Advanced learning algorithms

- Extending decision trees;
- Extraction of good classification rules;
- Support vector machines;
- Weighted instance-based learning;
- Design of Model Tree
- Clustering
- Association Mining

Reading Material: Chapters 6,7,8 of the textbook by Han and Chapter 6 of the textbook by Witten.

Extending ID3

Basic requirements for advanced learning algorithms:

- Can deal with numeric attributes
- Can deal with missing values
- Robust for noisy data
- Can approximate arbitrary concept descriptions

Extending basic schemes to fulfill these requirements.

Extending ID3 for decision trees

- Dealing with numeric attributes: easy;
- Dealing with missing values: tricky
- Stability for noisy data: requires sophisticated pruning mechanism

Final result of the extension: **Quinlans C4.5**, the best-known and most widely-used learning algorithm in the academic community and its commercial successor: C5.0

Dealing with Numeric Attri.

Standard method: binary splitting ($t < 45$)
Issue: How to select a good breaking point?
Remedy: Using the idea of information gain!

- Evaluate info gain for every possible split point of the attribute
- Choose best split point
- Info gain for best split point is the info gain for attribute

Conclusion: Computationally more demanding but more reliable in practice.

Example: splitting attribute 'temperature' from weather data:

64 65 68 69 70 71 72 72 75 75 80 81 83 85

Yes No Yes Yes Yes No| No Yes Yes Yes No Yes Yes No

The 'halfway' splitting at 71.5 (impossible to split at 72) yields two subsets:

$$[4('yes'), 2('no')], \quad [5('yes'), 3('no')]$$

Thus

$$Info([4,2],[5,3]) = \frac{6}{14} info([4,2]) + \frac{8}{14} info([5,3]) = 0.939 bits$$

Similarly we can compute the information gains for other splitting points.

Multi-way Splitting

Multi-way splitting is allowed for nominal attributes. Restriction to only binary splits on numeric attributes might lead to a messy tree that is relatively hard to read.

How to find a nice multi-way splitting for numeric attributes?

- Trade-off between time Complexity and efficient splitting!

Dynamic programming for generation an optimum multi(k)-way splits $O(n^k)$.

- Let $IMP(k,i,j)$ be the impurity of the best split of values $x_i, \cdots, x_j$ into k sub-intervals
- $IMP(k,i,j) = \min_{i<l<j}\{IMP(k-1,l+1,j) + IMP(1,i,l)\}$
- $IMP(k,1,N)$ provides the best k-way split.

The dynamic approach can find the optimal splitting strategy in term of information gain.

Speedup the splitting

Greedy algorithm of generating multi-way splitting: first find the best point for 1-splitting, and repeat this process for the resulting subset.

- O(n) time-complexity, works well for many practical problems.

Observation: The information gain hits its local minimum at some point where the major classes change!

- Focusing on only breaking points where the majority class change.

Repeated sorting

Instances need to be sorted according to the values of the numeric attribute considered: $O(n \log n)$ time.

Issue: How to avoid repeated sorting?

- Using the sorted-order from parent node to derive sort order for children within $O(n)$ time complexity.
- Create and store an array of sorted indices for each numeric attribute.

Example:

64 65 68 69 70 71 72 72 75 75 80 81 83 85
7 6 5 9 4 14 8 12 10 11 2 13 3 1

The weather problem

Outlook	Temper.	humidity	windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	**Mild**	High	False	**Yes**
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	**Mild**	High	False	**No**
Sunny	Cool	Normal	False	Yes
Rainy	**Mild**	Normal	False	**Yes**
Sunny	**Mild**	Normal	True	**Yes**
Overcast	**Mild**	High	True	**Yes**
Overcast	Hot	Normal	False	Yes
Rainy	**?**	High	True	**No**

Dealing with Missing Values

Missing values: splitting instances with missing values into pieces (weights sum to 1)

- A piece going down a branch receives a weight proportional to the popularity of the branch
- Info gain can be computed with fractional instances using sums of weights instead of counts

Example: Temperature

$$Hot : 4\frac{4}{13} = 2`yes' + 2\frac{4}{13}`no'$$
$$Cool : 4\frac{4}{13} = 3'yes' + 1\frac{4}{13}'no'$$
$$Mild : 5\frac{5}{13} = 4`yes' + 1\frac{5}{13}'no'$$

Calculate the information gain by yourself!

The same procedure is used for classification
• Probability distributions are merged using weights

Pruning

Pruning: a technique to simplify a decision tree and prevent it from overfitting.

Main pruning strategies:

- Postpruning: takes a fully- grown decision tree and discards unreliable parts.
- Prepruning: stops when information becomes unreliable, but almost impossible!

Postpruning:

- Attribute interactions are visible in fully-grown tree
- Problem: identification of subtrees and nodes that are due to chance effects

Two main pruning operations:

- 1.Subtree replacement ; selecting a subtree and replacing it by a leave
- 2.Subtree raising

Measurements: error estimation, significance testing, MDL principle

Estimating error rates

Pruning operation is performed if the operation does not increase the estimated error.

- Error on the training data is not helpful!
- Using hold-out set for pruning!

C4.5-method: using upper limit of 25% confidence interval derived from the training data

- Standard Bernoulli-process-based method.

Error estimate for subtree: the weighted sum of error estimates for all its leaves.

Error estimate for a node:

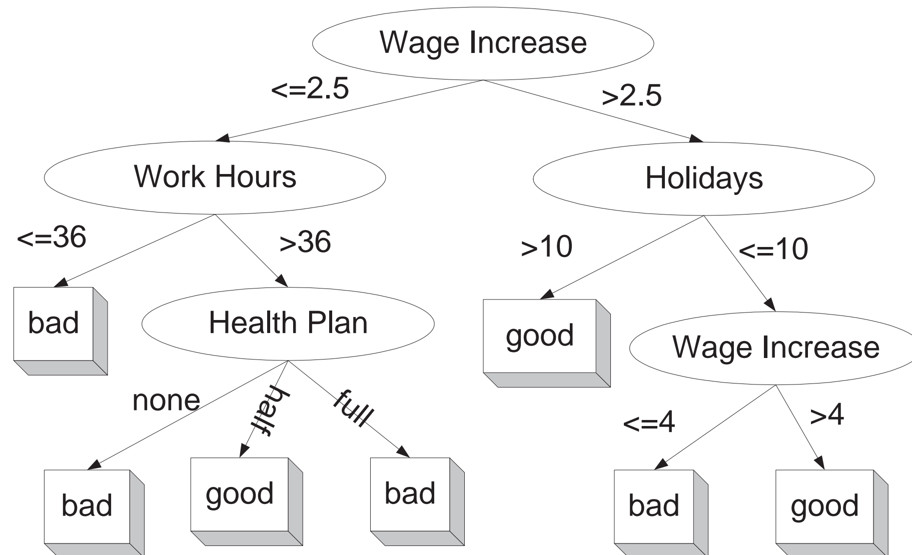
$$e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}.$$

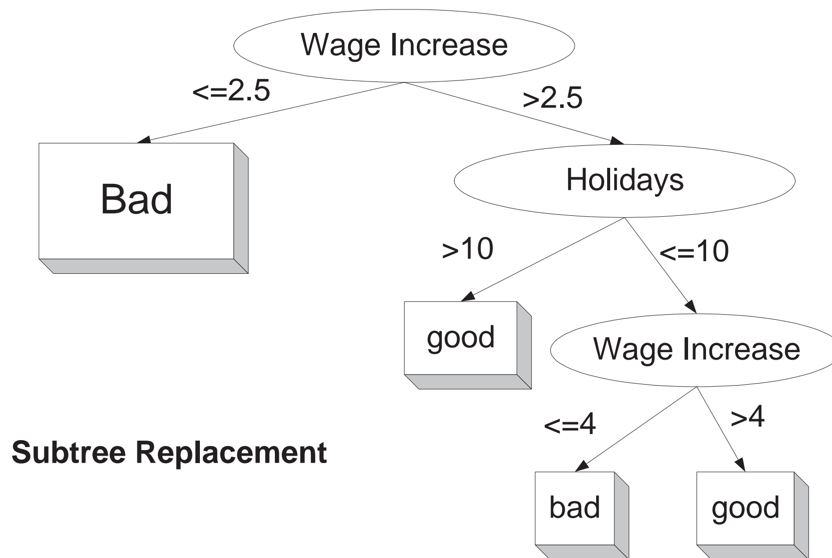
$Pr[X \geq z]$	0.1%	0.5%	1%	5%	10%	20%	40%
z	3.09	2.58	2.33	1.65	1.28	0.84	0.25

In C4.5, $c = 25\%, z = 0.69$ are default.
f is the error on the training data and N is the number of instances covered by the leaf.

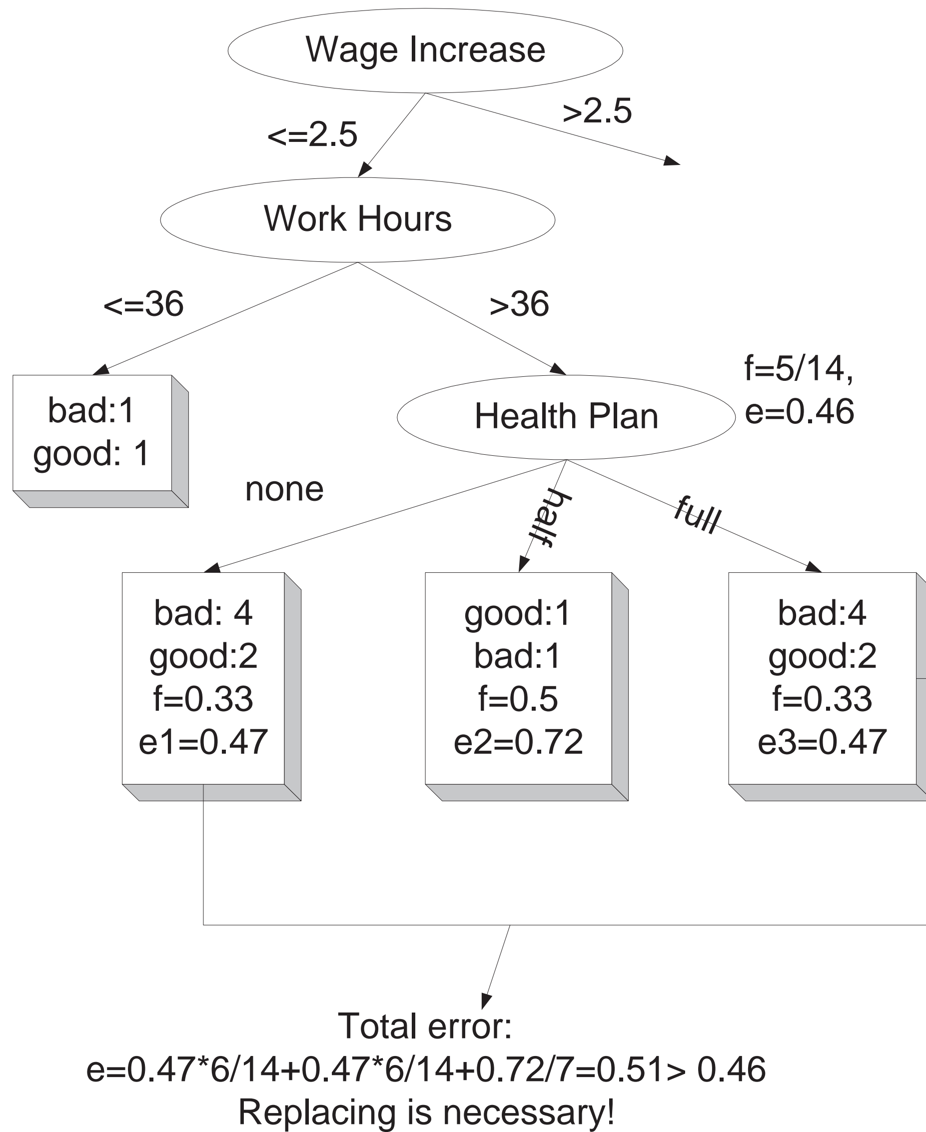
The principle for pruning can be applied in the extraction of classification rules.

Subtree Replacing: Figure 1

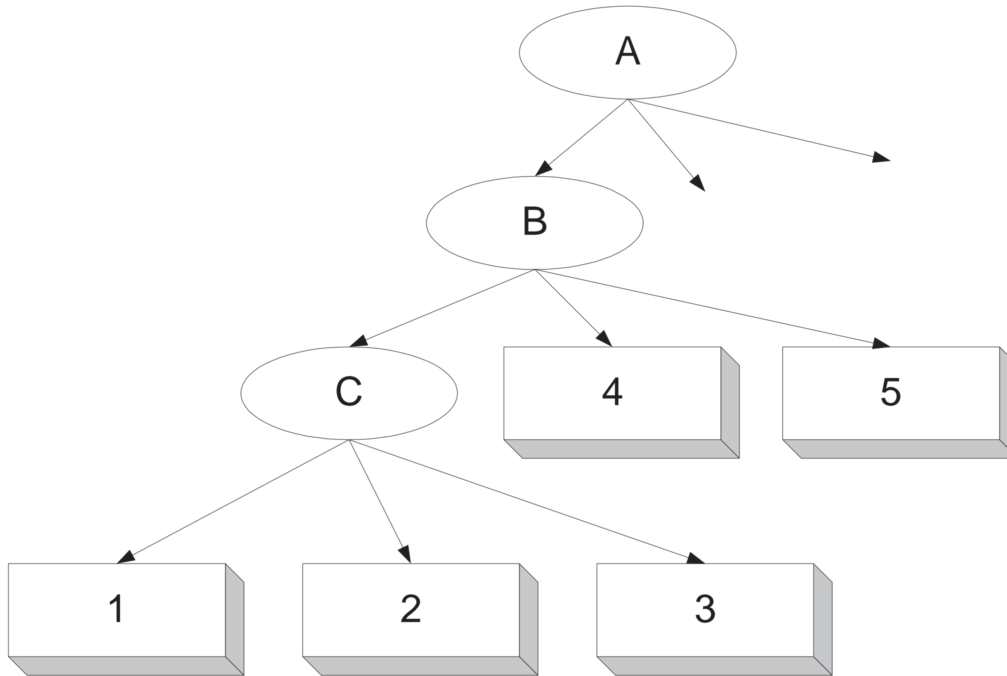


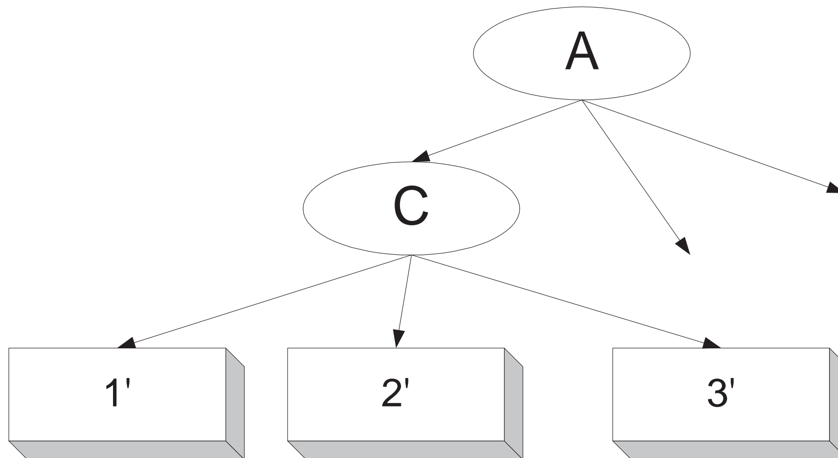


Subtree Replacing: Figure 2



Subtree Raising





Subtree Raising:
Delete nodes and redistributes examples
Slower than subtree replacement

Complexity of tree induction

Basic Assumptions:

- The training set has n instances each with m attributes.
- The tree is of depth $O(\log n)$.
- Most instances are different from each other thus m-tests is enough to differentiate an instance.

The cost for building a tree is $O(mn \log n)$.
Complexity for subtree replacement: $O(n)$
Complexity for subtree raising: $O(n(\log n)^2)$.

- Every instance may have to be redistributed at every node between its leaf and the root $O(\log n)$, thus in total: $O(n \log n)$
- Cost for redistribution (on average): $O(\log n)$

Total cost: $O(mn \log n) + O(n(\log n)^2)$

Comments on C4.5

C4.5 greedily prunes conditions from each rule if this reduces their estimated error.

- Duplicates have to be removed;

Can be slow for large and noisy data set

- Commercial version C5.0 uses a faster yet more accurate technique.

C4.5 offers two parameters

- ◦ The confidence value (default 25%): lower values incur heavier pruning
- ◦ A threshold on the minimum number of instances in each branch(default 2).

Comments: Probably the most popular ML algorithm used in data mining.

Various criteria for attribute/ test selection make a big difference.

Different pruning methods change the size of the resulting pruned tree.

Last, C4.5 builds univariate decision trees while some other TDIDT systems can build multi-variate tree.

Classification rules

Procedure: separate-and-conquer

Variants:

- Search method (e.g. greedy, beam search);
- Test selection criteria;
- Pruning method;
- Stopping criterion(min. accuracy);
- Post-processing step.

Interpretation: Decision list vs. one rule set for each class.

Test selection criteria: Accuracy: p/ t

- Try to produce rules that do not cover negative instances quickly ($1/1 \geq 999/1000$)
- May produce rules with very small coverage owing to special cases or noisy data.

Information gain: $p[\log(p/t) - \log(P/T)]$ where p and t, P and T are the numbers of positive instances and coverage of the new and initial rules.

- Biased to the covered positive instances. These interact with the pruning mechanism.

Missing Val. & Num. Attrib.

Missing values: let them fail any test

- Forces algorithm to either use other tests to separate out positive instances or to leave them uncovered until later on in the process
- In some cases it's better to treat "missing" as a separate value.

Numeric attributes are treated as they are in decision trees.

Pruning rules:

- Incremental pruning and global pruning.

Other difference: pruning criterion

- Error on hold-out set
- Statistical significance
- MDL principle

Also: post-pruning vs. pre-pruning

Computing the significance

Significance measure for a rule:

- Probability of a randomly selected rule when applied to the same data set as rule R would perform at least as well as R.

Default Rule: R1 (P, T) New Rule: R2(t,p)
Question: If R1 is applied to a subset with size t, what's the probability that R1 performs better than R2?
Hypergeometric distribution:

$$Pr[(i,t),(P,T)] = \frac{C(P,i)C(T-P,t-i)}{C(T,t)}$$

gives the probability that there are i positive instances out of t randomly selected instances from a data set with (P,T)-distribution, and $C(n,k)$ is the combinatorial function

$$C(n,k) = n!/(k! * (n-k)!).$$

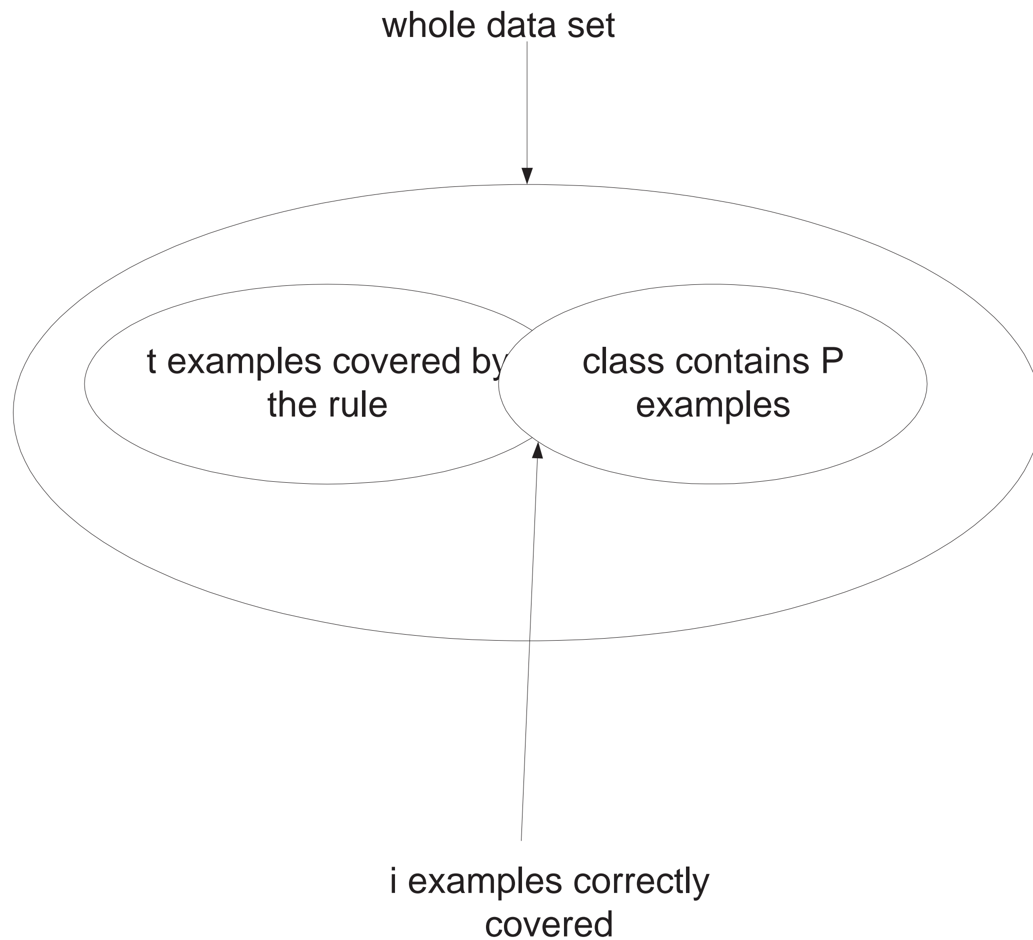
Thus the probability of R1 will over-perform R2 is

$$m(R) = \sum_{i=p}^{\min\{t,P\}} Pr[(i,t),(P,T)].$$

An approximation to the probability is

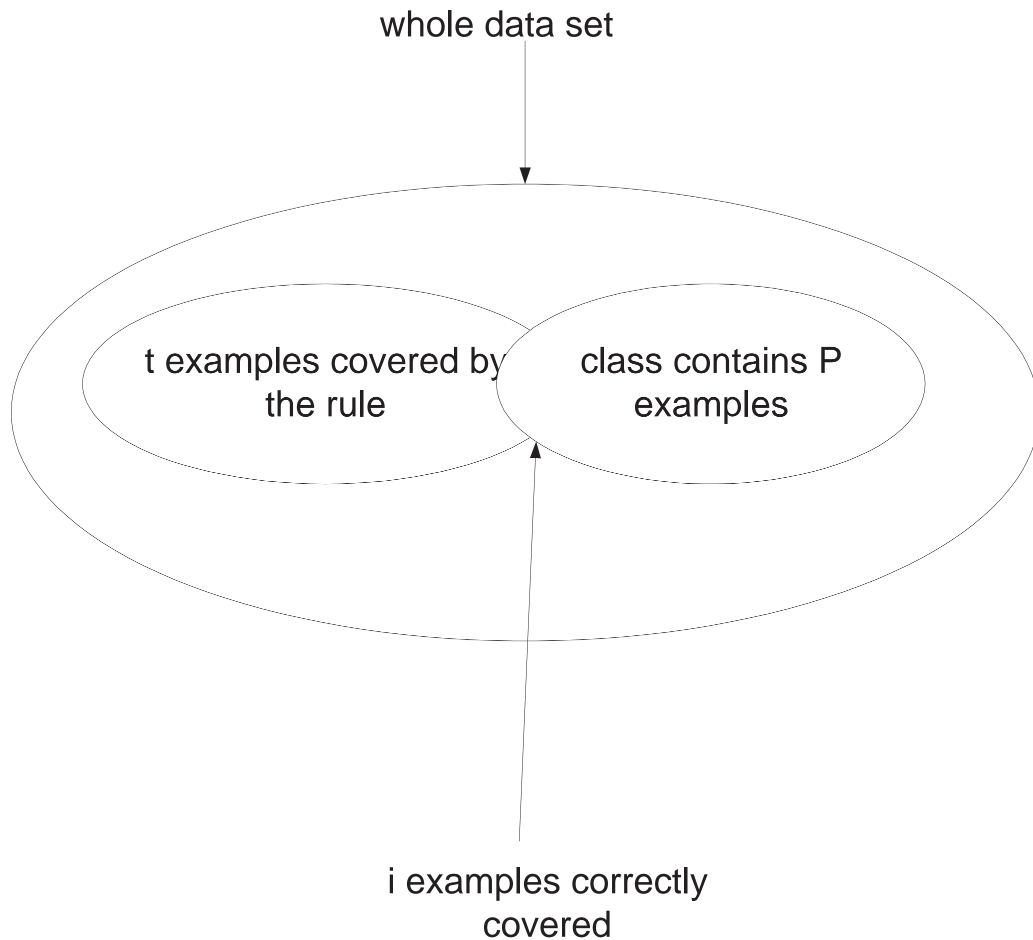
$$Pr[(i,t),(P,T)] = C(t,i)\frac{P^i(T-P)^{t-i}}{T^t}.$$

Hypergeometric Distribution



The hypergeometric distribution:
C(i,P)*C(T-P,t-i)/C(T,t)

Binomial Distribution



Goodness of rules

Criteria: The smaller the probability is, the better the new rule!

Examples: the lens contact problem
The defaulted rule:

if ? then recommendation=hard(4/24)

New rule:

If astigmatism=yes and tear production rate=normal then recommendation=hard (4/6)

Using hypergeo. distribution, this improvement has a probability

$$Pr[(4,6),(4,24)] = \frac{C(24-4,6-4)}{C(24,6)} = 0.0014.$$

If astigmatism=yes and tear production rate=normal and age=young then recommendation=hard (2/2)

The probability of this improvement is

$$Pr[(2,2),(4,24)] = \frac{C(4,2)}{C(24,2)} = 2.3\%,$$

not as impressive as that to (4/6). Adding the test *age=young* is too restrictive.

Generating good rules

A bad example is

If astigmatism=yes and tear production rate=normal then recommendation=none (2/6)

while the default rule is (15/24), the probability of this change is

$$Pr[(2,6),(15,24)] = \frac{C(15,2)C(9,4)}{C(24,6)} = 98.5\%.$$

The best rule for the contact lens problem is

If astigmatism=no and tear production rate=normal then recommendation=soft (5/6)

improve from the default rule (5/24) with a probability

$$Pr[(5,6),(5,24)] = \frac{C(24-5,6-5)}{C(24,6)} = 0.01\%.$$

A perfect rule is a rule makes no errors on the training set. Perfect rules can be derived by adding tests if there is no inconsistent instances in the training set. But a perfect rule might be overspecialized!

INDUCT

Performs incremental pruning:

- Initialize E to the instance set;

- Until E is empty do

  1. For each class C for which E contains an instance
     - Use basic covering algorithm to create best perfect rule for C;
     - Calculate significance $m(R)$ for rule and significance $m(R-)$ for rule with final condition omitted;
     - If $(m(R-) < m(R))$, prune rule and repeat previous step;
  2. From the rules for the different classes, select the most significant one (i.e. the one with smallest $m(R)$);
  3. Print the rule;
  4. Remove the instances covered by rule from E;

- Continue;

INDUCT combines basic covering and probability to find the best rule.

Ques. Can the basic covering provide the best candidate rule for pruning?

Ans. Not sure! But a comprehensive search is too time-consuming!

Using a pruning set

Make the pruning statistically sensible:

- Needs a growing set and a pruning set

Reduced-error pruning: builds a full unpruned rule set and simplifies it subsequently
Incremental reduced-error pruning: simplifies a rule right after it has been built

- Re-split data after rule has been pruned?

The hypergeometric distribution is nice, but too complex to compute.
Other alternatives: Let $N = T - P$, and $n = t - p$. If we apply the new rule (R2) to the original data set with T instances, then it has an overall success ratio $[p+(N-n)]/T$.

- Treats the negative instances in T equivalently as correctly classified by the rule.

A **Bad** Example:
$R1(2000, 3000), R2(1000, 1001),\ S \sim (P, T)$
R1 is better than R2, because in the first case, $p + (N - n) = N + 1000$, while for the second one, $p + (N - n) = N + 999$.

Incre. reduced-error pruning

- Initialize E to the instance set
- Divide E into Grow and Prune in the ratio 2 : 1
  1. For each class C for which Grow and Prune both contain an instance;
     - Use basic covering algorithm to create best perfect rule for C;
     - Calculate worth $w(R)$ for rule on Prune and worth $w(R-)$ for rule with final condition omitted;
     - If $w(R-) < w(R)$, prune rule and repeat previous step;
  2. From the rules for the different classes, select the one that's worth most (i.e. the one with the largest $w(R)$);
  3. Print the rule;
  4. Remove the instances covered by rule from E;
- Continue

Difference from INDUCT: An independent pruning set and various measures are employed in the new algorithm.

Real Learners

Speedup the process:

- Generating rules for classes in order (as we did in the basic covering approach!);
- Usually starting with the smallest class and leaving the largest class covered by the default rule;
- Stop if accuracy becomes too low;
- Used in Rule learners: **RIPPER** and **C4.5**

Over-pruning:

Rule	Data		Set	
	Growing set		Pruning set	
	Yes	No	Yes	No
$a = 1 \Rightarrow Yes$	90	8	30	5
$a = 0, b = 1 \Rightarrow Yes$	200	18	66	6
$a = 0, b = 0 \Rightarrow No$	1	10	0	3

The first rule '$a = 1 \Rightarrow Yes'$ will be pruned to '? $\Rightarrow Yes'$, Over-pruned!

Partial Tree

- Avoids global optimization steps as used other learners:
  Generates an unrestricted decision list using separate-and-conquer;
- Builds a partial decision tree to obtain a rule:
  A rule is only pruned if all its implications are known(avoid over-pruning) ;

Procedure

- Choose a test and and divide the whole data set into subsets;
- Sort subsets into increasing order of average entropy;
  - Expand the purest subset;
  - Pruning on the expanded tree and make it into a node;
- Remove the instances covered by the rule, and repeat the above process for the remaining data set.

Building a partial tree

Expand-subset(S):

- Choose test T and use it to split set of examples into subsets;
- Sort subsets into increasing order of average entropy;
- **While** (there is a subset X that has not yet been expanded AND all subsets expanded so far are leaves) **expand** subset (X);
- if (all the subsets expanded are leaves AND estimated error for subtree $\geq$ estimated error for node)
  undo expansion into subsets and make node a leaf;

Notes: Leaf with maximum coverage is made into a rule;

Missing values are treated as in **C4.5**;

- Instance is split into pieces;

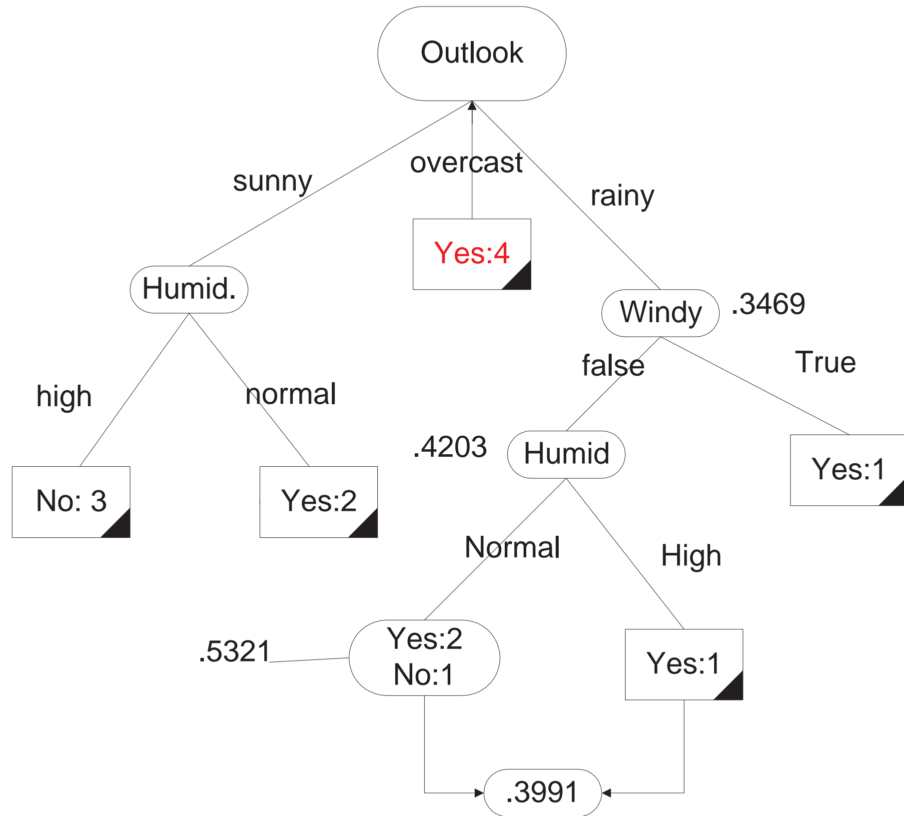
Time complexity for generating a rule:

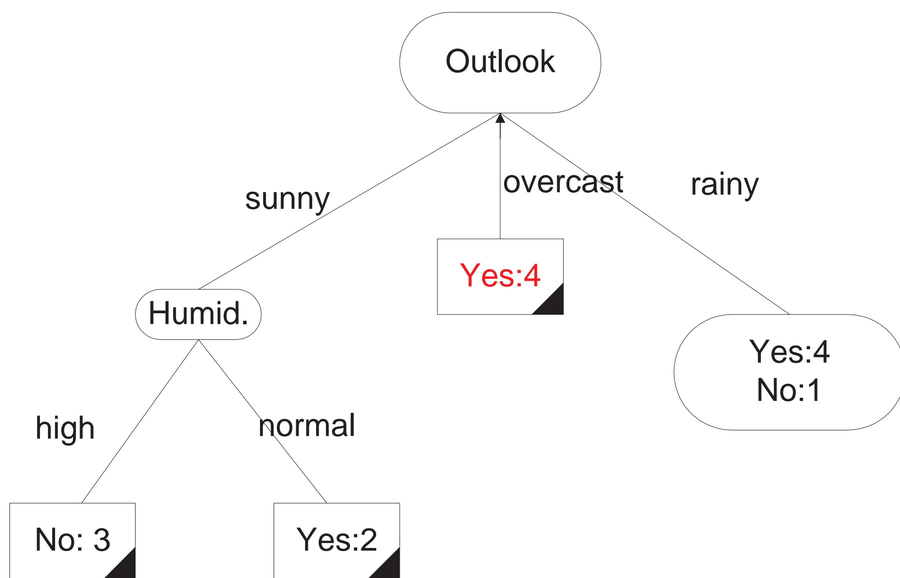
- Worst case: same as for building a pruned tree(occurs for noisy data);
- Best case: same as for a single rule(occurs when data is noise free).

The weather problem:I

Outlook	Temper.	humidity	windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	**Yes**
Rainy	**Cool**	**Normal**	**False**	**Yes**
Rainy	**Cool**	**Normal**	**False**	**No**
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	**Yes**

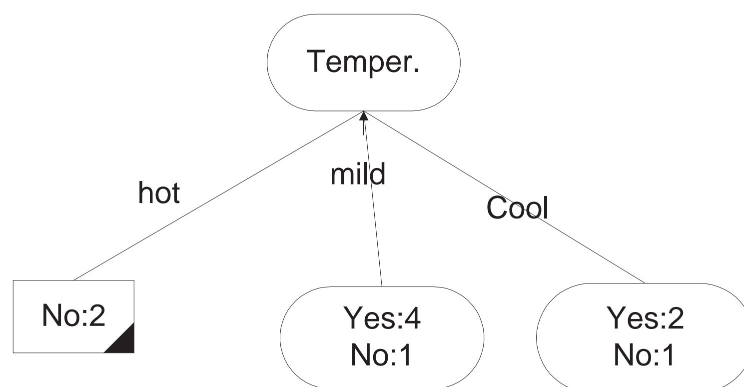
A partial tree





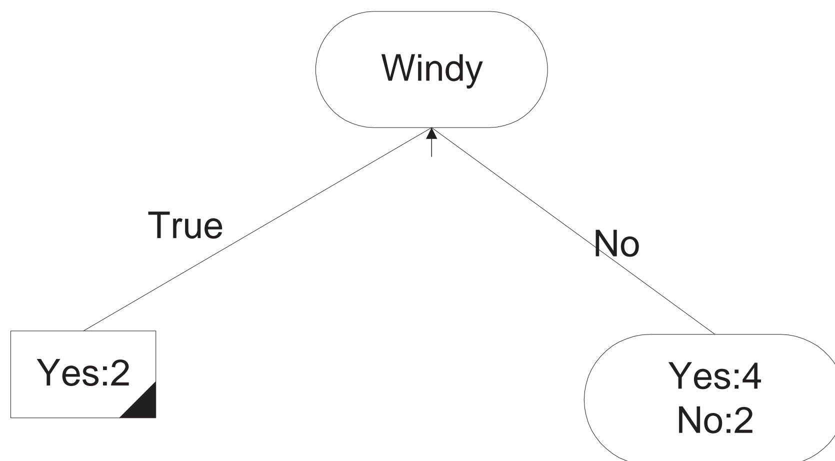
The weather problem:II

Outlook	Temper.	humidity	windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Rainy	Mild	High	False	**Yes**
Rainy	**Cool**	**Normal**	**False**	**Yes**
Rainy	**Cool**	**Normal**	**False**	**No**
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Rainy	Mild	High	True	**Yes**



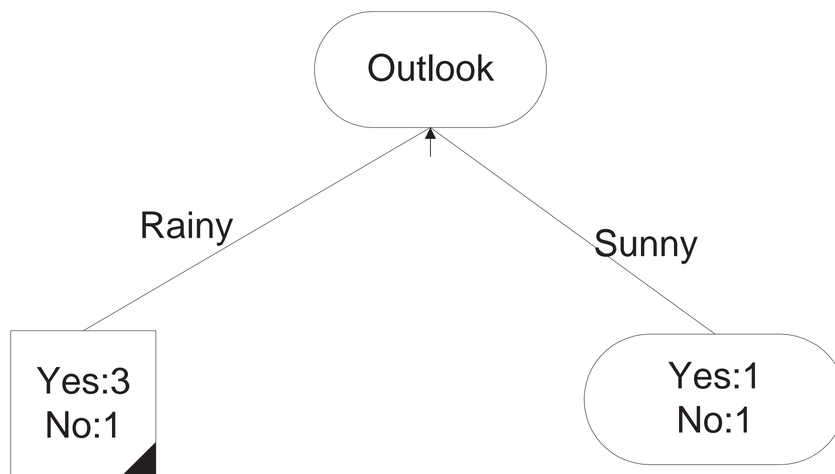
The weather problem:III

Outlook	Temper.	humidity	windy	Play
Rainy	Mild	High	False	**Yes**
Rainy	**Cool**	**Normal**	**False**	**Yes**
Rainy	**Cool**	**Normal**	**False**	**No**
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Rainy	Mild	High	True	**Yes**



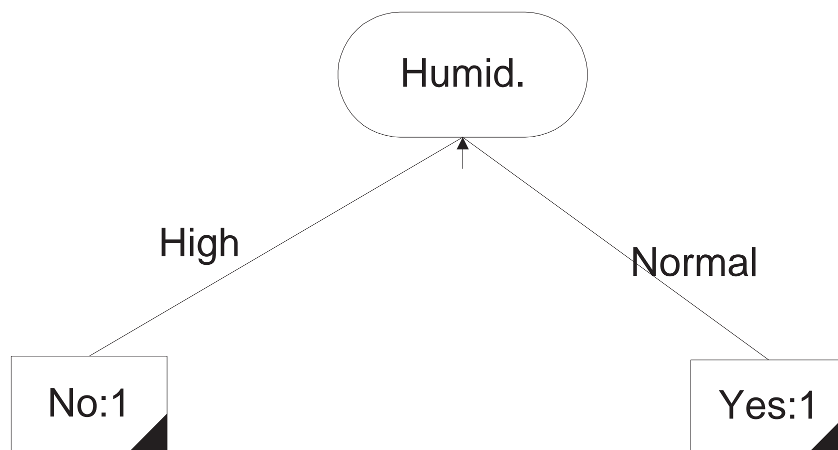
The weather problem:IV

Outlook	Temper.	humidity	windy	Play
Rainy	Mild	High	False	**Yes**
Rainy	**Cool**	**Normal**	**False**	**Yes**
Rainy	**Cool**	**Normal**	**False**	**No**
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes

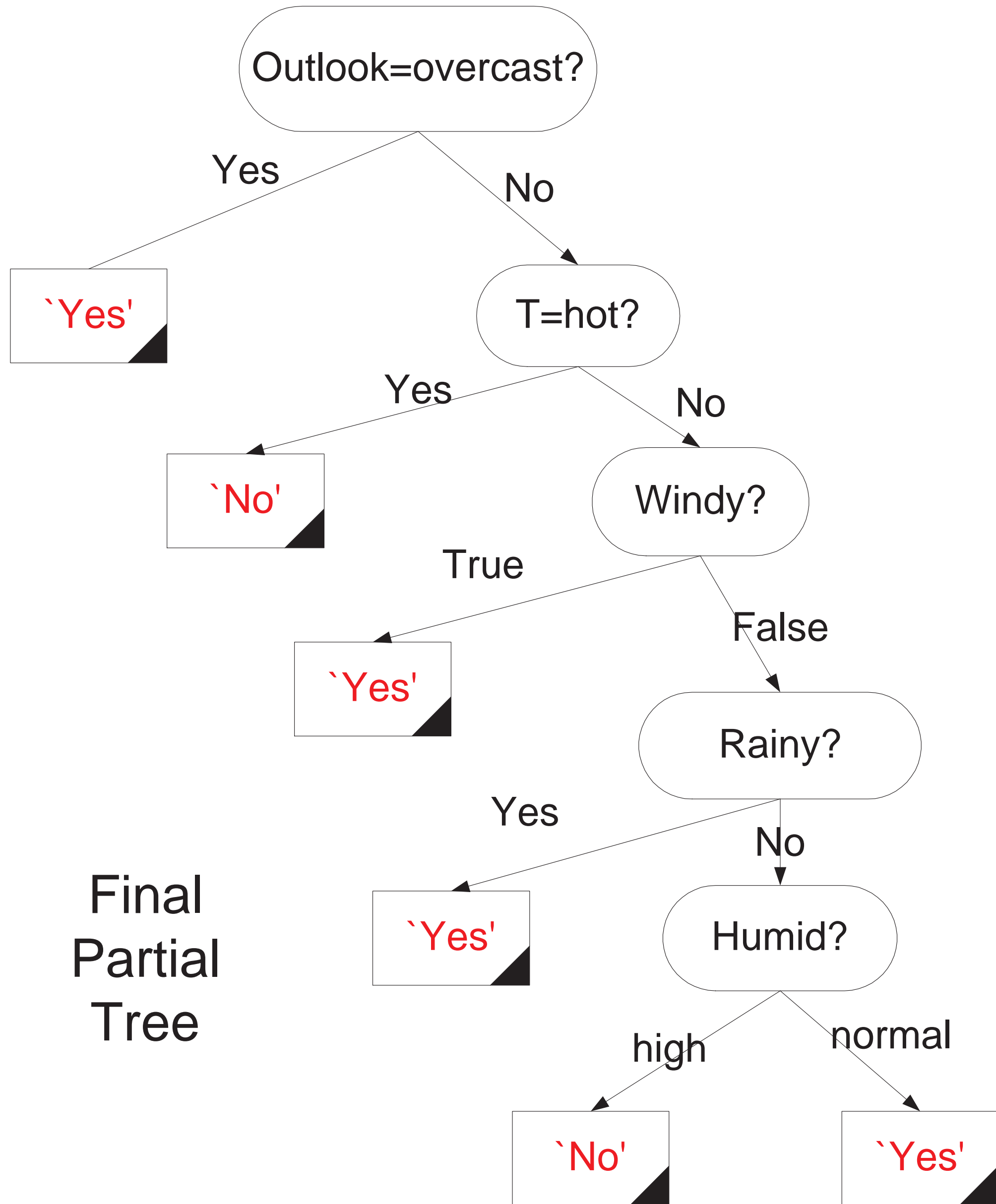


The weather problem:V

Outlook	Temper.	humidity	windy	Play
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes



Final Partial Tree



Rules with exceptions

Assume we have a method for generating a single good rule, then it's easy to generate rules with exceptions.
First: default class is selected for top-level rule. Then we generate a good rule for one of the remaining classes. In this way we separate the set into two parts: one subset whose instances are covered by the tests in the rule and another subset whose instances are not covered by the rule.

We then apply this method recursively to the two subsets produced by the rule. For the subset whose instances are covered by the tests of the rule, the default class is the the class specified by the rule. For the uncovered subset, the default class remains as it was before.

FP-Tree for Association

Observation: Apriori' Algorithm have to deal with a lot of candidate sets.

10^4 1-item sets will generate more than 10^7 candidate 2-item sets!

Need to scan the whole data set to get the support for all the 2-item sets!

Ques: How to speed up the process?

Frequent-pattern(FP-) growth method:

- Divide and conquer;
- Compress the f-items into FP-tree;
- Retain the itemset information;
- Divide the compressed database into a set of conditional databases, each associated with one frequent item;
- Mine every such database separately.

Mining using FP-Tree

The algorithm: FP-Tree

Step 0: Scan the date set the get the frequent 1-item sets and sort them in the order of descending support count;
Step 1: Construct FP-tree;
Step 2: Mining the FP-tree.

Constructing FP-tree

- Create the root with ‘null’;
- Scan each instance whose items are sorted to construct the FP-tree
  - for each instance associated branch:
    - The count of each node along a common prefix is increased by one,
    - Nodes for the items following the prefix are created and linked accordingly.

FP-Growth: Examples

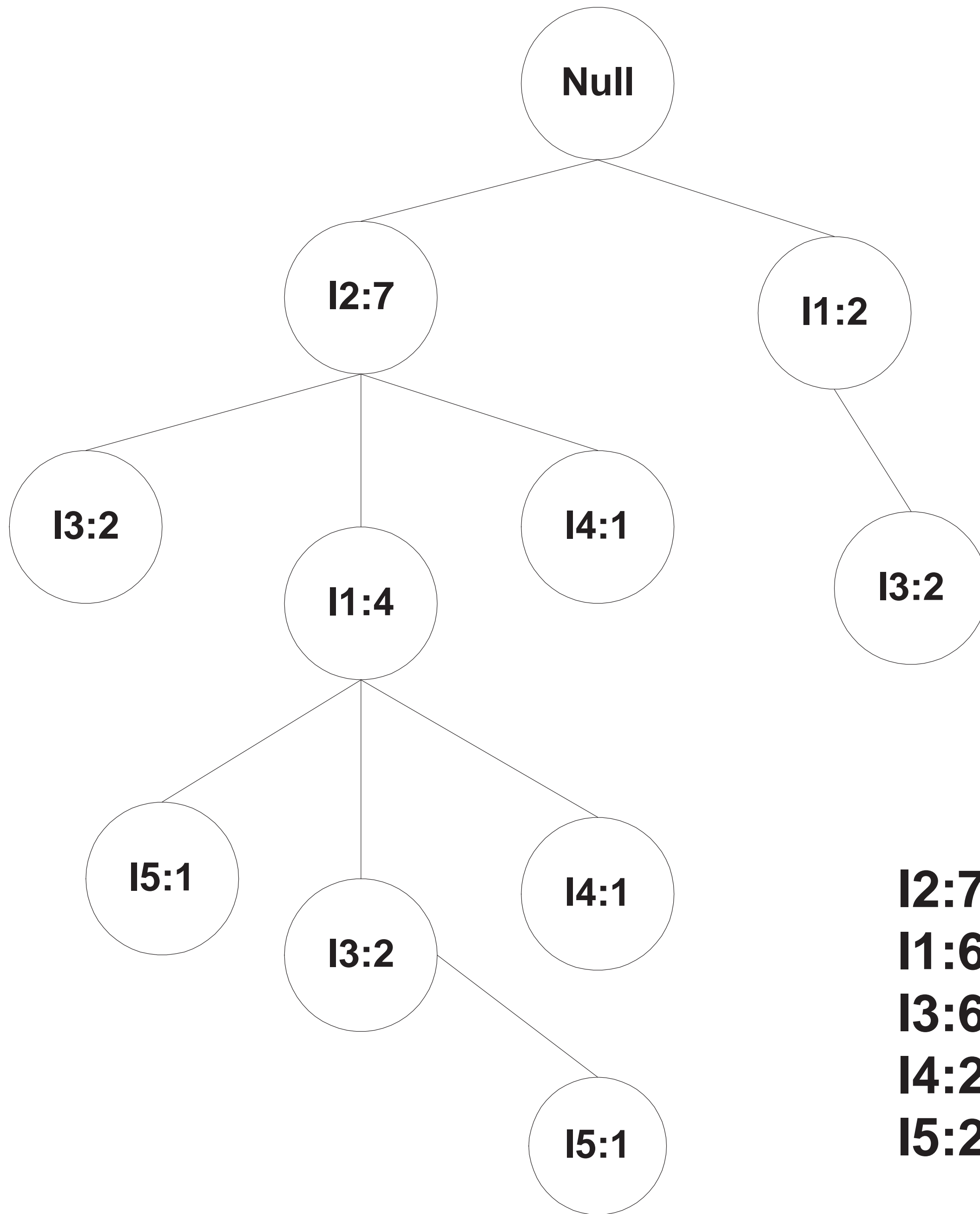
TID	Items
T1	I1, I2, I5
T2	I2, I4
T3	I2, I3
T4	I1, I2, I4
T5	I1, I3
T6	I2, I3
T7	I1, I3
T8	I1,I2,I3,I5
T9	I1, I2,I3

Sorted

1-item set

Item	Sup.
I2	7
I1	6
I3	6
I4	2
I5	2

FP-tree



FP-tree for Transaction Data

Mining FP-tree

Conditional Pattern Base: a sub-database which consists of the set of prefix paths in the FP-tree corresponding to the suffix pattern.

Conditional FP-tree: A tree built upon the conditional pattern base.

Mining $FP_{growth}(Tree, null)$

If $Tree$ contains a single path P then
  For each combination β of nodes in P
    generate pattern $\beta \cup \alpha$ with

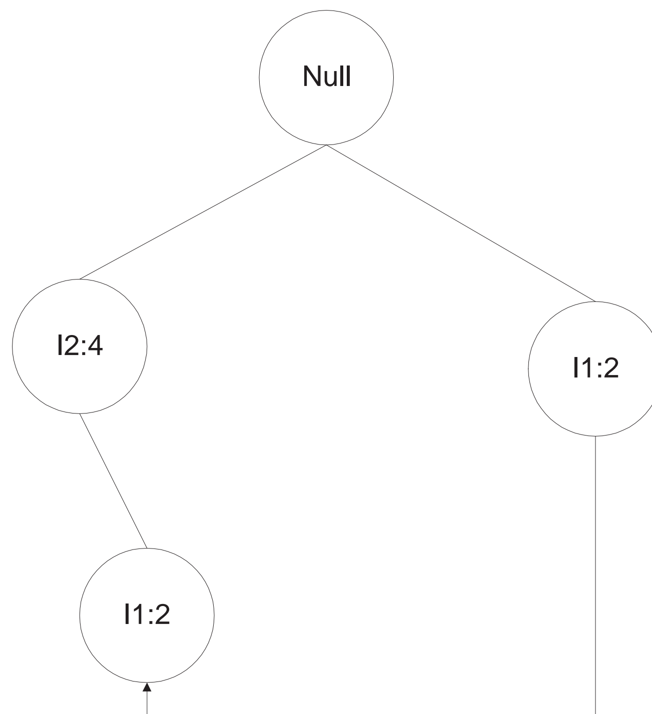
$$support = \text{min } support\ of\ nodes\ in\ P;$$

Else for each a_i in the header of $Tree$
  generate pattern $\beta = a_i + \alpha$ with $support = a_i.support$;
  construct β's conditional pattern base and then
  construct β's conditional $FP_{Tree} : Tree(\beta)$
  If $Tree(\beta) \neq \phi$ then
    Call $FP_{growth}(Tree(\beta), \beta)$

Cond. Patterns and Trees

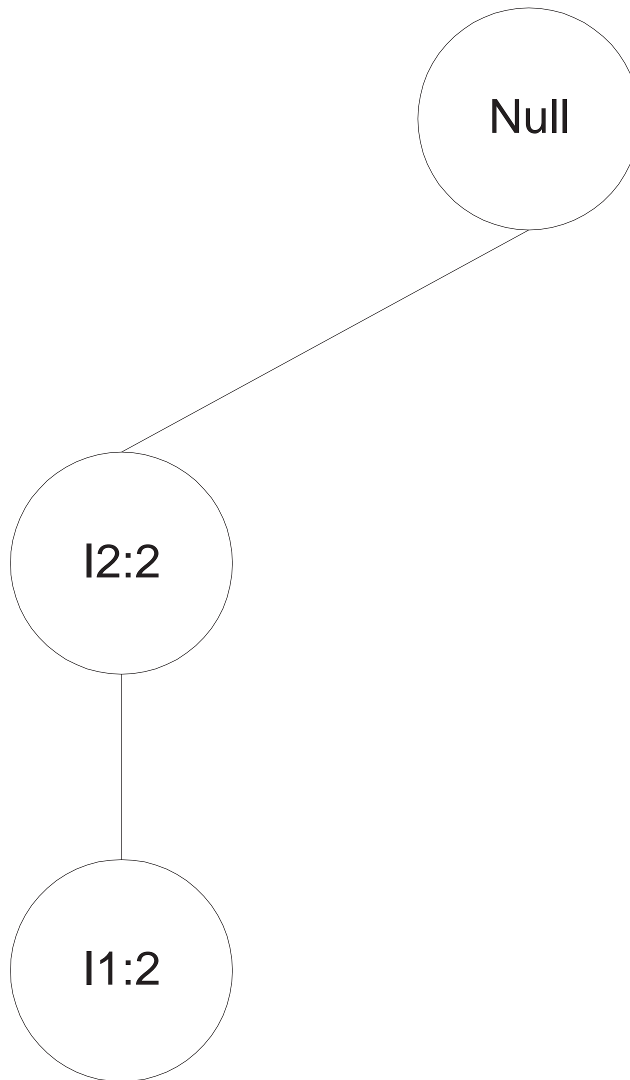
Mining the FP-tree by creating conditional pattern bases:

Item	cond. pat.	cond. tree	freq. pat.
I5	$I_2I_1(1)$ $I_2I_1I_3(1)$	$(I_2, I_1 : 2)$	$I_2I_5(2), I_1I_5(2)$ $I_2I_1I_5(2)$
I4	$I_2I_1(1), I_2(1)$	$(I_2 : 2)$	$I_2I_4(2)$
I3	$I_2I_1(2)$ $I_2(2), I_1(2)$	$(I_2 : 4, I_1 : 2)$ $(I_1 : 2)$	$I_2I_3(4), I_1I_3(4)$ $I_2I_1I_3(2)$
I1	$I_2(4)$	$(I_2 : 4)$	$I_2I_1(4)$



Conditional FP-tree for I3

Conditional Trees



Conditional FP-tree for I5