

For Wednesday

- Read chapter 20, sections 1-2, 5
- No homework

Program 4

- Any questions?

Exam 2

- Friday
- Covers material through chapter 18
- Take home due at the exam

Review questions?

Avoiding Overfitting

- Two basic approaches
 - **Prepruning**: Stop growing the tree at some point during construction when it is determined that there is not enough data to make reliable choices.
 - **Postpruning**: Grow the full tree and then remove nodes that seem to not have sufficient evidence.

Evaluating Subtrees to Prune

- Cross-validation:
 - Reserve some of the training data as a hold-out set (validation set, tuning set) to evaluate utility of subtrees.
- Statistical testing:
 - Perform some statistical test on the training data to determine if any observed regularity can be dismissed as likely to be random chance.
- Minimum Description Length (MDL):
 - Determine if the additional complexity of the hypothesis is less complex than just explicitly remembering any exceptions.

Beyond a Single Learner

- Ensembles of learners work better than individual learning algorithms
- Several possible ensemble approaches:
 - Ensembles created by using different learning methods and voting
 - Bagging
 - Boosting

Bagging

- Random selections of examples to learn the various members of the ensemble.
- Seems to work fairly well, but no real guarantees.

Boosting

- Most used ensemble method
- Based on the concept of a **weighted** training set.
- Works especially well with **weak** learners.
- Start with all weights at 1.
- Learn a hypothesis from the weights.
- Increase the weights of all misclassified examples and decrease the weights of all correctly classified examples.
- Learn a new hypothesis.
- Repeat

Different Ways of Incorporating Knowledge in Learning

- Explanation Based Learning (EBL)
- Theory Revision (or Theory Refinement)
- Knowledge Based Inductive Learning (in first-order logic - Inductive Logic Programming (ILP))

Explanation Based Learning

- Requires two inputs
 - Labeled examples (maybe very few)
 - Domain theory
- Goal
 - To produce operational rules that are consistent with both examples and theory
 - Classical EBL requires that the theory entail the resulting rules

Why Do EBL?

- Often utilitarian or speed-up learning
- Example: DOLPHIN
 - Uses EBL to improve planning
 - Both speed-up learning and improving plan quality

Theory Refinement

- Inputs the same as EBL
 - Theory
 - Examples
- Goal
 - Fix the theory so that it agrees with the examples
- Theory may be incomplete or wrong

Why Do Theory Refinement?

- Potentially more accurate than induction alone
- Able to learn from fewer examples
- May influence the structure of the theory to make it more comprehensible to experts

How Is Theory Refinement Done?

- Initial State: Initial Theory
- Goal State: Theory that fits training data.
- Operators: Atomic changes to the syntax of a theory:
 - Delete rule or antecedent, add rule or antecedent
 - Increase parameter, Decrease parameter
 - Delete node or link, add node or link
- Path cost: Number of changes made, or total cost of changes made.

Theory Refinement As Heuristic Search

- Finding the “closest” theory that is consistent with the data is generally intractable (NP-hard).
- Complete consistency with training data is not always desirable, particularly if the data is noisy.
- Therefore, most methods employ some form of greedy or hill-climbing search.
- Also, usually employ some form of over-fitting avoidance method to avoid learning an overly complex revision.

Theory Refinement As Bias

- Bias is to learn a theory which is syntactically similar to the initial theory.
- Distance can be measured in terms of the number of edit operations needed to revise the theory (edit distance).
- Assumes the syntax of the initial theory is “approximately correct.”
- A bias for minimal semantic revision would simply involve memorizing the exceptions to the theory, which is undesirable with respect to generalizing to novel data.

Inductive Logic Programming

- Representation is Horn clauses
- Builds rules using background predicates
- Rules are potentially much more expressive than attribute-value representations

Example Results

- Rules for family relations from data of primitive or related predicates.

uncle(A,B) :- brother(A,C), parent(C,B).

uncle(A,B) :- husband(A,C), sister(C,D),
parent(D,B).

- Recursive list programs.

member(X,[X | Y]).

member(X, [Y | Z]) :- member(X, Z).

ILP

- Goal is to induce a Horn-clause definition for some **target predicate** P given definitions of **background predicates** Q_i .
- Goal is to find a syntactically simple definition D for P such that given background predicate definitions B
 - For every positive example p_i : $D \cup B \models p$
 - For every negative example n_i : $D \cup B \not\models n$
- Background definitions are either provided
 - Extensionally: List of ground tuples satisfying the predicate.
 - Intensionally: Prolog definition of the predicate.

Sequential Covering Algorithm

Let P be the set of positive examples

Until P is empty do

Learn a rule R that covers a large number of positives without covering any negatives.

Add R to the list of learned rules.

Remove positives covered by R from P

- This is just an instance of the greedy algorithm for minimum set covering and does not guarantee that a minimum number of rules is learned but tends to learn a reasonably small rule set.
- Minimum set covering is an NP-hard problem and the greedy algorithm is a common approximation algorithm.
- There are several ways to learn a single rule used in various methods.

Strategies for Learning a Single Rule

- Top-Down (General to Specific):
 - Start with the most general (empty) rule.
 - Repeatedly add feature constraints that eliminate negatives while retaining positives.
 - Stop when only positives are covered.
- Bottom-Up (Specific to General):
 - Start with a most specific rule (complete description of a single instance).
 - Repeatedly eliminate feature constraints in order to cover more positive examples.
 - Stop when further generalization results in covering negatives.

FOIL

- Basic top-down sequential covering algorithm adapted for Prolog clauses.
 - Background provided **extensionally**.
 - Initialize clause for target predicate P to
 $P(X_1, \dots, X_r) :- .$
 - Possible specializations of a clause include adding all possible **literals**:
 - $Q_i(V_1, \dots, V_r)$
 - $\text{not}(Q_i(V_1, \dots, V_r))$
 - $X_i = X_j$
 - $\text{not}(X_i = X_j)$
- where X's are variables in the existing clause, at least one of V_1, \dots, V_r is an existing variable, others can be new.
- Allow recursive literals if not cause infinite regress.

Foil Input Data

- Consider example of finding a path in a directed acyclic graph.

- Intended Clause:

`path(X,Y) :- edge(X,Y).`

`path(X,Y) :- edge(X,Z), path (Z,Y).`

- Examples

`edge: { <1,2>, <1,3>, <3,6>, <4,2>, <4,6>, <6,5> }`

`path: { <1,2>, <1,3>, <1,6>, <1,5>, <3,6>, <3, 5>, <4,2>, <4,6>, <4,5>, <6, 5> }`

- Negative examples of the target predicate can be provided directly or indirectly produced using a **closed world assumption**. Every pair $\langle x,y \rangle$ not in positive tuples for path.

Example Induction

+ : { $\langle 1,2 \rangle$, $\langle 1,3 \rangle$, $\langle 1,6 \rangle$, $\langle 1,5 \rangle$, $\langle 3,6 \rangle$, $\langle 3,5 \rangle$, $\langle 4,2 \rangle$,
 $\langle 4,6 \rangle$, $\langle 4,5 \rangle$, $\langle 6,5 \rangle$ }

- : { $\langle 1,4 \rangle$, $\langle 2,1 \rangle$, $\langle 2,3 \rangle$, $\langle 2,4 \rangle$, $\langle 2,5 \rangle$, $\langle 2,6 \rangle$, $\langle 3,1 \rangle$,
 $\langle 3,2 \rangle$, $\langle 3,4 \rangle$, $\langle 4,1 \rangle$, $\langle 4,3 \rangle$, $\langle 5,1 \rangle$, $\langle 5,2 \rangle$, $\langle 5,3 \rangle$, $\langle 5,4 \rangle$,
 $\langle 5,6 \rangle$, $\langle 6,1 \rangle$, $\langle 6,2 \rangle$, $\langle 6,3 \rangle$, $\langle 6,4 \rangle$ }

- Start with empty rule: $\text{path}(X,Y) :-$.
- Among others, consider adding literal $\text{edge}(X,Y)$ (also consider $\text{edge}(Y,X)$, $\text{edge}(X,Z)$, $\text{edge}(Z,X)$, $\text{path}(Y,X)$, $\text{path}(X,Z)$, $\text{path}(Z,X)$, $X=Y$, and negations)
- 6 positive tuples and NO negative tuples covered.
- Create “base case” and remove covered examples:
 $\text{path}(X,Y) :- \text{edge}(X,Y)$.

+ : { <1,6>, <1,5>, <3, 5>, <4,5> }

- : { <1,4>, <2,1>, <2,3>, <2,4>, <2,5> <2,6>, <3,1>, <3,2>, <3,4>, <4,1>, <4,3>, <5,1>, <5,2>, <5,3>, <5,4> <5,6>, <6,1>, <6,2>, <6,3>, <6,4> }

- Start with new empty rule: $\text{path}(X,Y) :-$.
- Consider literal $\text{edge}(X,Z)$ (among others...)
- 4 remaining positives satisfy it but so do 10 of 20 negatives
- Current rule: $\text{path}(x,y) :- \text{edge}(X,Z)$.
- Consider literal $\text{path}(Z,Y)$ (as well as $\text{edge}(X,Y)$, $\text{edge}(Y,Z)$, $\text{edge}(X,Z)$, $\text{path}(Z,X)$, etc....)
- No negatives covered, complete clause.
 $\text{path}(X,Y) :- \text{edge}(X,Z), \text{path}(Z,Y)$.
- New clause actually covers all remaining positive tuples of path, so definition is complete.

Picking the Best Literal

- Based on information gain (similar to ID3).

$$|p| * (\log_2 (|p| / (|p| + |n|)) - \log_2 (|P| / (|P| + |N|)))$$

P is number of positives before adding literal L

N is number of negatives before adding literal L

p is number of positives after adding literal L

n is number of negatives after adding literal L

- Given n predicates of arity **m** there are $O(n2^m)$ possible literals to choose from, so branching factor can be quite large.

Other Approaches

- Golem
- CHILL
- Foidl
- Bufoidl

Domains

- Any kind of concept learning where background knowledge is useful.
- Natural Language Processing
- Planning
- Chemistry and biology
 - DNA
 - Protein structure