Lecture 26

Rule Learning and Extraction

Tuesday, November 30, 1999

William H. Hsu
Department of Computing and Information Sciences, KSU
http://www.cis.ksu.edu/~bhsu

Readings:
Sections 10.1-10.5, Mitchell
Section 21.4, Russell and Norvig
Section 5.4.5, Shavlik and Dietterich (Shavlik and Towell)

Lecture Outline

- Readings: Sections 10.1-10.5, Mitchell; Section 21.4 Russell and Norvig
- Suggested Exercises: 10.1, 10.2 Mitchell
- This Week's Paper Review (Last One!)
  - "An Approach to Combining Explanation-Based and Neural Network Algorithms", Shavlik and Towell
  - Due today, 11/30/1999
- Sequential Covering Algorithms
  - Learning single rules by search
    - Beam search
    - Alternative covering methods
  - Learning rule sets
- First-Order Rules
  - Learning single first-order rules
  - FOIL: learning first-order rule sets

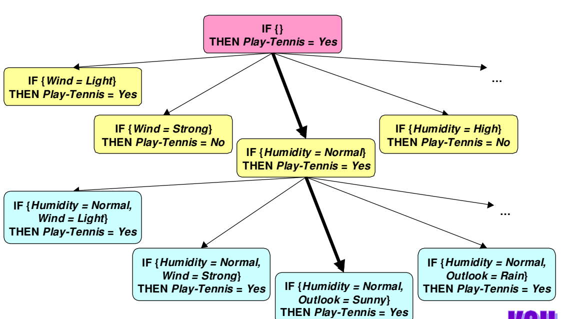
Learning Disjunctive Sets of Rules

- Method 1: Rule Extraction from Trees
  - Learn decision tree
  - Convert to rules
    - One rule per root-to-leaf path
    - Recall: can *post-prune* rules (drop pre-conditions to improve validation set accuracy)
- Method 2: Sequential Covering
  - Idea: greedily (sequentially) find rules that apply to (cover) instances in *D*
  - Algorithm
    - *Learn one rule* with high accuracy, any coverage
    - Remove positive examples (of target attribute) covered by this rule
    - Repeat

Sequential Covering: Algorithm

- Algorithm *Sequential-Covering* (*Target-Attribute*, *Attributes*, *D*, *Threshold*)
  - *Learned-Rules* ← {}
  - *New-Rule* ← *Learn-One-Rule* (*Target-Attribute*, *Attributes*, *D*)
  - WHILE *Performance* (*Rule*, *Examples*) > *Threshold* DO
    - *Learned-Rules* += *New-Rule* // add new rule to set
    - *D.Remove-Covered-By* (*New-Rule*) // remove examples covered by *New-Rule*
    - *New-Rule* ← *Learn-One-Rule* (*Target-Attribute*, *Attributes*, *D*)
  - *Sort-By-Performance* (*Learned-Rules*, *Target-Attribute*, *D*)
  - RETURN *Learned-Rules*
- What Does *Sequential-Covering* Do?
  - Learns one rule, *New-Rule*
  - Takes out every example in *D* to which *New-Rule* applies (every covered example)

Learn-One-Rule: (Beam) Search for Preconditions



Learn-One-Rule: Algorithm

- Algorithm *Sequential-Covering* (*Target-Attribute*, *Attributes*, *D*)
  - *Pos* ← *D.Positive-Examples*()
  - *Neg* ← *D.Negative-Examples*()
  - WHILE NOT *Pos.Empty*() DO // learn new rule
    - *Learn-One-Rule* (*Target-Attribute*, *Attributes*, *D*)
    - *Learned-Rules.Add-Rule* (*New-Rule*)
    - *Pos.Remove-Covered-By* (*New-Rule*)
  - RETURN (*Learned-Rules*)
- Algorithm *Learn-One-Rule* (*Target-Attribute*, *Attributes*, *D*)
  - *New-Rule* ← most general rule possible
  - *New-Rule-Neg* ← *Neg*
  - WHILE NOT *New-Rule-Neg.Empty*() DO // specialize *New-Rule*
    1. *Candidate-Literals* ← *Generate-Candidates*() // *NB*: rank by *Performance*()
    2. *Best-Literal* ← $\text{argmax}_{L \in \textit{Candidate-Literals}}$ *Performance* (*Specialize-Rule* (*New-Rule*, *L*), *Target-Attribute*, *D*) // all possible new constraints
    3. *New-Rule.Add-Precondition* (*Best-Literal*) // add the best one
    4. *New-Rule-Neg* ← *New-Rule-Neg.Filter-By* (*New-Rule*)
  - RETURN (*New-Rule*)

Learn-One-Rule: Subtle Issues

- How Does *Learn-One-Rule* Implement Search?
  - Effective approach: *Learn-One-Rule* organizes *H* in same general fashion as *ID3*
  - Difference
    - Follows only *most promising branch in tree* at each step
    - Only *one attribute-value pair* (versus splitting on all possible values)
  - General to specific search (depicted in figure)
    - Problem: greedy depth-first search susceptible to local optima
    - Solution approach: beam search (rank by performance, always expand *k* best)
    - Easily generalizes to multi-valued target functions (how?)
- Designing Evaluation Function to Guide Search
  - *Performance* (*Rule*, *Target-Attribute*, *D*)
  - Possible choices
    - *Entropy* (i.e., *information gain*) as for *ID3*
    - Sample accuracy (n_c / $n \equiv$ correct rule predictions / total predictions)
    - *m* estimate: ($n_c + mp$) / ($n + m$) where $m \equiv$ weight, $p \equiv$ prior of rule RHS

Variants of Rule Learning Programs

- Sequential or Simultaneous Covering of Data?
  - Sequential: isolate components of hypothesis (e.g., search for *one rule at a time*)
  - Simultaneous: whole hypothesis at once (e.g., search for *whole tree at a time*)
- General-to-Specific or Specific-to-General?
  - General-to-specific: add preconditions, *Find-G*
  - Specific-to-general: drop preconditions, *Find-S*
- Generate-and-Test or Example-Driven?
  - Generate-and-test: search through syntactically legal hypotheses
  - Example-driven: *Find-S*, *Candidate-Elimination*, *Cigol* (next time)
- Post-Pruning of Rules?
  - Recall (Lecture 5): very popular overfitting recovery method
- What Statistical Evaluation Method?
  - Entropy
  - Sample accuracy (*aka* relative frequency)
  - *m*-estimate of accuracy

First-Order Rules

- What Are First-Order Rules?
  - Well-formed formulas (WFFs) of first-order predicate calculus (FOPC)
  - Sentences of first-order logic (FOL)
  - Example (recursive)
    - *Ancestor* (*x*, *y*) $\leftarrow$ *Parent* (*x*, *y*).
    - *Ancestor* (*x*, *y*) $\leftarrow$ *Parent* (*x*, *z*) $\wedge$ *Ancestor* (*z*, *y*).
- Components of FOPC Formulas: Quick Intro to Terminology
  - Constants: e.g., *John*, *Kansas*, 42
  - Variables: e.g., *Name*, *State*, *x*
  - Predicates: e.g., *Father-Of*, *Greater-Than*
  - Functions: e.g., *age*, *cosine*
  - Term: constant, variable, or *function*(*term*)
  - Literals (atoms): *Predicate*(*term*) or negation (e.g., $\neg$*Greater-Than* (*age*(*John*), 42))
  - Clause: disjunction of literals with implicit universal quantification
  - Horn clause: at most one positive literal ($H \vee \neg L_1 \vee \neg L_2 \vee \ldots \vee \neg L_n$)

Learning First-Order Rules

- Why Do That?
  - Can learn sets of rules such as
    - *Ancestor* (*x*, *y*) $\leftarrow$ *Parent* (*x*, *y*).
    - *Ancestor* (*x*, *y*) $\leftarrow$ *Parent* (*x*, *z*) $\wedge$ *Ancestor* (*z*, *y*).
  - General-purpose (Turing-complete) programming language PROLOG
    - Programs are such sets of rules (Horn clauses)
    - Inductive logic programming (next time): kind of *program synthesis*
- Caveat
  - Arbitrary inference using first-order rules is semi-decidable
    - Recursive enumerable but not recursive (reduction to halting problem L_H)
    - Compare: resolution theorem-proving; arbitrary queries in Prolog
  - Generally, may have to restrict power
    - Inferential completeness
    - Expressive power of Horn clauses
    - *Learning part*

First-Order Rule: Example

- Prolog (FOPC) Rule for Classifying Web Pages
  - [Slattery, 1997]
  - *Course* (*A*) $\leftarrow$
    - *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
- How Are Such Rules Used?
  - Implement search-based (inferential) programs
  - References
    - Chapters 1-10, Russell and Norvig
    - Online resources at http://archive.comlab.ox.ac.uk/logic-prog.html

First-Order Inductive Learning (FOIL): Algorithm

- Algorithm *FOIL* (*Target-Predicate*, *Predicates*, *D*)
  - *Pos* $\leftarrow$ *D.Filter-By*(*Target-Predicate*) // examples for which it is true
  - *Neg* $\leftarrow$ *D.Filter-By*(*Not* (*Target-Predicate*)) // examples for which it is false
  - WHILE NOT *Pos.Empty*() DO // learn new rule
    - *Learn-One-First-Order-Rule* (*Target-Predicate*, *Predicates*, *D*)
    - *Learned-Rules.Add-Rule* (*New-Rule*)
    - *Pos.Remove-Covered-By* (*New-Rule*)
  - RETURN (*Learned-Rules*)
- Algorithm *Learn-One-First-Order-Rule* (*Target-Predicate*, *Predicate*, *D*)
  - *New-Rule* $\leftarrow$ the rule that predicts *Target-Predicate* with no preconditions
  - *New-Rule-Neg* $\leftarrow$ *Neg*
  - WHILE NOT *New-Rule-Neg.Empty*() DO // specialize *New-Rule*
    1. *Candidate-Literals* $\leftarrow$ *Generate-Candidates*() // based on *Predicates*
    2. *Best-Literal* $\leftarrow$ $argmax_{L \in \text{Candidate-Literals}}$ *FOIL-Gain* (*L*, *New-Rule*, *Target-Predicate*, *D*) // all possible new literals
    3. *New-Rule.Add-Precondition* (*Best-Literal*) // add the best one
    4. *New-Rule-Neg* $\leftarrow$ *New-Rule-Neg,Filter-By* (*New-Rule*)
  - RETURN (*New-Rule*)

Specializing Rules in FOIL

- Learning Rule: $P(x_1, x_2, \ldots, x_k) \leftarrow L_1 \wedge L_2 \wedge \ldots \wedge L_n$.
- Candidate Specializations
  - Add new literal to get more specific Horn clause
  - Form of literal
    - $Q(v_1, v_2, \ldots, v_r)$, 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

Information Gain in FOIL

- Function *FOIL-Gain* (*L*, *R*, *Target-Predicate*, *D*)

$$Foil\text{-}Gain \equiv t\left(\lg\left(\frac{p_1}{p_1+n_1}\right) - \lg\left(\frac{p_0}{p_0+n_0}\right)\right)$$

- Where
  - $L \equiv$ candidate predicate to add to rule R
  - $p_0 \equiv$ number of positive bindings of R
  - $n_0 \equiv$ number of negative bindings of R
  - $p_1 \equiv$ number of positive bindings of $R + L$
  - $n_1 \equiv$ number of negative bindings of $R + L$
  - $t \equiv$ number of positive bindings of R also covered by $R + L$
- Note
  - $-\lg(p_0 / p_0 + n_0)$ is optimal number of bits to indicate the class of a positive binding covered by R
  - Compare: entropy (information gain) measure in *ID3*

FOIL: Learning Recursive Rule Sets

- Recursive Rules
  - So far: ignored possibility of recursive WFFs
    - New literals added to rule body could refer to target predicate itself
    - i.e., predicate occurs in rule head
  - Example
    - *Ancestor* $(x, y) \leftarrow$ *Parent* $(x, z) \wedge$ *Ancestor* (z, y).
    - Rule: IF *Parent* $(x, z) \wedge$ *Ancestor* (z, y) THEN *Ancestor* (x, y)
- Learning Recursive Rules from Relations
  - Given: appropriate set of training examples
  - Can learn using *FOIL*-based search
    - Requirement: *Ancestor* $\in$ *Predicates* (symbol is member of candidate set)
    - Recursive rules still have to outscore competing candidates at *FOIL-Gain*
  - *NB*: how to ensure termination? (well-founded ordering, i.e., no infinite recursion)
  - [Quinlan, 1990; Cameron-Jones and Quinlan, 1993]

FOIL: Summary

- Extends *Sequential-Covering* Algorithm
  - Handles case of learning first-order rules similar to Horn clauses
  - Result: *more powerful rules for performance element* (automated reasoning)
- General-to-Specific Search
  - Adds literals (predicates and negations over functions, variables, constants)
  - Can learn sets of recursive rules
    - Caveat: might learn infinitely recursive rule sets
    - Has been shown to successfully induce recursive rules in some cases
- Overfitting
  - If no noise, might keep adding new literals until rule covers *no negative examples*
  - Solution approach: tradeoff (heuristic evaluation function on rules)
    - Accuracy, coverage, complexity
    - *FOIL-Gain*: an MDL function
    - Overfitting recovery in *FOIL*: post-pruning

Terminology

- Disjunctive Rules
  - Learning single rules by search
    - Beam search: type of heuristic search that maintains constant-width frontier
  - Learning rule sets
    - Sequential covering versus simultaneous covering
- First-Order Rules
  - Units of first-order predicate calculus (FOPC): constants, variables, predicates, functions, terms, literals (atoms), well-formed fomulas (wffs, clauses)
  - FOPC quantifiers: universal ($\forall$), existential ($\exists$)
  - Horn clauses
    - Sentences of Prolog (clauses with $\leq$ 1 positive literal)
    - Of the form: $H \vee \neg L_1 \vee \neg L_2 \vee \ldots \vee \neg L_n$ (implicit $\forall$), Prolog form H :- $L_1, L_2, \ldots, L_n$.
  - FOIL: algorithm for learning Horn clauses (including recursive rule sets)

Summary Points

- Learning Rules from Data
- Sequential Covering Algorithms
  - Learning single rules by search
    - Beam search
    - Alternative covering methods
  - Learning rule sets
- First-Order Rules
  - Learning single first-order rules
    - Representation: first-order Horn clauses
    - Extending *Sequential-Covering* and Learn-One-Rule: variables in rule preconditions
  - FOIL: learning first-order rule sets
    - Idea: inducing logical rules from observed relations
    - Guiding search in FOIL
    - Learning recursive rule sets
- Next Time: Inductive Logic Programming (ILP)