Hybrid Inductive Machine Learning: An Overview of CLIP Algorithms

Krzysztof J. Cios and Łukasz A. Kurgan

Computer Science and Engineering Department
University of Colorado at Denver
Campus Box 109
Denver, CO 80217-3364, U.S.A.
E-mail: kcios@carbon.cudenver.edu

1. Introduction

The chapter describes inductive machine learning methods for generating hypotheses about given training data. It focuses on hybrid algorithms that generate hypotheses in the form of production if… then… rules, which constitute the model of the data.

Machine learning (ML) can be defined as the ability of a computer program to improve its own performance, based on the past experience, by generation of a new data structure that is different from an old one, like production rules from input numerical or nominal data. The advantage of machine learning is that the generated description, in the form of rules or decision trees, is explicit. The rules can be analyzed, learned from, or modified by the user. Because of this machine learning is a preferred data mining method, in particular in situations where a decision maker needs to understand and validate the generated model. As such machine learning is often used as the key step in the knowledge discovery process (Cios et al., 2000). Here are a few reasons why there is a big interest in machine learning (Cios et al., 1998):

- We observe exponential growth of the amount of data and information due to the fast proliferation of the Internet, digital database systems, and information systems. Thus, automation of the processing of that huge data, which can be done via ML, becomes a crucial task.
- It can provide techniques for analysis, processing, granulation, and extraction of the data.

- In some areas machine learning can be used to generate "expert" rules from available data, especially in medical and industrial domains, where there may be no experts available to analyze the data.
- It helps in understanding human cognitive processes and enables further development of better machine-human learning strategies, while taking into account the accumulated knowledge, analogical reasoning, theory formation, etc.
- Production rules can be easily "fuzzyfied", they can also help in a neural network design and deciphering the knowledge stored in the network's weights and connections, to mention just two other important data mining tools.

A machine learning process consists of two phases. The learning phase, in which the system analyzes the data and generates the rules by finding some similarities among the data, and the validation phase, in which the generated rules must be verified by computing some performance evaluation function on new set of data.

Machine learning algorithms can be categorized in several ways. Most importantly they are divided into supervised and unsupervised algorithms. They can use inductive vs. deductive types of learning, incremental vs. non-incremental learning modes, etc.

In *supervised learning* the user is a teacher who provides examples labeled with class values. In case where there is no a priori knowledge of classes, supervised learning can be still applied if the data has a natural cluster structure. Then, a clustering algorithm has to be run first to reveal these natural groupings.

The training data set consists of M training data pairs (examples):

$$S = \{(\mathbf{x}_i, c_j) \mid i = 1,...,M; j = 1,...,C\}.$$

where: $\mathbf{x}_i$ – n-dimensional pattern vector, whose components are called features,
c_j – known class.

The algorithm's role is to search the space of possible hypotheses to discover the best estimate of the mapping function f, such that $c = f(x)$. For the search to be successful the assumption has to be made that the features represent only properties of the examples but not the relationships between the examples. A machine learning algorithm generates hypotheses by finding common features and their values for examples representing each class. Then, the generated hypotheses are applied to the new examples to predict their class membership.

In *unsupervised learning* the system learns the classes on its own. This type of learning learns the classification by searching trough common properties of the data. An example of unsupervised learning is conceptual clustering (Michalski,

1980; Fisher and Langley, 1986; Fisher, 1987), which is quite different from classical clustering. Conceptual clustering consists of two tasks: clustering itself, where the clusters in a given data set are found, and *characterization* where, for each found cluster, a concept description is generated. Conceptual clustering can be thought of as a hybrid of unsupervised (clustering) and supervised (characterization) learning. In theory, it is possible to transform a supervised machine learning algorithm into an unsupervised one (Langley, 1996) by running the supervised algorithm as many times as there are features describing the examples, each time with a different feature playing the role of the class attribute.

Two basic techniques for inferring the information from data are deduction and induction. *Deduction* infers information that is a logical consequence of the information in the database. The deduction technique can be used if the data describing some domain is proven to be correct. *Induction* infers generalized information, or knowledge, by searching for regularities among the data. Inductive learning produces results that are always correct for the data but only plausible outside of the data. Learning by induction is a search for a correct hypothesis/rule, or a set of them, which is guided by the given examples. Majority of the machine learning algorithms are inductive.

Incremental learning is performed by providing the learning examples to an algorithm one at a time, while in *non-incremental learning* all of the learning examples are provided to an algorithm simultaneously.

1.1. Machine learning issues

One of the main issues in machine learning is the presence of noise in the data. The noise can be present in the features, that constitute an example, and/or in the class descriptions, like false examples. Only some of the machine learning algorithms are noise-tolerant, which means that they can generate the rules that are not overfitted, i.e, they do not cover noisy examples.

Another issue is the generalization and specialization factor of the generated rules. A general rule covers more examples, and thus might perform better on unseen data then a more specific rule. An example is covered by a rule when it satisfies all conditions of the *if* part of the rule.

In cases where the number of the generated hypotheses is excessively large an algorithm has to choose a subset of them (Cios et al., 1998) by means of :

- heuristics, like *Occam's razor* (choosing the shortest rule is best)
- minimum description length principle (a generalization of the Occam's razor heuristic)
- background knowledge about the domain
- reasoning from first principles (like laws of physics, mathematical theorems)

- decisions made by the user, based on his/her knowledge of the problem.

There is also an issue of the size and dimensionality of the data. Only a few algorithms can deal with big and highly dimensional data. We are still dealing with some variation of the classic artificial intelligence problem of systems that work for several examples but are not really scalable (Schank, 1991). Among the algorithms that seem to overcome the problem is the CLIP4 algorithm (Cios and Kurgan, 2001).

1.2. Generation of Hypotheses

In the machine learning world one works with data that represent information. The definition of an information system is given below:

$$IS =< S, Q, V, f >$$

where :

S – a finite set of examples, $S = \{e_1, e_2, ..., e_M\}$, M – the number of examples
Q – a finite set of features, $Q = \{F_1, F_2, ..., F_n\}$, n - number of features
$V = \cup V_{F_j}$ – a set of feature values V_{F_j} – the domain of feature $F_j \in Q$
$v_i \in V_{F_j}$ – a value of feature F_j
$f = S \times Q \to V$ – an information function, that satisfies:

$$f(e_i, F_i) \in V_{F_j} \text{ for every } e_i \in S \text{ and } F_j \in Q$$

Note: S is often called the learning set, which is a subset of the entire universe, defined as a Cartesian product of feature domains V_{F_j} (j=1,2…n).

As an example let us define a patient whose condition is described by two features: F_1=temperature and F_2=blood pressure. Feature F_1 can take on three values: low (<36°C), normal (<36°C, 37°C>), and high (>37°C), while feature F_2 can take on values: low (< 90/170), normal (<90/70, 130/90>), and high (>130/90). Thus, the feature domains are:

$$V_{F_2} = \{low, normal, high\}$$
$$V_{F_2} = \{low, normal, high\}$$

A possible hypothesis, using the above information about the patient, can read: for any patient with normal temperature and normal blood pressure make a decision, or in the rule form:

IF F_1= normal AND F_2= normal THEN decision

To come up with such a rule we need a learning data set. The learning examples are then analyzed by a machine learning algorithm to generate production rules. One has to remember that learning examples do not cover the entire universe of possible examples. Thus, the generated rules need to be general enough to describe these unseen examples. In the next step, the data are divided into two parts: positive examples (examples that describe the concept), and negative examples (counterexamples).

Let us assume that we have the training data as shown in Table 1.

Table 1. Example training data

Patient No	F_1 (Temperature)	F_2 (Blood pressure)	Decision
1	Normal	Low	Go home
2	Normal	Normal	Go home
3	Low	Normal	Go home
4	High	High	Treatment

Thus, for our information system:

$S = \{1,2,3,4\}$, M=4

$Q = \{F_1, F_2\}$, n=2

Positive examples describe the situation when patient is sent home (patients 1,2and 3). By visual analysis of this data the following rules can be generated:

IF F_1=normal AND F_2=low THEN go home
IF F_1= normal AND F_2= normal THEN go home
IF F_1=low AND F_2= normal THEN go home

or

IF F_1= normal THEN go home
IF F_2= normal THEN go home

The first three rules are more specific because they cover one example each, they overfitt the data, and thus might produce poor results on test data. The last two rules are more general, because they can cover more examples while still not covering the fourth patient.

The goal of learning algorithms described in this chapter is to generate a set of rules (hypotheses) that best describe the learning data. After learning, the rules are tested using another set of unseen validation examples. If the rules fail to correctly

classify majority of the validation examples the learning should be repeated using procedures described in Appendix 3 (on Overfitting).

The common feature of all machine learning algorithms is their ability to almost perfectly classify the training examples. However, the true value of the rules generated by any algorithm can be evaluated only by testing them on new data. It is also important to establish balance between generalization and specialization to generate the best possible set of rules.

The rule can be generalized or specialized by the following operations (Cios et al, 1998):

- *Replacing constants with variables* (more general rule can be generated by replacing constants in rules that have the same outcome by a variable, and merging them into one rule), for instance two rules:

  IF F_1=normal AND F_2=low THEN go home
  IF F_1= normal AND F_2= normal THEN go home

  can be replaced by a more general rule:

  IF F_1=normal THEN go home

- *Using disjuncts for rule generalization and conjuncts for rule specialization*

- *Moving up in a hierarchy for generalization*. If there is a known hierarchy in a given problem domain, the generalization can be performed by replacing the conditions involving the knowledge of the lower level by the common conditions involving the knowledge of the upper level

- *Chunking*. This mechanism is based on the premise that given the goal, every problem encountered on the way to this goal can be treated as a sub-goal.

This chapter first briefly describes rule and decision tree algorithms and then concentrates on hybrid algorithms, its main topic. The rule algorithms are represented by the family of AQ algorithms (Michalski et al., 1986), and decision tree algorithms by ID algorithms (Quinlan, 1993). Hybrid algorithms combine the best features of the two approaches: they are represented in the chapter by the CN2 algorithm (Clark and Niblett, 1989) and the CLIP family of algorithms.

1.3. Rule Algorithms

They will be described by using AQ15 algorithm (Michalski et al., 1986). There were several improvements (Kaufman and Michalski, 1999) introduced in subsequent versions of AQ algorithm, but the main idea can still be described

using the AQ15 algorithm. AQ algorithm uses variable-value logic calculus (VL1) (Michalski, 1974). Below some basic definitions of VL1 are defined:

Selector - it is relational statement of the form: $(F_i \; ? \; v_i \;)$

where: ? – any relational operator like = or ≠,

v_i – a values of attribute F_i

e.g. (F_1≠low)

Complex (L) – a logical product of selectors: $L = \cap(F_i \; ? \; v_i \;)$

e.g. ((F_1≠low) AND (F_2=high OR low))

Cover (C) – a disjunction of complexes: $C = \cap L_i$

e.g. ((F_1≠low) AND (F_2=high OR low)) OR (F_1≠low)

Operations, which can be performed in VL1, are defined below:

Generation of a *star* $G(e_i \mid E_i)$ and generation of a *cover* $G(E_1 \mid E_2)$, where $e_i \in E_i$ and E_1 and E_2 are two sets such that $E_1 \cup E_2 = S$

Star is defined as: $G(e_i \mid E_i) = \cap G(e_i \mid e_j), \quad \forall e_j \in E_j, j \neq i$, so this is a conjunction of all $G(e_i|e_j)$. Each $G(e_i|e_j)$ value is obtained by comparing the features from e_i and e_j examples, skipping those which are the same, forbidding e_i from taking on the same values as e_j, and combining all generated selectors using disjunction., e.g.:

e_1={normal, low}, e_4={high, high}
$G(e_1|e_4)$ = (F_1≠high)OR(F_2≠high)

Cover is defined as: $G(E_i \mid E_j) = \cap E(e_i \mid E_j), \quad \forall e_i \in E_i, j \neq i$, so this is a conjunction of all evaluated stars.

To evaluate "goodness" of given *cover* the sparseness function is used; a number, defined as the total number of examples, which can be potentially covered by given *cover* minus the number of examples, which are actually covered by the *cover*. If value of sparseness is smaller then the cover is more compact. The cover with the smallest sparseness value is chosen, in agreement with the minimum description length principle

The pseudo-code for the AQ15 algorithm (Michalski, 1986) follows:

Given: positive and negative example training sets.
Part 1. While partial *cover* does not *cover* all positive examples do:
1. Select an uncovered positive example (a *seed*)
2. Generate a *star*, that determine maximally general *complexes covering* the *seed* and no negative examples
3. Select the best *complex* from the *star*, according to the user-defined criteria
4. Add the *complex* to the partial *cover*

Part 2. While partial *star covers* negative examples do:
1. Select a *covered* negative example
2. Generate a partial *star* (all maximally general *complexes*) that *covers* the seed and excludes the negative example
3. Generate a new partial *star* by intersecting the current partial *star* with the partial *star* generated so far
4. Trim the partial star if the number of disjoint complexes exceeds the predefined threshold, called maxstar (to avoid exhaustive search for covers which can grow out of control)

Result: Rules covering all positive examples and no negative examples.

The AQ15 algorithm performs a top-down search through all positive examples and generates a decision rule for each class in turn. At each step it starts with selecting one positive example (the seed) and generates all complexes (a star) that covers the seed, but does not cover any negative example. Then by using user-defined criteria (sparseness function and length of complexes) it selects the best complex from the star. Then this complex is added to the current (partial) cover.

Part 1 of the AQ15 algorithm can be rewritten in an easier to implement form, namely generation of cover involves these three steps:

For each positive example $e_i \in E_1$ (E_1 is a set of positive examples):
1. Find $G(e_i \mid e_j)$ for each $e_j \in E_2$, where E_2 is a negative set of examples
2. Find a *star* $G(e_i \mid E_2)$, as a conjunction of all $G(e_i \mid e_j)$ from step 1. If there is more than one $G(e_i \mid e_j)$ (after conversion into disjunctive form) select the best one according to smallest *sparseness*
3. Find a *cover* $G(E_1 \mid E_2)$ of all positive examples against all negative examples, as a disjunction of all *stars* from step 2. The final *cover* covers all positive examples and no negative examples

In part 2 of the AQ15 algorithm the partition between positive and negative examples, that initially do intersect, needs to be achieved. The goal is to come up with the information function (IF) that results in the partition of the learning data.

Having two disjoint sets of examples E_{01} and E_{02} we perform the following operations in step 2 of the AQ15 algorithm:

1. Generate information functions: IF_1 and IF_2 using these sets to generate subsets E_1 and E_2 which are covered by these information functions
2. If sets E_1 and E_2 do not intersect we have the partition - STOP, otherwise we calculate differences between sets E_1 and E_2 and the intersecting sets $E_p = E_1 - E_1 \cap E_2$ and $E_n = E_2 - E_1 \cap E_2$, generate corresponding to E_n and E_p information functions IF_p and IF_n.
3. For all examples e_i from the intersection we create sets $E_p \cup e_i$ and $E_n \cup e_i$ and generate information functions IF_{pi} and IF_{ni} for them
4. Check if (IF_p, IF_{ni}) and (IF_n, IF_{pi}) create partitions of S:
   - If yes, choose better partition using sparseness, chosen pair becomes new sets E_1 and E_2, go back to step 1 taking next example from the intersection
   - If not, go to step 2 and check another example from intersection.

Information function represents rule (hypothesis) that covers positive examples. This algorithm does not guarantee that all examples will be assigned into one of the two subsets if partition is not achieved.

The example how to generate rules using rule algorithms is given below, using the data from Table 1. (E_1={1,2,3}, E_2={4})

$G(e_1|e_4) = G(e_1|E_2) = (F_1 \neq \text{high})$ OR $(F_2 \neq \text{high})$

$G(e_2|e_4) = G(e_3|e_4) = G(e_1|e_4)$

Now the sparseness is calculated: for $F_1 \neq \text{high} = 6\text{-}3 = 3$, for $F_2 \neq \text{high} = 6\text{-}3 = 3$

Thus the first rule is: IF $(F_1 \neq \text{high})$ THEN class "go home"

Now the rule for "treatment" case is generated:

$G(e_4|e_1) = (F_1 \neq \text{normal})$ OR $(F_2 \neq \text{low})$

$G(e_4|e_2) = (F_1 \neq \text{normal})$ OR $(F_2 \neq \text{normal})$

$G(e_4|e_3) = (F_1 \neq \text{low})$ OR $(F_2 \neq \text{normal})$

$G(e_4|E_1) = ((F_1 \neq \text{normal})$ OR $(F_2 \neq \text{low}))$ AND $((F_1 \neq \text{normal})$ OR $(F_2 \neq \text{normal}))$ AND $((F_1 \neq \text{low})$ OR $(F_2 \neq \text{normal})) =$

$((F_1 \neq \text{normal})$ OR $(F_1 \neq \text{normal}$ AND $F_2 \neq \text{normal})$ OR $(F_2 \neq \text{low}$ AND $F_1 \neq \text{normal})$ OR $(F_2 \neq \text{low}$ AND $F_2 \neq \text{normal}))$ AND $(F_1 \neq \text{low}$ OR $F_2 \neq \text{normal}) =$

$(F_1 \neq \text{normal}$ AND $F_1 \neq \text{low})$ OR (sparseness: 3–1=2)

$(F_1 \neq \text{normal}$ AND $F_2 \neq \text{normal})$ OR (sparseness: 4–1=3)

$(F_1 \neq \text{normal}$ AND $F_2 \neq \text{normal}$ AND $F_1 \neq \text{low})$ OR (sparseness: 2–1=1)

$(F_1 \neq \text{normal}$ AND $F_2 \neq \text{normal})$ OR (sparseness: 4–1=3)

$(F_2 \neq \text{low}$ AND $F_1 \neq \text{normal}$ AND $F_1 \neq \text{low})$ OR (sparseness: 2–1=1)

$(F_2 \neq \text{low}$ AND $F_1 \neq \text{normal}$ AND $F_2 \neq \text{normal})$ OR (sparseness: 2–1=1)

$(F_2 \neq \text{low}$ AND $F_2 \neq \text{normal}$ AND $F_1 \neq \text{low})$ OR (sparseness: 2–1=1)

$(F_2 \neq \text{low}$ AND $F_2 \neq \text{normal})$ (sparseness: 3–1=2)

Thus the second rule is: IF ($F_1 \neq$normal) AND ($F_2 \neq$normal) AND ($F_1 \neq$low) THEN class "treatment"

The $G(e_4|E_1)$ star includes almost every permutation of the attribute/value pairs, and has to be stores in the memory. The rules generated using rule algorithm are following:

Rule 1: IF ($F_1 \neq$high)) THEN class "go home"
Rule 2: IF ($F_1 \neq$normal) AND ($F_2 \neq$normal) AND ($F_1 \neq$low) THEN class "treatment"

The above rules have 100% accuracy on the training data. Notice, that these rules do not agree with the rules we derived intuitively because the only relational operator used is inequality.

Major advantages and disadvantages of the AQ algorithms are listed below:

- Rules are independent; the rule sets can be added together. It enables incremental learning.
- The rules are modular
- The rules can be easily modified, because of their structure
- Evaluation of a cover and evaluation of the partition of two sets is computationally a **very expansive** process. We have to remember all examples as well as all generated stars and covers, which is very memory consuming and can be impossible to perform for very large data sets.
- It does not reveal relationships between produced rules, as is the case of decision trees. Because of this it is very difficult to see structure of the data on which the algorithm learned. The only way to deal with this disadvantage is to try cluster similar rules based on the similarity of examples which they cover.
- It handles noise outside of the algorithm itself, say by rule truncation.
- It cannot deal with continuous features.

1.4. Decision Tree Algorithms

Decision tree algorithms are represented by the family of ID (C4.5) algorithms (Quinlan, 1993). The decision tree is a model for approximation of discrete-value functions that is capable to learn disjunctive expressions. A decision tree consists of nodes and branches connecting the nodes. The top node in the tree is called the root, and contains all training examples. The bottom nodes of the tree are called leaves, and represent final subsets of the data with associated with them class labels. The decision nodes in the tree are all, but leaf, nodes since they correspond to decisions that are performed at these nodes using a single selected feature. ID3 algorithm is based on a psychological model (Hunt, 1966) of the process that people use when learning simple concepts (Quinlan, 1993). People do it by finding key distinguishing features from the set of training examples. The Hunt's

model is called the *concept learning system* (CLS), and it is similar to a *divide and conquer* method.

Lets assume that the learning set, S, consists of n examples belonging to c classes. The task is to divide this set into disjoint subsets based on a single feature, so that they create a partition. The following pseudocode summarizes the CLS algorithm.

Given: S – set of learning examples
1. Select the most discriminatory (significant) feature
2. Split the entire set S, located at the root of the tree, into several subsets using the selected feature. The number of children nodes originating from the root is equal to the number of possible values the selected feature takes on.
3. Recursively find the most significant feature for each subset generated in step 2 and top-down split it into subsets. If each subset contains examples belonging to one class only (a leaf node) then stop, otherwise go to step 3.

Result: The decision tree from which classification rules can be extracted

Quinlan (1993) used Shannon's entropy as a criterion for selecting the most discriminatory features:

$$Entropy(S)=\sum_{i=1}^{c} - p_i \cdot \log_2(p_i)$$

where: p_i - proportion of the examples belonging to the i-th class.

In ID3 the uncertainty in each node is reduced by choosing the minimal *entropy*. To reach this goal *Information Gain* is used, which measures expected reduction in *entropy* caused by knowing the value of a feature F_j.

$$Information\ Gain(S,F_i)=Entropy(S)-\sum_{v_i \in V_{F_j}} \frac{|S_{v_i}|}{|S|} \cdot Entropy(S_{v_i})$$

where: V_{F_j} - set of all possible values of feature F_j

S_{v_i} - subset of S, for which feature F_j has value v_i

Information Gain is used to select the best feature at each step of growing the decision tree. In later versions of the ID algorithm, the *Gain Ratio* was proposed to compensate for the bias of the *Information Gain* for cases with many outcomes.

$$Gain\ Ratio(S,F_j)=\frac{Information\ Gain(S,F_j)}{Split\ Information(S,F_j)}$$

where: $Split\ Information(S, F_j) = \sum_{i=1}^{C} \frac{|S_i|}{|S|} \cdot \log_2 (\frac{|S_i|}{|S|})$

Split Information is the entropy of S with respect to values of feature F_j. In a situation when two or more features have the same *Information Gain* value, the feature that has less number of values will be selected by the *Gain Ratio* test. Gain ratio gives better results, especially for bigger data sets, and results in smaller trees.

The pseudocode of the discrete ID3 algorithm is:

Given: S – set of learning examples
1. Create the root node containing the entire set S
2. If all examples are positive, or negative, then stop: decision tree has one node
3. Otherwise (general case)
   3.1 Select feature F_j that has the largest *Information Gain* value
   3.2 For each value v_i from the domain of feature F_j:
      a) add a new branch corresponding to this feature value v_i, and a new node, which stores all the examples that have value v_i for feature F_j
      b) if the node stores examples belonging to one class only then it becomes a leaf node, else below this node add a new sub-tree, and go to step 3

Result: The decision tree from which the rules can be extracted

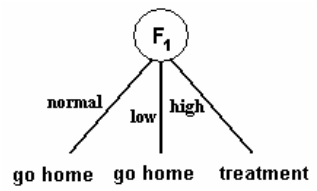
ID3 is using inductive bias during learning, i.e. it prefers small over large decision trees. Decision tree is represented as disjunction of conjunctions of the feature values. The trees can be represented by a set of if…then… rules.

To avoid overfitting decision trees are pruned. Pruning also helps to generate more general rules. The pruning can be done during the process of tree growing (pre-pruning), or after the entire tree is established (post-pruning). The extreme case of the pruning was proposed by Holte (1993). He proposed concept of decision trees that are only one level deep and has shown that the classification performance of such trees is equally good to other more complex algorithms.

Extension of the ID3, the C4.5 algorithm (Quinlan, 1993) allows the user to work with continuous features, to grow trees from data containing missing values, and introduces the windowing techniques to deal with larger data sets. The newest implementations of C4.5 use boosting (Schapire, 1999) and tree pruning techniques.

The decision tree for the training data from Table 1 is shown below. There is a single test in the root of the tree for the attribute F_1. For all three outcomes of the test the resulting subsets are of the uniform class, and thus the tree is 100% accurate for the training data. Notice that these rules do not agree with the rules

we derived intuitively, because only a single attribute can be tested at each tree node.



The above tree can be translated into the set of three rules:

Rule 1: IF (F_1 = normal) THEN class "go home"
Rule 2: IF (F_1 = low) THEN class "go home"
Rule 3: IF (F_1 = high) THEN class "treatment"

Decision tree and rule algorithms always create decision boundaries that are parallel to the coordinate axes of feature values; they create hypercube decision regions in high-dimensional spaces. Cios and Liu (1992) used that fact to design a neural network algorithm that can place decision boundaries at any angle. The Continuous ID3 (CID3) algorithm (Cios and Liu, 1992) is a self-generating neural network algorithm that uses the idea of entropy minimization for placing hyperplanes to solve a given classification problem. During the process of minimizing the entropy, CID3 also generates its own topology. It starts with just one neuron and adds new neurons and/or new hidden layers until a given problem is solved (indicated by the entropy value reduced to zero). Minimization of entropy is used for the "best" placing of separating hyperplanes, in terms of their orientation and position. Unlike ID algorithms that create many classification-boxes, CID3 uses a much smaller number of hyperplanes to achieve the same goal. Top part of Figure 1 illustrates the result of a decision tree algorithm on a two-class (indicated as pluses and minuses), two-feature problem (Cios et al, 1998). CID3 places hyperplanes at any angle, as shown in the lower part of Figure 1. CID3 does not guarantee finding optimal solution, like the one shown in Figure 1, but the number of the generated hyperplanes always is much smaller than the number of parallel hyperplanes generated by ID algorithms.

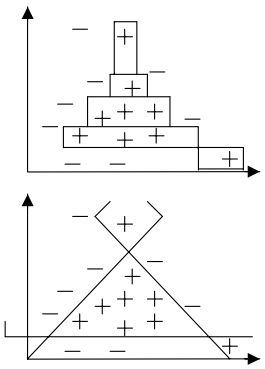


Figure 1. Comparison of decision boundaries created by ID3 and CID3 algorithms

Advantages and disadvantages of decision tree algorithms are listed below:

- They reveal relationships between the rules, which can be derived from the tree. Because of this it is easy to see the structure of the data.
- They produce rules that best describe all the classes in the training data set
- They are computationally inexpensive
- They may generate very complex (long) rules, which are very hard to prune
- They generate large number of rules. Their number can become excessively large unless some pruning techniques are used to make them more comprehensible.
- They require big amounts of memory to store the entire tree for deriving the rules.
- They do not easily support incremental learning. Although ID3 would still work if examples are supplied one at a time, but it would grow a new decision tree from scratch every time a new example is given.

1.5. Hybrid Algorithms

In this section we focus on hybrid algorithms that combine rule and decision tree algorithms. These algorithms incorporate the best ideas from the two families of algorithms as well as new features. Two examples of hybrid algorithms are CN2 algorithm (Clark and Niblett, 1989) and CLILP2 algorithm (Cios at al. 1995, 1997, 2001). Below we briefly overview the CN2 algorithm, however, our main interest is the CLIP family of algorithms.

The main structure of the CN2 algorithm is based on the AQ algorithm and applies some of the mechanisms from decision tree algorithms. The AQ algorithm generates complex that covers the "seed" example and excludes all negative examples. This approach generates rules that are fully consistent with the training

data and thus can overfit the data. The CN2 algorithm performs the search for the complex that does not necessarily exclude all negative examples. It checks all the possible specializations of the complex in a manner similar to the ID3 algorithm's attribute test to establish a new node in the tree. It generates an ordered list of production rules and evaluates the quality of a complex by using the entropy measure

$$Entropy = \sum_{i=1}^{C} p_i \log_2(p_i)$$

where p_i is probability that the complex classifies example into the i^{th} class.

To calculate the entropy, a set of examples covered by the complex has to be found first. The lower the entropy value the better the complex. The second evaluation criterion tests "significance" of the complex by calculating the likelihood ratio statistics. This measure calculates the distance between two distributions: distribution among classes of examples that satisfy the complex, and the expected distribution that would result if the examples were selected randomly. If the value of the significance measure is low then the regularities described by the complex are close to random. The bigger the value of the significance measure the better the complex.

The CN2 algorithm uses these two measures repeatedly to search for the best complex and at the same time it incorporates tree-pruning techniques, like stopping complex specialization when no further specialization is statistically significant. Further improvements of the CN2 algorithm (Clark and Boswell, 1991) include new methods for calculation of the significance measure for the complexes. This new measure uses Laplace heuristics that searches for more general, with higher value of predictive accuracy, complexes. The improved CN2 algorithm is able to generate unordered set of rules, where each rule can be used separately to classify the examples.

2. CLIP Family of Algorithms

The initial CLIP algorithm, CLILP2 (Cover Learning using Integer Linear Programming) was developed in 1995 (Cios and Liu). It was later improved into the CLIP3 algorithm (Cios at al. 1997). Most recently, several new significant features were added to the algorithm that resulted in the CLIP4 algorithm (Cios and Kurgan 2001). The new version of the algorithm incorporates features that make it more powerful and user friendly; they are described later. In this section detailed description of how the CLIP algorithm works is presented, along with an illustrative example.

We explain here the fundamental ideas of the CLIP algorithms. First, a brief description of the CLIP algorithm is provided. Then, the pseudocode of the algorithm is given along with detailed analysis.

The CLIP algorithm has three phases. In the first phase, a decision tree is grown, and pruned, to divide the data into subsets. In the second phase the set covering method is used to generate production rules. In the third phase, goodness of each of the generated rule is evaluated, and only the best rules are kept while the remaining (weaker) rules are discarded. Common feature of all three phases is the use of the Integer Programming (IP) model to perform crucial operations on the data, like selecting the most discriminating features, growing new branches of the tree, selecting the data subset that generates the least overlapping and the most general rules, and finally for generating the rules from the subsets. The use of the IP model is the heart of the algorithm and, as mentioned above, it is used to solve many diverse tasks.

Before the algorithm is used the available data is divided into "positive" data (positive examples), which describe the concept, for which we want to generate the rules, and the "negative" data (counterexamples).

In the first phase, the original set of positive training data is divided into smaller subsets of similar data in a decision-tree like manner. Tree pruning is performed to eliminate noise from the data and to avoid excessive growth of the tree. Each level of the tree is built using one negative example. This negative example is used to define distinguishing features between all positive and this particular negative example, to create new branches of the tree. Each node of the tree represents one data subset. The difference between growing the tree in the CLIP algorithm and a decision tree algorithm, like C4.5, is that the CLIP divides the data in many ways, generating not just one “best” division based on the “best” feature, say in terms of the highest value of the information gain, but a set of divisions based on any feature that distinguishes between positive and negative examples. In this way it generates several data subsets from which the rules can possibly be generated. The goodness of the subsets used for generating the rules is measured by the number of examples they cover. Because of this CLIP can generate more general rules, which is a one of the important features of the algorithm. The question arises what to do when the resulting tree becomes too bushy because of this type of search. CLIP deals with this problem using two techniques. It prunes the subsets that contain very small number of examples (below the predefined threshold) that are suspected to be noisy, and also prunes the redundant subsets, which are defined as subsets that are equal or are subsets of other subsets. One very important feature of the CLIP algorithm is the fact that the tree in fact does not exist; it is a virtual tree. The "tree" at any iteration consists of only one, most recently established tree level. The entire tree above this level is discarded.

In the second phase all terminal subsets (represented by the tree leaves) are judged whether they are good candidates for rule generation or not. There are two criteria used to accept or reject them. The first states that large subsets are preferred over small subsets (because they simply include more examples); thus the rules generated using them will be "stronger" and more general. The second states that all accepted subsets (between them) must cover the entire training data. The second criterion corresponds to the completeness condition, which states that the classification rules must correctly describe all the positive examples. After best subsets are selected, the rules are generated. Each rule is generated using one of the accepted subsets and the entire negative data set. As a result the number of generated rules is equal to the number of accepted subsets. The set of all generated rules covers the entire positive training data set. The algorithm also satisfies consistency condition, which states that every generated rule covers only positive examples and does not cover any negative examples.

In the third phase we are dealing with rules only. The task is to select the best rules from the generated rules. To do it each rule is tested on positive data and the rule that covers the most of them is chosen while the remaining rules are discarded. This promotes selection of strong and general rules. If there is a tie between two or more "best rules" the shortest rule is chosen, i.e. the rule that involves the minimal number of features. After the best rule is selected all the examples covered by this rule are removed from the positive data set and the entire process is repeated on the remaining positive (and the entire negative) data.

CLIP algorithm is able to deal with the noisy data. Noisy examples form small subsets during the first phase of the algorithm's execution and thus weak rules are generated from them. These rules, however, are not accepted in the final phase. Another important feature of the CLIP algorithm, in addition to its noise tolerance, is its ability of dealing with the data containing missing values.

CLIP algorithm uses three thresholds. These thresholds are used to prune the tree, to avoid overfitting of the data by the generated rules (by removing rules that cover too few positive examples, and that are suspected to contain noise), and to decide when to stop forming the rules. Detailed description of the thresholds is given later.

Integer Linear Programming Model

Integer programming models have been used for a long time in the field of operations research, mainly for resource allocation. They are used for minimization or maximization of a function, subject to a large number of constraints. A simple IP model in a standard form is shown in Figure 2 (Ravindran et al. 1987).

$$\begin{gathered}
\textit{Minimize}: \\
x_1 + x_2 + x_3 + x_4 = Z \\
\\
\textit{Subject to}: \\
-2x_1 + 2x_2 + x_3 = 4 \\
3x_1 + x_2 + x_4 = 6 \\
x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0 \\
\\
\textit{Solution}: \\
Z = 4, \quad \textit{when } x_1 = 1, x_2 = 3, x_3 = 0, x_4 = 0
\end{gathered}$$

Figure 2. An example IP model in a standard form and its solution

There are several solutions to an IP model depending on the method used. There are polynomial algorithms (Chvatal, 1979; Hochbaum, 1982) and non-polynomial algorithms (Ravindran et al., 1987).

In the CLIP algorithm the simplified version of the general IP model is used. The following simplifications are used: the function that is subject of optimization has all the coefficient values equal to one, constraint function coefficients have binary values, and all constraint functions are greater or equal to one. This integer linear programming problem is known in the literature as the set-covering problem (Grafinkel and Nembauser, 1972; Bellmore and Ratliff, 1971).

The simplified IP model example used in the CLIP algorithm is shown in Figure 3.

$$\begin{gathered}
\textit{Minimize}: \\
x_1 + x_2 + x_3 + x_4 = Z \\
\\
\textit{Subject to}: \\
x_1 + x_2 + x_3 \geq 1 \\
x_1 + x_4 \geq 1 \\
x_3 \geq 1 \\
\\
\textit{Solution}: \\
Z = 2, \quad \textit{when } x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0
\end{gathered}$$

Figure 3. Simplified IP model used by the CLIP algorithm and its solution

The simplified IP model can be transformed and solved in a matrix representation, see Figure 4.

$$Minimize:$$
$$x_1 + x_2 + x_3 + x_4 = Z$$

$$Subject\ \ to:$$
$$\begin{bmatrix} 1,1,1,0 \\ 1,0,0,1 \\ 0,0,1,0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \geq 1$$

Figure 4. Simplified IP model used in the CLIP algorithm

The solution to the simplified IP model can be found by using only constraint coefficients matrix, which is called BINary matrix. Columns of this matrix correspond to variables of the optimized function (features in case of CLIP). Rows correspond to function constrains (examples in case of CLIP). The solution for the problem is obtained by selecting minimal number of matrix columns in such a way that for every row there will be at least one matrix cell with the value of one for the selected matrix columns. The solution consists of the (binary) matrix of the selected columns. In order to solve the simplified IP model a heuristic was developed; its pseudocode follows.

Given: BINary matrix – a matrix (MxK) where each row represents an example and each column represents a feature

Initialize: SOL = 0, where SOL - a matrix containing solution list (1xK)
BIN_Row - number of rows in BIN

1. Sum each column of BINary matrix one at a time
2. Determine the column that has the largest summed value
3. Put the value of one in the corresponding column in SOL
4. Update BINary matrix and BIN_Row
5. IF BIN_Row > 0 THEN go to 1.

Result: SOL matrix

Calculations for the example shown in Figure 4 follow:

$$initialize \quad SOL = [0,0,0,0]$$

$$BIN = \begin{bmatrix} 1,1,1,0 \\ 1,0,0,1 \\ 0,0,1,0 \end{bmatrix} \rightarrow SOL = [1,0,0,0],\ BIN = \begin{bmatrix} 0,0,1,0 \end{bmatrix} \rightarrow SOL = [1,0,1,0]$$

$$sum \quad \begin{bmatrix} 2,1,2,1 \end{bmatrix} \qquad\qquad\qquad\qquad [0,0,1,0]$$

SOL = [1,0,1,0] means that the solution consists of features 1 and 3.

A detailed explanation of the algorithm follows by means of a numerical example. Consider the postoperative recovery room at a hospital. The doctor needs to make a decision whether the post-operative patient should be prepared for going home

or should undergo further treatment or observation. The doctor can use the following information:

- temperature, in the ranges: low <36°C, mid <36°C, 37°C>and high >37°C
- stability of patient's blood pressure: stable, moderately stable, and unstable
- patient's blood pressure: low < 90/170, mid <90/70, 130/90>, and high >130/90
- patient's level of discomfort: an integer from 1 to 5 (highest).

Available historical data consist of patient information together with doctor's decisions. The goal is to generate the rules that describe doctor's way of making decisions.

The data are shown in Table 2, (the values in the parentheses define integer coding of the data):

Table 2. Example problem data

Patient No	Surface temperature	Blood pressure stability	Last blood pressure	Patient discomfort	Doctor's decision
1	mid (2)	moderately-stable (2)	low (1)	4	home (1)
2	mid (2)	stable (1)	mid (2)	4	home (1)
3	mid (2)	stable (1)	high (3)	5	home (1)
4	high (3)	stable (1)	high (3)	5	home (1)
5	mid (2)	unstable (3)	mid (2)	3	home (1)
6	high (3)	unstable (3)	high (3)	5	treatment (2)
7	mid (2)	moderately-stable (2)	high (3)	4	treatment (2)
8	mid (2)	unstable (3)	low (1)	4	treatment (2)

The training data set for this example consists of eight examples (patients), divided into positive data (first five patients), and negative data (last three patients). Let us transform the above example into the form required by the CLIP algorithm:

M – number of positive examples, N – number of negative examples
K – number of features, which constitute the examples
POS –rectangle (NxK) matrix, which represents the positive training data
NEG –rectangle (MxK) matrix, which represents the negative training data
neg_i – i^{th} example from the negative training data set

Thus for our example we have:
M = 5, N = 3, K = 4

$$POS = \begin{bmatrix} 2,2,1,4 \\ 2,1,2,4 \\ 2,1,3,5 \\ 3,1,3,5 \\ 2,3,2,3 \end{bmatrix}, \quad NEG = \begin{bmatrix} 3,3,3,5 \\ 2,2,3,4 \\ 2,3,1,4 \end{bmatrix}, \quad neg_1 = [3,3,3,5]$$

Note: in what follows we use several terms, namely: matrix, node, and subset interchangeably, since all mean the same.

The high level pseudocode of the CLIP algorithm follows, for details refer to Cios et al. (1997):

Given: POSitive and NEGative training data
Initialize:
Assign POSitive matrix to the tree root.
Initialize the first subset of the positive training data POS_1=POS

Phase I
1. For all NEGative examples neg_i, create new level of the tree for each neg_i:
   1) For every node consisting of POS_j matrix, where j = 1,2…L, L – number of nodes at the current tree level, do:
      - generate BINary matrix by comparing POS_j matrix with neg_i
      - solve represented by the BINary matrix IP model
      - split the POS_j matrix based on the features indicated by the IP solution creating tree nodes for next tree level, which will contain sub-matrices of POS_j matrix and eliminate redundant sub-matrices
   2) Increment index of negative example i=i+1, substitute current tree level with next tree level

Phase II
1. Generate the TM (template matrix) BINary matrix
2. Solve represented by the TM BINary matrix IP model
3. Back project matrices indicated by the IP solution from step 2
4. Convert each resulting matrix to IP model and solve it
5. Generate rules based on generated in step 4 IP solutions

Phase III
1. Find the best rule
2. Eliminate positive examples covered by the best rule from POSitive matrix
3. If POSitive matrix is not empty go to Phase I

Result: Rules covering all positive examples and none of the negative

The solution of the example using the CLIP algorithm is shown in Figure 5.

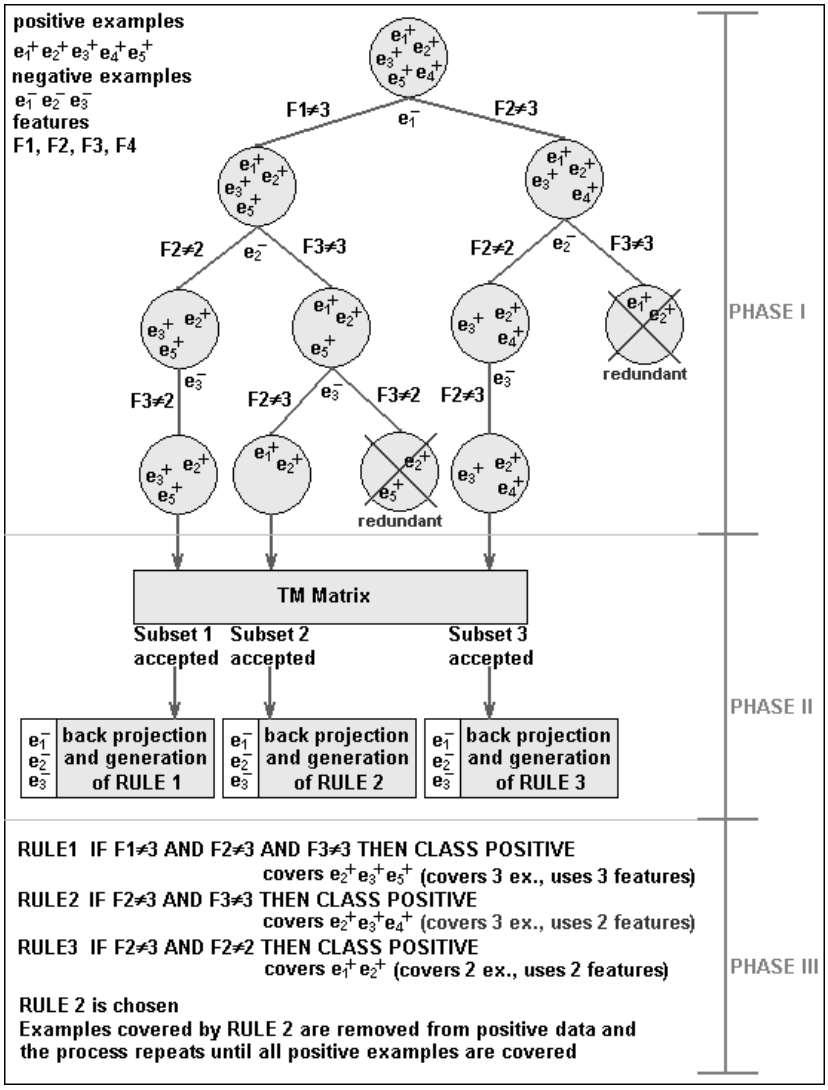


Figure 5. Example problem solution using CLIP algorithm.

Phase I explanation

The goal of phase I is to divide the positive data in a tree-like manner into many subsets. Each subset represents part of positive data, which is recognized as positive and separated from the negative data. The tree consists of nodes, which represent subsets, and branches, which represent subsequent divisions of the data.

Each level of the tree is built using one negative example. The tree has the number of levels equal to the number of negative examples. Each new tree level is built by dividing all the subsets from the previous level by using the same negative example. After new level of the tree is built, the previous level is deleted. As a result, the memory requirements are low because we do not need to store the entire tree but just one tree level. The process is repeated until all negative examples are used. Subsets at the bottom of the tree (leaves) are candidates for rule generation.
Below, the division of the data using the first negative example is shown:

- At the beginning we are at the tree root, which represents the entire positive data stored in matrix POS_1.
- First negative example, neg_1, is used to divide the data
- At this step we calculate the BINary matrix by comparing neg_1 with the entire matrix POS, row by row. If feature values are equal, in both being compared rows, we put a 0 in the corresponding cell in the BIN matrix, else we put a 1

$$POS_1 = \begin{bmatrix} 2,2,1,4 \\ 2,1,2,4 \\ 2,1,3,5 \\ 3,1,3,5 \\ 2,3,2,3 \end{bmatrix}, \quad neg_1 = [3,3,3,5], \qquad BIN = \begin{bmatrix} 1,1,1,1 \\ 1,1,1,1 \\ 1,1,0,0 \\ 0,1,0,0 \\ 1,0,1,1 \end{bmatrix}$$

- Next, the BIN matrix, which represents the IP model, is solved; the solution matrix, SOL, indicates features that can be used to divide the data
- Each 1 in the SOL matrix represents a feature that can be used to distinguish between all positive examples and the neg_i example

$$BIN = \begin{bmatrix} 1,1,1,1 \\ 1,1,1,1 \\ 1,1,0,0 \\ 0,1,0,0 \\ 1,0,1,1 \end{bmatrix} \qquad BIN = [0,1,0,0]$$

$$sum\ of\ ones\ in\ BIN columns = [4,4,3,3] \ \rightarrow \qquad [0,1,0,0]$$

$$SOL = [1,0,0,0] \qquad SOL = [1,1,0,0]$$

- The SOL matrix indicates which features of the negative example, neg_i, will be used to divide the matrix POS_i to create new nodes for the new level of the tree. Each value of 1 in the SOL matrix adds a new branch and a new node to the tree. It accepts examples from matrix POS_i that have the corresponding feature value different from the value of the same feature in neg_i.

- For the SOL matrix $SOL = [1,1,0,0]$, we have two 1s:

- Since the 1 in the first column corresponds to the first feature (patient's temperature) we forbid the corresponding value from the first column of the neg_1 example, because this value can distinguish between positive and negative example
- For condition F1≠3 we create a new branch of the tree and a new node containing subset of the POS matrix examples that satisfies the condition.
- The same mechanism is applied to the value of 1 in the second column of the matrix SOL (i.e. for F2≠3 we create a new branch of the tree and new node containing subset of matrix POS that satisfy the condition).

- In such a way the new level of the tree is created and consists of two nodes; the first contains matrix POS_1, which satisfies the condition F1≠3 and the second containing matrix POS_2, which satisfies the condition F2≠3

$$POS_1 = \begin{bmatrix} 2,2,1,4 \\ 2,1,2,4 \\ 2,1,3,5 \\ 2,3,2,3 \end{bmatrix}, \qquad POS_2 = \begin{bmatrix} 2,2,1,4 \\ 2,1,2,4 \\ 2,1,3,5 \\ 3,1,3,5 \end{bmatrix}$$

- After all the nodes from the preceding level of the tree underwent the above process (in our case it was just the root node), all created nodes constitute the new level of the tree
- The process repeats until all neg_i examples are used exactly once.

Phase II explanation

The goal of phase II is to determine which of the POS_i matrices, represented by the tree leaves, are best candidates for rule generation. First, Template Matrix (TM) is created. This matrix carries information about how many positive examples are covered by each POS_i matrix. The TM matrix is a BINary matrix, and by solving the corresponding IP model, the set of best POS_i matrices for the purpose of rule generation is selected. For each of the selected matrices a Back Projection (BP) matrix is created using the entire negative data set. Once the BP matrix is created, it is transformed into a BINary matrix. Then, the corresponding IP model is solved and the solution, in combination with all negative data, is used to produce the rules.

Below we show calculations performed in phase II:

- Terminal matrices (leaves of the tree) from phase I are shown below:

$$POS_1 = \begin{bmatrix} 2,1,2,4 \\ 2,1,3,5 \\ 2,3,2,3 \end{bmatrix}, \quad POS_2 = \begin{bmatrix} 2,2,1,4 \\ 2,1,2,4 \end{bmatrix}, \quad POS_3 = \begin{bmatrix} 2,1,2,4 \\ 2,1,3,5 \\ 3,1,3,5 \end{bmatrix}$$

- Matrix TM is of size MxL, where L is number of matrices in the terminal tree level
- Each column of the TM matrix corresponds to one terminal POS_i matrix, and each row corresponds to one positive example from the POS matrix. For every example from the terminal POS_i matrix, the corresponding cell in the TM matrix is set to 1 (the cell which is located at the column corresponding to the POS_i matrix, and at the row corresponding to the location of the example in the POS matrix). For all positive examples, which are not present in POS_i, the corresponding TM matrix cells are set to 0.
- Matrix POS_1 consist of examples 2, 3and 5 from matrix POS, matrix POS_2 consist of examples 1, 2 and 5 from matrix POS, and matrix POS_3 consists of examples 2, 3 and 4 from matrix POS. The corresponding TM matrix is shown below.

- Next, the TM matrix, which represents the IP model is solved; the solution matrix, SOL, is found and indicates subsets that are best candidates for rule generation

$$TM = \begin{bmatrix} 0,1,0 \\ 1,1,1 \\ 1,0,1 \\ 0,0,1 \\ 1,0,0 \end{bmatrix} \quad TM = \begin{bmatrix} 0,1,0 \\ 0,0,1 \end{bmatrix} \quad TM = [0,0,1]$$

$$sum\ of\ ones\ in\ TM\ columns = [3,3,3] \rightarrow \qquad [0,1,1] \rightarrow \qquad [0,0,1]$$

$$SOL = [1,0,0] \quad SOL = [1,1,0] \quad SOL = [1,1,1]$$

- The solution matrix SOL = [1, 1, 1] indicates that all three matrices are best candidates for rules generation, since a value of 1 in the SOL matrix indicates that the corresponding matrix is best candidate for rule generation
- For every accepted matrix and entire negative data we generate rules using back projection
- The goal of back projection is to determine selectors that can distinguish between positive examples from matrix POS_i and all negative examples. These selectors are used to generate production rules
- BP matrix is build by comparing each cell from matrix NEG with the entire corresponding column of matrix POS_i . If the value from the being compared cell from matrix NEG is different than any value from the entire corresponding column of the POS_i matrix, then the corresponding cell in the BP matrix is set to the same value as in the NEG matrix cell, otherwise it is set to 0.
- Below we show calculations of the BP matrix for POS_1 (calculations for POS_2 and POS_3 matrices are similar)

$$POS_1 = \begin{bmatrix} 2,1,2,4 \\ 2,1,3,5 \\ 2,3,2,3 \end{bmatrix}, \quad NEG = \begin{bmatrix} 3,3,3,5 \\ 2,2,3,4 \\ 2,3,1,4 \end{bmatrix} \rightarrow BP\ POS_1 = \begin{bmatrix} 3,0,0,0 \\ 0,2,0,0 \\ 0,0,1,0 \end{bmatrix}$$

- BP matrix is transformed into the BIN matrix by replacing all nonzero values with 1.

$$BP\ POS_1 = \begin{bmatrix} 3,0,0,0 \\ 0,2,0,0 \\ 0,0,1,0 \end{bmatrix} \rightarrow BIN = \begin{bmatrix} 1,0,0,0 \\ 0,1,0,0 \\ 0,0,1,0 \end{bmatrix}$$

- Next, the BIN matrix, which represents the IP model is solved, the solution materix SOL indicates features that will be used in the generated rules

$$BIN = \begin{bmatrix} 1,0,0,0 \\ 0,1,0,0 \\ 0,0,1,0 \end{bmatrix} \quad BIN = \begin{bmatrix} 0,1,0,0 \\ 0,0,1,0 \end{bmatrix} \quad BIN = [0,0,1,0]$$

$$sum\ of\ ones\ in\ TM\ columns = [1,1,1,0] \rightarrow \qquad [0,1,1,0] \rightarrow \qquad [0,0,1,0]$$

$$SOL = [1,0,0,0] \quad SOL = [1,1,0,0] \quad SOL = [1,1,1,0]$$

- From the information contained in the SOL matrix a rule is generated. For every column of the SOL vector with a 1, the feature values from the corresponding column of the BP matrix are used to generate selectors that constitute the rule.
- The rule generated from the above back projected POS_1 is:

  IF ($F_1 \neq 3$) AND ($F_2 \neq 2$) AND ($F_3 \neq 1$) THEN class positive

- The same calculations are repeated until all accepted, using the TM matrix, matrices are converted into rules

Phase III explanation

The goal of phase III is to select the best rule from all generated rules. To perform this selection two criteria are used. The first tests positive data against the rules and selects those rules that cover the most positive examples. If there is a tie, the rule that uses less number of features is selected. Then, the positive examples covered by the rule are removed from positive data, and the entire process is repeated. The process is terminated when all positive examples are covered, or if only a small number of positive examples remain uncovered (these examples are suspected to be noisy). The stop threshold specifies this number.

- The rules generated for our example problem are as follows:

RULE 1: IF ($F_1 \neq 3$) AND ($F_2 \neq 2$) AND ($F_3 \neq 1$) THEN class positive
RULE 2: IF ($F_2 \neq 3$) AND ($F_3 \neq 3$) THEN class positive
RULE 3: IF ($F_2 \neq 3$) AND ($F_2 \neq 2$) THEN class positive

- In the first step the rules are checked against the training data

RULE 1 covers three patients 2,3 and 4
RULE 2 covers two patients 1 and 2
RULE 3 covers three patients 2,3 and 4

- The rules that cover the most positive examples are selected:

RULE 1 and RULE 3 are selected

- If there is more than one rule selected then the most compact rule is selected

RULE 1 uses features F_1, F_2, and F_3
RULE 3 uses only feature F_2
Thus, RULE 3 is selected

- If still there is a tie, then any rule, say the first one, is selected

- After the best rule is selected, the examples that the rule covers are removed from matrix POS and the process repeats on the smaller positive data

$$POS = \begin{bmatrix} 2,2,1,4 \\ 2,3,2,3 \end{bmatrix} \quad NEG = \begin{bmatrix} 3,3,3,5 \\ 2,2,3,4 \\ 2,3,1,4 \end{bmatrix}$$

The finally generated rules are:

RULE 1: IF ($F_2 \neq 3$) AND ($F_2 \neq 2$) THEN class positive
RULE 2: IF ($F_2 \neq 3$) AND ($F_3 \neq 3$) THEN class positive
RULE 3: IF ($F_3 \neq 3$) AND ($F_3 \neq 1$) THEN class positive

which can be transformed into:

RULE 1: IF ($F_2 = 1$) THEN class positive
RULE 2: IF ($F_2 = 1$ OR $F_2 = 2$) AND ($F_3 = 1$ OR $F_3 = 2$) THEN class positive
RULE 3: IF ($F_3 = 2$) THEN class positive

For our example the rules read:

RULE 1:
IF (patient's stability of blood pressure is STABLE)
THEN go home
RULE 2:
IF (patient's stability of blood pressure is STABLE or MODERATELY-STABLE)
AND (patient's blood pressure is LOW or MID)
THEN go home
RULE 3:
IF (patient's blood pressure is MID) THEN go home

The rules generated for the training data, Table 1, are shown below. The CLIP algorithm generated 2 rules, one rule less than decision trees. The rules have 100% accuracy for the training data. Notice that these rules do not agree with the rules we derived intuitively, but they generalized the information from the training data, by finding that a single attribute is sufficient to describe the data.

RULE 1. IF ($F_1 \neq$ high) THEN go home
RULE 2. IF ($F_1 \neq$ normal) AND ($F_1 \neq$ low) THEN treatment

The comparison of all three types of ML algorithms on the training data from Table 1 is shown in Table 3. The decision trees have lower computational cost but the CLIP algorithm has much lower memory requirements. Also CLIP generates smaller number of rules.

Table 3. Comparison of the ML algorithm on the training data from Table 1

Algorithm	# of rules	attributes used	# of selectors used	Memory requirements	Computational cost
Rule Algorithm	2	F_1, F_2	4	high	high
Decision trees	3	F_1	3	moderate	low
CLIP algorithm	2	F_1	3	low	moderate

Below we describe the three thresholds used by the CLIP algorithm. One prunes nodes containing small number of positive examples, the second determines if the best generated rule is acceptable, and the third determines when to stop forming the rules. These thresholds control the complexity and number of generated rules. The user can use default values for all the thresholds: Noise Threshold, Best Rule Threshold, and Stop Threshold.

- Noise Threshold (NT) determines which nodes (possibly containing noisy positive examples) will be pruned from the virtual tree grown in the phase I of the algorithm. The threshold will prune every node that contains fewer examples than NT. Thus, if NT is 0% then none of the branches are eliminated.
- Best Rule Threshold (BRT) determines, which rules can be chosen as best rules. Only the rules that cover more than the BRT percentage of the (remaining) positive examples can be chosen as best rules.
- Stop Threshold (ST) determines when to stop the algorithm. The algorithm will be terminated when a smaller than ST percentage of positive examples remains uncovered (these examples are suspected to be noisy and probably would produce very specific rules).

Either the BRT or the ST threshold can stop the algorithm.

There are several advantages of the CLIP algorithm:

- It generates very **compact** rules, measured in terms of a small number of features used
- It generates **small** number of rules that describe the concepts from training data
- The **generalization** ability of the rules is high. Since the rules often overlap the performance on unseen validation data is higher. In most cases, by using default values of thresholds, the CLIP algorithm generates the rules that do not overfit the data.
- **Balance** between generalization and specialization of the generated rules can be controlled by the thresholds
- CLIP **deals with noisy data** without overfitting. The Stop Threshold can be used to remove noisy data from the training data.
- **Memory requirements are very low**, because there is no need to store the entire decision tree during training, like in case of decision tree algorithms. Only the bottom level of the tree is needed to generate the rules.
- CLIP splits positive data into many subsets to later generate the best rule. The partitioning mechanism used in the CLIP algorithm **prevents generation of rules that cover only a small number of examples.**

The disadvantages are:

- It does not support incremental learning. It generates rules when examples are supplied one at a time, but when a new example is provided then it generates the rules from scratch.
- It works with discrete features only.

The new version of the algorithm, CLIP4 (Cios and Kurgan, 2001) added these improvements and new features:

- Handling of missing-value data
- Improved tree-pruning methods
- Improved methods for solving IP model
- Application of evolutionary computation methods to improve partitioning data into subsets
- Addition of 3 front-end discretization algorithms to deal with continuous data
- Acceptance of many rules (not just one) in phase III of the algorithm
- Automatic generation of rules for multi-class problems
- Front-end nominal data encoding and decoding
- Analysis of the feature-value pairs using the CLIP4 rules. Each feature-value pair has assigned a goodness measure that quantifies its strength and usefulness.
- Calculation of confidence factors for each classification made by using the generated rules
- User-friendly, windows-based, implementation

CLIP4 algorithm can be used for data mining purposes because of its efficiency; the reader can download CLIP4's executable code from http://isl.cudenver.edu (under "Software").

In the next section, we summarize results of CLIP3 and CLILP2 algorithms, as well as comparison with other machine learning algorithms. The results show that the CLIP algorithm generates very accurate rules.

2.1. Results and comparison with other algorithms

The lymphatic cancer, breast cancer, and primary tumor data

CLILP2 algorithm was tested on the data from three medical domains: lymphatic cancer, prognosis of breast cancer recurrence, and location of primary tumor (Kononenko at al., 1984; Michalski, 1990, Clark and Niblett, 1989)

Lymphatic cancer data has 4 decision classes and 18 attributes, with 148 examples; the data is consistent. Prognosis of breast cancer data has 2 decision classes and 9 attributes with 186 examples; the data is inconsistent - some examples belonging to two different classes are identical. Location of primary tumor data has 22 classes and 17 attributes with 339 examples; the data is inconsistent.

All data sets were divided into training and validation parts in the same manner as reported in the literature: 70% were randomly selected for training and remaining 30% for testing.

Two values were calculated: accuracy and complexity. Complexity is calculated by counting the number of nodes, in case of the decision-tree based algorithms, and the number of complexes generated, in case of the rule-based algorithms.

The results shown in the Table 4 are repeated after Cios and Liu (1995).

As can bee seen, the rules obtained by the CLILP2 algorithm have the highest accuracy and comparable complexity on the lymphatic cancer and breast cancer data, and comparable accuracy and complexity on primary tumor location data.

Table 4. Diagnostic accuracy and complexity results comparison.

	Lymphatic cancer		Breast cancer		Primary tumour	
Algorithm	Accuracy [%]	Complexity	Accuracy [%]	Complexity	Accuracy [%]	Complexity
ASSISTANT						
No pruning	76	38	67	120	41	188
Pruning	77	25	72	16	46	35
Bayes	83	-	65	-	39	-
AQR	76	76	72	208	35	562
CN2						
90% threshold	78	24	70	23	37	33
95% threshold	81	22	70	20	36	42
99% threshold	82	12	71	4	36	19
AQTT-15						
Complete	81	12	66	41	39	104
Unique >q	80	10	68	32	41	42
Top rule	82	4	68	2	29	22
CLILP2						
No discard	85	18	76	40	31	108
Discard	84	16	-	-	37	83

Discrete MONK's data

The CLIP3 algorithm was compared with several other algorithms using MONK's problems (Thrun et al., 1991). The MONK's problem has 432 examples and 6 multi-value attributes. It is divided into 3 separate problems: M1 (positive and negative data sets includes 216 examples each, training data set of 124 examples was randomly chosen), M2 (142 positive examples and 290 negative examples, training data set of 189 examples was randomly chosen), M3 (156 positive examples and 276 negative examples, training data set of 122 examples was randomly chosen, 5% of examples were misclassified to induce noise into data). The MONK's data is an artificially created dataset. It defines a set of robots described by 6 features including: head shapes, body shapes, facial expressions, objects being held, jacket colors, and whether or not the robot is wearing a tie. The seventh feature is a decision attribute (whether a robot belongs to a positive or a negative class). This produces a domain of 432 unique robots. The training sets used were exactly the same as those used in (Thrun et al., 1991).

During training the CLIP3 algorithm used the following values for the thresholds for all data sets: NT = 1 or NT = 2 (all nodes containing less then NT examples were pruned), ST = 10 (The training was terminated when less then ST examples remained uncovered), and BRT = 50% (only rules that cover more then 50% of uncovered positive examples can be accepted).

The MONK's data were coded in the following manner: F1, head shape (1 = round, 2 = square, 3 = octagon); F2, body shape (1 = round, 2 = square, 3 = octagon); F3, smiling (1 = yes, 2 = no); F4, object holding (1 = sword, 2 = flag, 3 = balloon); F5, jacket color (1 = red, 2 = yellow, 3 = green, 4 = blue); and F6, has tie (1 = yes, 2 = no). For example, for the data set M1, a positive learning example (+) is a square headed, octagon bodied, smiling, balloon holding, red jacketed, and tie-less robot, was coded as (+ 2 3 1 3 1 2). The results shown in Table 5 are repeated after Cios et al. (1997).

On this test the CLIP3 algorithm performed better then other algorithm. It also generated fewer rules than any other algorithm with very high or the highest accuracy.

Table 5. Diagnostic accuracy and the number of generated rules comparison.

	M1		M2		M3	
Algorithm	Accuracy [%]	No of rules	Accuracy [%]	No of rules	Accuracy [%]	No of rules
CLIP3 (NT=1)	100	4	82.7	10	88.9	3
CLIP3 (NT=2)	100	4	72.7	7	97.2	2
ID3						
without Windowing	83.2	62	69.1	110	95.6	31
with Windowing	98.6	28	67.9	110	94.4	29
ID5R	79.8	52	69.2	99	95.3	28
AQR	95.9	36	79.6	83	87.0	36
CN2	100	10	69.0	58	89.1	24
C4.5 Decision Trees	75.7	*	65.0	*	97.2	*
C4.5 Tree Rules	100	*	65.3	*	96.3	*
C4.5 Trees with –S	100	*	70.4	*	100	*
C4.5 -S Tree Rules	100	*	67.1	*	100	*

* data not available

The breast cancer data

The CLIP3 algorithm was also tested on the Breast Cancer Data (Mangasarian and Wolberg, 1990). This data has 683 examples, 10 features, and two classes. The features have an integer value from 1 to 10. The early results on this data set cannot be compared because the data were continuously expanding by adding new examples. For instance an algorithm run in 1991, when the database contained 367 examples, cannot be compared to an algorithm run in 1995 when the database contained 683 examples.

The training set was generated by choosing every fifth point from the data. CLIP3 was run with ST of 0, 1and 2. The same data set was used to generate rules with the C4.5 algorithm with the -u option and the -s option. The results shown in Table 6 are repeated after Cios et al. (1997).

Table 6. Breast cancer results for CLIP3 and C4.5 algorithms

Algorithm	Accuracy [%]
CLIP3 (ST=0)	89.6
CLIP3 (ST=1)	86.8
CLIP3 (ST=2)	92.4
C4.5 (-u option)	89.3
C4.5 (-s option)	90.1

2.2. Other applications

The CLIP algorithm was also applied to several other problems, like:

- Generation of diagnostic rules to describe patients with coronary artery stenosis (Cios et al., 1993). This problem involved a database containing thallium-201 scintigraphic studies on 91 patients. The problem was to recognize typical patterns in the coronary artery stenosis data and compare rules generated by the CLILP2 algorithm with existing diagnostic expert system rules. The CLILP2 generated 15 as compared with 68 expert-specified rules. Accuracy of the rules generated by CLILP2 algorithm was between 89% and 96%. The rules generated using CLILP2 algorithm were in the same format as rules specified by the experts, thus showing that for domains where training data is available there may be no need to extract rules from experts.
- Problem of designing end-user satisfaction instrument (Torkzadeh et al., 1996). This problem involved data collected for development of an end-user computing satisfaction instrument. The data set included 618 cases, 12 attributes, and 2 target classes (dissatisfied and satisfied). The problem was to analyze the usefulness of the instrument, according to number of attributes used, and to create a shorter instrument with comparable end-user satisfaction factors. As a result the five-item vs. original twelve-item instrument was developed using CLILP2 algorithm with the average accuracy of 86.5% and higher then accuracy for the original instrument (75.5%).
- Problem of generation of diagnostic rules from SPECT Bull's-eye maps (Cios et al., 2000). This application was done using the database containing SPECT Bull's-eye heart studies on 184 patients. This is a two-classes problem: one class describes diagnoses for normal patients (160 examples), second for patients with coronary artery disease (24 examples). The goal of this investigation was to generate a set of rules, which can correctly recognize given example into one of these two classes, and to compare existing expert rules with the rules generated by CLIP3. As a result 14 diagnostic rules were generated.
- Generating diagnostic rules from cardiac SPECT data (Kurgan at al., 2001). This problem involved database containing cardiac SPECT heart images collected on 267 patients in stress and rest studies. CLIP3 algorithm was

applied to generate diagnostic rules for overall diagnosis of the patient's study, by using information of partial, in the predefined regions of the heart muscle, diagnoses. This is a two-classes problem: first class describes normal patients (55 examples), and second patients with coronary artery disease (212 examples). Three diagnostic rules were generated. The rules accuracy was 84%.

3. Concluding Remarks

We described three major families of ML algorithms that can generate production rules from the data. The descriptions were supplemented by self-explanatory, easy to understand examples that gave good insight into the described methods. In addition, we described discretization methods, hypothesis evaluation methods, and the problem of overfitting in the appendices. Because the chapter concentrated on the CLIP algorithms we omitted many other ML algorithms like CART (Breiman et al., 1984), S-Plus tree (Clark and Pregibon, 1993), FACT (Loh and Vanichsetakul, 1988), QUEST (Loh and Shih, 1997), IND (Buntine, 1992), OC1 (Murthy, Kasif and Salzberg, 1994), LMTD (Brodley and Utgoff, 1995), T1 (Holte, 1993), etc.

The CLIP family of algorithms has been developed over several years. The most recent is the CLIP4 algorithm, an efficient tool that can be used to generate production rules from numerical, nominal, and continuous data. The rules have high accuracy when tested on unseen data, and the computational cost is acceptable. The tests with large, over 40K examples, data using the CLIP4 algorithm were very successful.

References

Bellmore, M and Ratliff, H.D., 1971, Set Covering and Involuntary Bases, *Management Science,* vol. 18, no 3

Breiman L., Friedman J., Olse R., and Stone C., 1984, *Classification and Regresion Trees*, Chapman and Hall, New York, NY

Brodley C.E., Utgoff P.E., 1995, Multivariate decision trees, *Machine Learning,* 19:45-77

Buntine W., 1992, Learning classification trees, *Statistics and Computing*, 2:63-73

Chvatal, V, 1979, A Greedy-Heuristic for The Set-Covering Problem, *Math. Operations Research 4*, no 3

Cios, K.J. and Liu, N., 1992, Machine Learning in Generation of a Neural Network Architecture: a Continuous ID3 approach, *IEEE Trans. on Neural Networks*, 3(2): 280-291

Cios, K.J., Liu, N., Goodenday, L.S., 1993, Generation of Diagnostic Rules via Inductive Machine learning, *Kybernetes*, vol. 22, no 5, pp 44-56

Cios, K.J. and Liu, N., 1995, An algorithm which learns multiple covers via integer linear programming. Part I - The CLILP2 Algorithm, *Kybernetes*, vol. 24, no 2, pp. 29-50

Cios, K.J. and Liu, N., 1995, An algorithm which learns multiple covers via integer linear programming. Part II – experimental results and conclusions, *Kybernetes*, vol. 24, no 3, pp. 24-36

Cios, K.J., Wedding, D.K. and Liu, N, 1997, CLIP3: cover learning using integer programming. *Kybernetes*, 26(4-5): 513-536 (*The Norbert Wiener 1997 Outstanding Paper Award*)

Cios, K.J., Pedrycz, W., Swiniarski, R., 1998, Data Mining Methods for Knowledge Discovery, Kluwer

Cios, K.J., Teresinska, A., Konieczna, S., Potocka, J., Sharma, S., 2000, Diagnosing Myocardial Perfusion from SPECT Bull's-eye Maps - A Knowledge Discovery Approach, *IEEE Engineering in Medicine and Biology Magazine*, Special issue on Medical Data Mining and Knowledge Discovery

Cios, K.J., & Kurgan, L., Hybrid Inductive Machine Learning Algorithm that Generates Inequality Rules, *Information Sciences*, Special Issue on *Soft Computing Data Mining*, submitted, 2001

Clark, P. and Niblett, T., 1989, The CN2 algorithm, *Machine learning*, 3:261-283

Clark, P. and Boswell, R., 1991, Rule Induction with CN2: Some Recent Improvements, *Lecture Notes in Artificial Intelligence,* Proc. of European Working Session on Learning, Springer-Verlag

Clark L.A., Pregibon D., 1993, Tree-based models, in Chambers J.M., Hastie T.J. (ed.), *Statisticsl Models in S*, Chapman and Hall, New York, NY, p.377-419

Fisher, D.H. and Langley, P., 1986, Conceptual clustering and its relation to numerical taxonomy, *Artificial Intelligence and Statistics*, Gale W.A. (ed), Addison-Wesley

Fisher, D.H., 1987, Conceptual clustering, learning from examples and inference, *Proc. of the 4th Int. Workshop on Machine learning*, Morgan-Kaufmann, pp.38-49

Grafinkel, R.S. and Nembauser, G.L., 1972, *Integer Programming,* John Wiley & Sons, New York

Hochbaum, D.S., 1982, Approximation Algorithm for The Weighted Set-Covering and Vertex Cover Problems, *Siam J. Comput.*, vol. 11, no 3

Holte, R.C., 1993, Very simple classification rules perform well on most commonly used data sets, *Machine learning*, 11:63-90

Hunt, E.B., Marin, J. and Stone, P.J., 1966, *Experiments in Induction,* Academic Press

Kaufman, K.A. and Michalski, R.S., 1999, Learning from Inconsistent and Noisy Data: The AQ18 Approach, *Proc. of the Eleventh International Symposium on Methodologies for Intelligent Systems*, Warsaw, June 8-11

Kononenko, I., Bratko, L., Roskar, E., 1984, Experiments in automatic learning of medical diagnostic rules, Technical Report, E. Kardelj University, Faculty of Electrical Engineering, Ljubljana, Yugoslavia

Kurgan, L.A., Cios, K.J., Tadeusiewicz, R., Ogiela, M., Goodenday, L.S., 2001, Knowledge Discovery Approach to Automated Cardiac SPECT Diagnosis, *Artificial Intelligence in Medicine*, 23:2, pp.149-169

Langley, P., 1996, *Elements of Machine learning*, Morgan-Kaufmann

Loh W.Y., Shih Y.S., 1997, Split selection methods for classification trees, *Statistical Sinica*, 7:815-840

Loh, W.Y., Vanichsetakul, 1988, Tree structured classification via generalized discriminant analysis (with discussion), *Journal of the American Statistical Association,* 83:715-728

Mangasarian, O. L. and Wolberg, W. H., 1990, Cancer Diagnosis Via Linear Programming, *Siam News*, vol. 23, no 5, pp. 1-18

Michalski, R.S., 1974, Variable-valued logic: System VLl, *Proc. 1974 Int. Symposium on Multiple-Valued Logic and Pattern Recognition,* West Virginia University, Morgantown, pp. 323-346

Michalski, R.S., 1980, Knowledge acquisition through conceptual clustering: a theoretical framework and algorithm for partitioning data into conjunctive concepts, *Int. Journal of Policy Analysis and Information Systems*, 4:219-243

Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N., 1986, The multipurpose incremental learning system AQ15 and its testing application to three medical domains, *Proc. 5th National Conf. on Artificial Intelligence,* Morgan-Kaufmann, pp. 1041-1045

Michalski R.S., 1990, Learning flexible concepts: fundamental ideas and a method based on two-tiered representation, Kodratoff, Y. and Michalski, R.S., (eds), *Machine learning: An Artificial Intelligence Approach*, Morgan-Kaufmann, vol. III, pp. 63-102

Murthy S.K, Kasif S., Salzberg S., 1994, A system for induction of oblique decision trees, *Journal of Artificial Intelligence Research,* 2:1-33

Ravindran, A., Phillips, D.T., Solberg, J.J, 1987, Operations Research, Principles and Practice, Second addition, pp. 1-69, John Wiley & Sons.

Quinlan, J.R., 1993, C4.5 Programs for Machine learning, Morgan-Kaufmann

Schapire, R. E., (1999), A brief introduction to boosting**.** *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*

Shank R.C., Where is AI, (1991), *AI Magazine*, winter 1991, p.38-49

Thrun, S.B., et al., 1991, The MONK's problems: A Performance Comparison of Different Learning Algorithms, School of Computer Science, Carnegie Mellon University

Torkzadeh, G., Cios, K.J., Pfughoeft, K.A, 1996, Inductive machine learning for instrument development, *Information and Management*, vol. 31, pp 47-55

University of California, Irvine: UCI Machine learning Repository Content Summary. http://www.ics.uci.edu/~mlearn/MLSummary.html

Appendix 1: Discretization

Most machine learning algorithms accept only discrete feature values. In order to work with continuous features the CLIP4 algorithm (Cios and Kurgan, 2001) provides three front-end discretization schemes: equal frequency, Paterson-Niblett (Paterson and Niblett, 1982), and Class-Attribute Interdependence Uncertainty and Redundancy (CAIUR) based on the Class-Attribute Interdependence Redundancy (CAIR) algorithm (Wong and Liu, 1975). Description of the three schemes is given below.

Discretization is defined as a process of transforming a continuous range of values of a feature, into a finite number of intervals, and associating with each interval a discrete value. The quality of discretization is a one of the key factors that determine performance of a subsequently used algorithm, like the CLIP algorithm. There are two ways to include discretization capability into a machine learning algorithm. One is to include a discretization scheme as a part of the algorithm itself. The other, widely used, is to perform discretization as a front-end operation. Below we address the latter. For more detailed discussion about different discretization schemes see Cios et al. (1998).

In general, the discretization process consists of two steps: deciding the number of discrete intervals, and determining the width of these intervals. There are some heuristics used to choose the number of intervals. One says that the number of intervals should not be smaller than the number of classes; the other specifies the number of intervals, n, using M – number of learning examples and C – number of classes (Ching at al., 1995):

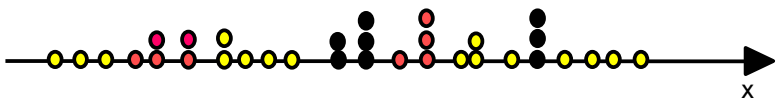
$$n = M / (3* C)$$

We assume that the user supplies the number of intervals for each feature: $N = \{n_{F1}, \ldots, n_{Fi}, \ldots, n_{Fn}\}$

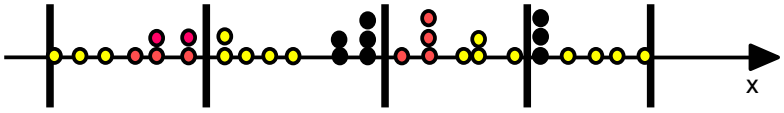
The ***equal frequency discretization scheme*** sorts the values of each discretized feature F_i in an ascending order. Then those values are divided into the user-specified number, n_{Fi}, of intervals, in such a way that each interval contains the same number of sorted feature values.

Example:

c – number of classes, c = 3 (represented by three different colors), M = 33,
n – number of discretization intervals, n = M / (3*c) = 33 / (3*3) = 4
the X axis – represents values of feature F_i



- Equal frequency discretization scheme solution
  Number of values per one interval = 33 / 4 = 8



Equal frequency discretization scheme is very simple. For most problems it produces satisfactory results. For harder problems, the user can use in the CLIP4 algorithm CAIUR or Paterson-Niblett discretization schemes. These schemes take into account relationship between feature values and target class values, minimizing the number of classes assigned to all values of the discretized feature in each interval. What is important, the CAIUR and Paterson-Niblett methods automatically assign proper number of discretization intervals. Thus, these schemes perform pre-classification for the discretized features, which improves the accuracy of the subsequently used learning algorithms. The price to pay is that the two methods are computationally expensive, especially for problems with large number of examples.

The machine learning algorithm's task is to discover the relationship between the class variable (conclusion of a rule) and the feature variable (condition of a rule). Thus, the discretization problem, especially in case of the CAIUR scheme, is formalized in view of the class-feature interdependence. Assume that the problem is described by n features $F_1,\ldots, F_j, \ldots, F_n$ and there are C classes, c_i , i = 1,…, C. Let the interval [a, b] be the range of the continuous-valued feature F_j. A *partition* T_j on F_j is defined as:

$$T_j: \{[e_0,e_1], (e_1,e_2], \ldots, (e_{Lj-1},e_{Lj}]\}$$

where e_0 = a represents the lowest observed value; e_{Lj} = b is the upper boundary value, and $e_{r-1} < e_r$ for r = 1, . . ., L_j, where L_j is the number of intervals. With the change of partition T_j, the class variable, c, and the interval variable, denoted as $v_{jr} = (e_{r-1}, e_r]$, can be understood as two random variables.

Paterson-Niblett discretization scheme

As said above the second discretization scheme used in the CLIP4 algorithm is based on the work of Paterson and Niblett (1982). Their discretization scheme can be formalized in the following manner.

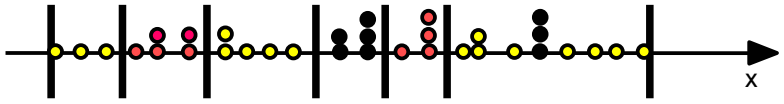
Given: Feature F_i that has n continuous values

1. All values of F_i are sorted
2. For all possible divisions on feature F_i (each division is created by adding a one division boundary N):
    a. the entire interval of F_i values is split, on value of added boundary N, into two intervals: those for which $F_i \leq N$ and those for which $F_i > N$.
    b. the value *information gain* is computed
3. The boundary corresponding to the largest value of *information gain*, or *gain ratio*, is added
4. If a stop criteria is satisfied then stop, else go to step 2.

Result: The set of discretization intervals for feature F_i

In case of this scheme, as well as most of other discretization schemes, candidate division boundary points are set as all the midpoints between all two adjacent values of a continuous feature. As the stop criterion one can use a threshold value for the maximal *information gain*, or *gain ratio* value established in step 2. The other way to define the stop criterion is to stop adding new intervals when the difference between the value of *information gain* in the previous iteration and the current iteration is small (below a specified threshold), which means that there will be no significant improvement in the class-feature interdependence by adding new interval (adding new decision boundary).

Example:
The boundaries established by using Paterson-Niblett scheme are shown below:



CAIUR discretization scheme

In order to explain this scheme we need several definitions (Ching et al., 1995, Cios et al., 1998).

A set of *boundary points* is defined as the set of ordered end points $e_0, e_1, \ldots, e_{Lj}$ that define the L_j intervals. Let Q_j denote a set of 2D frequency *quanta matrix* such that:

$$Q_j: \{q_{ir} \mid i = 1, \ldots, C;\ r = 1, \ldots, L_j\}$$

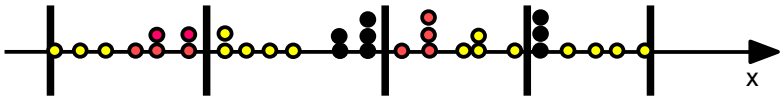
where q_{ir} is the number of examples from the i-th class in the r-th interval.

An example of quanta matrix is shown in Table 7.

Table 7. Discretization quanta matrix for feature F_j

Class (c)	Intervals					Total
	$[e_0,e_1]$	. . .	$(e_{r-1},e_r]$	. . .	$(e_{Lj-1},e_{Lj}]$	
C_1	q_{11}	. . .	q_{1r}	. . .	q_{1Lj}	q_{1+}
.	.	.	.	.	.	.
c_i	q_{i1}	. . .	q_{ir}	. . .	q_{iLj}	q_{i+}
.	.	.	.	.	.	.
c_C	q_{C1}	. . .	q_{Cr}	. . .	q_{CLj}	q_{C+}
Total	Q_{+1}	. . .	q_{+r}	. . .	q_{+Lj}	M

Example:



For the partition described by the above example, the corresponding quanta matrix is defined as:

Classes	Intervals				Total
	First	Second	Third	Fourth	
Class 1	3	5	4	4	16
Class 2	5	0	4	0	9
Class 3	0	5	0	3	8
Total	8	10	8	7	33

where the total number of objects, M, is: $M = \sum_{r=1}^{L_j} q_{+r} = \sum_{i=1}^{C} q_{i+}$

and q_{+r} is the number of objects in the r-th interval: $q_{+r} = \sum_{i=1}^{C} q_{ir}$

and q_{i+} is the number of objects in the i-th class: $q_{i+} = \sum_{r=1}^{L_j} q_{ir}$

The estimated joint probability of the event that an object belongs to class c_i while its feature value F_j falls within the interval v_{jr} is defined as:

$$p_{ir} = \frac{q_{ir}}{M}$$

and the marginal probability of c = c_i is defined as:

$$p_{i+} = \frac{q_{i+}}{M}$$

while the marginal probability of $F_j \in v_{jr}$ is defined as:

$$p_{+r} = \frac{q_{+r}}{M}$$

The Class-Attribute (CA) Mutual Information (CAMI) between the class variable c and the feature interval boundaries of F_j, from the associated quanta matrix Q_j, is defined as:

$$I(c : v_j) = \sum_{c} \sum_{v_j} p_{ir} \log_2 \frac{p_{ir}}{p_{i+} p_{+r}}$$

The CA Information (CAI) between the class variable and the feature F_j interval variable, from its associated quanta matrix, Q_j, is defined as:

$$INFO(c : v_j) = \sum_{c} \sum_{v_j} p_{ir} \log_2 \frac{p_{+r}}{p_{ir}}$$

The Shannon's entropy of the quanta matrix measures the randomness of the distribution of data points with respect to class variable, and interval variable, v_j, and is defined as:

$$H(c : v_j) = \sum_{c} \sum_{v_j} p_{ir} \log_2 \frac{1}{p_{ir}}$$

The Class-Attribute Interdependence Redundancy (CAIR) was introduced by Wong and Liu (1975). It is the CAMI normalized by entropy H. CAIR measure was defined as:

$$R(c : v_j) = \frac{I(c : v_j)}{H(c : v_j)}$$

To explore the CA interdependence relationship Class-Attribute Interdependence Uncertainty (CAIU) is defined as the CAI normalized by entropy H:

$$U(c : v_j) = \frac{INFO(c : v_j)}{H(c : v_j)}$$

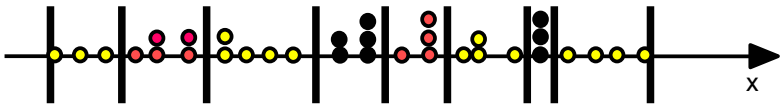
The goal of the CAIUR scheme is to maximize the interdependence between class labels and the attribute variables, and at the same time minimize the number of intervals.

In order to define CAIUR scheme, we need to define the following:

- Initial discretization
  All possible boundary points are set as candidates for the optimal interval scheme. Then their number is reduced by elimination. Candidate boundary points are set as all the midpoints between any two nearby values of a continuous feature.
- Criteria for a discretization scheme
  The CAIR and the CAIU are used as discretization criteria.

CAIUR scheme maximizes the interdependence relationship, and at the same time minimizes the number of intervals, and keeps the loss of information as small as possible. In order to achieve this CAIR criterion should be maximized, and CAIU criterion should be minimized.

Example:
The boundaries established by using the CAIUR scheme are shown below:



Appendix 2: Hypothesis evaluation

The generated hypotheses are verified by checking their ability to generalize on unseen test data. The decision rules (hypotheses) should recognize all positive test examples, for which they were generated, and none of the negative test examples.

In order to evaluate goodness of the generated hypotheses two measures are described below: accuracy test, and verification test (Cios et al., 1998). Assuming that the hypotheses were generated from large training data (if training data is small then cross-validation or bagging should be used to generate the hypotheses – see Appendix 3 on overfitting) these two measures provide good verification results.

Accuracy test

Machine learning algorithms are generally tested using the accuracy test. A test example is classified by matching with the rule (hypothesis) that describes it best. This way the example's class membership is determined.

An accuracy test is simply defined as:

$$accuracy = \frac{TP}{total} 100\%$$

where TP – (*true positive*), the number of correctly recognized test examples,
total – total number of test examples.

A test example is checked against the all rules describing all classes, row-wise.

Verification test

The accuracy test gives only very general information about the "goodness" of the generated hypotheses. The verification test, which is frequently used in evaluating medical diagnostic procedures, gives much better and very specific information about goodness of the generated rules.

The verification test consists of three evaluation criteria:

$$sensitivity = \frac{TP}{hypothesis \quad positive} 100\% = \frac{TP}{TP + FN} 100\%$$

$$specificity = \frac{TN}{hypothesis \quad negative} 100\% = \frac{TN}{FP + TN} 100\%$$

$$predictive \quad accuracy = \frac{TP + TN}{total} 100\% = \frac{TP + TN}{TP + TN + FP + FN} 100\%$$

where: TP (*true positive*) – number of correct positive classifications
TN (*true negative*) – number of correct negative predictions
FP (*false positive*) – number of incorrect positive predictions
FN (*false negative*) – number of incorrect negative predictions

Possible outcomes of a test are:

	Test result positive	Test result negative
Hypothesis positive	TP	FN
Hypothesis negative	FP	TN

The predicative accuracy is equivalent to accuracy in accuracy test. The remaining two criteria give very good insight into the goodness of the generated rules. The sensitivity measures how many of the examples classified by the rules as positive

were truly positive. The specificity measures how many of the examples classified by the rules as negative were truly negative. In this way we get the feeling how well the generated rules can perform on the positive and negative data separately. This is very important when the numbers of positive and negative examples are very different. Then, the accuracy measure provides just the average result for positives and negatives together, when truly accuracy for positives can be very different then accuracy for the negatives; this can be easily noticed while using sensitivity and specificity measures. Only the results with high values for all three measures can assure high confidence level in the generated hypotheses.

The CLIP4 algorithm (Cios and Kurgan, 2001) calculates accuracy, sensitivity, and specificity during rules validation. The results are given for each class separately, as well as for the entire test data.

Appendix 3: Overfitting

Overfitting, or overtraining is defined as a tendency of a learning method to agree with the training data too closely, in order to correctly describe all of the training examples (Cios et al., 1998). This phenomenon occurs in the learning methods such as neural networks, machine learning, and even statistics, and can lead to generation of overfitted weights in case of neural networks, or very specific rules in case of inductive machine learning. The overfitting causes that the trained network or the generated rules may achieve very poor results on unseen test data. Additionally, if the learning data contains inaccuracies, like noise-corrupted data points, or *inconsistent data* (a data point that belongs to more than one category), overfitting will cause that this "error" information will be also taken into account in the weights of a neural network or the generated rules.

In most of the cases we can expect two possible outcomes of learning: we can obtain rules, or a network, that perfectly, with almost a 100% accuracy, classifies all the learning data; or some of the learning examples are misclassified by the rules or a network, but the accuracy for unseen test data will be higher. Obviously, it is better to choose the rules or a network for the latter outcome. To avoid overfitting in case of neural networks it is to stop training early, although further training would continue to reduce the error value.

In case of inductive machine learning avoiding overfitting is a more difficult. There are, however, some techniques to prevent overfitting of generated rules:

- *cross-validation*.
  The training data set is divided into several disjoint subsets (e.g. randomly). Then the algorithm using all the subsets, except one, as a training data set at the time generates the rules. This repeats for the every created subset; each time one of the subsets is set aside for validation. The rule is that the smaller

the training data set the larger the number of training subsets should be used. In an extreme case just one example is set aside for validation, thus the algorithm is run as many times as there are examples. This latter method is called *hold-one-out*.

- *bootstrap aggregation*, or *bagging*
  The training data set is divided into about 2/3 of the entire training data for learning and 1/3 for validation. The algorithm is run several times, each time on a 2/3 subset of the original training data set. The subsets are chosen randomly, with replacement, from the entire training data set, so many training examples can appear several times in different subsets.

As the result of cross-validation or bagging, the rules that on average perform best on training data (so they also should perform well on unseen test data) are kept.

In case of decision trees there are also other approaches that can be used to prevent overfitting:

- Generation of several trees instead one, "best" tree. The classification, when using multiple trees, can be generated using a voting scheme.
- Stop growing the tree before it perfectly classifies all training data, thus allowing for the leaf nodes containing examples from more then one class
- Pruning the overfitted tree by deleting the branches that cover only a few examples or that will not cause a significant decrease in accuracy for the validation data. The problem associated with this method is the need to store the entire decision tree, before pruning; it is computationally expensive.

References

Ching, J.Y., Wong, A.K.C. and Chan, K.C.C., 1995, Class-dependent discretization for inductive learning from continuous and mixed-mode data, *IEEE Trans. on PAMI*, 17:641-651

Cios, K.J., Pedrycz, W., Swiniarski, R., 1998, Data Mining Methods for Knowledge Discovery, Kluwer

Cios, K.J., & Kurgan, L., Hybrid Inductive Machine Learning Algorithm that Generates Inequality Rules, *Information Sciences*, Special Issue on *Soft Computing Data Mining*, submitted, 2001

Paterson, A. and Niblett, T.B., 1982, *ACLS Manual*, Edinburgh: Intelligent Terminals, Ltd

Wong, A.K.C. and Liu, T.S., 1975, Typicality, diversity and feature pattern of an ensemble, *IEEE Trans. on Computers*, 24: 158-181