Learning First Order Rules

Lecture Outline:

- Why Learn First Order Rules?
- First Order Logic: Terminology
- The FOIL Algorithm

Reading:

Chapter 10.4-5 of Mitchell

Why Learn First Order Rules?

- Propositional logic allows the expression of individual propositions and their truth-functional combination.
  - E.g. propositions like *Tom is a man* or *All men are mortal* may be represented by single proposition letters such as P or Q (so, proposition letters may be viewed as variables which range over propositions)
  - Truth functional combinations are built up using connectives, such as $\wedge$, $\vee$, $\neg$, $\rightarrow$ – e.g. $P \wedge Q$
  - Inference rules are defined over propositional forms – e.g.

$$\begin{array}{l} P \rightarrow Q \\ \underline{P \qquad\qquad} \\ Q \end{array}$$

  - Note that if P is *Tom is a man* and Q is *All men are mortal*, then the inference that *Tom is mortal* does **not** follow in propositional logic

Why Learn First Order Rules?

- First order logic allows the expression of propositions and their truth functional combination, but it also allows us to represent propositions as assertions of predicates about individuals or sets of individuals
  - E.g. propositions like *Tom is a man* or *All men are mortal* may be represented by predicate-argument representations such as $man(tom)$ or $\forall x(man(x) \rightarrow mortal(x))$ (so, variables range over individuals)
  - Inference rules permit conclusions to be drawn about sets/individuals – e.g. $mortal(tom)$

Why Learn First Order Rules? (cont)

- First order logic is much more *expressive* than propositional logic – i.e. it allows a finer-grain of specification and reasoning when representing knowledge
- In the context of machine learning, consider learning the **relational** concept $daughter(x,y)$ defined over pairs of persons x, y, where
  - persons are represented by attributes: $\langle Name, Mother, Father, Male, Female \rangle$

Why Learn First Order Rules? (cont)

- First order logic is much more *expressive* than propositional logic – i.e. it allows a finer-grain of specification and reasoning when representing knowledge
- In the context of machine learning, consider learning the **relational** concept $daughter(x,y)$ defined over pairs of persons x, y, where
  - persons are represented by attributes: $\langle Name, Mother, Father, Male, Female \rangle$
- Training examples then have the form: $\langle person_1, person_2, target\_attribute\_value \rangle$

  E.g. $\langle\langle Name_1 = Ann, Mother_1 = Sue, Father_1 = Bob, Male_1 = F, Female_1 = T\rangle$

  $\langle Name_2 = Bob, Mother_2 = Gill, Father_2 = Joe, Male_2 = T, Female_2 = F\rangle$

  $Daughter_{1,2} = T\rangle$

Why Learn First Order Rules? (cont)

- First order logic is much more *expressive* than propositional logic – i.e. it allows a finer-grain of specification and reasoning when representing knowledge
- In the context of machine learning, consider learning the **relational** concept $daughter(x, y)$ defined over pairs of persons x, y, where
  - persons are represented by attributes: $\langle Name, Mother, Father, Male, Female\rangle$
- Training examples then have the form: $\langle person_1, person_2, target\_attribute\_value\rangle$

  E.g. $\langle\langle Name_1 = Ann, Mother_1 = Sue, Father_1 = Bob, Male_1 = F, Female_1 = T\rangle$
  $\langle Name_2 = Bob, Mother_2 = Gill, Father_2 = Joe, Male_2 = T, Female_2 = F\rangle$
  $Daughter_{1,2} = T\rangle$
- From such examples, a propositional rule learner such as ID3 or CN2 can only learn rules like:

  IF $(Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = T)$
  THEN $Daughter_{1,2} = T$

  This will not be useful in classifying future pairs of persons.

Why Learn First Order Rules? (cont)

- First order logic is much more *expressive* than propositional logic – i.e. it allows a finer-grain of specification and reasoning when representing knowledge
- In the context of machine learning, consider learning the **relational** concept $daughter(x,y)$ defined over pairs of persons x, y, where
  - persons are represented by attributes: $\langle Name, Mother, Father, Male, Female \rangle$
- Training examples then have the form: $\langle person_1, person_2, target\_attribute\_value \rangle$

  E.g. $\langle\langle Name_1 = Ann, Mother_1 = Sue, Father_1 = Bob, Male_1 = F, Female_1 = T\rangle$
  $\langle Name_2 = Bob, Mother_2 = Gill, Father_2 = Joe, Male_2 = T, Female_2 = F\rangle$
  $Daughter_{1,2} = T\rangle$
- From such examples, a propositional rule learner such as ID3 or CN2 can only learn rules like:

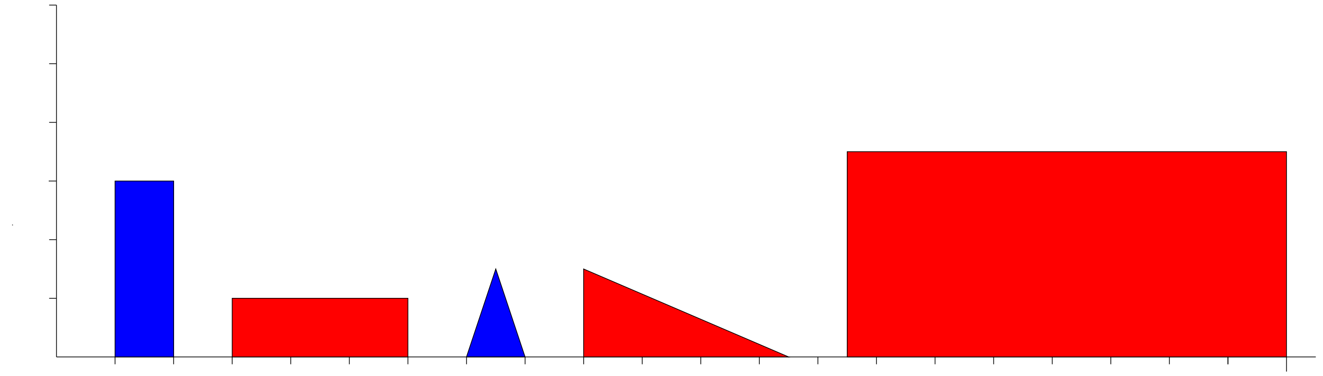
$$\begin{array}{ll} \text{IF} & (Father_1 = Bob) \wedge (Name_2 = Bob) \wedge (Female_1 = T) \\ \text{THEN} & Daughter_{1,2} = T \end{array}$$

  This will not be useful in classifying future pairs of persons.
- In contrast, a first order rule learner can learn the rule:

$$\text{IF } Father(y,x) \wedge Female(y) \text{ THEN } Daughter(x,y)$$

Why Learn First Order Rules? (cont)

- Consider another example. Suppose we want to learn the concept of “standing block”.
- Training instances are represented by 3 attributes: *height*, *width*, *number of sides*.
- In the following data the blue blocks are standing, red ones lying.



- In propositional learning we can test the values of one or more attributes against constants. E.g.
  - IF height(block) $>$ 2 THEN standing – doesn’t work (examples 3 and 5)
  - IF height(block) $>$ 2 and width(block) $<$ 2 THEN standing – doesn’t work (example 3)
- Can learn rules to cover any given finite dataset (effectively learning rules for just these cases), but no test against specific attribute values can capture the generalization we want
- Test we want is: IF height(block) $>$ width(block) THEN standing
- Propositional learners cannot capture such **relations** between attributes

Why Learn First Order Rules? (cont)

- First order rule learners can generalise over relational concepts (which propositional learners cannot).

- In addition, first order rule learners can also acquire **recursive** rules, e.g.:

$$\begin{array}{ll} \text{IF } \mathit{Parent}(x,y) & \text{THEN } \mathit{Ancestor}(x,y) \\ \text{IF } \mathit{Parent}(x,z) \wedge \mathit{Ancestor}(z,y) & \text{THEN } \mathit{Ancestor}(x,y) \end{array}$$

  Note, that variables may be introduced in the rule preconditions that do not occur in the postconditions (*z* in this case)

- Since the rules (Horn clauses) that are learnt by first rule learners are the same form as rules in logic programming languages such as Prolog, such rule learning is called **inductive logic programming (ILP)**

First Order Logic: Terminology

- All expressions in first order logic are composed of:
  - constants – e.g. *bob*, 23, *a*
  - variables – e.g. X, Y, Z
  - predicate symbols – e.g. $female$, $father$ – predicates take on the values $True$ or $False$ only
  - function symbols – e.g. *age* – functions can take on any constant as a value
  - connectives – e.g. $\wedge$, $\vee$, $\neg$, $\rightarrow$ (or $\leftarrow$)
  - quantifiers – e.g. $\forall$, $\exists$

First Order Logic: Terminology

- All expressions in first order logic are composed of:
  - constants – e.g. *bob*, 23, *a*
  - variables – e.g. X, Y, Z
  - predicate symbols – e.g. $female$, $father$ – predicates take on the values $True$ or $False$ only
  - function symbols – e.g. *age* – functions can take on any constant as a value
  - connectives – e.g. $\wedge$, $\vee$, $\neg$, $\rightarrow$ (or $\leftarrow$)
  - quantifiers – e.g. $\forall$, $\exists$
- A **term** is
  - any constant – e.g. *bob*
  - any variable – e.g X
  - any function applied to any term – e.g. $age(bob)$

First Order Logic: Terminology

- All expressions in first order logic are composed of:
  - constants – e.g. *bob, 23, a*
  - variables – e.g. X, Y, Z
  - predicate symbols – e.g. $female$, $father$ – predicates take on the values $True$ or $False$ only
  - function symbols – e.g. *age* – functions can take on any constant as a value
  - connectives – e.g. $\wedge$, $\vee$, $\neg$, $\rightarrow$ (or $\leftarrow$)
  - quantifiers – e.g. $\forall$, $\exists$
- A **term** is
  - any constant – e.g. *bob*
  - any variable – e.g X
  - any function applied to any term – e.g. $age(bob)$
- A **literal** is any predicate or negated predicate applied to any terms – e.g. $female(sue)$, $\neg father(X,Y)$
  - A **ground literal** is a literal that contains no variables – e.g. $female(sue)$
  - A **positive literal** is a literal that does not contain a negated predicate – e.g. $female(sue)$
  - A **negative literal** is a literal that contains a negated predicate – e.g $\neg father(X,Y)$

First Order Logic: Terminology (cont)

- A **clause** is any disjunction of literals $M_1 \vee \cdots \vee M_n$ whose variables are universally quantified (with wide scope)

First Order Logic: Terminology (cont)

- A **clause** is any disjunction of literals $M_1 \vee \cdots \vee M_n$ whose variables are universally quantified (with wide scope)
- A **Horn clause** is any clause containing exactly one positive literal:

$$H \vee \neg L_1 \vee \cdots \vee \neg L_n$$

Since $\neg L_1 \vee \cdots \vee \neg L_n \equiv \neg(L_1 \wedge \cdots \wedge L_n)$

and $(A \vee \neg B) \equiv (A \leftarrow B)$ (read $A \leftarrow B$ as “if B then A”)

then a Horn clause can be equivalently written:

$$H \leftarrow L_1 \wedge \cdots \wedge L_n$$

Note: the equivalent form in Prolog: `H :- L1, ..., Ln.`

First Order Logic: Terminology (cont)

- A **clause** is any disjunction of literals $M_1 \vee \cdots \vee M_n$ whose variables are universally quantified (with wide scope)
- A **Horn clause** is any clause containing exactly one positive literal:

  $$H \vee \neg L_1 \vee \cdots \vee \neg L_n$$

  Since $\neg L_1 \vee \cdots \vee \neg L_n \equiv \neg(L_1 \wedge \cdots \wedge L_n)$

  and $(A \vee \neg B) \equiv (A \leftarrow B)$ (read $A \leftarrow B$ as "if B then A")

  then a Horn clause can be equivalently written:

  $$H \leftarrow L_1 \wedge \cdots \wedge L_n$$

  Note: the equivalent form in Prolog: `H :- L1, ..., Ln.`
- A **substitution** is a function $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ which when applied to an expression C yields a new expression C' with each variable x_i in C replaced with term t_i.
  - $C\theta$ denotes the result of applying θ to C.
  - A **unifying substitution** for two expressions C_1 and C_2 is any substitution θ such that

    $$C_1\theta = C_2\theta$$

First Order Rule Learning: FOIL

- FOIL (Quinlan, 1990) is the natural extension of SEQUENTIAL-COVERING and LEARN-ONE-RULE to first order rule learning.

- FOIL learns first order rules which are similar to Horn clauses with two exceptions:
  - literals may not contain function symbols (reduces complexity of hypothesis space)
  - literals in body of clause may be negated (hence, more expressive than Horn clauses)

- Like SEQUENTIAL-COVERING, FOIL learns one rule at time and removes positive examples covered by the learned rule before attempting to learn a further rule.

- Unlike SEQUENTIAL-COVERING and LEARN-ONE-RULE, FOIL
  - only tries to learn rules that predict when the target literal is true – propositional version sought rules that predicted both true and false values of target attribute
  - performs a simple hill-climbing search (beam search of width one)

FOIL: Overview

FOIL searches its hypothesis space via two nested loops:

FOIL: Overview

FOIL searches its hypothesis space via two nested loops:

- The **outer loop** at each iteration adds a new rule to an overall disjunctive hypothesis (i.e. $rule_1 \vee rule_2 \vee \ldots$)

  This loop may be viewed as a specific-to-general search

  - starting with the empty disjunctive hypothesis which covers no positive instances
  - stopping when the hypothesis is general enough to cover all positive examples

FOIL: Overview

FOIL searches its hypothesis space via two nested loops:

- The **outer loop** at each iteration adds a new rule to an overall disjunctive hypothesis (i.e. $rule_1 \vee rule_2 \vee \ldots$)

  This loop may be viewed as a specific-to-general search

  - starting with the empty disjunctive hypothesis which covers no positive instances
  - stopping when the hypothesis is general enough to cover all positive examples

- The **inner loop** works out the detail of each specific rule, adding conjunctive constraints to the rule precondition on each iteration.

  This loop may be viewed as a general-to-specific search

  - starting with the most general precondition (empty)
  - stopping when the hypothesis is specific enough to exclude all negative examples

FOIL: Algorithm

```
FOIL($Target\_predicate, Predicates, Examples$)

• $Pos \leftarrow$ positive $Examples$
• $Neg \leftarrow$ negative $Examples$
• $Learned\_rules \leftarrow \{\}$
• while $Pos$, do
    Learn a NewRule
    – $NewRule \leftarrow$ most general rule possible (no preconditions)
    – $NewRuleNeg \leftarrow Neg$
    – while $NewRuleNeg$, do
      Add a new literal to specialize NewRule
      1. $Candidate\_literals \leftarrow$ generate candidates based on $Predicates$
      2. $Best\_literal \leftarrow$
                 $\text{argmax}_{L \in Candidate\_literals} Foil\_Gain(L, NewRule)$
      3. add $Best\_literal$ to $NewRule$ preconditions
      4. $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions
    – $Learned\_rules \leftarrow Learned\_rules + NewRule$
    – $Pos \leftarrow Pos - \{$members of $Pos$ covered by $NewRule\}$
• Return $Learned\_rules$
```

FOIL: Explanation

The principal differences between FOIL and SEQUENTIAL-COVERING + LEARN-ONE-RULE are:

- In its inner loop search to generate each new rule, FOIL needs to cope with variables in the rule preconditions
- The performance measure used in FOIL is not the entropy measure used in LEARN-ONE-RULE since
  - the performances of distinct bindings of rule variables need to be distinguished
  - FOIL only tries to discover rules that cover positive examples

FOIL: Specialising the Current Rule

- Suppose we are learning a rule of the form:

$$P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \ldots L_n$$

- Then candidate specialisations add a new literal of the form:
  - $Q(v_1, \ldots, v_r)$, where
    - Q is any predicate in the rule or training data;
    - 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; or
  - The negation of either of the above forms of literals

FOIL: Specialising the Current Rule – Example

Suppose we are trying to learn $granddaughter(X,Y)$, given instances described by the predicates $father$ and $female$:

- FOIL starts with the most general rule:

$$granddaughter(X,Y) \leftarrow$$

- The above procedure generates candidate additional literals:

$$equal(X,Y), female(X), female(Y), father(X,Y), father(Y,X),$$
$$father(X,Z), father(Z,X), father(Y,Z), father(Z,Y)$$

  plus the negations of each of these

- Suppose FOIL now chooses $father(Y,Z)$ as most promising:

$$granddaughter(X,Y) \leftarrow father(Y,Z)$$

- New candidate literals to specialise this rule include those from above, plus

$$equal(Z,X), equal(Z,Y), female(Z), female(Y), father(Z,W), father(W,Z)$$

  plus their negations

FOIL: Specialising the Current Rule – Example (cont)

- If FOIL next selects $father(Z,X)$, then $female(Y)$ the rule

$$granddaughter(X,Y) \leftarrow father(Y,Z) \wedge father(Z,X) \wedge female(Y)$$

  is generated.

- Assuming this rule covers only positive examples, no more specialisations for the current rule are sought.

  If more positive examples remain, search for another rule begins.

FOIL: Performance Evaluation Measure

- How do we decide which is the best literal to add when specialising a rule?
- To do this FOIL considers each possible binding of variables in the candidate rule specialisation to constants in the training examples.
- For example, suppose we have the training data:

$$\begin{array}{lll} granddaughter(bill, joan) & father(joan, joe) & father(tom, joe) \\ female(joan) & father(joe, bill) & \end{array}$$

  and we also assume ("closed world assumption") that any literals

  – involving predicates $granddaughter$, $father$, and $female$

  – involving constants $bill$, $joan$, joe, and tom

  – not in the training data

  are false. E.g. $\neg granddaughter(bill, tom)$ is also true.

- Given the initial rule: $granddaughter(X,Y) \leftarrow$

  FOIL considers all possible bindings of X and Y to the constants $bill$, $joan$, joe, and tom.

  Note that only $\{X/bill, Y/joan\}$ is a positive binding (i.e. corresponds to a positive training example). The other 15 bindings of constants to X and Y are negative.

FOIL: Performance Evaluation Measure (cont)

- At each stage of rule specialisation, candidate specialisations are preferred according to whether they possess more positive and fewer negative bindings.

- The precise evaluation measure used by FOIL is:

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

  Where

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

FOIL: Performance Evaluation Measure (cont)

- Note, in information-theoretic terms:
  - $-\log_2 \frac{p_0}{p_0+n_0}$ is minimum number of bits to encode the classification of a positive binding covered by R
  - $-\log_2 \frac{p_1}{p_1+n_1}$ is minimum number of bits to encode the classification of a positive binding covered by $R+L$
  - t is number of positive bindings covered by R that remain covered by $R+L$
  - So, $\mathit{Foil\_Gain}(R,L)$ is reduction due to L in total number of bits required to encode the classification of all positive bindings of R

FOIL: Summary/Observations

- FOIL extends the SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms for propositional rule learning to first order rule learning

FOIL: Summary/Observations

- FOIL extends the SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms for propositional rule learning to first order rule learning
- FOIL learns in two phases:
  - an outer loop which acquires a disjunction of Horn clause-like rules which together cover the positive examples
  - an inner loop which constructs individual rules by progressive specialisation of a rule through adding new literals selected according to the FOIL-gain measure until no negative examples are covered

FOIL: Summary/Observations

- FOIL extends the SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms for propositional rule learning to first order rule learning

- FOIL learns in two phases:
  - an outer loop which acquires a disjunction of Horn clause-like rules which together cover the positive examples
  - an inner loop which constructs individual rules by progressive specialisation of a rule through adding new literals selected according to the FOIL-gain measure until no negative examples are covered

- The literals introduced in the rule preconditions are drawn from the attributes used in describing the training examples
  - Variables used in new literals may be those occurring already in the rule pre-/postconditions, or they may be new
  - If the new literal is allowed to use the target predicate then recursive rules may be learned
    In this case special care must be taken to avoid learning rule sets that produce infinite recursion

FOIL: Summary/Observations

- FOIL can add literals until no negative examples are covered only in the case of noise-free data – otherwise some other strategy must be adopted
  - Quinlan proposed use of a minimum description length measure to stop rule extension when the description length of rules exceeds that of the training data they explain