Using Weighted Networks to Represent Classification Knowledge in Noisy Domains

MING TAN (TAN@G.GP.CS.CMU.EDU)
LARRY ESHELMAN (LJE@PS3.CS.CMU.EDU)

Department of Computer Science, Carnegie Mellon University,
Pittsburgh, PA 15213 U.S.A.

Abstract

Experience is not always a benign teacher. Examples often contain irrelevant features, the relevant features may be noisy, and the results may not be categorical. Most learning systems, however, assume that the teacher is benign, and in particular that the training cases are free of noise. This paper describes a system, **IWN**, which can learn classification knowledge from a relatively small number of training cases but whose performance does not rapidly deteriorate when the training cases contain noise. IWN uses a network of weighted links to represent classification knowledge. This use of a non-discrete knowledge representation enables IWN to be more robust in the face of noisy data. In this paper we describe IWN's procedure for building its weighted networks and how IWN's performance compares with other systems. A central focus will be on how several different evaluation functions for propagating values in the network affect the tradeoff between handling noisy data and handling exceptional cases.

1. Introduction

During the past few years a number of systems have been developed that can learn from a set of training cases to correctly classify a new example on the basis of its attributes. However, the performance of most of these learning systems deteriorates significantly if the training cases contain noisy data. **IWN** (**I**nduction of **W**eighted **N**etworks) was developed with noisy training cases in mind. The input to IWN is a set of attribute values and the output is a class name. IWN represents the knowledge needed to classify the attributes in a network of weighted links. The network represents the logical structure of the set of training cases, but in an analog fashion. This makes it easy to handle partial matches, and, thus, makes the system less brittle. IWN uses heuristics to structure the network and set the initial weights. The initial structure, representing the disjunction of training cases, is transformed into a structure that emphasizes the critical attributes -- those attributes which frequently occur in the training cases for a class.

Although the tree-like structures built by IWN resemble the decision trees built by Quinlan's ID3 (Quinlan, 1983), the resemblance is superficial. ID3 builds a decision tree by finding the most discriminatory attributes, one at a time in a top-down fashion. In the decision tree the leaf nodes are class names and internal nodes are attributes. IWN's network, on the other hand, consists of one tree for each class. The root node of each tree is a class name and the leaf nodes are attributes. The transformation, which emphasizes

critical attributes and generates new internal nodes (Fu, 1985), is bottom-up. IWN's representation is in many respects closer to Michalski's AQ11 (Michalski, 1980). Ignoring the weights, the trees can be viewed as syntax trees for AQ11's variable-valued logic rules. With proper weights, the representation can simulate AQ11's logic rules. However, unlike AQ11, IWN does not try to identify a set of class descriptions that will cover all training cases. Rather, it allows that some of the training cases may be noisy. Like Schlimmer's STAGGER (Schlimmer, 1986), IWN's knowledge representation uses both Boolean structures and real valued weights, but IWN combines them in one network in a way that is closer to the network representation used by connectionists (Feldman, 1982, Hinton, 1986). But unlike most connectionist learning algorithms, IWN determines the structure of the network as well as the weights of the connections. Furthermore, learning is not incremental. It must be presented with all the training cases before it can build the network.

The most important features of IWN are its tolerance for noisy data and its ability to learn from a small number of training cases. However, these features have a price -- IWN sometimes treats exceptional cases as noise. Nevertheless, we have found that this effect can often be minimized for a given domain by choosing the proper evaluation function for propagating values in IWN's network. In particular, we discuss two such evaluation functions and how they affect IWN's performance in different domains.

In the next section we describe IWN's knowledge representation and how it can be used to represent logical formulas. In section 3, we describe IWN's learning algorithm. In section 4, we describe IWN's performance in three domains that have been used for testing other learning systems. Then we characterize IWN's performance in terms of noise and exceptional cases by examining its performance in two domains specialized for this purpose. Finally, in section 5 we offer some concluding remarks.

2. IWN's Knowledge Representation

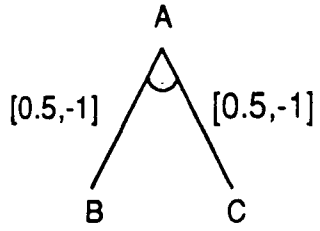
IWN represents knowledge by a directed acyclic graph with weighted links. Each link, l_{ij}, in the network connecting nodes n_i and n_j has two real valued weights, $[w^+_{ij}, w^-_{ij}]$, where $w^+_{ij} \geq 0$ and $w^-_{ij} \leq 0$. The activation level a_j of node n_j is calculated from the activation levels of the nodes that provide inputs to n_j and the weights of their connecting links:

$$a_j = \sum_i (a^s{}_i + w^s{}_{ij})$$

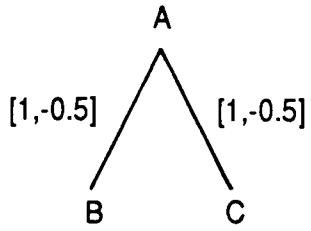
Critical to this calculation is the sign of the node's activation level, s_i (i.e., '+', '-', or '0'). It determines which weight on the link to use (and thus makes the function nonlinear). If the node is a nonleaf node, the superscript 's' on 'a_i' and 'w_{ij}' indicates that the sign of the activation level of the node and the weight on the link must correspond. Thus, if the activation level a_i is positive (s_i = '+'), then the positive weight ($w^+_{ij} \geq 0$) must be used. If a_i is 0, then neither weight on the link is used. On the other hand, if the node is a leaf node, the sign is set to correspond to the truth value of the attribute and the activation level is set to zero. (Therefore, the activation level of a nonleaf node is always a function of the weights of the activated links in its subtree.)

By interpreting a '+' value for s_i as meaning 'true' and a '-' value for s_i as meaning 'false', an IWN network can be used to represent the continuous analog of standard propositional logic:

(1) *A = B AND C*



(2) *A = B OR C*



As described in the previous paragraph, the activation level a_i of each leaf node (e.g., B or C) is set to 0, but the value of its sign s_i is set to '+' if the proposition is true and '-' if the proposition is false. Thus, in the first example, A will be true (i.e., have a positive value) if, and only if, both B and C are true. If B is true and C is false, the conjunction of B and C yields:

$$a_A = (0+0.5)+(0+-1.0) = -0.5$$

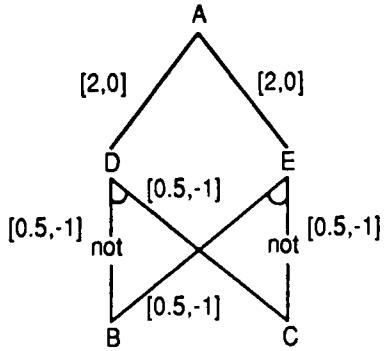
Since the activation level of A is negative, A is interpreted as being false. On the other hand, assuming the same truth values, the disjunction of B and C yields:

$$a_A = (0+1.0)+(0+-0.5) = 0.5$$

Therefore, A is interpreted as being true.

A negation operator, 'not', needs to be added to complete IWN's ability to represent logical formulas. 'Not' is attached to the links. Whenever a link is negated, the weight is used with the opposite sign from that of a_i. For example, exclusive-or could be represented by IWN as follows:

(3) *A = B XOR C*



where D represents "not B and C" and E represents "B and not C".

So far we have only discussed examples in which the sign of a node's value is either positive or negative. There is a third possibility: the sign is 0 (i.e., the node's activation level is 0 and so neither positive nor negative). IWN interprets this to mean that the truth value is

unknown. However, the behavior of the network is not quite what one would expect from this interpretation. For example, if B is true and C is unknown, then the above representation of the conjunction of B and C will result in A being true (with a low value), rather than unknown as one might expect from the standard semantics for "unknown":

$$a_A = (0+0.5)+0 = 0.5$$

Therefore, A is true.

IWN's representation was chosen to facilitate evidential reasoning and to handle noisy data. The representation can be use to represent logical relationships, but it can also handle partial matches and represent degrees of belief. However, it should be noted that the numeric activation levels are not meant to be interpreted as measures of confidence except in the crudest sense -- e.g., if node A's activation level is greater than that of another node, then the evidence for A is stronger. In the next section we describe how IWN's network is used for classification problems and how IWN acquires such networks.

3. IWN's Algorithm for Building Networks

IWN's learning algorithm constructs a network of nodes and weighted links from training cases. A training case is a set of attributes of the form "attribute-name = value" plus a class name. An attribute value can be true, false, unknown, or any other value which can be encoded as an integer with a limited range. The class name in a training case is the correct categorization in light of the given attributes. During a test run, leaf nodes corresponding to the attributes in the test case are activated and the effects are propagated through the network. If an attribute is specified as a "scalar attribute" (e.g., month or age), correspondence is determined by extrapolation (e.g., if 8 and 12 provide positive evidence, then so should 10). The class node with the highest value is interpreted as IWN's first choice. IWN's performance is evaluated by comparing IWN's choice with the class name in the test case.

IWN's learning algorithm consists of two stages: (1) *generate* the initial network from the training cases and initialize the weights; (2) *transform* the network structure and adjust the link weights. In the remainder of this section we describe the two stages of the algorithm in further detail.

3.1. Generating the Network

The initial network generated by IWN is a set of trees, one for each class. The network has three layers. The top layer consists of the root nodes which represent the classes. Each class (root) node's activation level is calculated from the "disjunction" of its training cases. The middle layer consists of nodes representing each of the training cases. The activation level of each of these nodes is calculated from the "conjunction" of its attributes. These attribute nodes form the third layer. The trees are only joined at the attribute nodes which they share.

It should be noted that "disjunction" and "conjunction" as used above are not the standard disjunction and conjunction, nor even the continuous analogs of disjunction and conjunction

discussed in the previous section. In order to get IWN's network to replicate the continuous analogs of conjunction and disjunction, the assignment of positive and negative weights must be coordinated so that the calculated value always has the right sign, i.e., '+' if true and '-' if false. IWN makes no attempt to do this. The function defined in the previous section for determining a node's activation level is used for "conjunction". However, the positive and negative weights are always assigned the same absolute values. Suppose, for example, A is the "conjunction" of B, C, and D, and the links have been assigned weights of 1, i.e., +1 for the positive weights and -1 for the negative weights. Then if two out of three of the leaf nodes (e.g., B and C) are true and the third (D) is false, A will have a numeric value of +1 (1 + 1 + -1) and, thus, will be interpreted as true. This version of "conjunction" will be referred to hereafter as *Add-AND*. Add-AND incorporates the heuristic that if the weighted "majority" of the attributes corresponding to a prior training case are present, then this constitutes a partial match, and, thus, positive evidence for the corresponding class.

We have experimented with two different versions of "disjunction" which we refer to as *Add-OR* and *Max-OR* and describe briefly below. In section 4 we discuss the strengths and weaknesses of these two variants of "disjunction".

The first version of "disjunction", Add-OR, uses the same function as Add-AND except that it only sums the positive terms. The negative terms in the summation are ignored. Thus, if all the disjuncts are false, the activation level of the node is 0, and the truth value of the node is interpreted as unknown rather than false. This incorporates the heuristic that the set of disjuncts may be incomplete. On the other hand, if at least one disjunct is true, the activation level of the node will be positive (true). The greater the number of disjuncts that are true, the greater will be the value of the "disjunction" node.

The second version of "disjunction", Max-OR, is closer to the standard version of disjunction and is used in some connectionist networks along with Add-AND (Feldman, 1982). Max-OR returns the maximum value of the set of disjuncts:

$$a_j = \max_i (a^s{}_i + w^s{}_{ij})$$

Here, if all the disjuncts are false, the activation level of the node will be negative (false) since the maximum value will be negative. If more than one disjunct are true, the activation level of the node reflects only the disjunct with the maximum value.

Once the initial structure has been generated, weights are assigned to the links. In fact, only the links attached to nodes representing attributes, i.e., the leaf nodes, are assigned nonzero weights. The initial weights of these attribute links are a function of either the number N of training cases for a class or the number M of attributes in the training cases. That is, $w^+ = 1/N$ and $w^- = -1/N$ when using Add-OR, and $w^+ = 1/M$ and $w^- = -1/M$ when using Max-OR. The weights of the interior links are all assigned values of 0. They carry the weights of the attribute links to the root nodes, subject to gating by the interior nodes.

3.2. Transforming the Network

The purpose of the transformation stage is to generalize over training cases. This is done by identifying the critical attributes for each class and moving them to a more prominent, and thus more influential, position in the network. An attribute for a class is considered to be more critical than another attribute if it occurs in more training cases for the class. Intuitively, a critical attribute is likely to be the necessary condition for the class. On the other hand, attributes which are the least critical, i.e., occur in the least number of training cases for the class, often are the product of noise.

The network is transformed by repeatedly combining two instances of the same attribute within a tree into one strong instance and by moving more critical attributes closer to the root node and less critical attributes further from the root node. Ignoring the weights, a single transformation step is simply an application of the distributive law of standard logic but with IWN's heuristic versions of "conjunction" and "disjunction". New attribute links are assigned weights which are the sum of the weights of the links they replaced, so the total value of the weights in a network remains unchanged. The weights of internal links are still always zero (see figure 3-1).[1] (As will be explained below, the resulting network is not equivalent to the initial network.) After each transformation step, the network is simplified (similar to the deletion of unnecessary parentheses in a standard logic expression). This transformation process is continued until no more instances of attributes can be combined.

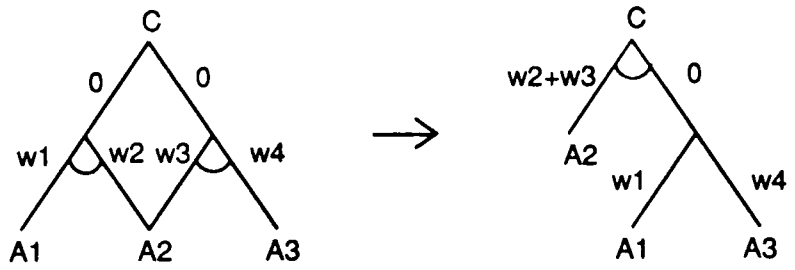


Figure 3-1: A transformation step

The order of the transformation sequence can affect the quality of the transformed network. In all the experiments presented in this paper, IWN used the rule of repeatedly transforming the pair of "conjunction" nodes with the same parent (a "disjunction" node) which share the greatest number of attributes. It can be shown that the computation time of the transformation using this selection rule is $O(M^2N^2)$ where M is the number of attributes used in the training cases and N is the number of training cases.[2]

Figure 3-2 illustrates the effect of the transformation algorithm. Class C has three training cases: (A1, A2, A3), (A2, A3, A4) and (A2, A5, A6) where A1...A6 are Boolean attributes. Before the transformation the weights of the attribute links are initialized to 0.33 and -0.33

---

[1] This, and subsequent figures, only show the positive weights.

[2] We have experimented with alternative selection rules, but they will not be discussed in this paper.

(figure 3-2 (a)). The total value of the weights has been preserved in the transformed network. However, after three transformation steps, critical attributes A2 and A3 have been moved closer to the root node C (figure 3-2 (d)). This can affect the outcome of a test case. Suppose attribute A2 is true, A3 is unknown, and the remaining attributes are false. The resulting value of C would be unknown (0) before the transformation, and true (1.0) after the transformation. (This is assuming that Add-OR is used for "disjunction". If Max-OR is used, the results would be false (-0.33) before the transformation and true (0.67) after the transformation.) The reason is that after the transformation, the influence of less critical attributes such as A1 is blocked by the failure of internal nodes such as N5 to be above "threshold" (> 0).

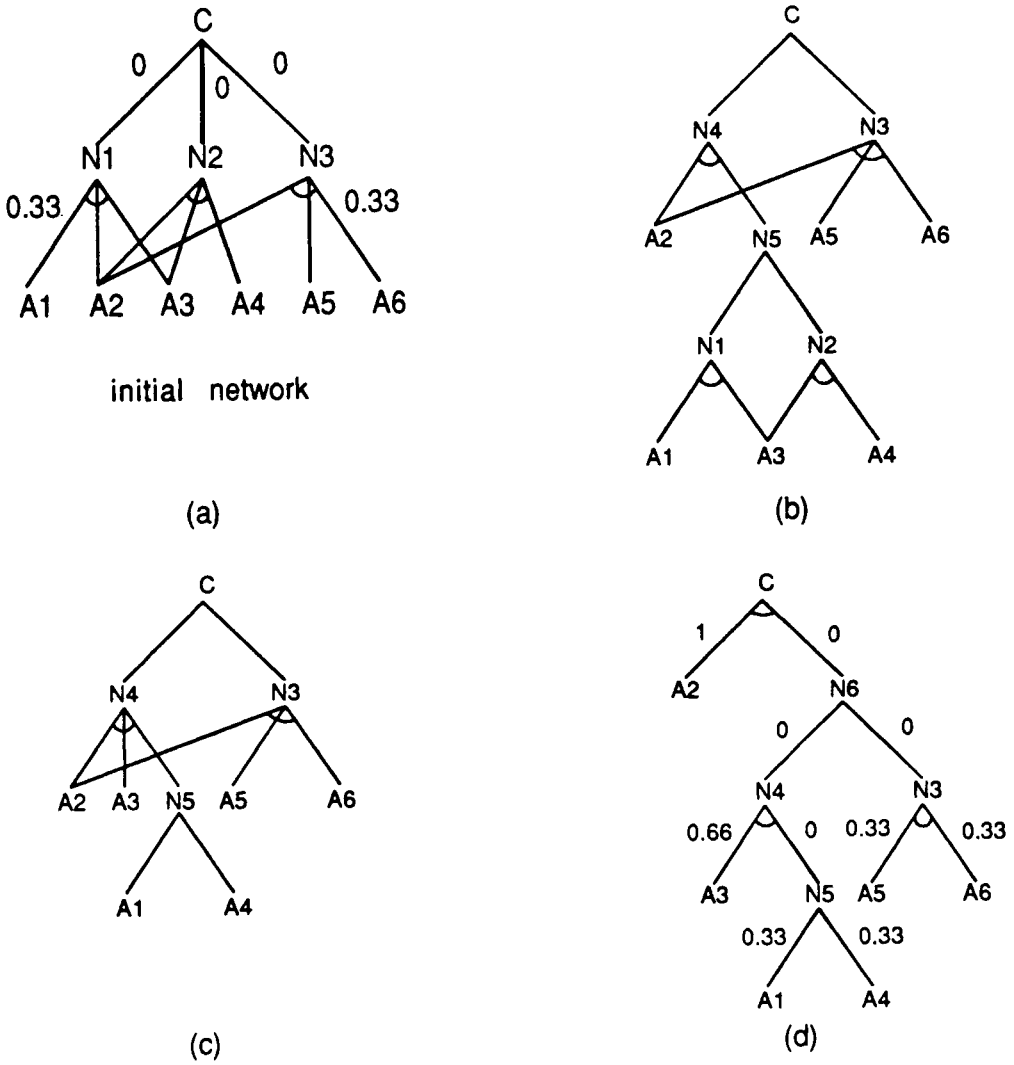


Figure 3-2: Effect of the transformation

The performance of the system is usually improved by the transformation. Because the more critical attributes now have shorter paths to a class node than the less critical attributes and their links have stronger weights, the more critical attributes can propagate their influence to the class node with less chance of obstruction, whereas propagation from

the less critical attributes is more likely to be blocked by internal nodes whose activation levels are below "threshold". The improvement is most pronounced when the training cases are small and incomplete. It should be noted that without the transformation the network, in effect, simply performs table-lookup with the ability to handle incomplete matches. In situations where the training cases are complete nothing is gained from the transformation since table-lookup will suffice. However, in such cases there is no significant loss by transforming the network. An additional effect of the transformation is that the size of the network is greatly decreased. Similar training cases are merged, and redundant information is eliminated, by the transformation of the network structure and the summation of the weights.

4. Experimental Results

We have tested IWN in three "real world" domains in which the performance of other systems is known. IWN's performance is as good, and sometimes better. Table 4-1 summarizes the performance for IWN, ID3, and AQ11/15[3] for the three domains: Soybean Diseases, Breast Cancer, and Congressional Voting. For the Soybean domain there are 290 training cases and 340 test cases, 15 diseases and 35 attributes. The Cancer domain has 286 cases of which 70 percent were chosen randomly to be used as training cases. This domain is characterized by 2 decision classes and 9 attributes. The Voting domain data consists of the votes of 163 republicans and 262 democrats in 1984 on 16 key issues. 100 votes (50/party) are randomly selected for training and the rest for testing.[4]

Table 4-1: Comparison of performance results

	SOYBEAN	CANCER	VOTING
IWN (Max-OR)	97.1 %	73.5 %	93.9 %
IWN (Add-OR)	96.6 %	73.4 %	90.8 %
ID3	92.4 %	69.3 %	94.3 %
AQ11/15	98.0 %	68.0 %	----

It will be noted from Table 4-1 that the Max-OR version of IWN performs slightly better than the Add-OR version, but the difference is not very significant. However, it turns out that once noise is introduced, the difference in performance becomes significant. Furthermore, we have found that the nature of the domain, in particular, the interdependency of attributes, affects the performance of the two versions differently. In the next two subsections we discuss the impact of noise and interdependency of attributes on the performance of IWN.

---

[3]The version of ID3 used in our experiments uses chi-square pruning, gain-ratio selection, and the majority method to handle noise (Quinlan, 1986a, Quinlan, 1986b). With a different pruning technique, the performance of ID3 in the Cancer domain can reach 72% (Michalski, 1986). AQ11's performances in the Soybean domain is reported in (Michalski, 1980) and AQ15's performance in the Cancer domain is reported in (Michalski, 1986).

[4]The Soybean and Breast Cancer data were provided by Ryszard Michalski. The Congressional Voting data was provided by U. C. Irvine and was used by Douglas Fisher for testing conceptual clustering (Fisher, 1987).

4.1. The Effect of Noise

Two different kinds of noise are considered in this paper: attribute and class noise. Given a noise rate, *attribute noise* is introduced by randomly determining for each attribute in the training cases whether it will be assigned an incorrect value; *class noise* is introduced by randomly determining for each class name in the training cases whether it will be assigned an incorrect value. Figures 4-1 and 4-2 show the performance of IWN and ID3, based on 90 training cases, when attribute and class noise, respectively, are introduced in the Soybean disease domain.[5] As can be seen from the figures, the Add-OR version of IWN performed better than the Max-OR version. This is especially noticeable when there is class noise. It will also be noticed that ID3's performance is consistently below that of the Add-OR version of IWN for both attribute noise and class noise and below that of the Max-OR version of IWN in the case of attribute noise. More importantly, the margin by which IWN performs better than ID3 is greater when the number of training cases is small. This can be seen by comparing the results in figures 4-1 and 4-2, which are based on 90 training cases, with the Soybean results in table 4-1, which are based on 290 training cases.

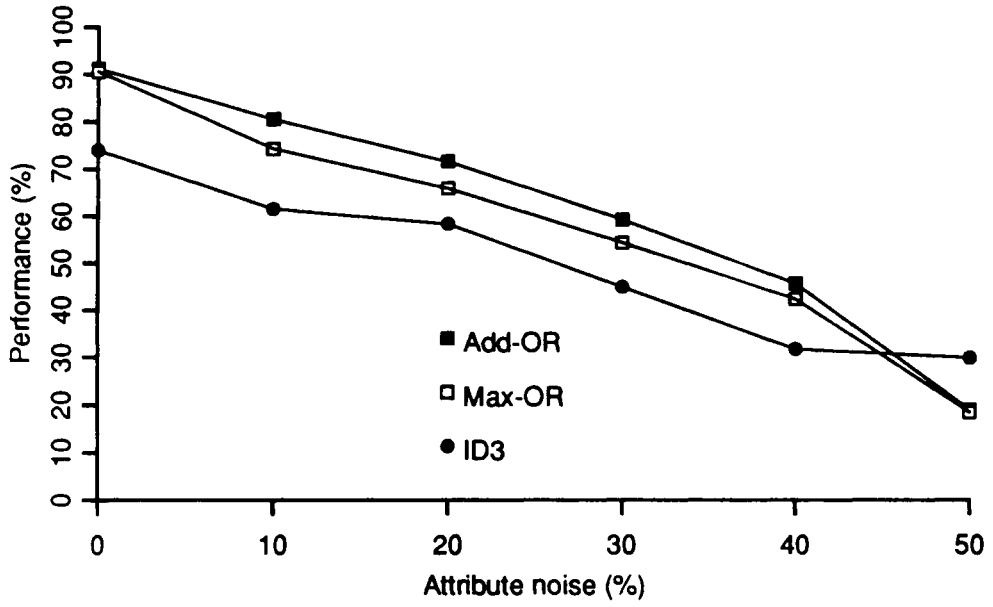


Figure 4-1: The performance in the Soybean domain with attribute noise

We have tested IWN's performance in the Breast Cancer and Congressional Voting domains under noisy conditions with similar results: The Add-OR version performs better than the Max-OR version, and IWN's performance degrades gracefully as the noise increases. This latter claim, i.e., that IWN's performance degrades gracefully under noise, is rather subjective. However, it can be seen from figures 4-1 and 4-2 that the rate of decrease in IWN's performance under noise is similar to that of ID3. In addition, we have tested IWN's performance in a domain for which the upper bound for performance using noisy data is known.

---

[5]The results shown for these experiments, and all subsequent experiments, are based on the average of 5 runs with the training cases selected randomly.

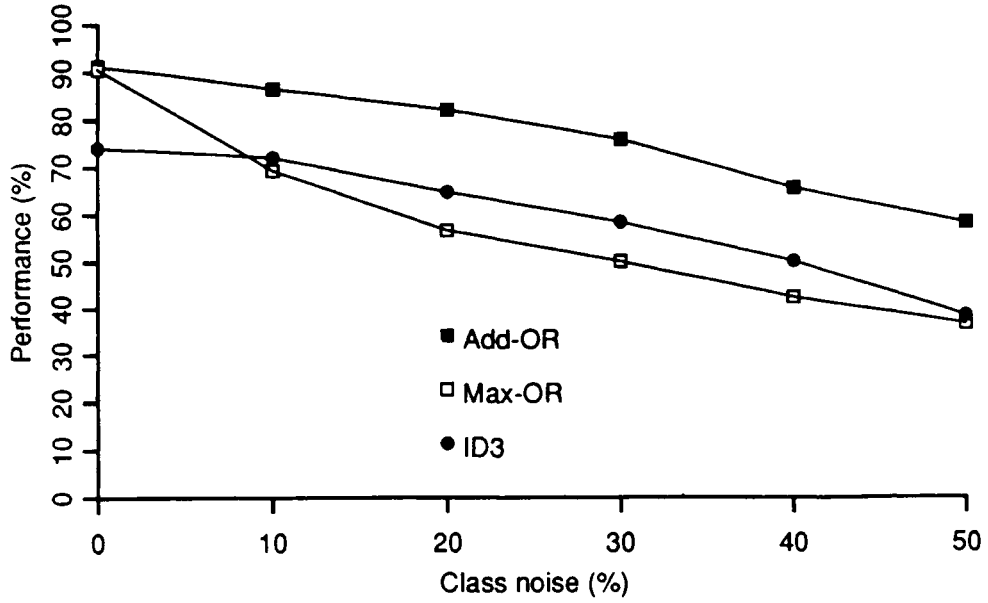


Figure 4-2: The performance in the Soybean domain with class noise

The domain is the LED domain introduced by Breiman (Breiman, 1984). Imagine a seven-element representation for a decimal digit in an LED display. Each element of a faulty display is subject to a 10% random error, i.e. the correct value is inverted with probability of 0.1. The task is to recognize the correct digit in a display with faulty elements. This requires that the learning system be able to handle noisy attributes in the training and test cases. Breiman has shown that for this noise rate, the upper bound of the performance of any system in the LED domain is 74%.

We conducted experiments with 50, 100, 200, 300, and 400 randomly generated training cases uniformly distributed among the 10 decimal digits. We used 500 test cases which were randomly generated. The performance curves are shown in figure 4-3. The performance of IWN using And-OR is 73.3% with 400 training cases (40/class). As a point of comparison, the performance of ID3 is 71.1% using 2000 training cases (Quinlan, 1986c). We also tested the Max-OR version on the LED domain using 400 training cases. Its performance is 66.3%.

4.2. The Tradeoff Between Handling Noise and Exceptional Cases

The Add-OR version of IWN consistently handles noise better than the Max-OR version. In this subsection we address the question why this is so. We conclude that there is a tradeoff between being relatively immune to noisy data and being able to handle exceptional cases. Add-OR can handle noise better, whereas Max-OR can handle exceptions better.

In order to understand why Add-OR works better than Max-OR in an inherently noisy domain, let us ignore for the moment the effect of the transformation and assume that the network built by IWN is simply three layers of nodes (the initial network). Each class (top layer) is a "disjunction" of cases, and each case (middle layer) is a "conjunction" of attributes

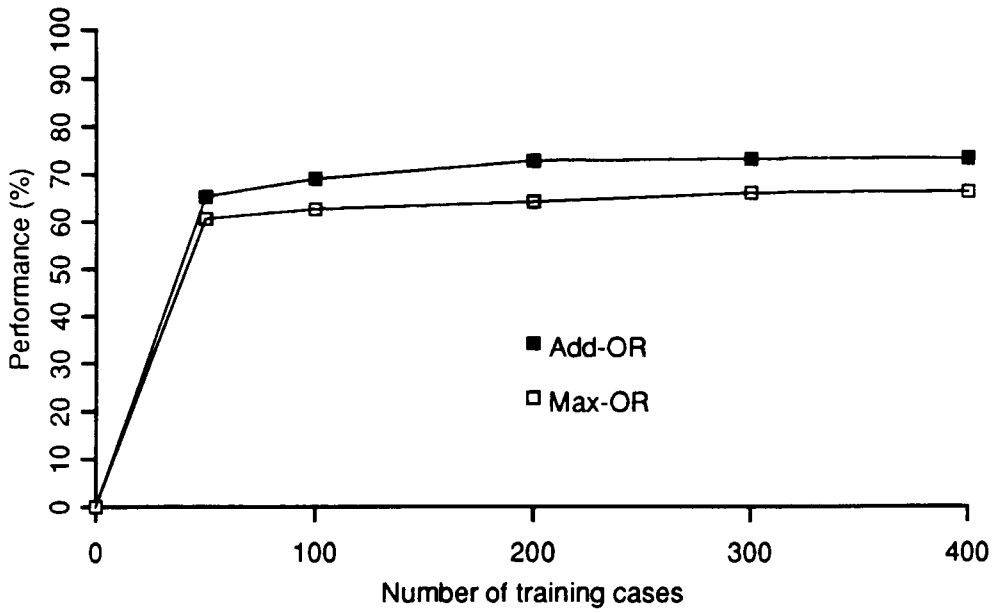


Figure 4-3: The performance of Add-OR and Max-OR in LED domain

(bottom layer). First note that the use of Add-AND for "conjunction" allows for partial matches. The closer a training case is matched, the greater its score (activation level). (However, if the score is not greater than 0, it is ignored by the next level of Add-OR nodes.) The use of Add-OR for "disjunction" means that the class (root) node with the highest score will be the one for which the test case is "closest" to the totality of the class's training cases. By initializing the weights of the links attached to the leaf nodes to 1/N, where N is the number of training cases for a class, the score of any class node is not affected by the number of training cases. It is the proportion of training cases that have high match scores which counts. In summary, the use of Add-OR in combination with Add-AND amounts to giving each case a weighted vote, the weight being determined by the degree of match. Noisy training cases, provided they are not extensive, are overwhelmed by the weighted votes of less noisy training cases.

Max-OR, on the other hand, is much more sensitive to noise. But this also means that it is more sensitive to exceptional cases. As in the case of Add-OR, cases are represented as the Add-AND "conjunction" of attributes, so IWN is still sensitive to partial matches. However, the Max-OR "disjunction" of training cases is only sensitive to the case (possibly noisy) that has the best match, i.e., the maximum score. By initializing the weights of the links attached to the leaf nodes to 1/M, where M is the number of attributes in the training case, each case has a maximum possible score of 1.0. Thus, Max-OR insures that the class for which one of its training cases best matches the test case wins.

The transformation of the network complicates the above explanation somewhat, but does not significantly alter its import. The transformation insures that the more critical attributes are closer to the root nodes than less critical attributes. The way the weights are initialized insures that two attributes of equal criticality (at the same level in the transformed network) will have the same weights on their links. The main effect of the transformation is that less

critical attributes are more likely to be "gated" since they must propagate their values through more interior nodes.

In light of the above explanation, one would expect that using Max-OR for "disjunction" would work better than using Add-OR in domains where exceptional cases cannot be ignored. The Multiplexer problem is such a domain (Wilson, 1987) where every training case is equally important. For our experiments we used the 6-multiplexer -- a logic gate with 4 input bits, 2 control bits, and 1 output bit (see figure 4-4). The 2 control bits indicate which of 4 inputs to pass through the gate to the output.

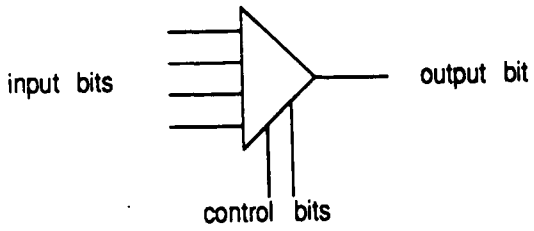


Figure 4-4: The 6-multiplexer

The tests were conducted for 8, 16, 24, 40, 80, 160, 240, and 320 randomly generated training cases divided equally between the two possible output values (0 and 1). We used 400 test cases which were randomly generated. The performance curves for the Max-OR and Add-OR versions of IWN and ID3 are shown in figure 4-5.

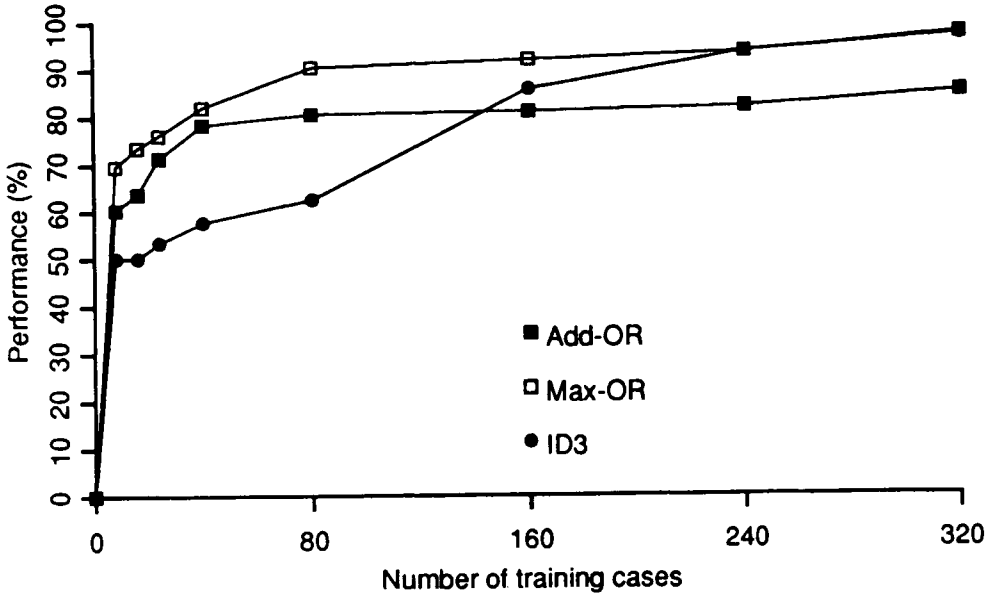


Figure 4-5: The performance of Add-OR, Max-OR and ID3 in the Multiplexer domain

As predicted, the performance of the Max-OR version was much better than the Add-OR version for this problem. Furthermore, increasing the number of training cases did not significantly improve the performance of the Add-OR version. When the number of training cases was 320 (160/class), the performance of Max-OR reached 97.6% compared with

85.1% for Add-OR.

The Multiplexer domain requires that the classifier look for a particular combination of attributes. Max-OR does this. The branch of the network representing a training case that most closely matches the test case will tend to dominate. When Add-OR is used, on the other hand, no one training case dominates. Instead, all training cases that are similar to the test case contribute. This property enables Add-OR to ignore noise, but it also causes it to ignore exceptional cases. This explains why the Max-OR version is more effective than the Add-OR version in domains where there are many exceptional cases and interdependent attributes.

Finally, it should be noted that, consistent with the results for the Soybean domain, IWN significantly out performs ID3 when the number of training cases is small, although for a large number of training cases ID3 reaches the same level of performance of the Max-OR version of IWN and is better than the Add-OR version. If the number of training cases is small and incomplete, both systems will typically represent the classification knowledge in small structures. ID3 only examines the attributes on one pathway of the decision tree, which is likely to be a proper subset of the attributes when the tree is small. This severely limits its ability to classify novel test cases. IWN, on the other hand, because it always examines all the attributes in parallel, is better able to extrapolate from the training cases.

5. Conclusion

This paper has presented a general classification system, IWN, which constructs a weighted network from training cases. IWN currently has a number of limitations. It cannot handle real valued attributes, and its method for handling attributes with a large number of discrete values can lead to a very large network. Although IWN can use either of two evaluation functions, depending upon the domain, there is no reliable way to decide *a priori* which version will work well in any real world domain that isn't well understood. One of our future projects is to try to automate the choice of the evaluation function most appropriate for a domain. In addition, we plan on examining several other possible evaluation functions. In the near future, we intend to investigate more thoroughly the effect of different types of noise on IWN's performance, the effect of pruning less critical attributes, and the effect of different ways of transforming the structure of the network. In this paper we have concentrated on how well IWN can handle noisy training cases, and, in particular, how two different implementations of "disjunction" make different tradeoffs with regard to handling noise and exceptional cases. The Add-OR version of IWN is more immune to noise. The Max-OR version, on the other hand, out performs the Add-OR version when there are a significant number of exceptional cases. Neither version requires a large number of training cases to reach a reasonable level of performance.

Acknowledgements

We would like to thank Tom Mitchell, Jeff Schlimmer, and Prasad Tadepalli who read an earlier draft of this paper and made many useful suggestions.

This research effort is supported in part by the National Aeronautics and Space

Administration under grant # NCC 2-463 and in part by Digital Equipment Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NASA or DEC.

References

Breiman, L., Fredman, L. H., Olshen, R. A. and Stone, C. J. *Classification and Regression Trees.* Belmont, California:Wadsworth International Group, 1984.

Feldman, J. A. and Ballard, D. H. Connectionist Models and Their Properties. *Cognitive Science*, 1982, *6*, 205-254.

Fisher, D. H. Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, 1987, *2*(2), 139-172.

Fu, L. M. and Buchanan, B. G. *Learning Intermediate Concepts in Constructing a Hierarchical Knowledge Base*, pages 502-507. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, 1985.

Hinton, G. E., McClelland, J. L. and Rumelhart, D. E. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. In Rumelhart, D. E., McClelland J. L. and the PDP research group (Ed.), *Distributed Representations,* Cambridge, MA: Bradford Books, 1986.

Michalski, R. S. and Chilausky, R. L. Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methodes of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. *International Journal of Policy Analysis and Information Systems*, 1980, *4*(2), 125-161.

Michalski, R. S., Mozetic, I., Hong, J. and Lavrac, N. *The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains*, pages 1041-1045. Proceedings AAAI-86, Philadelphia, PA, 1986.

Quinlan, J. R. Learning Efficient Classification Procedures and their Application to Chess End-Games. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), *Machine Learning,* Palo Alto, CA: Tioga, 1983.

Quinlan, J. R. Induction of Decision Trees. *Machine Learning*, 1986, *1*(1), 81-106.

Quinlan, J. R. The Effect of Noise on Concept Learning. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.), *Machine Learning,* Los Altos, CA: Morgan Kaufmann, 1986.

Quinlan, J. R. *Simplifying Decision Tress*, pages 36/0-36/15. Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986.

Schlimmer, J. C. and Granger, Jr. R. *Beyond incremental processing: Tracking concept drift*, pages 502-507. Proceedings AAAI-86, Philadelphia, PA, 1986.

Wilson, Stewart L. Classifier Systems and the Animat Problem. *Machine Learning*, 1987, *2*, 199-288.