Disjunctive Learning with a Soft-Clustering Method

Guillaume Cleuziou, Lionel Martin, and Christel Vrain

LIFO, Laboratoire d'Informatique Fondamentale d'Orléans
Faculté des Sciences
Rue Léonard de Vinci B.P. 6759
45067 Orléans cedex2 - FRANCE
{cleuziou,martin,cv}@lifo.univ-orleans.fr

Abstract. In the case of concept learning from positive and negative examples, it is rarely possible to find a unique discriminating conjunctive rule; in most cases, a disjunctive description is needed. This problem, known as disjunctive learning, is mainly solved by greedy methods, iteratively adding rules until all positive examples are covered. Each rule is determined by discriminating properties, where the discriminating power is computed from the learning set. Each rule defines a subconcept of concept to be learned with these methods. The final set of sub-concepts is then highly dependent from both the learning set and the learning method.
In this paper, we propose a different strategy: we first build clusters of similar examples thus defining subconcepts, and then we characterize each cluster by a unique conjunctive definition. The clustering method relies on a similarity measure designed for examples described in first order logic. The main particularity of our clustering method is to build "soft clusters", i.e. allowing some objects to belong to different groups. Once clusters have been built, we learn first-order rules defining the clusters, using a general-to-specific method: each step consists in adding a literal that covers all examples of a group and rejects as many negative examples as possible.
This strategy limits some drawbacks of greedy algorithms and induces a strong reduction of the hypothesis space: for each group (subconcept), the search space is reduced to the set of rules that cover all the examples of the group and reject the negative examples of the concept.

1 Introduction

In this paper, we are interested in learning a disjunctive definition of a concept from positive and negative examples. Introducing the disjunction into the hypotheses space is important because many concepts are truly disjunctive ones and their definitions require several rules. A very simple example of such a concept is the concept "parent", in fact composed of two subconcepts "father" and "mother" and defined by two clauses: $parent(X, Y) \leftarrow mother(X, Y)$ and $parent(X, Y) \leftarrow father(X, Y)$, each rule defines a subconcept of the initial

concept. Nevertheless, learning disjunctive concepts leads to several problems. First, there exists a trivial definition of the concept that covers all the positive examples and rejects the negative ones (except when an example is labeled as positive and negative), namely the disjunction of the positive examples. Secondly, the complexity of the search space increases. To deal with these problems, several algorithms have been developed. For instance, in the system INDUCE [HMS83], R.S. Michalski introduced the star method that consists in choosing a positive example e (often called the seed), building the star[1]of e w.r.t. the negative examples, choosing the best definition in the star, and iterating the process until all the positive examples are covered. This leads to a disjunctive definition composed of all the descriptions found so far. A similar strategy is applied in the system Progol [Mug95]: a positive example e is chosen and Progol explores the search space defined by the descriptions that are more general than e. In both cases, the result strongly depends on the choice of the seeds. The system Foil [QCJ95] also searches a disjunctive definition by iteratively building a conjunctive description that covers positive examples and rejects the negative ones. Nevertheless, the strategy used to build a conjunctive definition differs: Foil starts from the most general clause, namely the clause with an empty body and iteratively adds a literal to the body of the clause until all the negative examples are rejected; the choice of the literal depends on a heuristic function that computes for each literal its gain in terms of the number of positive examples still covered when adding this literal to the clause, the number of positive instantiations and the number of negative instantiations still covered. It suffers from several drawbacks. As mentioned in [QCJ95,SBP93], there exists determinate literals that cover all the positive examples and all the negative ones, such literals have thus a very poor gain and are usually not chosen by the system although they can be very important for building a good definition, allowing the introduction of a new variable in the definition. Moreover, the result depends on the heuristic function.

In this paper, we propose another strategy: first using a similarity measure, we build clusters of examples, thus defining subconcepts. Once subconcepts have been found, we learn a conjunctive description of each subconcept.The set of all rules gives us the definition of the concept. An important point in that work is that the similarity measure is based on a language specified by the user, allowing to capture more complex similarities between examples than properties depending on a single literal.

2 Overall presentation

2.1 Motivations

As already mentioned in the introduction, many disjunctive learning algorithms use a greedy method, iteratively adding rules until all positive examples are

[1] the star of an example e w.r.t. to negative examples is the set of all the conjunctive descriptions that cover e and reject all the negative examples

covered. Depending on the systems, the rules can be built with a still greedy approach, starting from a general rule and specializing it by adding at each step the most discriminant literal. Various methods are proposed in the literature to define the **discriminative power** (denoted by $\Gamma(l, C)$) of a literal l for a clause C. Let E^+ (resp. E^-) be the set of positive (resp. negative) examples and let $cov(E, C)$ be the number of examples[2] from E covered by the clause C, $\Gamma(l, C)$ depends at least on $cov(E^+, C \cup \{l\})$ and $cov(E^-, C \cup \{l\})$, where $C \cup \{l\}$ denotes the clause obtained by adding l in the body of C. The definition of $\Gamma(l, C)$ plays a central role in this process, and therefore in the set of subconcepts induced. Moreover, $\Gamma(l, C)$ is highly dependent on the learning sets E^+ and E^-: few changes in these sets may lead to very different solutions, and then to different subconcepts.

At last, if $h \leftarrow p, q$ is a clause which covers some positive examples and no negative ones, $\Gamma(p, h \leftarrow)$ and $\Gamma(q, h \leftarrow)$ are not necessarily high, as shown in Example 1. The consequence is that it may be impossible for a greedy method to learn the clause $h \leftarrow p, q$ because starting from $h \leftarrow$, neither p, nor q seems to be a discriminant literal. This problem is more general than determinate literals, already pointed out in [QCJ95,SBP93].

To sum up, the main drawback of greedy methods is that each subconcept is characterized by a clause $h \leftarrow b_1, \ldots b_n$ where b_i is a highly discriminant literal for $h \leftarrow b_1, \ldots b_{i-1}$. Instead of having subconcepts depending on the discriminating power function, we propose a two-step process: (1) building sets of similar positive examples, thus defining subconcepts in extension, (2) for each subconcept, searching for a unique rule characterizing it, i.e. a rule that covers all (resp. none of) the positive (resp. negative) examples of that subconcept.

- The main advantage of this approach is to strongly reduce the search space when searching for the subconcept definition: hypotheses are considered only if they cover all the positive examples of the subconcept[3]. Practically let G be a subconcept (a set of positive examples) and let us assume that $C = h \leftarrow b_1, \ldots b_i$ is the clause under construction. A literal l can be added to the clause only if $cov(G, C \cup \{l\}) = |G|$. Then, the discriminative power is no longer a combination of 2 criteria ($cov(E^+, C \cup \{l\})$ and $cov(E^-, C \cup \{l\})$) but is mainly characterized by $cov(E^-, C \cup \{l\})$. For example, we can choose to add to C the literal l that minimizes $cov(E^-, C \cup \{l\})$ among the set of literals l_i such that $cov(G, C \cup \{l_i\}) = |G|$.
- When it is not possible to find a clause that defines a subconcept G, then G is split into new subconcepts and definitions for these subconcepts are looked for. This process is repeated until a clause can be learned for each subconcept. In the worst case, this leads to subconcepts containing a single positive example, an overfitting situation which may also occur with greedy

[2] In Inductive Logic Programming, the number of instantiations of C that cover an element of E may also be used to define the discriminating power.

[3] This constraint can be reduced to handle noise: hypotheses are considered only if they cover most of the positive examples of the subconcept.

algorithms, but which has not been encountered in our experiments. If no clause can be found for such a subconcept, this means that there is no complete and consistent solution for the initial learning problem.

2.2 Concept decomposition and clustering

In greedy algorithms the definition of $\Gamma(l, C)$ plays a central role whereas in our approach, the decomposition of concept into subconcepts is essential. The goal of this paper is to propose a clustering method that allows the formation of subconcepts. An important point is then the evaluation of our clustering method, i.e., testing whether the decomposition leads to "good" subconcepts. The quality of the decomposition can be considered from different points of view that are not exclusive:

natural organization: In most cases, a concept can be "naturally" divided into subconcepts: each subconcept is composed of similar examples, which do share some characteristic properties.

simplicity: From a practical point of view, a decomposition is good if it produces a low number of simple (short) rules.

accuracy: The accuracy of the prediction on new examples is good.

We propose a two-steps decomposition method. First, we define a similarity measure and we compute the similarity between each pair of positive examples. Then, we use a clustering algorithm to build possibly overlapping groups of similar objects.

To motivate the approach, let us consider the following example. For sake of simplicity, we consider simple examples which could be considered in a attribute/value formalism. Nevertheless, as shown in Section 5, our approach can be applied on specific ILP problems.

Example 1 Let us consider background knowledge composed of the following ground atoms:

$$\{r(a), r(b), r(f), r(h), s(a), s(b), s(g), s(i), t(c), t(d), t(g), t(i), u(c), u(d), u(f),\\ u(h), v(b), v(c), v(f), r(e), s(e), t(e), u(e), v(e)\}$$

Let us assume that the set of positive examples is $\{p(a), p(b), p(c), p(d)\}$ and the set of negative examples is $\{p(f), p(g), p(h), p(i)\}$.

Considering a greedy algorithm, similar for instance to the Foil algorithm, the construction starts with the clause $p(X) \leftarrow$,

- if we add one of the literal $r(X), s(X), t(X)$ or $u(X)$, we get a clause covering 2 positive examples and 2 negative ones;
- if we add the literal $v(X)$, we get a clause which covers 2 positive examples and 1 negative one.

The literal $v(X)$ is chosen and the final learned program contains at least 3 clauses. However, the concept can be characterized by the two following clauses:

C_1: $p(X) \leftarrow r(X), s(X)$

C_2: $p(X) \leftarrow t(X), u(X)$

which lead to 2 subconcepts : $\{p(a), p(b)\}$ and $\{p(c), p(d)\}$. In order to learn these 2 clauses, our decomposition method should build these 2 subconcepts from the background knowledge, i.e. it should consider that $p(a)$ and $p(b)$ (resp $p(c)$ and $p(d)$) are highly similar.

We propose to use the similarity measure defined in [MM01]: it is based on the number of properties shared by two objects, from a specified set of properties called the *language*. To introduce this measure on the example, let us consider a language $\mathcal{L}$ containing all the clauses with head $p(X)$ and with a single literal in the body from $\{r(X), s(X), t(X), u(X), v(X)\}$. The similarity between two examples e_i and e_j is then defined by the number of clauses C in $\mathcal{L}$ such that either e_i and e_j are covered by C or e_i and e_j are not covered by C. For example, $p(a)$ and $p(b)$ are both covered by $p(X) \leftarrow r(X)$ and by $p(X) \leftarrow s(X)$; they are both not covered by $p(X) \leftarrow t(X)$ and by $p(X) \leftarrow u(X)$. In this case, the similarity between $p(a)$ and $p(b)$ is equal to 4. In the same way, we can compute the similarity matrix between each pair of positive examples, and we get the following matrix:

	$p(a)$	$p(b)$	$p(c)$	$p(d)$
$p(a)$	5	4	0	1
$p(b)$	4	5	1	0
$p(c)$	0	1	5	4
$p(d)$	1	0	4	5

In this matrix, the 2 expected subconcepts clearly appear. However, in many problems, subconcepts are not disjoint. For instance, let us now add to the previous example the new positive example $p(e)$. With this new example, the similarity becomes:

	$p(a)$	$p(b)$	$p(e)$	$p(c)$	$p(d)$
$p(a)$	5	4	2	0	1
$p(b)$	4	5	3	1	0
$p(e)$	2	3	5	3	2
$p(c)$	0	1	3	5	4
$p(d)$	1	0	2	4	5

From this matrix, the 2 initial subconcepts $\{p(a), p(b)\}$ and $\{p(c), p(d)\}$ still appear, but $p(e)$ can be inserted into both subconcepts. In fact, the new example is covered by the clauses C_1 and C_2 and then the subconcepts induced by these two clauses are $\{p(a), p(b), p(e)\}$ and $\{p(e), p(c), p(d)\}$.

In this paper, we propose a clustering method able to build non disjoint clusters. When applied to this example, it leads to the building of the two clusters $\{p(a), p(b), p(e)\}$ and $\{p(e), p(c), p(d)\}$.

Example 2 Let us consider now an example where the observations are described with numeric properties.

Let us consider a set of objects $\{a, b, \ldots u\}$ described by two attributes x and y. These objects are described in Figure 1 and are expressed in a background knowledge BK defining the predicates x ($x(U, V)$ is true when the object U satisfies $x = V$), y, $\geq$ and $\leq$. For instance, BK contains the ground atoms $x(a, 1)$, $y(a, 1)$, $x(b, 1)$, $y(b, 2)$, ..., $\geq (1, 1)$, $\geq (1, 2)$, ... Finally, the set of positive examples is $\{p(a), p(b), \ldots, p(j)\}$ and the set of negative examples is $\{p(l), p(m), \ldots, p(u)\}$

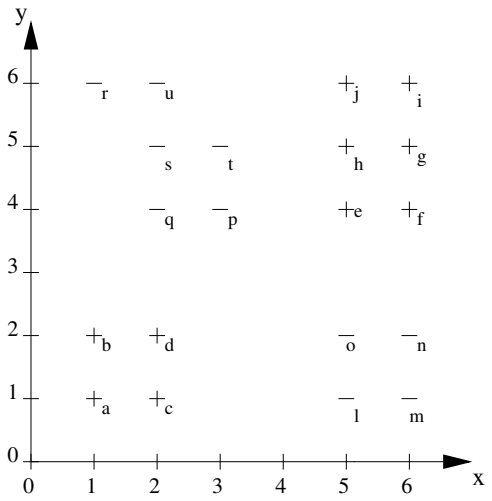

Fig. 1. Examples with numerical values

If we allow the constants 1, ..., 6 to appear in the learned clauses, it will be hard for a greedy algorithm to learn a definition of the target concept. Indeed, starting from $p(X) \leftarrow$, the literals $x(X, Y)$ and $y(X, Y)$ are determinate literals. Starting from the clause $p(X) \leftarrow x(X, Y)$ or from the clause $p(X) \leftarrow y(X, Y)$, the literals $\geq (Y, 1)$, $\geq (Y, 2)$, ..., $\geq (Y, 6)$ are not discriminant. In our framework, we can compute the similarity with respect to the language $\bigcup_{v=1..6} p(X) \leftarrow x(X, Y), \geq (Y, v) \cup \bigcup_{v=1..6} p(X) \leftarrow y(X, Y), \geq (Y, v)$. This language leads to a similarity measure between examples which is close to the Euclidean similarity, and which induces 2 clusters, corresponding to the subconcepts $\{p(a), p(b), \ldots, p(d)\}$ (characterized by the clause $p(X) \leftarrow x(X, Y), \leq (Y, 2), y(X, Z), \leq (Z, 2)$) and $\{p(e), p(f), \ldots, p(j)\}$ (characterized by $p(X) \leftarrow x(X, Y), \geq (Y, 4), y(X, Z), \geq (Z, 3)$).

Let us notice that on both Example 1 and Example 2, the clauses that characterize the obtained concepts do not belong to the language used to define the similarity. Experiments on this example are detailed in Section 5.

2.3 Similarity measure

Different approaches have been proposed to define a similarity measure between objects described in a first order logic formalism. The common way to build a similarity function consists in first producing a description of objects. Then, for descriptions based on sets of atoms, the similarity is defined by intersections [Bis92,EW96]; for descriptions based on rules, the similarity between two objects is given by the number of rules that are satisfied by the two objects [Seb97,SS94,MM01]. In any case, two objects are considered to be similar when they share some properties. We use the similarity measure proposed in [MM01], this measure is defined with respect to a finite set of clauses, called the language; for a clause C and an example e, we define the function $covered(C, e)$ as follows: $covered(C, e) = 1$ if e is covered by C, otherwise $covered(C, e) = 0$. Given a language $\mathcal{L}$, we define $\mathcal{L}(e_i, e_j)$ as the set of clauses C such that either e_i and e_j are covered by C or e_i and e_j are not covered by C:

$$\mathcal{L}(e_i, e_j) = \{C \in \mathcal{L} \text{ such that } covered(C, e_i) = covered(C, e_j)\}$$

The similarity between e_i and e_j, written $sim_{\mathcal{L}}(e_i, e_j)$, is then defined by:

$$sim_{\mathcal{L}}(e_i, e_j) = \frac{|\mathcal{L}(e_i, e_j)|}{|\mathcal{L}|}$$

This is the similarity measure we have used in Example 1. However, it does not take into account the set of negative examples. We propose to extend this measure by giving a weight on the clauses of the language, which depends on the set of positive and negative examples. More precisely, if E^+ and E^- are respectively the sets of positive and negative examples, the weight of a clause C is defined by $w(C) = cov(E^+, C)/(cov(E^+, C) + cov(E^-, C))$ if $cov(E^+, C) > 0$, otherwise $w(C) = 0$. The maximum value of this weight is equal to 1 for clauses which cover only positive examples. The weighted similarity we obtain is then:

$$w\_sim(e_i, e_j) = \frac{\sum_{C \in \mathcal{L}(e_i, e_j)} w(C)}{|\mathcal{L}|}$$

it gives a higher weight for clauses that are closer to the target concept. This measure may have a drawback when the language contains a large number of clauses with low weights, since it may reduce the influence of clauses with high weight. For this reason, we propose a definition of similarity, depending on a threshold α, such that clauses with a weight lower than α are not considered:

$$w\_sim_{\alpha}(e_i, e_j) = \frac{\sum_{C \in \mathcal{L}(e_i, e_j), w(c) > \alpha} w(C)}{|\mathcal{L}|}$$

The most general clause, which covers all positive and negative examples, has a weight equal to $|E^+|/(|E^+| + |E^-|)$. All the clauses that have a weight lower than this value have a lower accuracy than the most general clause. For this reason, practically we use in our experiments w_sim_{α} with the threshold $\alpha = |E^+|/(|E^+| + |E^-|)$.

3 The soft-clustering algorithm

3.1 Overview of the soft-clustering algorithm

The clustering process is the task that consists in organizing a set of objects into classes, so that similar objects belong to the same cluster and dissimilar ones belong to different clusters. Traditional clustering algorithms can be classified into two main categories:

- hierarchical methods that build a hierarchy of classes and illustrated by well known algorithms such as SAHN[4], COBWEB or by newer ones like CURE and CHAMELEON. Let us notice that in hierarchical methods, each level of the hierarchy forms a partition of the set of observed events.
- partitioning methods, such as BIRCH, K-MEANS or K-MEDOIDS algorithms that build a partition of the set of observed events, i.e., a training observation cannot belong to two distinct clusters.

An extensive survey about clustering methods is given in [Ber02].

The clustering algorithms are usually assessed regarding to the following characteristics: time efficiency, outliers processing, diversity of the clusters shapes, ability to deal with several data types and quality of the final clusters. However a common disadvantage of classical methods is that they require the clusters to be disjoint. In the application of clustering we are interested in - learning a disjunctive definition of a concept by first clustering its observations - learning disjoint clusters truly induces a loss of information. Indeed the main idea of the method we present in this paper is based on the hypothesis that most of the concepts that specify a data set are not clearly separated, and that some objects can be assigned to several clusters. The *soft-clustering* approach is a kind of compromise between *hard-clustering* methods, cited before and *fuzzy-clustering* methods which use a fuzzy membership function which gives to an element a membership value for each class [BB99]. The fuzzy-clustering approach is well known for the wealth of the information description and allows some of the observations to belong to several clusters. In our approach, we first determine a set of *poles* that represent strong concepts present in the data and are built from non-shared objects. In a second step, a function is defined for assigning objects that are not yet covered in one or several clusters.

3.2 The notion of "Pole"

The notion of *Pole* we define here in our algorithm can be linked to the recent definition of a *core* given by Ben-Dor and Yakhini in [BDSY99]. Their approach is based on the idea that "ideally, all elements within a cluster should be similar to each other, and not similar to any of others clusters". They identify small cores in a similarity graph which allows to approximate an ideal *clique graph*[5].

[4] Sequential Agglomerative Hierarchical Non-overlapping

[5] A clique is a fully connected subset of vertices.

These cores are small disjoint subsets of the vertices of the input graph. In our opinion, a *clique graph* does not represent classical real world situations: indeed, a clique has good properties to express the interactions between elements of a cluster, nevertheless, relations can appear between different clusters. Thus, the elements that form a concept[6] can be split into two categories: the elements that belong only to that concept (the non-shared objects) and the elements also linked to other concepts (the shared objects). In this way, a *pole* can be seen as the non-shared part of the concept.

Before giving a complete definition of a pole, we first define the notion of similarity graph and we describe the method for building a pole.

Definition 1. *Let V be a set of elements, S be a similarity matrix ($S : V \times V \rightarrow \mathbb{R}$) and $\alpha \in \mathbb{R}$ be a threshold. The similarity graph G derived from V (denoted by $G(V, E_\alpha)$) is an undirected, valued graph such that the set of vertices of G is equal to V and the set, E_α, of undirected edges is defined by:*

$$(v_i, v_j) \in E_\alpha \text{ and is valued by } s \; iff \; S(v_i, v_j) \geq \alpha \text{ and } S(v_i, v_j) = s$$

The threshold α is decisive for the whole process. The higher α is, the less $G(V, E_\alpha)$ contains edges. Therefore the number of final clusters is linked to the choice of this threshold. Rather than giving an arbitrary value for α, we notice that several natural thresholds can be used. For instance :

$$\alpha = \begin{cases} \underline{\textit{independent from the considered vertices}} : \\ \\ (\alpha_1) \quad \alpha = 0 \\ (\alpha_2) \quad \alpha = \frac{1}{|V|.|(V-1)|} \sum_{v_i \in V} \sum_{v_j \in V \setminus \{v_i\}} S(v_i, v_j) \text{ (average value over } S) \\ (\alpha_3) \quad \alpha \text{ such as } \mid \{(v_i, v_j) \mid S(v_i, v_j) > \alpha\} \mid = \mid \{(v_i, v_j) \mid S(v_i, v_j) < \alpha\} \mid \\ \qquad = \frac{1}{2} \mid V \mid . \mid (V-1) \mid \text{ the median value over } S \\ \\ \underline{\textit{dependent from the considered vertices}} : \\ \\ (\alpha_4) \quad \alpha(i,j) = Max(\frac{1}{|V|} \sum_{v_k \in V} S(v_i, v_k); \frac{1}{|V|} \sum_{v_k \in V} S(v_j, v_k)) \\ (\alpha_5) \quad \alpha(i,j) = Max(S(v_i, v_{i,k}); S(v_j, v_{j,k})) \text{where } v_{i,k} \text{ is the } k^{th} \text{ nearest} \\ \qquad \text{neighbor of } v_i \text{ (for a given } k). \end{cases}$$

The threshold α_1 suppose that the similarity matrix have negative values, thus two elements are considered to be in relation if their similarity is positive (α_1), or if their similarity is greater than the average similarity (α_2). In the case of α_3, since the threshold splits the space of potential edges into two equal parts, the similarity graph contains exactly 50% of the edges. In contrast with the three first thresholds, α_4 and α_5 take into account the situation of the elements compared with all others in order to define a possible relation between two

[6] The term of concept denotes the intuitive notion of cluster.

elements. For instance, $\alpha_4(v_i, v_j)$[7] allows to consider that v_i is in relation with v_j if v_i (resp. v_j) is near from v_j (resp. v_i) on average than from the other elements in the space. The threshold $\alpha_5(v_i, v_j)$ determines the existence of a relation between two elements if each element is among the k nearest neigbours of the other.

With the previous definition of a similarity graph, the similarity matrix over $\{p(a), p(b), p(e), p(c), p(d)\}$ given in Example 1 (section 2.1) leads to the similarity graph induced by the following adjacency matrix:

	$p(a)$	$p(b)$	$p(e)$	$p(c)$	$p(d)$
$p(a)$	5	4	2	0	1
$p(b)$	4	5	3	1	0
$p(e)$	2	3	5	3	2
$p(c)$	0	1	3	5	4
$p(d)$	1	0	2	4	5

similarity matrix

	$p(a)$	$p(b)$	$p(e)$	$p(c)$	$p(d)$
$p(a)$	-	1	1	0	0
$p(b)$	1	-	1	0	0
$p(e)$	1	1	-	1	1
$p(c)$	0	0	1	-	1
$p(d)$	0	0	1	1	-

adjacency matrix

This adjacency matrix is obtained using α_1,α_2 or α_4; two cliques clearly appear: $\{p(a), p(b), p(e)\}$ and $\{p(e), p(c), p(d)\}$. Because of the small size of this example, the threshold α_3 does not exist. The similarity graph obtained with α_5 for (k=2) differs from the previous one for two edges ($(p(a), p(e))$ and $(p(d), p(e))$ are not connected) thus four small cliques appear: $\{p(a), p(b)\}$, $\{p(b), p(e)\}$, $\{p(e), p(c)\}$, and $\{p(c), p(d)\}$.

The construction of a pole from the similarity graph consists in searching a maximal-clique in the graph centered on a given point. Because of the computational complexity of the maximum clique problem[8], efficient heuristics for approximating such a clique have been developed. For a more detailed introduction to heuristics that approximate a maximal clique, see for instance [BBPP99]. In our method we use a *sequential greedy heuristic* based on the principle *"Best in"*: it constructs a maximal clique by repeatedly adding a vertex that is the nearest one among the neighbours of the temporary clique. The algorithm that implements that heuristic is given in Table 1. Once all possible cliques are built, final poles are reduced to their non-shared part.

Definition 2. *Let G be an undirected, valued graph, and let C be a set of vertices of G. A* neighbor *of C in G is a vertex v of G such that $\forall v_j \in C, (v, v_j) \in E_\alpha$. Let $\mathcal{V}_C$ denotes the set of neighbors of C in G. The* nearest neighbor *of C in G is a neighbor v of C such that $v = Argmax_{v_i \in \mathcal{V}_C} \ \frac{1}{|C|} \sum_{v_j \in C} S(v_i, v_j)$, i.e; it is the neighbor of C that is the nearest in average of all the elements of C.*

[7] α_4 is the threshold retained for our experiments.

[8] The maximum-clique search is a NP-complete problem.

Table 1. The *Best in* heuristic for the construction of one clique

Inputs:	$G(V, E_\alpha)$ a valued graph, v_s a starting vertex
	$C \leftarrow \{v_s\}$
	Build $\mathcal{V}$, the set of neighbors of C in G
While:	$\mathcal{V}$ is not empty
	Select v_n the nearest vertex from C in $\mathcal{V}$
	where $S(v_i, v_j)$ is the weight of the edge (v_i, v_j) in E_α
	$C \leftarrow \{v_n\}$
	Build $\mathcal{V}$ the set of neighbors of C in G
Output:	The clique C

3.3 The relative function for assignment

Let us now suppose that we have iteratively built a set of disjoint cliques in the graph G. Each clique thus represents a class and we have now to assign the remaining vertices of G (that do not belong to a clique) to one or several classes. We define a boolean function that determines whether an element must be assigned to a class or not. It takes into account the relative proximity between an element and a class, defined as the average similarity between that element and each element belonging to the class. The backbone of this task is that when assigning an element to a concept we take into account the other concepts. More formally, this function is defined as follows:

Definition 3. *Let V be a set of elements, $V' = C_1 \cup C_2 \cup \cdots \cup C_m$ a subset of V with $C_i \cap C_j = \emptyset$ $\forall i, j \in \{1, \ldots m\}$, S a similarity matrix $(S : V \times V \rightarrow \mathbb{R})$. The k^{th} nearest concept from an element $v_i \in V$ is written ${C_i}^k$. The relative assignment function of an element v_i to a class ${C_i}^k$ is defined by :*

$$\forall k \ : \ f(v_i, {C_i}^k) = \begin{cases} 1 \ \ if \ \ \bullet \ \forall l < k \ f(v_i, {C_i}^l) = 1 \\ \qquad\quad \bullet \ S(v_i, {C_i}^k) > 0 \\ \qquad\quad \bullet \ S(v_i, {C_i}^k) > \frac{1}{2}(S(v_i, {C_i}^{k-1}) + S(v_i, {C_i}^{k+1})) \\ \qquad\quad for \ 1 < k < m \\ \qquad\quad \bullet \ S(v_i, {C_i}^k) > \frac{1}{2}(S(v_i, {C_i}^{k-1})) \\ \qquad\quad for \ k = m \\ \\ 0 \ \ otherwise \end{cases}$$

We note that for the nearest concept $({C_i}^1)$ we have $f(v_i, {C_i}^1)$ iff $S(v_i, {C_i}^1) > 0$.

Let us now give an important property of the assignment function f:

$$(P)\ \forall v_i \in V\ :\ 0 \leq \sum_{k=1}^{m} f(v_i, {C_i}^k) \leq m$$

This property shows that when using f it is possible that an element (for instance an outlier) is not assigned to any class; contrary to an outlier, a very central element could be assigned to all the classes. The relative function is thus different to probabilistic membership, as for instance [KK93], which characterizes an element according to its distribution over the classes (or poles).

3.4 Description of the algorithm

In this section, we propose a formal description of the soft-clustering algorithm. It relies on the definitions of *pole* and *assignment function* given in the previous sections. The clustering process (Table 2) iterates the step of pole construction until no more starting point is available. Then the remaining elements are assigned to one or several poles, using the assignment function.

Table 2. The Soft-Clustering Algorithm

Input:	V the set of elements, S the similarity matrix over V
Initialization:	$\mathcal{C} = \emptyset,\ \mathcal{P} = \emptyset$
	// $\mathcal{P}$: the set of poles, $\mathcal{C}$: the set of vertices appearing in poles
Step 1:	Construction of the similarity graph $G(V, E_\alpha)$
Step 2:	$v_s = \mathit{startpoint}(\mathcal{C}, S, G)$
Step 3:	Build a pole P centered on v_s
	$\mathcal{P} \leftarrow \mathcal{P} \cup \{P\},\ \mathcal{C} = \mathcal{C} \cup P$
Step 4:	$v_s = \mathit{startpoint}(\mathcal{C}, S, G)$
	if a startpoint v_s is found, GOTO step 3
Step 5:	Each Pole P is reduced to its non-shared objects $\tilde{P} \subset P$
Step 6:	For each element $v_i \in V \backslash \mathcal{C}$ assign v_i to poles of $\mathcal{P}$ using f
	where f is the relative assignment function
Output:	A set of overlapping clusters $\tilde{P}'_1, \tilde{P}'_2, \ldots$
	where $\tilde{P}'_i = \tilde{P}_i \cup \{v_j \in V \backslash \mathcal{C} \mid f(v_j, \tilde{P}_i) = 1\}$

The *startpoint* function provides the element among $V \backslash \mathcal{C}$ which is "the most distant" from the set $\mathcal{C}$. The result differs depending whether $\mathcal{C}$ is empty or not. Several definitions of the function *startpoint* can be given. In our experiments, we use the definition $\mathit{startpoint}_1$ that corresponds to the intuitive idea of "the most distant":

$$\mathit{startpoint}_1(\mathcal{C}, S, G) = \begin{cases} \mathit{when}\ \mathcal{C} = \emptyset \quad \mathit{Argmin}_{(v_i \in V)}\ \mathit{degree}(v_i, G) \\ \qquad\qquad\qquad \mathit{with}\ (\mathit{degree}(v_i, G) \geq 1) \\ \\ \mathit{when}\ \mathcal{C} \neq \emptyset \quad \mathit{Argmin}_{(v_i \in V \backslash \mathcal{C})}\ S(v_i, \mathcal{C}) \end{cases}$$

where $degree(v, G)$ represents the number of edges of v in G.
Another definition could be:

$$startpoint_2(\mathcal{C}, S, G) = \begin{cases} when\ \mathcal{C} = \emptyset \\ \quad Argmin_{(v_i \in V)}\ \frac{1}{K+1}(degree(v_i, G) + \sum_{k=1}^{k=K} degree(v_{i,k}, G)) \\ when\ \mathcal{C} \neq \emptyset \\ \quad Argmin_{(v_i \in V \setminus \mathcal{C})}\ \frac{1}{K+1}(S(V_i, k) + \sum_{k=1}^{k=K} S(v_{i,k}, \mathcal{C})) \end{cases}$$

where $v_{i,k}$ denotes the k^{th} nearest neighbor of v_i and K a given constant.

In Example 1, the four elements $\{p(a), p(b), p(c), p(d)\}$ have a minimal degree equal to 2. If $p(a)$ is randomly chosen as the starting point, the first clique built is $P_1 = \{p(a), p(b), p(e)\}$. Then the most distant element from P_1 among $\{p(c), p(d)\}$ is $p(d)$ and the clique obtained from this vertex is $P_2 = \{p(e), p(c), p(d)\}$. Because $p(e)$ is shared by P_1 and P_2, the restriction of the cliques to their non-shared objects provides the two poles $\tilde{P}_1 = \{p(a), p(b)\}$ and $\tilde{P}_2 = \{p(c), p(d)\}$. Finally, the assignment step allows $p(e)$ to be member of $\tilde{P}_1$ and $\tilde{P}_2$. The final clusters are thus the two non disjoint ones covered by the two rules:

C_1 : $p(X) \leftarrow r(X), s(X)$
C_2 : $p(X) \leftarrow t(X), u(X)$

The soft-clustering algorithm we propose has several properties highly interesting for the application to learning disjunctive concepts:
(1) the clusters that are built can overlap,
(2) the number of final clusters is not decided *a priori*,
(3) the input of the algorithm is a similarity matrix, thus allowing the method to be applied to very different kinds of data.

4 General presentation of the method

The general learning algorithm is presented in Figure 2. Inputs of the method are: the target concept, specified by positive and negative examples, a background knowledge and a language associated to the similarity measure. We assume that the target concept cannot be characterized by a single clause (otherwise, the algorithm simply outputs this clause).

The first step of the method is the computation of the similarity between each pair of positive examples. Then, the similarity matrix is used by the clustering algorithm to produce a set of possibly non-disjoint groups. In some cases, this algorithm produces a large number of groups or some groups are highly similar. For this reason, we organize these groups into a hierarchical way: we use an average-link agglomerative algorithm producing a tree where the leaves are the groups obtained by the clustering method and the root of this tree represents the entire initial concept. In this tree, each node (group of examples) is either a leaf or has two direct sub-groups.

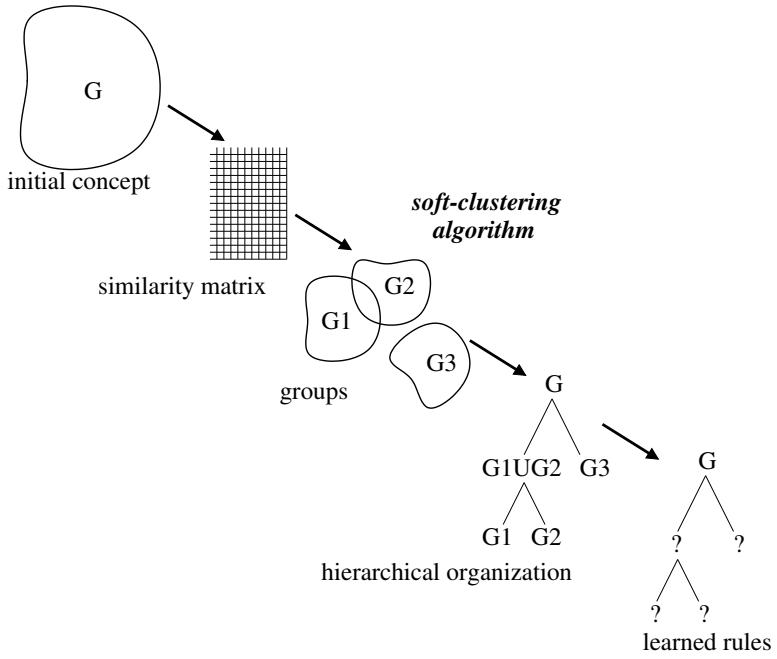

Fig. 2. Overview of the learning algorithm

Then the method tries to build a clause which characterizes groups (nodes) of the tree, starting with the two sub-groups of the root group. If a clause is found for a group, this clause is added to the learned program; if no clause is found for a group, this process is recursively repeated on its direct sub-groups (if the associated node is not a leaf).

Finally, if some groups have not been characterized by a clause, the learning method (Figure 2) is recursively applied on these groups.

4.1 Learning a clause

As mentioned above, the decomposition method induces a strong reduction on the search space. Given a set of positive examples G and the set of negative examples E^-, we try to build a clause which covers all the examples in G and no negative ones. The learning method we use is a greedy search, starting from the clause $p(X_1, \ldots, X_n) \leftarrow$ (where p is the n-ary target predicate) and adding literals one by one until each negative example is rejected by the clause. Since the clause has to cover each example in G, we choose the literal which allows to reject as many negative examples as possible, among the set of literals which cover all the examples in G (no backtrack is needed).

4.2 Complexity

The time complexity of the method comes from the three main steps of the algorithm:

similarity: To compute the similarity matrix, we have to test whether each example is covered or not by each clause of the language $\mathcal{L}$. It requires $(|E^+| + |E^-|) * |\mathcal{L}|$ covering tests; then the similarity between each pair of positive examples is computed: $|E^+|^2 * |\mathcal{L}|$ operations,

clustering: The time required for this step is mainly due to the pole construction: for each pole, it needs at most $|E^+|^2$ operations,

building clauses: The greedy search is reduced by the restriction of the search space.

The space complexity is determined by the number of positive examples since we have to compute a $|E^+| \times |E^+|$ similarity matrix. If $|E^+|$ is too high (more than 1000), we can use a sample of this set: the number of positive examples needed depends mainly on the number of learned clauses. For example, assume that the target concept can be characterized with 10 clauses, with 1000 positive examples we still have an average of 100 examples per groups.

5 Experiments

We propose here some preliminary experiments of our approach: we have tested the method on examples for which a "natural" decomposition into subconcepts is known, and we compare the results obtained with our clustering method w.r.t expected subconcepts.

Example 1

The first experiment concerns Example 1: if we use the similarity w_sim_α with respect to the language $\mathcal{L}$ proposed in Example 1, we get similarity matrices closed to those presented in the example. With the first matrix we get the 2 expected subconcepts $\{p(a), p(b)\}$ and $\{p(c), p(d)\}$; the second matrix gives also the 2 expected subconcepts $\{p(a), p(b), p(e)\}$ and $\{p(e), p(c), p(d)\}$ ($p(e)$ belongs to both subconcepts).

Example 2

The second experiment concerns Example 2. If we consider the similarity associated to the language $\mathcal{L}_1 = \bigcup_{v=1..6}(p(X) \leftarrow x(X,Y), \geq (Y,v)) \cup \bigcup_{v=1..6}(p(X) \leftarrow y(X,Y), \geq (Y,v))$, we obtain the expected subconcepts : $\{p(a), p(b), \ldots, p(d)\}$ and $\{p(e), p(f), \ldots, p(j)\}$. If we consider now the similarity associated to the language $\mathcal{L}_2$ containing all the clauses having $p(X)$ as head and having at most two literals in the body, the clustering algorithm gives a different result: the produced groups are

$G_1 = \{p(a), p(b), p(c), p(d)\}$
$G_2 = \{p(e), p(b), p(g), p(j)\}$
$G_3 = \{p(g), p(h), p(i), p(j)\}$

The explanation is based on the difference between $\mathcal{L}_1$ and $\mathcal{L}_2$: the clause $p(X) \leftarrow x(X,Y), y(X,Y)$ belongs to $\mathcal{L}_2$ but not to $\mathcal{L}_1$. This definition is correct since it covers 4 positive examples : $\{p(a), p(d), p(h), p(i)\}$ and no negative ones,

and then it has a weight equal to 1. The consequence is that the similarity between $p(h)$ and $p(i)$ is increased and for instance, the similarity between $p(e)$ and $p(h)$ is reduced. This example shows that when different decompositions exists, this may induce a fragmentation of some subconcepts (particularly when the number of examples is low). For this reason, the result of our soft-clustering algorithm is transformed into a hierarchical soft-clustering one before learning a definition. In this example, G_2 and G_3 are the most similar groups, then the hierarchical clustering result is

E^+ is divided into G_1 and $G_{2,3}$,
$G_{2,3}$ is divided into G_2 and G_3

where the first level of decomposition leads to the expected result.

Example 3: ancestor

The following example was proposed in [dRL93]. The background knowledge contains ground atoms with predicates $father$, $mother$, $male$ and $female$ over a 19 persons family. The target concept is $ancestor$. To compute the similarity between examples, we consider the language containing all the clauses having $ancestor(X, Y)$ as head and having at most two literals in the body. On the complete set of 56 positive examples, our clustering algorithm produces 3 disjoint groups.

G_1 **:** the first group is the set $\{ancestor(X_i, Y_i) \mid X_i \text{ is the father of } Y_i\}$
G_2 **:** the second group is the set $\{ancestor(X_i, Y_i) \mid X_i \text{ is the mother of } Y_i\}$
G_3 **:** the third group contains all other examples.

The most similar groups are G_2 and G_3 but no rule can be found to characterize $G_2 \cup G_3$. The algorithm tries to build a characterization for G_1, G_2 and G_3. A rule is learned for G_1 and for G_2 but not for G_3. Then the clustering algorithm is applied on the group G_3 which produces two disjoint groups associated to the recursive definitions $ancestor(X, Y) \leftarrow father(X, Z)ancestor(Z, Y)$ and $ancestor(X, Y) \leftarrow mother(X, Z), ancestor(Z, Y)$. Finally, the decomposition produces 4 groups, corresponding to the usual definition of ancestor.

Example 4

The last example is introduced to test the ability to build non-disjoint groups. Consider a graph containing two types of edges r and s. BK contains an atom $r(X, Y)$ (resp. $s(X, Y)$) if an edge of type r (resp. s) exists from X to Y. BK contains the following set of ground atoms:

$$\{r(b,f), r(b,g), r(f,l), r(g,m), r(k,r), r(l,o), r(m,p), r(m,o),$$
$$r(j,n), r(a,f), r(e,k), s(a,f), s(e,k), s(j,n), s(c,g), s(c,h),$$
$$s(c,i), s(d,i), s(d,j), s(h,l), s(h,m), s(h,n), s(i,n), s(n,q),\}$$

The target concept is $linked$, specified by the set of atoms $linked(X, Y)$ such that there exists a path from X to Y. To compute the similarity between examples, we also consider the language containing all the clauses having $linked(X, Y)$ as head and having at most two literals in the body. In this example, there exists

some vertices linked by different paths. From 44 positive examples, the clustering algorithm produces 5 groups G_1, G_2, G_3, G_4 and G_5.

- G_1 and G_2 have 3 common examples: G_1 is defined by $linked(X,Y) \leftarrow r(X,Y)$ and G_2 is defined by $linked(X,Y) \leftarrow s(X,Y)$. There are exactly 3 examples $linked(X,Y)$ such that both $r(X,Y)$ and $s(X,Y)$ hold. G_1 and G_2 have no common objects with other groups.
- G_3 has no common examples with other groups. It is defined by the clause $linked(X,Y) \leftarrow r(X,Z), linked(Z,Y)$ and G_3 contains all the examples covered by this clause, except for some examples which are also covered by the clause $linked(X,Y) \leftarrow s(X,Z), linked(Z,Y)$.
- G_4 and G_5 have one common example and these groups are the most similar. $G_4 \cup G_5$ is characterized by the clause $linked(X,Y) \leftarrow s(X,Z), linked(Z,Y)$.

In this example, intersections between groups are not as large as we could have expected. This is mainly due to specificities which increase the similarity between some examples and reduce the possibility for some examples to be assigned to several poles, since the assignment function is base on relative similarities. However, it is preferable to obtain "incomplete" groups since for each group, we try to learn a unique clause. Then, the decomposition obtained on this example is good and allows to produce a satisfying program.

6 Conclusion and further works

We have presented a clustering method used to split the set of positive examples of a target concept into groups of similar examples. Each group is supposed to correspond to a subconcept and we try to learn one clause for each subconcept. This method induces a strong reduction of the search space during the learning process, however it can be applied only if the clustering algorithm produces "good" groups.

To analyze the quality of the obtained groups, we have performed experiments on several examples for which a good decomposition was known. For all these examples, the results obtained with the clustering algorithm correspond to the subconcepts induced by the expected definitions. Moreover, these experiments show that the soft-clustering method is able to produce non-disjoint groups, corresponding to overlapping subconcepts.

In future works, we have to test whether this method produces better clauses than usual greedy algorithms. We plan to test the method on real examples such as Mutagenesis: to achieve this, we have to specify a language (associated to the similarity measure) such that numerical and symbolic values have an fair influence.

We plan also to study the relationship between the language and the similarity induced. In our experiments, we have considered languages made of simple rules (having at most two literals in the body and non recursive), this language is only used for the similarity measure and the search space for rules may be much more complex, it may be infinite and contain recursive definitions.

References

[BB99] A. Baraldi and P. Blonda. A survey of fuzzy clustering algorithms for pattern recognition. II. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 29:786–801, 1999.

[BBPP99] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, Boston, MA, 1999.

[BDSY99] Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.

[Ber02] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[Bis92] G. Bisson. Learning in FOL with a similarity measure. In *11th National Conf. on Artificial Intelligence (AAAI), San Jose, CA.*, pages 82–87. AAAI Press, 1992.

[dRL93] Dzeroski S. de Raedt L., Lavrac N. Multiple predicate learning. In *Proceedings of the Thirteen International Joint Conference on Artificial Intelligence, Chambéry, France*, pages pp. 1037–1043. Springer-Verlag, 1993.

[EW96] W. Emde and D. Wettschereck. Relational instance-based learning. In Saitta L., editor, *13th Int. Conf. on Machine Learning (ICML'96), Bari, Italy*, pages 122–130. Morgan & Kaufmann, 1996.

[HMS83] William A. Hoff, Ryszard S. Michalski, and Robert E. Stepp. INDUCE 2: A program for learning structural descriptions from examples. Technical Report 904, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1983.

[KK93] R. Krishnapuram and J. Keller. A possibilistic approach to clustering. *IEEE Transactions on Fuzzy Systems, Vol. 1, No. 2*, pages 98–110, 1993.

[MM01] Lionel Martin and Frédéric Moal. A language-based similarity measure. In *Machine Learning: ECML 2001, 12th European Conference on Machine Learning, Freiburg, Germany, September 5-7, 2001, Proceedings*, volume 2167 of *Lecture Notes in Artificial Intelligence*, pages 336–347. Springer, 2001.

[Mug95] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[QCJ95] J. R. Quinlan and R. M. Cameron-Jones. Induction of logic programs: FOIL and related systems. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):287–312, 1995.

[SBP93] Giovanni Semeraro, Clifford A. Brunk, and Michael J. Pazzani. Traps and pitfalls when learning logical theories: A case study with FOIL and FOCL. Technical Report ICS-TR-93-33, July 1993.

[Seb97] M. Sebag. Distance induction in first order logic. In *Proceedings of ILP'97*, pages 264–272. Springer-Verlag, 1997.

[SS94] M. Sebag and M. Schoenauer. *Topics in Case-Based Reasonning*, volume 837 of *LNAI*, chapter A Rule-based Similarity Measure, pages 119–130. Springer-Verlag, 1994.