Measuring Concept Change

Carla E. Brodley and Edwina L. Rissland
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003 USA
brodley@cs.umass.edu, rissland@cs.umass.edu

Abstract

In this paper we present a general approach to learning in domains in which concepts change over time. We introduce a new metric, concept instability, that detects whether a concept is changing by examining changes in the concept representation formed by a supervised incremental learning algorithm. When a change is detected, we form the hypothesis that the change is due to a shift in the underlying concept. If the hypothesis is accepted, then a learning method that adapts the concept representation to the new concept is applied. We illustrate the approach in conjunction with an incremental decision tree algorithm. The approach is applicable to a wide class of learning systems because it is designed to be used in conjunction with any incremental learning algorithm that adjusts its concept representation in response to each observed positive or negative example of the target concept.

1 Introduction

The ability to incorporate new information and adapt to a changing world is necessary for intelligent behavior. Many concepts change over time and intelligent agents must be able to adapt their concept representations to reflect these changes. For example, consider society's concept of a desirable car. Prior to the energy crisis in 1973, a desirable car was one that had a smooth ride and a roomy interior, whereas in the 90's a desirable car is one that gets high gas mileage. This type of change has been called *concept drift* (Schlimmer, 1987).

Although incremental supervised learning algorithms adjust the concept representation in response to each observed example of the concept, many cannot detect or adapt the concept representation when the concept drifts. A concept is drifting when the stream of examples begins to represent a new or slightly altered concept definition. In this paper we present a general approach to detecting and verifying when a concept is changing over time. In addition, we illustrate how the learning strategy can be altered in order to adapt the concept representation in response to change. The approach can be used in conjunction with any supervised incremental learning algorithm that has the following properties: any part of the current concept representation is subject to change and the representation defines at least a partial order on the importance of the attributes, which describe each example, to recognizing members of the concept.

In this paper, we first describe the different ways in which a concept may change and how change is manifested in a concept representation. We then describe a new approach for detecting, verifying and adapting to concept change. Specifically, Section 2 describes a taxonomy of change. Section 3 introduces the idea of *concept stabil-*

ity as a heuristic metric for measuring concept change over time. In Sections 4 and 5 we describe a method for detecting and verifying concept change using this metric. We apply the method in conjunction with an incremental decision tree algorithm ID5R (Utgoff, 1989). In Section 6 we demonstrate how the method works in response to varying amounts of concept drift and finally, in Section 7 we discuss how our approach relates to other approaches for handling concept drift.

2 Concept Change

There are many ways in which a concept can change over time: a concept can become more general or more specific, the relevance of the attributes describing the concept can change, or the correlation between the value of an attribute and the concept definition can change. In this section we present a (partial) taxonomy of change and we describe how changes are manifested in a variety of concept representations.

2.1 Taxonomy of Concept Changes

We describe a taxonomy of changes that may occur for concepts that can be represented in Disjunctive Normal Form (DNF). To illustrate the types of change, we use five Boolean Attributes, A, B, C, D and E to describe a concept and we use a decision tree to represent the concept. Each example is labeled positive (+) or negative (-) to indicate membership or non-membership in the concept respectively. In a decision tree a node is either a leaf containing the name of a class or a decision node containing a test. For each possible outcome of the test there is a branch to a decision tree or a leaf. To classify an instance, one starts at the root of the tree and follows the branch indicated by the outcome of each test until a leaf node is reached. The name of the class at the leaf is the resulting classification. We describe six types of change and depict each type by drawing the decision trees before and after the change. Note that for several of the examples, not all five attributes will be important to classification and therefore will not appear in the tree. The types of change are:

1. **Adding a new disjunctive condition generalizes the concept.** For example, in Figure 1 the concept changes from $AB \vee C$ on the right to $AB \vee C \vee D$ on the left. Adding a disjunctive clause to the concept causes one of more nodes to be added to a decision tree.

2. **Removing a disjunctive condition makes the concept more specific.** For example, in Figure 1 the concept changes from $AB \vee C \vee D$ on the right to $AB \vee C$ on the left. Removing a disjunctive clause from the concept causes one or more nodes to be removed from the decision tree.

3. **Adding an additional exception makes the concept more specific.** For example, in Figure 2 the concept changes from $AB \vee CD$ on the left to $AB \vee CDE$ on the right. Adding a conjunction to one of the clauses adds one or more nodes to the tree.

4. **Deleting an exception generalizes the concept.** For example, in Figure 2 the concept changes from $AB \vee CDE$ on the right to $AB \vee CD$ on the left. Removing a condition from a clause removes one or more nodes from the tree.

5. **The importance of an attribute changes.** For example, in Figure 3 the concept changes from $AB \vee C$ on the left to $AD \vee C$ on the right. The new concept has the same number of clauses, but the change in importance causes nodes to be both added and deleted from the tree.

6. **An attribute's value with respect to the concept has been inverted.** For example, in Figure 4 the concept changes from $AB \vee C$ on the left to $AB \vee \neg C$ on the

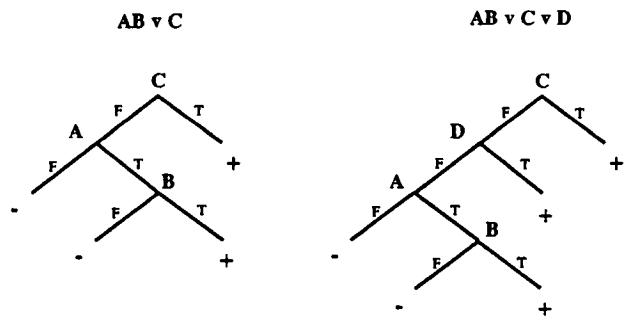

Figure 1: Concepts $AB \vee C$ and $AB \vee C \vee D$

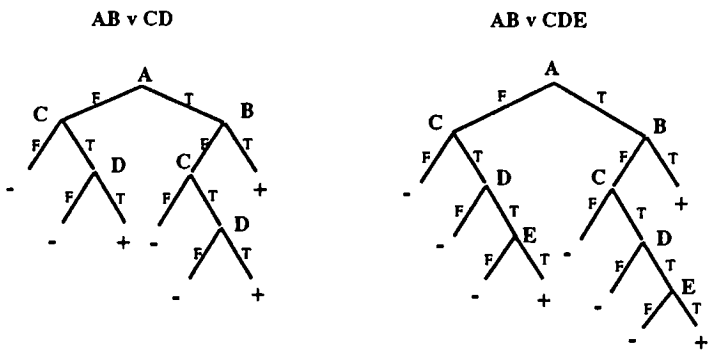

Figure 2: Concepts $AB \vee CD$ and $AB \vee CDE$

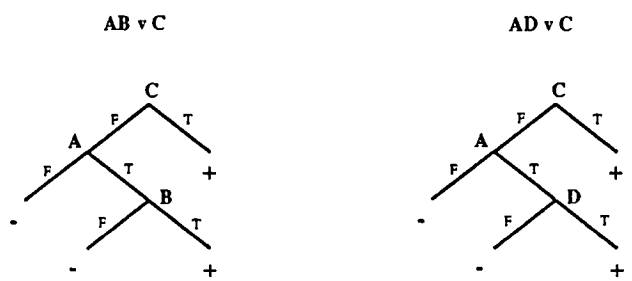

Figure 3: Concepts $AB \vee C$ and $AD \vee C$

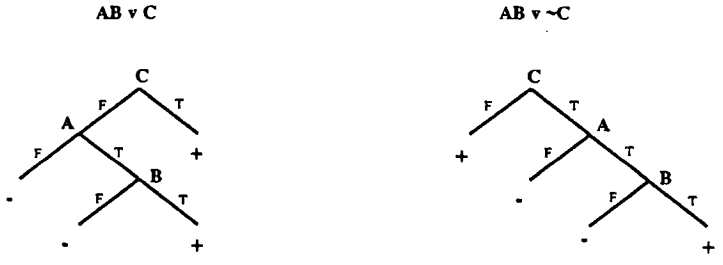

Figure 4: Concepts $AB \vee C$ and $AB \vee \neg C$

right. Negation of an attribute in a clause causes the subtrees of the corresponding test to switch branches.

2.2 Computational Manifestations of Concept Change

Given a set of positive and negative examples of a concept, each described by a set of attributes, the goal of a supervised learning algorithm is to induce a generalization of the examples that can accurately distinguish objects as members or nonmembers of the concept. Typically, the values of some attributes are more indicative of concept membership than others. For example, if the concept to be learned is "chair" and each example is described by the attributes *number of legs* and *color*, then clearly, the value of *number of legs* is more predictive of whether an object is a chair.

Most inductive learning algorithms attempt to learn which attributes are most predictive of the concept. For example, ID5R generates a decision tree which defines a partial ordering of the importance of the attributes; one attribute is judged more important than another if it is closer to the root of the decision tree. (The depth of nodes in a tree does not define a total ordering on the attributes because two attributes at the same depth in the tree have the same importance.) As a second example, in a concept hierarchy formed by COBWEB (Fisher, 1987), attributes tested higher in the hierarchy are more important for categorization than those tested lower in the hierarchy. In the case-based systems, HYPO (Ashley, 1990) and CABARET (Rissland & Skalak, 1991), cases are ordered by how "on point" they are for argumentative purposes. The partial order is represented in the "claim lattice" in which cases nearer the root of the lattice are more on point than those lower down; cases appearing at the same level are considered equally on point.

3 Recognizing When a Concept is Changing

The ordering of the attributes by their importance for predicting concept membership leads directly to the idea of *concept stability*. We define concept stability to be a measure of how much the ordering changes in relation to the number of observed examples. If the concept is *not* changing, then as the number of observed examples increases the ordering stabilizes and therefore, the concept stability increases or equivalently, the concept instability decreases. Note that initially the ordering will fluctuate because when few examples have been observed it is difficult to ascertain correctly which attributes are most important. In this section, we introduce a measure for analyzing change and we describe how to compute concept instability over time.

3.1 Measuring Structural Change

By examining the structure of a concept representation we can measure the change in importance of the attributes used in predicting the concept, for instance as reflected by the structural changes in a decision tree representation. A structural measure of change can be implemented with any concept representation that defines a partial order of the attributes' importance to predicting class membership. For the purpose of clarity, in the following discussion we illustrate the measure using a decision tree representation.

In a decision tree, each time a node is created, deleted, or the attribute at the node is replaced by a different one, the concept representation changes, as discussed in Section 2.1. The importance of the change is determined by its depth in the decision tree; since attributes tested further away from the root of the tree are less important to classification, we want to pay less attention to changes further away from the root. Our structural measure of change achieves this by weighting a change at depth i in the tree by $(n-i)/n$, where n the maximum possible depth of the decision tree and is equal to the number of attributes plus one. After each new example is observed, to measure the change in the tree representation we can compute the sum of the changes at each depth in the tree. Specifically, if K nodes at depth i in the tree change, then the change at depth i is computed as: $change_i = K \times (n-i)/n$. The total change for the tree is then:

$$Tchange = \sum_{i=1}^{n} change_i$$

Table 1 shows how to compute the measure for each of the examples of change presented in Section 2.1. In each example of change there were five attributes which described the instances. Therefore, the maximum depth of a decision tree is 6; the depth of the root of the tree is 1, the maximum depth of a *test* node in the tree is 5, and the maximum depth of any *leaf* in the tree is 6. In Table 1, columns 1-5 show the measure for each level in the tree and the last column, "Total", shows the sum of all changes. For example, if a disjunctive clause is added to the concept (see Figure 1), then one node changes at level 2 and $change_2 = 1 * (6-2)/6$. At level 3 a leaf changes to a test and a test changes to a leaf, $change_3 = 2 * (6-3)/6$, at level 4 two leaves are removed, a leaf is added and a test is added, $change_4 = 4 * (6-4)/6$, and at level 5 two leaves are added, $change_5 = 2 * (6-5)/6$. Total change for the decision tree is the sum of the $change_i$ for $i = 1, .., 6$, and in this case, is equal to 2.83.

3.2 Concept Instability

Concept instability is a measure of change over time. To determine if a concept representation is becoming stable one looks for *trend* in the measure of change. A stable concept is one where change decreases over time. To compute the amount of change over time we update the statistic:

$$instability = \frac{1}{E} \sum_{j=1}^{E} Tchange_j$$

Where $Tchange_j$ is a measure of change between the tree based on $j-1$ instances and the tree after the j^{th} instance in added, and E is the total number of instances observed thus far. Because the number of changes in the tree is normalized by the number of observed examples, E, as E increases the instability will decrease if the concept is not changing. The actual value of *instability* at any one point in time may not in itself yield any information about the concept stability since it only reflects change over one "time" step.

4 Detecting Concept Change

In this section we describe how to use concept instability to detect concept change. If a concept becomes unstable, there are two possible explanations: the partial ordering of the attributes (eg., the structure of the tree) has changed because new examples were observed from a different part of the example space than observed previously or because the concept is drifting. De-

Table 1: A Structural Measure of Change

Type of Change	1	2	3	4	5	Total
Add a Clause	0	1*(6-2)/6	2*(6-3)/6	4*(6-4)/6	2*(6-5)/6	2.83
Delete a Clause	0	1*(6-2)/6	2*(6-3)/6	4*(6-4)/6	2*(6-5)/6	2.83
Add Exception	0	0	0	1*(6-4)/6	3*(6-5)/6	0.83
Delete Exception	0	0	0	1*(6-4)/6	3*(6-5)/6	0.83
Relevance Change	0	0	1*(6-3)/6	4*(6-4)/6	0	1.83
Clause Negated	0	2*(6-2)/6	3*(6-3)/6	4*(6-4)/6	0	3.50

pending on the cause of the increase in instability, different learning strategies should be employed. If the increase is due to more information, then we can continue to train the existing concept representation. On the other hand, if the concept is drifting, then the effect of examples observed in the past should be removed from the concept representation. If the drift is not radical then removing all examples prior to the drift would throw away all previous effort (Schlimmer, 1987). Thus, we retain the the current concept representation and as new examples are observed, we discount the effect of examples observed further in the past. Rather than choose erroneously between the two explanations, we test the *hypothesis* that a increase in instability is caused by drift. In Section 5 we describe a method for verifying concept drift by testing this hypothesis.

To detect possible concept drift we examine the instability of the concept representation. To detect whether the instability is increasing or decreasing we use a statistical measure of trend, Kendell's τ test (Kendall, 1962). The τ test is a nonparametric test of correlation between two variables, x and y. Given n data points, (x_i, y_i), the τ statistic measures the degree of positive or negative correlation between x and y. In our implementation, x_i is the value of the instability metric, defined in Section 3.2, after example i has been observed and y_i counts how many examples have been observed ($y_i = i$). The τ statistic is approximately normally distributed. Values close to $+1$ indicate a high positive correlation and values close to -1 indicate a high negative correlation. To measure whether the concept instability is increasing or decreasing we compute the τ statistic for a window of n data points, where n equals the number of attributes.

To detect that the concept may be drifting we must first observe that it has stabilized. We define a concept as stable if the probability that there is a decreasing trend in the concept instability measure is higher than 95% as computed by the τ test. If we observe an increasing trend in the instability after it has stabilized, then we form the hypothesis that the concept is drifting.

5 Verifying Concept Drift

In this section we describe how we can verify concept drift when using an incremental supervised learning algorithm. Although we illustrate the approach in conjunction with an incremental decision tree algorithm, ID5R (Utgoff, 1989), the approach can be applied to any algorithm that is able to remove the effect of instances observed in the past. ID5R retains all observed examples in the tree so that at each test node in the tree enough information is kept to calculate the information theoretic measure (used to select which attribute to test at each node) of each attribute not tested above this node in the tree. The remainder of each example is stored in the appropriate leaf of the tree. During training, if an attribute tested at a node becomes less predictive

than another attribute tested closer to the root of the tree (as measured by the information-gain ratio metric), then ID5R restructures the tree so that the desired test attribute is pulled up to the root of the tree or subtree. This method of restructuring the tree permits ID5R to recalculate the data necessary for the information theoretic measure without re-examining every attribute in training examples.

When there is an increase in concept instability it is not obvious whether the increase is caused by concept drift or due to observing new examples. Rather than choose erroneously, we form and test the hypothesis that the increase is due to drift. To this end, we begin by making a copy of the current ID5R tree, called the *new tree*. We then continue to train the original tree using the ID5R algorithm. We train the new tree using a different procedure; for each new example observed we remove the example observed furthest in the past, which is still part of the new tree. Over time, this removes the influence of old examples from the new tree and may result in ID5R restructuring the tree. This procedure requires that there be a way to determine the age of each instance. We achieve this by marking each instance with the time that it arrives.

To test the drift hypothesis, we continue to train both the old and the new tree until either one or both trees have an increasing concept stability, as measured by the τ test. If the old tree becomes stable and the new tree does not, then we reject the hypothesis. In this case, the instability is attributable to not having observed enough examples to converge to a stable concept description. The case where only the new tree becomes stable will never arise. Even if a concept is drifting the old tree will eventually become stable (albeit with an inaccurate concept description) and we want to wait until that point to make a knowledgeable choice. If both trees are stable, then we examine the accuracy of each tree for the examples observed *after* the possible drift was detected. If the accuracy of the new tree is higher, then we accept the hypothesis that the concept is drifting. If the accuracy of the new tree is the same or lower than the accuracy of the old tree, then we reject the hypothesis. If the concept is not drifting, then either the old tree will be more accurate (due to the fact that it is based on more examples) or there will be no difference in the accuracies of the two trees. However, if the concept is drifting, then the new tree will obtain a higher accuracy because some of examples of the concept observed prior to the drift have been removed.

If the drift hypothesis is accepted, then we continue removing examples observed prior to the point when drift was detected. Because we may verify the hypothesis before the effect of all the examples of the old concept are removed, we need to continue to remove the old examples. Once we reach the point where further removal causes the tree to become incorrect for examples observed after the drift, we revert to ID5R's training method, retaining all remaining and new examples in the tree. Once the hypothesis is accepted or rejected, we continue training with the chosen tree. Note that when we resume training with one tree, the concept drift detection mechanism is turned on again.

6 Illustration

To understand how the stability metric can be used to detect concept drift, we show results for an artificial domain in which the concept is changing. The concept changes from $AB \vee DE \vee CG$ to $AB \vee CDE$. We created 256 unique examples for each concept (there are 8 Boolean attributes and 2 classes). We presented ID5R with examples of the concept $AB \vee DE \vee CG$ in random order. We then presented examples of the concept $AB \vee CDE$ in random order. A graph of the concept instability measure,is shown in Figure 5. The y-axis shows the instability and the x-axis shows the number of instances observed thus far. The instability measure is cumulative:

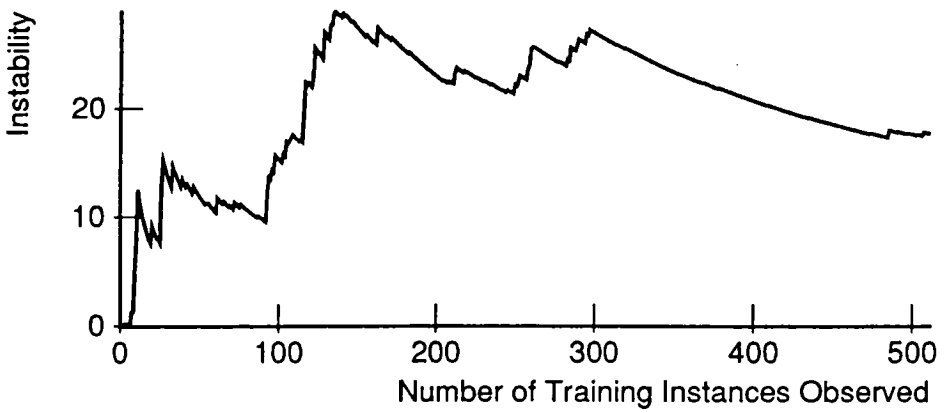

Figure 5: Instability for the concept: $AB \vee DE \vee CG \Longrightarrow AB \vee CDE$

a positive slope for some number of instances indicates that the tree is undergoing structural changes, whereas a negative slope indicates that the decision tree representation is stabilizing.

The trend measure detects stability after 71 instances have been observed. After the 100'th instance is observed the tree representation begins to change again. After 112 examples have been observed the system detects that the concept may be drifting. At this point it makes a copy of the current concept representation, and begins to train the two trees. After 163 examples have been observed, both the new tree and the old tree have stabilized. At this point the system rejects the hypothesis that the concept is drifting, because the accuracy of the old tree for the examples observed after the drift is higher than that of the new tree. The system then continues to train only the original tree (the original tree is based on all 163 observed examples).

The small rises in instability at 156 instances and 220 instances are not significant enough to cause the drift detection mechanism to signal a possible change. After 271 examples have been observed the system again detects a possible drift. After 322 examples have been observed both trees are stable, but the accuracy of the new tree is higher, verifying the drift hypothesis. At this point the system discards the original tree and continues training the new tree. It continues to train the tree until all 512 examples have been observed. The accuracy of the final concept description was 100% for the examples observed after a possible drift was detected, whereas without drift detection, the tree found by ID5R had an accuracy of 76%.

Table 2 illustrates how the method works with varying amount of concept change. For each change, the table reports the percentage of change (the number of instances that differ from the original concept to the new concept). For each change, we ran the system ten times, using a different random order for the examples before and after the drift. The number of times the system was able to correctly verify concept drift is shown in the column labeled "Correct". The table reports the average number of instances observed when drift was detected and the average number of instances observed when drift was verified. The last two columns report the accuracy for the instances of the new concept for ID5R and for the concept drift version, CD ID5R.

Table 2: Varying Degrees of Change from $AB \vee CDE$

New Concept	% Change	Correct	Detect	Verify	ID5R	CD ID5R
$AB \vee CDEF$	4.7	7	349	387	95.3	99.9
$AB \vee CGH$	14.1	9	309	338	85.9	99.5
$BH \vee CGH$	21.9	10	262	302	78.1	99.8
$AB \vee CDE \vee F$	32.8	10	279	309	67.2	98.8
$A \vee C \vee D$	53.1	10	253	368	46.9	99.1

7 Discussion and Conclusion

Incremental learning systems have the ability to incorporate new information and some have the additional ability to forget old information. However only a few have the ability to recognize when old information has become incorrect and therefore should be forgotten. This ability allows a learning system to handle concept drift. One such system is STAGGER, which has the ability to backtrack and therefore can remove elements (features) that have become less predictive of the concept to be learned (Schlimmer, 1987). In addition, STAGGER's weight learning method permits the system to change each attribute's relevance. Moore's system removes old exemplars that have a large discrepancy between the value of a robot manipulator and the value that the system predicts (Moore, 1990). The system has the ability to adapt to changing behavior in a robotic system by comparing the observed behavior (the proximal acceleration) to the predicted behavior. The methods for handling concept drift in both STAGGER and Moore's system depend on the characteristics of the particular learning method. In contrast, the method described in this paper is a general method for detecting and handling concept drift.

Another approach to handling concept drift is to check if an new example is inconsistent with an example observed in the past (Utgoff, 1989; Aha, 1990). When this situation arises one could simply remove the effect of the example observed further in the past. Although this approach will work if there is no noise in the instances, in noisy domains, one has no basis for believing that the new example is better than the old example. In such situations, using the method presented in this paper would allow the system to check whether the inconsistency arises because of noise or concept drift. In addition, the presented method can adapt to a drift in the concept that does not cause any inconsistencies in the examples, which may happen when many of the attributes are numeric.

In this paper, we presented a measure of concept change and we illustrated how to use this measure to detect concept drift. We described an approach for verifying concept drift and how to modify the learning strategy to track concepts that change over time. The approach can be used in conjunction with any incremental learning algorithm that has the following three properties: its representation of the concept permits at least a partial ordering of the attributes by their importance to classification; at any stage during learning, the system can change this ordering in response to new examples; and the influence of any example can be removed from the concept representation.

Acknowledgments

This material is based upon work supported by the National Aeronautics and Space Administration under Grant No. NCC 2-658, by the Office of Naval Research through a University Research Initiative Program, under contract num-

ber N00014-86-K-0764, and the Air Force Office of Sponsored Research under contract 90-0359. We would like to thank Paul Utgoff for providing us with a copy of ID5R and Jeffery Clouse for his helpful comments.

References

Aha, David W. (1990). *A study of instance-based algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations.* Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine, CA.

Ashley, K. D. (1990). *Modelling legal argument: Reasoning with cases and hypotheticals.* Cambridge, MA: M.I.T. Press.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139-172.

Kendall, M. G. (1962). *Rank correlation methods.* New York: Hafner Publishing Company, Inc.

Moore, A. W. (1990). Aquisition of dynamic control knowledge for a robot manipulator. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 244-252). Austin, TX: Morgan Kaufmann.

Rissland, E.L., & Skalak, D. B. (1991). CABARET: Rule interpretation in a hybrid architecture. *Internation Journal of Manmachine Studies, 34*, 839-887.

Schlimmer, J. C. (1987). *Concept acquisition through representational adjustment.* Doctoral dissertation, University of California, Irvine.

Utgoff, P. E. (1989). Incremental induction of decision trees. *Machine Learning, 4*, 161-186.