[3] One can select among alternative actions using some reasonable evaluation metric, and one can organize search with any of the heuristic methods we have seen in earlier chapters. Incremental hill-climbing versions of this scheme are also possible, provided one allows bidirectional operators for recovering from early decisions that generate poor hypotheses.

3.4.2 Spherical induction through gradient descent

Although spherical threshold units are similar to criteria tables in their use of prototypes, they are analogous to linear threshold units in their use of nonintegral weights. The basic versions of both frameworks require $a+1$ numeric weights, one for each of a attributes and one for the threshold. Taken together, these weights determine the position and extent of the region covered by the concept.

Given this analogy, it seems natural to adapt algorithms for inducing linear threshold units to handle spherical ones. For example, one could use the IWP method from Section 3.3.3, which would iteratively perturb first the radius and then the weights that determine the unit's center, continuing until no further improvement occurs. Alternatively, one could construct a variant of the PCP algorithm from Section 3.3.1, which would revise all weights together when a classification error occurs but leave them unchanged otherwise, continuing in this manner until arriving at a spherical unit that correctly labels the entire set of training instances.

One can also modify the LMS algorithm to induce spherical threshold units. In particular, suppose that we assume a spherical sigmoid function, as described in Section 3.1.3, for which we want to determine

3. A 'spherical' unit that omits some attributes actually produces a hypercylinder that extends throughout the instance space along the irrelevant dimensions.

weights that minimize the mean squared error on the training data. Given this learning task, an appropriate update method computes

$$\Delta c_i = \eta \ \delta \ \sqrt{d} \ (v_i - c_i)$$

for each center weight c_i, and calculates

$$\Delta r = \eta \ \delta$$

as the change in the weight that determines the radius. In these expressions, d is the Euclidean distance between the instance and the spherical unit's center, $\eta > 0$ is a gain factor, and $\delta = p\ (1-p)(o-p)$ is an error term based on p, the predicted value for the instance, and o, its observed value. The first expression moves the center of the hypersphere, whereas the second alters its radius.

Let us consider the behavior of this adapted LMS algorithm, with η set to 2 and momentum set to 0, on four positive and four negative training cases from the height/girth domain. The initial spherical unit, with center at $(1.0, 7.0)$ and radius 1.0, covers one positive and one negative instance, but none of the other training data, giving 2.272 as the mean squared error. The difference equations produce a revised threshold unit with center at $(2.052, 5.948)$ and radius 1.547, with 1.502 as its error; this decision boundary includes all of the instances, negative as well as positive. The third spherical unit has nearly the same center weights, $(2.054, 5.946)$, but a reduced radius of 0.996, giving 1.367 as the error. This description correctly labels all the training data, but it seems a bit off center. The fourth description, with center at $(2.021, 5.979)$ and radius 1.076, reduces the error to 1.361, after which further iterations produce only minor changes to the weights.

3.5 Summary of the chapter

In this chapter we discussed the induction of threshold concepts. When interpreted by a partial matcher, such descriptions produce decision boundaries that are quite different from those generated by logical conjunctions. We studied three types of threshold concepts – criteria tables, linear threshold units, and spherical units – each of which introduces a different representational bias into the induction process.

We found that techniques for inducing criteria tables, or m-of-n concepts, have much in common with those for conjunctions, since one can use a partial ordering on the space of concept descriptions to constrain

search. We introduced this idea with a simple algorithm that changes only the threshold on an m-of-n concept, then turned to a more sophisticated algorithm, HCT, that also includes an operator for identifying relevant features. The space of criteria tables includes concept descriptions that do not exist within the conjunctive space, and thus requires both types of operator to move through it.

We examined three algorithms for inducing linear threshold units. Most work in this area relies on gradient descent, a form of hill climbing that uses prediction errors to compute the change in weights. One version of this idea, the perceptron revision method, increases or decreases weights depending on the form of error, whereas the least mean square method computes the amount of change based on the difference between the actual and predicted values. We saw that the latter method can be invoked either incrementally or nonincrementally, but that even incremental versions are typically run over the training set repeatedly until they produce the desired output. We also saw that gradient descent is not the only approach to inducing linear threshold units. One such algorithm, iterative weight perturbation, determines the best setting for each weight in turn, producing a quite different style of search through the space of weights that describe an LTU.

Spherical threshold units combine features of both criteria tables and linear units. One can adapt the HCT algorithm directly to this representational formalism to induce descriptions with spherical or cylindrical decision boundaries of varying radius. Alternatively, one can adapt the PCP and LMS algorithms to carry out gradient descent search to find the best center and radius for a spherical unit. The most appropriate algorithm, like the most appropriate type of threshold concept, will depend largely on the domain.

Exercises

1. Using the cell notation from Figure 3-1, draw the extensional definitions for a conjunctive concept with three relevant features, a 1-of-3 concept, and a linear threshold concept than cannot be cast in either conjunctive or m-of-n terms. Show the weights and threshold for the latter concept.
2. Trace the behavior of the TRS algorithm (Section 3.2.1) on the eight instances depicted in Figure 3-1 (b), showing the evaluation score (accuracy on the training set) for each hypothesis considered during the search process. Do the same for the TRG algorithm from the same section.
3. Trace the behavior of the HCT algorithm (Section 3.2.2) on the data from Exercise 2, using a beam size of 1 and showing the hypotheses considered at each step and their evaluation score (accuracy on the training set).
4. Alter the training instances in Figure 3-3 so that the top two cases are positive and the bottom two are negative. Starting with the same initial weights as in the figure, trace the behavior of the PRM algorithm (Section 3.3.1) on one pass through the training data, showing the weights and the decision boundary after each step.
5. Trace the behavior of the LMS algorithm (Section 3.3.2) on one pass through the training data from Exercise 4, starting with the same initial weights as in that problem. Again, show the weights and the decision boundary after processing each instance.
6. Show the weights generated at each step of the IWP algorithm (Section 3.3.3) on the training data from Exercise 4. Assume that the method first revises the threshold and then the weight for `girth`, as in Figure 3-4. Run the algorithm for two iterations, showing the scores for each possible weight change on each step.
7. Adapt the PRM algorithm to work with spherical units, and trace its behavior on one pass through the training data from Exercise 1, starting with a spherical unit having its (`girth`, `height`) center at (1.0, 6.0) and 1.0 as its radius.
8. Adapt the IWP algorithm to operate on spherical units, and trace its behavior on two passes through the weights, using the training data and initial weights from Exercise 7.

Historical and bibliographical remarks

The computational study of threshold concepts goes back at least to Rosenblatt's (1958, 1962) work on perceptron learning and to Widrow and Hoff's (1960) development of the LMS procedure. Nilsson (1965) provides an excellent review of related work through the mid-1960s, whereas Duda and Hart (1973) report on slightly later developments. Minsky and Papert's (1967) analysis of linear threshold units is held to be widely responsible for discouraging work on neural networks until