Scribe Notes on FOIL and Inverted Deduction

by Gabor Melli (melli@sfu.ca)

This document summarizes the two inductive logic programming (ILP) approaches of the FOIL algorithm and induction by inverted deduction (inverted deduction). The information for these notes was drawn from chapters 10.4 through 10.6 in Tom Mitchell's "Machine Learning" textbook, from Oliver Schulte's April 1, 2004 lecture, and from background knowledge.

1 Introduction

Inductive Logic Programming (ILP) is the research area which studies machine learning algorithms that produce logic programs. Typically these logic programs restricted to first-order logic. First order logic is Turing complete so it is more expressive than "propositional" representations used by most other machine learning algorithms. As we will see the need for this level of expressiveness can be necessary for even simple domains such as family relations. A further advantage of ILP is that it is not produce black-box models. Its hypotheses can generally be translated into English text which can then enable knowledge discovery. The ILP approach has also been successfully applied to tasks such as Natural Language Processing, Web click-stream mining, and protein folding pattern discovery.

The document is structured as follows: First we review some relevant logic terminology. An example is then presented to illustrate when ILPs can be useful. Next the FOIL algorithm is described. Finally the topic of induction by inverted deduction is introduced. Some familiarity with the concepts of statistics and propositional rule learners is assumed although a glossary is provided at the end of the document to further support the terminology.

2 First-Order, Terminology

This section addresses one of challenges with the use of first-order logic: its terminology. We limit ourselves to the terminology required to understand the operation of FOIL and inverted deduction. The terminology is divided into three sections:

1) Literals: e.g. *Female*(*Sharon*), ¬*Father*(*Sharon*, *Bob*), *Female*(*X*)

The first example presents three literals. Literals are composed of predicates (e.g. *Female*) that are combined with either constants (e.g. *Sharon*) or variables (e.g. *X*). What predicates do is express the relations between the objects (terms) in the world that is being modeled. Conventionally predicate expression are read starting with the first term and then the predicate. For example, *Father*(*Sharon*, *Bob*) reads "Sharon has father Bob". A literal composed of a predicate and a constant is an assertion. The ¬ symbol reverses a terms truth function (negates).

2) Horn Clauses: e.g. $\forall X \forall Y \quad Daughter(X,Y) \leftarrow Parent(Y,X) \wedge Female(Y)$

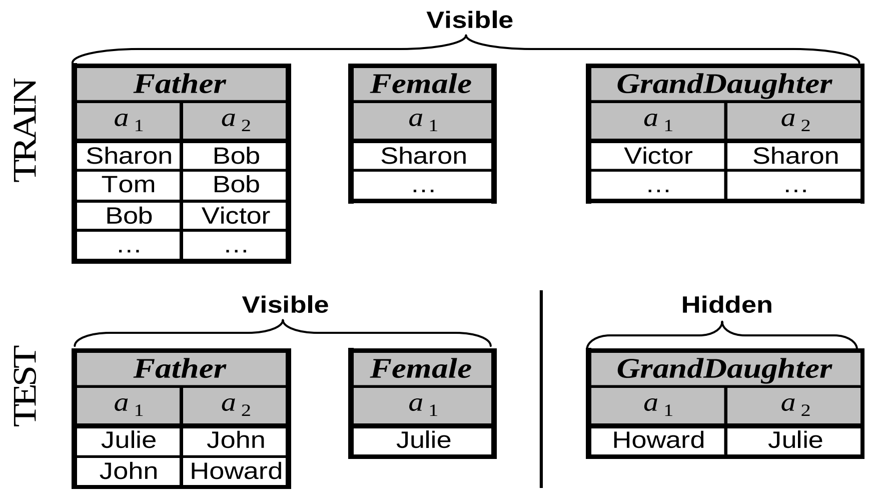
This second example presents a Horn clause. A Horn clause is composed of a head or consequent (e.g. $Daughter(X,Y) \leftarrow$) and a conjunction of one or more positive literals in its body or antecedent (e.g. $Parent(Y,X) \wedge Female(Y)$). The variables in the predicates of a Horn clause (e.g. *X*, *Y*) are always universally quantified. The quantification enables the variables to bind to assertions. The negative bindings includes the observations that do not match the constraints.

3) IF/THEN rule: e.g. `IF` L_1 `and` … L_n `THEN` H

A common way to present the resulting Horn clause theory from an ILP system is by converting the resulting clauses into IF/THEN rules, where the consequent is place after the THEN and the antecedents are place in the IF expression.

3 Example: Familial Relations can be Complicated

One of the typical domains used to demonstrate the need and power of ILP over propositional attribute-value based algorithms is that of familial relations. In the supervised learning example below the learner is challenged to discover the rule that describes whether a person has a granddaughter. The successful model must be able to infer that *Howard* has a *GrandDaughter*, while *Julie* and *John* do not.


Visible
TRAIN
Visible
Hidden
TEST

TRAIN

Father	
a_1	a_2
Sharon	Bob
Tom	Bob
Bob	Victor
…	…

Female
a_1
Sharon
…

GrandDaughter	
a_1	a_2
Victor	Sharon
…	…

TEST

Father	
a_1	a_2
Julie	John
John	Howard

Female
a_1
Julie

GrandDaughter	
a_1	a_2
Howard	Julie

Figure 1 – Sample train dataset from p.288 in Mitchell's textbook. The test set and relational format is not from the textbook. The tables can be read either as "IS A" or "HAS", e.g. Victor has granddaughter Sharon. The hidden relation is used to test model accuracy.

3.1.1 Propositional Learner with simple data transformation.

Before showing how an ILP system would handle this task we demonstrate how a propositional learner would be challenged. One of the first challenges that a propositional learner would encounter with this dataset is that the dataset is not structured as a set of fixed length-vectors of attribute-value pairs. This situation is typically resolved by JOINing the relations, as in Figure 2.

Predictors			**Target**
Father	***Child***	***Child is Fem.***	***Has Gdaugh***
Bob	Sharon	TRUE	FALSE
Victor	Bob	FALSE	TRUE
…	…	…	…

Figure 2 – The resulting dataset after a simple JOIN of the Father, Female, and GrandDaughter data in Figure 1.

A propositional learner would not locate a predictive model for this dataset. It would not be able to state that *Sharon* is *Victor*'s granddaughter. At best it may discover that a child's gender has some influence on the likelihood that that child is a parent, or even a parent to a female child.

3.1.2 Propositional Leaner with complex data transformation

One way to interpret the problem encountered on this dataset by a propositional learner is that the dataset invalidates the assumption that observations are independent and identically distributed.[1] The algorithm cannot make the connection in one observation (*Bob* as a father) and another (*Bob* as child). A common way to enable a propositional learner to produce a predictive model on this data is to transform the data so that the required relations appear as attributes in the data. See Figure 3. This transformation is sometimes referred to as ‘flattening’ the data.

Predictors					**Target**
Father	***Child***	***Child is Fem.***	***Child's Child***	***C's C is Fem.***	***Has Gdaugh***
Bob	Sharon	TRUE	NULL	NULL	FALSE
Victor	Bob	FALSE	Sharon	TRUE	TRUE
…	…	…		…	…

Figure 3 – The resulting dataset after a complex transformation of the Father, Female, GrandDaughter and a second instantiation of the Father relation data in Figure 1. Two new predictor attributes are added: Child’s Child and C’s C is Fem.

Now the search for a rule is trivial. A decision tree would locate the pattern:

```
IF Child’s Child is Female = TRUE
        THEN HasGrandDaughter = TRUE.
        ELSE HasGrandDaughter = FALSE
```

As we will see, the transformation of the data performed in this example is what FOIL does automatically to objectively search out a pattern.

[1] Note that the Father relation breaks the iid assumption commonly required by most simple machine learning algorithms. Knowing that a person, e.g. Bob, is someone’s father tells you that they are also someone’s child (but not every child is a father).

4 The FOIL Algorithm

The Foil algorithm is a supervised learning algorithm that produces rules in first-order logic [Q90] The algorithm is a generalization of the SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms (see Appendix 1). The main modification is that search can also specialize on predicates with variables. The resulting rules differ from Horn clauses in two ways: negated symbols are allowed within the body, and FOIL's rules will not include function symbols.

4.1 FOIL Example

Some intuition into how FOIL operates will be given by way of example before a formal definition is provided. FOIL is a top-down algorithm that starts out with a general rule and explores the search space by greedily specializing the current rule. The diagram in Figure 4 illustrates how the rule for the sample dataset is expanded towards the correct model of *IF Father(y,z) ^ Father(z,x) ^ Female(x) THEN GrandDaughter (x,y).*

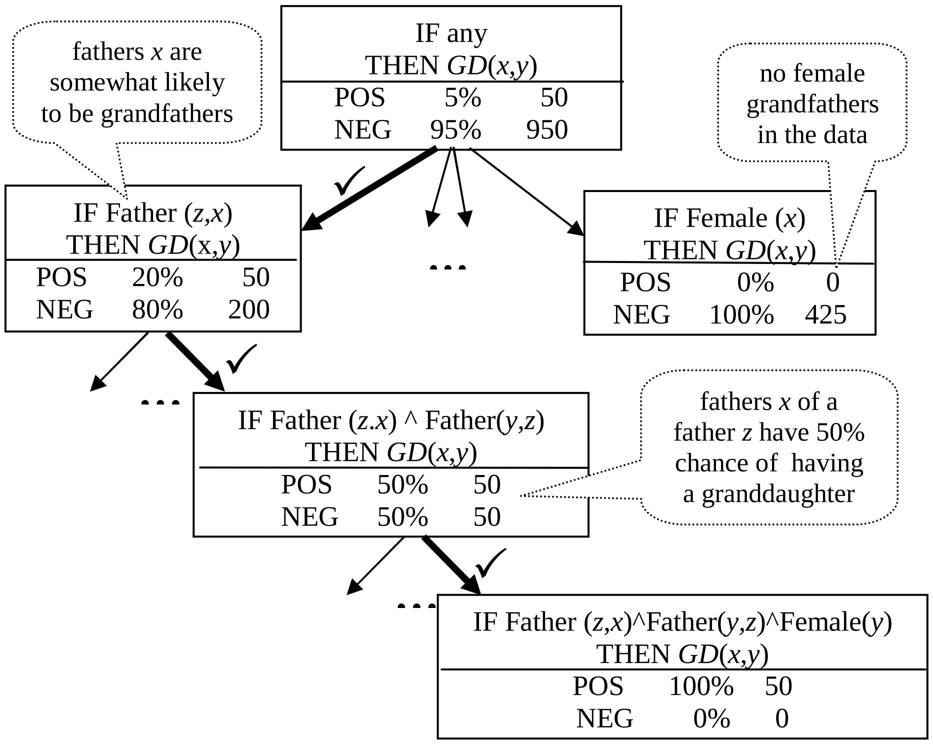
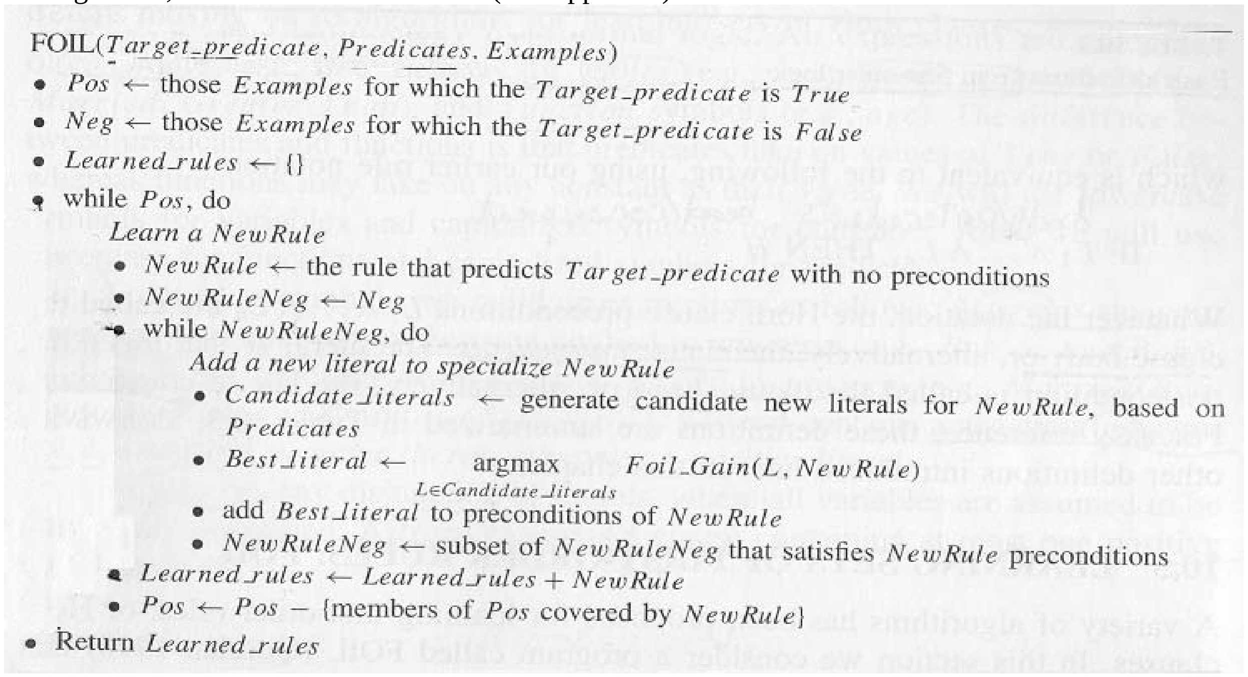

IF any THEN GD(x,y)
POS 5% 50
NEG 95% 950
fathers x are somewhat likely to be grandfathers
no female grandfathers in the data
IF Father (z,x) THEN GD(x,y)
POS 20% 50
NEG 80% 200
IF Female (x) THEN GD(x,y)
POS 0% 0
NEG 100% 425
IF Father (z.x) ^ Father(y,z) THEN GD(x,y)
POS 50% 50
NEG 50% 50
fathers x of a father z have 50% chance of having a granddaughter
IF Father (z,x)^Father(y,z)^Female(y) THEN GD(x,y)
POS 100% 50
NEG 0% 0

Figure 4 – Sample run of FOIL on the Figure 1 dataset. The algorithm starts with the most general rule. In this example the population has 1,000 individuals, 50 of which are granddaughters to someone in the data. As FOIL adds literals to the rule it will greedily choose the rule that increases the proportion of POSitives to NEGatives (marked with ✓).

4.2 FOIL

The FOIL algorithm is summarized in Figure 5. The outer loop adds new rules to the output until no more positive examples are covered. The inner loop searches for the next best rule by incremental specialization. The outer loop corresponds to the SEQUENTIAL-CONVERING algorithm, the inner to FIND-A-RULE (see Appendix).



```
FOIL(Target_predicate, Predicates, Examples)
• Pos ← those Examples for which the Target_predicate is True
• Neg ← those Examples for which the Target_predicate is False
• Learned_rules ← {}
• while Pos, do
    Learn a NewRule
    • NewRule ← the rule that predicts Target_predicate with no preconditions
    • NewRuleNeg ← Neg
    • while NewRuleNeg, do
        Add a new literal to specialize NewRule
        • Candidate_literals ← generate candidate new literals for NewRule, based on Predicates
        • Best_literal ← $\underset{L \in Candidate\_literals}{\text{argmax}}$ Foil_Gain(L, NewRule)
        • add Best_literal to preconditions of NewRule
        • NewRuleNeg ← subset of NewRuleNeg that satisfies NewRule preconditions
    • Learned_rules ← Learned_rules + NewRule
    • Pos ← Pos − {members of Pos covered by NewRule}
• Return Learned_rules
```

Figure 5 – Pseudocode description of the Foil algorithm from [M97].
The definition for *Candidate_literals* and *Foil_Gain* is given below.

4.3 Expand the Search (Candidate_literals)

FOIL expands its search space by specializing rules through the addition of literals to the rule body. If the current rule has rule head $P(x_1,\ldots x_k)$ and body literals of $L_i \ldots L_n$ the following three type of literal additions will be attempted:

1) $Q(v_i,\ldots,v_r)$ where Q is a valid predicate and at least one of variable v_i is already in the rule body.
2) Equal (x_j,x_k), where variables x_j, x_k are already in the rule.
3) The negation of the above: $\neg Q$(v1,…vr), or $\neg Equal(x_j,x_k)$.

4.3.1 Enter Recursion

A more sophisticated form of rule specialization involves the addition of a literal that contains the target predicate. This specialization is necessary to discover rules such as IF Parents(x,z)^**Ancestor(z,y)** THEN Ancestor(x,y). The highlighted literal initiates a recursive description. Some care must be taken to avoid infinite recursion.

4.4 Guiding the Search (Foil_Gain)

FOIL uses a version of the gain algorithm to determine which newly specialized rule to favour. Each rule's utility is estimated by the number of bits required to encode all of the positive bindings.

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

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 binding of $R + L$
- n_1 = number of negative bindings of $R + L$
- t is the number of positive bindings of R also covered by $R + L$

Note that $-\log_2 \frac{p_o}{p_0 + n_0}$ is the optimal number of bits to indicate class of a positive binding covered by R.

4.5 Pruning

As defined the FOIL algorithm attempts to fit any noise found in the data. To be robust to noise FOIL can be updated with pruning to avoid this overfitting. Refer to decision tree notes for further discussion on pruning.

5 Induction as Inverted Deduction

An alternative approach to first-order logic classification is based on the observation that induction is the inverse of deduction. Inverted deduction searches for hypothesis *h* that explains the target *f*(x_i) for all observations x_i in the data *D* and for background knowledge *B*. Formally[2]:

$$(\forall < x_i, f(x_i) > \in D)(B \wedge h \wedge x_i) \mapsto f(x_i)$$

5.1 Example:

Consider the target "pairs of people <*u*, *v*> such that the child of *u* is *v*": Child(*u*,*v*), and:

- *D:* *Male(Bob), Female(Sharon), Father(Sharon, Bob)*
- *f(u,v):* *Child(Bob, Sharon)* *// one positive example*
- *B:* *Parent(u,v) ← Father(v,u)*

Recall that we want to find *h* such that $\forall x_i$ *(B^h^*x_i*)* |-- *f(xi).*

Two candidate hypotheses are:

h1: Child(u,v) ← Father(v,u)

h2: Child(u,v) ← Parents(v,u)

5.2 Background Knowledge

In the example above the background rule *Parent(u,v) ← Father(v,u)* played a role in the search. Most machine learning algorithms do not allow for a declarative statement of background knowledge that would assist the algorithm more quickly locate a better fitting hypothesis. Interestingly this would allow a system to interactively present candidate hypothesis to the user for validation. As the user enters more background knowledge the search could be expanded.

5.3 Practical Difficulties

The application of inverted deduction to induce first-order hypothesis has several practical difficulties.

5.3.1 Intolerance to Noisy Data

Does not work with noisy data because inverted deduction will fit every pattern in the data.

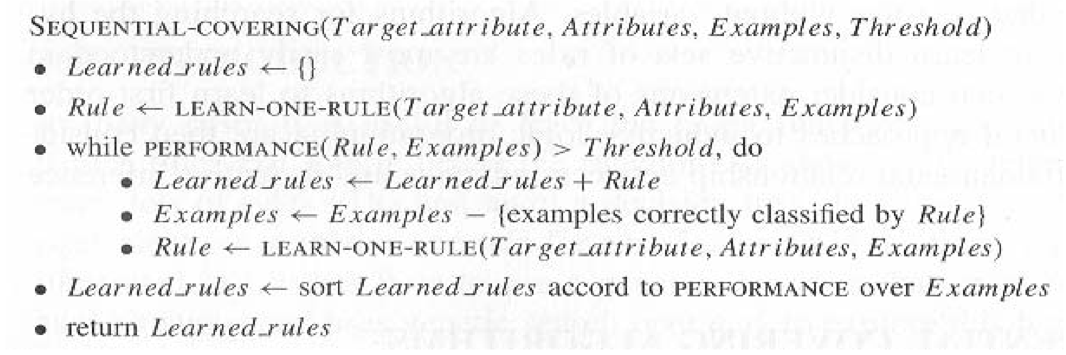
5.3.2 Computational Complexity

The first order logic representation creates an intractably large search space for hypothesis *h*. Unfortunately, the addition of background knowledge also increases the complexity.

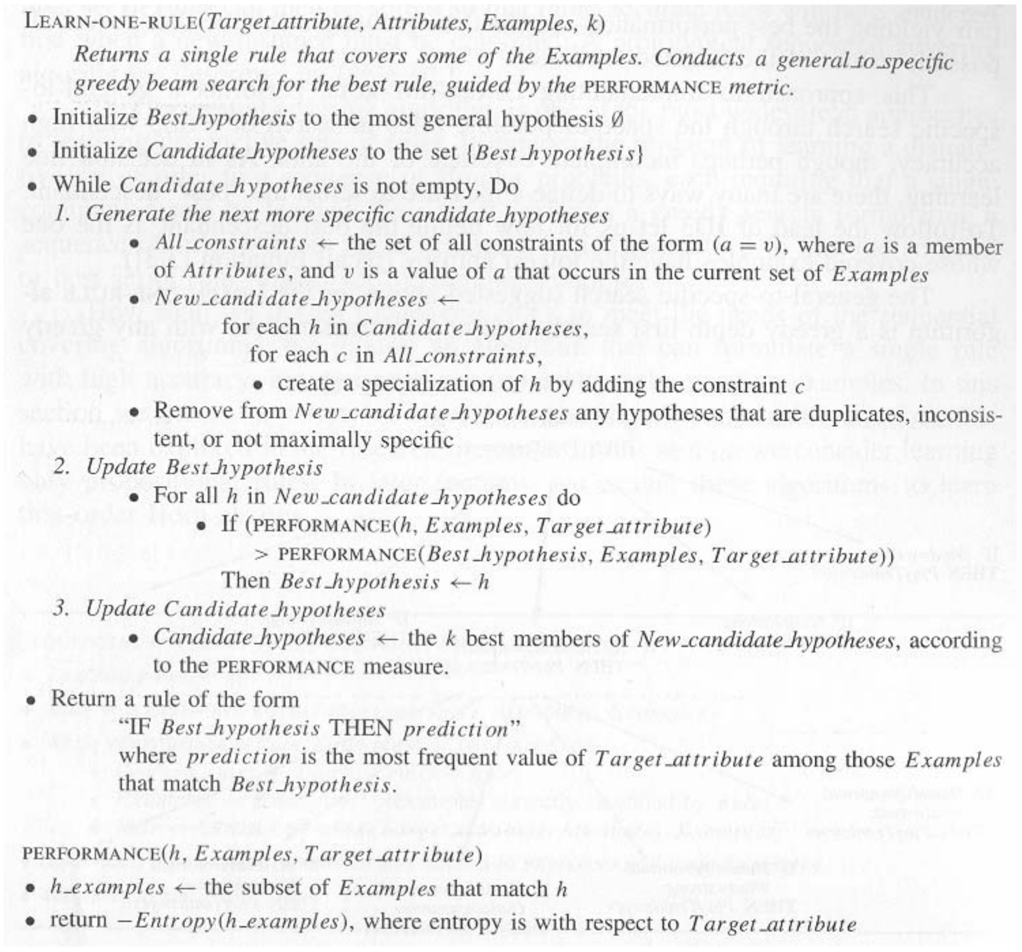
[2] The expression $X \vdash Y$ can be read as *X* "entails" *Y*, or *Y* is "follows from" *X*.

6 Appendix

The SEQUENTIAL-COVERING and LEARN-ONE-RULE algorithms described in chapter 10.2 of Mitchell's textbook are included below in order to review how FOIL generalizes them in order to handle first-order rules.



```
SEQUENTIAL-COVERING($Target\_attribute, Attributes, Examples, Threshold$)
• $Learned\_rules \leftarrow \{\}$
• $Rule \leftarrow$ LEARN-ONE-RULE($Target\_attribute, Attributes, Examples$)
• while PERFORMANCE($Rule, Examples$) $> Threshold$, do
    • $Learned\_rules \leftarrow Learned\_rules + Rule$
    • $Examples \leftarrow Examples -$ {examples correctly classified by $Rule$}
    • $Rule \leftarrow$ LEARN-ONE-RULE($Target\_attribute, Attributes, Examples$)
• $Learned\_rules \leftarrow$ sort $Learned\_rules$ accord to PERFORMANCE over $Examples$
• return $Learned\_rules$
```



```
LEARN-ONE-RULE($Target\_attribute, Attributes, Examples, k$)
  Returns a single rule that covers some of the Examples. Conducts a general_to_specific
  greedy beam search for the best rule, guided by the PERFORMANCE metric.
• Initialize $Best\_hypothesis$ to the most general hypothesis $\emptyset$
• Initialize $Candidate\_hypotheses$ to the set $\{Best\_hypothesis\}$
• While $Candidate\_hypotheses$ is not empty, Do
  1. Generate the next more specific candidate_hypotheses
      • $All\_constraints \leftarrow$ the set of all constraints of the form $(a = v)$, where $a$ is a member
        of $Attributes$, and $v$ is a value of $a$ that occurs in the current set of $Examples$
      • $New\_candidate\_hypotheses \leftarrow$
            for each $h$ in $Candidate\_hypotheses$,
              for each $c$ in $All\_constraints$,
                • create a specialization of $h$ by adding the constraint $c$
      • Remove from $New\_candidate\_hypotheses$ any hypotheses that are duplicates, inconsis-
        tent, or not maximally specific
  2. Update $Best\_hypothesis$
      • For all $h$ in $New\_candidate\_hypotheses$ do
          • If (PERFORMANCE($h, Examples, Target\_attribute$)
                $>$ PERFORMANCE($Best\_hypothesis, Examples, Target\_attribute$))
            Then $Best\_hypothesis \leftarrow h$
  3. Update $Candidate\_hypotheses$
      • $Candidate\_hypotheses \leftarrow$ the $k$ best members of $New\_candidate\_hypotheses$, according
        to the PERFORMANCE measure.
• Return a rule of the form
      "IF $Best\_hypothesis$ THEN $prediction$"
      where $prediction$ is the most frequent value of $Target\_attribute$ among those $Examples$
      that match $Best\_hypothesis$.

PERFORMANCE($h, Examples, Target\_attribute$)
• $h\_examples \leftarrow$ the subset of $Examples$ that match $h$
• return $-Entropy(h\_examples)$, where entropy is with respect to $Target\_attribute$
```

7 Glossary

Black-box Techniques: Are typically good at creating accurate models for supervised learning problems, but their result does not provide knowledge about the system that was modeled. Neural networks, Support Vector Machines, and Instance-Based Learning are typically described as black-box algorithms.

CN2: A supervised classification algorithm that generates decision lists. The algorithm is a specialization of the SEQUENTIAL-COVERING algorithm that uses beam search. See SEQUENTIAL-COVERING.

Definite clause: A clause without negation. See Horn Clause.

First-Order Logic (aka First-Order Predicate Calculus): A logic system that allows quantification on variables. First-order logic is Turing complete. See: Second-Order Logic.

FOIL: A machine learning program proposed by Quinlan [Q90] that discovers patterns in the data expressed as first-order rules. The algorithm is a greedy-search implementation of the SEQUENTIAL-COVERING. See: SEQUENTIAL-COVERING.

Horn Clause: A clause containing at most one positive literal, written: (a definite clause) or (a definite goal).

Inductive Logic Programming (ILP): Inductive Logic Programming (ILP) is the research area that studies machine learning algorithms which produce logic programs, and typically programs restricted to first-order logic. See: FOIL, Horn Clause.

Independence: Two events A and B are statistically independent if the chance that they both happen simultaneously is the product of the chances that each occurs individually: i.e., if $P(A\wedge B) = P(A)P(B)$. Intuitively two events are independent when learning that one of the events occur does not help you determine whether the other event also occurred: i.e., $P(A|B) = P(A)$.

iid: A set of random variables are iid (independent and identically distributed) if they are independent and have the same probability distribution.

Prolog: A programming language which as the name suggest if adapted to "Programming in Logic". Prolog was originally designed by A. Colmerauer and P. Roussel in 1971 for natural-language processing but has since been applied to several other AI problems.

Propositional Learners: A term typically used by the ILP research community for algorithms that are incapable of learning relations between observations, such as *Ancestor*(*x*,*y*). The term captures techniques such as decision trees, neural networks, and instance based learners.

Propositional Logic (Propositional Calculus): A logic system that operates on individual members of the domain. See: First-Order Logic.

Second-Order Logic (aka Second-Order Predicate Calculus): A logic system that allows quantification over predicates. The system is impractical for computational purpose, but is helpful in the theoretical analysis of uncountable theories, such as Cantor sets. See: First-Order Logic.

SEQUENTIAL-COVERING: A supervised classification algorithm that performs a general-to-specific beam search through rule-space. The algorithm removes training examples covered by each discovered rule and then repeats until all the positive examples have been covered. The algorithm does not backtrack so the underlying LEARN-ONE-RULE must be effective. See: CN2, FOIL.

Top-Down Learning: Refers to the technique of starting from a general rule and to proceed by specializing it.

8 References

[1]. P. Clark and R. Niblett. *The CN2 Induction Algorithm*. Machine Learning, 3:261-284, 1989.
[2]. T. M. Mitchell. *Machine Learning*. McGraw Hill. 1997.
[3]. J. Ross Quinlan. *Learning Logical Definitions from Relations*. Machine Learning 5: 1990.