Using Constraints to Building Version Spaces

Michèle Sebag

LMS-CNRS URA 317, Ecole Polytechnique, 91128 Palaiseau Cedex, France

Abstract. Our concern is building the set G of maximally general terms covering positive examples and rejecting negative examples in propositional logic.
Negative examples are represented as constraints on the search space. This representation allows for defining a partial order on the negative examples and on attributes too. It is shown that only minimal negative examples and minimal attributes are to be considered when building the set G. These results hold in case of a non-convergent data set.
Constraints can be directly used for a polynomial characterization of G. They also allow for detecting erroneous examples in a data set.

1 Introduction

The Version Space frame defines two bounds in the search space in empirical inductive learning [7] : the upper bound, set G, includes terms maximally general rejecting negative examples ; the lower bound, set S, includes terms maximally specific covering positive examples. Many works in the machine learning field shows how inspiring this frame is : to mention but a few, Smith and Rosenbloom [11] show that in propositional logic, in the case of a convergent data set (leading to $S = G$), learning only needs to consider those negative examples that are near-misses, in the sense defined by Winston [12]. H. Hirsh [5] defines a set of operations on Version Spaces and studies their computational complexity within a propositional formalism.

This paper focuses on building set G from positive and negative examples in propositional logic. Our motivations for building set G are both cognitive and pragmatic. From a cognitive point of view, human learning seems to perform specialization only when forced to by negative examples or instructors [9]. The practical advantages of such doing are clear : when learning from a few examples[1], specific learning leads to concepts of little further applicability.

But building set G is critical [2] : within a formalism as simple as the boolean one, its size may be exponential with respect to the number of attributes [4]. We propose, following the line of [5] and [8], a new formalization of the negative examples so to deal with the exponential size problem. Negative examples are formalized as constraints on the generalization ; a partial order on the negative examples is then derived. Therefore this formalisation results in extending the notion of near-miss and the polynomial results of [11] to a *non* convergent data

[1] This is the case for children ; this is the case also for many industrial problems where gathering examples is expensive.

set. It also allows for pruning the attributes of the problem domain. Moreover, it allows detection of erroneous examples in a ML-Smart -like way [1].

This paper is organized as follows. Section 2 formalizes negative examples as constraints on generalization. A partial order on negative examples is derived from the constraints ; it is shown that only minimal negative examples are necessary to learning. Section 3 defines the partial order on attributes derived from the constraints and shows that only minimal attributes are to be explored when building G. Section 4 focuses on expressing G and classifying a further case. The complexity of the proposed classification is in $\mathcal{O}(K \times P \times N)$, where K is the number of attributes, P the number of positive examples and N the number of negative examples. Section 5 describes how to detect erroneous examples by using constraints. Last, section 6 briefly reviews some related works.

2 From negative examples to constraints

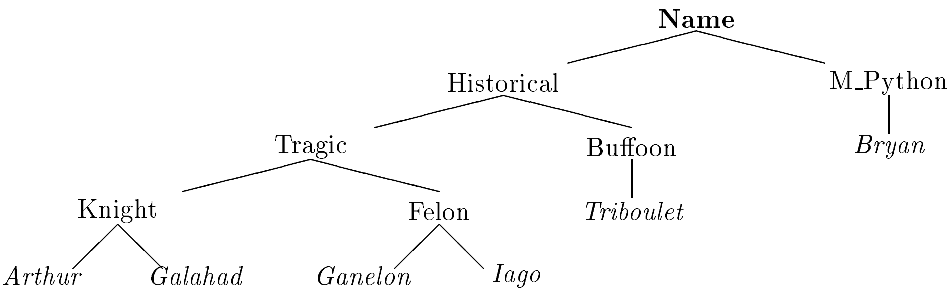
The problem domain is described by K attributes $x_1, .., x_K$; attributes are linear i.e. integer- or real-valued [6], or valued in a tree-structured domain. Given a conjunctive term S and a set of negative examples $Ce_1, \ldots Ce_N$, the solution space is that of conjunctive terms covering S and rejecting any negative example. In this section, our goal is to propose a representation of the negative examples enabling ordering and pruning of both the negative examples (2.4) and the attributes (in section 3).

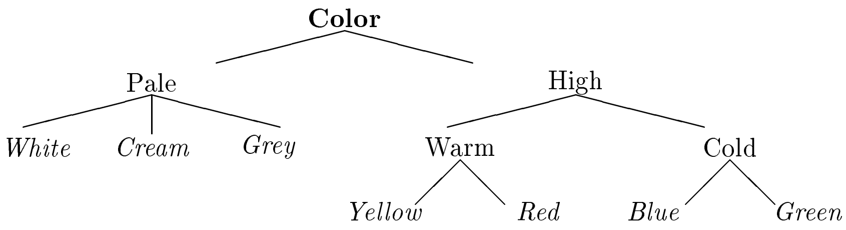
2.1 A negative example induces a constraint

Let us consider a toy problem for the purpose of illustration. The concept to learn is that of *Hero* ; attributes are the name of the person, its favorite color and the number of questions s/he asks :

	Name	*Color*	*Nb Questions*
Ex	*Arthur*	*Blue*	*3*
Ce_1	*Ganelon*	*Grey*	*7*
Ce_2	*Iago*	*Yellow*	*4*
Ce_3	*Bryan*	*Green*	*10*
Ce_4	*Triboulet*	*Cream*	*5*

Attribute *Nb_Questions* is linear; *Name* and *Favorite Color* are tree-structured :





Any discriminant generalization of example Ex must reject negative example Ce_1. In the case of attribute *Color*, the most general value covering $Blue$ and rejecting $Grey$ is $High$. The corresponding selector [6] rejecting Ce_1 thus is : $[Color = High]$. The disjunction of the most general selectors covering Ex and rejecting Ce_1 is :

$$[Name = Knight] \bigvee [Color = High] \bigvee [Nb\_Questions = [0,6]]$$

This disjunction can be thought of as a constraint upon the generalization of Ex : any term in the solution space must satisfy this logical constraint.

More generally, such a constraint upon the generalization of any positive term can be derived from any negative example. After proposing a representation of such constraints, we focus on their pruning (2.4) and handling in order to express G (4.1) or classify a further case (4.2).

2.2 Formalizing Constraints

In the following, the notation $x(T)$ stands for the value of attribute x in term T ; $x(T)$ thus is a qualitative value in the case of a tree-structured attribute, and an interval or a single value in the case of a linear attribute. The G set associated to a positive term S and negative examples $Ce_1, \ldots, Ce_N$ is noted G$(S, Ce_1, ..Ce_N)$.

Now, for any negative example Ce_j, for any attribute x_i,

• Let $E_{i,j}$ be, if it exists, the most general value (respectively the widest interval) such that it covers (resp. includes) value $x_i(S)$ and does not cover (resp. include) value $x_i(Ce_j)$ if attribute x_i is qualitative (resp. linear).

• Let $I(j)$ be the set of attributes such that value $E_{i,j}$ is defined. $I(j)$ denotes the set of possibly guilty attributes, according to the terminology of [11].

By definition, the most general selectors covering S and rejecting Ce_j (if we restrict to operator '='), are the $[x_i = E_{i,j}]$, for x_i in $I(j)$. So, one has :

$$G(S, Ce_j) \;=\; \bigvee_{x_i \in I(j)} [x_i = E_{i,j}]$$

The above expression, called *constraint induced by Ce_j on the generalization of S*, is given an extensional representation.

Let O_i be the domain of attribute x_i, and let Ω denote the cross product of domains O_i : $\Omega = O_1 \times \ldots \times O_K$. A constraint may then be represented as a subset of Ω.

Definition (constraint) :
Given term S, one associates to any negative example Ce_j the subset of Ω denoted $Constraint(S, Ce_j)$, defined by :

$$Constraint(S, Ce_j) = (V_{i,j})_{i=1}^{K}$$

$$where\ V_{i,j} = \begin{cases} E_{i,j}\ if\ E_{i,j}\ is\ defined & (\Leftrightarrow\ x_i\ \in I(j)) \\ \phi \quad otherwise \end{cases}$$

The empty value ϕ is assumed more specific than any value V_i in any domain O_i.

The negative examples described in 2.1 give rise to the following constraints :

	Name	Color	Nb Questions
$Constraint(Ex, Ce_1)$	$Knight\_Name$	$High\_Color$	$[0, 6]$
$Constraint(Ex, Ce_2)$	$Knight\_Name$	$Cold\_Color$	$[0, 3]$
$Constraint(Ex, Ce_3)$	$Historical\_Name$	$Blue$	$[0, 9]$
$Constraint(Ex, Ce_4)$	$Tragic$	$High\_Color$	$[0, 4]$

2.3 Ordering Constraints and Negative Examples

The partial order relations defined on domains O_i classically induce a partial order relation on their cross-product Ω, denoted $\leq_\Omega$. One has :

$$((V_i)_{i=1}^{K} \leq_\Omega (W_i)_{i=1}^{K}) \Longleftrightarrow (\forall i = 1..K,\ V_i \leq W_i)$$

where $V_i \leq W_i$ means that value V_i is covered by value W_i, if attribute x_i is tree-structured, and that interval V_i is included in interval W_i if x_i is linear.

This order relation enables comparing constraints and thus, negative examples.

Definition (nearest-miss) :
Given term S and negative examples $Ce_1, \ldots Ce_N$, Ce_i is called nearest miss *to S iff $Constraint(S, Ce_i)$ is minimal with respect to the order relation $\leq_\Omega$ among $Constraints(S, Ce_j)$, $j = 1..N$.*

Negative example Ce_2 is a nearest miss ; Ce_4 is not for we have :

$$Constraint(Ex, Ce_2) \leq_\Omega Constraint(Ex, Ce_4)$$

(*Knight* is less general than *Tragic* ; *Cold* is less general than *High* ; last, one has $[0, 3] \subset [0, 4]$). This shows that a nearest-miss is not necessarily a near-miss: Ce_2 is discriminated from Ex by more than one attribute.

2.4 Pruning negative examples

Given the above definition, a result parallel to that of [11] holds : bottom-up learning only needs positive examples and nearest-miss negative examples.
Proposition 1 :
Given positive term S and negative examples $Ce_1, \ldots Ce_N$, assume without loss of generality that nearest-misses are examples $Ce_1, \ldots Ce_L, L \leq N$. Then

$$G(S, Ce_1, \ldots Ce_N) = G(S, Ce_1, \ldots Ce_L)$$

Proof.

- We first associate to any ω in Ω, $\omega = (V_i)_{i=1}^K$, the disjunction of selectors $[x_i = V_i]$, for V_i not empty :

$$\omega \in \Omega \rightarrow g(\omega) = \bigvee_{i \ / \ V_i \neq \phi} [x_i = V_i]$$

It is straightforward to show that (where $\leq$ stands for 'less general than') :

$$\forall\, \omega_1,\ \omega_2\ \in\ \Omega,\ (\ \omega_1\ \leq_\Omega\ \omega_2\ )\ \Longleftrightarrow\ (\ g(\omega_1)\ \leq\ g(\omega_2)\ ) \qquad (1)$$

- By definition, one has

$$G(S, Ce_i) = g\ (Constraint(S, Ce_i))$$

and

$$G(S, Ce_1, \ldots, Ce_N) = \bigwedge_{i=1}^{N} (\ g\ (Constraint(S, Ce_i)))$$

- For any negative example Ce_j which is not a nearest-miss, $(j \geq L)$, there is a nearest-miss Ce_i with $Constraint(S, Ce_i) \leq_\Omega Constraint(S, Ce_j)$. Hence from (1), one has $g(Constraint(S, Ce_i)) \leq g(Constraint(S, Ce_j))$. So

$$g(Constraint(S, Ce_i)) \bigwedge g(Constraint(S, Ce_j)) = g(Constraint(S, Ce_i))$$

And

$$G(S, Ce_1, ..Ce_L) = G(S, Ce_1, ..Ce_L) \bigwedge G(S, Ce_{L+1}, ..Ce_N) = G(S, Ce_1, ..Ce_N)$$

□

This proposition then extends the result of [11] in case of a non convergent data set[2]. Therefore, negative examples that are not near-misses can be pruned without any loss of information.

[2] From [11], we have then : a data set is convergent iff there are at most N minimal constraints, each of them involving a single attribute.

2.5 Complexity

The notations used are that of Smith and Rosenbloom [11] : K denotes the number of attributes, P the number of positive examples and N the number of negative examples. It is assumed that exploring the domain hierarchy of any attribute can be done in constant time ; then

- Following [11], the complexity of building set S is in $\mathcal{O}(P \times K)$. Storing S requires a $\mathcal{O}(K)$ memory size.
- The constraints building is in $\mathcal{O}(N \times K)$ according to 2.2 ; their update when S is generalized is in $\mathcal{O}(N \times K)$ too[3]. The incremental building of the constraints so is in $\mathcal{O}(P \times N \times K)$. Their storage is in $\mathcal{O}(N \times K)$.
- The constraints pruning, according to definition 2.3, is in $\mathcal{O}(N^2 \times K)$.

3 Ordering and Pruning Attributes

A partial order on the attributes $x_1, \dots x_K$ of the problem domain can be derived from a set of constraints. It is shown that only minimal attributes with respect to this order are to be explored when building the G set.

3.1 Definition

Given constraints $C_1, \dots, C_L$, for any attribute x_i, we denote O_i^* the set of values $x_i(C_m)$ for $m = 1..L$. This set O_i^* induces a partition over the set of constraints, denoted $\mathcal{P}_i$: two constraints belong to the same subset if they have same value for attribute x_i. Subset $\mathcal{E}_{i,k}$ in partition $\mathcal{P}_i$ is the set of constraints C_m such that $x_i(C_m)$ equals value $U_{i,k}$, belonging to O_i^*.

So, if we consider the set of minimal constraints $C_1, \dots C_3$:

	Name	Color	Nb Questions
C_1	*Knight*	*High*	$[0, 6]$
C_2	*Knight*	*Cold*	$[0, 3]$
C_3	*Historical*	*Blue*	$[0, 9]$

we have $O_1^* = \{Knigt, Historical\}$, which induces the partition $\mathcal{P}_1 = \{\{C_1, C_2\}, \{C_3\}\}$.

The partitions over a given domain are partially ordered ; their order relation noted $\leq$, classically is defined by :

$$(\mathcal{P}_1 \leq \mathcal{P}_2) \Leftrightarrow (\forall\ \mathcal{E}_{1,i} \in \mathcal{P}_1,\ \exists\ \mathcal{E}_{2,j} \in \mathcal{P}_2\ /\ \mathcal{E}_{1,i} \subset \mathcal{E}_{2,j})$$

If two partitions $\mathcal{P}_i$ and $\mathcal{P}_j$ are such that $\mathcal{P}_i \leq \mathcal{P}_j$, we can define an application $\varphi_{i,j}$ from O_i^* to O_j^* which associates to value $U_{i,k}$ value $U_{j,l}$ such that for

[3] For every attribute x_i, for every constraint C_j, if x_i is in $I(j)$ (value $E_{i,j}$ is defined), it is checked whether $E_{i,j}$ still is more general than $x_i(S)$. If not, x_i is removed from set $I(j)$. If set $I(j)$ is empty for a constraint C_j, negative example Ce_j is no longer discriminated from S, and the version space fails.

any constraint C_m, $x_i(C_m) = U_{i,k} \Rightarrow x_j(C_m) = U_{j,l}$.
For instance, the partition $\mathcal{P}_3 = \{\{C_1\},\{C_2\},\{C_3\}\}$ derived from attribute *Nb_Questions* is finer than the partition $\mathcal{P}_1$. The corresponding application $\varphi_{3,1}$ is defined by :

$$\varphi_{3,1}([0,6]) = \varphi_{3,1}([0,3]) = Knight$$
$$\varphi_{3,1}([0,9]) = Historical$$

This partial order allows for defining a partial order on attributes :

Definition (finer attribute) :
Given a set of constraints $C_1,\ldots,C_L$, attribute x_i is finer *than attribute x_j, noted $x_i \leq_{att} x_j$ iff :*
- *Sets O_i^* and O_j^* are totally ordered with respect to set inclusion*[4]*.*
- *The partition induced by x_i is finer than the partition induced by x_j : $\mathcal{P}_i \leq \mathcal{P}_j$*
- *Function $\varphi_{i,j}$ defined above is monotonic from O_i^* into O_j^**

$$\forall U_1, U_2 \in O_i^*, ((U_1 < U_2) \Rightarrow (\varphi_{i,j}(U_1) \leq \varphi_{i,j}(U_2)))$$

- *For every k, constraint C_k involves attribute x_i ($x_i(C_k) \neq \phi$) iff it also involves attribute x_j.*

Attribute *Nb_Questions* is finer than attribute *Name* : the set of values O_3^* = { [0,3], [0,6], [0,9]} is ordered and $\varphi_{3,1}$ is monotonic.

3.2 Pruning attributes

The partial order defined on the attributes enables a result parallel to that of the section 2.4 : only minimal attributes with respect to this order are to be explored when building G.
Proposition 2 :
Let x_i and x_j be two attributes such that $x_i \leq_{att} x_j$.
Let G$(S, C_1, \ldots C_L)$ denote the set of maximally general terms covering term S and satisfying constraints $C_1, \ldots C_L$. Let G' be the subset of G given by the terms not involving attribute x_j.
For any term T in G', one defines term T^* by :
- if T does not involve attribute x_i, $T^* = T$.
- Otherwise, let $[x_i = V_i]$ be the selector involving x_i in T[5].
  - If V_i is such that $V_i = sup$ { V_k / $\phi_{i,j}(V_k) = \phi_{i,j}(V_i)$}, then T^* is the term obtained by replacing selector $[x_i = V_i]$ in T by selector $[x_j = \varphi_{i,j}(V_i)]$.
  - Otherwise $T^* = T$.

[4] If x_i is a tree-structured attribute then O_i^* is totally ordered : all values in O_i^* can be compared for they all are more general than $x_i(S)$ by construction. If x_j is a linear attribute, O_j^* is not necessarily totally ordered w.r.t. set inclusion.
[5] This selector is unique : only the most specific value V_i is retained.

Then any term in G either belongs to G', or is a T^* for some T in G'.

The proof is given in appendix.

In practice, set G' is obtained by considering minimal attributes only. (The attributes pruning thus only takes place during the expensive phase of the G building.) Then, G is obtained from G' by making straightforward use of functions $\varphi_{i,j}$. In the example, only attributes *Color* and *Nb_Questions* are considered to build G' ; the terms in set G' are :
$T_1 = [Color = Blue]$
$T_2 = [Color = Cold] \bigwedge [Nb\_Questions = [0,9]]$
$T_3 = [Color = High] \bigwedge [Nb\_Questions = [0,3]]$

Terms T_2 and T_3 give rise to terms T_2^* and T_3^*
$T_2^* = [Color = Cold] \bigwedge [Name = Historical\_Name]$
$T_3^* = [Color = High] \bigwedge [Name = Knight\_Name]$

3.3 Complexity

Pruning attributes involves, for any (ordered) pair of attribute (x_i, x_j) :

- Building application $\varphi_{i,j}$ from O_i^* into O_j^* ; the size of O_i^* is upper bounded by the number of constraints, (which is upper bounded by the number of negative examples N) and by the number of values in O_i. Hence, if L denotes the maximum number of values in any O_i, this step is in $\mathcal{O}(min(N, L))$. The storage of $\varphi_{i,j}$ also is in $\mathcal{O}(min(N, L))$.
- Checking the monotonicity of $\varphi_{i,j}$, which is in $\mathcal{O}(min(N, L)log(min(N, L))\ )$.

Finally, the complexity of the attributes pruning is in
$\mathcal{O}(K^2 \times min(N, L)log(min(N, L))\ )$.

4 Building G or Classifying ?

This section addresses the characterization of G from the constraints.

4.1 Building G

In a first step, S is built from the set of positive examples ; constraints are built and updated as detailed in 2.2 and 2.5.

Constraints are then explored in order to build the terms in G. Here is the pseudo-code of the constrained generalization algorithm[6]. Array *Selector[i]* stores the active selector of constraint C_i ; i_0 is the index of the current constraint.

<u>Initialize()</u>
For i = 1..N

[6] This algorithm is implemented in C^{++}.

```
        Selector[i] = 0;
    G = { } ; T = true ;  i0 = 1.
Main()
    Initialize();
    While (0 < i0 ≤ N),
        If (constraint C_i0 is not yet satisfied by term T)
            If (Selector[i0] ≠ 0)
                Remove from T the Selector[i0]-th selector of C_i0;
            Increment Selector[i0];
            Continue :
            If (C_i0 has Selector[i0] selectors)
                T = T ⋀ the Selector[i0]-th selector of C_i0;
            Else
                Selector[i0] = 0;
                If (Backtrack()) goto Continue;
                Else stop.
        Increment i0;
    EndWhile
    G = G ⋃ T ;// Term T is a solution
    if (Backtrack()) goto Continue; // to find other solutions.
    else stop.
Backtrack()
Let j be the index of the last active constraint (with Selector[j] ≠ 0);
    While (j > 0)
        Remove from T the Selector[j]-th selector of C_j;
        Increment Selector[j];
        If (C_j has Selector[j] selectors)
            i0 = j;
            return True;
        Else
            Selector[j] = 0;
            Decrement j;
    EndWhile
    return False;
```

This procedure allows to find all terms in G ; however, terms found may be not all maximally general : a selector added to satisfy a given constraint may become useless because of a selector added further on and satisfying also this constraint.

So one has to check whether a term T is maximally general or not. This is done by building $G(T, Ce_1, ..Ce_N)$; T is maximally general iff $G(T, Ce_1, ..Ce_N) = T$.

In spite of the pruning of the search space enabled by constraints, the number of conjunctive terms in G may still be exponential with respect to the number of attributes. However, the constraints are sufficient to characterize G with a polynomial complexity.

4.2 Using constraints to classify

Let E be the description of a further case. The diagnosis function is as usual given by

$$Diagnosis(E) = \begin{cases} True & if\ E \leq S \\ False & if\ E \not\leq G \\ Unknown & otherwise \end{cases}$$

The only point is checking whether E belongs to G, i.e. whether E satisfies all constraints $Constraint(S, Ce_i)$. This can be done with a complexity $\mathcal{O}(K \times N)$ (still assuming that exploring the generalization hierarchy of any attribute is done in constant time).

Therefore, the proposed formalization provides the user with a polynomial characterization of G, even in case of a non-convergent data set. This result is discussed with reference to that of H. Hirsh [5] in the last section.

5 Detecting erroneous examples

Dealing with noisy data has long been recognized an unavoidable task in machine learning [3]. However, detecting and rejecting outliers and/or erroneous examples could ease greatly the learning task [10]. This section deals with detecting erroneous examples in the data set. More precisely, it gives sufficient conditions for an example to be erroneous. An example is said to be erroneous if either its description or its conclusion differ from what it should be.

Negative examples are represented as constraints with respect to a positive term to generalize ; this representation holds, be this positive term either the actual S, or any positive example Ex_i in the data set. Let $C(E_i, Ce_k)$ denote the constraint put by negative example Ce_k upon the generalization of positive example E_i. This constraint is a boolean function computable on the problem domain. Should positive example E_j satisfy constraint $C(E_i, Ce_k)$? Yes : if E_j belongs to concept C, then E_j belongs to set G, and then it belongs to any set G$(E_i, Ce_1, \ldots, Ce_N)$ in case of a conjunctive concept C. So, let us denote $\delta(i, j, k)$ the boolean value $C(E_i, Ce_k)(E_j)$; this boolean should be true for any positive example E_j.

Assume now that $\delta(i, j, k)$ is false : then, according to the above discussion, either E_i or E_j or Ce_k is erroneous, i.e. has a corrupted description or conclusion. This fact only gives a hint ; but one can appreciate where the problem eventually comes from, by aggregating hints and considering all $\delta(i, j, k)$.

The procedure is inspired from the one exposed in [1]. Bergadano et al. state that along the specialization process, the number of examples belonging to the extension of the current term is to decrease ; the point is that, when specialization is based on a litteral defined by the domain theory - and *if this definition is too specific*, then this decreasing is much higher than the average decreasing of the extension size, averaged along the specialization process. When an unusual decrease is observed, the current predicate and its definition are submitted for correction to the expert. In the same line, we associate to example Ex_i the number of booleans $\delta(i,j,k)$ that are false, for j ranking from 1 to P and k from 1 to N. Let $\sigma(Ex_i)$ denote this sum. This allows to order positive examples by ascending order of "suspicion" : the greater $\sigma(Ex_i)$, the more likely Ex_i is erroneous. Similarly, one can associate to a negative example Ce_k the number of $\delta(i,j,k)$ that are false, for i and j ranking from 1 to P. This quantity allows for ordering negative examples by ascending order of suspicion.

Of course, the final decision to reject an example as erroneous belongs to the expert ; the only crisp information provided by our approach is to state that, given a boolean $\delta(i,j,k)$ false, one at least among examples Ex_i, Ex_j and Ce_k, is erroneous. In practice, we proceed as follows : the expert provides the system with a rate of erroneous examples. Then, until this rate of examples has been rejected, or until the stack is empty, the positive and negative examples maximizing function σ are proposed to the expert. If they are discarded by the expert, function σ is updated; otherwise, the next positive and negative examples maximizing σ are considered.

6 Related works

6.1 An exponential size

Among the related works, we must first mention Haussler [4] who showed that the number of conjunctive terms in set G could be exponential with respect to the number of attributes. The example is as follows :
The problem domain is $\{0,1\}^{2m}$. One is given positive example Ex, whose all components are true, and m negative examples Ce_i, $i = 1..m$; components of Ce_i are all true, except the i-th and the $(m+i)$-th.

Any negative example Ce_i leads to specialise set G ; this specialization may be done along any feature discriminating Ce_i from $Ex = S$, i.e. one of attributes i or $m+i$. The number of choices (and of conjunctive terms in G) is multiplied by 2 at each negative example. The final number of elements in G is thus 2^m.

Many strategies have been proposed so to deal with that number of choices and terms.

6.2 Using near-misses

Smith and Rosenbloom [11] first consider the negative examples which are discriminated from S by only one attribute (so there is no choice for specialization). Such negative examples are called near-misses, according to Winston [12]. A major result of [11] is to show that when the data set is convergent, learning only needs positive examples and near-miss negative examples to converge. Accordingly, they propose an algorithm linear with respect to the number of attributes K, the number of positive examples P and the number of negative examples N:

Negative examples are stocked in a waiting list.
When set S is generalized from a positive example, the waiting list is scanned : if there is a negative example which is a near-miss (i.e. with exactly one attribute discriminating this example from S), then G is specialized with respect to this guilty attribute so to reject the negative example.
This way, the memory size required is in $\mathcal{O}((N+2) \times K)$ (stocking of S, G and the waiting list).

This process leads to G = S if the data set is convergent. Otherwise, after having considered all positive examples and near-miss negative examples, the usual *Candidate Elimination Algorithm* [7] is used to update set G from the remaining negative examples.

6.3 Another representation of the G set

H. Hirsh [5] proposes to represent a Version Space by $[S, \mathcal{N}]$ where $\mathcal{N}$ stands for the list of negative examples. For conjunctive tree-structured languages, this representation supports a polynomial computation of some functions defined on a Version Space :
• *Collapse* : when data are inconsistent, or the description langage does not allow to describe the concept to learn ; *Collapse* is true if S is empty ; (see *Update* below);
• *Converge* : in case of convergence, from [11] there is a near-miss for any attribute in list $\mathcal{N}$;
• *Update* given new example E : if E is negative it is added to list $\mathcal{N}$ and elements in S covering it are removed ; otherwise E is used to generalize S, and resulting terms covering some example in $\mathcal{N}$ are removed ;
• *Classify* a new case E : if E satisfies any term in S, then it belongs to the concept ; otherwise, compute the Version Space that would result if E were a positive example ; if this new Version Space collapses, E does not actually belong to the concept ; otherwise, the diagnosis is unknown.

The drawback of this representation is that maintaining the list of negative examples gives few hints to what G could be. Addressing this remark, J. Nicolas [8] proposes a disjunctive formalization such that the G set induced by a single negative example is represented by a single term. The trouble comes from intersecting several Version Spaces when several negative examples are considered. This operation is very expensive ; the actual number of (disjunctive) terms may still be exponential.

6.4 Discussion

Our approach is very near from that of H. Hirsh, with a slightly higher complexity of our update (learning) phase ; the complexity of the classifying phase is equivalent to that of Hirsh : it is equivalent to check whether case E satisfies

Constraint (S, Ce_i), or whether the generalization of E and S covers negative example Ce_i.

From the intelligibility standpoint the constraint derived from a negative example is more general and thus understandable by the expert than the negative example itself (furthermore, by pruning the constraints the number of useful informations can be much decreased). So, the presented approach achieves some trade-off between efficiency and understandability. But the major advantage of our formalization compared to that of H. Hirsh, it that it enables to detect the erroneous examples by an all-at-once handling of the data.

Compared to the approach of J. Nicolas [8], the final expensive phase of the G building is performed in a reduced search space : constraints allow for pruning both negative examples and attributes. This phase can also be completely escaped as shown in 4.2.

7 Summary and Perspectives

Representing the negative examples as constraints on the generalization of a positive term enables to prune the negative examples and the attributes to be explored when building the G set.

This representation also allows for a computable polynomial characterization of G ; whatever the actual number of conjunctive terms in G, this characterization is linear with respect to the number of attributes, the number of positive examples and the number of negative examples. The price to pay lies in the fact that a set of constraints is less understandable by the expert than a set of conjunctive terms. Last, our approach enables detecting erroneous examples.

Further research aims at extending this approach to learning a disjunctive concept. The constraints building applies, be the positive term considered the S set or any positive example Ex_i. One may then consider the G set derived from a positive example Ex_i and the negative examples $Ce_1, \ldots Ce_N$, (the *star of* Ex_i by analogy with the star algorithm [6]). A next step is to cluster these stars, so to identify the (conjunctive) subconcepts involved in a disjunctive concept.

References

1. F. Bergadano, A Giordana, *A Knowledge Intensive Approach to Concept Induction*, ICML 1988, pp 305-317.
2. A. Bundy, B. Silver, D. Plummer, *An analytical Comparizon of Some Rule Learning Programs*, Artificial Intelligence, 27, 1985, pp 137-181.
3. P. Clark T. Niblett *Induction in noisy domains* Progress in machine learning, Proc. EWSL 1987, I. Bratko N. Lavrac Eds, Sigma Press.
4. D. Haussler, *Quantifying Inductive Bias : AI Learnign Algorithms and Valiant's Learning Framework*, Artificial Intelligence, 36, 1988, pp 177-221.
5. H. Hirsh, *Polynomial-Time Learning with Version Spaces*, Proc. National Conference on Artificial Intelligence, 1992 pp 117-122.

6. Michalski R.S. *A theory and methodology for inductive learning* Machine Learning: An Artificial Intelligence Approach, I, R.S. Michalski, J.G. Carbonnell, T.M. Mitchell Eds, Springer Verlag, (1983), p 83-134.
7. T.M. Mitchell, *Generalization as Search*, Artificial Intelligence Vol 18, pp 203-226, 1982.
8. J. Nicolas, *Une Représentation Efficace pour les Espaces de Version*, JFA 1993.
9. J. Piaget, *Six études de psychologie*, Denoel 1964.
10. R. Quinlan, *The effect of noise on concept learning* Machine Learning: An Artificial Intelligence Approach, I, R.S. Michalski, J.G. Carbonnell, T.M. Mitchell Eds, Vol 2, Morgan Kaufman, 1986.
11. B. Smith, P. Rosenbloom, *Incremental non-backtracking focussing : A polynomially- bounded generalization algorithm for version space*, Proc. National Conference on Artificial Intelligence, 1990, pp 848-853.
12. P.H. Winston, *Learning Structural Descriptions from Examples* The Psychology of Computer Vision, P.H. Winston Ed, Mc Graw Hill, New York, 1975, pp 157-209.

Appendix

Proposition 2 :
Let x_i and x_j be two attributes such that $x_i \leq_{att} x_j$.
Let G$(S, C_1, \ldots C_L)$ denote the set of terms maximally general covering term S and satisfying constraints $C_1, \ldots C_L$. Let G' be the subset of G given by the terms not involving attribute x_j.
For any term T in G', one defines term T^* by :
• if T does not involve attribute x_i, $T^* = T$.
• Otherwise, let $[x_i = V_i]$ be the selector involving x_i in T.

• If V_i is such that $V_i = sup \; \{ \; V_k \; / \; \phi_{i,j}(V_k) = \phi_{i,j}(V_i)\}$, then T^* is the term obtained by replacing selector $[x_i = V_i]$ in T by selector $[x_j = \phi_{i,j}(V_i)]$.
• Otherwise $T^* = T$.

Then any term in G either belongs to G', or is a T^* for some T in G'.

Proof.
A preliminary remark is the following : as set O_l^* is totally ordered for $l = i \; or \; j$[7], it induces a total order on partition $\mathcal{P}_l$ too. So selector $[x_l = U_{l,m}]$ enables to satisfy any constraint C_k such that $U_{l,m} \leq x_l(C_k)$, and more generally, any constraint belonging to some $\mathcal{E}_{l,j}$ in $\mathcal{P}_l$, with index j greater than m.

$$Let \; G" = \bigvee_{T \;\in\; G'} (T \bigvee T^*)$$

A.1 $G \subset G$"

Let Z be a maximal term in G.

• We first show that Z involves at most one among attributes x_i or x_j. Let us suppose that Z includes two selectors $[x_i = V_i]$ and $[x_j = V_j]$. Compare V_j and $\varphi_{i,j}(V_i)$:

• If $V_j \leq \varphi_{i,j}(V_i)$, then from the preliminary remark, all constraints satisfied by selector $[x_i = V_i]$ are satisfied by selector $[x_j = V_j]$ too. Hence selector $[x_i = V_i]$ can be suppressed - which contradicts the fact that Z is maximal.
• Similarly if $\varphi_{i,j}(V_i) \leq V_j$, then all constraints satisfied by $[x_j = V_j]$ are satisfied by $[x_i = V_i]$, which contradicts the fact that Z is maximal.

[7] By definition, if $x_i \leq_{att} x_j$ then sets O_i^* and O_j^* are totally ordered.

So Z involves at most one among attributes x_i and x_j.

• Suppose that Z does not involve x_j. Then by construction, Z belongs to G', and to G".

• Suppose that Z involves x_j and includes selector $[x_j = V_j]$. Let V_i be such that

$$V_i = sup\ \{V_k\ /\ \varphi_{i,j}(V_k) = V_j\}$$

Let T be the term defined from Z by replacing selector $[x_j = V_j]$ by $[x_i = V_i]$, which satisfies the same constraints by construction. We show that T belongs to G'. In opposition, suppose that there exists T' in G' such that $T' > T$.

- • If T' does not involve x_i, (as T' does not involve x_j neither for T' belongs to G'), then from $T' > T$ one has $T' > Z$; this contradicts the fact that Z is maximal.
- • So T' must involve attribute x_i ; assume that T' includes selector $[x_i = W_i]$; then $T' > T$ implies $W_i \geq V_i$.
  Consider now term Z', built by replacing in T' selector $[x_i = W_i]$ by $[x_j = \varphi_{i,j}(W_i)]$. By definition of $\varphi_{i,j}$, $W_i \geq V_i$ implies $\varphi_{i,j}(W_i) \geq V_j$; hence $Z' \geq Z$; however, Z' satisfies the same constraints as Z, and so belongs to G. But, Z being maximal in G, one has $Z = Z'$, so $\varphi_{i,j}(W_i) = V_j$; by definition of V_i, this implies $V_i \geq W_i$, which contradicts the fact $T' \neq T$.

Then there exists a term T in G' such that $Z = T^*$. So, all terms in G are obtained from G' by the procedure given in the proposition 2.

A.2 *G*" $\subset$ *G*

We show now that any term in G" belongs to G. By construction, if T belongs to G' then it belongs to G. It remains to show that all terms T^* as defined in proposition 2 eventually belong to G. Let T be a term in G' including selector $[x_i = V_i]$, with V_i such that :

$$V_i = sup\{V_k\ /\ \varphi_{i,j}(V_k) = \varphi_{i,j}(V_i)\}$$

Let T^* be the term built from T by replacing $[x_i = V_i]$ by $[x_j = \varphi_{i,j}(V_i)]$.

Suppose that there exists Z in G such that $Z > T^*$.

• If Z does not involve x_j, as Z does not involve x_i neither (for $Z > T^*$ and T^* does not involve x_i), Z belongs to G'. Hence $Z > T^*$ implies $Z > T$, which contradicts the fact that T is maximal.

• If Z involves x_j, then there exists in G' a term S such that $Z = S^*$ (from A.1). It is straightforward to show that $Z = S^* > T^*$ implies $S \geq T$; now, as T is maximal, $T = S$; so $Z = T^*$, and T^* is maximal.

□

This article was processed using the LaTeX macro package with LLNCS style