Separate-and-Conquer Rule Learning by Johannes Fürnkranz

Sukanya Basu (09305908)

February 4, 2012

Outline

- Motivation
- The problem
- Separate-and-conquer strategy
- Algorithm
- Bias
- Observations
- Conclusion

The problem

The goal of the algorithm is to discover a description of the target concept in the form of explicit rules.

Given:

- a target concept
- positive and negative examples
- described with several features and
- optional background knowledge

Find:

- A simple set of rules that discriminate between (unseen) positive and negative examples of the target concept

The resulting rule set should be able to:
- correctly recognize instances of the target concept and
- discriminate them from objects that do not belong to the target concept.

Why separate-and-conquer strategy

- Developed for a variety of different learning tasks.
- Most commonly used alternative: decision tree learning via divide-and-conquer strategy.
- Decision trees are often quite complex and difficult to understand.
- Algorithms restricted to non overlapping rules.
- Different separate-and-conquer algorithms developed to learn propositional rule sets, decision lists, logic programs functional relations and regression rules.

Algorithm

Following is the separate-and-conquer algorithm in the most simple form.

```
procedure SIMPLE_SEPARATE_AND_CONQUER(Examples)

Theory = ∅
while POSITIVE(Examples) ≠ ∅
    BestRule = {true}
    Rule = BestRule
    while NEGATIVE(Cover) ≠ ∅
        for Condition ∈ Conditions
            Refinement = Rule ∪ Condition
            if PURITY(Refinement,Examples) > PURITY(BestRule,Examples)
                BestRule = Refinement
        Rule = BestRule
    Theory = Theory ∪ Rule
    Examples = Examples − Cover
return(Theory)
```

More generic algorithm

```
procedure SeparateAndConquer(Examples)

Theory = ∅
while Positive(Examples) ≠ ∅
    Rule = FindBestRule(Examples)
    Covered = Cover(Rule,Examples)
    if RuleStoppingCriterion(Theory,Rule,Examples)
        exit while
    Examples = Examples \ Covered
    Theory = Theory ∪ Rule
Theory = PostProcess (Theory)
return(Theory)
```

More generic algorithm (contd.)

```
procedure FindBestRule(Examples)

InitRule = InitializeRule(Examples)
InitVal = EvaluateRule(InitRule)
BestRule = <InitVal,InitRule>
Rules = {BestRule}
while Rules ≠ ∅
    Candidates = SelectCandidates(Rules, Examples)
    Rules = Rules \ Candidates
    for Candidate ∈ Candidates
        Refinements = RefineRule(Candidate, Examples)
        for Refinement ∈ Refinements
            Evaluation = EvaluateRule(Refinement, Examples)
            unless StoppingCriterion(Refinement, Evaluation, Examples)
                NewRule = <Evaluation,Refinement>
                Rules = InsertSort(NewRule, Rules)
                if NewRule > BestRule
                    BestRule = NewRule
    Rules = FilterRules(Rules, Examples)
return(BestRule)
```

Bias

- Learning algorithms need an appropriate bias for making an inductive leap.
- Bias is defined as, any basis for choosing one generalization over another, other tha strict consistency with the observed training instances.
- A learning algorithm can be charecterized with the bias it employs.
- Here, separate-and-conquer algorithms are characterized along three dimensions.
  - Language bias
  - Search bias
  - Overfitting avoidance bias

Language bias

User has to choose suitable representation language for the hypothesis to learn.

- Static approaches: Fix the language bias before the induction task. Wide variety of condition types include:
  - Selectors
  - Literals
  - Syntactic restrictions
- Dynamic approaches: Dynamically adjust their language bias to the problem at hand.
  - Language hierarchies
  - Constructive induction

Search bias

- Different search algorithms can be employed:
  - Hill-climbing (ATRIS)
  - Beam search (AQ, CN2, mFOIL, BEXA)
  - Best-first-search (ML-SMART, PROGOL)
  - Stochastic search (MILP, SFOIL, GA-SMART)
- Hypothesis space can be searched in different directions.
  - Top-down search (AQ, CN2, FOIL)
  - Bottom-up search (GOLEM, ITOU)
  - Bidirectional search (IBL-SMART, ATRIS)
- Various heuristic evaluation functions can be used to compare different candidate clauses.
  - Basic heuristics: Accuracy (PROGOL, I-REP) , Purity (GREEDY3, SWAP-1) , Information content (PRISM) , etc.
  - Complexity estimates: Rule length (PROGOL) , Positive coverage (FOIL) , etc.

Overfitting avoidance bias

- Real life data may be noisy
- Complete and consistent theories generated from noisy data usually very complex
- Show low predictive accuracy on classifying unseen examples.

Following are some techniques to avoid overfitting:

- Pre-pruning: Deal with noise during concept generation. (FOIL, BEXA)
- Post-pruning: Attempt to improve the learned theory in a post-processing phase. (AQ, POSEIDON)
- Combining pre and post-pruning. (GROW)

Conclusion

- Survey of best-known members of the family of separate-and-conquer algorithms.
- Has been in use for nearly 30 years, still popular.
- Confined to binary concept learning tasks: can be defined by a number of positive and negative examples for target concept.
- Help the practitioner to pick the right algorithm based on its charecteristics.

Personal aims:

- To focus on methods learning propositional rule sets and logic programs.
- Explore scope in bioinformatics