Learning Sets of Rules

- Sequential covering algorithms
- FOIL
- Induction as the inverse of deduction
- Inductive Logic Programming

Learning Disjunctive Sets of Rules

Method 1: Learn decision tree, convert to rules
Method 2: Sequential covering algorithm
1. Learn one rule with high accuracy, any coverage
2. Remove positive examples covered by this rule
3. Repeat

Sequential Covering Algorithm

\[
\text{SEQUENTIAL-COVERING}(\text{Target\_attr}, \text{Attrs}, \text{Examples}, \text{Thresh})
\]

\[
\text{Learned\_rules} \leftarrow \emptyset \\
\text{Rule} \leftarrow \text{LEARN-ONE-RULE}(\text{Target\_attr}, \text{Attrs}, \text{Examples}) \\
\text{while } \text{PERFORMANCE}(\text{Rule}, \text{Examples}) > \text{Thresh} \text{ do} \\
\quad \text{Learned\_rules} \leftarrow \text{Learned\_rules} + \text{Rule} \\
\quad \text{Examples} \leftarrow \text{Examples} - \{\text{examples correctly classified by Rule}\} \\
\quad \text{Rule} \leftarrow \text{LEARN-ONE-RULE}(\text{Target\_attr}, \text{Attrs}, \text{Examples}) \\
\text{Learned\_rules} \leftarrow \text{sort} \text{Learned\_rules according to PERFORMANCE over Examples} \\
\text{return Learned\_rules}
\]

Learn-One-Rule

\[
\begin{align*}
\text{IF} & \quad \text{THEN CoolCar=Yes} \\
\text{IF Type = SUV} & \quad \text{THEN CoolCar=Yes} \\
\text{IF Doors = 4} & \quad \text{THEN CoolCar=Yes} \\
\text{IF Type = Car} & \quad \text{THEN CoolCar=Yes} \\
\text{IF Doors = 2} & \quad \text{THEN CoolCar=Yes} \\
\text{IF Type = SUV AND Color = Red} & \quad \text{THEN CoolCar=Yes} \\
\text{IF Type = SUV AND Doors = 4} & \quad \text{THEN CoolCar=Yes}
\end{align*}
\]

Covering Rules

\[
\begin{align*}
\text{Pos} & \leftarrow \text{positive Examples} \\
\text{Neg} & \leftarrow \text{negative Examples} \\
\text{while } \text{Pos} \text{ do (Learn a New Rule)} \\
\text{NewRule} & \leftarrow \text{most general rule possible} \\
\text{NegExamplesCovered} & \leftarrow \text{Neg} \\
\text{while NegExamplesCovered do} \\
\quad \text{Add a new literal to specialize NewRule} \\
\quad \text{Best\_literal} \leftarrow \text{argmax}_{\ell \in \text{candidate\_literals}} \text{PERFORMANCE(SPECIALIZE-RULE(NewRule, \ell))} \\
\quad \text{Add Best\_literal to NewRule\_preconditions} \\
\quad \text{NegExamplesCovered} \leftarrow \text{subset of NegExamplesCovered that satisfies NewRule\_preconditions} \\
\text{Learned\_rules} & \leftarrow \text{Learned\_rules} + \text{NewRule} \\
\text{Pos} & \leftarrow \text{Pos} - \{\text{members of Pos covered by NewRule}\} \\
\text{Return Learned\_rules}
\end{align*}
\]

Subtleties: Learning One Rule

1. May use beam search
2. Easily generalize to multi-valued target functions
3. Choose evaluation function to guide search:
   - Entropy (i.e., information gain)
   - Sample accuracy: \[
   \frac{n_c}{n}
   \]
   where \(n_c\) = correct predictions, \(n\) = all predictions
   - \(m\) estimate: \[
   \frac{n_c + mp}{n + mR}
   \]
Variants of Rule Learning Programs

• Sequential or simultaneous covering of data?
• General → specific, or specific → general?
• Generate-and-test, or example-driven?
• Whether and how to post-prune?
• What statistical evaluation functions?

Learning First Order Rules

Why do that?

• Can learn sets of rules such as
  \[ \text{Ancestor}(x,y) \leftarrow \text{Parent}(x,y) \]
  \[ \text{Ancestor}(x,y) \leftarrow \text{Parent}(x,z) \land \text{Ancestor}(z,y) \]
• General purpose programming language
  PROLOG: programs are sets of such rules

First Order Rule for Classifying Web Pages

From (Slattery, 1997)

\[
\text{course}(A) \leftarrow \\
\text{has-word}(A,\text{instructor}), \\
\text{NOT has-word}(A,\text{good}), \\
\text{link-from}(A,B) \\
\text{has-word}(B,\text{assignment}), \\
\text{NOT link-from}(B,C)
\]

Train: 31/31, Test 31/34

Specializing Rules in FOIL

Learning rule: \( P(x_1, x_2, \ldots, x_k) \leftarrow L_1, \ldots, L_n \)
Candidate specializations add new literal of form:

• \( Q(v_1, \ldots, v_r) \), where at least one of the \( v_i \) in the created literal must already exist as a variable in the rule

• \( \text{Equal}(x, x_i) \), where \( x \) and \( x_i \) are variables already present in the rule

• The negation of either of the above forms of literals

Information Gain in FOIL

\[
\text{FOIL}_G L A I N(L, R) = \log_2 \left( \frac{P_1}{P_1 + n_1} \right) - \log_2 \left( \frac{P_0}{P_0 + n_0} \right)
\]

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 \( R + L \)

Note

• \( \frac{P_0}{P_0 + n_0} \) is optimal number of bits to indicate the class of a positive binding covered by \( R \)
Induction as Inverted Deduction

Induction is finding $h$ such that

\[ (\forall x_i, f(x_i) \in D) B \land h \land x_i \not\vdash f(x_i) \]

where

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

So let’s design inductive algorithms by inverting operators for automated deduction!

---

Induction and Deduction

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 question of deduction … (Jevons, 1874)

---

Induction as Inverted 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 $(\forall x_i, f(x_i) \in D) B \land h \land x_i \not\vdash f(x_i)$?

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

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

---

Induction as Inverted Deduction

We have mechanical deductive operators

\[ F(A,B) = C, \text{ where } A \land B \not\vdash C \]

need inductive operators

\[ O(B,D) = h \text{ where} \]

\[ (\forall x_i, f(x_i) \in D) B \land h \land x_i \not\vdash f(x_i) \]

---

Induction as Inverted Deduction

Positives:

- Subsumes earlier idea of finding $h$ that “fits” training data
- Domain theory $B$ helps define meaning of “fit” the data $B \land h \land x_i \not\vdash f(x_i)$
- Suggests algorithms that search $H$ guided by $B$

Negatives:

- Doesn’t allow for noisy data. Consider
  \[ (\forall <x_i,f(x_i)> \in D) B \land h \land x_i \not\vdash f(x_i) \]
- First order logic gives a huge hypothesis space $H$
  - overfitting…
  - intractability of calculating all acceptable $h$’s

---

Deduction: Resolution Rule

\[
\frac{P \lor L}{\neg L \lor R} \quad \frac{P \lor R}{P \lor L}
\]

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 “-” set difference.
Inverting Resolution

\[ C_1: \text{PassExam} \lor \neg \text{KnowMaterial} \]
\[ C_2: \text{KnowMaterial} \lor \neg \text{Study} \]
\[ C: \text{PassExam} \lor \neg \text{Study} \]

Cigol

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

Progo1

PROGOL: Reduce combinatorial explosion by generating the most specific acceptable \( h \)
1. User specifies \( H \) by stating predicates, functions, and forms of arguments allowed for each
2. PROGOL uses sequential covering algorithm.
   - Find most specific hypothesis \( h \), s.t.
     \[ B \land h \land x_i \models \neg f(x) \]
     actually, only considers \( k \)-step entailment
   - Conduct general-to-specific search bounded by specific hypothesis \( h \), choosing hypothesis with minimum description length

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

Learning Rules Summary

- Rules: easy to understand
  - Sequential covering algorithm
  - generate one rule at a time
  - general to specific - add antecedents
  - specific to general - delete antecedents
  - Q: how to evaluate/stop?
- First order logic and covering
  - how to connect variables
  - FOIL
Learning Rules Summary (cont)

- Induction as inverted deduction
  - what background rule would allow deduction?
  - resolution
  - inverting resolution
  - and first order logic
    - Cigol, Progol