

## Rule Induction Overview

- Generic separate-and-conquer strategy
- CN2 rule induction algorithm
- Improvements to rule induction

## Problem

- Given:
  - A target concept
  - Positive and negative examples
  - Examples composed of features
- Find:
  - A simple set of rules that discriminates between (unseen) positive and negative examples of the target concept

## Sample Unordered Rules

- If X then C1
- If X and Y then C2
- If NOT X and Z and Y then C3
- If B then C2
  
- What if two rules fire at once? Just OR together?

## Target Concept

- Target concept in the form of rules. If we only have 3 features, X, Y, and Z, then we could generate the following possible rules:
  - If X then...
  - If X and Y then...
  - If X and Y and Z then...
  - If X and Z then ...
  - If Y then ...
  - If Y and Z then ...
  - If Z then...
- Exponentially large space, larger if allow NOT's

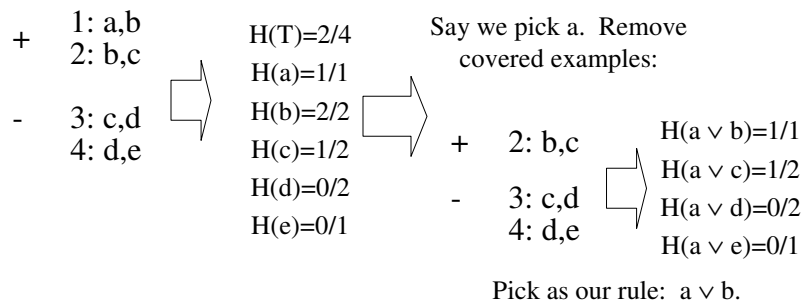
## Generic Separate-and-Conquer Strategy

```

TargetConcept = NULL
While NumPositive(Examples) > 0
  BestRule = TRUE
  Rule = BestRule
  Cover = ApplyRule(Rule)
  While NumNegative(Cover) > 0
    For each feature ∈ Features
      Refinement = Rule ∪ feature
      If Heuristic(Refinement, Examples) >
        Heuristic(BestRule, Examples)
        BestRule = Refinement
  Rule = BestRule
  Cover = ApplyRule(Rule)
TargetConcept = TargetConcept ∪ Rule
Examples = Examples - Cover
  
```

## Trivial Example

$$\text{Heuristic}(\text{rule}, \text{examples}) = \frac{\# \text{Positive}}{\# \text{Negative} + \# \text{Positive}}$$



## CN2 Rule Induction (Clark & Boswell, 1991)

- More specialized version of separate-and-conquer:

```
CN2Unordered(allexamples, allclasses)
  Ruleset ← {}
  For each class in allclasses
    Generate rules by CN2ForOneClass(allexamples, class)
    Add rules to ruleset
  Return ruleset
```

## CN2

```
CN2ForOneClass(examples, class)
  Rules ← {}
  Repeat
    Bestcond ← FindBestCondition(examples, class)
    If bestcond <> null then
      Add the rule "IF bestcond THEN PREDICT class"
      Remove from examples all + cases in
      class covered by bestcond
  Until bestcond = null
  Return rules
```

Keeps negative examples around so future rules won't impact existing negatives (allows unordered rules)

## CN2

```
FindBestCondition(examples, class)
  MGC ← true      ' most general condition
  Star ← MGC, Newstar ← {}, Bestcond ← null
  While Star is not empty (or loopcount < MAXCONJUNCTS)
    For each rule R in Star
      For each possible feature F
        R' ← specialization of Rule formed by adding F as an
              Extra conjunct to Rule (i.e. Rule' = Rule AND F)
              Removing null conditions (i.e. A AND NOT A)
              Removing redundancies (i.e. A AND A)
              and previously generated rules.
        If LaPlaceHeuristic(R', class) > LaPlaceHeuristic (Bestcond, class)
          Bestcond ← R'
        Add R' to Newstar
        If size(NewStar) > MAXRULESIZE then
          Remove worst in Newstar
        until Size=MAXRULESIZE
  Star ← Newstar
  Return Bestcond
```

## LaPlace Heuristic

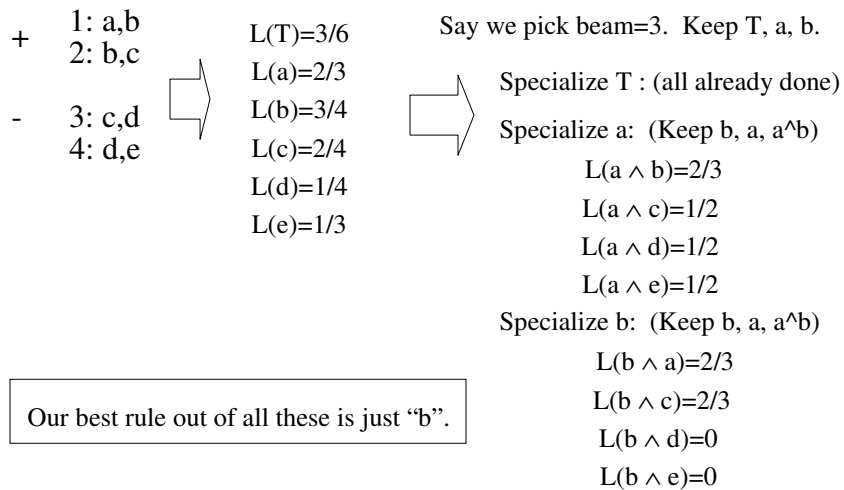
$$LaPlace(rule) = \frac{NumCorrectCovered(rule) + 1}{NumTotalCovered(rule) + NumClasses}$$

In our case, NumClasses=2.

A common problem is a specific rule that covers only 1 example.

In this case, LaPlace =  $1 + 1/1 + 2 = 0.6667$ . However, a rule that covers say 2 examples gets a higher value of  $2 + 1/2 + 2 = 0.75$ .

## Trivial Example Revisited



Our best rule out of all these is just "b".

Continue until out of features, or max num of conjuncts reached.

## Improvements to Rule Induction

- Better feature selection algorithm
- Add rule pruning phase
  - Problem of overfitting the data
  - Split training examples into a GrowSet (2/3) and PruneSet (1/3)
    - Train on GrowSet
    - Test on PruneSet with pruned rules, keep rule with best results
  - Needs more training examples!

## Improvements to Rule Induction

- Ripper / Slipper
  - Rule induction with pruning, new heuristics on when to stop adding rules, prune rules
  - Slipper builds on Ripper, but uses boosting to reduce weight of negative examples instead of removing them entirely
- Other search approaches
  - Instead of beam search, genetic, pure hill climbing (would be faster), etc.

## In-Class VB Demo

- Rule Induction for Multiplexer