Inductive learning of compact rule sets by using efficient hypotheses reduction

Thomas Koch[1]

TR 92-069
October 1992

Abstract:
A method is described which reduces the hypotheses space with an efficient and easily interpretable reduction criteria called α - **reduction.** A learning algorithm is described based on α - reduction and analyzed by using probability approximate correct learning results. The results are obtained by reducing a rule set to an equivalent set of kDNF formulas.
The goal of the learning algorithm is to induce a compact rule set describing the basic dependencies within a set of data. The reduction is based on criterion which is very flexible and gives a semantic interpretation of the rules which fulfill the criteria. Comparison with syntactical hypotheses reduction show that the α - reduction improves search and has a smaller probability of missclassification.

1. ICSI on leave from Krupp Forschungsinstitut GmbH, HA Informationstechnik, Muenchener Str. 100, W- 4300 Essen 1, Germany

Acknowledgment

I thank Anna Morpurgo, Klaus-Peter Jantke, Michael Luby and Philip Kohn for the fruitful discussions about draft versions. I'm deeply grateful to Prof. Jerome Feldman has supported my work here very strongly and to the management of the Krupp Forschungsinstitut, namely to Dr. Bernd Schoenwald, Dr. Bernd Fehsenfeld and Dr. Diethard Bergers who sent me to Berkeley in order to improve research.

1. Introduction

Rule based systems are the standard technique used in expert systems. Because of the difficulties in extracting rules from a human expert, automatic rule generation is a helpful way to bridge the knowledge acquisition bottleneck.
The scenario can be described as given a data set (i.e. set of examples) from a database, construct a set of rules which describe the dependencies within the data set. Systems that perform this task are for example ID3 family [Quinlan 86] and [Utgoff 89], AQ15 [Michalski 83] or CN2 [Clark, Niblett 89] and RULEARN [Koch 88]. The questions which arises after finding the rules can be formulated as: "With what level of confidence do the generated rules describe the "true" dependencies in the data set?"
This question is solved by showing that the task of finding rules is PAC (probability approximately correct) identifiable. Given a representation, a function describing the dependencies between the size of the data set and probability and confidence can be formulated. In general this can be done independently from the probability distribution which generates the examples (data set) from the sample space.
In this approach the PAC identification is done by constructing a set of kDNF's which are equivalent to a set of rules. Because the kDNF Problem has been analyzed within the many PAC publications (f. e. [Shackelford and Volper 88],[Kearns 89)] these results can also be used for rule-based systems.
We also describe the hypotheses reduction method used in the RULEARN system, called α - reduction within the framework of PAC. This technique can be used in various applications beside Machine Learning. It is very efficient, gives a semantic interpretation of the reduction and enables an algorithm to find compact descriptions with only a small probability of missclassification.
This report describes the rule learning system RULEARN within the PAC learning framework. The RULEARN software has been developed at the Krupp Forschungsinstitut GmbH [Fehsenfeld et. al 91] and has been applied to various data sets from all sections of the production process from product development up to after sales. RULEARN is used as a data evaluation method but also as a knowledge acquisition tool for building expert systems [Kirchheiner and Koch 92]. For reference of expert system development see for example [Pfeiffer et al 88].

In section 2 we describe the ideas of the RULEARN system, which uses three user defined evaluation criteria to distinguish between good and bad rules.
In the following section we show the equivalence of concept descriptions using a set of rules and a set of kDNF's.
In section 4 we describe the PAC identification of a kDNF. This allows us to derive the number of needed examples to ensure that the found solution is ε - close to the "true" description with at least a given probability (sample complexity).
After this we describe the trade - off between search effort and the fault of a solution using a reduced search space. We formally describe the reduction of specifying a threshold in the number of literals in a term and on the α - value of a term. By using the PAC framework we can derive upper bounds on the maximal fault. We also show how the search is reduced within typical examples.
Section 7 outlines the search for rules within RULEARN. In the last section we summarize the results and compare the differences between the different reduction methods.

2. Rule generation methods

Within an expert system shell the programmer can formulate complex rules, which for example contain arbitrary functions in premise and conclusion in the underlying programming language. In this case they are nearly as powerful as the underlying programming language. In order to find rules automatically the rule representation must be restricted to a specific subset. In the following we use propositional logic as representation language.
We assume that each of the given examples can be expressed by a finite number *n* of attributes $A_1,...,A_n$ with possible values $X_1,..., X_n$. The sample space for an attribute value based inductive system can be described by the sample space $\aleph = X_1 \times X_2 \ldots \times X_{n-1} \times X_n$, where each attribute X_i has a finite set of possible values denoted by $|X_i|$. To handle unknown values a special attribute value "*" can be included in the set of possible values.
An example can be written as a vector $(x_1, x_2, \ldots, x_n)$. Each rule component describing possible examples has the form $a_{i,j}$ which means the attribute A_i has the value $x_j \in X_i$. For each example it can be tested whether $a_{i,j}$ has the value true or false. A rule can be written as $a_{i_1, j_1} \wedge a_{i_2, j_2} \wedge \ldots \wedge a_{i_k, j_k} \rightarrow a_{n, j_{k+1}}$ where $\forall (i \le k+1 \le n)\, j_i \le |X_i|$.

Without loss of generality we assume the last attribute is the one for which rules should be found (conclusion attribute).

This assumption is also typical for the induction of decision trees.

As an example for a rule induction method we choose the RULEARN machine learning system. The goal of RULEARN is to find a short (compact) set of rules from a given data set which

- describes as many data sets as possible
- describes as precisely as possible
- uses as few rules as possible.

There is a trade - off between the three goals so that all goals can not be optimized by one set of rules. The idea is to combine these goals so that a learned rule set covers each goal to at least a specific degree.
In order to do that, RULEARN uses three evaluation criteria which are measures of the rule quality. The user of the system defines thresholds for each of the measures.

The quality measures are defined as follows:

$$reliability\,(\pi) \;=\; \frac{\text{number of correct applications of the rule}}{\text{number of possible applications of the rule}}$$

$$specilization\,(\sigma) \;=\; \frac{\text{number of examples in which the conclusion is fulfilled}}{\text{number of examples in which the premise is fulfilled}}$$

$$universality\,(\alpha) \;=\; \frac{\text{number of possible applications of the rule}}{\text{number of examples}}$$

The measure of a rule is defined if the rule applicable is at least once.

The values of the criteria can have following values: $0 \le \pi \le 1$, $\frac{1}{m} \le \sigma \le m$ and $0 \le \alpha \le 1$ where m is the number of examples.

The settings of the demands determine the number of rules to be learned and the time which is needed to find the rules. If the demands are minimal, that means $\pi = 0$, $\sigma = m$ and $\alpha = 0$, RULEARN generates all possible rules. In general a set of at most 20 rules can be generated, which describe dependencies within the data. In the next sections we focus on the threshold for the universality (α) and show how the threshold for α reduces the search space.

3. Reduction

This section describes the formalization of rules using attribute value assignments. In the case the learning of rules is reduced to the learning of DNF's.

A **DNF** is a set of terms $t_1, t_2, \ldots, t_l$ where each term $t_i = x_1 \wedge x \wedge \ldots \wedge x_p$ is a conjunction of p literals where each literal is negated or not. If no literal is negated, we call the DNF monotone. If no term of the DNF has more than k literals, the DNF is called a **kDNF**.

We reduce the learning of kDNF to learning rules by using a set of kDNF's in which each element corresponds to a set of rules describing one conclusion attribute value. The result is a set of $|X_n|$ separate kDNF's. For each rule we construct a term in the kDNF which belongs to the conclusion attribute value which is described by the rule.

To construct a term which is equivalent to a given rule we define a set of vectors $\vec{a} = (\vec{a}[1], \ldots, \vec{a}[|\vec{a}|]) = (a_{1,1}, a_{1,2}, \ldots a_{1,|X_1|}, \ldots, a_{n,1}, \ldots, a_{n,|X_n|})$ called attribute-value-vector with

$|\vec{a}| = \sum_{i=1}^{n} |X_i|$ which contains all possible rule components $a_{i,j}$.

For each term in the kDNF we construct a vector $\vec{a}$ with $\vec{a}[i] = 1$ iff the corresponding rule component occurs in the actual rule. This component must be included in the kDNF which corresponds to the conclusion attribute value which is inferred by the rule.

A inference using the rules is equivalent to the testing of $|X_n|$ kDNF's one after another. If one kDNF has the value true, the conclusion value which corresponds to the actual kDNF can be inferred. If the rule set is consistent, at most one of the kDNF's is true for any possible combination of attribute values. If the rule set does not match the given example, none of the $|X_n|$ kDNF's is true.

Notice that this construction allows a given attribute to be represented by more than one value. For example the value for an attribute color may have the values "red" and "black". If a rule set should be generated in which each attribute has only two possible values and which classifies all examples, the attribute value vector needs to be only half the size.
As mentioned above, it is also possible that none of the rules match a given example even in the training set.
This property is used in the RULEARN system to keep the generated rule set small. If a rule set

must cover all training examples it would normally consist of many rules which are generally more complicated. The reason for this is that in general if there is an exception to a general rule, the exception must be included in the general rule (which makes the rule more complicated) and the exception must be described by an individual rule in order to describe all examples (which increases the number of rules).

4. PAC identification

Learning of kDNF's is one of the first problems investigated by Valiant in 1984 [Valiant 84]. The question: "How many examples do I need in order to ensure that my generated concept c differs only ε form the true "unknown" concept with probability at least 1 - δ?" has been solved under various conditions. We can use this results for the task of rule learning algorithms because we have reduced the problem of finding rules to finding a set of kDNF's.
Within PAC learning there are different ways to treat noise. The easiest way is to assume that there is no noise in the given examples. If you handle the noise you have to distinguish whether only the conclusion attribute is affected by noise (example has false classification) or whether each attribute is affected by noise (the "whole" example is noisy). The number of needed data depends on the used noise model.

4.1 Each attribute value is affected by a known noise factor.

In this noise model, each rule component $a_{i,j}$ (literal) is affected by a known noise rate β so that

$a_{i,j} = a_{i,j}$	with probability 1 - β
$a_{i,j} = 0$	with probability 0.5 β
$a_{i,j} = 1$	with probability 0.5 β

Shackelford and Volper [Shackelford, Volper 88] showed that the class of kDNF is **PAC - identifiable** which means there exists as algorithm A, so that for $\forall (c \in kDNF)$, $\forall (\varepsilon \geq 0)$, $\forall (\delta \geq 0)$ and for all distributions D using as Oracle function E(f) outputs a Boolean function *c* such that:

$$p\left(p_D\left(h \nabla c < \varepsilon\right)\right) \geq 1 - \delta \qquad (1)$$

where *h* is the "true" concept to be learned and $h \nabla c$ is a measure of the region in which *h* and *c* disagree. p_D is the probability given the distribution D.

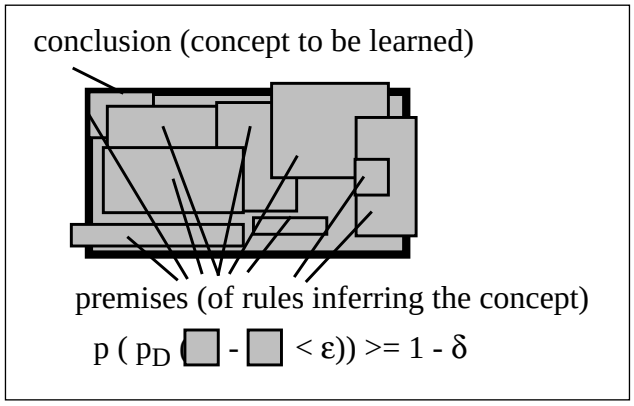

Figure 1: learning a kDNF which is ε close to the “true” concept

The algorithm to construct the kDNF essentially tests for all negative examples whether they are “true” negative or whether they are “true” positive examples which seem to be negative due to the noise. By assuming a specific noise model it can be found out which case is preferred. The terms which correspond to the negative examples are deleted so that the remaining set of terms describes the “true” concept within the demanded bounds. The algorithm finds the concept to be learned because it is assumed to be expressed by a kDNF.

They showed that if

$$m \geq \frac{2^{k+3} K^2}{\varepsilon^2 (1-\beta)^{2k}} \log\left(\frac{2^{2k+1} E}{\delta}\right)$$

then the probability is in the assumed bounds where K is the number of terms with at most k literals and E is the number of equivalence classes over the set of terms.

$K = \sum_{r=0}^{k} \binom{n}{r} 2^r \leq (2n)^{k+1}$ [Valiant 84] and the equivalence relation over the set of terms is defined so that two terms are in the same class iff they consist of exactly the same literals. Therefore $E = \sum_{i=0}^{k} \binom{n}{i}$. The number of needed examples is called the **sample complexity**. In section 5 we show that the RULEARN algorithm uses a reduced search space which is less complex.

4.2 Conclusion attribute is affected by noise

In this noise model we have the correct classification of a given example with probability 1 - β. The other attributes aren't affected by noise. This conditions have been analyzed by Valiant [Valiant 85]

Let M be the number of possible rule premises (terms), then $M \leq \sum_{i=1}^{n} \binom{n}{i} l^i$, where l is the maximal number of possible values for each attribute. If $\beta = \frac{\delta}{4|M|}$ and $\varepsilon = 2\beta$ then a concept h is PAC - identifiable by a DNF c if

$$m \geq \frac{36}{\delta} |M| \log\left(\frac{|M|}{\delta}\right) \quad .$$

4.2 Noise free case

Before we analyze the number of needed examples we introduce two definitions:
If C is a class of representations over X we say that C **shatters** a set $Y \subseteq X$ if the set $\{c \cap Y | c \in C\}$ is the power set of Y. The **Vapnic-Chervonenkis dimension** (or VC-dimension) of C denoted VCD(C) is the greatest integer d such that there exists a set of cardinality d that is shattered by C. In one sense the VC dimension is a measure of the number of "degrees of freedom" possessed by C. The VC -dimension was originally introduced by Vapnic and Chervonenkis [Vapnic and Chervonenkis 71].

Kearns [Kearns 89] derived a lower bound for the sample space complexity in the noise free case. If $0 < \varepsilon \leq \frac{1}{32}$, $0 < \delta \leq \frac{1}{1000}$ and $VCD(C) \geq 2$ then C is PAC - identifiable if $m \geq \frac{VCD(C) - 2}{128\varepsilon}$ where $VCD(C)$ is the Vapnic-Chervonenkis dimension.

If we don't specify bounds on ε and δ Natrajan [Natrajan 91] showed that if

$$m \geq \frac{1}{\varepsilon} \left((n' + 1)\, VCD(C) \ln(2) + \ln\left(\frac{1}{\delta}\right) \right)$$

then C is PAC - identifiable (without noise).

This results can be used for determining the sample complexity for rule learning systems of

kDNF's. The VC - dimension of the concept class of all rule sets or kDNF is the number of possible terms. So the above results can be applied by substituting VCD(C) to *M*.

In the result from Natrajan *n'* is a length parameter which specifies the number of possible variables. In our case $n' = n\,l$ where *n* is the number of attributes and *l* is the maximal number of possible values for each attribute.

Because this results are based on a worst case analysis, the number of needed examples is much higher than the number available in practical applications. This results can be improved by assuming a underlying distribution or by reducing *M*.

In the next section we describe how the set *M* can be reduced to a small fraction by specifying a α threshold. If we assume that a concept is learnable in respect to a give α value the number of needed examples becomes much smaller.

5. Trade - off between finding a approximate good and a compact set of rules

After we have described the syntactical equivalence between the set of kDNF's and the set of rules we want to figure out some differences between the semantics of learning kDNF and rule induction within RULEARN. The goal of learning the kDNF as described before is to find a concept description which differs from the 'true" concept at most ε assuming that the true concept can be described by a kDNF.

The idea of finding rules within the RULEARN system is to generate a compact rule set which consists of only a few general and precise rules. As a result of this, the goal is not to be as close to the "true" concept as possible. In terms of the kDNF this means that there should be only a few terms.
There is a trade - off between the correctness and the compactness of the classification.

Figure 2 and 3 illustrate the different view to classification.

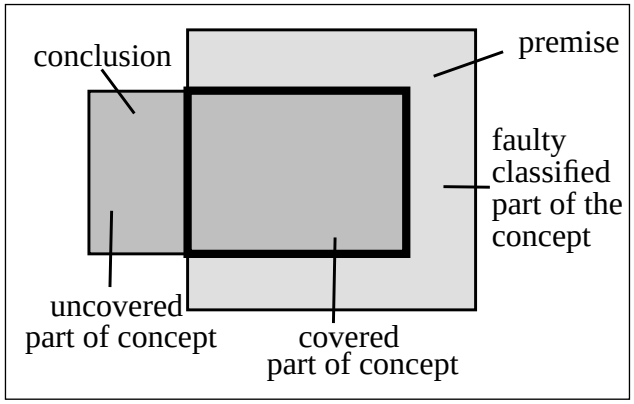

Figure 2: describing a rule for the best cover

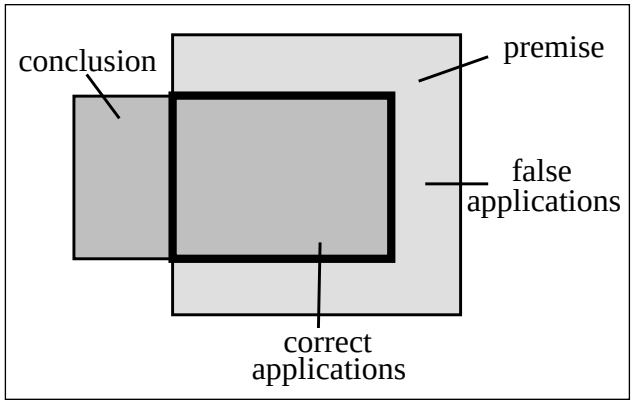

Figure 3: Describing a rule in general

The main difference is that the RULEARN system does not care about the part of the concept which is not described by the rule. In contrast to that the kDNF construction algorithm and also most rule-learning systems try to find a ε-close cover in which the unclassified parts of the concept are counted as faults.

In order to construct a compact rule set from the set of kDNF's which cover the "true" concept with a symmetric difference from at least ε, we delete all terms which do not fulfill the demanded thresholds. In the following we describe the fault which is done by dropping out some rules which cover a part of the "true" concept.

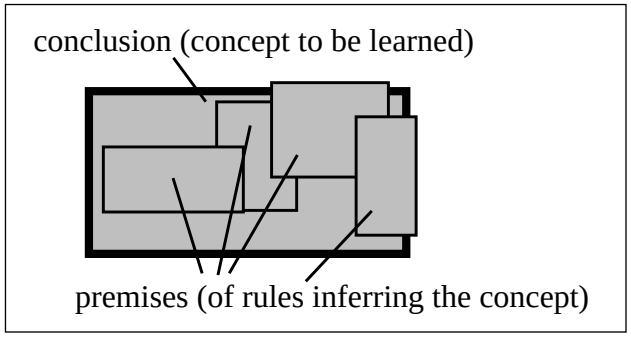

Figure 4: Finding a compact rule set describing the concept

The threshold for the universality reduces the set of possible rules (which correspond to the terms in the kDNF) by demanding that there must be at least a number of occurrences in which the premise is fulfilled. All rules or terms describing less examples are dropped out. The set of possible hypotheses H (hypotheses space) is the power set of the set of possible rules (terms of the DNF) M. Because $|H|$ is exponential in the number of rules ($H = 2^M$), a reduction of M reduces the hypotheses H very efficient.

The process of defining a threshold for the universality can be compared with defining a maximal number of literals k in a term to reduce the search space from DNF to kDNF. This is also a reduction of the set of terms. After describing the reduction we compare the results with kDNF.

Proposition 1

For each fixed set of attributes there are at most $\frac{1}{\alpha}$ *combinations of attribute values which have a universality of at least* α.

Proof

A: We first evaluate the case when the rule consists of only one element in the premise.

If the attribute has more than $\frac{1}{\alpha}$ attribute values which occur at least α times, the sum of all occurrences is greater than the number of examples, which is a contradiction.

B: If the rule consists of k components (literals), there are at most l^k possible combinations of attribute values for k fixed attributes where l is the maximal number of different attribute values. Because the sum of all occurrences of each combination is the number of examples, there can be at most $\frac{1}{\alpha}$ combinations which occur at least α times. ■

In the calculation we only consider rules in which all attribute values in a premise (term) are taken from different attributes, because the conjunction of two components (literals) of the same attribute is always false. In the reduction to the DNF described before such combinations are possible. This increases the set of rules and the hypotheses space more than actually needed because such rules do not make sense.

Theorem 1

Let l be the maximal number of different attribute values, then the sum of all possible rules in respect to α: M_α is for all $\alpha > \frac{1}{l}$:

$$M_\alpha \leq \sum_{i=1}^{n} \binom{n}{i} \frac{1}{\alpha} \qquad (2)$$

where n is the number of attributes.

Proof

From Proposition 1 there are at most $\frac{1}{\alpha}$ attribute values for each combination of attributes. The sum of all possible combinations of attribute values is $\sum_{i=1}^{k} \binom{n}{i}$.■

For low values for α l^i might be larger than $\frac{1}{\alpha}$, so that result of the theorem can be improved.

Because the number of possible combinations of k (fixed) attributes is $\prod_{i=j_1}^{j_k} |X_i|$, this is the number of possible values for each combination.

For each $\alpha \geq \frac{1}{\prod_{i=1}^{n} |X_i|}$ there is a constant *d*, so that $\frac{1}{\prod_{i=1}^{d} |X_i|} < \alpha \leq \frac{1}{\prod_{i=1}^{d-1} |X_i|}$.

For all rules containing more than d components (terms with more than d literals), α reduces the number of rules to be considered. From (3) we can therefore derive:

$$M_\alpha \leq \sum_{i=1}^{d} \binom{n}{i} l^d + \sum_{i=d+1}^{n} \binom{n}{i} \frac{1}{\alpha} \qquad (2.1)$$

Because there are at most l^d combinations of attribute values, from (2) and (2.1) we derive:

$$M_\alpha \leq \sum_{i=1}^{n} \binom{n}{i} \min \{ \frac{1}{\alpha}, l^i \} \qquad (2.2)$$

If we reduce the search from DNF to kDNF the number of possible terms M^k is

$$M^k \leq \sum_{i=1}^{k} \binom{n}{i} l^i \qquad (2.3)$$

Because this is only the exponent for the number of hypotheses, the α-reduction is a very efficient way of reducing the hypotheses space.

Notice that we made no assumption about the distribution which generates the examples from the sample space. In the derivation we used the worst case in which each attribute value occurs the same number of times. The more the occurrences of the attribute values differ, the lower is the number of possible rules with respect to α.

The effect of defining a minimum universality α on the hypotheses space is shown in figure 5.

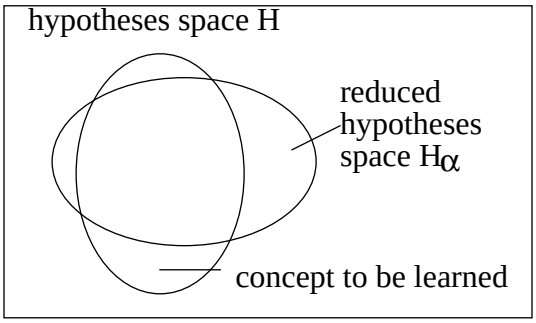

Figure 5: Reduction of the hypotheses space

The threshold for α may cause only a subset of the possible concept to be learned. If α is less than

$\frac{1}{m}$ and each possible rule is applicable at least once, than H_α is equal H. If α is near to one then H_α may be empty. In this case the concept is not learnable with respect to α. As discussed before, instead of finding an exact description of the concept, the algorithm finds a subset so that each element describes a part of the "true" concept with at least a given probability. This problem is equivalent to learning a kDNF for an unknown concept with a fixed *k*. If we assume that the concept may be described with the chosen representation, the algorithm will find it.

In the following we assume that there is a concept to be learned *h* and there exists a DNF *c*, so that

$$p\,(p_D\,(h \nabla c < \varepsilon)\,) \geq 1 - \delta \qquad (1)$$

Let t be a term, |t| denotes the number of literals in t and we call |t| the length of t. Let $c \in DNF$, than $|c| = \sum_{t \in c} |t|$ and we call |*c*| the length of *c*. The function *cover*(t) gives the subset of possible examples with respect to the given distribution which is covered by *t*. We define $c_\alpha \subseteq c = \{t \in c \,|\, |cover\,(t)| > \alpha m\}$ the terms in the given DNF c which cover at least α percent of the data and $c_k \subseteq c = \{t \in c \,|\, |t| \leq k\}$ the terms which have at most k literals.

We want to compare the hypotheses reduction of a DNF *c* to:

- to a kDNF c_k
- to a subset c_α

For $k_{max} = max\,\{|t| \in c\}$ and $\alpha_{min} = min\,\{|cover\,(t)|\,|t \in c\}$ clearly $c_{k_{max}} = c_{\alpha_{min}} = c$.

The question which remains is: "What is the maximum fault in respect to *c* which is made by defining an α higher than α_{min} or a k less than k_{max}?"

1. $\alpha > \alpha_{min}$

There exists a set of terms $t = t_1,...,t_l$ with $\forall\,(i \leq l)\; t_i \in c \wedge t_i \notin H_\alpha$. Let |*cover*(t)| = m_t and |*cover*(c-t)| = $m_{c\text{-}t}$ then $m_t + m_{c\text{-}t} >= m$. The sum is equal to m if the two sets are disjunct.

In formula (1) ε is a upper bound for the
uncovered examples $u = \{x \,|\, x \in cover\,(h) \wedge x \notin cover\,(c)\}$ and the faulty covered examples

$f = \{x | x \notin cover(h) \wedge x \in cover(c)\}$. In the worst case *cover*(t) and f are disjunct, this means that the terms which cover less than α percent of the data don't imply any faulty classification.
In other words, the probability for a faulty classification of the remaining set of terms increases because the terms which are dropped out have no missclassification.

In contrast to the standard PAC identification we want to distinguish between
- the probability that an example is classified false
- the probability that an example is uncovered

The phrase $h \nabla c$ includes both cases.

For the first case we can derive from (1)

$$p(p_D(\text{missclassification of } (c_\alpha) < \varepsilon \frac{cover(c)}{cover(c_\alpha)})) \geq 1 - \delta \qquad (1.1)$$

Because ε is a upper bound for the missclassification, in the worst case each term which is dropped out has no missclassification. In this case the smaller remaining subset has the same (absolute) amount of missclassification. The percentage of missclassification decreases with the number of dropped out terms.

Of course the probability that an example of the true concept is classified decreases by specifying α. The amount of lost coverage is $\bigcup_{t \in c \wedge t \notin c_\alpha} cover(t)$, so in the worst case is if the dropped out terms are disjunct ($\bigcap_{t \in c \wedge t \notin c_\alpha} cover(t) = \emptyset$).

If the set a DNF contains lots of specific terms, it also contains a super set which contains the smaller terms. To be precise we refer to the definition of a **sunflower** as: A family of sets S is called a sunflower if there is a set C, called center, such that for every $A, B \in C$ if $A \neq B$ then $A \cap B = C$.

The theorem by Erdoes and Rado [Erdoes Rado 60] claims that if there is a family of sets F and two integers v and w, so that every element in F has size at most v and $F > v!\,(w-1)^v$ than F contains a sunflower of size w. By using this result we can guarantee that if the description of h con-

tains $v!\,(w-1)^v$ terms it has a sunflower which is not dropped out if the threshold for α is increased to drop out the small terms.

2. $k < k_{max}$

We now consider the case where we reduce the concept c to a kDNF c_k with k less than k_{max}.
In this case there exists a set of terms $t_1, \ldots, t_j \mid \forall i \le j\,(t_i \in c \wedge t_i \notin c_k)$ with $|t_i| > k$. Because $cover(c) \subset cover(c_k)$ we must only consider missclassifications. Each of this terms misses at least one subterm (at most one literal) l_i. In the worst case t_i classifies all examples where $t_i \wedge l$ classifies only examples within h. From (1) we derive

$$p\left(p_D\left(h \nabla c_k < min\{1, \varepsilon + \frac{\bigcup_{i \le j} cover(t_i) \cap cover(l_i)}{cover(TRUE)}\}\right)\right) \ge 1 - \delta \qquad (1.3)$$

If one of the terms l_i covers all examples the upper bound is 1.

In comparison of both reduction methods the α-reduction decreases the number of applications (i.e. number classifications) of a rule by keeping the probability of a missclassification small. On the other hand the reduction on the term length increases the number of applications of the rule set by allowing a high probability of missclassification.
In that sense the α-reduction is more secure. If the rule is applicable, then the probability that the result is correct is within a reasonable bound.

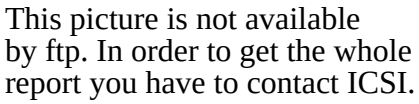

Figure 5 a) Number of possible rule-components (terms) in relation to different values for α and k in linear scale

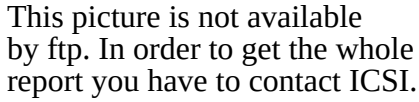

Figure 5 b): Number of possible rule-components (terms) in relation to different values for α and k in half-logarithmic scale

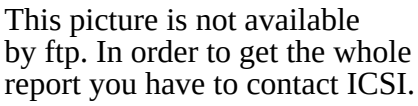

Figure 6: Number of possible rule-components (terms) in relation to different values for α and k in logarithmic scale (in linear scaling the alpha reduction can not be distinguished from the x-axis).

6. RULEARN algorithm

The RULEARN algorithm can be described as

1. determine the set of possible rules with respect to α
2. test, whether the rule fulfills the thresholds for σ and π
   - true: store rule and test the next
   - false: look for a new rule to be tested

The idea is to find as high thresholds as possible in order to keep the rule set small. Given a true rule $r \in h$ the system should have thresholds to find r even if the examples are affected by noise. If we use the noise model in which each attribute is affected by noise, we can derive a threshold for the reliability π.

If $r = a_{i_1, j_1} \wedge a_{i_2, j_2} \wedge \ldots \wedge a_{i_k, j_k} \rightarrow a_{n, j_{k+1}}$ the counterexamples which must be considered fulfill the premise, but have a different value for the conclusion attribute value a_{n, j_l} with $j_l \leq |X_n| \wedge j_l \neq j_{k+1}$. Because each component in the attribute value vector can be affected by

noise, a counterexample is an example in which at least one of the components for the conclusion is changed. In the noise model where each attribute value is affected by noise the probability that a correct true value (1) is set to 0 is $\frac{\beta}{2}$, which is equal to the probability that a correct false value (0) is 1. If the examples are generated independently from an oracle, the probability that an example becomes a counterexample is $1 - (1 - \frac{\beta}{2})^{|X_n|}$. So if the reliability threshold is less than this value the rule would be generated.

The influence of the universality (α) has been described in the previous sections. Independent from this, if the specialization is set to *m* and the universality values are set to 0, no “true” rule is dropped out.

6. Conclusions

We have represented a very efficient way of reducing the search space that can be used for learning rules or kDNF formulas. The reduced space decreases exponentially with only a small fault for missclassification. In contrast to syntactical reduction, thresholds in the number of literals within a term, the α-reduction gives a easy semantic interpretation. It allows the user to have a deeper understanding about the definition of the threshold and is easier to adjust.
Combining both reduction techniques gives a even better reduction of the search space.

7. References

[Bloendorn Michalski 91] Bloendorn E. and Michalski R. S.: Data-driven Constructive Induction in AQ17-DCI: A Method and Experiments, Reports of Machine Learning and Inference Laboratory, Center for Artificial Intelligence, George Mason University, 1991
[Clark Niblett 89] Clark P. and Niblett T. : The CN2 induction algorithm, Machine Learning. 3:261-283, 1989
[Erdoes Rado] Erdoes, P. and Rado R.: Intersection theorems for systems of sets, Journal of the London Math. Society, vol.35 pp. 85-90, 1960
[Fehsenfeld et al 91] Fehsenfeld B., Harris S., Koch T. and Kirchheiner R. : Automatic learning with RULEARN in the example of corrosion tests, in Technische Mitteilungen Krupp 1/1991
[Kearns 89] Kearns M. J.: The Computational Complexity of Machine Learning, MIT press, 1989
[Kirchheiner Koch 92] Kirchheiner R. and Koch T. : Computer aided learning of corrosion rules

from factual data bases, Proceedings of the NACE-CORRSION ‘92, Nashville 1992

[Koch 88] Koch T.: Effizientes Lernen und Bewerten von Regeln, Kuenstliche Intelligenz Informatik Fachberichte 181, 186-195, Springer Berlin 1988

[Michalski 83] Michalski R. S.: A Theory and Methodology of Inductive Learning in “Machine Learning: An Artificial Intelligence Approach, Michalski R.S.,Carbonell J. and Mitchell T. (Eds.) pp. 83-134, Morgan Kaufmann Publishing Co., Mountain View, CA, 1983

[Natrajan 91] Natrajan B. K. : Machine learning: A theoretical approach, Morgan Kaufmann Publishers 1991

[Pfeiffer et al 88] Pfeiffer J., Siepmann T. and Teichmann W.: Expert system for metal machining practise, in Technische Mitteilungen Krupp 46 (1988) pp 113 - 124

[Shackelford, Volper 88] Shackelford G. and Volper D. : Learning k-DNF with noise in thee Attributes. Proceedings of the first Annual ACM Workshop on Computational Learning Theory 1988, 97-103

[Quinlan 86] Quinlan J.R.: Induction of decision trees, Machine Learning, 1:81-106, 1986

[Utgoff 89] Utgoff P.E.: Incremental Learning of Decision trees., Machine Learning 4:161-186, 1989

[[Valiant 84] Valiant L.G. : A Theory of the learnable, Communications of the ACM, 27:11, 1134-1142, 1984

[Valiant 85] Valiant L.G. : Learning disjunctions of conjunctions,Proc. of the 9 th IJCAI, Los Angeles California, Morgan Kaufman 1985, pp 560-566

[Vapnic and Chervonenkis 71] Vapnik V. N. and Chervonenkis A. Ya.: On uniform convergence of relative frequencies of events to their probabilities , Theory of Probability and its Applications, 16(2).1971, pp. 264-280

Inductive learning of compact rule sets by using efficient hypotheses reduction

Thomas Koch[1]

TR 92-069
October 1992

Abstract:
A method is described which reduces the hypotheses space with an efficient and easily interpretable reduction criteria called α - **reduction.** A learning algorithm is described based on α - reduction and analyzed by using probability approximate correct learning results. The results are obtained by reducing a rule set to an equivalent set of kDNF formulas.
The goal of the learning algorithm is to induce a compact rule set describing the basic dependencies within a set of data. The reduction is based on criterion which is very flexible and gives a semantic interpretation of the rules which fulfill the criteria. Comparison with syntactical hypotheses reduction show that the α - reduction improves search and has a smaller probability of missclassification.

1. ICSI on leave from Krupp Forschungsinstitut GmbH, HA Informationstechnik, Muenchener Str. 100, W- 4300 Essen 1, Germany

Acknowledgment

I thank Anna Morpurgo, Klaus-Peter Jantke, Michael Luby and Philip Kohn for the fruitful discussions about draft versions. I'm deeply grateful to Prof. Jerome Feldman has supported my work here very strongly and to the management of the Krupp Forschungsinstitut, namely to Dr. Bernd Schoenwald, Dr. Bernd Fehsenfeld and Dr. Diethard Bergers who sent me to Berkeley in order to improve research.

1. Introduction

Rule based systems are the standard technique used in expert systems. Because of the difficulties in extracting rules from a human expert, automatic rule generation is a helpful way to bridge the knowledge acquisition bottleneck.
The scenario can be described as given a data set (i.e. set of examples) from a database, construct a set of rules which describe the dependencies within the data set. Systems that perform this task are for example ID3 family [Quinlan 86] and [Utgoff 89], AQ15 [Michalski 83] or CN2 [Clark, Niblett 89] and RULEARN [Koch 88]. The questions which arises after finding the rules can be formulated as: "With what level of confidence do the generated rules describe the "true" dependencies in the data set?"
This question is solved by showing that the task of finding rules is PAC (probability approximately correct) identifiable. Given a representation, a function describing the dependencies between the size of the data set and probability and confidence can be formulated. In general this can be done independently from the probability distribution which generates the examples (data set) from the sample space.
In this approach the PAC identification is done by constructing a set of kDNF's which are equivalent to a set of rules. Because the kDNF Problem has been analyzed within the many PAC publications (f. e. [Shackelford and Volper 88],[Kearns 89)] these results can also be used for rule-based systems.
We also describe the hypotheses reduction method used in the RULEARN system, called α - reduction within the framework of PAC. This technique can be used in various applications beside Machine Learning. It is very efficient, gives a semantic interpretation of the reduction and enables an algorithm to find compact descriptions with only a small probability of missclassification.
This report describes the rule learning system RULEARN within the PAC learning framework. The RULEARN software has been developed at the Krupp Forschungsinstitut GmbH [Fehsenfeld et. al 91] and has been applied to various data sets from all sections of the production process from product development up to after sales. RULEARN is used as a data evaluation method but also as a knowledge acquisition tool for building expert systems [Kirchheiner and Koch 92]. For reference of expert system development see for example [Pfeiffer et al 88].

In section 2 we describe the ideas of the RULEARN system, which uses three user defined evaluation criteria to distinguish between good and bad rules.
In the following section we show the equivalence of concept descriptions using a set of rules and a set of kDNF's.
In section 4 we describe the PAC identification of a kDNF. This allows us to derive the number of needed examples to ensure that the found solution is ε - close to the "true" description with at least a given probability (sample complexity).
After this we describe the trade - off between search effort and the fault of a solution using a reduced search space. We formally describe the reduction of specifying a threshold in the number of literals in a term and on the α - value of a term. By using the PAC framework we can derive upper bounds on the maximal fault. We also show how the search is reduced within typical examples.
Section 7 outlines the search for rules within RULEARN. In the last section we summarize the results and compare the differences between the different reduction methods.

2. Rule generation methods

Within an expert system shell the programmer can formulate complex rules, which for example contain arbitrary functions in premise and conclusion in the underlying programming language. In this case they are nearly as powerful as the underlying programming language. In order to find rules automatically the rule representation must be restricted to a specific subset. In the following we use propositional logic as representation language.
We assume that each of the given examples can be expressed by a finite number *n* of attributes $A_1,...,A_n$ with possible values $X_1,..., X_n$. The sample space for an attribute value based inductive system can be described by the sample space $\aleph = X_1 \times X_2 \ldots \times X_{n-1} \times X_n$, where each attribute X_i has a finite set of possible values denoted by $|X_i|$. To handle unknown values a special attribute value "*" can be included in the set of possible values.
An example can be written as a vector $(x_1, x_2, \ldots, x_n)$. Each rule component describing possible examples has the form $a_{i,j}$ which means the attribute A_i has the value $x_j \in X_i$. For each example it can be tested whether $a_{i,j}$ has the value true or false. A rule can be written as $a_{i_1, j_1} \wedge a_{i_2, j_2} \wedge \ldots \wedge a_{i_k, j_k} \rightarrow a_{n, j_{k+1}}$ where $\forall (i \le k+1 \le n)\, j_i \le |X_i|$.

Without loss of generality we assume the last attribute is the one for which rules should be found (conclusion attribute).

This assumption is also typical for the induction of decision trees.

As an example for a rule induction method we choose the RULEARN machine learning system. The goal of RULEARN is to find a short (compact) set of rules from a given data set which

- describes as many data sets as possible
- describes as precisely as possible
- uses as few rules as possible.

There is a trade - off between the three goals so that all goals can not be optimized by one set of rules. The idea is to combine these goals so that a learned rule set covers each goal to at least a specific degree.
In order to do that, RULEARN uses three evaluation criteria which are measures of the rule quality. The user of the system defines thresholds for each of the measures.

The quality measures are defined as follows:

$$reliability\,(\pi) \;=\; \frac{\text{number of correct applications of the rule}}{\text{number of possible applications of the rule}}$$

$$specilization\,(\sigma) \;=\; \frac{\text{number of examples in which the conclusion is fulfilled}}{\text{number of examples in which the premise is fulfilled}}$$

$$universality\,(\alpha) \;=\; \frac{\text{number of possible applications of the rule}}{\text{number of examples}}$$

The measure of a rule is defined if the rule applicable is at least once.

The values of the criteria can have following values: $0 \le \pi \le 1$, $\frac{1}{m} \le \sigma \le m$ and $0 \le \alpha \le 1$ where m is the number of examples.

The settings of the demands determine the number of rules to be learned and the time which is needed to find the rules. If the demands are minimal, that means $\pi = 0$, $\sigma = m$ and $\alpha = 0$, RULEARN generates all possible rules. In general a set of at most 20 rules can be generated, which describe dependencies within the data. In the next sections we focus on the threshold for the universality (α) and show how the threshold for α reduces the search space.

3. Reduction

This section describes the formalization of rules using attribute value assignments. In the case the learning of rules is reduced to the learning of DNF's.

A **DNF** is a set of terms $t_1, t_2, \ldots, t_l$ where each term $t_i = x_1 \wedge x \wedge \ldots \wedge x_p$ is a conjunction of p literals where each literal is negated or not. If no literal is negated, we call the DNF monotone. If no term of the DNF has more than k literals, the DNF is called a **kDNF**.

We reduce the learning of kDNF to learning rules by using a set of kDNF's in which each element corresponds to a set of rules describing one conclusion attribute value. The result is a set of $|X_n|$ separate kDNF's. For each rule we construct a term in the kDNF which belongs to the conclusion attribute value which is described by the rule.

To construct a term which is equivalent to a given rule we define a set of vectors $\vec{a} = (\vec{a}[1], \ldots, \vec{a}[|\vec{a}|]) = (a_{1,1}, a_{1,2}, \ldots a_{1,|X_1|}, \ldots, a_{n,1}, \ldots, a_{n,|X_n|})$ called attribute-value-vector with

$|\vec{a}| = \sum_{i=1}^{n} |X_i|$ which contains all possible rule components $a_{i,j}$.

For each term in the kDNF we construct a vector $\vec{a}$ with $\vec{a}[i] = 1$ iff the corresponding rule component occurs in the actual rule. This component must be included in the kDNF which corresponds to the conclusion attribute value which is inferred by the rule.

A inference using the rules is equivalent to the testing of $|X_n|$ kDNF's one after another. If one kDNF has the value true, the conclusion value which corresponds to the actual kDNF can be inferred. If the rule set is consistent, at most one of the kDNF's is true for any possible combination of attribute values. If the rule set does not match the given example, none of the $|X_n|$ kDNF's is true.

Notice that this construction allows a given attribute to be represented by more than one value. For example the value for an attribute color may have the values "red" and "black". If a rule set should be generated in which each attribute has only two possible values and which classifies all examples, the attribute value vector needs to be only half the size.
As mentioned above, it is also possible that none of the rules match a given example even in the training set.
This property is used in the RULEARN system to keep the generated rule set small. If a rule set

must cover all training examples it would normally consist of many rules which are generally more complicated. The reason for this is that in general if there is an exception to a general rule, the exception must be included in the general rule (which makes the rule more complicated) and the exception must be described by an individual rule in order to describe all examples (which increases the number of rules).

4. PAC identification

Learning of kDNF's is one of the first problems investigated by Valiant in 1984 [Valiant 84]. The question: "How many examples do I need in order to ensure that my generated concept c differs only ε form the true "unknown" concept with probability at least 1 - δ?" has been solved under various conditions. We can use this results for the task of rule learning algorithms because we have reduced the problem of finding rules to finding a set of kDNF's.
Within PAC learning there are different ways to treat noise. The easiest way is to assume that there is no noise in the given examples. If you handle the noise you have to distinguish whether only the conclusion attribute is affected by noise (example has false classification) or whether each attribute is affected by noise (the "whole" example is noisy). The number of needed data depends on the used noise model.

4.1 Each attribute value is affected by a known noise factor.

In this noise model, each rule component $a_{i,j}$ (literal) is affected by a known noise rate β so that

$a_{i,j} = a_{i,j}$	with probability 1 - β
$a_{i,j} = 0$	with probability 0.5 β
$a_{i,j} = 1$	with probability 0.5 β

Shackelford and Volper [Shackelford, Volper 88] showed that the class of kDNF is **PAC - identifiable** which means there exists as algorithm A, so that for $\forall (c \in kDNF)$, $\forall (\varepsilon \geq 0)$, $\forall (\delta \geq 0)$ and for all distributions D using as Oracle function E(f) outputs a Boolean function *c* such that:

$$p\left(p_D\left(h \nabla c < \varepsilon\right)\right) \geq 1 - \delta \qquad (1)$$

where *h* is the "true" concept to be learned and $h \nabla c$ is a measure of the region in which *h* and *c* disagree. p_D is the probability given the distribution D.

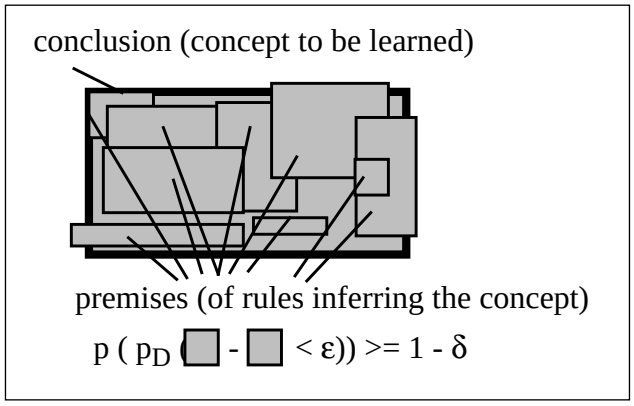

Figure 1: learning a kDNF which is ε close to the “true” concept

The algorithm to construct the kDNF essentially tests for all negative examples whether they are “true” negative or whether they are “true” positive examples which seem to be negative due to the noise. By assuming a specific noise model it can be found out which case is preferred. The terms which correspond to the negative examples are deleted so that the remaining set of terms describes the “true” concept within the demanded bounds. The algorithm finds the concept to be learned because it is assumed to be expressed by a kDNF.

They showed that if

$$m \geq \frac{2^{k+3} K^2}{\varepsilon^2 (1-\beta)^{2k}} \log\left(\frac{2^{2k+1} E}{\delta}\right)$$

then the probability is in the assumed bounds where K is the number of terms with at most k literals and E is the number of equivalence classes over the set of terms.

$K = \sum_{r=0}^{k} \binom{n}{r} 2^r \leq (2n)^{k+1}$ [Valiant 84] and the equivalence relation over the set of terms is defined so that two terms are in the same class iff they consist of exactly the same literals. Therefore $E = \sum_{i=0}^{k} \binom{n}{i}$. The number of needed examples is called the **sample complexity**. In section 5 we show that the RULEARN algorithm uses a reduced search space which is less complex.

4.2 Conclusion attribute is affected by noise

In this noise model we have the correct classification of a given example with probability 1 - β. The other attributes aren't affected by noise. This conditions have been analyzed by Valiant [Valiant 85]

Let M be the number of possible rule premises (terms), then $M \leq \sum_{i=1}^{n} \binom{n}{i} l^i$, where l is the maximal number of possible values for each attribute. If $\beta = \frac{\delta}{4|M|}$ and $\varepsilon = 2\beta$ then a concept h is PAC - identifiable by a DNF c if

$$m \geq \frac{36}{\delta} |M| \log\left(\frac{|M|}{\delta}\right) \quad .$$

4.2 Noise free case

Before we analyze the number of needed examples we introduce two definitions:
If C is a class of representations over X we say that C **shatters** a set $Y \subseteq X$ if the set $\{c \cap Y | c \in C\}$ is the power set of Y. The **Vapnic-Chervonenkis dimension** (or VC-dimension) of C denoted VCD(C) is the greatest integer d such that there exists a set of cardinality d that is shattered by C. In one sense the VC dimension is a measure of the number of "degrees of freedom" possessed by C. The VC -dimension was originally introduced by Vapnic and Chervonenkis [Vapnic and Chervonenkis 71].

Kearns [Kearns 89] derived a lower bound for the sample space complexity in the noise free case. If $0 < \varepsilon \leq \frac{1}{32}$, $0 < \delta \leq \frac{1}{1000}$ and $VCD(C) \geq 2$ then C is PAC - identifiable if $m \geq \frac{VCD(C) - 2}{128\varepsilon}$ where $VCD(C)$ is the Vapnic-Chervonenkis dimension.

If we don't specify bounds on ε and δ Natrajan [Natrajan 91] showed that if

$$m \geq \frac{1}{\varepsilon} \left((n' + 1)\, VCD(C) \ln(2) + \ln\left(\frac{1}{\delta}\right) \right)$$

then C is PAC - identifiable (without noise).

This results can be used for determining the sample complexity for rule learning systems of

kDNF's. The VC - dimension of the concept class of all rule sets or kDNF is the number of possible terms. So the above results can be applied by substituting VCD(C) to *M*.

In the result from Natrajan *n'* is a length parameter which specifies the number of possible variables. In our case $n' = n\,l$ where *n* is the number of attributes and *l* is the maximal number of possible values for each attribute.

Because this results are based on a worst case analysis, the number of needed examples is much higher than the number available in practical applications. This results can be improved by assuming a underlying distribution or by reducing *M*.

In the next section we describe how the set *M* can be reduced to a small fraction by specifying a α threshold. If we assume that a concept is learnable in respect to a give α value the number of needed examples becomes much smaller.

5. Trade - off between finding a approximate good and a compact set of rules

After we have described the syntactical equivalence between the set of kDNF's and the set of rules we want to figure out some differences between the semantics of learning kDNF and rule induction within RULEARN. The goal of learning the kDNF as described before is to find a concept description which differs from the 'true" concept at most ε assuming that the true concept can be described by a kDNF.

The idea of finding rules within the RULEARN system is to generate a compact rule set which consists of only a few general and precise rules. As a result of this, the goal is not to be as close to the "true" concept as possible. In terms of the kDNF this means that there should be only a few terms.
There is a trade - off between the correctness and the compactness of the classification.

Figure 2 and 3 illustrate the different view to classification.

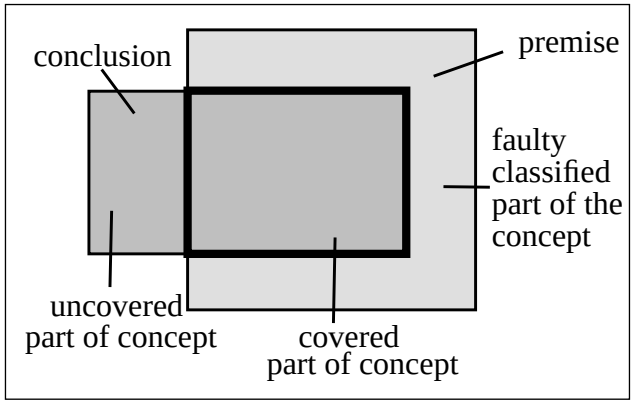

Figure 2: describing a rule for the best cover

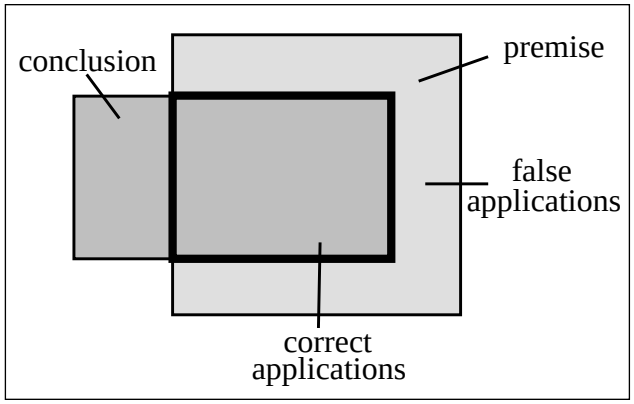

Figure 3: Describing a rule in general

The main difference is that the RULEARN system does not care about the part of the concept which is not described by the rule. In contrast to that the kDNF construction algorithm and also most rule-learning systems try to find a ε-close cover in which the unclassified parts of the concept are counted as faults.

In order to construct a compact rule set from the set of kDNF's which cover the "true" concept with a symmetric difference from at least ε, we delete all terms which do not fulfill the demanded thresholds. In the following we describe the fault which is done by dropping out some rules which cover a part of the "true" concept.

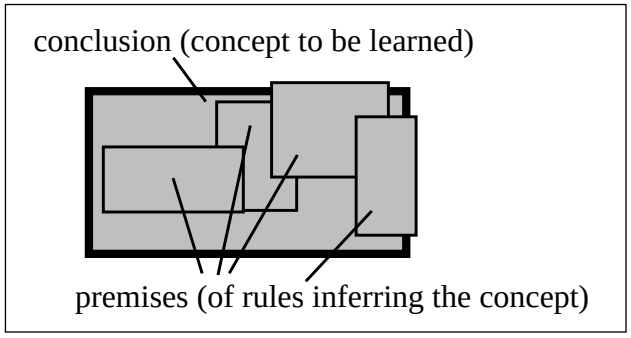

Figure 4: Finding a compact rule set describing the concept

The threshold for the universality reduces the set of possible rules (which correspond to the terms in the kDNF) by demanding that there must be at least a number of occurrences in which the premise is fulfilled. All rules or terms describing less examples are dropped out. The set of possible hypotheses H (hypotheses space) is the power set of the set of possible rules (terms of the DNF) M. Because $|H|$ is exponential in the number of rules ($H = 2^M$), a reduction of M reduces the hypotheses H very efficient.

The process of defining a threshold for the universality can be compared with defining a maximal number of literals k in a term to reduce the search space from DNF to kDNF. This is also a reduction of the set of terms. After describing the reduction we compare the results with kDNF.

Proposition 1

For each fixed set of attributes there are at most $\frac{1}{\alpha}$ *combinations of attribute values which have a universality of at least* α.

Proof

A: We first evaluate the case when the rule consists of only one element in the premise.

If the attribute has more than $\frac{1}{\alpha}$ attribute values which occur at least α times, the sum of all occurrences is greater than the number of examples, which is a contradiction.

B: If the rule consists of k components (literals), there are at most l^k possible combinations of attribute values for k fixed attributes where l is the maximal number of different attribute values. Because the sum of all occurrences of each combination is the number of examples, there can be at most $\frac{1}{\alpha}$ combinations which occur at least α times. ■

In the calculation we only consider rules in which all attribute values in a premise (term) are taken from different attributes, because the conjunction of two components (literals) of the same attribute is always false. In the reduction to the DNF described before such combinations are possible. This increases the set of rules and the hypotheses space more than actually needed because such rules do not make sense.

Theorem 1

Let l be the maximal number of different attribute values, then the sum of all possible rules in respect to α: M_α is for all $\alpha > \frac{1}{l}$:

$$M_\alpha \leq \sum_{i=1}^{n} \binom{n}{i} \frac{1}{\alpha} \qquad (2)$$

where n is the number of attributes.

Proof

From Proposition 1 there are at most $\frac{1}{\alpha}$ attribute values for each combination of attributes. The sum of all possible combinations of attribute values is $\sum_{i=1}^{k} \binom{n}{i}$.■

For low values for α l^i might be larger than $\frac{1}{\alpha}$, so that result of the theorem can be improved.

Because the number of possible combinations of k (fixed) attributes is $\prod_{i=j_1}^{j_k} |X_i|$, this is the number of possible values for each combination.

For each $\alpha \geq \frac{1}{\prod_{i=1}^{n} |X_i|}$ there is a constant *d*, so that $\frac{1}{\prod_{i=1}^{d} |X_i|} < \alpha \leq \frac{1}{\prod_{i=1}^{d-1} |X_i|}$.

For all rules containing more than d components (terms with more than d literals), α reduces the number of rules to be considered. From (3) we can therefore derive:

$$M_\alpha \leq \sum_{i=1}^{d} \binom{n}{i} l^d + \sum_{i=d+1}^{n} \binom{n}{i} \frac{1}{\alpha} \qquad (2.1)$$

Because there are at most l^d combinations of attribute values, from (2) and (2.1) we derive:

$$M_\alpha \leq \sum_{i=1}^{n} \binom{n}{i} \min \{ \frac{1}{\alpha}, l^i \} \qquad (2.2)$$

If we reduce the search from DNF to kDNF the number of possible terms M^k is

$$M^k \leq \sum_{i=1}^{k} \binom{n}{i} l^i \qquad (2.3)$$

Because this is only the exponent for the number of hypotheses, the α-reduction is a very efficient way of reducing the hypotheses space.

Notice that we made no assumption about the distribution which generates the examples from the sample space. In the derivation we used the worst case in which each attribute value occurs the same number of times. The more the occurrences of the attribute values differ, the lower is the number of possible rules with respect to α.

The effect of defining a minimum universality α on the hypotheses space is shown in figure 5.

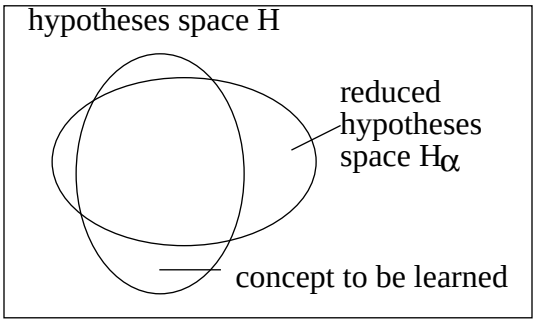

Figure 5: Reduction of the hypotheses space

The threshold for α may cause only a subset of the possible concept to be learned. If α is less than

$\frac{1}{m}$ and each possible rule is applicable at least once, than H_α is equal H. If α is near to one then H_α may be empty. In this case the concept is not learnable with respect to α. As discussed before, instead of finding an exact description of the concept, the algorithm finds a subset so that each element describes a part of the "true" concept with at least a given probability. This problem is equivalent to learning a kDNF for an unknown concept with a fixed *k*. If we assume that the concept may be described with the chosen representation, the algorithm will find it.

In the following we assume that there is a concept to be learned *h* and there exists a DNF *c*, so that

$$p\,(p_D\,(h \nabla c < \varepsilon)\,) \geq 1 - \delta \qquad (1)$$

Let t be a term, |t| denotes the number of literals in t and we call |t| the length of t. Let $c \in DNF$, than $|c| = \sum_{t \in c} |t|$ and we call |*c*| the length of *c*. The function *cover*(t) gives the subset of possible examples with respect to the given distribution which is covered by *t*. We define $c_\alpha \subseteq c = \{t \in c \,|\, |cover\,(t)| > \alpha m\}$ the terms in the given DNF c which cover at least α percent of the data and $c_k \subseteq c = \{t \in c \,|\, |t| \leq k\}$ the terms which have at most k literals.

We want to compare the hypotheses reduction of a DNF *c* to:

- to a kDNF c_k
- to a subset c_α

For $k_{max} = max\,\{|t| \in c\}$ and $\alpha_{min} = min\,\{|cover\,(t)|\,|t \in c\}$ clearly $c_{k_{max}} = c_{\alpha_{min}} = c$.

The question which remains is: "What is the maximum fault in respect to *c* which is made by defining an α higher than α_{min} or a k less than k_{max}?"

1. $\alpha > \alpha_{min}$

There exists a set of terms $t = t_1,...,t_l$ with $\forall\,(i \leq l)\; t_i \in c \wedge t_i \notin H_\alpha$. Let |*cover*(t)| = m_t and |*cover*(c-t)| = $m_{c\text{-}t}$ then $m_t + m_{c\text{-}t} >= m$. The sum is equal to m if the two sets are disjunct.

In formula (1) ε is a upper bound for the
uncovered examples $u = \{x \,|\, x \in cover\,(h) \wedge x \notin cover\,(c)\}$ and the faulty covered examples

$f = \{x | x \notin cover(h) \wedge x \in cover(c)\}$. In the worst case *cover*(t) and f are disjunct, this means that the terms which cover less than α percent of the data don't imply any faulty classification.
In other words, the probability for a faulty classification of the remaining set of terms increases because the terms which are dropped out have no missclassification.

In contrast to the standard PAC identification we want to distinguish between
- the probability that an example is classified false
- the probability that an example is uncovered

The phrase $h \nabla c$ includes both cases.

For the first case we can derive from (1)

$$p(p_D(\text{missclassification of } (c_\alpha) < \varepsilon \frac{cover(c)}{cover(c_\alpha)})) \geq 1 - \delta \qquad (1.1)$$

Because ε is a upper bound for the missclassification, in the worst case each term which is dropped out has no missclassification. In this case the smaller remaining subset has the same (absolute) amount of missclassification. The percentage of missclassification decreases with the number of dropped out terms.

Of course the probability that an example of the true concept is classified decreases by specifying α. The amount of lost coverage is $\bigcup_{t \in c \wedge t \notin c_\alpha} cover(t)$, so in the worst case is if the dropped out terms are disjunct ($\bigcap_{t \in c \wedge t \notin c_\alpha} cover(t) = \emptyset$).

If the set a DNF contains lots of specific terms, it also contains a super set which contains the smaller terms. To be precise we refer to the definition of a **sunflower** as: A family of sets S is called a sunflower if there is a set C, called center, such that for every $A, B \in C$ if $A \neq B$ then $A \cap B = C$.

The theorem by Erdoes and Rado [Erdoes Rado 60] claims that if there is a family of sets F and two integers v and w, so that every element in F has size at most v and $F > v!\,(w-1)^v$ than F contains a sunflower of size w. By using this result we can guarantee that if the description of h con-

tains $v!\,(w-1)^v$ terms it has a sunflower which is not dropped out if the threshold for α is increased to drop out the small terms.

2. $k < k_{max}$

We now consider the case where we reduce the concept c to a kDNF c_k with k less than k_{max}.
In this case there exists a set of terms $t_1, \ldots, t_j \mid \forall i \le j\,(t_i \in c \wedge t_i \notin c_k)$ with $|t_i| > k$. Because $cover(c) \subset cover(c_k)$ we must only consider missclassifications. Each of this terms misses at least one subterm (at most one literal) l_i. In the worst case t_i classifies all examples where $t_i \wedge l$ classifies only examples within h. From (1) we derive

$$p\left(p_D\left(h \nabla c_k < min\{1, \varepsilon + \frac{\bigcup_{i \le j} cover(t_i) \cap cover(l_i)}{cover(TRUE)}\}\right)\right) \ge 1 - \delta \qquad (1.3)$$

If one of the terms l_i covers all examples the upper bound is 1.

In comparison of both reduction methods the α-reduction decreases the number of applications (i.e. number classifications) of a rule by keeping the probability of a missclassification small. On the other hand the reduction on the term length increases the number of applications of the rule set by allowing a high probability of missclassification.
In that sense the α-reduction is more secure. If the rule is applicable, then the probability that the result is correct is within a reasonable bound.

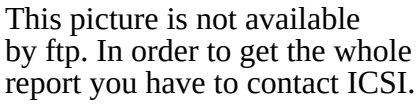

Figure 5 a) Number of possible rule-components (terms) in relation to different values for α and k in linear scale

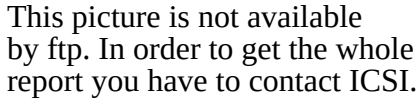

Figure 5 b): Number of possible rule-components (terms) in relation to different values for α and k in half-logarithmic scale

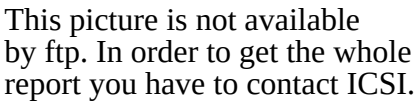

Figure 6: Number of possible rule-components (terms) in relation to different values for α and k in logarithmic scale (in linear scaling the alpha reduction can not be distinguished from the x-axis).

6. RULEARN algorithm

The RULEARN algorithm can be described as

1. determine the set of possible rules with respect to α
2. test, whether the rule fulfills the thresholds for σ and π
   - true: store rule and test the next
   - false: look for a new rule to be tested

The idea is to find as high thresholds as possible in order to keep the rule set small. Given a true rule $r \in h$ the system should have thresholds to find r even if the examples are affected by noise. If we use the noise model in which each attribute is affected by noise, we can derive a threshold for the reliability π.

If $r = a_{i_1, j_1} \wedge a_{i_2, j_2} \wedge \ldots \wedge a_{i_k, j_k} \rightarrow a_{n, j_{k+1}}$ the counterexamples which must be considered fulfill the premise, but have a different value for the conclusion attribute value a_{n, j_l} with $j_l \leq |X_n| \wedge j_l \neq j_{k+1}$. Because each component in the attribute value vector can be affected by

noise, a counterexample is an example in which at least one of the components for the conclusion is changed. In the noise model where each attribute value is affected by noise the probability that a correct true value (1) is set to 0 is $\frac{\beta}{2}$, which is equal to the probability that a correct false value (0) is 1. If the examples are generated independently from an oracle, the probability that an example becomes a counterexample is $1 - (1 - \frac{\beta}{2})^{|X_n|}$. So if the reliability threshold is less than this value the rule would be generated.

The influence of the universality (α) has been described in the previous sections. Independent from this, if the specialization is set to *m* and the universality values are set to 0, no “true” rule is dropped out.

6. Conclusions

We have represented a very efficient way of reducing the search space that can be used for learning rules or kDNF formulas. The reduced space decreases exponentially with only a small fault for missclassification. In contrast to syntactical reduction, thresholds in the number of literals within a term, the α-reduction gives a easy semantic interpretation. It allows the user to have a deeper understanding about the definition of the threshold and is easier to adjust.
Combining both reduction techniques gives a even better reduction of the search space.

7. References

[Bloendorn Michalski 91] Bloendorn E. and Michalski R. S.: Data-driven Constructive Induction in AQ17-DCI: A Method and Experiments, Reports of Machine Learning and Inference Laboratory, Center for Artificial Intelligence, George Mason University, 1991
[Clark Niblett 89] Clark P. and Niblett T. : The CN2 induction algorithm, Machine Learning. 3:261-283, 1989
[Erdoes Rado] Erdoes, P. and Rado R.: Intersection theorems for systems of sets, Journal of the London Math. Society, vol.35 pp. 85-90, 1960
[Fehsenfeld et al 91] Fehsenfeld B., Harris S., Koch T. and Kirchheiner R. : Automatic learning with RULEARN in the example of corrosion tests, in Technische Mitteilungen Krupp 1/1991
[Kearns 89] Kearns M. J.: The Computational Complexity of Machine Learning, MIT press, 1989
[Kirchheiner Koch 92] Kirchheiner R. and Koch T. : Computer aided learning of corrosion rules

from factual data bases, Proceedings of the NACE-CORRSION ‘92, Nashville 1992

[Koch 88] Koch T.: Effizientes Lernen und Bewerten von Regeln, Kuenstliche Intelligenz Informatik Fachberichte 181, 186-195, Springer Berlin 1988

[Michalski 83] Michalski R. S.: A Theory and Methodology of Inductive Learning in “Machine Learning: An Artificial Intelligence Approach, Michalski R.S.,Carbonell J. and Mitchell T. (Eds.) pp. 83-134, Morgan Kaufmann Publishing Co., Mountain View, CA, 1983

[Natrajan 91] Natrajan B. K. : Machine learning: A theoretical approach, Morgan Kaufmann Publishers 1991

[Pfeiffer et al 88] Pfeiffer J., Siepmann T. and Teichmann W.: Expert system for metal machining practise, in Technische Mitteilungen Krupp 46 (1988) pp 113 - 124

[Shackelford, Volper 88] Shackelford G. and Volper D. : Learning k-DNF with noise in thee Attributes. Proceedings of the first Annual ACM Workshop on Computational Learning Theory 1988, 97-103

[Quinlan 86] Quinlan J.R.: Induction of decision trees, Machine Learning, 1:81-106, 1986

[Utgoff 89] Utgoff P.E.: Incremental Learning of Decision trees., Machine Learning 4:161-186, 1989

[[Valiant 84] Valiant L.G. : A Theory of the learnable, Communications of the ACM, 27:11, 1134-1142, 1984

[Valiant 85] Valiant L.G. : Learning disjunctions of conjunctions,Proc. of the 9 th IJCAI, Los Angeles California, Morgan Kaufman 1985, pp 560-566

[Vapnic and Chervonenkis 71] Vapnik V. N. and Chervonenkis A. Ya.: On uniform convergence of relative frequencies of events to their probabilities , Theory of Probability and its Applications, 16(2).1971, pp. 264-280

Inductive learning of compact rule sets by using efficient hypotheses reduction

Thomas Koch[1]

TR 92-069
October 1992

Abstract:
A method is described which reduces the hypotheses space with an efficient and easily interpretable reduction criteria called α - **reduction.** A learning algorithm is described based on α - reduction and analyzed by using probability approximate correct learning results. The results are obtained by reducing a rule set to an equivalent set of kDNF formulas.
The goal of the learning algorithm is to induce a compact rule set describing the basic dependencies within a set of data. The reduction is based on criterion which is very flexible and gives a semantic interpretation of the rules which fulfill the criteria. Comparison with syntactical hypotheses reduction show that the α - reduction improves search and has a smaller probability of missclassification.

1. ICSI on leave from Krupp Forschungsinstitut GmbH, HA Informationstechnik, Muenchener Str. 100, W- 4300 Essen 1, Germany

Acknowledgment

I thank Anna Morpurgo, Klaus-Peter Jantke, Michael Luby and Philip Kohn for the fruitful discussions about draft versions. I'm deeply grateful to Prof. Jerome Feldman has supported my work here very strongly and to the management of the Krupp Forschungsinstitut, namely to Dr. Bernd Schoenwald, Dr. Bernd Fehsenfeld and Dr. Diethard Bergers who sent me to Berkeley in order to improve research.

1. Introduction

Rule based systems are the standard technique used in expert systems. Because of the difficulties in extracting rules from a human expert, automatic rule generation is a helpful way to bridge the knowledge acquisition bottleneck.
The scenario can be described as given a data set (i.e. set of examples) from a database, construct a set of rules which describe the dependencies within the data set. Systems that perform this task are for example ID3 family [Quinlan 86] and [Utgoff 89], AQ15 [Michalski 83] or CN2 [Clark, Niblett 89] and RULEARN [Koch 88]. The questions which arises after finding the rules can be formulated as: "With what level of confidence do the generated rules describe the "true" dependencies in the data set?"
This question is solved by showing that the task of finding rules is PAC (probability approximately correct) identifiable. Given a representation, a function describing the dependencies between the size of the data set and probability and confidence can be formulated. In general this can be done independently from the probability distribution which generates the examples (data set) from the sample space.
In this approach the PAC identification is done by constructing a set of kDNF's which are equivalent to a set of rules. Because the kDNF Problem has been analyzed within the many PAC publications (f. e. [Shackelford and Volper 88],[Kearns 89)] these results can also be used for rule-based systems.
We also describe the hypotheses reduction method used in the RULEARN system, called α - reduction within the framework of PAC. This technique can be used in various applications beside Machine Learning. It is very efficient, gives a semantic interpretation of the reduction and enables an algorithm to find compact descriptions with only a small probability of missclassification.
This report describes the rule learning system RULEARN within the PAC learning framework. The RULEARN software has been developed at the Krupp Forschungsinstitut GmbH [Fehsenfeld et. al 91] and has been applied to various data sets from all sections of the production process from product development up to after sales. RULEARN is used as a data evaluation method but also as a knowledge acquisition tool for building expert systems [Kirchheiner and Koch 92]. For reference of expert system development see for example [Pfeiffer et al 88].

In section 2 we describe the ideas of the RULEARN system, which uses three user defined evaluation criteria to distinguish between good and bad rules.
In the following section we show the equivalence of concept descriptions using a set of rules and a set of kDNF's.
In section 4 we describe the PAC identification of a kDNF. This allows us to derive the number of needed examples to ensure that the found solution is ε - close to the "true" description with at least a given probability (sample complexity).
After this we describe the trade - off between search effort and the fault of a solution using a reduced search space. We formally describe the reduction of specifying a threshold in the number of literals in a term and on the α - value of a term. By using the PAC framework we can derive upper bounds on the maximal fault. We also show how the search is reduced within typical examples.
Section 7 outlines the search for rules within RULEARN. In the last section we summarize the results and compare the differences between the different reduction methods.

2. Rule generation methods

Within an expert system shell the programmer can formulate complex rules, which for example contain arbitrary functions in premise and conclusion in the underlying programming language. In this case they are nearly as powerful as the underlying programming language. In order to find rules automatically the rule representation must be restricted to a specific subset. In the following we use propositional logic as representation language.
We assume that each of the given examples can be expressed by a finite number *n* of attributes $A_1,...,A_n$ with possible values $X_1,..., X_n$. The sample space for an attribute value based inductive system can be described by the sample space $\aleph = X_1 \times X_2 \ldots \times X_{n-1} \times X_n$, where each attribute X_i has a finite set of possible values denoted by $|X_i|$. To handle unknown values a special attribute value "*" can be included in the set of possible values.
An example can be written as a vector $(x_1, x_2, \ldots, x_n)$. Each rule component describing possible examples has the form $a_{i,j}$ which means the attribute A_i has the value $x_j \in X_i$. For each example it can be tested whether $a_{i,j}$ has the value true or false. A rule can be written as $a_{i_1, j_1} \wedge a_{i_2, j_2} \wedge \ldots \wedge a_{i_k, j_k} \rightarrow a_{n, j_{k+1}}$ where $\forall (i \le k+1 \le n)\, j_i \le |X_i|$.

Without loss of generality we assume the last attribute is the one for which rules should be found (conclusion attribute).

This assumption is also typical for the induction of decision trees.

As an example for a rule induction method we choose the RULEARN machine learning system. The goal of RULEARN is to find a short (compact) set of rules from a given data set which

- describes as many data sets as possible
- describes as precisely as possible
- uses as few rules as possible.

There is a trade - off between the three goals so that all goals can not be optimized by one set of rules. The idea is to combine these goals so that a learned rule set covers each goal to at least a specific degree.
In order to do that, RULEARN uses three evaluation criteria which are measures of the rule quality. The user of the system defines thresholds for each of the measures.

The quality measures are defined as follows:

$$reliability\,(\pi) \;=\; \frac{\text{number of correct applications of the rule}}{\text{number of possible applications of the rule}}$$

$$specilization\,(\sigma) \;=\; \frac{\text{number of examples in which the conclusion is fulfilled}}{\text{number of examples in which the premise is fulfilled}}$$

$$universality\,(\alpha) \;=\; \frac{\text{number of possible applications of the rule}}{\text{number of examples}}$$

The measure of a rule is defined if the rule applicable is at least once.

The values of the criteria can have following values: $0 \le \pi \le 1$, $\frac{1}{m} \le \sigma \le m$ and $0 \le \alpha \le 1$ where m is the number of examples.

The settings of the demands determine the number of rules to be learned and the time which is needed to find the rules. If the demands are minimal, that means $\pi = 0$, $\sigma = m$ and $\alpha = 0$, RULEARN generates all possible rules. In general a set of at most 20 rules can be generated, which describe dependencies within the data. In the next sections we focus on the threshold for the universality (α) and show how the threshold for α reduces the search space.

3. Reduction

This section describes the formalization of rules using attribute value assignments. In the case the learning of rules is reduced to the learning of DNF's.

A **DNF** is a set of terms $t_1, t_2, \ldots, t_l$ where each term $t_i = x_1 \wedge x \wedge \ldots \wedge x_p$ is a conjunction of p literals where each literal is negated or not. If no literal is negated, we call the DNF monotone. If no term of the DNF has more than k literals, the DNF is called a **kDNF**.

We reduce the learning of kDNF to learning rules by using a set of kDNF's in which each element corresponds to a set of rules describing one conclusion attribute value. The result is a set of $|X_n|$ separate kDNF's. For each rule we construct a term in the kDNF which belongs to the conclusion attribute value which is described by the rule.

To construct a term which is equivalent to a given rule we define a set of vectors $\vec{a} = (\vec{a}[1], \ldots, \vec{a}[|\vec{a}|]) = (a_{1,1}, a_{1,2}, \ldots a_{1,|X_1|}, \ldots, a_{n,1}, \ldots, a_{n,|X_n|})$ called attribute-value-vector with

$|\vec{a}| = \sum_{i=1}^{n} |X_i|$ which contains all possible rule components $a_{i,j}$.

For each term in the kDNF we construct a vector $\vec{a}$ with $\vec{a}[i] = 1$ iff the corresponding rule component occurs in the actual rule. This component must be included in the kDNF which corresponds to the conclusion attribute value which is inferred by the rule.

A inference using the rules is equivalent to the testing of $|X_n|$ kDNF's one after another. If one kDNF has the value true, the conclusion value which corresponds to the actual kDNF can be inferred. If the rule set is consistent, at most one of the kDNF's is true for any possible combination of attribute values. If the rule set does not match the given example, none of the $|X_n|$ kDNF's is true.

Notice that this construction allows a given attribute to be represented by more than one value. For example the value for an attribute color may have the values "red" and "black". If a rule set should be generated in which each attribute has only two possible values and which classifies all examples, the attribute value vector needs to be only half the size.
As mentioned above, it is also possible that none of the rules match a given example even in the training set.
This property is used in the RULEARN system to keep the generated rule set small. If a rule set

must cover all training examples it would normally consist of many rules which are generally more complicated. The reason for this is that in general if there is an exception to a general rule, the exception must be included in the general rule (which makes the rule more complicated) and the exception must be described by an individual rule in order to describe all examples (which increases the number of rules).

4. PAC identification

Learning of kDNF's is one of the first problems investigated by Valiant in 1984 [Valiant 84]. The question: "How many examples do I need in order to ensure that my generated concept c differs only ε form the true "unknown" concept with probability at least 1 - δ?" has been solved under various conditions. We can use this results for the task of rule learning algorithms because we have reduced the problem of finding rules to finding a set of kDNF's.
Within PAC learning there are different ways to treat noise. The easiest way is to assume that there is no noise in the given examples. If you handle the noise you have to distinguish whether only the conclusion attribute is affected by noise (example has false classification) or whether each attribute is affected by noise (the "whole" example is noisy). The number of needed data depends on the used noise model.

4.1 Each attribute value is affected by a known noise factor.

In this noise model, each rule component $a_{i,j}$ (literal) is affected by a known noise rate β so that

$a_{i,j} = a_{i,j}$	with probability 1 - β
$a_{i,j} = 0$	with probability 0.5 β
$a_{i,j} = 1$	with probability 0.5 β

Shackelford and Volper [Shackelford, Volper 88] showed that the class of kDNF is **PAC - identifiable** which means there exists as algorithm A, so that for $\forall (c \in kDNF)$, $\forall (\varepsilon \geq 0)$, $\forall (\delta \geq 0)$ and for all distributions D using as Oracle function E(f) outputs a Boolean function *c* such that:

$$p\left(p_D\left(h \nabla c < \varepsilon\right)\right) \geq 1 - \delta \qquad (1)$$

where *h* is the "true" concept to be learned and $h \nabla c$ is a measure of the region in which *h* and *c* disagree. p_D is the probability given the distribution D.

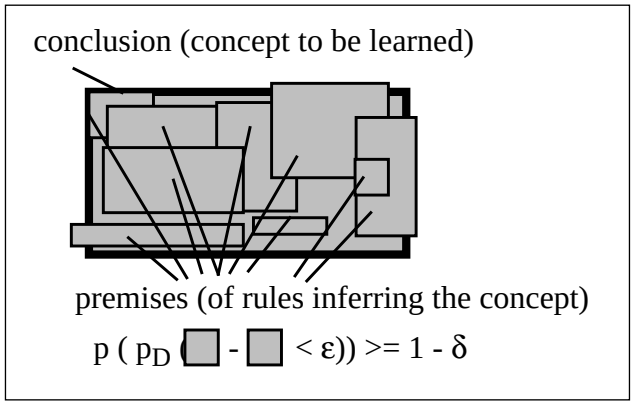

Figure 1: learning a kDNF which is ε close to the “true” concept

The algorithm to construct the kDNF essentially tests for all negative examples whether they are “true” negative or whether they are “true” positive examples which seem to be negative due to the noise. By assuming a specific noise model it can be found out which case is preferred. The terms which correspond to the negative examples are deleted so that the remaining set of terms describes the “true” concept within the demanded bounds. The algorithm finds the concept to be learned because it is assumed to be expressed by a kDNF.

They showed that if

$$m \geq \frac{2^{k+3} K^2}{\varepsilon^2 (1-\beta)^{2k}} \log\left(\frac{2^{2k+1} E}{\delta}\right)$$

then the probability is in the assumed bounds where K is the number of terms with at most k literals and E is the number of equivalence classes over the set of terms.

$K = \sum_{r=0}^{k} \binom{n}{r} 2^r \leq (2n)^{k+1}$ [Valiant 84] and the equivalence relation over the set of terms is defined so that two terms are in the same class iff they consist of exactly the same literals. Therefore $E = \sum_{i=0}^{k} \binom{n}{i}$. The number of needed examples is called the **sample complexity**. In section 5 we show that the RULEARN algorithm uses a reduced search space which is less complex.

4.2 Conclusion attribute is affected by noise

In this noise model we have the correct classification of a given example with probability 1 - β. The other attributes aren't affected by noise. This conditions have been analyzed by Valiant [Valiant 85]

Let M be the number of possible rule premises (terms), then $M \leq \sum_{i=1}^{n} \binom{n}{i} l^i$, where l is the maximal number of possible values for each attribute. If $\beta = \frac{\delta}{4|M|}$ and $\varepsilon = 2\beta$ then a concept h is PAC - identifiable by a DNF c if

$$m \geq \frac{36}{\delta} |M| \log\left(\frac{|M|}{\delta}\right) \quad .$$

4.2 Noise free case

Before we analyze the number of needed examples we introduce two definitions:
If C is a class of representations over X we say that C **shatters** a set $Y \subseteq X$ if the set $\{c \cap Y | c \in C\}$ is the power set of Y. The **Vapnic-Chervonenkis dimension** (or VC-dimension) of C denoted VCD(C) is the greatest integer d such that there exists a set of cardinality d that is shattered by C. In one sense the VC dimension is a measure of the number of "degrees of freedom" possessed by C. The VC -dimension was originally introduced by Vapnic and Chervonenkis [Vapnic and Chervonenkis 71].

Kearns [Kearns 89] derived a lower bound for the sample space complexity in the noise free case. If $0 < \varepsilon \leq \frac{1}{32}$, $0 < \delta \leq \frac{1}{1000}$ and $VCD(C) \geq 2$ then C is PAC - identifiable if $m \geq \frac{VCD(C) - 2}{128\varepsilon}$ where $VCD(C)$ is the Vapnic-Chervonenkis dimension.

If we don't specify bounds on ε and δ Natrajan [Natrajan 91] showed that if

$$m \geq \frac{1}{\varepsilon} \left((n' + 1)\, VCD(C) \ln(2) + \ln\left(\frac{1}{\delta}\right) \right)$$

then C is PAC - identifiable (without noise).

This results can be used for determining the sample complexity for rule learning systems of

kDNF's. The VC - dimension of the concept class of all rule sets or kDNF is the number of possible terms. So the above results can be applied by substituting VCD(C) to *M*.

In the result from Natrajan *n'* is a length parameter which specifies the number of possible variables. In our case $n' = n\,l$ where *n* is the number of attributes and *l* is the maximal number of possible values for each attribute.

Because this results are based on a worst case analysis, the number of needed examples is much higher than the number available in practical applications. This results can be improved by assuming a underlying distribution or by reducing *M*.

In the next section we describe how the set *M* can be reduced to a small fraction by specifying a α threshold. If we assume that a concept is learnable in respect to a give α value the number of needed examples becomes much smaller.

5. Trade - off between finding a approximate good and a compact set of rules

After we have described the syntactical equivalence between the set of kDNF's and the set of rules we want to figure out some differences between the semantics of learning kDNF and rule induction within RULEARN. The goal of learning the kDNF as described before is to find a concept description which differs from the 'true" concept at most ε assuming that the true concept can be described by a kDNF.

The idea of finding rules within the RULEARN system is to generate a compact rule set which consists of only a few general and precise rules. As a result of this, the goal is not to be as close to the "true" concept as possible. In terms of the kDNF this means that there should be only a few terms.
There is a trade - off between the correctness and the compactness of the classification.

Figure 2 and 3 illustrate the different view to classification.

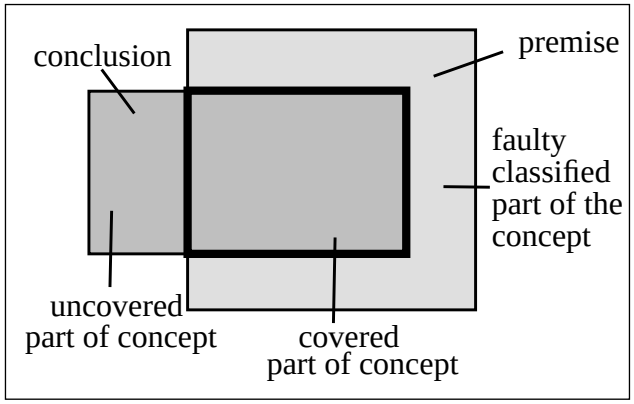

Figure 2: describing a rule for the best cover

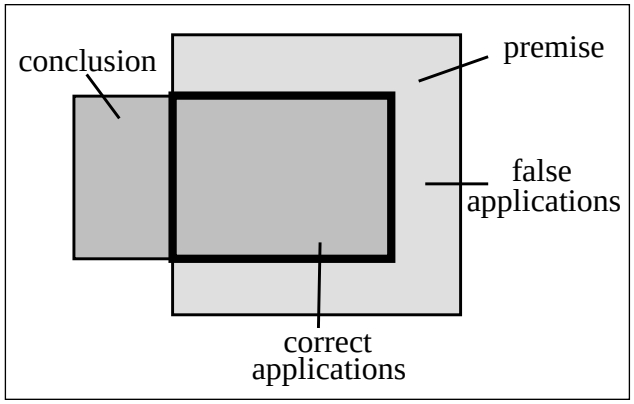

Figure 3: Describing a rule in general

The main difference is that the RULEARN system does not care about the part of the concept which is not described by the rule. In contrast to that the kDNF construction algorithm and also most rule-learning systems try to find a ε-close cover in which the unclassified parts of the concept are counted as faults.

In order to construct a compact rule set from the set of kDNF's which cover the "true" concept with a symmetric difference from at least ε, we delete all terms which do not fulfill the demanded thresholds. In the following we describe the fault which is done by dropping out some rules which cover a part of the "true" concept.

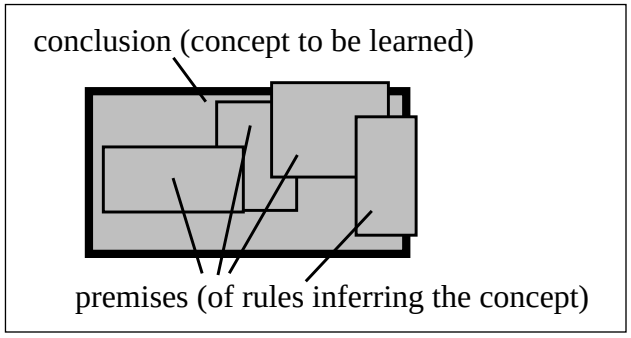

Figure 4: Finding a compact rule set describing the concept

The threshold for the universality reduces the set of possible rules (which correspond to the terms in the kDNF) by demanding that there must be at least a number of occurrences in which the premise is fulfilled. All rules or terms describing less examples are dropped out. The set of possible hypotheses H (hypotheses space) is the power set of the set of possible rules (terms of the DNF) M. Because $|H|$ is exponential in the number of rules ($H = 2^M$), a reduction of M reduces the hypotheses H very efficient.

The process of defining a threshold for the universality can be compared with defining a maximal number of literals k in a term to reduce the search space from DNF to kDNF. This is also a reduction of the set of terms. After describing the reduction we compare the results with kDNF.

Proposition 1

For each fixed set of attributes there are at most $\frac{1}{\alpha}$ *combinations of attribute values which have a universality of at least* α.

Proof

A: We first evaluate the case when the rule consists of only one element in the premise.

If the attribute has more than $\frac{1}{\alpha}$ attribute values which occur at least α times, the sum of all occurrences is greater than the number of examples, which is a contradiction.

B: If the rule consists of k components (literals), there are at most l^k possible combinations of attribute values for k fixed attributes where l is the maximal number of different attribute values. Because the sum of all occurrences of each combination is the number of examples, there can be at most $\frac{1}{\alpha}$ combinations which occur at least α times. ■

In the calculation we only consider rules in which all attribute values in a premise (term) are taken from different attributes, because the conjunction of two components (literals) of the same attribute is always false. In the reduction to the DNF described before such combinations are possible. This increases the set of rules and the hypotheses space more than actually needed because such rules do not make sense.

Theorem 1

Let l be the maximal number of different attribute values, then the sum of all possible rules in respect to α: M_α is for all $\alpha > \frac{1}{l}$:

$$M_\alpha \leq \sum_{i=1}^{n} \binom{n}{i} \frac{1}{\alpha} \qquad (2)$$

where n is the number of attributes.

Proof

From Proposition 1 there are at most $\frac{1}{\alpha}$ attribute values for each combination of attributes. The sum of all possible combinations of attribute values is $\sum_{i=1}^{k} \binom{n}{i}$.■

For low values for α l^i might be larger than $\frac{1}{\alpha}$, so that result of the theorem can be improved.

Because the number of possible combinations of k (fixed) attributes is $\prod_{i=j_1}^{j_k} |X_i|$, this is the number of possible values for each combination.

For each $\alpha \geq \frac{1}{\prod_{i=1}^{n} |X_i|}$ there is a constant *d*, so that $\frac{1}{\prod_{i=1}^{d} |X_i|} < \alpha \leq \frac{1}{\prod_{i=1}^{d-1} |X_i|}$.

For all rules containing more than d components (terms with more than d literals), α reduces the number of rules to be considered. From (3) we can therefore derive:

$$M_\alpha \leq \sum_{i=1}^{d} \binom{n}{i} l^d + \sum_{i=d+1}^{n} \binom{n}{i} \frac{1}{\alpha} \qquad (2.1)$$

Because there are at most l^d combinations of attribute values, from (2) and (2.1) we derive:

$$M_\alpha \leq \sum_{i=1}^{n} \binom{n}{i} \min \{ \frac{1}{\alpha}, l^i \} \qquad (2.2)$$

If we reduce the search from DNF to kDNF the number of possible terms M^k is

$$M^k \leq \sum_{i=1}^{k} \binom{n}{i} l^i \qquad (2.3)$$

Because this is only the exponent for the number of hypotheses, the α-reduction is a very efficient way of reducing the hypotheses space.

Notice that we made no assumption about the distribution which generates the examples from the sample space. In the derivation we used the worst case in which each attribute value occurs the same number of times. The more the occurrences of the attribute values differ, the lower is the number of possible rules with respect to α.

The effect of defining a minimum universality α on the hypotheses space is shown in figure 5.

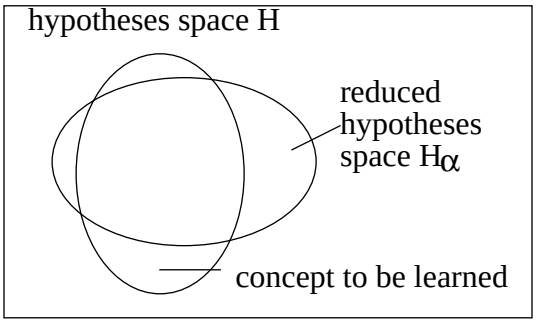

Figure 5: Reduction of the hypotheses space

The threshold for α may cause only a subset of the possible concept to be learned. If α is less than

$\frac{1}{m}$ and each possible rule is applicable at least once, than H_α is equal H. If α is near to one then H_α may be empty. In this case the concept is not learnable with respect to α. As discussed before, instead of finding an exact description of the concept, the algorithm finds a subset so that each element describes a part of the "true" concept with at least a given probability. This problem is equivalent to learning a kDNF for an unknown concept with a fixed *k*. If we assume that the concept may be described with the chosen representation, the algorithm will find it.

In the following we assume that there is a concept to be learned *h* and there exists a DNF *c*, so that

$$p\,(p_D\,(h \nabla c < \varepsilon)\,) \geq 1 - \delta \qquad (1)$$

Let t be a term, |t| denotes the number of literals in t and we call |t| the length of t. Let $c \in DNF$, than $|c| = \sum_{t \in c} |t|$ and we call |*c*| the length of *c*. The function *cover*(t) gives the subset of possible examples with respect to the given distribution which is covered by *t*. We define $c_\alpha \subseteq c = \{t \in c \,|\, |cover\,(t)| > \alpha m\}$ the terms in the given DNF c which cover at least α percent of the data and $c_k \subseteq c = \{t \in c \,|\, |t| \leq k\}$ the terms which have at most k literals.

We want to compare the hypotheses reduction of a DNF *c* to:

- to a kDNF c_k
- to a subset c_α

For $k_{max} = max\,\{|t| \in c\}$ and $\alpha_{min} = min\,\{|cover\,(t)|\,|t \in c\}$ clearly $c_{k_{max}} = c_{\alpha_{min}} = c$.

The question which remains is: "What is the maximum fault in respect to *c* which is made by defining an α higher than α_{min} or a k less than k_{max}?"

1. $\alpha > \alpha_{min}$

There exists a set of terms $t = t_1,...,t_l$ with $\forall\,(i \leq l)\; t_i \in c \wedge t_i \notin H_\alpha$. Let |*cover*(t)| = m_t and |*cover*(c-t)| = $m_{c\text{-}t}$ then $m_t + m_{c\text{-}t} >= m$. The sum is equal to m if the two sets are disjunct.

In formula (1) ε is a upper bound for the
uncovered examples $u = \{x \,|\, x \in cover\,(h) \wedge x \notin cover\,(c)\}$ and the faulty covered examples

$f = \{x | x \notin cover(h) \wedge x \in cover(c)\}$. In the worst case *cover*(t) and f are disjunct, this means that the terms which cover less than α percent of the data don't imply any faulty classification.
In other words, the probability for a faulty classification of the remaining set of terms increases because the terms which are dropped out have no missclassification.

In contrast to the standard PAC identification we want to distinguish between
- the probability that an example is classified false
- the probability that an example is uncovered

The phrase $h \nabla c$ includes both cases.

For the first case we can derive from (1)

$$p(p_D(\text{missclassification of } (c_\alpha) < \varepsilon \frac{cover(c)}{cover(c_\alpha)})) \geq 1 - \delta \qquad (1.1)$$

Because ε is a upper bound for the missclassification, in the worst case each term which is dropped out has no missclassification. In this case the smaller remaining subset has the same (absolute) amount of missclassification. The percentage of missclassification decreases with the number of dropped out terms.

Of course the probability that an example of the true concept is classified decreases by specifying α. The amount of lost coverage is $\bigcup_{t \in c \wedge t \notin c_\alpha} cover(t)$, so in the worst case is if the dropped out terms are disjunct ($\bigcap_{t \in c \wedge t \notin c_\alpha} cover(t) = \emptyset$).

If the set a DNF contains lots of specific terms, it also contains a super set which contains the smaller terms. To be precise we refer to the definition of a **sunflower** as: A family of sets S is called a sunflower if there is a set C, called center, such that for every $A, B \in C$ if $A \neq B$ then $A \cap B = C$.

The theorem by Erdoes and Rado [Erdoes Rado 60] claims that if there is a family of sets F and two integers v and w, so that every element in F has size at most v and $F > v!\,(w-1)^v$ than F contains a sunflower of size w. By using this result we can guarantee that if the description of h con-

tains $v!\,(w-1)^v$ terms it has a sunflower which is not dropped out if the threshold for α is increased to drop out the small terms.

2. $k < k_{max}$

We now consider the case where we reduce the concept c to a kDNF c_k with k less than k_{max}.
In this case there exists a set of terms $t_1, \ldots, t_j \mid \forall i \le j\,(t_i \in c \wedge t_i \notin c_k)$ with $|t_i| > k$. Because $cover(c) \subset cover(c_k)$ we must only consider missclassifications. Each of this terms misses at least one subterm (at most one literal) l_i. In the worst case t_i classifies all examples where $t_i \wedge l$ classifies only examples within h. From (1) we derive

$$p\left(p_D\left(h \nabla c_k < min\{1, \varepsilon + \frac{\bigcup_{i \le j} cover(t_i) \cap cover(l_i)}{cover(TRUE)}\}\right)\right) \ge 1 - \delta \qquad (1.3)$$

If one of the terms l_i covers all examples the upper bound is 1.

In comparison of both reduction methods the α-reduction decreases the number of applications (i.e. number classifications) of a rule by keeping the probability of a missclassification small. On the other hand the reduction on the term length increases the number of applications of the rule set by allowing a high probability of missclassification.
In that sense the α-reduction is more secure. If the rule is applicable, then the probability that the result is correct is within a reasonable bound.

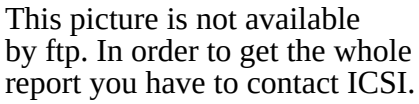

Figure 5 a) Number of possible rule-components (terms) in relation to different values for α and k in linear scale

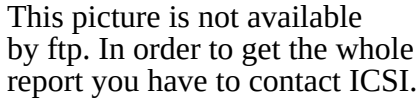

Figure 5 b): Number of possible rule-components (terms) in relation to different values for α and k in half-logarithmic scale

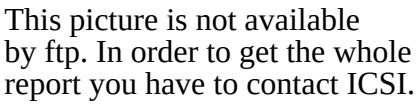

Figure 6: Number of possible rule-components (terms) in relation to different values for α and k in logarithmic scale (in linear scaling the alpha reduction can not be distinguished from the x-axis).

6. RULEARN algorithm

The RULEARN algorithm can be described as

1. determine the set of possible rules with respect to α
2. test, whether the rule fulfills the thresholds for σ and π
   - true: store rule and test the next
   - false: look for a new rule to be tested

The idea is to find as high thresholds as possible in order to keep the rule set small. Given a true rule $r \in h$ the system should have thresholds to find r even if the examples are affected by noise. If we use the noise model in which each attribute is affected by noise, we can derive a threshold for the reliability π.

If $r = a_{i_1, j_1} \wedge a_{i_2, j_2} \wedge \ldots \wedge a_{i_k, j_k} \rightarrow a_{n, j_{k+1}}$ the counterexamples which must be considered fulfill the premise, but have a different value for the conclusion attribute value a_{n, j_l} with $j_l \leq |X_n| \wedge j_l \neq j_{k+1}$. Because each component in the attribute value vector can be affected by

noise, a counterexample is an example in which at least one of the components for the conclusion is changed. In the noise model where each attribute value is affected by noise the probability that a correct true value (1) is set to 0 is $\frac{\beta}{2}$, which is equal to the probability that a correct false value (0) is 1. If the examples are generated independently from an oracle, the probability that an example becomes a counterexample is $1 - (1 - \frac{\beta}{2})^{|X_n|}$. So if the reliability threshold is less than this value the rule would be generated.

The influence of the universality (α) has been described in the previous sections. Independent from this, if the specialization is set to *m* and the universality values are set to 0, no “true” rule is dropped out.

6. Conclusions

We have represented a very efficient way of reducing the search space that can be used for learning rules or kDNF formulas. The reduced space decreases exponentially with only a small fault for missclassification. In contrast to syntactical reduction, thresholds in the number of literals within a term, the α-reduction gives a easy semantic interpretation. It allows the user to have a deeper understanding about the definition of the threshold and is easier to adjust.
Combining both reduction techniques gives a even better reduction of the search space.

7. References

[Bloendorn Michalski 91] Bloendorn E. and Michalski R. S.: Data-driven Constructive Induction in AQ17-DCI: A Method and Experiments, Reports of Machine Learning and Inference Laboratory, Center for Artificial Intelligence, George Mason University, 1991
[Clark Niblett 89] Clark P. and Niblett T. : The CN2 induction algorithm, Machine Learning. 3:261-283, 1989
[Erdoes Rado] Erdoes, P. and Rado R.: Intersection theorems for systems of sets, Journal of the London Math. Society, vol.35 pp. 85-90, 1960
[Fehsenfeld et al 91] Fehsenfeld B., Harris S., Koch T. and Kirchheiner R. : Automatic learning with RULEARN in the example of corrosion tests, in Technische Mitteilungen Krupp 1/1991
[Kearns 89] Kearns M. J.: The Computational Complexity of Machine Learning, MIT press, 1989
[Kirchheiner Koch 92] Kirchheiner R. and Koch T. : Computer aided learning of corrosion rules

from factual data bases, Proceedings of the NACE-CORRSION ‘92, Nashville 1992

[Koch 88] Koch T.: Effizientes Lernen und Bewerten von Regeln, Kuenstliche Intelligenz Informatik Fachberichte 181, 186-195, Springer Berlin 1988

[Michalski 83] Michalski R. S.: A Theory and Methodology of Inductive Learning in “Machine Learning: An Artificial Intelligence Approach, Michalski R.S.,Carbonell J. and Mitchell T. (Eds.) pp. 83-134, Morgan Kaufmann Publishing Co., Mountain View, CA, 1983

[Natrajan 91] Natrajan B. K. : Machine learning: A theoretical approach, Morgan Kaufmann Publishers 1991

[Pfeiffer et al 88] Pfeiffer J., Siepmann T. and Teichmann W.: Expert system for metal machining practise, in Technische Mitteilungen Krupp 46 (1988) pp 113 - 124

[Shackelford, Volper 88] Shackelford G. and Volper D. : Learning k-DNF with noise in thee Attributes. Proceedings of the first Annual ACM Workshop on Computational Learning Theory 1988, 97-103

[Quinlan 86] Quinlan J.R.: Induction of decision trees, Machine Learning, 1:81-106, 1986

[Utgoff 89] Utgoff P.E.: Incremental Learning of Decision trees., Machine Learning 4:161-186, 1989

[[Valiant 84] Valiant L.G. : A Theory of the learnable, Communications of the ACM, 27:11, 1134-1142, 1984

[Valiant 85] Valiant L.G. : Learning disjunctions of conjunctions,Proc. of the 9 th IJCAI, Los Angeles California, Morgan Kaufman 1985, pp 560-566

[Vapnic and Chervonenkis 71] Vapnik V. N. and Chervonenkis A. Ya.: On uniform convergence of relative frequencies of events to their probabilities , Theory of Probability and its Applications, 16(2).1971, pp. 264-280