ELSEVIER

Data & Knowledge Engineering 25 (1998) 99–123

DATA & KNOWLEDGE ENGINEERING

Machine learning from examples: Inductive and Lazy methods

Ramon Lopez de Mantaras*, Eva Armengol

Artificial Intelligence Research Institute, CSIC, Campus UAB, 08193 Bellaterra, Spain

Received 14 November 1997; accepted 14 November 1997

Abstract

Machine Learning from examples may be used, within Artificial Intelligence, as a way to acquire general knowledge or associate to a concrete problem solving system. *Inductive learning methods* are typically used to acquire general knowledge from examples. *Lazy methods* are those in which the experience is accessed, selected and used in a problem-centered way. In this paper we report important approaches to inductive learning methods such as propositional and relational learners, with an emphasis in Inductive Logic Programming based methods, as well as to lazy methods such as instance-based and case-based reasoning. ©1998 Elsevier Science B.V.

Keywords: Machine learning; Inductive learning; Inductive logic programming; Lazy learning; Instance-based learning; Case-based reasoning

1. Inductive learning

Inductive methods can be applied in two ways: as interactive tools acquiring knowledge from examples or as a part of learning systems. When inductive methods are used to acquire knowledge, the user provides examples and strongly controls the use of the method. Used as part of learning systems, inductive methods are activated when another system component has the necessity to learn from positive and/or negative examples that constitute the feedback from which the system can achieve the current task. An example of a system having inductive learning integrated with problem solving is LEX [85].

Induction is based on specific facts (examples) instead of general axioms as in deduction. The goal of the induction is to formulate plausible general assertions that both explain the given facts and are capable of predicting unseen facts. In other words, the inductive inference tries to obtain a complete and correct description of a given phenomenon from specific (maybe partial) observations of it. The description inductively obtained is true at least for already seen examples but nothing can be assured for unseen examples.

* Corresponding author.

PII S0169-023X(97)00053-0

The most frequent application of inductive learning is *concept learning*. The goal of concept learning is to find symbolic descriptions expressed in high-level terms that are understandable by people. Concept learning can be defined as follows:

Given: A set of (positive and negative) examples of a concept and, possibly, some background knowledge.

Find: A general description (hypothesis) of the concept describing all the positive examples and none of the negative examples.

Background knowledge defines assumptions and constraints imposed on examples and generated descriptions, and any relevant domain knowledge. Background knowledge can be in different forms [52], e.g. in declarative form or in procedural form, as sequences of instructions for executing specific tasks (control knowledge).

Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation language. The goal of this search is to find the hypothesis (description) that best explains the examples. The language used is very important since it defines the hypothesis space, i.e. what knowledge can be expressed, and, therefore, what knowledge can be learned. The language has to be chosen with care in order to easily express all the desired knowledge. Most commonly used languages are constrained forms of predicate calculus, like decision trees, production rules, semantic nets and frames.

The background knowledge includes a preference criterion allowing the reduction of the set of hypotheses to a smaller one containing the most preferable hypotheses. Typically, the preference criterion characterises the desired properties of the searched inductive hypothesis. This criterion is necessary when the description language is complete, i.e. all possible hypotheses can be expressed. An alternative way to constrain the hypothesis space is using a biased description language in which not all the possible hypotheses can be expressed (i.e. the language is incomplete).

Utgoff and Mitchell [86] defined *bias* as anything influencing the way in which the induction is made. Thus, the notion of bias includes any input besides examples, and any parameter or strategy that may be modified by the user of a learning system. The declarative bias, i.e. a bias explicitly given by the user, has as an advantage the possibility to be used in several systems and allows meta-level reasoning about it. This second advantage is important since if a current bias is considered insufficient, it may be changed [85].

Inductive concept learning methods can be classified according to the following perspectives:

Supervised/unsupervised: Supervised methods need an oracle that provides the classes (concepts) to which the examples belong and that classifies the examples. Unsupervised methods have to discover the concepts to which the examples belong.

Single/multiple concept learning: This classification is according to the number of concepts that have to be learned. Single concept learning can be achieved in two situations: (1) inputs are only positive examples, (2) inputs are positive and negative examples. Multiple concept learning also distinguishes two cases: (1) when the descriptions of the different classes (concepts) are mutually disjoint, (2) when the descriptions of concepts overlap, i.e. an example can satisfy the descriptions of several classes. Multiple concept learning has been implemented in AQVAL/1 and AQ11 [53].

Propositional/relational learners: Methods using a formalism equivalent to propositional calculus are called *attribute-value learners* or *propositional learners*. These methods use objects described as a fixed collection of attributes, each of them taking a value from a corresponding pre-specified set of

values. Methods that learn first-order relational descriptions are called *relational learners*. They induce descriptions of relations and use objects described in terms of their components and relations among components. The background knowledge is formed by relations which, as the language of examples and concept descriptions, are typically subsets of first order logic. In particular, learners that induce hypotheses in the form of logic programs (Horn clauses) are called *inductive logic programming* systems.

An important application of inductive learning is the automatic construction of KB for Expert Systems, being an alternative to classic knowledge acquisition methods. Inductive techniques can also be used for the refinement of existing KB since they allow the detection of inconsistencies, redundancies, and lack of knowledge, and also allow the simplification of the rules provided by the domain expert. Application domains such as biology, psychology, medicine and genetics have benefited from the capability of inductive methods to detect patterns present in the input examples. In the following sections both propositional and relational learners are explained in more detail.

1.1. Propositional learners

Propositional learners use examples described in terms of a fixed set of attributes, each one having its own set of values. Examples are classified, i.e. the class to which they belong is known. The abstraction level of attributes affects the induced description, since high-level attributes produce more understandable descriptions, i.e. more compact descriptions. Thus, the selection of attributes describing the examples determines the scope of the descriptions that can be learned (representational bias). Propositional learners assume that all the examples are described using the same set of attributes, otherwise they need mechanisms for handling imperfect data. The learning task of propositional learners can be described as follows:

Given: a set of correctly classified training examples.

Find: a rule predicting the class to which seen or unseen examples belong.

Representative propositional learners are the family of AQ algorithms [52] and the ID3 algorithm [71]. AQ algorithms induce description rules (having an if-then form) for each class. A class is described by a disjunctive logic expression, i.e. the disjunction of several clauses. The description of each class (hypothesis) is searched taking as negative examples the training examples belonging to other classes and applying the *set covering* algorithm explained in Section 1.2.1.

The ID3 algorithm expresses the learned knowledge using *decision trees*. Each internal node of a decision tree is labelled by an attribute and links from a node are labelled by the possible values of the attribute (see Fig. 1). The tree construction process is based on the selection of the most informative attribute, trying to minimise the number of tests (i.e. the length of paths from root to leaves). The algorithm used by ID3 to build a decision tree is the following [45]:

Initialise $E_{curr} := E$; $T := nil$

function **DT**(T, E_{curr})

  if all examples in E_{curr} belong to the same class C_s

    then generate a leaf labelled C_s

    else {generation of a new node}

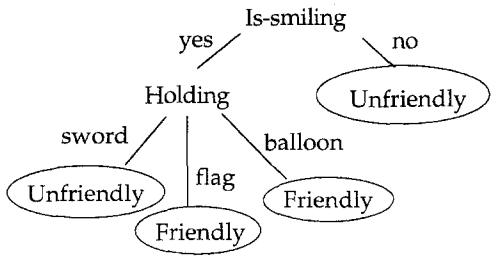


Fig. 1. An example of decision tree belonging to robots domain [45]. The set of values for the is-smiling attribute is {yes, no}, and the set of values for the attribute holding is {sword, flag, balloon}.

select the most informative attribute A with values $\{v_1 \ldots v_n\}$
Split E_{curr} into subsets $E_1 \ldots E_n$ according to the values $v_1 \ldots v_n$
for i = 1 **to** n **do**
DT(T_i, E_i)
end-for
end-if
output: Decision tree T with the root A and subtrees $T_1 \ldots T_n$

If all the input examples in E_{curr} belong to the same class C (represent the same concept), the decision tree contains only a leaf corresponding to C. Otherwise, input examples belong to several classes $C_1 \ldots C_n$. In that situation, the algorithm selects an attribute and divides E in disjoint sets $E_1 \ldots E_n$. Each set E_i of the partition contains examples having the same value in the selected attribute. The algorithm is applied to each partition set E_i until the obtained sets contain elements belonging to a unique class. Tree leaves are classes $C_1 \ldots C_n$ in which the examples can be classified.

An unseen example is classified starting from the root, testing attributes of internal nodes, to a leaf. The assumption made in the decision tree is that examples belonging to different classes have different values in at least one of their attributes.

The selection of the attribute that divides the set of examples is a crucial decision. There are many possible measures to make this selection, in particular the entropy-based measures are the most commonly used. The main idea is to select an attribute producing a maximum information gain. ID3 uses the following expression to compute the gain obtained in selecting the attribute A_k

$$\text{Gain } (A_k, X) = I(X) - E(A_k, X)$$

where $I(X) = -\Sigma_{j=1}^{m} P_j \log_2 P_j$ with $P_j = |X \cap F_j|/|X|$ and $E(A_k, X) = \Sigma_{i=1}^{n} (|X_i|/|X|) I(X_i)$, where m is the number of possible classes, n the number of possible values of the attribute A_k, $|X|$ the number of examples in the node, and $|X_i|$ the number of examples in X having the value v_i in the attribute A_k. $I(X)$ measures the randomness of the distribution of the examples in X over m possible classes. P_j is the probability of occurrence of each class F_j in the set of examples, defined as the proportion of examples in X belonging to the class F_j.

The gain is computed for all the attributes describing the examples, and the attribute having maximum gain is selected. This measure preferentially selects attributes with a large set of values, for this reason Quinlan [71] introduced a correction defining the Gain Ratio as follows:

$$G_R(A_k, X) = \frac{I(X) - E(A_k, X)}{IV(A_k)} \quad \text{where} \quad IV(A_k) = -\sum_{i=1}^{n} \frac{|X_i|}{|X|} \log_2 \frac{|X_i}{|X|}$$

This normalization has, however, a rather ad hoc justification. An alternative to the Gain Ratio is the distance-based measure introduced in [51]. This measure is based on defining a distance between partitions of the data. Each attribute is evaluated based on the distance between the data partition it creates, and the correct partition (i.e. the partition that perfectly classifies the training data). The attribute whose partition is closest to the correct one is chosen. It is formally proved (see [51]) that such distance measure is not biased towards attributes with a large number of values. Furthermore, it avoids the practical difficulties associated with the Gain Ratio and produces statistically significant smaller trees.

In [43] an analysis of the behaviour showed by measures commonly used to divide the training set can be found.

Problems exhibited by decision trees can be classified in two categories: algorithmic problems and problems inherent to the representation. The cost of top-down decision tree induction algorithms can be reduced by implementing a greedy approach searching for a small tree. Then, selection measures can be used to estimate which attribute provides the maximum information gain if it is included in the decision tree. Several algorithms have been developed to improve the decision tree when domain concepts are hard (concepts represented by many relevant attributes with high interaction among them). For example, FRINGE [67] uses the decision tree produced by a greedy splitter to construct new features improving the tree quality. The Lookahead Feature Construction algorithm (LFC) [73] is a global search algorithm that caches search information as newly constructed features.

Two problems inherent to the decision tree representation are replication and fragmentation [67]. The *replication* problem produces the duplication of sub-trees in disjunctive concepts. Replication degrades accuracy, consistency and comprehensibility. The *fragmentation* problem causes the partition of examples in small sets. Replication always implies fragmentation, but fragmentation can occur without replication if many attributes are tested (long paths). Another problem inherent to the representation is the handling of unknown values. When an attribute of an internal node has an unknown value in the current example, it is not possible to decide how the remaining sub-tree has to be explored. Usually, algorithms have special (and expensive) mechanisms to deal with unknown values.

The main limitations of propositional learners are the limited expressiveness of the representational formalism and their limited capability of taking into account the available background knowledge. The predictivity of propositional learners is highly dependent on the form in which the training set is divided by the heuristics used.

Some systems expressing the learned knowledge as decision trees are CART [14] and ASSISTANT [17]. ASSISTANT extends the ID3 algorithm allowing the manipulation of incompletely specified examples, the binarisation of continuous attributes, the construction of binary trees, the pruning of the tree and the plausible classification based on the naive Bayesian principle.

1.2. Relational learners

Relational learners are able to deal with structured objects, i.e. objects described in terms of their components and relations among components. Learned knowledge are descriptions of relations (i.e.

definitions of predicates). In relational learners the languages used to represent examples, background knowledge and concept descriptions are usually subsets of first-order logic.

In the next section we describe FOIL, a system that learns Horn clauses from data expressed as relations. FOIL is based on ideas that have proved to be effective in propositional learners, and extends them to a first-order formalism. In Section 1.2.2 we introduce a group of relational learners, called *inductive logic programming* systems, that uses knowledge represented as Horn clauses. Many authors consider FOIL as one of the early systems that can be included in the Inductive Logic Programming framework.

1.2.1. FOIL

FOIL [72] is a system incorporating ideas of both propositional and relational learners. Objects are described using relations from which FOIL generates classification rules expressed in a restricted form of first-order logic. In particular, FOIL uses the set covering approach as in AQ [52], a heuristic information-based search taken from ID3 [71] and the idea of the top-down searching of refinement graphs taken from MIS [79].

FOIL follows three main steps: (1) pre-processing of the training set, (2) construction of hypotheses, and (3) post-processing of hypotheses. During pre-processing, negative examples are generated using the closed-world assumption. Post-processing is a pruning process in order to reduce the complexity of the constructed hypothesis. Basically, pruning consists of removing irrelevant literals from a clause and removing irrelevant clauses from the hypothesis (see [45]).

Hypothesis construction is made using the *set covering algorithm*. Let us suppose that we want to find the description of N classes $C_1 \ldots C_N$. This problem can be transformed in N problems each one finding the description of one class C_i. In this transformation, examples belonging to the class C_i form the set of positive examples and examples belonging to other classes are considered negative examples of the class C_i. The set covering algorithm to construct a hypothesis has the following three steps:

(1) Search for a conjunction of conditions satisfying some examples of the class C_i and none of other classes (clause construction step).
(2) Add the obtained conjunction as a disjunct of the hypothesis that is searching for (hypothesis construction step).
(3) Delete from the training set the examples satisfying the obtained conjunction. If the obtained set is not empty the whole process is repeated until all positive examples are covered.

Clauses are constructed using a *beam search* algorithm. This algorithm begins with a clause c containing all the attributes that describe the examples and each attribute has as value the disjunction of all the possible values. The clause c is successively specialised until no negative examples are covered. The specialisation consists of removing values from attributes (there are many possible values to remove). The obtained specialisations are evaluated using a quality criterion. When at each step there are several rules to choose, they are ordered according to several criteria. First, the algorithm prefers those clauses covering as many examples as possible; then it prefers those clauses having a smaller number of attributes; and finally it prefers those clauses having the least total number of values in the internal disjunctions.

From this set covering algorithm several improvements have been developed. For example AQ15 [54] and NEWGEM [45] incorporate incrementality and initial hypothesis provided by the user; AQ17

[89] incorporates constructive induction; CN2 [18] combines the ID3 algorithm to handle imperfect data using the same flexible strategy used by AQ.

Imperfect data are handled in FOIL using a stopping criterion based on the encoding length restriction. Thus, the construction of a clause stops when no negative examples are covered or when no more bits are available to add more literals to the body hypothesis. The search for more clauses stops when all the positive examples are covered or when no more clauses may be constructed under the encoding length constraint. In [72] the application of FOIL to several domains can be found.

1.2.2. ILP

Inductive Logic Programming (ILP) has been defined as the intersection of inductive learning and computational logic [50] since it uses techniques of both fields. ILP inherits two goals from inductive learning: (1) to develop tools and techniques to induce hypotheses from observations (or examples), and (2) to synthesise new knowledge from experience (background knowledge). ILP uses computational logic as the representation mechanism of both hypotheses and observations, avoiding the two main limitations of the classic Machine Learning techniques: (1) use of a limited knowledge representation formalism (usually propositional logic), and (2) problems in using background knowledge. Thus, ILP inherits the representational formalism from computational logic, its semantics and several well-established techniques. For an overview of ILP see [62].

The main concerns of ILP are inference rule properties, algorithm convergence and computational complexity of the procedures. ILP extends theory and practice of computational logic using induction as basic inference rule. Plotkin's work on inductive generalisation [69], the work on model inference by Shapiro [79], and the work of Sammut and Banerji [75] inspired the efforts made by Lapointe and Matwin [44], and Idestam-Almquist [35] in studying the implication operator. These authors studied inductive rules regarding them as inverse of deduction rules. This inversion is made by introducing a partial order: the θ-subsumption. Nevertheless, θ-subsumption is not enough to handle recursive clauses since it is incomplete with respect to implication.

ILP research has commonly focused on concept learning, where the examples are implications and the goal is to induce a hypothesis able to classify the examples correctly. Concept learning uses positive and negative examples to induce a discriminant description for a concept. In Data Mining and in Knowledge Discovering in Databases, there is a large amount of data available and the main goal is to find properties or regularities that they present instead of discriminant descriptions (since all examples are considered positive). Flach [25] formalised these two types of induction as explanatory and confirmatory induction respectively.

The use of a relational formalism allows the application of induction over domains in which an attribute-value representation is not sufficient. Thus, ILP is being successfully applied in areas as knowledge acquisition, knowledge discovery in databases, scientific knowledge discovery, logic program synthesis and inductive engineering. Classical ILP applications were protein secondary-structure prediction [61], finite element mesh design [23] and automatic construction of qualitative models [13]. An important application is also the construction of programming assistants, that is tools supporting software design and implementation [79,72,38,21].

In the following sections we will describe the basic concepts and techniques of ILP, how the systems using ILP can be classified, and some representative ILP systems.

1.2.2.1. Basic concepts. Lavrac and Dzeroski [45] give the following description of the Inductive Logic Programming (ILP) problem:

Given: Background knowledge BK, and a set of training examples $E = E^- \cup E^+$, where E^- are negative examples and E^+ positive examples.

Find: A hypothesis H expressed in some concept description language L, such that H is complete and consistent with respect to BK and E.

Given background knowledge BK, a hypothesis H, and a set of examples E, we say that:

- A hypothesis H *covers* an example $e \in E$ if $BK \cup H \models e$.

According to this definition, *completeness* means that all the positive training examples are covered by H ($BK \cup H \models e_i^+ \ \forall e_i^+ \in E^+$) and *consistency* means that no negative example is covered by the hypothesis H ($BK \cup H \not\models e_j^- \ \forall e_j^- \in E^-$). ILP can be viewed as a search problem [62] since there is a candidate solution space and an acceptance criterion characterising the solutions. This is a similar vision of that provided by Mitchell [55] for concept learning in ML in general. The hypothesis space contains descriptions of concepts and the goal is to find one or more states satisfying some given quality criteria. The space of possible hypotheses is very wide, thus pruning and heuristic techniques are needed to find an appropriate hypothesis. As in concept learning, the hypothesis space in ILP is structured according to the notions of specialisation and generalisation.

- An hypothesis G is *more general than* a hypothesis S if and only if $G \models S$ (we can also say that S is *more specific than* G).

Generalisation corresponds to induction and specialisation corresponds to deduction. Therefore, some authors propose that induction can be viewed as the inverse of deduction. From this assertion both the specialisation and the generalisation rules can be defined as follows [62]:

- A deductive inference rule is a *specialisation rule* when it maps a conjunction of clauses G into a conjunction of clauses S such that $G \models S$.
- An inductive inference rule is a *generalisation rule* when it maps a conjunction of clauses S into a conjunction of clauses G such that $G \models S$.

The generalisation and specialisation allow pruning the hypothesis space since:

(1) if $BK \wedge H \not\models e^+$ no specialisation of H will satisfy e^+,

(2) if $BK \wedge H \wedge e^- \models \square$, any generalisation of H will be consistent with $B \wedge e^-$.

The hypothesis space can also be constrained using a bias. The definition of bias used in ILP is the same that Mitchell [56] introduced. ILP has focused on the analysis of the *declarative bias*, i.e. the bias explicitly provided by the user and that may be a modifiable parameter of the system.

Nedellec et al. [65] propose three kinds of declarative bias: language bias, search bias and validation bias. The *language bias* defines the language that determines the space of possible descriptions of concepts. The *search bias* determines which part of the hypothesis space is explored and how it is done. Finally, the *validation bias* determines an acceptance criterion, i.e. when the system considers that the searched hypothesis has been found. In [65] a further analysis of these three kinds of bias can be found.

Declarative bias influences learning in the sense that a strongly biased (less expressive) language produces smaller search space and, thus a more efficient learning [87]. Strongly biased languages have a shortcoming: sometimes the solution cannot be expressed in this language, and, therefore, it cannot be found. On the other hand, an expressive language allows more possible hypothesis, therefore the method has to perform more search in order to find the appropriate hypothesis.

Once the hypothesis space has been defined, a systematic procedure to explore it has to be defined.

The usual way is to introduce a partial order between clauses. This partial order can be based on the θ-subsumption defined by Plotkin [69].

- A clause c *θ-subsumes* a clause c′ if there exists a substitution θ such that $c\theta \subseteq c'$.
- Two clauses c and c′ are *θ-subsumption equivalent* if c θ-subsumes c′ and c′ θ-subsumes c.
- A clause is *reduced* if it is not θ-subsumption equivalent to any proper subset of itself.

According to this definition a clause c is *at least as general as* a clause c′, written $c \leqslant c'$, if c θ-subsumes c′. A clause c is *more general than* c′ if $c \leqslant c'$ but not $c' \leqslant c$.

In [62] several properties of the θ-subsumption can be found. We want to emphasise two of these properties:

(1) if c θ-subsumes c′ then $c \models c'$ (the inverse is not true [25]),
(2) θ-subsumption defines a lattice over the set of reduced clauses. This means that there is a unique least upper bound (*lub*) and a unique greatest lower bound (*glb*).

θ-subsumption between clauses is decidable and easy to compute but sometimes does not exist. Nevertheless, using Horn clauses with a common predicate symbol and having the same sign the lgg under θ-subsumption can always be found. Thanks to the lattice formed under the θ-subsumption the least general generalisation can be defined as follows [45]:

- The *least general generalisation* (lgg) of two reduced clauses c and c′, lgg(c, c′) is the least upper bound of c and c′ in the θ-subsumption lattice.

The length of the lgg of two clauses C_1 and C_2 is $|C_1| \times |C_1|$ at most. A set of literals has a unique lgg, but several lgg can exist for a set of clauses.

Summarising, θ-subsumption is important because it allows to structure the hypothesis space and prune its exploration. Also, θ-subsumption serves as basis for two strategies exploring the hypothesis space: bottom-up (or specific to general) strategy and top-down (or general to specific) strategy. Both strategies will be analysed in the following section.

1.2.2.2. Bottom-up strategy. Bottom-up methods are based on the generalisation of a set of positive examples. The bottom-up strategy proposed by Plotkin [69] consists of building the least general generalisation of the training examples (without using the negative examples). There are three basic generalisation techniques used to generate hypotheses: relative least general generalisation, inverse resolution and inverse implication.

Relative least general generalisation [69]: The least general generalisation (lgg) generalises the positive examples without using background knowledge. Relative least general generalisation (rlgg) is introduced to take into account the background knowledge. Thus, rlgg is defined from the lgg as follow:

- The *relative least general generalisation* (rlgg) of two clauses c and c′ is their least general generalisation lgg(c, c′) relative to background knowledge BK.

Plotkin showed that the rlgg of a set of clauses sometimes does not exist, or if it exists, the rlgg can have infinite length or it may be hard to compute. Using the ij-determinacy as language bias [60], the construction of a unique and finite rlgg is assured. Buntine [15] defined a special case of relative subsumption, called *generalised subsumption*, only applicable to definite clauses.

- A clause c is *more general than* a clause c′ with respect to a background knowledge BK, if any example e that can be explained using c′ and BK ($BK \cup c' \models e$) can also be explained using c ($BK \cup c \models e$).

Generalised subsumption degenerates into Plotkin's θ-subsumption in absence of background

knowledge. Thus, θ-subsumption can be considered as a special case of generalised subsumption. On the other hand, generalised subsumption is a special case of rlgg. Generalised subsumption produces a hypothesis space smaller than the one obtained by the θ-subsumption, but it is harder to compute. In order to make easy the computation of generalised subsumption, Buntine introduced the *most specific generalisation* notion. Nevertheless, generalised subsumption has remained only as a theoretic result.

Inverse resolution: Inverse resolution was introduced in first order logic by Muggleton and Buntine [59]. These authors introduced inverse resolution in ILP as a generalisation technique inverting the Robinson's resolution rule. The idea is that because the resolution rule is complete for deduction, inverse resolution could be complete for the induction. Muggleton [58] proposed four inverse resolution rules: absorption, identification, intra-construction and inter-construction. The absorption and the identification rules invert a single resolution step and they are called *V-operators*. The intra-construction and the inter-construction rules are called *W-operators*. In [10] more information about these operators can be found. W-operators are used in *predicate invention*, a field (like *constructive induction*[1] in Machine Learning) with the goal of automatically extending the vocabulary used by the system whenever the vocabulary is insufficient to learn the desired concept. In [49] there is an analysis of the conditions under which predicate invention is necessary.

Inverse implication: Since implication is undecidable [76], implication inversion may be very expensive. Nevertheless, the implication inversion can be made using restrictions that limit the kind of clauses that can be learned. Implication inversion is interesting to induce self-recursive clauses. There are three approaches to invert implication: searching structural regularities of terms, finding structural regularities of literals and finding internal and external connections. These approaches are explained in depth in [10].

Some systems using a bottom-up strategy are the following: ITOU [74], CLINT [21], MARVIN [75], and GOLEM [23].

1.2.2.3. Top-down strategy. Top-down strategy is achieved by means of specialisation techniques. Specialisation techniques search in the hypothesis space from general to specific using specialisation operators, also called *refinement operators* [79], based on θ-subsumption. A refinement operator can be defined as follows (from [45]):

- Given a language bias L, a *refinement operator* ρ maps a clause c to a set of clauses $\rho(c) = \{c' \in L,\ c < c'\}$ which are specialisations (refinements) of c.

A refinement operator performs two operations: (1) applies a substitution θ to a clause, and (2) adds a literal (or a set of literals) to a clause. The properties of the refinement operators: global completeness, local completeness and optimality can be found in [62]. These properties are desirable to assure both that all the possible hypotheses are considered (global and local completeness) and that each clause is considered only once (optimality). A generic top-down algorithm is the following [45]:

[1] *Constructive induction* [52] is a research field of Machine Learning studying the problems that appear when the desired goal cannot be learned due to the bias at the vocabulary level. Induction is *constructive* when it generates descriptors different from those used to describe input facts. Some main issues are to determine when a new descriptor is necessary, when the new descriptors can be applied and how to evaluate their quality. INDUCE-I [52] is one of the early systems using constructive induction.

Initialise $E_{curr} := E$
Initialise $H := \emptyset$
repeat {covering}
  Initialise $c := T\leftarrow$.
  repeat {specialisation}
    Find the best refinement $c_{best} \in \rho(c)$
    Assign $c := c_{best}$
  until *Necessity stopping criterion* is satisfied
  $H' := H \cup \{c\}$;; Add c to H to get new hypothesis
  $E'_{curr} := E_{curr}$-covers(B, H', E^{+}_{curr}) ;; Remove positive examples covered by c from E_{curr}
  Let $E_{curr} := E'_{curr}$ and $H := H'$
until *sufficiency stopping criterion* is satisfied

The top-down algorithm needs the whole training set (containing positive and negative examples). The construction of a hypothesis starts from the most general clause (the empty clause) and repeatedly refines (specialises) it until no negative examples are covered. The refinement of a clause is made by adding a new literal to the existing clause. A hill-climbing strategy can be used to obtain the best refinement, which is then taken as the new clause to refine. This process is repeated until a clause satisfying the stopping criteria is found. The necessity criterion decides when it is not necessary to add more literals to a clause. The sufficiency criterion decides when it is not necessary to add more clauses to an hypothesis. Several stopping criteria can be used to decide whether the domain data is perfect or not. If data are perfect, i.e. without errors, necessity and sufficiency stopping criteria are, respectively, consistence (no negative examples are covered) and sufficiency (all the positive examples are covered). Data are imperfect when some kind of (random or systematic) error is present. Most usual errors are the following [45]:

- noise, random errors in the examples and/or in the background knowledge
- the problem space is insufficiently covered, i.e. the known examples are not a good sample of the problem space
- inaccuracy, the description language is not sufficient or is inappropriate to express the exact description of the concept to learn
- unknown values in the examples.

Noise, inadequate samples and inaccuracy are usually solved by relaxing the completeness and consistency criteria, allowing the obtained concept description to cover a few negative examples and not covering a few positive examples. The usual solution to the unknown values problem is to suppose that the unknown value is the most frequent value appearing in the examples of the same class to which the current example belongs [45].

Some existing systems using a top-down strategy are the following: CLAUDIEN [22], MIS [79], MOBAL [37], GRENDEL [19] and ML-SMART [11].

1.2.2.4. Classification of ILP systems. ILP systems can be classified according to four dimensions:

incremental/non-incremental, interactive/non-interactive, single/multiple predicate learning, and theory revision.

Incremental/non-incremental: This classification is based on the way in which the examples are obtained. Thus, a system is non-incremental if all the examples are given before any learning takes place. A system is incremental if the examples are provided to the system one by one while learning. Non-incremental systems may be top-down or bottom-up whereas incremental systems use a mixture of both strategies. Some incremental systems are MIS [79], CLINT [21], MOBAL [37], and CIGOL [59]. Some non-incremental systems are GOLEM [23], FOCL [68], GRENDEL [19], CLAUDIEN [22], and LINUS [46].

Interactive/non-interactive: This classification is depending on the external support that a system has. Thus, during the process of learning, interactive ILP systems generate their own examples and ask the user (oracle) about their label (i.e. if the generated examples are positive or negative). They may also ask the user about the validity of the generalisations constructed. Interactivity allows pruning the search space and implies incrementality. Most systems are non-interactive, and the best known interactive systems are CIGOL [59], MIS [79] and CLINT [21].

Single/multiple predicate learning: In single predicate learning a single predicate is learned from the examples whereas in multiple predicate learning the goal is to learn a set of predicate definitions. Examples of multiple predicate learning systems are MARVIN [75], MIS [79], BLIP-MOBAL [91], ML-SMART [11], CIGOL [59] and CLINT [21].

Theory revision: The theory revision problem consists of modifying the hypothesis according to new evidence. Theory revision is a usual form of incremental multiple predicate learning. A theory may be inconsistent due to several predicates, for this reason multiple predicate learning is necessary. In [91] an in-depth analysis of the theory revision problem in ILP can be found. Classic theory revisers are the systems MARVIN [75], MIS [79], BLIP-MOBAL [90], ML-SMART [11], CIGOL [59] and CLINT [21]. The current research tries to build theory revisers without an oracle (as ML-SMART and BLIP-MOBAL).

1.2.2.5. Existing ILP systems. According to [62], there are six systems that have strongly contributed to ILP development: MIS [79], MOBAL-BLIP [37], CIGOL [59], GOLEM [23], FOIL[2] [72] and CLAUDIEN [22]. Each one of them has allowed an advance in a special ILP issue and they have been the basis for the construction of other ILP systems.

The Model Inference System (MIS) was the first top-down system that used ILP, introducing techniques as graph refinement, location of incorrect clauses, and manipulation of multiple predicates. Given a language L containing definite clauses and background knowledge BK, MIS uses the following algorithm [45] to build a hypothesis H:

Initialise the hypothesis H to a (possibly empty) set of clauses in L

repeat

Read the next (positive or negative) example

repeat

[2] Muggleton and De Raedt consider FOIL as an ILP system. As we explained before, FOIL's author does not follow this classification, and calls FOIL a relational learning system.

if there exists a covered negative example e **then**
Delete incorrect clauses from H
if there exists a positive example e not covered by H **then**
with breadth-first search of the refinement graph develop a
clause c which covers e and add it to H
until H is complete and consistent
output: hypothesis H
forever

MIS interactively accepts new training examples. Clause construction is made searching for a new clause that covers a positive example not covered by the current hypothesis. Search begins by the most general clause and continues by searching clause refinements in a top-down manner obtaining at each step all minimal refinements of the current hypothesis. From the set of minimal refinements those that do not cover the current positive example are rejected. The process finishes when an acceptable consistent hypothesis is found. Some systems based on MIS are CLINT [21] and MARKUS [30].

MOBAL [57] is an integrated knowledge acquisition environment consisting of several tools: a model acquisition tool to design rule models that constraints search in the hypothesis space; a sort taxonomy tool that clusters constant terms occurring as arguments in training examples; and a predicate structuring tool that abstracts rule sets to a topology hierarchy. Using these tools MOBAL can constraint the search space.

MARVIN [75] was the first system implementing inverse resolution but without a solid theoretical basis. The CIGOL [59] system was the first system that formalised the theory of inverse resolution when definite clauses are used. The LOPSTER [44] system formalises the inversion of the implication.

GOLEM is based on the notion of relative least general generalisation defined by Plotkin. The training examples and the background knowledge are ground facts and function symbols in the terms are allowed. To generate a clause GOLEM randomly takes several pairs of positive examples and their rlgg. The rlgg covering more positive examples is selected and then a new generalisation is attempted. This generalisation is made by taking a new positive example and computing the rlgg of this clause (also a rlgg) and the new positive example. The process finishes when no new positive examples are covered. In post-processing, irrelevant literals are eliminated. To generate more than one clause, GOLEM uses the set covering approach, i.e. builds a clause covering a subset of positive examples, deleting the covered examples from the training set and searching for new clauses that may cover the remaining positive examples.

Since rlgg can contain infinitely many literals or at least grow exponentially with the number of examples, GOLEM uses some restrictions to avoid the growth of rlgg. One of these constraints is the use of determinate clauses[3] in the rlgg body. Negative examples also allow the reduction of the rlgg size. Therefore, if the elimination of a literal does not result in the covering of negative examples,

[3] A clause is *determinate* if each of its literals is determinate. A literal is *determinate* if each of its variables that do not appear in preceding literals has only one possible binding given the bindings of its variables that appear in preceding literals [45].

then it is redundant and can be eliminated. This process is repeated until all the redundant literals are eliminated.

The main contribution of FOIL [72] is to recognise the expressivity the logic programming as a representation language for inductive learning. This system applies a combination of already known Machine Learning techniques but using a more expressive language. Following the same idea of using classic Machine Learning techniques, LINUS [45] transforms some ILP problems to an attribute representation form, uses a propositional learner and finally translates the results to Horn clauses.

CLAUDIEN [22] combines data mining principles with ILP. It can be considered the first system that discovers clausal regularities from unclassified data. CLAUDIEN is based on a top-down iterative depth search using refinement under θ-subsumption.

2. Lazy learning

Some authors have defined lazy learning methods as those methods that use extensional descriptions of concepts without generating intensional descriptions. In fact, this is the main difference between inductive and lazy learning methods. Inductive learning methods produce intensional descriptions of concepts from descriptions of examples (instances).

In lazy learning methods, both the learning process and the process of using the learned knowledge for solving new problems cannot be considered separately. When solving a new problem P using a lazy learning method, the solution for an old problem (perhaps transformed) is transferred to P. There is an implicit generalisation between the old problem and the new problem P. Thus, lazy learning methods use extensional descriptions of concepts (the instances) and the generalisation step is delayed to the problem solving phase. We define the lazy learning methods as those that are *problem-centred*, in the sense that the generalisation step is made on-demand, when a new problem has to be solved.

The basis of many lazy learning algorithms is the k-nearest neighbour classifier[4], k-NN [20] that stores the training set and postpones all effort towards classification decision until problem solving. Given a new example to classify, the k-NN algorithm retrieves the k most similar instances and predicts the class to which the new example can belong. The quality of the k-NN depends on which instances are considered as more similar. The similarity is estimated using the following distance function:

$$d(x, e) = \left(\sum_{f \in F} \omega(f) \cdot \delta(x_f, e_f)^2 \right)^{1/2}$$

where $e = (e_1, \ldots, e_n)$ is the new example to classify, $x = (x_1, \ldots, x_n)$ is a training example, $\omega(f)$ defines a weighting function and the function $\delta(\ )$ defines how the values of a given feature differ. In particular, $\delta(\ )$ can be defined as follows:

$$\delta(x_f, e_f) = \begin{cases} |x_f - e_f| & \text{if } f \text{ is continuous} \\ 0 & \text{if } f \text{ is discrete and } x_f = e_f \\ 1 & \text{if } f \text{ is discrete and } x_f \neq e_f \end{cases}$$

[4] k-Nearest Neighbour algorithm is a noise-tolerant extension of nearest neighbour algorithms [14].

Notice that $d(x, e)$ is computed for each precedent x, thus precedents can be ordered according to their similitude with the new example e. One consequence is that several precedents can be at the same distance of e, therefore a new example can have several solutions.

The result produced by the k-NN is the most plausible class for e, in other words, e is classified as belonging to a class such that $k - NN(e) = \max_{c_j \in J} p(c_j \mid e)$, where

$$p(c_j \mid e) = \frac{\Sigma_{X \in K_e} 1(x_c = c_j) \cdot K(d(x, e))}{\Sigma_{X \in K_e} K(d(x, e))}$$

where $1(x)$ is a function that yields 1 iff the argument x is true, and K is a kernel function defined as the inverse of the distance above.

The main shortcoming of k-NN algorithms is its sensitivity to the distance function being used. Nevertheless several variations of the algorithm have been proposed in order to reduce this sensitivity. In [88] an in-depth-study of some of these variations and the improvements that they produce can be found.

Lazy learning approaches have been typically associated to analogical reasoning [16]. Some well known lazy learning methods are derivational analogy, instance-based learning methods and Case-based Reasoning (CBR). Nevertheless, they can also be combined with 'pure' inductive methods as decision trees. In particular, LazyDT method [26] builds decision trees in a problem-centred way, constructing the best decision tree for each test instance. In fact, the LazyDT method only needs constructing one path, the one classifying the current instance.

In the following sections two lazy methods such as instance-based and CBR are analysed.

2.1. Instance-based learning

Instance-based learning (IBL) algorithms are derived from the k-NN classifier [20] since they assume that similar instances have similar classifications. IBL algorithms are inspired by the exemplar-based models of categorisation [81]. An exemplar model represents each concept as a set of exemplars, where each exemplar may be either a concept abstraction or an individual instance of the concept. The main characteristics of exemplar models are: (1) concepts are not represented as a set of necessary and sufficient conditions abstracted from exemplars, (2) descriptions are disjuncts, and (3) properties of a concept are a function or a combination of the properties of exemplars. Smith and Medin proposed two basic exemplar models, the *proximity model* and the *best examples model*. The *proximity model* stores all the training instances without performing any abstraction over them. Thus, a new instance is classified by computing its similarity with each one of the existing instances. The new instance is classified as belonging to the concept of the most similar instance. A variation of this model is the *best examples model* in which only the more typical instances (prototypes) of each concept are stored.

One of the early works on IBL was made by [36]. At that time, there was much research about how to generalise concepts from examples but there was not much research about directly using the stored examples. The primary output of IBL algorithms is a concept description, in the form of a function that maps instances to concepts. An instance-based concept description includes a set of stored instances and, possibly, some information concerning their past performance during classification (e.g., their number of correct and incorrect classification predictions). This set of instances can change

after each training set instance is processed. Concept descriptions are determined by how the similarity and classification functions use the training instances. The following algorithm, called IB1, is the simplest IBL algorithm [5]:

```
concept-description←Ø
for each x ∈ concept-description do
  for each y ∈ concept-description do
    Sim[y]←Similarity (x, y)
  end-for
  y_max←some y ∈ concept-description with maximal Sim[y]
  if class (x) = class (y_max)
    then classification←correct
    else classification←incorrect
  end if
  concept-description←concept-description ∪ {x}
end-for
```

Where $Similarity(x, y) = \sqrt{\Sigma_{i=1}^{n} f(x_i, y_i)}$ is the similarity function assuming that instances are described by n attributes. The function f is defined as follows:

$$f(x_i, y_i) = \begin{cases} (x_i - y_i)^2 & \text{for numeric-valued attributes} \\ 1 & \text{if } x_i \neq y_i \text{ for boolean and symbolic-valued attributes} \\ 0 & \text{if } x_i = y_i \text{ for boolean and symbolic-valued attributes} \\ 1 & \text{if both } x_i \text{ and } y_i \text{ are missing attributes} \end{cases}$$

IB1 is identical to the nearest neighbour algorithm except that it normalises the ranges of attributes, processes instances incrementally, and has a simple policy for tolerating missing values.

The model presented by Kibler and Aha [36] does not store all the instances and also avoids the prototype construction process. Their proposal was the *Grow (additive) algorithm* and the *Shrink (subtractive) algorithm*. The growth algorithm (also called IB2) is like the IB1 algorithm but only stores instances that have not been correctly classified. The Grow algorithm reduces storage space but its main shortcoming is that the classification accuracy decreases, especially when domains are noisy or have many exceptional instances.

The Shrink algorithm (also called IB3) tries to classify each instance from the others. If it does, the instance is not to be stored. The idea behind this algorithm is that the system accuracy is not necessarily better if all the instances are stored. In fact, the system accuracy may decrease if the stored instances are not typical. The Shrink algorithm is highly sensitive to the number of irrelevant attributes used to describe the instances. In [5] an in depth analysis of the behaviour of the three algorithms (IB1, Grow and Shrink) can be found.

The main advantages of IBL algorithms are its simplicity, its support to relatively robust learning algorithms (allowing noise and irrelevant attributes), and its relatively low updating costs.

Emde and Wettschereck [24] have proposed the adaptation of instance-based methods to ILP representation. They have developed the Relational Instance-Based Learner (RIBL) algorithm that has been implemented as an external tool of MOBAL. RIBL first generates cases from the examples. Each

case (represented as a conjunction of literals) contains an object, its description and its relations with other objects. The similarity between cases is estimated using a modified version of Bisson's algorithm [12]. Finally, a generalised form of k-NN algorithm that handles relational representation is applied.

Instance-based learning methods, like inductive methods, can improve their behaviour by means of an accurate selection of attributes describing the instances. Instance-based learning methods can also be improved by introducing constructive induction in order to adequate the instance language and the language required to represent concept descriptions. An instance-based method introducing constructive induction of features is IB3-CI [4].

2.2. Case-based reasoning

The experience in solving problems is a human quality that is necessary to capture in order to create a model of intelligent behaviour. Research in *Case-based Reasoning* [39,40] began in the 1980s as an attempt to provide a problem solving model closer to psychological models than ruled-based systems. The cognitive model behind Case-based Reasoning (CBR) was inspired by *scripts* [78] and by *dynamic memory* models of cognition [77].

CBR is associated to problem solving experiences and it is based on the human capability to solve new situations according to the similarity among the new situation and past situations already solved. Thus, when a new situation is similar to one or several old situations, the decisions taken and the knowledge contained in old situations provide a starting point to interpret or solve the new situation. In this way a new situation is solved taking advantage of inferences and decisions already made in past situations. Usually, each experience is considered a *case*. Experienced situations are *past cases* which may be reused to solve new problems and a *new case* is the description of a new problem to be solved.

CBR assumes that cases give operational knowledge, i.e. instances show how the knowledge can be used and applied. General knowledge has advantages such as the ability to use a domain model capable to deal with most situations. Nevertheless, sometimes general knowledge is too abstract to be applied to a particular situation and sometimes not all the needed knowledge is available. Another shortcoming of general knowledge is the difficulty to deal with exceptional situations. CBR can handle specific knowledge that is difficult to obtain using a general domain model and, in addition, general domain knowledge (rules or domain models) can be also used. On the other hand, systems using only general knowledge do not learn from experience, whereas in CBR each new solved case may be used to solve future problems.

CBR represents an integrated approach for learning and reasoning. A case-based reasoner learns from each problem solving session by storing the relevant information about the solved problem, and converting it in an available experience to solve future problems. Learning becomes a process of extracting relevant information from past problem solving experiences. Each new problem (case) has to be indexed in the system's knowledge structure in a way that can be easily retrieved when a similar problem is encountered later. Thus, from a new case a case-based reasoner can learn several things: the solution, the adaptation method used, relations with other cases, possible failures, etc.

The quality of CBR depends on several issues: (1) the experiences (cases) available, (2) the ability to understand new situations in terms of past situations, (3) the ability to adapt old solutions, (4) the ability to evaluate and fix old solutions, and (5) the capability to integrate new experiences.

CBR offers some advantages with regard to rule-based reasoning. The first one is that a library of cases seems more similar to the human expertise than a set of rules. A second advantage is that in real-world problems, cases and their solutions are easily acquired whereas it may be extremely difficult to specify all the necessary rules. On the other hand, CBR can be considered as a methodology complementary to model-based reasoning (MBR). MBR is used in domains where the causality can be well represented whereas CBR does not need a complete domain knowledge to produce correct results. Another difference is that MBR uses general knowledge whereas CBR uses specific knowledge. There are several systems, such as CASEY [42] and KRITIK [27,28] integrating MBR and CBR.

The main advantages of CBR with respect to rule-based and model-based reasoning are the following: (1) it is applicable in domains with incomplete knowledge; (2) it obtains solutions efficiently; (3) it may solve problems when no algorithmic methods are available; (4) it avoids possible problems that have already appeared; and (5) it focuses on the most important parts of a new case.

A main disadvantage of CBR is that generally it does not completely explore the whole space of solutions. As a consequence, locally optimal solutions may be found that are not the global optimal solution. Other disadvantages of the CBR are related to the bias introduced by the cases in the library, the set of retrieved cases and the blind use of old cases (i.e. the validity of an old case for the new case is not evaluated).

Case-based Reasoning is useful in a wide variety of problem solving tasks, such as planning, design and diagnosis. Planning is the process of obtaining a sequence of steps (i.e. a plan) or a schedule to achieve some goals. A planner has to assure that preconditions of each step are preserved before performing a new step. CBR handles these problems reusing old plans [6,32,29,31,3,63]. The proposed plan has to be adapted to the new goals.

Design problems are commonly defined as a set of constraints over a collection of possible components. The solution of design problems is a composite object satisfying all the problem constraints. Problem constraints can underspecify the problem and thus several solutions are possible. Sometimes problem constraints overspecify the problem and no solution is possible without relaxing some constraints. CBR suggest new designs from old designs created under similar constraints. These past cases can be adapted to obtain a new design satisfying all the desired constraints [84,64,33,82,9,34,83,92].

Diagnosis is a particular explanation problem where a problem solver has to explain a set of symptoms. A case-based diagnostician can use cases to suggest explanations for symptoms and to prevent inappropriate explanations [47,8,66,70,48].

In the next sections we explain the main CBR concepts and main CBR tasks following the CBR cycle defined in [2]: *retrieval*, that obtains precedents similar to the current problem and chooses the best precedent; *reuse*, that decides if the solution of the best precedent may be applied to the current problem or how it has to be adapted; *revision* that assesses the goodness of the adapted solution; and *retain* that decides which parts of the new case may be useful for solving new problems.

2.2.1. Library of cases

Kolodner [41] provides the following definition for cases: A case is a contextualised piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner. A case contains the description of a problem according to goals to be achieved, restrictions

about these goals, characteristics of the problem and relations between parts of the problem. Cases have been represented using different notations. Thus, CASEY [42] and PROTOS [7] use an attribute-value representation. MEDIATOR [80] uses structured representations as frames. CHEF [32] uses a hybrid representation: cases are organised as frames but slot fillers of the frame are represented using first-order predicate calculus.

The *library* (or *memory*) *of cases* contains the expertise of a case-based reasoner. The structure, content and organisation of solved cases are essential for reuse. This organisation has to be dynamic since the incorporation of a new case can influence the organisation of already existing cases. There are two possible organisations of the library of cases: flat and hierarchical. In *flat* organisation, cases are sequentially stored using a list or a file. In *hierarchic* organisation cases are organised according to their characteristics. Each node of the hierarchy corresponds to a characteristic shared by a set of cases. This organisation is appropriate when the library has a big size.

Cases are organised in the library using *indexes*. The determination of useful indexes is one of the main issues of the CBR. Some properties that indexes have to satisfy are predictivity, utility, abstraction and concretion. Thus, indexation has to allow to reuse appropriate cases supporting the achievement of the task goals (predictivity and utility). On the other hand, indexes have to be both abstract enough to make a case useful in a variety of future situations, and concrete enough to be easily recognisable in future situations. Using indexes, a case-base reasoner can retrieve a subset of cases that are potentially useful to solve the new situation.

2.2.2. Retrieval task

The main problem of a case-based reasoner is to determine past situations relevant (similar) to solve a new situation. Past situations have to be labelled and organised in the library of cases in order to be retrieved using the features of the new situation. The retrieval task has as input a (partial) problem description and provides a past case or a ranking of past cases having the best matching with the new problem. An important issue of the retrieval task is to determine useful indexes to retrieve past cases. Indexes may be the input features of the new case. Nevertheless, for knowledge-intensive methods (using general domain knowledge) more elaborated indexes can be determined.

The organisation of the library influences the retrieval of old cases. In flat organisation cases are retrieved using a matching function that is sequentially applied to each case. The main advantage of this scheme is that all the library is explored and the retrieval quality depends on the matching function. The main disadvantage is that the retrieval time increases with the number of cases. When the library of cases has a large size, a hierarchic organisation is recommended. The main advantage of this scheme is the efficiency in the retrieval process. The main disadvantages are the difficulty in adding new cases and the memory space that is used.

The similarity between cases may be evaluated in two ways: syntactically and semantically. The *syntactic similarity* uses as indexes the problem descriptors. This approach is appropriate for domains having not much general knowledge available. An example of system using syntactic similarity is CYRUS [39]. The *semantic similarity* (also called *knowledge intensive*) uses more elaborated indexes obtained by applying general domain knowledge to case descriptors. This approach is used in PROTOS [7], CASEY [42] and CREEK [1].

Similarity assessment may be knowledge-intensive, using the process of understanding the new case as guide for the matching. Another option is to weight the problem features according to their importance for characterising the problem during the learning phase. The similarity assessment task

can obtain a set of cases as best match. In this situation, an explanation justifying non-identical features for each past case of this set may be generated. When a strong enough explanation is found, the corresponding past case is selected as the best match.

2.2.3. Reuse task

The reuse of a past case solution in the context of the new case focuses on two aspects: the differences among the past cases and the new case, and which part of the retrieved case can be transferred to the new case. In classification systems, the solution of the retrieved case is directly the solution of the new case. However, other systems need a transformation process (adaptation) of the solution. Adaptation processes modify an old solution to provide a new one. The kind of adaptation varies according to the differences between the new case and the retrieved one.

There are two kinds of reuse [2]: transformational and derivational. In *transformational reuse* the past case solution is not directly a solution for the new case but there exists some knowledge in the form of transformational operators such that when applied to the old solution, transform it into a solution for the new case. Transformational reuse focuses on the equivalence of solutions, and this requires a strong domain-dependent model containing transformational operators and how they can be applied.

Another approach is when the retrieved case holds information about the method used for solving it (including a justification of the operators used, subgoals considered, alternatives generated, failed search paths, etc.). From this information *derivational reuse* reinstantiates the retrieved method to the new case and executes it into the new context.

2.2.4. Revision task

During the revision task the solution generated by the reuse task is evaluated. This evaluation is made by asking a teacher or by a real world simulation (as the CHEF system [32]). If the evaluation result determines that the proposed solution is correct, the new case may be retained. Otherwise, an opportunity to learn from failure appears since the solution case has to be repaired. Case repair involves detecting the errors of the current solution and retrieving or generating explanations for them. Errors are repaired using general causal knowledge and domain knowledge about how to avoid or compensate errors in the domain. If the revision task assures the correctness of the repaired solution, this can be retained. Otherwise the repaired plan has to be in turn evaluated and repaired [32].

2.2.5. Retain task

The retain task determines which information about the new case can be useful for solving new cases and incorporates it to the existing knowledge. Relevant problem descriptors and problem solutions are useful information but explanations of why a solution is successful or not may also be useful to store.

A main issue in retaining new cases is the *indexing problem*, i.e. to decide what type of indexes to use for future retrieval and how to structure the search space of indexes. The approach used by syntax-based methods is using all input features as indexes. Other approaches, such as those used in CASEY [42], also use as indexes the observed features.

The integration of a new case in the library of cases implies a modification of the existing indexes. The first consideration about a successful case is to consider if its incorporation to the library of cases may be useful. Notice that the systematic incorporation of all the new cases can increase the system's

inefficiency. Commonly, the future utility of the successful case is estimated before deciding whether it has to be incorporated in the library. If a successful case is similar to an existing case, the construction of a prototype or scheme generalising both cases can be considered. From a failed case, the system can learn from failure by modifying the corresponding indexes in order to prevent the retrieval of the same case in a similar situation.

Usually, failed cases produce the adjustment of the index strengths for a particular case or solution.

3. Concluding remarks

We have reported several approaches to the problem of Machine Learning from examples. These approaches have been grouped into two sets of methods: *Inductive* and *Lazy* learning. Within the inductive methods we have addressed propositional learners, relational learners and ILP-based methods. The lazy learning methods reported are instance-based learners and case-based learners. Although we did not pretend to be completely exhaustive, we believe that the approaches reported in this paper can be considered as very representative of the main existing work in this area. We expect to see much more research devoted to the subject, especially in ILP and Case-Based Learning because in both areas there are more and more active groups working worldwide.

References

[1] A. Aamodt, A knowledge-intensive, integrated approach to problem solving and sustained learning, *Ph. Dissertation* (University of Trondheim, Norwegian Institute of Technology, Department of Electrical Engineering and Computer Science, 1991).

[2] A. Aamodt and E. Plaza, Case-based reasoning: Foundational issues, methodological variations and system approaches, *Artificial Intelligence Communications* 7 (1994) 39–59.

[3] R. Aarts and J. Rousu, Towards CBR for bioprocess planning, in I. Smith, B. Faltings (eds.), *Advances in Case-based Reasoning, Third European Workshop on Case-based Reasoning*, Lausanne, Switzerland, Lecture Notes in Artificial Intelligence, Vol. 1168 (Springer-Verlag, 1996) pp. 16–27.

[4] D.W. Aha, Incremental constructive induction: An Instance-based approach, *Proc. 8th Intl. Workshop on Machine Learning* (Morgan Kaufmann, San Mateo, CA) (1991) pp. 117–121.

[5] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, *Machine Learning* 6 (1991) 37–66.

[6] R. Alterman, An adaptive planner, in *Proc. of AAAI-86* (AAAI Press/MIT Press, Cambridge, MA, 1986).

[7] E.R. Bareiss, *Exemplar-based knowledge acquisition: A unified approach to concept representation, classification and learning* (Academic Press, 1989).

[8] E.R. Bareiss, B.W. Porter, K.S. Murray, Supporting start-to-finish development of knowledge bases, *Machine Learning* 4(3/4) (1989) 259–283.

[9] B. Bartsch-Spörl, Towards the integration of case-based, schema-based and model-based reasoning for supporting complex design tasks, in M. Veloso, A. Aamodt (eds.), *Case-based Reasoning Research and Development, First Intl. Conf., ICCBR-95*, Sesimbra, Portugal, Lecture Notes in Artificial Intelligence, Vol. 1010 (Springer-Verlag, 1995) 145–156.

[10] F. Bergadano, D. Gunetti, *Inductive Logic Programming, From Machine Learning to Software Engineering* (Massachusetts Institute of Technology, 1996).

[11] F. Bergadano, A. Giordana, L. Saitta, Concept acquisition in noisy environments, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10 (1998) 555–578.

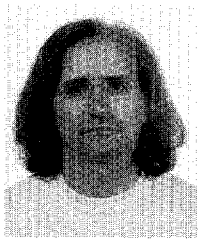
[12] G. Bisson, Learning in FOL with a similarity measure, *Proc. of the AAAI-1992* (1992) pp. 82–87.

[13] I. Bratko, S. Muggleton, A. Varsek, Learning qualitative models of dynamic systems, in *Proc. 8th Intl. Workshop on Machine Learning* (Morgan-Kaufmann, San Mateo, CA, 1991) 385–388.
[14] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, *Classification and Regression Trees* (Wadsworth, Belmont, CA, 1984).
[15] W. Buntine, Generalized subsumption and its applications to induction and redundancy, *Artificial Intelligence* 36(2) (1988) 149–176.
[16] J.G. Carbonell, Derivational analogy: A theory of reconstructive problem solving and expertise acquisition, in R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. II (Tioga Publishing Company, Palo Alto, CA, 1986) pp. 371–392.
[17] B. Cestnik, I. Kononenko, I. Bratko, ASSISTANT 86: A knowledge elicitation tool for sophisticated users, in I. Bratko, N. Lavrac (eds.), *Progress in Machine Learning* (Sigma Press, Wilmslow, 1987) pp. 31–45.
[18] P. Clark, T. Niblett, The CN2 induction algorithm, *Machine Learning* 3 (1989) 261–284.
[19] W. Cohen, Gramatically biased learning: Learning logic programs using an explicit antecedent description language, *Artificial Intelligence* (1994) pp. 301–366.
[20] B.V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques* (IEEE Computer Society Press, Los Alamitos, CA, 1991).
[21] L. De Raedt, *Interactive Theory Revision: An Inductive Logic Programming Approach* (Academic Press, 1992).
[22] L. De Raedt, M. Bruynooghe, A theory of clausal discovery, *Proc. 13th Intl. Joint Conf. on Artificial Intelligence* (Morgan Kaufmann, 1993) 1058–1063.
[23] B. Dolsak, S. Muggleton, The application of inductive logic programming to finite element mesh design, in S. Muggleton (ed.), *Inductive Logic Programming* (Academic Press, London, 1992) 453–472.
[24] W. Emde, D. Wettschereck, Relational instance-based learning, in Lorenza Saitta (ed.), *Intl. Conf. on Machine Learning*, Bari, Italy (1996) pp. 122–130.
[25] P. Flach, Logical approaches to machine learning: An overview, *THINK* 1(2) (1992) 25–36.
[26] J.H. Friedman, R. Kohavi, Y. Yun, Lazy decision trees, *Proc. AAAI-96* (1996) 717–724.
[27] A. Goel, A model-based approach to case adaptation, in *Proc. 8th Intl. Machine Learning Workshop* (Morgan Kaufmann, 1991).
[28] A. Goel, B. Chandrasekaran, Use of device models in adaptation of design cases, in K.J. Hammond (ed.), *Proc. 2nd Workshop on Case-Based Reasoning*, Pensacola Beach, FL (Morgan Kaufmann, 1989).
[29] M. Goodman, CBR in battle planning, in *Proc. Workshop on Case-based Reasoning (DARPA)*, Pensacola Beach, FL (Morgan Kaufmann, 1989).
[30] M. Grobelnik, Markus: An optimized model inference system, *Proc. ECAI Workshop on Logical Approaches to Machine Learning* (1992).
[31] K.Z. Haigh, M. Veloso, Route planning by analogy, in M. Veloso, A. Aamodt (eds.), *Case-based Reasoning Research and Development, 1st Intl. Conf., ICCBR-95*, Sesimbra, Portugal, Lecture Notes in Artificial Intelligence, Vol. 1010 (Springer Verlag, 1995) pp. 169–180.
[32] K.J. Hammond, Case-based planning, Viewing planning as a memory task, *Perspectives in Artificial Intelligence*, Vol. 1 (Academic Press, 1989).
[33] T. Hinrichs, J. Kolodner, The roles of adaptation in case-based design, in *Proc. AAAI-91* (AAAI Press/MIT Press, Cambridge, MA, 1991) pp. 28–33.
[34] N. Hurtley, Evaluating the application of CBR in mesh design for simulation problems, in M. Veloso, A. Aamodt (eds.), *Case-based Reasoning Reasearch and Development, 1st Intl. Conf., ICCBR-95*, Sesimbra, Portugal, Lecture Notes in Artificial Intelligence, Vol. 1010 (Springer-Verlag, 1995) pp. 193–204.
[35] P. Idestam-Almquist, Generalization under implication using or-introduction, in *Proc. 6th European Conf. on Machine Learning*, Vol. 667, Lecture Notes in Artificial Intelligence (1993) pp. 56–64.
[36] D. Kibler, D.W. Aha, Learning representative exemplars of concepts: An initial case study, *Proc. 4th Intl. Workshop on Machine Learning* (University of California, Irvine, 1987) pp. 24–30.
[37] J.U. Kietz, S. Wrobel, Controlling the complexity of learning in logic through syntactic and task-oriented models, in S. Muggleton (ed.), *Inductive Logic Programming* (Academic Press, 1992) pp. 335–359.
[38] M. Kirschenbaum, L. Sterling, Refinement strategies for inductive learning of simple prolog programs, in *Proc. 12th Intl. Joint Conf. on Artificial Intelligence* (Morgan-Kaufmann, Darling Harbour, Sidney, Australia, 1991) pp. 757–761.

[39] J. Kolodner, Maintaining organization in a dynamic long-term memory, *Cognitive Science* 7 (1983) 243–280.
[40] J.L. Kolodner, *Proc. Workshop on Case-based Reasoning (DARPA)*, Clearwater, FL (Morgan Kaufmann, San Mateo, CA, 1988).
[41] J.L. Kolodner, *Case-Based Reasoning* (Morgan Kaufman, 1993).
[42] P. Koton, Reasoning about evidence in causal explanation, in *Proc. of AAAI-88* (AAAI Press/MIT Press, Cambridge, MA, 1988) pp. 256–261.
[43] I. Kononenko, On biases in estimating multi-valued attributes, in *Proc. Intl. Joint Conf. on Artificial Intelligence*, Montreal, Canada, 1995) pp. 1031–1040.
[44] S. Lapointe, S. Matwin, Sub-unification: A tool for efficient induction of recursive programs, in *Proc. 9th Intl. Workshop on Machine Learning* (Morgan Kaufmann, 1992) pp. 273–281.
[45] N. Lavrac, S. Dzeroski, *Inductive Logic Programming Ellis Horwood Series in Artificial Intelligence* (Ellis Horwood, 1994).
[46] N. Lavrac, S. Dzeroski, M. Grobelnik, Learning non-recursive definitions of relations with LINUS, in Yves Kodratoff (ed.), *Proc. 5th European Working Session on Learning*, Vol. 482, Lecture Notes in Artificial Intelligence (Springer-Verlag, 1991) pp. 265–281.
[47] D.B. Leake, *Evaluating Explanations: A Content Theory* (Erlbaum, Northvale, NJ, 1992).
[48] M. Lenz, H.D. Burkhard, S. Brückner, Applying case retrieval nets to diagnostic tasks in technical domains, in I. Smith, B. Faltings (eds.), *Advances in Case-based Reasoning, 3rd European Workshop on Case-based Reasoning*, Lausanne, Switzerland, Lecture Notes in Artificial Intelligence, Vol. 1168 (Springer Verlag, 1996) pp. 219–233.
[49] C. Ling, Inventing necessary theoretical terms in scientific discovery and inductive logic programming, *Technical Report 302* (Dept. of Computer Science, University of Western Ontario, 1991).
[50] J. Lloyd, in J. Lloyd (ed.), *Computational Logic* (Springer-Verlag, Berlin, 1990).
[51] R. Lopez de Mantaras, A distance-based attribute selection measure for decision tree induction, *Machine Learning* 6 (1991) 81–92.
[52] R.S. Michalski, A theory and methodology of inductive learning, in R.S. Michalski, J.G. Carbonell, T.M. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. I (Tioga Publishing Company, Palo Alto, CA, 1983) pp. 83–134.
[53] R.S. Michalski, J.B. Larson, Selection of most representative training examples and incremental generation of VL1 hypotheses: The underlying methodology and the description of programs ESEL and AQ11, *Report No. 867* (Dept. of Computer Science, University of Illinois, Urbana, 1978).
[54] R.S. Michalski, I. Mozetic, J. Hong, N. Lavrac, The Multi-purpose incremental learning system AQ15 and its testing application on three medical domains, in *Proc. 5th National Conf. on Artificial Intelligence* (Morgan Kauffman, San Mateo, CA, 1986).
[55] T.M. Mitchell, Generalization as search, *Artificial Intelligence* 18(2) (1982) 203–226.
[56] T.M. Mitchell, The need for biases in learning generalizations, in *Readings in Machine Learning* (Morgan Kaufmann, 1991) pp. 184–191.
[57] K. Morik, Balanced cooperative modelling, *Proc. 1st Intl. Workshop on Multistrategy Learning* (George Mason University, Fairfax, VA, 1991) pp. 65–80.
[58] S. Muggleton, Duce, An oracle based approach to constructive induction, *Proc. 10th Intl. Joint Conf. on Artificial Intelligence* (Morgan-Kaufmann, 1987) pp. 287–292.
[59] S. Muggleton, W. Buntine, Machine invention of first-order predicates by inverting resolution, in *Proc. 5th Intl. Conf. on Machine Learning* (Morgan-Kaufmann, 1988) pp. 339–352.
[60] S. Muggleton, C. Feng, Efficient induction of logic programs, in *Proc. 1st Conf. on Algorithmic Learning Theory*, Ohmsha, Tokyo (1990).
[61] S. Muggleton, R. King, M. Sternberg, Protein secondary structure prediction using logic-based machine learning, *Protein Engineering* 5(7) (1992) 647–657.
[62] S. Muggleton, L. DeRaedt, Inductive logic programming: Theory and methods, *Journal of Logic Programming* (19–20) (1994) 629–679.
[63] H. Muñoz-Avila, J. Hüllen, Feature weighting by explaining case-based planning episodes, in I. Smith, B. Faltings (eds.), *Advances in Case-based Reasoning, Third European Workshop on Case-based Reasoning*, Lausanne, Switzerland, Lecture Notes in Artificial Intelligence, Vol. 1168 (Springer-Verlag, 1996) pp. 280–294.

[64] D. Navichandra, Case-based reasoning in CYCLOPS, A design problem solver, in *Proc. Workshop on Case-based Reasoning (DARPA)*, Clearwater, FL (Morgan Kaufmann, San Mateo, CA, 1988) pp. 286–301.
[65] C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, B. Tausend, Declarative bias in inductive logic programming, in L. De Ruedt (ed.), *Advances in Inductive Logic Programming* (IOS Press, 1996) pp. 82–103.
[66] B.D. Netten, R.A. Vingerhoeds, Large-scale fault diagnosis for on-board train systems, in M. Veloso, A. Aamodt (eds.), *Case-based Reasoning Research and Development, First Intl. Conf., ICCBR-95*, Sesimbra, Portugal, Lecture Notes in Artificial Intelligence, Vol. 1010 (Springer-Verlag, 1995) pp. 67–76.
[67] G. Pagallo, D. Haussler, Two algorithms that learn DNF by discovering relevant features, *Proc. 6th Intl. Workshop on Machine Learning* (Ithaca, NY, 1989) pp. 119–123.
[68] M. Pazzani, D. Kibler, The utility of knowledge in inductive learning, *Machine Learning* 9 (1992) 57–94.
[69] G. Plotkin, A note on inductive generalization, in B. Meltzer, D. Michie (eds.), *Machine Intelligence* 5 (1969) 153–163 (Edinburgh University Press, Edinburgh).
[70] L. Portinale, P. Torasso, ADAPTER: An integrated diagnostic system combining case-based and abductive reasoning, in M. Veloso, A. Aamodt (eds.), *Case-based Reasoning Research and Development, 1st Intl. Conf., ICCBR-95*, Sesimbra, Portugal, Lecture Notes in Artificial Intelligence, Vol. 1010 (Springer-Verlag, 1995) pp. 277–288.
[71] J.R. Quinlan, Induction of decision trees, *Machine Learning* 1 (1986) 81–106.
[72] J.R. Quinlan, Learning logical definitions from relations, *Machine Learning* 5 (1990) 239–266.
[73] H. Ragavan, L. Rendell, Lookahead feature construction for learning hard concepts, *Proc. 10th Intl. Conf. on Machine Learning* (University of Massachusetts, Amherst, 1994) pp. 252–259.
[74] C. Rouveirol, Extensions of inversion of resolution applied to theory completion, in S. Muggleton (ed.), *Inductive Logic Programming* (Academic Press, London, 1992) pp. 63–92.
[75] C. Sammut, R.B. Banerji, Learning concepts by asking questions, in R. Michalski, J. Carbonell, T. Mitchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Vol. 2 (Kaufmann, Los Altos, CA, 1986) pp. 167–192.
[76] M. Schmidt-Schauss, Implication of clauses is undecidable, *Theoretical Computer Science* 59 (1988) 287–296.
[77] R. Schank, *Dynamic Memory: A Theory of Learning in Computers and People* (Cambridge Univ. Press, New York, 1982).
[78] R. Schank, R. Abelson, *Scripts, Goals and Understanding* (Erlbaum, Northvale, NJ, 1977).
[79] E.Y. Shapiro, *Algorithmic Program Debugging* (MIT Press, 1983).
[80] R.L. Simpson, A computer model of case-based reasoning in problem Solving: An investigation in the domain of dispute mediation, *Technical Report GIT-ICS-85/18* (Georgia Institute of Technology, 1985).
[81] E.E. Smith, D.L. Medin, *Categories and Concepts* (Harvard University Press, Cambridge, MA, 1981).
[82] B. Smyth, M.T. Keane, Experiments on adaptation-guided retrieval in case-based design, in M. Veloso, A. Aamodt (eds.), *Case-based Reasoning Research and Development, First Intl. Conf., ICCBR-95*, Sesimbra, Portugal, Lecture Notes in Artificial Intelligence, Vol. 1010 (Springer-Verlag, 1995) pp. 313–324.
[83] J. Surma, B. Braunschweig, REPRO: Supporting flowsheet design by case-based retrieval, in I. Smith, B. Faltings (eds.), *Advances in Case-based Reasoning, 3rd European Workshop on Case-based Reasoning*, Lausanne, Switzerland, Lecture Notes in Artificial Intelligence, Vol. 1168 (Springer-Verlag, 1996) pp. 400–412.
[84] K. Sycara, Using case-based reasoning for plan adaptation and repair, in *Proc. Workshop on Case-based Reasoning (DARPA)*, Clearwater, FL (Morgan Kaufmann, San Mateo, CA, 1988).
[85] P.E. Utgoff, *Machine Learning of Inductive Bias* (Kluwer Academic Publishers, 1986).
[86] P.E. Utgoff, T.N. Mitchell, Acquisition of appropriate bias for inductive concept learning, in *Proc. National Conf. on Artificial Intelligence* (Morgan Kaufmann, Los Altos, CA, 1982) pp. 414–417.
[87] V.N. Vapnik, Y.A. Chervonenkis, Necessary and sufficient conditions for the uniform convergence of means to their expectations, *Theory of Probability and its Applications* 26 (1981) 532–553.
[88] D. Wettschereck, D.W. Aha, T. Mohri, A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, *AI-Review* 11(1–5) (1997) 273–314, Special Issue on Lazy Learning.
[89 J. Wnek, R.S. Michalski, Hypothesis-driven constructive induction in AQ17: A method and experiments, in *Proc. Intl. Joint Conf. on Artificial Intelligence* (Sydney, Australia, 1991).
[90] S. Wrobel, Automatic representation adjustment in an observational discovery system, in *Proc. 3rd European Working Session on Learning* (Pitman, 1988) pp. 253–262.
[91] S. Wrobel, First order theory refinement, in L. De Raedt (ed.), *Advances in Inductive Logic Programming* (IOS Press, 1996) pp. 14–33.

[92] Z. Zdrahal, E. Motta, Case-based problem solving methods for parametric design tasks, in I. Smith and B. Faltings (eds.), *Advances in Case-based Reasoning, 3rd European Workshop on Case-based Reasoning*, Lausanne, Switzerland, Lecture Notes in Artificial Intelligence, Vol. 1168 (Springer Verlag, 1996) pp. 473–486.



Ramon López de Mántaras is Research Full Professor and Deputy Director of the Artificial Intelligence Research Institute of the CSIC (Spanish Council for Scientific Research). He received a Ph.D. in Applied Physics (Automatic Control) in 1977 from the University of Toulouse (France), a M.Sc. in Engineering (Computer Science) from the University of California at Berkeley and a Ph.D. in Computer Science in 1981 from the Technical University of Catalonia. He held a Visiting Professor appointment at the University of Paris VI in 1986. From 1981 to 1985 he taught at the Computer Science Department of the Polytechnical University of Catalonia. He is a member of the editorial board of twelve international journals and is former Editor-in-Chief of Artificial Intelligence Communications. He was a recipient of the 'City of Barcelona' Research Prize in 1982, the 'European Artificial Intelligence Research Paper Award in 1987 and the 'Swets & Zeitlinger Distinguished Paper Award' of the International Computer Music Association in 1997, among other awards. He is presently working on the relationship between Similarity Logics and Case-Based Reasoning, on the use of Dynamic Logic to formalize Reflective Architectures, on the problem of dealing with Uncertainty in Autonomous Robots, on Multi-Agent Systems and on Case-Based Reasoning applications to Music. He is author or co-author of over 100 scientific papers and of the book 'Approximate Reasoning Models' published in the Ellis Horwood Artificial Intelligence Series.

Eva Armengol i Voltas received her Ph.D. in Computer Science from the Technical University of Catalonia in 1997. She has been active in AI research since 1987 at the UPC and since 1990 she has been working at the Artificial Intelligence Research Institute of the CSIC (Spanish Council for Scientific Research). Her research has been focused in the areas of knowledge representation, machine learning, validation of KBS and integration of learning methods. Her current interest is focused on systems integration problem solving and learning.