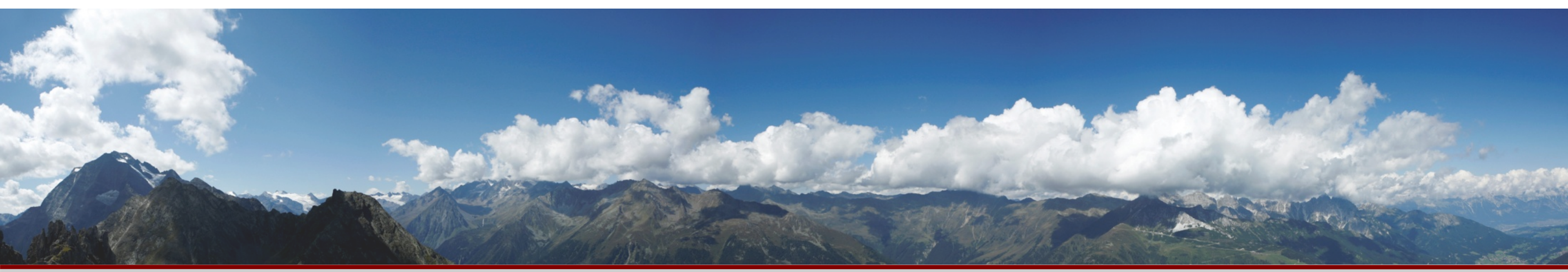
Artificial Intelligence

Inductive Logic Programming



Where are we?

STI · INNSBRUCK

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Reasoning
5	Search Methods
6	CommonKADS
7	Problem-Solving Methods
8	Planning
9	Software Agents
10	Rule Learning
11	**Inductive Logic Programming**
12	Formal Concept Analysis
13	Neural Networks
14	Semantic Web and Services

Agenda

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

MOTIVATION

Motivation

- There is a vast array of different machine learning techniques, e.g.:
  - Decision Tree Learning (see previous lecture)
  - Neural networks
  - and… ***Inductive Logic Programming (ILP)***

- 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

STI · INNSBRUCK

Inductive Logic Programming

=

Inductive Learning ∩ Logic Programming

The inductive learning and logic programming sides of ILP

- 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

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

STI · INNSBRUCK

- ILP inherits from logic programming its
  - Representational formalism
  - Semantical orientation
  - Various wellestablished techniques

- ILP systems benefit from using the results of logic programming
  - E.g. by making use of work on termination, types and modes, knowledgebase updating, algorithmic debugging, abduction, constraint logic programming, program synthesis and program analysis

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

STI · INNSBRUCK

- 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

Introduction – Basic example

- Imagine 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 positive examples concerning the relationships between particular grandfathers and their grandchildren ($\mathbf{E}^+$):

  grandfather(henry, john) ←

  grandfather(henry, alice) ←

Introduction – Basic example

STI · INNSBRUCK

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

  $\leftarrow$ *grandfather(john, henry)*

  $\leftarrow$ *grandfather(alice, john)*

- Believing **B**, and faced with examples **E**$^+$ and **E**$^-$ you might guess the following hypothesis $\mathbf{H_1} \in \mathbf{H}$

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

- **H** is the set of hypotheses and contain an arbitrary number of individual speculations that fit the background knowledge and examples
- Several conditions have to be fulfilled by a hypothesis
  - Those conditions are related to completeness and consistency with respect to the background knowledge and examples

Introduction – Basic example

STI · INNSBRUCK

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

    $$B \cup E^- \not\models \Box \ (\textit{prior satisfiability})$$

    - If one of the negative examples can be proved to be true from the background information alone, then any hypothesis we find will not be able to compensate for this. The problem is not satisfiable.
  - B and H are consistent with E^-:

    $$B \cup H \cup E^- \not\models \Box \ (\textit{posterior satisfiability})$$

    - After adding a hypothesis it should still not be possible to prove a negative example.
- Completeness:
  - However, H allows us to explain E^+ relative to B:

    $$B \cup H \models E^+ \ (\textit{posterior sufficiency})$$

    - This means that H should fit the positive examples given.

TECHNICAL SOLUTIONS

Model Theory of ILP

Model Theory – Normal Semantics

- The problem of inductive inference:
  - Given is background (prior) knowledge B and evidence E
  - The evidence $E = E^+ \cup E^-$ 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: $B \cup E^- \not\models \square$

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

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

    Posterior Sufficiency: $B \cup H \models E^+$
- 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 this general setting, background-theory, examples, and hypotheses can be any (well-formed) formula

Model Theory – Definite Semantics

- In most ILP practical systems background theory and hypotheses are restricted to being definite clauses
  - Clause: A disjunction of literals
  - Horn Clause: A clause with at most one positive literal
  - Definite Clause: A Horn clause with **exactly** one positive literal

$$\neg p \vee \neg q \vee \cdots \vee \neg t \vee u$$

- This setting has the advantage that definite clause theory T has a unique minimal Herbrand model $M^{+}(T)$
  - Any logical formulae is either true or false in this minimal model (all formulae are decidable and the Closed World Assumption holds)

Model Theory – Definite Semantics

- The definite semantics again require a set of conditions to hold
- We can now refer to every formula in *E* since they are guaranteed to have a truth value in the minimal model

- Consistency:

  Prior Satisfiability: all e *in* E^- are false in $M^+(B)$
  - Negative evidence should not be part of the minimal model

  Posterior Satisfiability: all e *in* E^- are false in $M^+(B \cup H)$
  - Negative evidence should not be supported by our hypotheses
- Completeness

  Prior Necessity: some e *in* E^+ are false in $M^+(B)$
  - If all positive examples are already true in the minimal model of the background knowledge, then no hypothesis we derive will add useful information

  Posterior Sufficiency: all e *in* E^+ are true in $M^+(B \cup H)$
  - All positive examples are true (explained by the hypothesis) in the minimal model of the background theory and the hypothesis

Model Theory – Definite Semantics

STI · INNSBRUCK

- An additional restriction in addition to those of the definite semantics is to only allow true and false ground facts as examples (evidence)
- This is called the example setting
  - The example setting is the main setting employed by ILP systems
  - Only allows factual and not causal evidence (which usually captures more knowledge)

- Example:
  - B: *grandfather(X, Y)* $\leftarrow$ *father(X, Z), parent(Z, Y)*
    father(henry, jane) $\leftarrow$
    etc.
  - E: *grandfather(henry, john)* $\leftarrow$
    grandfather(henry, alice) $\leftarrow$

Not allowed in example setting:

$\leftarrow$ *grandfather(X, X)* — Not allowed in definite semantics

grandfather(henry, john) $\leftarrow$ *father(henry, jane), mother(jane, john)*

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

STI · INNSBRUCK

- Since only positive evidence is present, it is assumed to be part of the background theory:

$$B' = B \cup E$$

- 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 $H \models g$
    - All information that is valid in the minimal model of B' should follow from the hypothesis
- Additionally the following ***can*** be a requirement:
  - Minimality: there is no proper subset *G* of *H* which is valid and complete
    - The hypothesis should not contain redundant clauses

- Example for B (definite clauses):

  $male(luc) \leftarrow$

  $female(lieve) \leftarrow$

  $human(lieve) \leftarrow$

  $human(luc) \leftarrow$

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

  $\leftarrow female(X), male(X)$

  $human(X) \leftarrow male(X)$

  $human(X) \leftarrow female(X)$

  $female(X), male(X) \leftarrow human(X)$

- One more example to illustrate the difference between the example setting and the non-monotonic setting

- Consider:
  - Background theory B

    bird(tweety) ←

    bird(oliver) ←
  - Examples E^+:

    flies(tweety)

  - For the non-monotonic setting B' = B ∪ E^+ because positive examples are considered part of the background knowledge

STI · INNSBRUCK

- Example setting:
  - An acceptable hypothesis H_1 would be
    flies(X) $\leftarrow$ *bird(X)*
  - It is acceptable because if fulfills the completeness and consistency criteria of the definite semantics
  - This realizes can inductive leap because *flies(oliver)* is true in
    $M^+(B \cup H) = \{ bird(tweety), bird(oliver), flies(tweety), flies(oliver) \}$
- Non-monotonic setting:
  - H_1 is not a solution since there exists a substitution {X $\leftarrow$ oliver} which makes the clause false in $M^+(B')$ (the validity criteria is violated:

    $M^+(B') = \{ bird(tweety), bird(oliver), flies(tweety) \}$

    $\{X \leftarrow oliver\}$: ***flies(oliver)*** $\leftarrow$ *bird(oliver)*

    $\{X \leftarrow tweety\}$: *flies(tweety)* $\leftarrow$ *bird(tweety)*

TECHNICAL SOLUTIONS

A Generic ILP Algorithm

ILP as a Search Problem

- ILP can be seen as a search problem - this view follows immediately from the modeltheory 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

Specialisation and Generalisation Rules

- 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.
- In search algorithms, the notions of generalisation and specialisation are incorporated using inductive and deductive inference rules:
  - A deductive inference rule *r* maps a conjunction of clauses G onto a conjunction of clauses S such that $G \models S$
    - *r* is called a specialisation rule
  - An inductive inference rule *r* maps a conjunction of clauses S onto a conjunction of clauses G such that $G \models S$
    - *r* is called a generalisation rule

Pruning the search space

- Generalisation and specialisation form the basis for pruning the search space; this is because:
  - When $B \cup H \nvDash e$, where $e \in E^+$, B is the background theory, H is the hypothesis, then none of the specialisations H' of H will imply the evidence
    - They can therefore be pruned from the search.
  - When $B \cup H \cup \{e\} \vDash \square$, where $e \in E^-$, B is the background theory, H is the hypothesis, then all generalisations H' of H will also be inconsistent with $B \cup E$
    - We can again drop them

A Generic ILP Algorithm

- Given the key ideas of ILP as search a generic ILP system is defined as:

QH := **Initialize**
repeat
  Delete H *from* QH
  Choose *the inference rules* $r_1, ..., r_k \in \mathtt{R}$ *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 stopcriterion 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 depthfirst (Delete = LIFO), breadthfirst Delete = FIFO) or bestfirst algorithm
- Choose determines the inference rules to be applied on the hypothesis H

Algorithm – Generic Parameters (2)

- Prune determines which candidate hypotheses are to be deleted from the queue
  - This can also be done by relying on the user (employing an “oracle”)
  - Combining **Delete** with **Prune** it is easy to obtain advanced search
- The Stopcriterion 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

TECHNICAL SOLUTIONS

Proof Theory of ILP

Proof Theory of ILP

STI · INNSBRUCK

- 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

- Example: θ-subsumption
  - The background knowledge is supposed to be empty, and the deductive inference rule corresponds to θ-subsumption among single clauses

STI · INNSBRUCK

- θ-subsumes is the simplest model of deduction for ILP which regards clauses as sets of (positive and negative) literals
- A clause c_1 θ-subsumes a clause c_2 if and only if there exists a substitution θ such that $c_1\theta \subseteq c_2$
  - c_1 is called a generalisation of c_2 (and c_2 a specialisation of c_1) under θsubsumption
  - θ-subsumes The θsubsumption inductive inference rule is:

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

- For example, consider:

  $c_1 = \{\ father(X,Y) \leftarrow parent(X,Y),\ male(X)\ \}$
  $c_2 = \{\ father(jef,paul) \leftarrow parent(jef,paul),\ parent(jef,ann),\ male(jef),\ female(ann)\ \}$

  With $\theta = \{X = jef,\ Y = paul\}$ c_1 θsubsumes c_2 because

  $\{\ father(jef,paul) \leftarrow parent(jef,\ paul),\ male(jef)\ \} \subseteq$
  $father(jef,paul) \leftarrow parent(jef,paul),\ parent(jef,ann),\ male(jef),\ female(ann)\ \}$

Some properties of θ-subsumption

- θsubsumption has a range of relevant properties

- Example: Implication
- If c_1 θ-subsumes c_2, then $c_1 \vDash c_2$
  - Example: See previous slide

- This property is relevant because typical ILP systems aim at deriving a hypothesis H (a set of clauses) that implies the facts in conjunction with a background theory B, i.e. $B \cup H \vDash E^+$
  - Because of the implication property, this is achieved when all the clauses in E^+ are θ-subsumed by clauses in $B \cup H$

- Example: Equivalence
- There exist different clauses that are equivalent under θsubsumption
  - E.g. $parent(X,Y) \leftarrow mother(X,Y), mother(X,Z)$ θ-subsumes $parent(X,Y) \leftarrow mother(X,Y)$ and vice versa
  - Two clauses equivalent under θsubsumption are also logically equivalent, i.e. by implication
  - This is used for optimization purposes in practical systems

TECHNICAL SOLUTIONS

ILP Systems

Characteristics of ILP systems

- Incremental/nonincremental: describes the way the evidence E (examples) is obtained
  - In nonincremental or empirical ILP, the evidence is given at the start and not changed afterwards
  - In incremental ILP, the examples are input one by one by the user, in a piecewise fashion.

- Interactive/ Noninteractive
  - 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 non-interactive

Concrete ILP implementations

- A well known family of related, popular 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

An ILP system: CProgol

STI · INNSBRUCK

- CProgol uses a covering approach: It selects an example to be generalised and finds a consistent clause covering the example

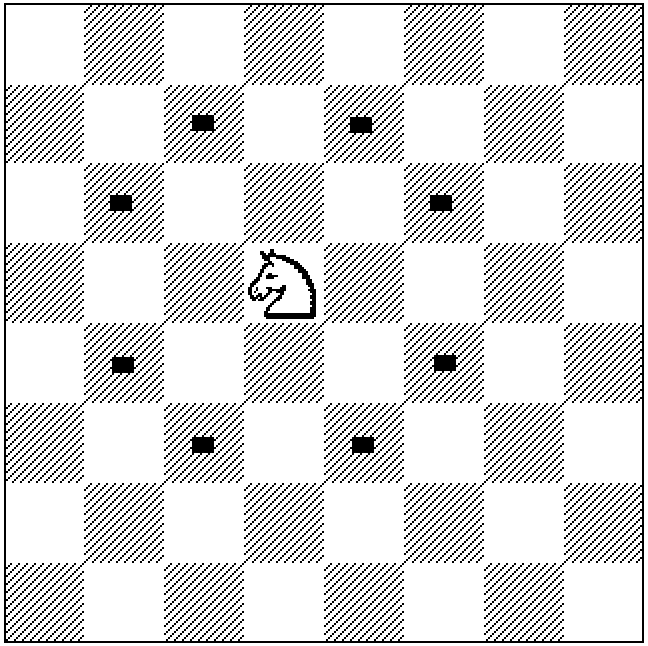
- Basic algorithm for CProgol:
  1. Select an example to be generalized.
  2. Build most-specific-clause. Construct the most specific clause that entails the example selected, and is within language restrictions provided. This is usually a definite clause with many literals, and is called the "bottom 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.

An ILP system: CProgol

- Example: CProgol can be used to learn legal moves of chess pieces (Based on rank and File difference for knight moves)
  - Example included in CProgol distrubtion

- Input:

  % Typespos(b,3),pos(d,2)).
  knight(pos(e,7),pos(f,5)).
  rank(1). rank(2). rank(3). rank(4).
  rank(5). rank(6). rank(7). rank(8).
  knight(pos(c,4),pos(a,5)).
  file(a). file(b). file(c). file(d).
  file(e). file(f). file(g). file(h).
  knight(pos(c,7),pos(e,6)).
  Etc.



An ILP system: CProgol

- Output:

[Result of search is]

knight(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,-2),
invent(q4, E).

[17 redundant clauses retracted]

knight(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,2),
invent(q4,E).

knight(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,1),
invent(q2,E).

knight(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,-1),
invent(q2, E).

knight(pos(A,B),pos(C,D)) :- rdiff(B,D,E), fdiff(A,C,-2),
invent(q4,E).

[Total number of clauses = 4]

[Time taken 0.50s]

Mem out = 822

/cygdrive/c/temp/examples4.4

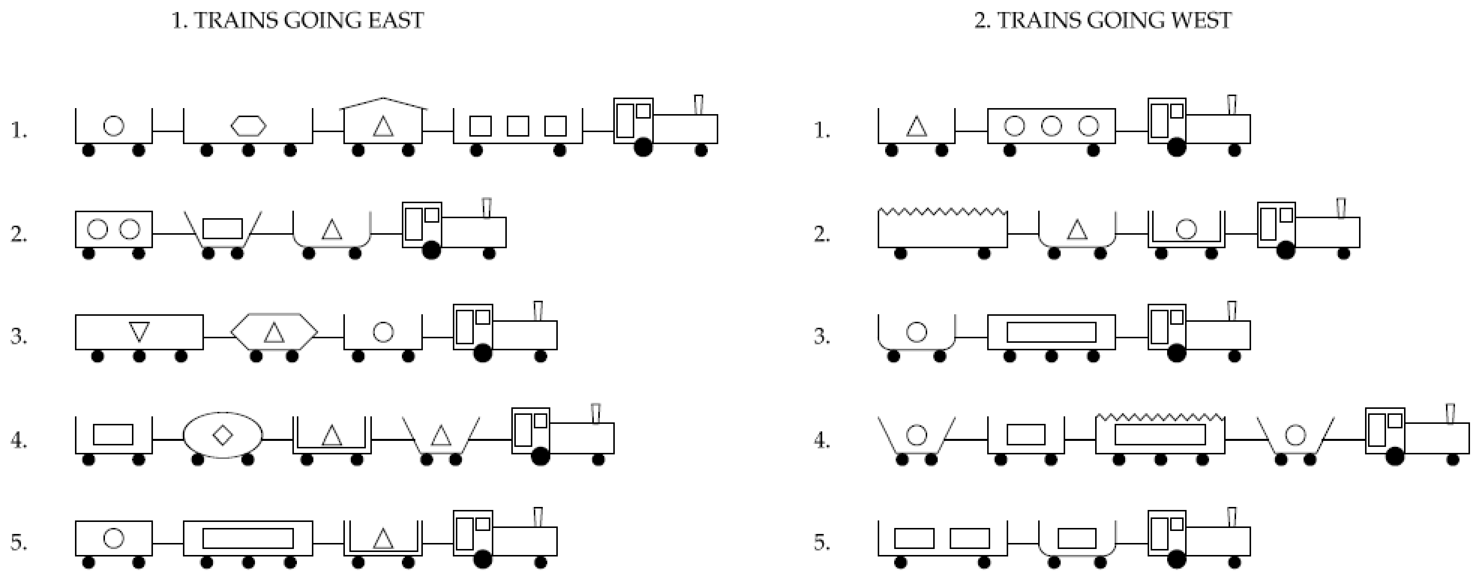
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Progol4.4 demonstration example file.
%
% Predicate invention example for learning legal knight moves.
% Background knowledge contains rank and file difference rather
% than symmetric difference. Progol invents predicates to
% define rank and file difference symmetries for the knight move.
% The target theory is learnable with or without invention, but
% requires smaller samples to learn with invention than without,
% since target expression is more compact with the invented predicates.
%
% See the file `reuchess.pl' for the effect of re-using invented predicates.
%
% The moves of the chess pieces
%
%       Pieces = {King, Queen, Bishop, Knight and Rook}
%
% are learned from examples. Each example is represented by
% a triple from the domain
%
%       Piece x (Rank x File) x (Rank x File)
%
% For instance, the following illustrates a knight (n) move example.
%
%          _______________
%       8|.   .   .   .   |
%       7|  .   .   .   . |
%       6|.   .   .   .   |
%       5|  .   .   .   . |
%       4|.   n   .   .   |
%       3|  .   .   .   . |
%       2|. n .   .   .   |
%       1|  .   .   .   . |
%        |_______________|
%         a b c d e f g h
%
% The only background predicate used is difference, i.e.
%
%       diff(X,Y) = difference between X and Y (either positive or negative)


:- fixedseed?
:- set(h,10000), set(c,5), set(i,2), set(nodes,200)?
:- modeh(1,knight(pos(+file,+rank), pos(+file,+rank)))?
:- modeb(1,rdiff(+rank,+rank,-pmrank))?
:- modeb(1,fdiff(+file,+file,-pmrank))?
:- modeb(1,rdiff(+rank,+rank,#pmrank))?
:- modeb(1,fdiff(+file,+file,#pmrank))?
:- modeb(1,invent(#pn,+pmrank))?

:- constraint(invent/2)?

:- commutative(rdiff/3)?
:- commutative(fdiff/3)?

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Types

rank(1).  rank(2).  rank(3).  rank(4).
rank(5).  rank(6).  rank(7).  rank(8).

file(a).  file(b).  file(c).  file(d).
file(e).  file(f).  file(g).  file(h).

pn(q0). pn(q1). pn(q2). pn(q3). pn(q4). pn(q5). pn(q6). pn(q7). pn(q8).
pn(q9).
pn(q10). pn(q11). pn(q12). pn(q13). pn(q14). pn(q15). pn(q16). pn(q17).
pn(q18). pn(q19).
pn(q20). pn(q21). pn(q22). pn(q23). pn(q24). pn(q25). pn(q26). pn(q27).
pn(q28). pn(q29).
pn(q30). pn(q31). pn(q32). pn(q33). pn(q34). pn(q35). pn(q36). pn(q37).
pn(q38). pn(q39).
pn(q40). pn(q41). pn(q42). pn(q43). pn(q44). pn(q45). pn(q46). pn(q47).
pn(q48). pn(q49).
pn(q50). pn(q51). pn(q52). pn(q53). pn(q54). pn(q55). pn(q56). pn(q57).
pn(q58). pn(q59).
pn(q60). pn(q61). pn(q62). pn(q63). pn(q64). pn(q65). pn(q66). pn(q67).
pn(q68). pn(q69).
pn(q70). pn(q71). pn(q72). pn(q73). pn(q74). pn(q75). pn(q76). pn(q77).
pn(q78). pn(q79).
pn(q80). pn(q81). pn(q82). pn(q83). pn(q84). pn(q85). pn(q86). pn(q87).
pn(q88). pn(q89).
invchess.pl
```

ILLUSTRATION BY A LARGER EXAMPLE

Michalski’s train problem

Michalski's train problem

- 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 (2)

- 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.

Michalski's train problem (3)

- 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).

Michalski's train problem (4)

- 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).

Michalski's train problem (5)

- An ILP systems could generate the following hypothesis:

  $$eastbound(A) \leftarrow has\_car(A,B),\ not(open(B)),\ not(long(B)).$$

  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

Michalski's train problem – Demo

- 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

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

References

- Mandatory Reading:
  - 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.

- Further Reading:
  - N. Lavrac and S. Dzeroski. Inductive Logic Programming: Techniques and Applications. 1994.
    - http://www-ai.ijs.si/SasoDzeroski/ILPBook

- Wikipedia:
  - http://en.wikipedia.org/wiki/Inductive_logic_programming

Next Lecture

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Reasoning
5	Search Methods
6	CommonKADS
7	Problem-Solving Methods
8	Planning
9	Software Agents
10	Rule Learning
11	Inductive Logic Programming
12	**Formal Concept Analysis**
13	Neural Networks
14	Semantic Web and Services

Questions?