Induction in Logic

Luc De Raedt
Dept. of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
B-3001 Heverlee, Belgium
Email : Luc.DeRaedt@cs.kuleuven.ac.be

Abstract

Various inductive machine learning approaches and problem specifications are analyzed from a logical perspective. This results in a unifying framework for the logical aspects of inductive machine learning and data mining. The framework explains logical similarities and differences between different machine learning settings, and allows us to relate past and present work on inductive machine learning using logic (such as structural matching, inductive logic programming, data mining, etc.).

Central to the unifying framework are three dimensions: learning from entailment versus learning from interpretations, learning CNF versus DNF, learning characteristic versus discriminant descriptions.

Though the exposition handles both first order and propositional logic, it is meant to be understandable for everyone familiar with propositional logic.

Motivation

Most machine learning and knowledge discovery in databases approaches use logical representation languages to represent data and hypotheses. Despite this uniform representation language, it is often hard to appreciate the differences and similarities in this approaches. In my oppinion this is not only due to the technical nature of some of these papers (in particular in the field of inductive logic programming (Muggleton and De Raedt 1994; Muggleton 1992; De Raedt 1996)), but also to the lack of a unifying framework for these approaches.

In this paper, an attempt is made to build such a unifying framework. The unifying framework builds on past work along at least three different dimensions. The first dimension, learning from entailment versus learning from interpretations (Angluin *et al.* 1992; Frazier and Pitt 1993), determines whether the observations are logical statements that are (resp. are not) logically entailed by the target theory, or are interpretations (or variable-assignments) that satisfy (resp. do not satify) the target theory. Whereas propositional learners learn from interpretations (as for instance in computational learning theory and attribute value learning, (Valiant 1984)), most first order learners learn from entailment (in particular the field of inductive logic programming). Past work along this dimension includes (Angluin *et al.* 1992; Frazier and Pitt 1993; De Raedt and Lavrač 1993). The second dimension distinguishes conjunctive versus disjunctive normal forms, e.g. (Mooney 1995; Haussler 1988; De Raedt and Van Laer 1995). The third dimension is based on the type of learning, i.e. characteristic versus discriminant versus data-mining, and is derived from the work of Michalski (Michalski 1983). Furthermore, the unifying framework allows to give - in logical terms - precise definitions of what is meant by deduction, induction, generalization, specialization, and their relations, not only formalising these terms but also showing how their relation can be exploited. At the same time, some links between different machine learning settings are formulated in a novel way, in particular inductive logic programming versus structural matching (as studied by e.g. (Vere 1975; Vrain 1990)), and concept-learning versus data-mining.

Though the unifying framework offers some new insights, it extensively borrows from previous work, in particular from (Michalski 1983; 1994) on generalization and specialization and various other aspects, from (Niblett 1988; Kodratoff 1988) on logic and generalization, and from (De Raedt *et al.* 1996; Mooney 1995; De Raedt and Van Laer 1995) on the relation between CNF and DNF.

Though the exposition handles first order as well as propositional logic, it is meant to be understandable for everyone familiar with propositional logic.

This paper is structured as follows: first, a short review of logic is presented, followed by a short review of concept-learning; second, learning from entailment and from interpretations is formalized; third, the relation between generality, induction and logic is discussed, fourth, the differences and relations between CNF and DNF are analysed; fifth, characteristic concept-learning is reformulated as data mining; finally, some conclusions are drawn and relations to

earlier work are presented.

Logic

We first review some notions of logic.

Term A term t is either a constant, a variable or a compound term $f(t_1, ..., t_n)$ composed of a function symbol f, and n different terms t_i.

Atom An atom is a logical formula of the form $p(t_1, .., t_n)$, where p is a predicate symbol and t_i are terms.

Literal A literal is an atom or the negation $\neg A$ of an atom A. Atoms are positive literals, negated atoms are negative literals.

For instance, $pair(card(j, hearts), card(j, diamonds))$, $larger\_rank(ace, X)$ and $face\_card(j, hearts)$ are atoms, and $\neg face\_card(j, hearts)$ is a negative literal.

The main difference between terms and atoms is that one can always assign a truth-value to an atom (not containing variables) whereas a term has no truth-value. In propositional logic, no terms are considered. Thus in propositional logic, only predicate symbols are used in atoms.

CNF A CNF expression is of the form

$$(l_{1,1} \vee ... \vee l_{1,n_1}) \wedge ... \wedge (l_{k,1} \vee ... \vee l_{k,n_k})$$

where all $l_{i,j}$ are propositional literals.

DNF A DNF expression is of the form

$$(l_{1,1} \wedge ... \wedge l_{1,n_1}) \vee ... \vee (l_{k,1} \wedge ... \wedge l_{k,n_k})$$

where all $l_{i,j}$ are propositional literals.

CNF and DNF expressions are well-known in the machine learning literature, as they underly most well-known sytems. They also have a natural upgrade in first order logic, the UCNF and EDNF expressions[1].

UCNF A UCNF expression is of the form

$$(\forall V_{1,1}, ..., V_{1,v_1} : l_{1,1} \vee ... \vee l_{1,n_1})$$
$$\wedge ... \wedge$$
$$(\forall V_{k,1}, ..., V_{1,v_k} : l_{k,1} \vee ... \vee l_{k,n_k})$$

where all $l_{i,j}$ are literals and $V_{i,1}, ..., V_{i,v_i}$ are all variables occurring in $l_{i,1} \vee ... \vee l_{i,n_i}$.

EDNF A EDNF expression is of the form

$$(\exists V_{1,1}, ..., V_{1,v_1} : l_{1,1} \wedge ... \wedge l_{1,n_1})$$
$$\vee ... \vee$$
$$(\exists V_{k,1}, ..., V_{1,v_k} : l_{k,1} \wedge ... \wedge l_{k,n_k})$$

where all $l_{i,j}$ are literals and $V_{i,1}, ..., V_{i,v_i}$ are all variables occurring in $l_{i,1} \wedge ... \wedge l_{i,n_i}$.

[1] In first order logic one might also use CNF or DNF expressions with mixed quantifiers. However, as such prenex normal forms have not yet been considered in the machine learning literature, we will ignore these here.

The symbol $\forall$ reads 'for all' and stands for universal quantification, and $\exists$ reads 'there exists' and stands for existential quantification.

For instance, the formula

$$\exists C, T : triangle(T) \wedge circle(C) \wedge in(C, T)$$

states that there exist a triangle and a circle such that the circle is inside the triangle.

Notice also that UNCF form corresponds to the clausal subset of first order logic. Therefore, we will sometimes write UCNF expressions in clausal notation (where $\leftarrow$ reads 'if' and stands for implication, and each disjunction corresponds to a *clause*). E.g. the UCNF expression :

$$(\forall X : flies(X) \vee \neg bird(X)) \wedge bird(tweety)$$

would be written as:

$$flies(X) \leftarrow bird(X)$$
$$bird(tweety) \leftarrow$$

An interesting subset of clausal logic, is definite clause logic. It consists of UCNF expressions that have exactly one positive literal in each disjunction (or clause). Definite clause logic is the basis of the programming language Prolog and of the machine learning technique known under the name of inductive logic programming.

To reason about the relation among different logical formulae and about what is truth, logicians have developed model theory. Model theory starts with interpretations[2].

Herbrand Interpretation A Herbrand interpretation is a set of ground atoms.

A Herbrand interpretation correponds in the boolean or propositional case to a variable assignment. The meaning of a Herbrand interpretation is that all atoms in the interpretation are true, and all other atoms are false.

Substitution A substitution $\theta = \{V_1 \leftarrow t_1, ..., V_n \leftarrow t_n\}$ is an assignment of terms $t_1, ..., t_n$ to variables $V_1, ..., V_n$.

Applying a substitution The formula $F\theta$ where F is a term, atom, literal or expression, and $\theta = \{V_1 \leftarrow t_1, ..., V_n \leftarrow t_n\}$ is a substitution is the formula obtained by simultaneously replacing all variables $V_1, ..., V_n$ in F by the terms $t_1, ..., t_n$.

By now, we can define truth and falsity of an expression in a Herbrand interpretation.

Literal A ground literal l is true in an interpretation I if and only if l is a positive literal and $l \in I$, or l is a negative literal and $l \notin I$.

[2] For the purposes of this paper, we will only employ Herbrand interpretations, this in order to keep the exposition simple.

UCNF A UCNF expression

$$(\forall V_{1,1},...,V_{1,v_1} : l_{1,1} \vee ... \vee l_{1,n_1})$$
$$\wedge ... \wedge$$
$$(\forall V_{k,1},...,V_{1,v_k} : l_{k,1} \vee ... \vee l_{k,n_k})$$

is true in an interpretation if and only if for all i and for all substitutions θ such that $(l_{i,1} \vee ... \vee l_{i,n_i})\theta$ is ground, at least one of the $l_{i,j}\theta$ is true in I.

EDNF A EDNF expression

$$(\exists V_{1,1},...,V_{1,v_1} : l_{1,1} \wedge ... \wedge l_{1,n_1})$$
$$\vee ... \vee$$
$$(\exists V_{k,1},...,V_{1,v_k} : l_{k,1} \wedge ... \wedge l_{k,n_k})$$

is true in an interpretation I if and only if there exists i and an substitution θ such that $(l_{i,1} \wedge ... \wedge l_{i,n_i})\theta$ is ground, and all $l_{i,j}$ are true in I.

If an expression is true in an interpretation we also say that the interpretation is a model for the expression.

Let us first illustrate this rather complicated definition. It states e.g. $flies \vee \neg bird \vee \neg abnormal$ is true in the interpretations $\{flies\}, \{abnormal\}$ but false in $\{bird, abnormal\}$. Similarly, it allows us to say that

$$\exists C, T : triangle(T) \wedge circle(C) \wedge in(C, T)$$

is true in $\{triangle(t1), circle(c1), in(c1, t1), large(c1), small(t1)\}$ and false in $\{triangle(t1), circle(c1)\}$. Furthermore, $\forall X : polygon(X) \vee \neg square(X)$ is true in $\{square(s1), polygon(s1)\}$ and in $\{circle(c1)\}$ but false in $\{square(s1)\}$.

Model theory is the basis for reasoning about the declarative meaning of logical formulae. It is also used to define logical entailment.

Logical entailment An expression F_1 logically entails an expression F_2 if and only if all models of F_1 are also models of F_2. We will write $F_1 \models F_2$.

Logical entailment is typically verified by theorem provers using the well-known resolution principle, the propositional version of which is defined below.

Resolution - propositional Let $C_1 = l \vee l_1 \vee ... \vee l_n$, $C_2 = l' \vee l'_1 \vee ... \vee l'_m$, C is a resolvent of clauses C_1 and C_2 (denoted by $C \in res(C_1, C_2)$) if and only if $C = l_1 \vee ... \vee l_n \vee l'_1 \vee ... \vee l'_m$ and $l = \neg l'$.

Soundness of resolution For all $C \in res(C_1, C_2)$: $C_1 \wedge C_2 \models C$.

For example, $flies \leftarrow bird \wedge sparrow$ is a resolvent of $flies \leftarrow bird \wedge normal$ and $normal \leftarrow sparrow$.

Inductive Learning

The general definition of *discriminant* inductive concept-learning is the following:

Given:

- a language of hypotheses L_H, which defines the set of a priori acceptable hypotheses
- a language of examples L_e, which defines the set of a priori acceptable examples
- the *covers* relation between L_H and L_e, which defines when an example is covered by a hypothesis, i.e. $covers(H, e)$ is true if and only if the example e is covered by the hypothesis H
- sets of positive and negative examples P and N, expressed in the L_e,

Find: a hypotheses $H \in L_H$ which is

- complete, i.e. covers all positive examples in P
- consistent, i.e. does not cover any of the negative examples in N.

Discriminant learning starts from two classes of examples; its aim is to derive a hypothesis that is able to discriminate the examples as belonging to one of the two classes. In contrast, *characteristic* learning starts from a single class of examples and aims at a maximally specific hypothesis that covers all of the examples in the class. For readability, we formally define this notion as well:

Given:

- a language of hypotheses L_H, which defines the set of a priori acceptable hypotheses
- a language of examples L_e, which defines the set of a priori acceptable examples
- the *covers* relation between L_H and L_e, which defines when an example is covered by a hypothesis, i.e. $covers(H, e)$ is true if and only if the example e is covered by the hypothesis H
- a set of unclassified examples E, expressed in the L_e,

Find: a hypothesis $H \in L_H$ which is

- complete, i.e. covers all examples in E and
- maximally specific in L_H.

The maximally specific hypothesis is the most informative in the sense that it characterizes the examples best, and results in the smallest possible inductive leap.

At the end of this paper, I will argue that the difference between concept-learning and data mining is akin to the difference between discriminant and characteristic induction.

The above two definitions are generally accepted and unify all approaches in concept-learning. This makes them the ideal starting point for a unifying logical framework for induction. As the concept-learning definitions have the representation as a generic parameter, we merely need to instantiate them in order obtain a logical framework.

Induction in Logic

To instantiate the concept-learning framework, we need to make choices for:

- the language of hypotheses, L_H,
- the language of examples, L_e, and
- the *covers* relation.

From the short review of logic, it follows that there are two different choices for the *covers* relation, i.e. entailment and truth in an interpretation. This determines the first dimension.

Learning from interpretations

In this framework the following choices are made: examples are (Herbrand) interpretations and a hypothesis covers an example if the hypothesis is true in the interpretation :

Hypothesis space The hypothesis space is either the set of all UCNF expressions or the set of all EDNF expressions, i.e. $\mathcal{L}_\mathcal{H}$ = UCNF or $\mathcal{L}_\mathcal{H}$ = EDNF.

Example space The example space is the set of all Herbrand interpretations, positive examples are interpretations in which the target concept is true, negative examples are interpretations in which the target concept is false.

Coverage A hypothesis H covers an example e if and only if H is true in e. So, $covers(H, e)$ is true if and only if H is true in e.

Notice that these notions apply to characteristic as well as discriminant concept-learning.

Historically speaking, learning from interpretations (or a slight variant of it) has been employed by researchers such as (Winston 1975; Vere 1975; Hayes-Roth and McDermott 1978; Ganascia and Kodratoff 1986), who worked on structural matching and learned EDNF expressions, (Valiant 1984; Haussler 1988; Natarajan 1991), who worked on computational learning theory within propositional logic (i.e. without using quantifiers), and (De Raedt and Džeroski 1994; De Raedt and Van Laer 1995), who upgraded Valiant's results on CNF to UCNF, and (Michalski 1983), who – in a sense – learns DNF expressions in this framework.

We now illustrate this framework, first using an example in UCNF, and then using an example from structural matching (EDNF).

Example 1 *An UCNF learning task Consider the following family descriptions:*

$$p_1 = \{human(jef), male(jef), female(an), human(an)\}$$
$$p_2 = \{human(paul), male(paul)\}$$
$$p_3 = \{human(mary), female(mary)\}$$
$$n_1 = \{female(tweety), bird(tweety)\}$$
$$n_2 = \{male(dracula), bat(dracula)\}$$

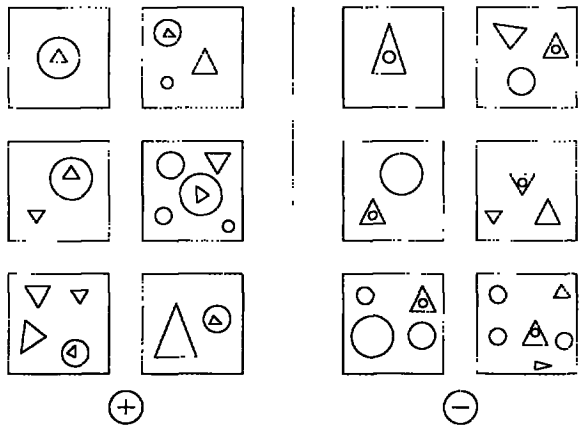

Figure 1: A Bongard Problem

In this example, the p_i are positive examples, the n_j negative ones. Furthermore, assume $\mathcal{L}_\mathcal{H}$ = UCNF. A possible solution for this concept-learning task is then:

$$H = (\forall X : human(X) \vee \neg male(X)) \wedge$$
$$(\forall X : human(X) \vee \neg female(X))$$

This learning problem could be handled by the recent ICL system of (De Raedt and Van Laer 1995).

Example 2 *An EDNF learning task Consider the Bongard problem shown in Figure 1. Some of the examples are represented as follows.*

$$p_1 = \{triangle(t1), circle(c1), in(t1, c1)\}$$
$$p_2 = \{circle(c1), circle(c2), triangle(t1), triangle(t2),$$
$$small(c1), large(c2), small(t1), large(t2), in(t1, c2)\}$$
$$\ldots$$
$$n_1 = \{triangle(t1), circle(c1), in(c1, t1)\}$$
$$n_2 = \{triangle(t1), triangle(t2), circle(c1), circle(c2),$$
$$in(c1, t2), small(c1)\}$$
$$\ldots$$

The target hypothesis can then be formulated as follows:

$$\exists T, C : triangle(T) \wedge circle(C) \wedge in(T, C)$$

This learning problem could be handled by techniques known as structural matching, cf. above.

Learning from entailment

An alternative logical formulation of concept-learning is used in inductive logic programming. In contrast to learning from interpretations, it employs the single-representation trick. In logical approaches to learning, this is often achieved by using the clausal logic subset of first order logic (i.e. UCNF[3].), which also forms the basis of logic programming and the programming language Prolog.

Hypothesis space The hypothesis space is the set of all clausal theories. In practice, one restricts it to definite clause theories.

Example space The example space is the set of all clauses; positive examples are logically entailed by the target hypothesis, negatives ones are not. In practice, one again restricts the example space to definite clauses.

Coverage A hypothesis H covers an example e if and only if H logically entails e. So, $covers(H, e)$ is true if and only if $H \models e$.

Note that the coverage relation in this framework is only semi-decidable. This may lead to various computational properties.

Learning from entailment is the main setting studied in the fashionable inductive logic programming paradigm. It is illustrated in Example 3.

Example 3 *Learning from entailment. Consider the following examples.*

$$p_1 = grandfather(leo, luc) \leftarrow father(leo, rose) \wedge mother(rose, luc)$$

$$p_2 = grandfather(rene, lieve) \leftarrow father(rene, william) \wedge father(william, lieve)$$

$$n_1 = grandfather(alice, luc) \leftarrow mother(alice, rose) \wedge mother(rose, luc)$$

$$n_2 = grandfather(X, X) \leftarrow$$

The following hypothesis is complete and consistent with regard to the positive examples p_1 and p_2 and the negative ones n_1 and n_2.

$$grandfather(X, Y) \leftarrow father(X, Z) \wedge parent(Z, Y)$$

$$parent(X, Y) \leftarrow mother(X, Y)$$

$$parent(X, Y) \leftarrow father(X, Y)$$

This example (or variants thereof) could be handled by inductive logic programming systems, such as e.g. (Muggleton 1995; Quinlan 1990).

[3]Instead of using UCNF, one might also want to use the EDNF representation. However, then it is more natural to reverse the coverage relation, i.e. to define $covers(H, e) = e \models H$. This has for instance been considered in structural matching, as studied by (Kodratoff 1988). However, we do not elaborate on this any further here.

Instead of learning from entailment, we might as well call this framework *open* because – in contrast to learning from interpretations which could be called *closed* – it does not assume complete knowledge about the examples. Indeed, in the first example it may be the case that *leo* is married to *alice* but this need not be specified. This difference can best be illustrated on a small example. Suppose there is a bird called tweety, and we know that tweety does not fly, is an ostrich, is black, but we do not know whether tweety is normal. In the closed framework, this cannot be represented because complete knowledge about tweety is assumed. There are two possible interpretations which are in agreement with our knowledge and this is not permitted[4]. Using clauses, the above example can naturally be represented as the negative example $flies(tweety) \leftarrow bird(tweety) \wedge ostrich(tweety) \wedge black(tweety)$. Moreover in learning from entailment it would be possible to include in the induced hypothesis the clause $abnormal(X) \leftarrow ostrich(X)$. Including this clause in the hypothesis would realize an inductive leap *on the example*, as using this clause, one can write the negative example as $flies(tweety) \leftarrow bird(tweety) \wedge ostrich(tweety) \wedge black(tweety) \wedge abnormal(tweety)$, making additional assumptions about tweety. This is impossible when learning from interpretations. This difference between the closed and open framework has great impact on the complexity of the induction problem. The reason is that in learning from entailment different clauses may make contradicting assumptions about particular examples. Therefore learning from entailment is computationally harder than learning from interpretations. In learning from interpretations, it is true that clauses $c_1 \wedge c_2$ cover an example if and only if clause c_1 and clause c_2 cover the example. In learning from entailment this need not be the case. For instance, $(flies \leftarrow bird \wedge abnormal) \wedge (abnormal \leftarrow ostrich)$ covers $flies \leftarrow bird \wedge ostrich$ whereas the individual clauses do not.

A further difference between the two frameworks concerns the use of implication. Indeed, as examples in learning from entailment are implications each example can be understood in causal terms. For instance, the properties of a bird cause (or imply) that the bird flies (or does not fly). As such learning from entailment promotes certain predicates as the target predicates, which are to be learned. In learning from interpretations all predicates are equal. Therefore, it is easy to represent problems of learning from intepretations as learning from entailment, but not the other way around. For instance, consider the Bongard problem of Figure 1. It can be modelled as a learning from entailment problem as follows :

[4]For completeness sake, we should mention that there exist in the literature few approaches and proposals to cope with such situations (see e.g. (Helft 1989)). However, as these have not received a lot of attention, we will not consider these here.

Example 4 *Reconsider Figure 1 and Example 2. The examples can be represented as follows:*

$$p_1 = \oplus \leftarrow triangle(t1) \wedge circle(c1) \wedge in(t1, c1)$$

$$p_2 = \oplus \leftarrow circle(c1) \wedge circle(c2) \wedge triangle(t1) \wedge triangle(t2) \wedge small(c1) \wedge large(c2) \wedge small(t1) \wedge large(t2) \wedge in(t1, c2)$$

$$\ldots$$

$$n_1 = \oplus \leftarrow triangle(t1) \wedge circle(c1) \wedge in(c1, t1)$$

$$n_2 = \oplus \leftarrow triangle(t1) \wedge triangle(t2) \wedge circle(c1) \wedge circle(c2) \wedge in(c1, t2) \wedge small(c1)$$

$$\ldots$$

Notice that the negative examples are represented also as clauses for the predicate $\oplus$. However, in contrast to the positive examples, negative examples are false and should not be entailed by the target theory. The target hypothesis can then be formulated as follows:

$$\oplus \leftarrow triangle(T) \wedge circle(C) \wedge in(T, C)$$

Implicit in learning from interpretations is that all facts not stated are false. To model this in learning from entailment one should also add negated atoms in the condition part of the examples. However, in the current state of the art - due to the use of definite clauses in inductive logic programming, this is not considered here.

Using background knowledge

This section may be skipped by the casual reader, less interested in technical aspects. Up till now we have completely ignored the use of background knowledge. Here, we show how the two frameworks can be adapted to take into account background knowledge. In both frameworks we will assume that background knowledge is specified as a definite clause theory B.

Learning from interpretations In learning from interpretations, background knowledge is used to expand the examples into an interpretation. This is usually done by taking the least Herbrand model of the background theory and example. The least Herbrand model of a definite theory consists of all ground facts constructed with the predicate, constant and functor symbols of the theory that are logically entailed by the theory. This can be formalized as follows:

Examples An example is a set of definite clauses. Often one will only use facts.

Coverage A hypothesis H covers an example e w.r.t. a definite clausal background theory B if and only if H is true in $M(B \wedge e)$.

Learning from entailment In learning from entailment, background knowledge can easily be integrated in the induction process as follows.

Coverage A hypothesis H covers an example e w.r.t. background theory B if and only if $B \wedge H \models e$.

Deduction and induction

Now that we have seen how the problem of concept-learning can be formulated in terms of logic, the question arises as how well-known concept-learning techniques and algorithms can be adapted to use logic as their representation language. This question will be answered by first studying the generality relation for the two frameworks, which will then motivate the introduction of logical notions of induction.

Generality and Entailment

It is well-known in the literature (cf. (Mitchell 1982; Michalski 1983; De Raedt and Bruynooghe 1992), that the generality relation is crucial for developing concept-learning algorithms. The generality relation is typically defined in terms of coverage. Indeed, one concept is more general than another concept if the first concept covers all examples that are covered by the second concept. Reformulating this in terms of learning interpretations leads to:

Generality - learning from interpretations
A hypothesis G is more general than a hypothesis S if and only if $S \models G$.

Because of the definition of generality in terms of coverage and the definition of logical entailment, generality and logical entailment coincide when learning from interpretations.

This notion of generality also means that the most specific statement, one can make is $\square$, the inconsistent theory which does not have any model, and the most general statement is the empty hypothesis, for which all interpretations are a model.

Let us now investigate whether this notion of generality also applies to learning from entailment.

It turns out that the generality relation in learning from entailment is the inverse as in learning from interpretations.

Generality - learning from entailment
A hypothesis G is more general than a hypothesis S if and only if $G \models S$.

This property follows from the transitivity of logical entailment. Again we see that – as in learning from interpretations – generality and logical entailment coincide. However, this time entailment is reversed. This means that the most general hypothesis is $\square$, the inconsistent theory, because it logically entails everything. The most specific hypothesis then is the empty theory. Notice also that it is this view of generality that machine learning research has typically adopted (due to its use in the inductive logic programming paradigm).

	generalisation	specialisation
interpretations	deduction	induction
entailment	induction	deduction

Induction and deduction

Because the logical framework employed determines whether logical entailment and generality coincide or whether the inverse of logical entailment and generality coincide, it will turn out useful to abstract away from this using deduction and induction:

Deduction - induction equation Let F_1 and F_2 be two logical formulae. If $F_1 \models F_2$ we say that F_2 follows deductively from F_1 or equivalently that F_1 follows inductively from F_2.

As we have seen in the previous section, generalization corresponds to induction when learning from entailment and to deduction when learning from interpretations. Dually, specialisation corresponds to deduction when learning from entailment and to induction otherwise. For instance, let $F_1 =$ *flies* $\leftarrow$ *bird*, $F_2 =$ *flies* $\leftarrow$ *bird* $\wedge$ *normal*. Then $F_1 \models F_2$ and F_1 is a generalization of F_2 when learning from entailment and a specialisation when learning from interpretations. These facts are summarized in the above Table.

The reader may notice that in the induction-deduction equation we view induction as the inverse of deduction. Whereas this viewpoint may appear to be controversial, especially for cognitive scientists and philoshophers, it will prove to be a very operational framework for concept-learning operators.

The question now is how to exploit these dual notions in order to obtain generalisation and specialisation operators for use in concept-learning algorithms such as those presented by (Mitchell 1982; De Raedt and Bruynooghe 1992). As the meaning of deduction (and induction) with regard to the generality relation depends on which logical framework is used, we will talk about deductive and inductive operators rather than about generalisation and specialisation operators. The equation also shows that one deductive operator can be used to perform both generalisation as well as specialisation, depending on the framework. This should clarify terms such as inductive specialisation and deductive generalisation.

Deductive operator A deductive operator ρ maps a formula F_1 onto a set of formulae $\rho(F_1)$ such that for all $F_2 \in \rho(F_1) : F_1 \models F_2$.

Inductive operator An inductive operator ρ maps a formula F_1 onto a set of formulae $\rho(F_1)$ such that for all $F_2 \in \rho(F_1) : F_2 \models F_1$.

When looking at these definitions, it should be clear that deductive operators are well-known in the literature on theorem proving. Many different deductive operators have been studied; usually they are denoted by the symbol $\vdash$ as they implement $\models$ in one way or another. As there exist many deductive operators, and deduction is just the opposite of induction, one can obtain inductive operators by inverting deductive ones. Furthermore, as some of these deductive operators restrict hypotheses in one way or another, the inductive ones will have to work within the same restrictions. Restrictions that are often applied are restrictions on clauses being definite, and also on the number of clauses in the hypotheses.

Using this idea, one can clarify the three main different notions of entailment (i.e. of $\vdash$) used as the basis of operators in inductive logic programming (Muggleton and De Raedt 1994). These are θ-subsumption (Plotkin 1970) which works on single clauses, (inverse) implication (Lapointe and Matwin 1992; Muggleton 1994; Idestam-Almquist 1993) which works on single clauses and inverse resolution (Muggleton and Buntine 1988) which works on sets of (definite) clauses.

θ-subsumption Let c and c' be two clauses. Clause c *θ-subsumes* c' (i.e. $c \vdash c'$) if there exists a substitution θ, such that $c\theta \subseteq c'$[5].

Resolution cf. above.

Implication Let c and c' be two clauses. Then c implies c' (i.e. $c \vdash c'$) if and only if $c \models c'$.

These different options for $\vdash$ result in different deductive and inductive operators used within inductive logic programming. For more information, we refer to (Muggleton and De Raedt 1994).

UCNF and EDNF

We now show that UCNF and EDNF are dual representations for concept-learning. The duality can be exploited at the level of operators, and sometimes as well at the level of algorithms.

UCNF and EDNF operators

To show the duality at the first level, consider a deductive operator ρ for UCNF. We will now show that this operator also can be used as an inductive operator for EDNF.

Indeed, let F_1, F_2 be two UCNF formulae such that $F_2 \in \rho(F_1)$, i.e. $F_1 \models F_2$. Because of the properties of logical entailment this is equivalent to $\neg F_2 \models \neg F_1$. Let

$$F_1 = (\forall l_{1,1} \vee ... \vee l_{1,n_1}) \wedge ... \wedge (\forall l_{r,1} \vee ... \vee l_{r,n_r})$$

$$F_2 = (\forall k_{1,1} \vee ... \vee k_{1,n'_1}) \wedge ... \wedge (\forall k_{r',1} \vee ... \vee k_{r',n'_{r'}}).$$

$$\text{Then } \neg F_1 = \neg\{(\forall l_{1,1} \vee ... \vee l_{1,n_1}) \wedge ... \wedge (\forall l_{r,1} \vee ... \vee l_{r,n_r})\} =$$

$$\neg(\forall l_{1,1} \vee ... \vee l_{1,n_1}) \vee ... \vee \neg(\forall l_{r,1} \vee ... \vee l_{r,n_r}) =$$

$$(\exists \neg l_{1,1} \wedge ... \wedge \neg l_{1,n_1}) \vee ... \vee (\exists \neg l_{r,1} \wedge ... \wedge \neg l_{r,n_r}).$$

[5] In this definition of θ-subsumption, we consider a clause $h_1 \vee ... \vee h_n \leftarrow b_1 \wedge ... \wedge b_m$ as the set of literals $\{h_1, ..., h_n, \neg b_1, ..., \neg b_m\}$.

One can apply the same steps for F_2. This results in :

$$(\forall l_{1,1} \vee ... \vee l_{1,n_1}) \wedge ... \wedge (\forall l_{r,1} \vee ... \vee l_{r,n_r}) \models$$

$$(\forall k_{1,1} \vee ... \vee k_{1,n'_1}) \wedge ... \wedge (\forall k_{r',1} \vee ... \vee k_{r',n'_{r'}})$$

if and only if

$$(\exists \neg k_{1,1} \wedge ... \wedge \neg k_{1,n'_1}) \vee ... \vee (\exists \neg k_{r',1} \wedge ... \wedge \neg k_{r',n'_{r'}}) \models$$

$$(\exists \neg l_{1,1} \wedge ... \wedge \neg l_{1,n_1}) \vee ... \vee (\exists \neg l_{r,1} \wedge ... \wedge \neg l_{r,n_r})$$

For instance,

$$(\forall X : flies(X) \vee \neg normal(X)) \wedge$$

$$(\forall Y : normal(Y) \vee \neg sparrow(X)) \models$$

$$\forall Z : flies(Z) \vee \neg sparrow(Z).$$

Therefore

$$\exists Z : \neg flies(Z) \wedge sparrow(Z) \models$$

$$(\exists X : \neg flies(X) \wedge normal(X)) \vee$$

$$(\exists Y : \neg normal(Y) \wedge sparrow(X))$$

What does this mean ? It means that any deductive operator on UCNF can be mapped into an inductive operator on EDNF (and dually an inductive UCNF on a deductive EDNF, or also dually an inductive EDNF on a deductive UCNF, ...) as follows. First map the EDNF formula onto an UCNF by negating all literals, changing all $\wedge$ into $\vee$, all $\vee$ into $\wedge$, and all $\forall$ into $\exists$, then apply the operator onto the obtained UCNF formula, and then map the resulting UCNF formulae back to EDNF by again negating all literals, changing all $\wedge$ into $\vee$, all $\vee$ into $\wedge$, and all $\forall$ into $\exists$. Formally speaking, this yields:

Mapping UCNF operator on EDNF Let ρ be a deductive (resp. inductive) operator on UCNF. Then ρ^d is an inductive (resp. deductive) operator on EDNF where $\rho^d = f^{-1}\ o\ \rho\ o\ f$, where $f((\exists k_{1,1} \wedge ... \wedge k_{1,n'_1}) \vee ... \vee (\exists k_{r',1} \wedge ... \wedge k_{r',n'_{r'}})) = (\forall \neg l_{1,1} \vee ... \vee \neg l_{1,n_1}) \wedge ... \wedge (\forall \neg l_{r,1} \vee ... \vee \neg l_{r,n_r})$

The definition of the mapping from EDNF to UCNF is left to the reader as an exercise.

Let us illustrate this on an example. Suppose we want to apply induction on the formula

$$(\exists X : \neg flies(X) \wedge normal(X)) \vee$$

$$(\exists Y : \neg normal(Y) \wedge sparrow(X))$$

Then we first use the mapping f, which yields:

$$(\forall X : flies(X) \vee \neg normal(X)) \wedge$$

$$(\forall Y : normal(Y) \vee \neg sparrow(X))$$

Then we apply a deductive operator (in this case resolution) on the UCNF formula, which yields :

$$\forall Z : flies(Z) \vee \neg sparrow(Z)$$

Finally, applying f^{-1} then results in the desired formula

$$\exists Z : \neg flies(Z) \wedge sparrow(Z).$$

Because of this duality between UCNF and EDNF, it suffices to analyse operators for UCNF. Transformers for EDNF can then be obtained by the above mappings. This shows that the operators used within the field of inductive logic programming directly apply to structural matching as well. Using this technique, (De Raedt *et al.* 1996) have modified Plotkin's well-known least general generalization and relative least general generalization operators for use in structural matching. The resulting operator solves some of the problems with existing ones for structural matching.

UCNF and EDNF algorithms

There exists also a more direct mapping between UCNF and EDNF, which is well-known in computational learning theory (cf. (Haussler 1988) and which makes that one should only develop one type of algorithm for *learning from interpretations*. The property is:

$$(\exists l_{1,1} \wedge ... \wedge l_{1,n_1}) \vee ... \vee (\exists l_{k,1} \wedge ... \wedge l_{k,n_k})$$

is a solution to an EDNF concept-learning task with as positives P and as negatives N if and only if

$$(\forall \neg l_{1,1} \vee ... \vee \neg l_{1,n_1}) \wedge ... \wedge (\forall \neg l_{k,1} \vee ... \vee \neg l_{k,n_k})$$

is a solution to the UCNF concept-learning task with as positives N and as negatives P.

This property holds because e is covered by H if and only if e is not covered by $\neg H$, and the negation of a UCNF is an EDNF, and vice versa.

The main implication of this property is that a EDNF learner can be used to learn UCNF, and vice versa. One only has to switch to a dual problem formulation where the positives and the negatives are inverted, and the result is negated.

This has implications e.g. for the study of (Mooney 1995) who developed two algorithms, one for learning CNF and one for DNF, as well as for appraoches to structural matching and inductive logic programming, which can be used to address the opposite task. E.g. the ICL system of (De Raedt and Van Laer 1995) can easily be used to address structural matching (and EDNF).

Data Mining and Concept-Learning

Data mining and knowledge discovery (Mannila 1995; Fayyad *et al.* 1995) have recently enjoyed a lot of attention. It has its roots in machine learning, data bases and statistics. Nevertheless, the relation to classical concept-learning techniques is often unclear, as the aim in data mining is to discover regularities (often rules or clauses) that are valid in the data, rather than develop hypotheses with the aim of classification.

Using the logical tools developed above, it is possible to analyse and relate both problems from a logical perspective. In this analysis, we will focuss on the UCNF

or clausal representation mostly employed in data mining.

We start by looking at the specification of the problem as recently formulated by Heikki Mannila (Mannila 1995). He views data mining as the process of a constructing a theory $Th(L_S, r, q)$, where L_S is a set of sentences to consider, r the data(base), and q the quality criterion. The aim then is to find all sentences ϕ in the language L_S that satisfy the quality criterion w.r.t. the data r, i.e.

$$Th(L_S, r, q) = \{\phi \in L_S \mid q(r, \phi) \textit{ is true.}\}$$

In the practice of data mining, Th typically consists of a set of rules or clauses, i.e. Th is in a kind of $UCNF$ format, and each ϕ in L_S thus corresponds to a single clause. Furthermore, the aim is to find all possible $\phi \in L_S$ that are valid. Also, validity roughly corresponds to certain relaxations of ϕ being true in r.

Under these conditions, the problem of data mining can roughly be formulated as that of characteristic learning of UCNF expressions from interpretations. Indeed, let r contain a set of unclassified interpretations (positive examples only), let L_S contain all clauses allowed in UCNF expressions (when characteristic learning), and let $q(r, \phi) = covers(\phi, r) = r$ is a model for ϕ. The conjunction of the clauses in $Th(L_S, r, q)$ then denotes the solution to the characteristic concept-learning task, i.e. the maximally specific hypothesis covering all positives. From this it follows that, the main difference between data mining and characteristic learning of interpretations in UCNF form lies in the relaxation of the quality-criterion, which typically requires rules to be approximately true (according to user-specified criteria) instead of completely true on the data.

This view of data mining has been exploited in the Clausal Discovery Engine of (De Raedt and Bruynooghe 1993; De Raedt and Dehaspe 1995), which addresses characteristic learning of interpretations using UCNF representations. It is probably the first data mining system working with full clausal representations.

Related Work and Conclusions

Various researchers have addressed similar issues as I have. For instance, (Michalski 1983; 1994) has studied induction, deduction, generalization and specialisation. My contribution here is to relate this discussion to the two standard notions of coverage employed in machine learning, and to observe that generalization and entailment always coincide (in one direction or another). Following (Niblett 1988), various logical notions of generalization have been analyzed and their use has been outlined. The relation between UCNF and EDNF is based on work by (Haussler 1988), though it was upgraded here to first order logic. Also, learning from entailment and learning from interpretations has been studied in computational learning theory, where their relation has been (partly) studied for propositional logic (Angluin *et al.* 1992; Frazier and Pitt 1993). More novel aspects of the current work include: the relation between data mining and characteristic concept-learning, the transformation of operators from UCNF to EDNF, the relation between inductive logic programming and structural matching, and the integration of these aspects. There are also several remaining questions such as: what is the relation between learning from entailment and learning from interpretations, and what is the relation between UCNF and EDNF (when learning from entailment) ?

Acknowledgements

Luc De Raedt is supported by the Belgian National Fund for Scientific Research, and by the ESPRIT IV project no. 20237 on Inductive Logic Programming II (ILP^2). He is grateful to Maurice Bruynooghe, Peter Flach, Wim Van Laer, Hendrik Blockeel, Luc Dehaspe, and especially to Nada Lavrac for many inspiring discussions.

References

D. Angluin, M. Frazier, and L. Pitt. Learning conjunctions of horn clauses. *Machine Learning*, 9:147–162, 1992.

L. De Raedt and M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.

L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann, 1993.

L. De Raedt and L. Dehaspe. Clausal discovery. Technical Report KUL-CW, Department of Computer Science, Katholieke Universiteit Leuven, 1995. submitted.

L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

L. De Raedt and N. Lavrač. The many faces of inductive logic programming. In J. Komorowski, editor, *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1993. invited paper.

L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1995.

L. De Raedt, P. Idestam-Almquist, and G. Sablon. θ-subsumption for structural matching. Technical Report KUL-CW, Department of Computer Science, Katholieke Universiteit Leuven, 1996. draft.

L. De Raedt, editor. *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 1996.

U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. The MIT Press, 1995.

M. Frazier and L. Pitt. Learning from entailment. In *Proceedings of the 9th International Conference on Machine Learning*, 1993.

J.G. Ganascia and Y. Kodratoff. Improving the generalization step in learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 2, pages 215–241. Morgan Kaufmann, 1986.

D. Haussler. Quantifying inductive bias : AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177 – 221, 1988.

F. Hayes-Roth and J. McDermott. An interference matching technique for inducing abstractions. *Communications of the ACM*, 21:401–410, 1978.

N. Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 149–156. Morgan Kaufmann, 1989.

P. Idestam-Almquist. Generalisation under implication using or-introduction. In *Proceedings of the 6th European Conference on Machine Learning*, volume 667, pages 56–64. Lecture Notes in Artificial Intelligence, 1993.

Y. Kodratoff. *Introduction to Machine Learning*. Pitman, 1988.

S. Lapointe and S. Matwin. Sub-unification: a tool for efficient induction of recursive programs. In *Proceedings of the 9th International Workshop on Machine Learning*. Morgan Kaufmann, 1992.

H. Mannila. Aspects of data mining. In Y. Kodratoff, G. Nakhaeizadeh, and G. Taylor, editors, *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 1–6, Heraklion, Crete, Greece, 1995.

R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.

R.S. Michalski. Inferential theory of learning: developping foundations for multistrategy learning. In R.S. Michalski and G. Tecuci, editors, *Machine Learning: A Multistrategy Approach*, volume 4, pages 3–61. Morgan Kaufmann, 1994.

T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

R.J. Mooney. Encouraging experimental results on learning cnf. *Machine Learning*, 19:79–92, 1995.

S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.

S. Muggleton and L. De Raedt. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, 19,20:629–679, 1994.

S. Muggleton, editor. *Inductive Logic Programming*. Academic Press, 1992.

S. Muggleton. Inverting implication. *Artificial Intelligence*, 1994. To appear.

S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13, 1995.

B.K Natarajan. *Machine Learning : A Theoretical Approach*. Morgan Kaufmann, 1991.

T. Niblett. A study of generalisation in logic programs. In D. Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning*, pages 131–138. Pitman, 1988.

G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

S.A. Vere. Induction of concepts in the predicate calculus. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 282–287. Morgan Kaufmann, 1975.

C. Vrain. Ogust: A system that learns using domain properties expressed as theorems. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning: an artificial intelligence approach*, volume 3, pages 360–381. Morgan Kaufmann, 1990.

P.H. Winston. Learning structural descriptions from examples. In P.H. Winston, editor, *Psychology of Computer Vision*. The MIT Press, 1975.