# Case-Based Classification: A New Approach

Arthur M. Farley

CIS-TR-91-16
July 17, 1991

Univerisity of Oregon
Computer and Information Science Department

# Case-Based Classification: A New Approach

Arthur M. Farley
Computer and Information Science
University of Oregon
Eugene, OR 97403
art@cs.uoregon.edu

## Abstract

In this paper we introduce a new approach to case-based classification, i.e., classification based solely on previously classsified cases. Our approach is characterized by on-the-fly generation of concepts, done at the time of new instance presentation. Best concepts are formed that would classify the new instance as being from each of a set of competing classes. Concepts are represented as constraints that cases must satisfy. Concepts are rated according to an evaluation fuinction that captures aspects of necessity and sufficiency. We define a case representation and case-space structure that make the process reasonably efficient.

Our approach has the advantage that it is not vulnerable to the problem of concept drift, which can arise when concepts formed in the past do not reflect recent experience. Since concepts are formed at classification time, we can justify classification decisions in terms of constraints that a new instance satisfies, something most, purely case-based approaches can not do.

**Topics:** Machine Learning (classification, case-based reasoning)

## Introduction

In this paper, we introduce a new, case-based approach to classification. By *classification*, we mean the assignment of a new input instance to one of a pre-specified set of classes. A *class* refers to a (named) subset of possible input instances; input instances are represented in a given input language, which serves to determine the range of possible inputs. A class may be represented intensionally as a *concept* according to some description language [Rendell, 1986]. A concept correctly represents a class to the extent that members of the class satisfy the concept's description and non-members do not.

The central idea of our approach is that when a new input instance is presented for classification, the system first forms the best concepts (i.e., classification rules) that it can for each of the competing classes. Each competing concept is formulated so as to classify the new input instance as a member of its associated class. These rules are created at the time of classificiation by referring directly to the exemplars of competing classes presently in memory. The system never attempts to determine rules capable of completely describing a class; it only determines a best concept that would include the current input instance within the class. The system determines a classification for the new input instance based upon an evaluative comparison of these competing concepts.

From one perspective, our approach to classification can be characterized as case-based. By case-based, we mean an approach whereby classification decisions are made on the basis of previously presented input instances (i.e., cases) and their (presumably correct) prior classifications. In other words, the memory used as basis for new classification decisions is that of past exemplars of different classes; no explicit concept is maintained in memory.

Our research was instigated by informal consideration of the decisionmaking process in law. In most case domains, the legal decision problem can be viewed as one of assigning a new case to one of several (often two) possible outcome classes (e.g., guilty or not guilty, contract or no contract). As noted by Gardner [Gardner, 1987], based on the writings of other legal scholars [e.g., Hart, 1961; Dworkin, 1977], consideration of case precedence is an integral part of the legal argumentation and decisionmaking process.

Legal scholars are quick to eschew the notion that the legal decisionmaking process can be modeled as being based upon a set of a priori rules that determine outcomes in a given legal domain. Yet in written court opinions, there is typically a discussion of the "rule of law" that has been applied in a case. Legal argumentation, especially in appellate court, often focuses on which rule of law is the correct one to be applied or whether a rule has been applied correctly.

Where do these rules of law come from if they are not just sitting there, agreed upon by all lawyers and judges, ready to be used for deciding new cases? It would appear that a fundamental task for lawyers is the formulation of new legal rules,

based on prior cases, that are most favorable to their particular, desired outcome. Even in areas of statutory law, rules that bind particulars of a given case to general preconditions of statutes must be created. Of course, the "other side" in any case is doing the same thing with respect to its desired outcome. These rules serve to focus the issues that are debated in legal arguments that ensue. Legal argumentation focuses on establishing (or denying) the credibility of proposed classification rules. Eventually, the judge must rule that one side's rule(s) are better than the other side's rule(s), thereby deciding for one side over the other.

Our approach to classificiation reflects this precedence-based, "rule-generation-on-the-fly" outlook toward classification decisionmaking. Competing rules are generated at the time of input instance presentation and then ranked according to some evaluation function; the rule with the best evaluation is the one selected, and the associated classification decision is made.

In addition to capturing a central feature of legal reasoning, our approach exhibits two useful properties:

(1) Since the sole form of knowledge is the set of previously classified cases, the issue of how to deal with concept drift [Schlimmer and Granger, 1988] due to conflicting experience does not arise.

All concept rules used during any current classification process depend only upon extant case memory at the time of classificiation. Thus, the concepts generated would automatically change with time, if experience, in the form of new cases added to memory, were to warrant that change.

(2) While our approach is essentially case-based in nature, a concept, represented in the form of constraints justifying the classification decision, is generated as part of the process.

This new concept is available for use as an explanation (reason) of how or why a particular classification was made (as are the best concepts for losing classifications, indicating why they were not chosen). The concept indicates a basis for the decision.

As this discussion indicates, while being primarily case-based, our technique is actually a mixed approach. The long-term representation of each class is an indexed set of past instances. These previous cases are used at the time of classification to form a best concept that argues for a new input instance being in each possible class. The final classification decision is made based on a ranking of these best concepts. While long-term representation is purely case-based, short-term decisionmaking is concept-based.

We first give a very brief overview of related approaches to classification before returning to describe an initial implementation of our ideas. We then present examples of our system's behavior, thereby demonstrating some of its properties. We conclude with a discussion of research directions that remain open for further exploration, based upon the notions we introduce here.

## Related Approaches

Our research is in the general area of empirical classification, i.e., classification based upon previous experience. We briefly consider two common approaches to empirical classification. First we review the case-based approach, where previous experience is represented directly in the form of individual, prior instances. We then consider the inductive, concept-based approach, where previous experience is represented as generalized descriptions of classes.

Given a case-based approach to classification, the primary issues of interest are case representation, case indexing, and measures of inter-case distance. Case representation is fundamental to all approaches for classification, reflecting the prerequisite need to define an input language for instances. We will not deal with this issue at this point, beyond noting its basic importance. We later will describe the representation we have used in our initial implementation, discussing pros and cons of our choice at that time.

Indexing of cases is critical for efficient retrieval of those cases that are relevant to classifying a new case. The indexing issue is especially critical for case-based approaches to classification, as the number of cases typically is several orders of magnitude greater than the number of partial concepts (e.g., rules) describing a given class. For a given indexing scheme, trade-offs must be considered between the time required to insert new cases and the time required to retrieve relevant cases.

Measures of inter-case distances are also essential elements of the case-based approach, both for the retrieval of relevant cases and in the evaluation of alternative classifications. A case-based approach normally reflects the use of similarity judgements as bases for classification. A single case can be understood to occupy a point in the space of all possible cases (i.e., those representable in the input language). When considering a new input instance, one wants to retrieve those cases that are in some way "near", and thus "similar", to the new instance.

Distance metrics are framed in terms of the input, case representation language. Such metrics are best designed to meet several criteria, including the identity property (i.e., that the measure between identical case descriptions is 0) and the triangle property (i.e., that the distance between two instances, a first and second, is less than or equal to the sum of the distances between the first and a third case and that third case and the second) [Jain and Dubes, 1988].

When the number of cases is small, a distance metric may reflect the complexity of inter-case mappings used to form potential analogies [Falkenhainer, 1989]. Identifying sub-parts of structured case descriptions and using these identifications to "complete" descriptions in both given and target domains can determine the complexity and degree of consistency of proposed mappings. Part of such case completions can be viewed as being equivalent to making a classification.

When the number of cases is large, distance measures between pairs of cases can be used to organize instances into hierarchical or partitional clusters. Various techniques have been defined for this clustering task, including successive nearest neighbor, least mean square-error, agglomeration, decomposition, multi-

dimensional scaling, and graph-theoretic approaches [Jain and Dubes, 1988]. The classification decision for a new case can be made on the basis of the cluster having the nearest neighbor (i.e., most on-point case [Ashley and Rissland, 1988]) or the nearest center of gravity, or being least distorted by addition of the new case.

The notion of cluster as an intermediate-level representation leads us to the other inductive, concept learning approach. Here an intensional representation of a class is created as one or more concepts that generalize presented input instances. One common concept representation is that of a (disjunctive) set of rules, each rule indicating a conjunction of constraints that a case must satisfy to be assigned to a given class. These rules are induced by a process of instance set generalization. Various algorithms has been proposed for this induction process, each exhibiting its own interesting behavioral properties. These algorithms include the AQ algorithm [Michalski, 1983], version space search [Mitchell, 1982], general heuristic search [Rendell, 1986], and genetic algorithms [Goldberg, 1989].

Another form of abstract concept representation is the discimination tree. Here a set of classes to be discriminated amongst is represented by a branching structure of internal nodes that are tests and leaves that are classification decisions. A new case is filtered down through the structure according to outcomes on tests, until a decision node is reached. From the initial definition and verbal learning algorithm explored by EPAM [Feigenbaum, 1963], a variety of optimal and near-optimal algorithms [Quinlan, 1986], as well as recent incremental learning approaches have been defined and developed [Utgoff, 1988].

One advantage of the concept-based approach to classification is that a justification (i.e., in the form of rule contraints satisfied or discrimination tree nodes traversed) is generated as part of the classification process. This can be important for any system expected to explain its behavior. A disadvantage is that the concept representations are determined at some time prior to the time of a current classification task and thus may not adequately reflect the influence of recent cases encountered since formation of the concept. This issue has been discussed as the problem of concept drift [Schlimmer and Granger, 1986]. The approach we introduce here maintains the first advantage while overcoming the second disability.

## Our Approach

### case-space representation

While the general definition of our approach does not depend critically on choice of case representation, the representation we have chosen provides a convenient framework within which to investigate underlying algorithms and issues. We have chosen a representation that is essentially content-free, in order to focus attention on relevant algorithms and issues rather than on a particular domain wherein results could be generated that might appear impressive but avoid essential issues of generality and complexity.

We represent each case as a vector of positive integers. Each element of the vector represents some dimension of information (i.e., feature, attribute) about the

case domain. The integer values allowed for each dimension are assumed to lie along a linear scale; the integer difference between two values corresponds to the degree of semantic difference between the values represented in the case domain. We assume that all of the dimensional scales are normalized, i.e., an integral difference on one dimension represents the same degree of semantic difference as on another dimension of the domain.

While at first appearing rather primitive, this representation actually can be viewed as a generalization of the dimension-based representation employed by Rissland's group in their investigation of legal reasoning [Ashley and Rissland, 1988]. Each dimension in their representation only can indicate the presence or not of that dimension in a current case; this is equivalent to only taking on the values 0 or 1 for each dimension. They do concern themselves with the representation of knowledge necessary to decide upon the presence of a given dimension, which knowledge we consider external to our system.

We organize a set of cases into a *case-space*, an indexing structure that represents selected properties of a set of case instances and provides accessing mechanisms to the cases themselves. To define a case-space for our representation, one simply indicates the number*d* of dimensions and the number $n_i$ of integer values for each dimension $i$. The values on each diimension are assumed to range from 1 to $n_i$.

A case-space has been implemented as a vector [0 ... d], where the 0th element maintains general features of the case space, such as number of cases, dimensions, and possible values on each dimension. Elements 1 through d of the case-space vector are themselves vectors having indices [0 ... $n_i$]. The 0th element of a dimensional vector holds general information about the distribution of cases on that dimension. Each other index value holds a list of those cases having that value on that dimension; these lists start with a count of cases on the lists. Each case is a vector [0 ... d], where indices 1 .. d hold the value on that dimension for that case; the 0th position is used for a variety of purposes by case-accessing functions.

One of the primary considerations in the design of a case-space indexing structure is the trade-off between the time required to insert a new case and the time required to retrieve cases that have given values on given dimensions. Our implementation requires O(d) effort on insertion and constant effort to retrieve a list of cases having a given value on a given dimension. As d is constant for a given case-space, insertion time is essentially constant, as well. The time to retrieve cases having given values on a given set of dimensions is linear in the sum of the number of cases having specified value(s) for each dimension. We realize this performance by using the 0th element of each case description vector to count the number of times a case has been referenced during retrieval.

We provide several functions to create randomly distributed case-spaces. These functions are used as tools for experimentation. A call to

```
(make-random-case-space class-A (10 10 10 8 8 10) 1000)
```

would create a 6-dimensional case-space named class-A containing 1000 cases, with values on dimensions distributed uniformly over the ranges indicated by the first argument. A call to

```
(make-random-case-space class-B  (10 5 8)  500
                        (distributions  ((1 20) (6 50) (7 20) (10 10))
                                        ((2 40) (3 60))
                                        ((1 10) (2 20) (3 40) (4 20) (5 10))))
```

would create a 3-dimensional case-space named class-B containing 500 cases, with values distributed over the various dimension values in direct proportion to the weightings given; any dimension values not listed in the distributions argument will have no cases assigned.

Presently, each class of interest we represent as a separate case-space. Thus, by using the distributions option, we can easily create interesting interactions among class memberships in order to illustrate or investigate behaviors of ours or other classification algorithms.

### concept formation

When an input instance (represented as a case) is presented to our system for classification, new concepts are formed for each possible class which are then used as bases for making a classification decision. Here we describe the process whereby the candidate concepts are formed.

A concept is represented as a list, in which each element has the form

(dimension (low-range high-range)),

where $1 \leq \text{dimension} \leq d$, and $1 \leq \text{low-range} \leq \text{high-range} \leq n_{\text{dimension}}$.

A concept is essentially a constraint expression. For each dimension in the list, the value on that dimension of any exemplar must lie within the closed interval [low-range high-range]. A concept list is interpreted as a conjunction over internal disjunctions on the dimensions. As an example, the concept ((1 (2 5)) (3 (3 3))) describes the class where values on the first dimension can be either 2, 3, 4, or 5 and on the third dimension the value must be 3.

When forming a new concept, we first create a list of possible (sub)ranges for each dimension. This is done by comparing the value of the new case on each dimension with the range of values on that dimension that is associated with current cases in the case space. This information is kept as part of the information about a dimension on the 0th element of the dimension's vector. If the new value lies within one of the subranges previously seen, that subrange is used as generator for specific possibilities. If it does not, the nearest range is chosen and extended to include the new case, if possible. One might put an upper limit on how far (over how many values) one is willing to extend a subrange, possibly based upon density of cases in the case space. We currently limit this distance to 1 and generate no suitable subrange for a dimension if the least distance is greater.

Given this so-called generating subrange for a dimension, we create a list of all subranges that include the new case value on that dimension and at least one value from previous cases. These ranges become the possible subranges for that dimension in a concept rule. The set of possible concepts, which is the search space for defining a best concept that includes the new case, consists of all possible concept lists formed by taking zero or one possible subrange from each dimension.

For example, suppose in a 2-dimensional case-space, the new case is (2 7) and the previous ranges for the class of interest on the two dimensions are (1 (1 4) (8 9)) and (2 (4 6)), respectively. The set of possible ranges for the first dimension is ((1 2) (2 2) (2 3) (2 4) (1 4)). Notice we ignore the subrange involving (8 9) as being irrelvant and only consider those subranges of (1 4) that include the value 2. For the second dimension, the set is ((4 7) (5 7) (6 7)). Notice here that we extend the previously seen range to include 7, and we only include subranges that also include a previously seen value, i.e., not just 7 alone.

There are 23 possible concepts that can be formed by matching elements from the two lists given above or by taking only one of them (ignoring the other). In the worst case, the number of candidates is on the order of $((s+ 2) / 2)^{2d}$, where $s$ is the maximum dimension size. This is not a small number, but fortunately, it is not encountered in most reasonable situations where classes only occupy parts of each dimension's range.

### concept evaluation

Now that we can generate a set of candidate concepts for a given class, how do we proceed to evaluate them? What are some properties of an evaluation function? Presently, we have considered measures of a concept's necessity and sufficiency.

A concept's sufficiency represents the degree to which satisfaction of the concept's constraints indicates that the instance is a member of the class. Being a member is equivalent to not not being a member. Thus, sufficiency will be evaluated in terms of the percentage of elements from competing classes that satisfy concept constraints. The sufficiency *Suff* of a candidate concept will be equal to $(1 - p_{out})$, where $p_{out}$ is the percentage ($0 \leq p_{out} \leq 1$) of instances in competing classes that satisfy the concept's constraints. A concept c will be said to be *sufficient* if its *Suff(c)* score is 1. This means that no instance of another class satisfies the constraints of the concept; thus, satisfying the constraints of the concept is sufficient to guarantee that the instance is in the class.

A concept's necessity represents the degree to which satisfaction of the concept constraints is required for membership in the class. Necessity will be evaluated in terms of the percentage of members of the class that satisfy a concept's constraints. The necessity *Nec* of a candidate concept will be equal to $p_{in}$, where $p_{in}$ is the percentage ($0 \leq p_{in} \leq 1$) of instances that are in the class that satisfy the concept's constraints. A concept c will be said to be *necessary* if its *Nec(c)* score is 1.

We will evaluate the overall suitability *Suit(c)* of a candidate concept *c* by combining its sufficiency and necessity, letting *Suit(c)* equal *Nec(c)* * *Suff(c)*. If a concept is both necessary and sufficient, its suitability will be 1. Anything less and its suitability value will lie between 1 and 0, where 0 is possible only if all members of competing classes satisfy a concept's constraints. (We guarantee a non-zero *Nec* value for all candidate concepts by our concept formation strategy.) If we order evaluation of concepts according to decreasing generality, when we reach a concept that is sufficient, we need not consider more specific concepts, as they will only include the same or fewer instances of the class (i.e., yield an unimproved*Nec* score). We have chosen to continue search until the *Nec* score does in fact decrease, thus finding the most specific concept with a given suitability score.

### classification decision

Now that we can evaluate candidate concepts for a given class, how do we make a classification decision among a set of possible classes?

We implement a function (classify <instance> <class-list>), which takes an input instance as a case <instance> and a list of possible classes <class-list>, each class of which is represented as a case-space (all with common dimensional definitions).

The function classify forms candidate concepts for each class, evaluates those concepts, and selects the best one(s) for each, as discussed above. It finally returns as its value the name of the class associated with the concept(s) having the highest suitability value among the best candidate concepts associated with possible classes. We also tag the class name with the best concepts and their suitability value that led to the decision.

### Examples

We demonstrate our system on a number of examples. These are meant to illustrate interesting properties of our approach.

Suppose we define class-A by

```
(make-random-case-space class-A (10 5 8)  500
                        (distributions  ((1 20) (2 50) (3 30))
                                        ((2 40) (3 60))
                                        ((1 10) (2 20) (3 30) (4 40)) ))
```

and class-B by

```
(make-random-case-space class-B (10 5 8)  500
                        (distributions  ((6 20) (7 50) (8 30))
                                        ((4 50) (5 50))
                                        ((6 30) (7 30) (8 30)) )).
```

Note that these two classes do not overlap. If we ask to classify an input instance that falls inside one or the other, we get necessity and sufficiency. For instance, if we pose (classify (make-case 2 2 2) '(class-A class-B)), we get the

result class-A with a list of several necessary and sufficient concepts, including the most specific concept ((1 (1 3)) (2 (2 3)) (3 (1 4))), with a suitability score of 1. No candidates could even be formed for class-B as we currently do not allow a class to extend a subrange by more than 1 on any dimension.

If we ask to classify a case that is just beyond the edge of one of the two classes and not near the other, we also get necessity and sufficiency with a concept extended to include the new case. Thus, if we pose (classify (make-case 4 1 2) '(class-A class-B)), we get the result class-A with several best concepts, including the most specific ((1 (1 4)) (2 (1 3)) (3 (1 4))), again with a suitability score of 1.

If a new input instance has some values in range of one class on one dimension and another values within range of the other class on another dimension, we can get necessary and sufficient rules for two classes. An example is (classify (make-case 1 5 1) '(class-A class-B)). Here the best concepts for class-A include ((1 (1 3)) (3 (1 4))) while for class-B it is ((2 (4 5))), both having suitability scores of 1. Notice that the competing concepts focus on favorable dimensions their respective class, as lawyers would do in finding reasons for a desired decision in a law case.

Suppose we redefine class-A to include disjoint value distributions on the first and third dimensions and a partial overlap with class-B on the second, as follows:

```
(make-random-case-space class-A (10 5 8)  500
                        (distributions  ((1 20) (2 30) (9 20) (10 30))
                                        ((4 40) (5 60))
                                        ((1 10) (2 20) (7 30) (8 40)) )).
```

If we pose the following classification problem (classify (make-case 2 3 2) '(class-A class-B)), we get the results class-A with a score of 0.496 realized by best concepts (1 (1 2)) (2 (3 5))) and ((1 (1 2)))). The first is rather non-intuitive, but is equivalent in suitability value to the second, as (2 (3 5)) contains all class-A cases and all class-B cases, but no class-B case can satisfy that and the other constraint (1 (1 2)). More importantly, here we see the best concepts only cover relevant, nearby portions of class-A's membership, not attempting to capture all.

Now let's further explore overlap among classes, redefining class-A again as

```
(make-random-case-space class-A (10 5 8)  500
                        (distributions  ((4 50) (5 30) (6 20))
                                        ((3 30) (4 70))
                                        ((3 40) (4 20) (5 30) (6 10)) )),
```

so that class-A shares its highest value on each dimnension with class-B. When we pose the following classification problem (classify (make-case 4 3 3) '(class-A class-B)), we get the fully expected result that class-A is preferred; however, the best concept (((1 (4 6)) (2 (3 4)) (3 (3 6)))), with a suitability of 0.964, is a bit surprising. This concept covers all of class-A, so its necessity value is 1; even though it overlaps class-B on all dimensions, very few cases in class-B share all these ranges, so sufficiency is near 1.

If we choose to classify a border case by posing (classify (make-case 6 4 6) '(class-A class-B)), we get the same best concept for class-A, but class-B now wins with its all-encompassing (necessary) concept ((1 (6 8)) (2 (4 5)) (3 (6 8))), that has a suitability score of 0.992 (most likely due to the first and third dimensions).

## Conclusion

In this paper, we have introduced a new case-based classification technique that bases its decisions on best possible concept descriptions formed at classification time. Our scheme has the flavor of goal-based, on-the-fly, concept formation, i.e., finding the best concept to classify an input instance as a member of each class as goal and select the class with concepts of highest evaluation.

Several avenues for further research present themselves, based upon our initial definition and implementation of the method. One is to explore alternative case representations. One closely related extension is to introduce hierarchical value spaces for case dimensions. Value generalization for concept formation would depend on traversing a value hierarchy rather than expanding a subrange.

Another direction for research would be to contrast and compare various evaluation functions. One change in evaluation, sparked by consideration of the overlap cases above, is to only allow sufficient concepts (those with no counter-examples). Using this evaluation, when we rerun our last two examples involving overlappedclasses, for (classify (make-case 4 3 3) '(class-A class-B)) we get class-A with concept ((1 (4 6)) (2 (3 4)) (3 (3 5))) and score of 0.904 and for (classify (make-case 6 4 6) '(class-A class-B)) we get no possible concepts for class-A or class-B. Possibly of more interest is (classify (make-case 5 4 6) '(class-A class-B)), which returns class-A with concept ((1 (4 5)) (2 (3 4)) (3 (3 6))) and score 0.78.

Research exploring mixed case-based and rule-based decisionmaking would be a natural extension [Rissland and Skalak, 1990]. Saving the best concept used to classify a case at the time it was input could be considered when addressing the relevance of cases in supporting formation of concepts and eventual classification of new inputs (e.g., what was the rule of law that case was thought to represent).

Exploring the role that previous cases and the new concepts that are generated on the fly could play in a general theory of argumentation is another intriguing, potentially enlightening direction.

Finally, an in-depth consideration of a particular application domain, defining relevant dimensions and determining knowledge necessary to map given situations onto dimension values, hopefully would demonstrate the usefulness and naturalness of our classification approach.

## References


[Ashley and Rissland, 1988] Ashley, K. D. and Rissland, E. L., "A Case-based Approach to Modeling Legal Expertise", in *IEEE Expert*, Fall, 1988, p70-77.

[Dworkin, 1977] Dworkin, R., *Taking Rights Seriously*, Harvard University Press : Cambridge, MA, 1977.

[Feigenbaum, 1963] Feigenbaum, E. A., "The Simulation of Verbal Learning Behavior", in *Computers and Thought*, Feigenbaum, E.A. and Feldman, J. (eds), McGraw-Hill : New York, NY, 1963, p297-309.

[Gardner, 1987] Gardner, A. vdL., *An Artificial Intelligence Approach to Legal Reasoning*, Bradford Books/MIT Press : Cambridge, MA, 1987.

[Falkenhainer, 1989] Falkenhainer, B., Forbus, K.D. and Gentner, D., "The Structure-mapping Engine: Algorithms and Examples", *Artificial Intelligence*, 41, 1989, 1-63.

[Goldberg, 1989], *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley : Boston, MA, 1989.

[Hart, 1961] Hart, H.L.A., *The Concept of Law,* Clarendon Press, Oxford, 1961.

[Jain and Dubes, 1988] Jain, A. K. and Dubes, R. C., *Algorithms for Clustering Data*, Prentice Hall : Englewood Cliffs, NJ, 1988.

[Michalski, 1983] Michalski, R. S., A Theory and Methodology of Inductive Learning", in *Machine Learning: An Artificial Intelligence Approach*, Michalski, R., Carbonell, J., and Mitchell, T. (eds), Tioga : Palo Alto, 1983, p83-134.

Mitchell, T. M., "Generalization as Search", *Artificial Intelligence*, 18 (1982), p203-206.

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

[Rendell, 1986] Rendell, L. "A General Framework for Induction and a Study of Selective Induction", *Machine Learning*, 1(1), 1986, p177-226.

[Rissland and Skalak, 1990] Rissland, E. L. and Skalak, D.B., "CABARET: Rule Interpretation in a Hybrid Architecture", COINS Technical Report, 90-97, University of Massachusetts, 1990.

[Schlimmer and Granger, 1986] Schlimmer, J. and Granger, R., "Beyond Incremental Processing: Tracking Concept Drift", in *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, p 502-507, 1986.

[Utgoff, 1988] Utgoff, P.E., "ID5: An Incremental ID3" in *Proceeedings of the Fifth International Workshop on Machine Learning*, 1988, p 107-120.