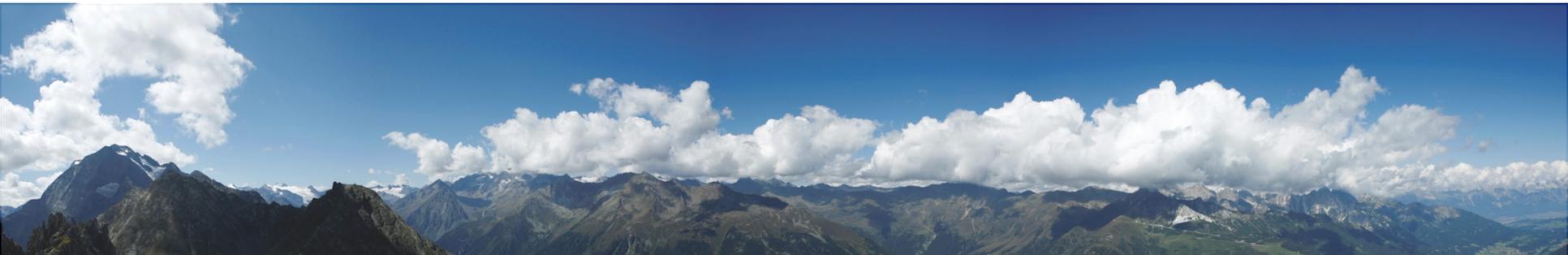# Intelligent Systems

Lecture XI – xx 2009
Inductive Logic Programming

Dieter Fensel and Florian Fischer

Where are we?

#	Date	Title
1		Introduction
2		Propositional Logic
3		Predicate Logic
4		Theorem Proving, Description Logics, and Logic Programming
5		Search Methods
6		CommonKADS
7		Problem Solving Methods
8		Planning
9		Agents
10		Rule Learning
11		**Inductive Logic Programming**
12		Formal Concept Analysis
13		Neural Networks
14		Semantic Web and Exam Preparation

- Motivation
- Technical Solution
  - Model Theory of ILP
  - A Generic ILP Algorithm
  - Proof Theory of ILP
  - ILP Systems and Applications
- Illustration by a Larger Example
- Summary and Conclusions

MOTIVATION

- There is a vast array of different machine learning techniques, i.e.:
  - Reinforcement learning
  - Neural networks
  - and… *Inductive Logic Programming (ILP)*

- ILP combines logic with machine learning
  - Combine inductive machine learning with representation of logic programming
  - Idea: Induce general rules starting from specific observations + background knowledge

- Advantages over other ML approaches
  - ILP uses an expressive First-Order framework instead of simple attribute-value framework of other approaches
  - ILP can take background knowledge into account

Inductive Logic Programming

=

Inductive Learning ∩ Logic Programming

- From inductive machine learning, ILP inherits its goal: to develop tools and techniques to
  - Induce hypotheses from observations (examples)
  - Synthesise new knowledge from experience
- By using computational logic as the representational mechanism for hypotheses and observations, ILP can overcome the two main limitations of classical machine learning techniques:
  - The use of a limited knowledge representation formalism (essentially a propositional logic)
  - Difficulties in using substantial background knowledge in the learning process

- ILP inherits from logic programming its
  - Representational formalism
  - Semantical orientation
  - Various well-established techniques
- Compared to other approaches to inductive learning, ILP is interested in
  - Properties of inference rules
  - Convergence of algorithms
  - Computational complexity of procedures
- ILP systems benefit from using the results of logic programming
  - E.g. by making use of work on termination, types and modes, knowledge-base updating, algorithmic debugging, abduction, constraint logic programming, program synthesis and program analysis

The inductive learning and logic programming sides of ILP (cont')

- Inductive logic programming extends the theory and practice of logic programming by investigating induction rather than deduction as the basic mode of inference
  - Logic programming theory describes deductive inference from logic formulae provided by the user
  - ILP theory describes the inductive inference of logic programs from instances and background knowledge
- ILP contributes to the practice of logic programming by providing tools that assist logic programmers to develop and verify programs

- Imagine yourself as learning about the relationships between people in your close family circle
  - You have been told that your grandfather is the father of one of your parents, but do not yet know what a parent is
  - You might have the following beliefs (**B**):

    grandfather(X, Y) ← father(X, Z); parent(Z, Y)
    father(henry, jane) ←
    mother(jane. john) ←
    mother(jane, alice) ←

  - You are now given the following facts (positive examples) concerning the relationships between particular grandfathers and their grandchildren (**E$^+$**):

    grandfather(henry, john) ←
    grandfather(henry, alice) ←

  - You might be told in addition that the following relationships do not hold (negative examples) (**E$^-$**)

    ← grandfather(john, henry)
    ← grandfather(alice, john)

- Believing **B**, and faced with the new facts E^+ and E^- you might guess the following hypothesis **H**

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

- First, we must check that our problem has a solution:

$$B \wedge E^- \not\models \Box \ (prior \ satisfiability)$$

- However, H allows us to explain E^+ relative to B:

$$B \wedge H \models E^+ \ (posterior \ sufficiency)$$

- B and H are consistent with E^-:

$$B \wedge H \wedge E^- \not\models \Box \ (posterior \ satisfiability)$$

- The question: how it is possible to derive (even tentatively) the hypothesis H?

Model Theory of ILP, A Generic ILP Algorithm, Proof Theory of ILP, ILP Systems and Applications

TECHNICAL SOLUTIONS

- ## The problem of inductive inference:
  - Given is background (prior) knowledge B and evidence E
  - The evidence $E = E^+ \wedge E^-$ consists of positive evidence E^+ and negative evidence E^-
  - The aim is then to **find** a hypothesis H such that the following conditions hold:

    Prior Satisfiability: $B \wedge E^- \not\models \square$

    Posterior Satisfiability: $B \wedge H \wedge E^- \not\models \square$

    Prior Necessity: $B \not\models E^+$

    Posterior Sufficiency: $B \wedge H \models E^+$

- The Sufficiency criterion is sometimes named completeness with regard to positive evidence
- The Posterior Satisfiability criterion is also known as consistency with the negative evidence

- In most ILP systems background theory and hypotheses are restricted to being definite
  - This setting is simpler than the general setting because a definite clause theory T has a unique minimal Herbrand model $M^+(T)$, and any logical formulae is either true or false in the minimal model
- This setting is formalised as follows:

  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 Necessity: some e in E^+ are false in $M^+(B)$

  Posterior Sufficiency: all e in E^+ are true in $M^+(B \wedge H)$

- The special case of the definite semantics, where the evidence is restricted to true and false ground facts (examples), is called the example setting
  - The example setting is equivalent to the normal semantics, where B and H are definite clauses and E is a set of ground unit clauses

- In the non-monotonic setting:
    - The background theory is a set of definite clauses
    - The evidence is empty (Reason: the positive evidence is considered part of the background theory and the negative evidence is derived implicitly, by making a kind of closed world assumption)
    - The hypotheses are sets of general clauses expressible using *the same alphabet* as the background theory

- The following conditions should hold for H and B:
  - Validity: all *h in H* are true in $M^+(B)$
    - All clauses belonging to a hypothesis hold in the database B, i.e. that they are true properties of the data
  - Completeness: if general clause *g* is true in $M^+(B)$ then *H |= g*
    - All information that is valid in the database should be encoded in the hypothesis
  - Minimality: there is no proper subset *G* of *H* which is valid and complete
    - Aims at deriving non redundant hypotheses

- Example for B:

  male(luc) ←

  female(lieve) ←

  human(lieve) ←

  human(luc) ←

- A possible solution is then H (a set of general clauses):

  ← female(X), male(X)

  human(X) ← male(X)

  human(X) ← female(X)

  female(X), male(X) ← human(X)

- Example for B_1
  bird(tweety) ←
  bird(oliver) ←

- Example for E_1^+
  flies(tweety)

- In the example setting an acceptable hypothesis H_1 would be *flies(X) ← bird(X)*

- In the non-monotonic setting H_1 is not a solution since there exists a substitution {X ← oliver} which makes the clause false (non-valid)

- The non-monotonic semantics realises induction by *deduction*

- The induction principle of the non-monotonic setting states that the hypothesis H, which is, in a sense, *deduced* from the set of observed examples E and the background theory B (using a kind of closed world and closed domain assumption), holds for *all* possible sets of examples

  – This produces generalisation beyond the observations

  – As a consequence, properties derived in the non-monotonic setting are more conservative than those derived in the normal setting

- ILP can be seen as a search problem - this view follows immediately from the model-theory of ILP

  - In ILP there is a space of candidate solutions, i.e. the set of hypotheses, and an acceptance criterion characterizing solutions to an ILP problem

- Question: how the space of possible solutions can be structured in order to allow for pruning of the search?

  - The search space is typically structured by means of the dual notions of generalisation and specialisation

    - Generalisation corresponds to induction
    - Specialisation to deduction
    - Induction is viewed here as the inverse of deduction

- A hypothesis G is more general than a hypothesis S if and only if G |= S
  - S is also said to be more specific than G.
- In search algorithms, the notions of generalisation and specialisation are incorporated using inductive and deductive inference rules:
  - A deductive inference rule *r in R* maps a conjunction of clauses G onto a conjunction of clauses S such that G |= S
    - *r* is called a specialisation rule
  - An inductive inference rule *r in R* maps a conjunction of clauses S onto a conjunction of clauses G such that G |= S
    - *r* is called a generalisation rule

Specialisation and Generalisation – Examples

- Example of deductive inference rule is resolution; also dropping a clause from a hypothesis realises specialisation

- Example of an inductive inference rule is Absorption:

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

- 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 generalise by adding a clause to a hypothesis, or by dropping a negative literal from a clause

- Inductive inference rules, such as Absorption, are not sound

- This soundness problem can be circumvented by associating each hypothesised conclusion H with a label $L = p(H \mid B \wedge E)$ 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

Pruning the search space

- Generalisation and specialisation form the basis for pruning the search space; this is because:
  - When $B \wedge H \not\models e$, where B is the background theory, H is the hypothesis and e is positive evidence, then none of the specialisations H' of H will imply the evidence. Each such hypothesis will be assigned a probability label $p(H' \mid B \wedge E) = 0$.
    - 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 generalisations H' of H will also be inconsistent with $B \wedge E$. These will again have $p(H' \mid B \wedge E) = 0$.

A Generic ILP Algorithm

- Given the key ideas of ILP as search, inference rules and labeled hypotheses, a generic ILP system is defined as:

$QH :=$ Initialize
repeat
  Delete H from QH
  Choose *the inference rules* $r_1, ..., r_k \in \mathbf{R}$ *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 algorithm 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 process continues until the stop-criterion is satisfied

Algorithm – Generic Parameters

- **Initialize** denotes the hypotheses started from
- **R** denotes the set of inference rules applied
- **Delete** influences the search strategy
  - Using different instantiations of this procedure, one can realise a depth-first (Delete = LIFO), breadth-first Delete = FIFO) or best-first algorithm
- **Choose** determines the inference rules to be applied on the hypothesis H

- Prune determines which candidate hypotheses are to be deleted from the queue
  - This is usually realized using the labels (probabilities) of the hypotheses on QH or relying on the user (employing an "oracle")
  - Combining **Delete** with **Prune** it is easy to obtain advanced search strategies such as hill-climbing, beam-search, best-first, etc.

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

STI · INNSBRUCK

- "specific-to-general" systems
  - Start from the examples and background knowledge, and repeatedly generalize their hypothesis by applying inductive inference rules
  - During the search they take care that the hypothesis remains satisfiable (i.e. does not imply negative examples)

- "general-to-specific" systems
  - Start with the most general hypothesis (i.e. the inconsistent clause) and repeatedly specializes the hypothesis by applying deductive inference rules in order to remove inconsistencies with the negative examples
  - During the search care is taken that the hypotheses remain sufficient with regard to the positive evidence

Proof Theory of ILP

- Inductive inference rules can be obtained by inverting deductive ones
  - Deduction: Given $B \wedge H \models E^+$, derive E^+ from $B \wedge H$
  - Induction: Given $B \wedge H \models E^+$, derive H from B and B and E^+
- Inverting deduction paradigm can be studied under various assumptions, corresponding to different assumptions about the deductive rule for $\models$ and the format of background theory B and evidence E^+
$\Rightarrow$ Different models of inductive inference are obtained
  - **θ-subsumption**
    - **The background knowledge is supposed to be empty, and the deductive inference rule corresponds to θ-subsumption among single clauses**
  - **Inductive inference**
    - **Takes into account background knowledge and aims at inverting the resolution principle, the best-known deductive inference rule**
  - **Inverting implication**

Rules of inductive inference and operators

- Inference rules = what can be inferred from what
- A well-known problem: the unrestricted application of inference rules results in combinatorial explosions
  - To control the application of inference rules, the search strategy employs operators that expand a given node in the search tree into a set of successor nodes in the search
- A specialisation operator maps a conjunction of clauses *G* onto a set of maximal specialisations of *S;* A *maximal specialisation S* of *G* is a specialisation of *G* such that
  - *G* is not a specialisation of *S*
  - There is no specialisation *S'* of *G* such that *S* is a specialisation of *S'*
- A generalisation operator maps a conjunction of clauses S onto a set of minimal generalisations of S; A *minimal generalisation G* of *S* is a generalisation of *S* such that
  - *S* is not a generalisation of *G*
  - There is no generalisation *G'* of *S* such that *G* is a generalisation of *G'*

- A clause c_1 *θ-subsumes* a clause c_2 if and only if there exists a substitution *θ* such that *c_1θ in c_2*
  - c_1 is a generalisation of c_2 (and c_2 a specialisation of c_1) under *θ*-subsumption
  - Clauses are seen as sets of (positive and negative) literals
- The *θ*-subsumption inductive inference rule is:

$$\theta\text{-\textbf{subsumption:}} \quad \frac{c_2}{c_1} \text{ where } c_1\theta \subseteq c_2$$

- For example
  father(X,Y) ← parent(X,Y), male(X)
  θ-subsumes
  father(jef,paul) ← parent(jef,paul), parent(jef,ann), male(jef), female(ann)
  with *θ = {X = jef, Y = ann}*

- Implication: If c_1 *θ*-subsumes c_2 then $c_1 \models c_2$
  - The opposite does not hold for self-recursive clauses: let $c_1 = p(f(X)) \leftarrow p(X)$; $c_2 = p(f(f(Y))) \leftarrow p(Y)$; $c_1 \models c_2$ but c_1 does not *θ*-subsume c_2
  - Deduction using *θ*-subsumption is not equivalent to implication among clauses
- Infinite Descending Chains: There exist infinite descending chains, e.g.

  $h(X_1,X_2) \leftarrow$
  $h(X_1,X_2) \leftarrow p(X_1,X_2)$
  $h(X_1,X_2) \leftarrow p(X_1,X_2),p(X_2,X_3)$
  $h(X_1,X_2) \leftarrow p(X_1,X_2),p(X_2,X_3),p(X_3,X_4)$

  ...
  This series is bounded from below by $h(X,X) \leftarrow p(X,X)$

- Equivalence: there exist different clauses that are equivalent under *θ*-subsumption
  - E.g. *parent(X,Y) ← mother(X,Y), mother(X,Z) θ*-subsumes *parent(X,Y) ← mother(X,Y)* and vice versa
- Reduction: for equivalent classes of clauses, there is a unique representative (up to variable renamings) of each clause (reduced clause)
  - The reduced clause *r* of a clause *c* is a minimal subset of literals of *c* such that *r* is equivalent to *c*
- Lattice: the set of reduced clauses form a lattice, i.e. any two clauses have unique *lub* (the least general generalisation - *lgg*) and any two clauses have a unique *glb*

- Four rules of inverse resolution

Absorption:	$q \leftarrow A$	$p \leftarrow A, B$
	$q \leftarrow A$	$p \leftarrow q, B$

Identification:	$p \leftarrow A, B$	$p \leftarrow A, q$
	$q \leftarrow B$	$p \leftarrow A, q$

Intra-construction:

$$\frac{p \leftarrow A, B \qquad\qquad\qquad p \leftarrow A, C}{q \leftarrow B \qquad p \leftarrow A, q \qquad q \leftarrow C}$$

Inter-construction:

$$\frac{p \leftarrow A, B \qquad\qquad\qquad q \leftarrow A, C}{p \leftarrow r, B \qquad r \leftarrow A \qquad q \leftarrow r, C}$$

- Lower-case letters are atoms and upper-case letters are conjunctions of atoms

- Both Absorption and Identification invert a single resolution step
- This is shown diagrammatically as a V with the two premises on the base and one of the arms

$$q \twoheadleftarrow A \qquad\qquad p \twoheadleftarrow q, B$$

$$p \twoheadleftarrow A, B$$
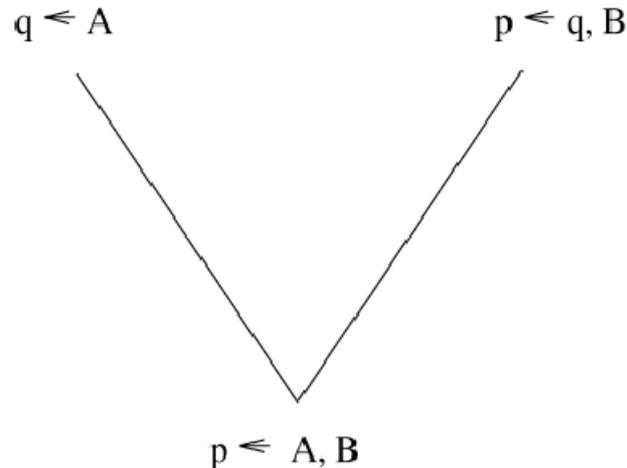
- The new clause in the conclusion is the clause found on the other arm of the V
- Absorption and Identification were called collectively V-operators

Inter- and Intra-construction

- The rules of Inter- and Intra-construction introduce a new predicate symbol
- The new predicate can then be generalised using a V-operator; diagrammatically the construction operators can be shown as two linked V's, or a W, each respresenting a resolution

$$q \Leftarrow B \qquad p \Leftarrow A, q \qquad q \Leftarrow C$$

$$p \Leftarrow A, B \qquad p \Leftarrow A, C$$
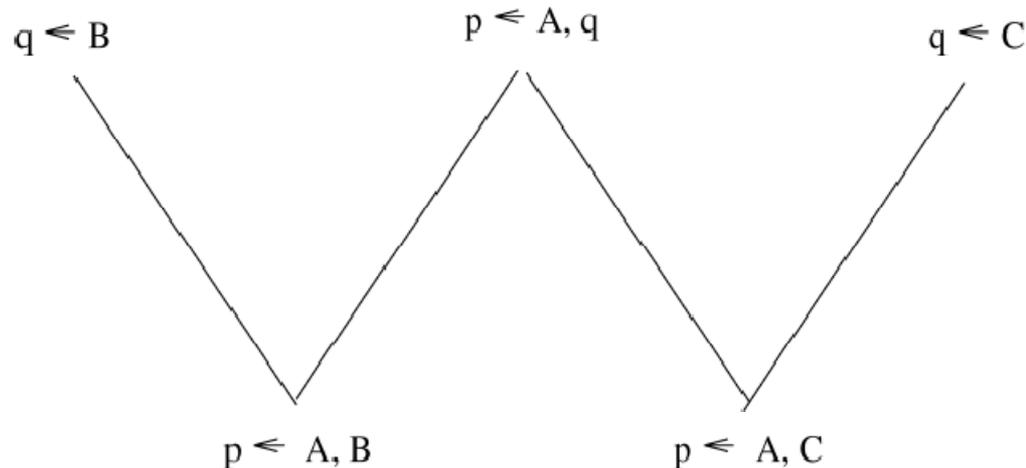
- The premises are placed at the two bases of the W and the three conclusions at the top of the W
  - One of the clauses is shared in both resolutions.
- Intra- and Inter-construction are collectively called W-operators

STI · INNSBRUCK

- The V and W operators have most specific forms:

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

Identification↓:
$$\frac{p \leftarrow A, B \qquad p \leftarrow A, q}{q \leftarrow A, B \qquad p \leftarrow A, q}$$

Intra-construction↓:
$$\frac{p \leftarrow A, B \qquad\qquad\qquad p \leftarrow A, C}{q \leftarrow A, B \qquad p \leftarrow A, q \qquad q \leftarrow A, C}$$

Inter-construction↓:
$$\frac{p \leftarrow A, B \qquad\qquad\qquad q \leftarrow A, C}{p \leftarrow r, A, B \qquad r \leftarrow A \qquad q \leftarrow r, A, C}$$

- In this form the V-operators realise both generalisation and specialisation since the conclusions entail the premises
- Use of most specific operators is usually implemented by having a two stage operation
  - In the first phase, inverse resolution operators are applied to examples
  - In the second phase clauses are reduced by generalisation through the θ-subsumption lattice

- Just as resolution requires unification to match terms, inverse resolution operators require a matching operation

- In matching operation all clauses, including the examples are *flattened*

  – This involves introducing a new (n+1)-ary predicate for every n-ary function symbol, e.g. the clause *member(a,[a,b]) ←* becomes

  member(U,V) ← a(U), dot(V,U,X), dot(X,Y,Z),b(Y),nil(Z).

  – Each new predicate symbol is then separately defined, e.g. *dot([X|Y],X,Y) ←*

  – After flattening the problem of matching clauses when applying the inverse resolution operators reduces to one-sided matching of clause bodies

- Incremental/non-incremental: describes the way the evidence E (examples) is obtained
  - In non-incremental or empirical ILP, the evidence is given at the start and not changed afterwards
    - Non-incremental systems search typically either specific-to-general or general-to-specific.
  - In incremental ILP, the examples are input one by one by the user, in a piecewise fashion.
    - Incremental systems usually employ a mixture of specific-to-general and general-to-specific strategies as they may need to correct earlier induced hypotheses
- Interactive/ Non-interactive
  - In interactive ILP, the learner is allowed to pose questions to an oracle (i.e. the user) about the intended interpretation
    - Usually these questions query the user for the intended interpretation of an example or a clause.
    - The answers to the queries allow to prune large parts of the search space
  - Most systems are noninteractive

- Single/Multiple Predicate Learning/Theory Revision
  - Suppose *P(F)* represent the predicate symbols found in formula *F*
  - In single predicate learning from examples, the evidence E is composed of examples for one predicate only, i.e. P(E) is a singleton
  - In multiple predicate learning, P(E) is not restricted as the aim is to learn a set of possibly interrelated predicate definitions
  - Theory revision is usually a form of incremental multiple predicate learning, where one starts from an initial approximation of the theory
- Numerical Data
  - With a small number of examples it is hard to get enough examples in which the prediction is an exact number
  - Instead rules should predict an interval

Concrete ILP implementations

- Extensive list of ILP systems is included in references
- A well known family of related systems: Progol
  - CProgol, PProgol, Aleph
- Progol allows arbitrary Prolog programs as background knowledge and arbitrary definite clauses as examples
- Most comprehensive implementation: CProgol
  - Homepage: http://www.doc.ic.ac.uk/~shm/progol.html
    - General instructions (download, installation, etc.)
    - Background information
    - Example datasets
  - Open source and free for research and teaching

- CProgol uses a covering approach: It selects an example to be generalised and finds a consistent clause covering the example
  - Theoretical foundations:
    - Inverse entailment with general-to-specific search through a refinement graph
    - S.H. Muggleton. *Inverse entailment and Progol*. New Generation Computing, Special issue on Inductive Logic Programming, 13(3-4):245-286, 1995.
- Basic algorithm for CProgol:
  1. Select an example to be generalized.
  2. Build most-specific-clause. Construct the most specific clause that entails the example selected, and is within language restrictions provided. This is usually a definite clause with many literals, and is called the "bottom clause."
  3. Find a clause more general than the bottom clause. This is done by searching for some subset of the literals in the bottom clause that has the "best" score.
  4. Remove redundant examples. The clause with the best score is added to the current theory, and all examples made redundant are removed. Return to Step 1 unless all examples are covered.
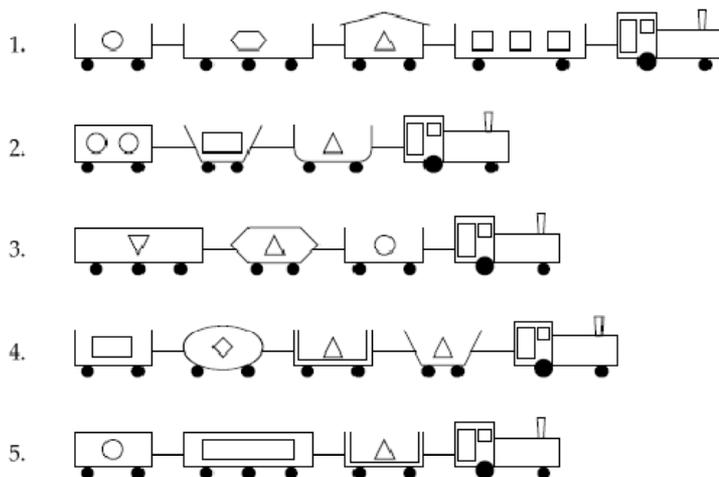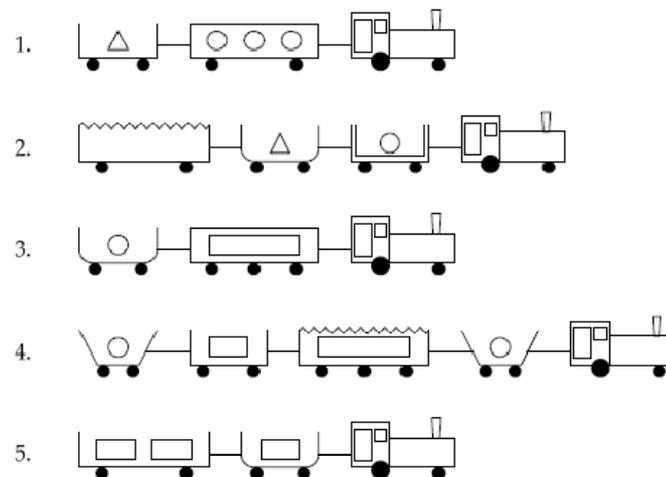
Michalski's train problem

ILLUSTRATION BY A LARGER EXAMPLE

- Assume ten railway trains: five are travelling east and five are travelling west; each train comprises a locomotive pulling wagons; whether a particular train is travelling towards the east or towards the west is determined by some properties of that train

- The learning task: determine what governs which kinds of trains are Eastbound and which kinds are Westbound

- Michalski's train problem can be viewed as a classification task: the aim is to generate a classifier (theory) which can classify unseen trains as either Eastbound or Westbound

- The following knowledge about each car can be extracted: which train it is part of, its shape, how many wheels it has, whether it is open (i.e. has no roof) or closed, whether it is long or short, the shape of the things the car is loaded with. In addition, for each pair of connected wagons, knowledge of which one is in front of the other can be extracted.

- Examples of Eastbound trains
  - Positive examples:

    eastbound(east1).

    eastbound(east2).

    eastbound(east3).

    eastbound(east4).

    eastbound(east5).

  - Negative examples:

    eastbound(west6).

    eastbound(west7).

    eastbound(west8).

    eastbound(west9).

    eastbound(west10).

- Background knowledge for train *east1*. Cars are uniquely identified by constants of the form *car_xy*, where *x* is number of the train to which the car belongs and *y* is the position of the car in that train. For example *car_12* refers to the second car behind the locomotive in the first train
  - short(car_12). short(car_14).
  - long(car_11). long(car_13).
  - closed(car_12).
  - open(car_11). open(car_13). open(car_14).
  - infront(east1,car_11). infront(car_11,car_12).
  - infront(car_12,car_13). infront(car_13,car_14).
  - shape(car_11,rectangle). shape(car_12,rectangle).
  - shape(car_13,rectangle). shape(car_14,rectangle).
  - load(car_11,rectangle,3). load(car_12,triangle,1).
  - load(car_13,hexagon,1). load(car_14,circle,1).
  - wheels(car_11,2). wheels(car_12,2).
  - wheels(car_13,3). wheels(car_14,2).
  - has_car(east1,car_11). has_car(east1,car_12).
  - has_car(east1,car_13). has_car(east1,car_14).

STI · INNSBRUCK

- An ILP systems could generate the following hypothesis:

    eastbound(A) ← has_car(A,B), not(open(B)), not(long(B)).

    i.e. A train is eastbound if it has a car which is both not open and not long.

- Other generated hypotheses could be:
  - If a train has a short closed car, then it is Eastbound and otherwise Westbound
  - If a train has two cars, or has a car with a corrugated roof, then it is Westbound and otherwise Eastbound
  - If a train has more than two different kinds of load, then it is Eastbound and otherwise Westbound
  - For each train add up the total number of sides of loads (taking a circle to have one side); if the answer is a divisor of 60 then the train is Westbound andotherwise Eastbound

- Download Progrol
  - http://www.doc.ic.ac.uk/~shm/Software/progol5.0
- Use the Progol input file for Michalski's train problem
  - http://www.comp.rgu.ac.uk/staff/chb/teaching/cmm510/michalski_train_data
- Generate the hypotheses

SUMMARY

- ILP is a subfield of machine learning which uses logic programming as a uniform representation for
  - Examples
  - Background knowledge
  - Hypotheses
- Many existing ILP systems
  - Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesised logic program which entails all the positive and none of the negative examples
- Lots of applications of ILP
  - E.g. bioinformatics, natural language processing, engineering
- IPL is an active research filed

REFERENCES

- S.H. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8(4):295-318, 1991.
- S.H. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19,20:629-679, 1994.
- S.H. Muggleton. Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming, 13(3-4):245-286, 1995.
- S.H. Muggleton. Learning from positive data. In S. Muggleton, editor, *Proceedings of the 6th International Workshop on Inductive Logic Programming*, p. 225-244, 1996.
- http://en.wikipedia.org/wiki/Inductive_logic_programming
- J. Lloyd. Logic for learning: learning comprehensible theories from structured data. 2003.
  - http://www.cs.bris.ac.uk/~ILPnet2/Tools/Reports/Abstracts/2003-lloyd.html
- S. Dzeroski and N. Lavrac, editors. Relational Data Mining. September 2001.
  - http://www.cs.bris.ac.uk/~ILPnet2/Tools/Reports/Abstracts/2001-dzeroski.html
- L. De Raedt, editor. Advances in Inductive Logic Programming. 1996.
  - http://www.cs.bris.ac.uk/~ILPnet2/Tools/Reports/Abstracts/rae96:book.html
- N. Lavrac and S. Dzeroski. Inductive Logic Programming: Techniques and Applications. 1994.
  - http://www.cs.bris.ac.uk/~ILPnet2/Tools/Reports/Abstracts/lavdze94:book.html

- PROGOL (http://www.doc.ic.ac.uk/~shm/Software/progol5.0)
- Golem (ILP) (http://www.doc.ic.ac.uk/~shm/Software/golem)
- Aleph (http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/)
- Foil (ftp://ftp.cs.su.oz.au/pub/foil6.sh)
- Claudien (http://www.cs.kuleuven.ac.be/~ml/CWIS/claudien-E.shtml)
- Lime (http://cs.anu.edu.au/people/Eric.McCreath/lime.html)
- ACE (http://www.cs.kuleuven.ac.be/~dtai/ACE/)
- DMax (http://www.pharmadm.com/dmax.asp)
- Warmr (http://www.cs.kuleuven.ac.be/~ml/Doc/TW_User/)
- RSD (http://labe.felk.cvut.cz/~zelezny/rsd/)
- Mio (http://kd.cs.uni-magdeburg.de/~pena/)
- DL-Learner (http://dl-learner.org)

Next Lecture

#	Date	Title
1		Introduction
2		Propositional Logic
3		Predicate Logic
4		Theorem Proving, Description Logics, and Logic Programming
5		Search Methods
6		CommonKADS
7		Problem Solving Methods
8		Planning
9		Agents
10		Rule Learning
11		Inductive Logic Programming
12		**Formal Concept Analysis**
13		Neural Networks
14		Semantic Web and Exam Preparation

Questions?