Aims

This lecture will introduce you to theoretical and applied aspects of representing hypotheses for machine learning in first-order logic. Following it you should be able to:

- describe the problem of learning relations and some applications
- reproduce the basic algorithm for FOIL
- describe the problem of induction in terms of inverse deduction
- define inverse resolution
- define a generality ordering in terms of $\theta$-subsumption

[Recommended reading: Mitchell, Chapter 10]
[Recommended exercises: 10.5 – 10.7 (10.8)]
Representation in Propositional Logic

Propositional variables: $P, Q, R, \ldots$
Negation: $\neg S, \neg T, \ldots$
Logical connectives: $\land, \lor, \leftarrow, \leftrightarrow$
Well-formed formulae: $P \lor Q, (\neg R \land S) \rightarrow T$, etc.
Inference rules:

- **modus ponens**: Given $B$ and $A \leftarrow B$ infer $A$
- **modus tollens**: Given $\neg A$ and $A \leftarrow B$ infer $\neg B$

Enable **sound** or **valid** inference.

Meaning in Propositional Logic

Propositional variables stand for declarative sentences (properties):
$P$ the paper is red
$Q$ the solution is acid
Potentially useful inferences:
$P \rightarrow Q$ If the paper is red then the solution is acid
Meaning of such formulae can be understood with a **truth table**:

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P \rightarrow Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

Representation in First-Order Predicate Logic

We have a richer language for developing formulae:
constant symbols: Fred, Jane, Copper, Manganese, \ldots
function symbols: Cons, Succ, \ldots
variable symbols: $x, y, z, \ldots$
predicate symbols: Parent, Likes, Binds, \ldots
We still have:
Negation: $\neg$Likes(Bob, Footy), \ldots
Logical connectives: $\land, \lor, \leftarrow, \leftrightarrow$
but we also have quantification:
$\forall x$Likes($x, Fred$), $\exists y$Binds(Copper, $y$)
And we still have well-formed formulae and inference rules \ldots

Meaning in First-Order Logic

Same basic idea as propositional logic, but more complicated.
Give meaning to first-order logic formulae by **interpretation** with respect to a given domain $D$ by associating
- each constant symbol with some element of $D$
- each $n$-ary function symbol with some function from $D^n$ to $D$
- each $n$-ary predicate symbol with some relation in $D^n$
For variables, essentially consider associating all or some domain elements in the formula, depending on quantification.
Interpretation is association of a formula with a truth-valued statement about the domain.
Learning First Order Rules

Why do that?

- Can learn sets of rules such as

  $\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, y)$
  $\text{Ancestor}(x, y) \leftarrow \text{Parent}(x, z) \wedge \text{Ancestor}(z, y)$

- General purpose programming language PROLOG: programs are sets of such rules

Prolog definitions for relational concepts

Some Prolog syntax:

- all predicate and constant names begin with a lower-case letter
  - predicate (relation) names, e.g. uncle, adjacent
  - constant names, e.g. fred, banana
- all variable names begin with an upper-case letter
  - X, Y, Head, Tail
- a predicate is specified by its name and arity (number of arguments), e.g.
  - male/1 means the predicate “male” with one argument
  - sister/2 means the predicate “sister of” with two arguments

- predicates are defined by sets of clauses, each with that predicate in its head
  - e.g. the recursive definition of ancestor/2

  \[
  \text{ancestor}(X, Y) \ :- \ \text{parent}(X, Y).
  \text{ancestor}(X, Y) \ :- \ \text{parent}(X, Z), \ \text{ancestor}(Z, Y).
  \]

- clause head, e.g. ancestor/2, is to the left of the ':-'
- clause body, e.g. parent(X,Z), ancestor(Z,Y), is to the right of the ':-'

- each instance of a relation name in a clause is called a literal
- a definite clause has exactly one literal in the clause head
- a Horn clause has at most one literal in the clause head
- Prolog programs are sets of Horn clauses
- Prolog is a form of logic programming
First Order Rule for Classifying Web Pages

[Slattery, 1997]

course(A) ←
    has-word(A, instructor),
    not has-word(A, good),
    link-from(A, B),
    has-word(B, assign),
    not link-from(B, C)

Train: 31/31, Test: 31/34

Can learn graph-type representations.

Learning First Order Rules

• to learn logic programs we can adopt propositional rule learning methods
• the target relation is clause head, e.g. ancestor/2
    – think of this as the consequent
• the clause body is constructed using predicates from background knowledge
    – think of this as the antecedent
• unlike propositional rules first order rules can have
    – variables
    – tests on more than one variable at a time
    – recursion
• learning is set up as a search through the hypothesis space of first order rules

FOIL(Target predicate, Predicates, Examples)

Pos := positive Examples
Neg := negative Examples

while Pos, do
    // Learn a NewRule
    NewRule := most general rule possible
    NewRuleNeg := Neg
    while NewRuleNeg, do
        // Add a new literal to specialize NewRule
        Candidate_literals := generate candidates
        Best_literal := argmax_{L \in Candidate_literals} Foil_Gain(L, NewRule)
        add Best literal to NewRule preconditions
        NewRuleNeg := subset of NewRuleNeg that satisfies NewRule preconditions
    Learned_rules := Learned_rules + NewRule
    Pos := Pos − {members of Pos covered by NewRule}

Return Learned_rules

Specializing Rules in FOIL

Learning rule: P(x_1, x_2, ..., x_k) ← L_1 ... L_n Candidate specializations
add new literal of form:

• Q(v_1, ..., v_r), where at least one of the v_i in the created literal must already exist as a variable in the rule.
• Equal(x_j, x_k), where x_j and x_k are variables already present in the rule
• The negation of either of the above forms of literals
Variable Bindings

- A substitution replaces variables by terms
- Substitution $\theta$ applied to literal $L$ is written $L\theta$
- If $\theta = \{x/3, y/z\}$ and $L = P(x, y)$ then $L\theta = P(3, z)$

FOIL bindings are substitutions mapping each variable to a constant:

- $\text{GrandDaughter}(x, y) \rightarrow$

With 4 constants in our examples we have 16 possible bindings:

$\{x/\text{Victor}, y/\text{Sharon}\}, \{x/\text{Victor}, y/\text{Bob}\}, \ldots$

With 1 positive example of GrandDaughter, other 15 bindings are negative:

- $\text{GrandDaughter}(\text{Victor}, \text{Sharon})$

Information Gain in FOIL

$$\text{Foil\_Gain}(L, R) \equiv t \left( \log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

Where

- $L$ is the candidate literal to add to rule $R$
- $p_0 = \text{number of positive bindings of } R$
- $n_0 = \text{number of negative bindings of } R$
- $p_1 = \text{number of positive bindings of } R + L$
- $n_1 = \text{number of negative bindings of } R + L$
- $t$ is the number of positive bindings of $R$ also covered by $R + L$

Note

- $-\log_2 \frac{p_0}{p_0 + n_0}$ is minimum number of bits to identify an arbitrary positive binding among the bindings of $R$
- $-\log_2 \frac{p_1}{p_1 + n_1}$ is minimum number of bits to identify an arbitrary positive binding among the bindings of $R + L$
- $\text{Foil\_Gain}(L, R)$ measures the reduction due to $L$ in the total number of bits needed to encode the classification of all positive bindings of $R$

Completeness and Consistency (Correctness)

$H$: complete, consistent

$covers(H, \mathcal{E})$

$H$: incomplete, consistent

$covers(H, \mathcal{E})$
Completeness and Consistency (Correctness)

\[ \mathcal{H}: \text{complete, inconsistent} \]
\[ \mathcal{H}: \text{incomplete, inconsistent} \]

\[ \text{covers}(\mathcal{H}, \mathcal{E}) \]

\[ \text{Target Predicate: ancestor} \]

\[ \text{New clause: ancestor}(X,Y) :- \]
\[ \text{Best antecedent: parent}(X,Y) \quad \text{Gain: 31.02} \]
\[ \text{Learned clause: ancestor}(X,Y) :- \text{parent}(X,Y). \]

\[ \text{New clause: ancestor}(X,Y) :- \]
\[ \text{Best antecedent: parent}(Z,Y) \quad \text{Gain: 13.65} \]
\[ \text{Best antecedent: ancestor}(X,Z) \quad \text{Gain: 27.86} \]
\[ \text{Learned clause: ancestor}(X,Y) :- \text{parent}(Z,Y), \]
\[ \text{ancestor}(X,Z). \]

Definition: ancestor(X,Y) :- parent(X,Y).
\[ \text{ancestor}(X,Y) :- \text{parent}(Z,Y), \]
\[ \text{ancestor}(X,Z). \]

Learning with FOIL

Background Family Tree

Fred - Mary
Alice - Tom Bob - Cindy
John - Barb Ann - Frank
Carol Ted

Target Predicate: ancestor

FOIL Example

0 1 2 3 4 5 6 7 8
x y represents \( \text{LinkedTo}(x,y) \)
**FOIL Example**

Instances:
- pairs of nodes, e.g. \( (1, 5) \), with graph described by literals \( \text{LinkedTo}(0,1) \), \( \neg \text{LinkedTo}(0,8) \) etc.

Target function:
- \( \text{CanReach}(x,y) \) true iff directed path from \( x \) to \( y \)

Hypothesis space:
- Each \( h \in H \) is a set of Horn clauses using predicates \( \text{LinkedTo} \) (and \( \text{CanReach} \))

**FOIL as a propositional learner**

- target predicate is usual form of class value and attribute values
  - \( \text{Class1}(V_1, V_2, \ldots, V_m), \text{Class2}(V_1, V_2, \ldots, V_m), \ldots \)
- literals restricted to those in typical propositional learners
  - \( V_i = \text{const}, V_i > \text{num}, V_i \leq \text{num} \)
- plus extended set
  - \( V_i = V_j, V_i \geq V_j \)
- FOIL results vs C4.5
  - accuracy competitive, especially with extended literal set
  - FOIL required longer computation
  - C4.5 more compact, i.e. better pruning

**FOIL learns Prolog programs from examples**

- from I. Bratko’s book “PROLOG Programming for Artificial Intelligence”
- introductory list programming problems
- training sets by randomly sampling from universe of 3 and 4 element lists
- FOIL learned most predicates completely and correctly
  - some predicates learned in restricted
  - some learned in more complex form than in book
  - most learned in few seconds, some much longer
Identifying document components

• Problem: learn rules to locate logical components of documents
• documents have varying numbers of components
• relationships (e.g. alignment) between pairs of components
• inherently relational task
• target relations to identify sender, receiver, date, reference, logo.

background knowledge
– 20 single page documents
– 244 components
– 57 relations specifying
  * component type (text or picture)
  * position on page
  * alignment with other components
• test set error from 0% to 4%

Induction as Inverted Deduction

Induction is finding $h$ such that

$$(\forall \langle x_i, f(x_i) \rangle \in D) B \land h \land x_i \vdash f(x_i)$$

where

• $x_i$ is $i$th training instance
• $f(x_i)$ is the target function value for $x_i$
• $B$ is other background knowledge

So let’s design inductive algorithm by inverting operators for automated deduction!
“pairs of people, \((u, v)\) such that child of \(u\) is \(v\),”

\[ f(x_i) : \text{Child}(Bob, Sharon) \]
\[ x_i : \text{Male}(Bob), \text{Female}(Sharon), \text{Father}(Sharon, Bob) \]
\[ B : \text{Parent}(u, v) \leftarrow \text{Father}(u, v) \]

What satisfies \((\forall \langle x_i, f(x_i) \rangle \in D \) \(B \land h \land x_i \vdash f(x_i)\)?

\[ h_1 : \text{Child}(u, v) \leftarrow \text{Father}(v, u) \]
\[ h_2 : \text{Child}(u, v) \leftarrow \text{Parent}(v, u) \]

Induction is, in fact, the inverse operation of deduction, and cannot
be conceived to exist without the corresponding operation, so that
the question of relative importance cannot arise. Who thinks of
asking whether addition or subtraction is the more important process
in arithmetic? But at the same time much difference in difficulty
may exist between a direct and inverse operation; ... it must be
allowed that inductive investigations are of a far higher degree of
difficulty and complexity than any questions of deduction.

(W.S. Jevons, 1874)

We have mechanical deductive operators \(F(A, B) = C\), where \(A \land B \vdash C\)
need inductive operators

\[ O(B, D) = h \text{ where } (\forall \langle x_i, f(x_i) \rangle \in D \) \(B \land h \land x_i \vdash f(x_i)\)

Positives:

- Subsumes earlier idea of finding \(h\) that “fits” training data
- Domain theory \(B\) helps define meaning of “fit” the data
  \[ B \land h \land x_i \vdash f(x_i) \]
- Suggests algorithms that search \(H\) guided by \(B\)
Induction as Inverted Deduction

Negatives:

- Doesn’t allow for noisy data. Consider
  \[(\forall (x_i, f(x_i)) \in D) (B \land h \land x_i) \vdash f(x_i)\]

- First order logic gives a huge hypothesis space \(H\)
  → overfitting...
  → intractability of calculating all acceptable \(h\)'s

Deduction: Resolution Rule

\[
P \lor L \\
\neg L \lor R
\]

1. Given initial clauses \(C_1\) and \(C_2\), find a literal \(L\) from clause \(C_1\) such that \(\neg L\) occurs in clause \(C_2\).

2. Form the resolvent \(C\) by including all literals from \(C_1\) and \(C_2\), except for \(L\) and \(\neg L\). More precisely, the set of literals occurring in the conclusion \(C\) is

\[
C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})
\]

where \(\cup\) denotes set union, and “−” denotes set difference.

Inverting Resolution

1. Given initial clauses \(C_1\) and \(C\), find a literal \(L\) that occurs in clause \(C_1\), but not in clause \(C\).

2. Form the second clause \(C_2\) by including the following literals

\[
C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}
\]

Inverting Resolution (Propositional)

1. Given initial clauses \(C_1\) and \(C\), find a literal \(L\) that occurs in clause \(C_1\), but not in clause \(C\).

2. Form the second clause \(C_2\) by including the following literals

\[
C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}
\]

3. Given initial clauses \(C_2\) and \(C\), find a literal \(\neg L\) that occurs in clause \(C_2\), but not in clause \(C\).

4. Form the second clause \(C_1\) by including the following literals

\[
C_1 = (C - (C_2 - \{\neg L\})) \cup \{L\}
\]
Duce operators

<table>
<thead>
<tr>
<th>Op</th>
<th>Same Head</th>
<th>Different Head</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>Identification</td>
<td>Absorption</td>
</tr>
<tr>
<td></td>
<td>( p \leftarrow A, B )</td>
<td>( p \leftarrow A, B )</td>
</tr>
<tr>
<td></td>
<td>( q \leftarrow B )</td>
<td>( q \leftarrow A )</td>
</tr>
<tr>
<td>W</td>
<td>Intra-construction</td>
<td>Inter-construction</td>
</tr>
<tr>
<td></td>
<td>( p \leftarrow A, B )</td>
<td>( p \leftarrow w, B )</td>
</tr>
<tr>
<td></td>
<td>( w \leftarrow B )</td>
<td>( w \leftarrow A )</td>
</tr>
<tr>
<td></td>
<td>( p \leftarrow A, B )</td>
<td>( p \leftarrow w, B )</td>
</tr>
<tr>
<td></td>
<td>( p \leftarrow A, B )</td>
<td>( p \leftarrow w, B )</td>
</tr>
<tr>
<td></td>
<td>( w \leftarrow A )</td>
<td>( w \leftarrow A )</td>
</tr>
</tbody>
</table>

Each operator is read as: pre-conditions on left, post-conditions on right.

First order resolution

1. Find a literal \( L_1 \) from clause \( C_1 \), literal \( L_2 \) from clause \( C_2 \), and substitution \( \theta \) such that \( L_1 \theta = \neg L_2 \theta \)
2. Form the resolvent \( C \) by including all literals from \( C_1 \theta \) and \( C_2 \theta \), except for \( L_1 \theta \) and \( \neg L_2 \theta \). More precisely, the set of literals occurring in the conclusion \( C \) is

\[
C = (C_1 - \{L_1\}) \theta \cup (C_2 - \{L_2\}) \theta
\]

Inverting First order resolution

Factor \( \theta \)

\[
C = (C_1 - \{L_1\}) \theta_1 \cup (C_2 - \{L_2\}) \theta_2
\]

\( C_2 \) should have no common literals with \( C_1 \)

\[
C - (C_1 - \{L_1\}) \theta_1 = (C_2 - \{L_2\}) \theta_2
\]

By definition of resolution \( L_2 = \neg L_1 \theta_1 \theta_2^{-1} \)

\[
C_2 = (C - (C_1 - \{L_1\}) \theta_1 \theta_2^{-1} \cup \{\neg L_1 \theta_1 \theta_2^{-1}\}
\]

Cigol

\[
\text{Father}(Tom, Bob) \quad \text{GrandChild}(y, x) \lor \neg \text{Father}(x, z) \lor \neg \text{Father}(z, y)
\]

\[
\{Bob/y, Tom/z\}
\]

\[
\text{Father}(Shannon, Tom) \quad \text{GrandChild}(Bob, x) \lor \neg \text{Father}(x, Tom)
\]

\[
\{Shannon/x\}
\]

\[
\text{GrandChild}(Bob, Shannon)
\]
Subsumption and Generality

$\theta$-subsumption $C$ $\theta$-subsumes $D$ if there is a substitution $\theta$ such that $C_\theta \subseteq D$.

$C$ is at least as general as $D$ ($C \leq D$) if $C$ $\theta$-subsumes $D$.

If $C$ $\theta$-subsumes $D$ then $C$ logically entails $D$ (but not the reverse).

$\theta$-subsumption is a partial order, thus generates a lattice in which any two clauses have a least-upper-bound and a greatest-lower-bound.

The least general generalisation (LGG) of two clauses is their least-upper-bound in the $\theta$-subsumption lattice.

LGG

- LGG of clauses is based on LGGs of literals
- Lgg of literals is based on LGGs of terms, i.e. constants and variables
- LGG of two constants is a variable, i.e. a minimal generalisation

LGG of clauses

The LGG of two clauses is formed by taking the LGGs of each literal in $C_1$ with every literal in $C_2$.

Relative LGGs with respect to background knowledge.

There's much more to it than this!

LGG of atoms

Two atoms are compatible if they have the same predicate symbol and arity (number of arguments)

$lgg(a, b)$ for different constants or functions with different function symbols is the variable $X$

$lgg(f(a_1, ..., a_n), f(b_1, ..., b_n))$ is $f(lgg(a_1, b_1), ..., lgg(a_n, b_n))$

$lgg(Y_1, Y_2)$ for variables $Y_1, Y_2$ is the variable $X$

Note

1. must ensure that the same variable appears everywhere its bound arguments do in the atom
2. must ensure introduced variables appear nowhere in the original atoms
Subsumption lattice

Text applications of first-order logic in learning

Q: why use first-order logic in machine learning?
A: when relations are important.

Representation for text

Example: text categorization, i.e. assign a document to one of a finite set of categories.

Propositional learners:

- use a “bag-of-words”, often with frequency-based measures
- disregards word order, e.g. equivalence of

  \begin{align*}
  \text{That’s true, I did not do it} \\
  \text{That’s not true, I did do it}
  \end{align*}

First-order learners: word-order predicates in background knowledge

\begin{align*}
\text{has\_word(Doc, Word, Pos)} \\
\text{Pos1 < Pos2}
\end{align*}

Learning information extraction rules

What is information extraction? fill a pre-defined template from a given text.

Partial approach to finding meaning of documents.

Given: examples of texts and filled templates

Learn: rules for filling template slots based on text
SOFTWARE PROGRAMMER

Position available for Software Programmer experienced in generating software for PC-Based Voice Mail systems. Experienced in C Programming. Must be familiar with communicating with and controlling voice cards; preferable Dialogic, however, experience with others such as Rhetorix and Natural Microsystems is okay. Prefer 5 years or more experience with PC Based Voice Mail, but will consider as little as 2 years. Need to find a Senior level person who can come on board and pick up code with very little training. Present Operating System is DOS. May go to OS-2 or UNIX in future.

Please reply to:
Kim Anderson
AdNET
(901) 458-2888 fax
kimander@memphisonline.com
A learning method for Information Extraction

Example rules from text to fill the city slot in a job template:

“... located in Atlanta, Georgia.”
“... offices in Kansas City, Missouri.”

Pre-Filler Pattern  Filler Pattern  Post-Filler Pattern
1) word: in 1) list: max length: 2 1) word: ,
tag: npntag: ,
tag: nnp
tag: np
semantic: state
semantic: state

where np denotes a proper noun (syntax) and state is a general label from the WordNet ontology (semantics).

Prolog

PROGOL: Reduce combinatorial explosion by generating most specific acceptable h as lower bound on search space

1. User specifies H by stating predicates, functions, and forms of arguments allowed for each
2. PROGOL uses sequential covering algorithm.
   For each (x_i, f(x_i))
   - Find most specific hypothesis h_i s.t. B ∧ h_i ∧ x_i ⊢ f(x_i)
     - actually, considers only k-step entailment
3. Conduct general-to-specific search bounded by specific hypothesis h_i, choosing hypothesis with minimum description length

Protein structure

fold('Four-helical up-and-down bundle',P) :-
  helix(P,H1),
  length(H1,hi),
  position(P,H1,Pos),
  interval(1 <= Pos <= 3),
  adjacent(P,H1,H2),
  helix(P,H2).

"The protein P has fold class 'Four-helical up-and-down bundle' if it contains a long helix H1 at a secondary structure position between 1 and 3 and H1 is followed by a second helix H2".
Protein structure classification

- Protein structure largely driven by careful inspection of experimental data by human experts
- Rapid production of protein structures from structural-genomics projects
- Machine-learning strategy that automatically determines structural principles describing 45 classes of fold
- Rules learnt were both statistically significant and meaningful to protein experts

A. Cootes, S.H. Muggleton, and M.J.E. Sternberg
available at http://www.doc.ic.ac.uk/~shm/jnl.html

Immunoglobulin:–

Has antiparallel sheets B and C; B has 3 strands, topology 123; C has 4 strands, topology 2134.

TIM barrel:–

Has between 5 and 9 helices; Has a parallel sheet of 8 strands.

SH3:–

Has an antiparallel sheet B. C and D are the 1st and 4th strands in the sheet B respectively. C and D are the end strands of B and are 4.360 (+/- 2.18) angstroms apart. D contains a proline in the c-terminal end.

Summary

- can be viewed as an extended approach to rule learning
- BUT: much more ...
- learning in a general-purpose programming language
- use of rich background knowledge
- incorporate arbitrary program elements into clauses (rules)
- background knowledge can grow as a result of learning
- control search with declarative bias
- learning probabilistic logic programs