Inductive Logic Programming: Theory and Methods

Daniel Schäfer

14 March, 2018

Abstract

This paper provides an overview of the topic Inductiv Logic Programming (ILP). It contains an introduction to define a frame of the topic in general and will furthermore provide a deeper insight into the implementations of two ILP-algorithms, the First-Order Inductive Learner (FOIL) and the Inverse Resolution. ILP belongs to the group of classifiers, but also differs from them, because it uses the first-order logic. The paper explains what ILP is, how it works and some approaches related to it. Additionally an example shall provide a deeper understanding and show ILPs ability for recursion.

1 Introduction

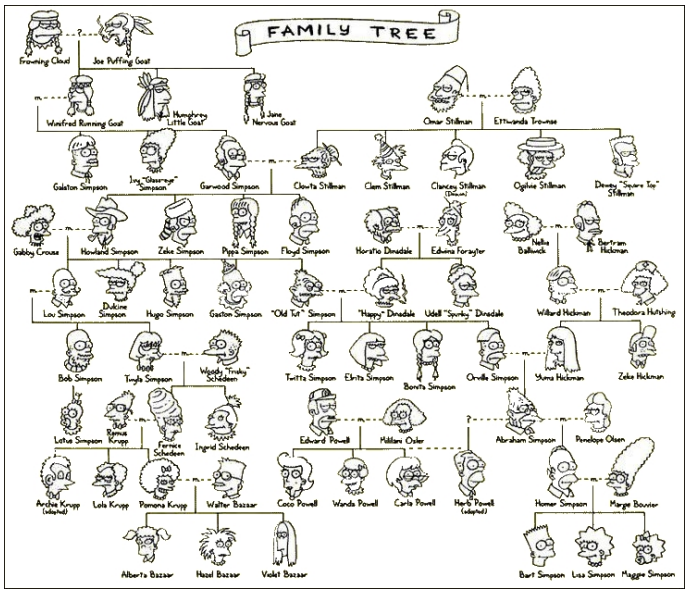

Figure 1: Familytree show a basic concept of relationships between humans, that even children understand, but for machines it is hard to learn these bounds[1]

[1]Imagesource: http://the-simpson-chow.skyrock.com

A lot of challenges today can be solved by computers. One challenge is the classification of data into classes and sub-classes. These challenges can be solved by algorithms, that need a database as background knowledge, to identify the categories and are then able to classify to which of these categories a new observation belongs. A machine learning algorithm, who's task is to map input data to a category is known as *classifier*. A major problem to these challenges so far is, how to teach the machine the general setting and the background knowledge it needs to understand this problem and especially the relationship between the data. Ideally in a way, that it is also human readable and adaptable on multiple settings.

Many approaches exist, that pass a set of data to an algorithm and then generate a classification of this input-data. Normally this data is represented as set of *key-value-pairs* and one or multiple target-predicates, that should be fulfilled by this input. The result of these classifiers is a set of categories and the parameters that describe these categories. A new observation is then compared to the parameters of the category to identify, to which it belongs. For example with a *Support Vector Machine (SVM)* a vector is created that divides the data in two sets. One that satisfies the target-predicate and one that does not. A new observation can then be identified to which category it belongs, where an observation is a fixed set of quantifiable properties called explanatory variables or features. Other such classifiers besides SVM are:

Overview 1.1 *Other Classifiers*

- ***Linear Classifiers:*** *linear combination of the feature values of an object. Often used for document classification[10].*
- ***Quadratic Classifiers:*** *is a more general version of the Linear Classifier, used for classification of measurement values of two or more classes of objects or events by a quadric surface[3].*
- ***K-Nearest Neighbours:*** *classification by majority-vote of an objects direct neighbors, depending on similarity of the feature-values[8].*
- ***Neural Networks:*** *inspired by biological neural networks, a network of nodes is learned based on examples and continually improved. Classifies by considering for each layer of the network, where the input belongs to[16].*
- ***Decision Trees:*** *for example Random Forest is a classification approach, where multiple decision trees are created during training time, that enable to classify new observations[19].*

ILP is a form of supervised machine learning which uses logic programming (typically *Prolog*) as a uniform representation for background knowledge, examples, and induced theories. This basically means that the inputs and outputs to ILP systems are logic programs, in contrast to most other forms of machine learning. For example in SVM, where typically the inputs and outputs are vectors of real numbers. By using logic programming as a uniform representation, ILP has three main advantages over other classifiers known in machine learning, which can be seen in definition 1.2.

Definition 1.2 *ILP as classifier[13]*

- ***Expressability:*** *ILP systems can, in theory, learn anything which is computable*
- ***Background knowledge:*** *Can include additional information in the learning problem*
- ***Human readable theories:*** *Because the output is a logic program, it is more intelligible than a vector of real numbers.*

Especially for machine learning approaches, one of the big advantages of ILP appears, when trying to model relationships. For example the relations between family members in a big family tree like the one in figure 1. While the direct connections might be really easy to display and also to learn for most algorithms (a simple relation can be seen in the examples 2.2 and 2.3), complex connections would result in long statements and confusing outputs. ILP shows further strengths, when it comes to showing recursive relations of data.

Section 2 provides a brief introduction to ILP. After that, background information about model theory is provided in section 3, which is necessary to create expressions and to be able to provide interpretations of them. In section 4, a generic algorithm is presented, defining the frame conditions for an ILP algorithm. The section Proof Theory of Inductive Logic Programming shows possibilities to control the generated logic expressions, inverse resolution is shown in more detail. Section 6 presents the FOIL algorithm and explains it using the initial example from figure 1. Finally a conclusion completes the paper.

2 Inductive Logic Programming

To understand ILP, this section first explains the difference between induction and deduction and then describes the basic frame of the classification with the ILP-approach.

2.1 Inductive vs. Deductive Reasoning

Reasoning by inductive inference is the way to see the everyday world. The term inference can be described as 'logical reasoning'. From the knowledge that all birds observed can fly, a person can easily conclude that all birds that exist fly. This is a false statement but it holds on the given premises. Inductive inference allows for the conclusion to be false and therefore does not follow logically from the statements.

The deductive inference specifies premises by the valuations of sentences in truth values, and provides an inference procedure which is considered to lead to certain conclusion on the base of given premises. The inductive inference specifies premises by the valuations of sentences in possible (plausible) values and provides an inference procedure which is considered to lead to most possible (most plausible) conclusions on the base of given premises [9]. This stands in contrast to deductive reasoning, where by definition it always results in a valid conclusions, which must be true given that their premises are true. The most famous example for deductive reasoning is the following:

Example 2.1 *Deductive Reasoning:*
All men are mortal, Socrates is a man, Socrates is mortal.

The two premises are true and therefore the conclusion is true as well. Otherwise Sokratis would either not be a man or there are men that are not mortal. The way of deductive reasoning is also described as going from general to specific, general knowledge is used to describe a specific condition, in case of example 2.1 the mortality of Socrates.

2.2 Inductiv Logic Programming in General

The term Inductiv Logic Programming (ILP) as defined by Stephen Muggleton[11] describes an intersection between *inductive learning* and *logic programming* and includes techniques from classical *machine learning* and from *logic programming*. From machine learning it takes the idea of generating programs that are able to generate hypotheses from a given set of examples (also called observations)[13]. This process is fundamental in enabling artificial intelligence to generate new knowledge and to generalize rules from an observed environment to an unobserved space.

This approach combined with the representation of logic programming, as known for example from *Prolog*, is able to overcome problems of other Machine Learning approaches[14]. Quinlan describes these two limitations as (1) the limited knowledge representation formalism and (2) the difficulties, which appear when using background-knowledge during machine learning.

The first point describes the problem that many situations, that artificial intelligence wants to learn need a representation in first order logic. While it is possible to represent facts with propositional logic, it comes to its limits when it comes to objects or relations. For example the relation *„When I paint a wall green, it becomes green.“* would have to be stated explicitly for every wall. It is not possible to make a general clause that describes this situation for every wall.

The second problem is, that it can be difficult to use substantial background knowledge in the learning process. But researches have shown, that it is crucial to use these domain specific knowledge to achieve intelligent behaviour.

Machine learning is nearly always inductive. The idea is to generate and validate rules based on a given example-set. This is called a specific-to-general approach. The examples define and explain a small part of knowledge, if this set is big enough it can be concluded that the rules are applicable also on unseen situations. Mitchell describes inductive machine learning like this:

> *„Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.“*
>
> *Mitchell*[10]

The representation of logic programming, as known for example from programming in *Prolog*, enables the programs to show and validate relations between objects. This so called First-Order framework or First-Order Logic (FOL) is a human- and machine-readable knowledge representation. Instead of key-value pairs, where a key is used as unique identifier of a value, a complex relation

can be stated. Most other machine learning techniques are only able to display one relation, for example decision trees or neural networks. A simple example for the statement *Homer is the father of Bart* can be seen in the examples 2.2 and 2.3, where the first example is a Key-Value pair and the second describes the statement in FOL.

Example 2.2 *Key-Value Pair:*
FatherOfBart = "Homer"

Example 2.3 *First-Order logic:*
father(Homer, Bart)

The advantage is, that the relation shown is universally usable, by replacing a value of the relation. In the given example „Bart“could be replaced by „Lisa“, which would again be a correct literal. For the given key-value pair example, a new key would be needed, that describes this scenario. Beside this and the readability, another advantage of this approach is, that it is possible to replace values with variables. This might not be a big advantage, when looking at example 2.3, but the use can be seen when it comes to generalization of relations. This makes it possible to directly generalize a rule and even make it recursive. An example stating a general clause describing a father can be seen in the following example (2.4).

Example 2.4 *Generalizing a father clause:*
father(Homer, Bart) :- parent(Homer,Bart), male(Homer).
can be generalized to:
father(X,Y) :- parent(X,Y), male(X).

The representation of a single rule, as shown in example 2.3 is called a *literal*. The disjunction of two or more literals is called a clause (see example 2.4). Inductiv Logic Programming (ILP) learns a special form of clauses, called Horn clauses. They differ from normal clauses, because they only contain one positive literal and an arbitrarily amount of negative literals. The positive literal is called the head and the set of negative literals build the body of the clause.

3 Model-Theory of Inductive Logic Programming

For the logic used in inductive inference the logical elements (the semantics) involved in inductive inference will now be described, together with the relationships which should hold between them.

3.1 General

The term *Model Theory* describes the study of classes of mathematical structures. It is an area of mathematical logic. As such it investigates these mathematical structures in a perspective of mathematical logic. In this process the relation between formal expressions of a language and their meaning is examined. The formal expressions are on a *syntactical layer* while the meaning of them are on a *semantic layer*. A theory is a set of expressions in a formal language. A model of a theory is a structure or interpretation, that satisfies the

expressions of that theory. The goal of this approach (and most of mathematical logic) is concerned with the use of language to extract information about the objects under investigation.

Like Wilfrid Hodges explains in his article about model-theory[7], if an interpretation I of a sentence S results in a valid result and therefore makes this sentence *true*, than I is a *model* of S. This can be noted as: $I \models S$.

Definition 3.1 *Model Theory*

- *Part of mathematical logic*
- ***Theory:*** *set of sentences*
- ***Model:*** *a structure/interpretation that satisfies these sentences*

The semantics can be distinguished between *normal* and *nonmonotonic* semantics, which will be looked upon now.

3.2 Normal Semantics

The normal semantics of logic is the study of the interpretations of formal and natural languages. These semantics usually try to capture the pre-theoretic notion of consequences. A problem in normal semantics consists of a given background knowledge B (sometimes called *prior* knowledge). Furthermore there is some evidence E given as well. This consists of two parts: the positive evidence E^+ and the negative evidence E^-. The goal is to find a Hypothesis H, which is equal to an interpretation, that satisfies the following conditions:

Definition 3.2 *Normal Semantics*

- ***Prior Satisfiability:*** $B \wedge E^- \not\models \square$ *→It is necessary that for the background knowledge B and the negative evidence E^- always follow that the result is not true.*
- ***Posterior Satisfiability:*** $B \wedge H \wedge E^- \not\models \square$ *→It is necessary that for the background knowledge B, the Hypothesis H and the negative evidence E^- always follows that the result is not true.*
- ***Prior Necessity:*** $B \not\models E^+$ *→The background knowledge should not already entail the positive evidence E^+.*
- ***Posterior Sufficiency:*** $B \wedge H \models E^+$ *→The background knowledge B and the hypothesis H have to entail the positive evidence E^+*

These conditions are the general setting for normal semantics. But in ILP systems the *definite semantics* are used, which is a special form of the normal semantics. The difference is, that the background knowledge and the hypothesis have to be definite clauses, which makes it simpler than the use of normal semantics, since any logical formula can be either true or false. This is because a definite clause theory follows the model theory of Herbrand, where two models satisfy the same sentences, or they both disagree on each other, e.g. one satisfies a certain part of that sentence, that the other does not [2]. This leads to the following conditions that need to be satisfied by an ILP system, considering the Herbrand model $M^+(T)$, where T is the theory, which here is the background knowledge B with or without a hypothesis B:

Definition 3.3 *Definite Semantics*

- ***Prior Satisfiability:*** *all* $e \in E^-$ *are false in* $M^+(B)$
- ***Posterior Satisfiability:*** *all* $e \in E^-$ *are false in* $M^+(B \wedge H)$
- ***Prior Necessety:*** *some* $e \in E^+$ *are false in* $M^+(B)$
- ***Posterior Sufficiency:*** *all* $e \in E^+$ *are true in* $M^+(B \wedge H)$

The presented Posterior Satisfiability criterion is also known as *consistency* with the negative evidence. Furthermore, the Posterior Sufficiency criterion is also referred to as *completeness*. The definite clause theory is also called the example setting, because the evidence is restricted to true and false examples. The example setting is thus the normal semantics with B and H as definite clauses and E as set of ground unit clauses. The example setting is the main setting of ILP employed by the large majority of ILP systems.

3.3 Non-monotonic Semantics

A non-monotonic semantic (also referred to as non-monotonic logic) is a formal logic where the conclusions drawn are not final, they can change over time through new information. This enables the reasoners to make conclusions on the current set of information and withdraw it, when further evidence proofs that the conclusion was false. Defeasible inferences can be found in everyday reasoning, expert reasoning (e.g. medical diagnosis) and also in scientific reasoning[21].

In non-monotonic semantics, the conditions differ slightly from normal semantics. The background knowledge B is still a set of definite clauses, but the evidence E is now empty, because it is part of the background knowledge. The hypotheses H are sets of general clauses. They use the same alphabet, meaning the same form of statements, as the background knowledge. Again the Herbrand model is used, which makes the non-monotonic semantics a closed world assumption[11]. These are the conditions, that have to hold in the non-monotonic setting:

Definition 3.4 *Conditions in Non-monotonic Semantics*

- ***Validity:*** *all* $h \in H$ *are true in* $M^+(B)$
- ***Completeness:*** *if general clause* g *is true in* $M^+(B)$ *then* $H \models g$
- ***Minimality:*** *there is no proper subset* G *of* H *which is valid and complete*

With defeasible reasoning, it is for example possible to say that *Tweety flies*, when the only known fact is that it is a bird and birds normally fly (see example 3.5. This is true in most of the cases. If a further evidence proofs, that *Tweety* is a penguin, the conclusion no longer holds and has to be adjusted or withdrawn.

3.4 Comparing Normal and Non-monotonic Semantics

With the additional bird *Oliver* and the positive evidence E^+ shown in example 3.5 it is possible to get a better understanding of the difference between normal and non-monotonic semantics.

Example 3.5 *Bird-example*

$$B_1 \begin{cases} bird(tweety) \\ bird(oliver) \end{cases}$$

$$E_1^+ = flies(tweety)$$

The difference becomes clearer now, when looking at the possible hypothesis H_1 for the example setting and comparing it to the hypothesis for the non-monotonic setting. In the example setting a valid hypothesis H_1 would be $flies(X) \leftarrow bird(X)$. This leads to an inductive leap where $flies(oliver)$ is also true in regard to the minimal Herbrand model $M^+(B_1 \wedge H_1)$. The inductive leap means, that due to the knowledge about one bird that flies, the conclusion is drawn that the other bird must fly too, even though there is no evidence about it so far.

Since in the non-monotonic setting, the evidence is always empty and part of the background knowledge, the previous hypothesis is not a solution. The Herbrand-model is the reason for this. The example setting is the normal semantics with B and H as definite clauses. The evidence E is a set of ground unit clauses. This semantics are used by the majority of if ILP systems and thus the main setting for ILP[20].

4 A Generic Inductive Logic Programming Algorithm

In this section a generic ILP algorithm is presented based on the description made from Muggleton and De Raedt [13]. The ILP is build on the **GEN**eric **CO**ncept-**L**earning algorithm (GENCOL) model concept from De Raedt und Bruynooghe[4] which was the first generic algorithm for empirical concept learning proposed in ILP[18].

The algorithm is seen as a search problem. There is a space of candidate solutions such as the set of „well formed“ hypotheses and an acceptance criterion characterizing solutions to an ILP problem. On one hand, to solve ILP the enumeration algorithm, which follows the artificial intelligence principles, can be used. However, this algorithm is computationally too expensive and this is the reason why it is not very practicable. On the other hand, in concept learning and ILP the search space is typically structured by means of the dual notions of generalization and specification. This allows pruning of the search. Generalization corresponds to induction and specialization to deduction. Induction is viewed here as the inverse of deduction (see chapter 2.1).

4.1 Generalization and Specialization Rules

The following three definitions give a frame to the rules that have to apply for generalization and specialization of hypotheses and for deductive and inductive inference rules.

Definition 4.1 *A hypothesis G is more general than a hypothesis S if and only if $G \models S$. S is also said to be more specific than G.*

Furthermore, in search algorithms the notions of generalization and specialization are incorporated using inductive and deductive inference rules:

Definition 4.2 *A deductive inference rule r in R maps a conjunction of clauses G onto a conjunction of clauses S such that $G \models S$. r is called a specialization rule*

An example of deductive inference rule is resolution; also dropping a clause from a hypothesis realizes specialization. In figure 2 a resolution is shown to proof that the statement r is true.

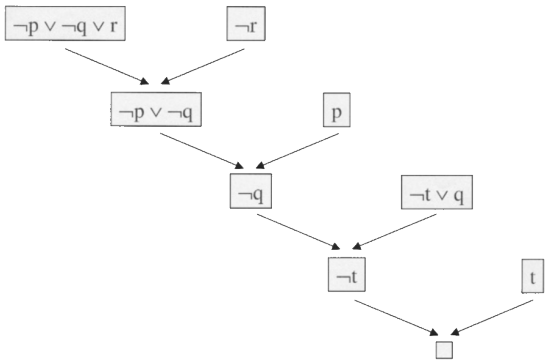

Figure 2: Resolution to proof $r = true$

Definition 4.3 *An inductive inference rule r in R maps a conjunction of clauses S onto a conjunction of clauses G such that $G \models S$. r is called a generalization rule.*

An example of an inductive inference rule is Absorption:

$$\textbf{Absorption:} \quad \frac{p \leftarrow A, B \qquad q \leftarrow A}{p \leftarrow q, B \qquad q \leftarrow A}$$

In the rule of Absorption the conclusion entails the condition. Applying the rule of Absorption in the reverse direction, i.e. applying resolution, is a deductive inference rule. Other inductive inference rules generalize by adding a clause to a hypothesis, or by dropping a negative literal from a clause.

4.2 Soundness of Inductive Inference Rules

Inductive inference rules, such as Absorption are not sound. This soundness problem can be circumvented by associating each hypothesized concussion H with a label $L = p(H|B \wedge E)$ where L is the probability that H holds given that the background knowledge B and evidence E hold. Assuming the subjective assignment of probabilities to be consistent, labelled rules of inductive inference are „as sound as“ deductive inference. The conclusions are simply claimed to hold in a certain proportion of interpretations.

4.3 Pruning the Search Space

Generalization and specialization form the basis for pruning the search space. The reasons are the followings:

- When $BH \not\models e$, where B is the background knowledge, H is the hypothesis and e is positive evidence, then none of the specializations H' of H will imply the evidence. Each such hypothesis will be assigned a probability label $p(H'|B \wedge E) = 0$. They can therefore be pruned from the search.
- When $B \wedge H \wedge e \models \square$, where B is the background theory, H is the hypothesis and e is negative evidence, then all generalizations H' of H will also be inconsistent with $B \wedge E$. These will again have $p(H'|B \wedge E) = 0$.

4.4 The Inductiv Logic Programming Algorithm

Using the above ideas of ILP as search, inference rules and the labeled hypotheses, a generic ILP system can be defined as the following:

$QH :=$ `Initialize`
repeat
  `Delete` H *from* QH
  `Choose` *the inference rules* $r_1, ..., r_k \in$ `R` *to be applied to* H
  Apply the rules $r_1, ..., r_k$ *to* H *to yield* $H_1, H_2, ..., H_n$
  Add $H_1, ..., H_n$ *to* QH
  `Prune` QH
until `stop-criterion`(QH) *satisfied*

The algorithms works as follows. It keeps track of a queue of candidate hypotheses QH. It repeatedly deletes a hypothesis H from the queue and expands that hypotheses using inference rules. The expanded hypotheses are then added to the queue of hypotheses QH, which may be pruned to discard unpromising hypotheses from further consideration. This loop continues until the stop criterion is satisfied.

- `Initialize` set of theories as hypotheses to start from in QH.
- R is the set of inference rules applied.
- `Delete` influences the search strategy. Using different instantiations of this procedure, one can realize a depth-first, breadth-first or best-first algorithm.
- Choose determines the inference rules to be applied on the hypothesis H.
- `Prune` determines which candidate hypotheses are deleted from the queue. This is usually realized using the labels(probabilities) of the hypotheses on QH or relying on the user. Combining ”delete” with ”prune” it is easy to obtain advanced search strategies such as hill-climbing, beam-search, best-first and other.

- The Stop-criterion states the conditions under which the algorithm stops. Some frequently employed criteria require that a solution is found, or that it is unlikely that an adequate hypothesis can be obtained from the current queue.

5 Proof Theory of Inductive Logic Programming

In contrast to model theory, which has its roots in the semantic mathematics, proof theory is of syntactic nature. This chapter provides a short overview over the different frameworks of inductive inference. It is a part of mathematical logic, where proofs are represented as mathematical objects. Previously was shown that induction is the inverse of deduction. An example for that on base of the formula $B \wedge H \models E^+$ would mean that deriving E^+ from $B \wedge H$ is deduction while an example for induction would be to conclude H on basis of B and E^+. This means it is possible to obtain inductive inference rules by inverting deductive ones [13].

5.1 General

The basics for proof theory were created by David Hilbert in the beginning of the 20th century. Hilbert stated the *proving of consistency of the arithmetic of the real numbers* as one of his 23 mathematical problems and later initiated a program that proofs consistency [22]. There exist four most common types of inductive inference *θ-Subsumption*, *Relative Subsumption*, *Inverted Resolution* and *Inverting Implication*. This paper provides a detailed view on the Inverted Resolution (see section 5.2), which provides the best results and is most widely used for proving clauses. Furthermore it also builds a basis for the programming language Prolog.

- **θ-subsumption:** Here the background knowledge is supposed to be empty. Furthermore the deductive inference rule corresponds to θ-subsumption among single clauses. Since θ-subsumption is decidable it is often used over inverting implication, but it has its flaws in regard to completeness of implication among clauses[5].
- **Inverting Implication:** contains approaches that extend the θ-subsumption. They aim on overcoming the incompleteness of this approach.
- **Relative Subsumption:** extensions of θ-subsumption, that take the background-knowledge into account, instead of leaving it empty[13].

5.2 Inverted Resolution

An inductive inference rule can always be viewed as the inverse of a deductive rule of inference. On the basis that deductive rule of resolution is complete, the inverse of a resolution is complete as well for induction. The inductive framework for inverse resolution was first published by Muggleton and Buntine 1988[12]. The principle of normal resolution, due to Robinson (1965), is a method of theorem proving that proceeds by constructing refutation proofs, i.e., proofs by contradiction[17].

This basis is used in inverted resolution to proof clauses of hypotheses by iteratively applying the resolution rule in a suitable way which allows to tell if a formula is satisfiable. This is done by negating the goal and verify that it is unsatisfiable, it can be proved that the goal follows logical from the given premises.

For example to prove the grandfather relation from the family-tree in figure 1 between *Bart* and *Abe* the following inverse resolution could be generated. The clause C written as first-order literal *grandfather(abe,bart)*[2] describes the investigated relation. For this the *Absorption*-operator is used. This operator together with the *Identification*-operator are also called the V-operators, because when drawn as a tree, these operations resemble a „V“. C_1 is a premise that has to hold for the rule to be valid. For example *Abe* has to be the father of *Homer*, in order to be the grandfather of *Bart*, resulting in the literal C_1 = *father(abe,homer)*. This can also be drawn as a tree as shown in figure 3.

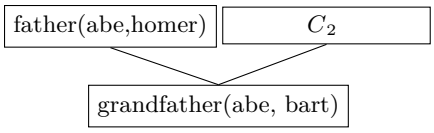

Figure 3: Setup of Inverse Resolution for the grandfather-relation

The next step is to find a literal L and generate clause C_2. The literal L should occur in C_1 but not in clause C. For the grandfather-relation this would result in replacing *Abe* by substitution with an *x*. Then the clause from the base is disjunctive connected with the negation of the other arm of the „V“. This results in the tree in figure 4.

$$C_2 = (C - (C_1 - L)) \cup \neg L$$

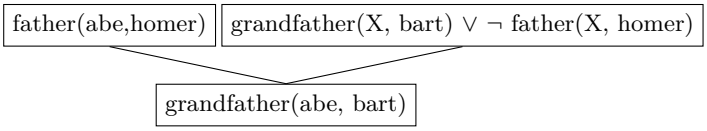

Figure 4: *abe* is substituted by X resulting in C_2, the conjunction of C and $\neg C_1$

Since there are still values that could be replaced, another inverse resolution step can be applied. For that the parent-relation between *bart* and his father *homer* can be used resulting in the general clause describing a grandfather as can be seen graphically in figure 5.

[2]Variables are written with capital letters, therefore values are written with small letters

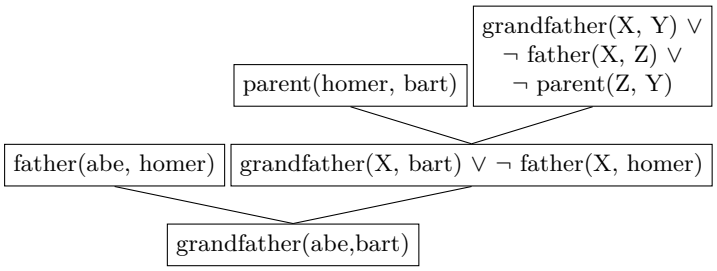

Figure 5: The clause parent(homer, bart) enables the substitution of *bart* with Y and *homer* with Z

Now it is possible to prove, that the initial clause *grandfather(abe,bart)* is valid by inserting the values into the general rule:

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

that was generated by the inverse resolution.

6 Inductive Logic Programming Implementations and Systems

There exist a lot of systems that base on the ILP approach. Besides FOIL which will be described in detail in this section and which is one of the most well-known systems for inductive logic programming. Another widely recognised approach is *GOLEM*, which was developed by Stephen Muggleton[11]. It uses the *relative least general generalization* idea from Gordon Plotkin and is a bottom up approach. There are other systems like *LINUS* and *Progol* to name the most common.

6.1 FOIL - First Order Inductive Learner

The First-Order Inductive Learner (FOIL) is a rule-based learning algorithm. Developed in 1990 by Ross Quinlan[15] FOIL was one of the first well-known ILP-systems. It learns function-free Horn clauses, a subset of first-order predicate calculus. Given positive and negative evidence of a concept and a set of background-knowledge predicates, FOIL inductively generates a logical concept definition or rule for the concept. The learned rule must not involve any constants (*father(X,bart)* becomes *father(X,Y)*). Functions in literals are restricted but literals in the body can be negated, which makes it more expressive. This means it might not result in a true Horn Clause. FOIL is able to learn recursive concepts.

6.2 The FOIL Algorithm

The FOIL algorithm as shown in figure 6 is based on the generic top-down approach which is also described in section 4. FOIL consists of two `while`-loops: the outer loop which searches from specific-to-general and the inner loop that searches from general-to-specific.

```
FOIL(Target_predicate, Predicates, Examples)
    Pos ←those Examples for which the Target_predicate is True
    Neg ←those Examples for which the Target_predicate is False
    Learned_rules ←{}
    while Pos, do
        Learn a NewRule
        NewRule ←predicts Target_predicate with no preconditions
        NewRuleNeg ←Neg
        while NewRuleNeg, do
            Add new literal to specialize NewRule
            Candidate_literals ←generate candidate new literals for
                NewRule, based on Predicates
            Best_literal ←argmax Foil-Gain(L,NewRule) L∈Candidate_literals
            add Best_literal to preconditions of NewRule
            NewRuleNeg ←subset of NewRuleNeg that satisfies NewRule
                preconditions
        Learned_rules ←Learned_rules + NewRule
        Pos ←Pos - {members of Pos covered by NewRule}
    Return Learned_rules
```

Figure 6: The FOIL-algorithm from Quinlan[15]

6.2.1 Outer-Loop

The outer loop (sometimes called *covering-loop*) of the algorithm runs as long as there are positive examples not covered from the learned rules. Initially there are no rules learned, which induces, that all examples are classified negative. With each new rule learned by FOIL the number of examples classified positive should rise until all positive examples are classified as positive. At this moment the outer loop stops.

6.2.2 Inner-Loop

The inner loop is responsible for generating new rules. Since there are no rules in the initial condition of the inner loop, all examples will satisfy the rule. By adding precondition(s), the rule becomes more specific. This is done until the rule is only satisfied by positive examples. Then this rule will be added to the list of learned rules.

6.3 Example for the Inner Loop of FOIL

To learn the rule *grandfather(X, Y)*, the inner loop would do the following steps. Initially the rule would not have any preconditions, resulting in it accepting every example, no matter if negative or positive. To learn this rule, some training examples need to be defined and some background knowledge is needed, which is shown in table 1.

Training examples	Background knowledge	
⊕grandfather(abe, bart).	father(abe, homer).	parent(homer, bart).
⊕grandfather(abe, lisa).	father(homer, bart).	parent(homer, lisa).
⊕grandfather(abe, maggie).	father(homer, lisa).	parent(homer, maggie).
⊖grandfather(abe, marge).	father(homer, maggie).	parent(abe, homer).
⊖grandfather(homer, abe).	mother(marge, bart).	parent(marge, bart).
⊖grandfather(maggie, lisa).	mother(marge, lisa).	parent(marge, lisa).
	mother(marge, lisa).	parent(marge, maggie).

Table 1: Training examples and background knowledge based on the family tree from figure 1, ⊕ marks the positive, ⊖ the negative training examples

Target: *grandfather(X, Y)* ←{}

In the next step a literal is selected greedily from the possible predicates and set as new precondition.

(2) *father(Y,X)*, *father(X,Z)*, *father(Z,X)*, *father(Y,Z)*, *father(Z, Y)*, *parent(X, Y)*, *parent(Y,X)*, *parent(X,Z)*, *parent(Z,X)*, *parent(Y,Z)*, *parent(Z, Y)*, *daughter(X)*, *daughter(Y)*, and all negations
Resulting in: *grandfather(X, Y)* ←*father(Y,Z)*

This improved rule is already able to exclude some of the negative examples, because X has to be the father of someone, who is not equal to Y. This excludes the negative example *grandfather(maggie, lisa)*, since the background knowledge does not contain any information about Maggie being the father of someone. There are still two negative examples covered by the rule, which means the inner loop has to run again to make the rule more precise. This is done by selecting greedily another literal, resulting in the rule: *grandfather(X, Y)*←*father(X,Z)*, *parent(Z, Y)*. This means X has to be father of Z and Z has to be a parent of Y. If this holds, then X is grandfather of Y. The generated rule is able to exclude all negative examples and accepts some (in this case all) positive examples and is therefore added to the list of rules. Since there are no positive examples left, that are not covered, the outer loop finishes and the algorithm comes to a hold at this point.

6.4 Recursive Modelling

The shown example introduces a basic understanding, of what FOIL is capable of. The generated rule can be general applied to conclude whether a person X is a grandfather of a person Y. This is a basic clause, that can be used to form even more complex ones. In the given family-tree, there are more complex relations which can be shortened to the term *ancestor*. All persons in the tree are ancestors to each other in some way. It would be possible to display these relations with rules for all possible combinations. While this can be done for the close relations (e.g. *ancestor(X, Y) :- sister(X, Y)*), it becomes more complex for further away relations (e.g. man of the aunt of an aunt).

This can be circumvented by making use of ILPs ability for recursion. This way an ancestor can be modeled the following way: *ancestor(X, Y) :- parent(Z,X)* ∧*ancestor(Y,Z)*. This way only the close relations would need to be modeled and the further relations are enabled by the recursion of these clauses.

7 Areas of Application

The presented example from section 6.3 shows the basic abilities of the ILP approach. But Inductiv Logic Programming is able to face much more complex and scientifically relevant problems. It has certain useful areas of application. Due to its use of FOL and the human readable results this approach produces, it has a special advantage in scientific theory formation tasks. It is particularly useful in bioinformatics and natural language processing[1]. One area of application is during the development of antibiotics, where it is used to evaluate new combinations of chemicals for their influence on the human body on basis of already known critical relations of chemicals.

Furthermore, can inductive programming be used in the field of domain specific data from a user of a program and therefore provide useful help to the user[6]. This can be done by observing the user and refine the applications behaviour to the users habits.

8 Conclusion

The ILP-approach on machine-learning has advantages over other classifiers when it comes to relations. Due to the usage of the FOL it is possible to display complex relations over multiple nodes. Since ILP is able to handle recursion, rules can be short and still display complex connections in a human-readable way. Moreover the results can be used for example in *Prolog*, to implement smart programs.

The described model theory (see chapter 3) for ILP is the basis for inducing first-order rules. It describes how clauses and literals can be combined to model interpretations, which enable the derivation of more complex relations. This provides the basis to programs, that are able to generate new hypotheses and interpretations, depending on a given input/output relation. This relation can come for example from a sensory system and a supervisor, that first teaches the system. These models are then passed to an implementation of ILP.

The generic ILP algorithm is used as a basis for the implementation of the sequential covering algorithm FOIL, which is a well known approach for classification used in FOL. The created resulst can afterwards be checked using the presented ideas of proof-theory from chapter 5.

Especially the ability of ILP approaches like the described FOIL algorithm, which is able to induce programs from a small amount of input/output examples is the strength which is becoming more important in the recent development of artificial intelligence. The ability to generalize from a small amount of input data can be a chance for smart assistants and smart home applications, where the user mit get tired of telling the system multiple times how a task should be done.

Acronyms

FOIL First-Order Inductive Learner

FOL First-Order Logic

ILP Inductiv Logic Programming

SVM Support Vector Machine

References

[1] Ivan Bratko and Stephen Muggleton. Applications of inductive logic programming. *Commun. ACM*, 38(11):65–70, November 1995.

[2] Samuel R. Buss. On herbrand's theorem. In Daniel Leivant, editor, *Logic and Computational Complexity*, pages 195–209, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[3] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. on Electronic Computers*, EC-14:326–334, 1965.

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

[5] Stefano Ferilli, Nicola Di Mauro, Teresa Maria Altomare Basile, and Floriana Esposito. Theta-subsumption and resolution: A new algorithm. In *Foundations of Intelligent Systems, 14th International Symposium*, pages 384–391. Springer-Verlag, Maebashi City, Japan, January 2003.

[6] Sumit Gulwani, Jose Hernandez-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, and Benjamin Zorn. Inductive programming meets the real world. 58:90–99, 10 2015.

[7] Wilfrid Hodges. Model theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2013 edition, 2013.

[8] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[9] Ryszard Pawel Kostecki. On principles of inductive inference. *Proceedings of the 31st International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, January 2012.

[10] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1 edition, March 1997.

[11] Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8:295–318, 1990.

[12] Stephen Muggleton and Wray L. Buntine. Machine invention of first order predicates by inverting resolution. In John E. Laird, editor, *ML*, pages 339–352. Morgan Kaufmann, 1988.

[13] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19(20):629–679, 1994.

[14] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[15] J. R. Quinlan. Learning logical definitions from relations. *MACHINE LEARNING*, 5:239–266, 1990.

[16] Brian D Ripley. *Pattern recognition and neural networks*. Cambridge university press, 2007.

[17] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.

[18] Céline Rouveirol and Patrick Albert. Agent-based knowledge acquisition. In *Proceedings of the 8th European Knowledge Acquisition Workshop on A Future for Knowledge Acquisition*, EKAW '94, pages 374–393, London, UK, 1994. Springer-Verlag.

[19] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[20] Arul Siromoney and Katsushi Inoue. The generic rough set inductive logic programming (grs-ilp) model. Studies in Fuzziness and Soft Computing, pages 499 – 517. Physica-Verlag HD, 2013.

[21] Christian Strasser and G. Aldo Antonelli. Non-monotonic logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[22] Jan von Plato. The development of proof theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.