Machine Learning Techniques for Data Mining

Eibe Frank
University of Waikato
New Zealand

PART VI

Implementations: Real machine learning schemes

Industrial-strength algorithms

- Requirements for an algorithm to be useful in a wide range of real-world applications:
  - Can deal with numeric attributes
  - Doesn’t fall over when missing values are present
  - Is robust in the presence of noise
  - Can (at least in principle) approximate arbitrary concept descriptions
- Basic schemes (may) need to be extended to fulfill these requirements

Decision trees

- Extending ID3 to deal with numeric attributes: pretty straightforward
- Dealing sensibly with missing values: a bit trickier
- Stability for noisy data: requires sophisticated pruning mechanism
- End result of these modifications: Quinlan’s C4.5
- Best-known and (probably) most widely-used learning algorithm
- Commercial successor: C5.0

Numeric attributes

- Standard method: binary splits (i.e. temp < 45)
- Difference to nominal attributes: every attribute offers many possible split points
- Solution is straightforward extension:
  - Evaluate info gain (or other measure) for every possible split point of attribute
  - Choose “best” split point
  - Info gain for best split point is info gain for attribute
- Computationally more demanding

An example

- Split on temperature attribute 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

  - Eg. 2 `yeses` and 2 `nos` for `temperature < 71.5` and 4 `yeses` and 2 `nos` for `temperature ≥ 71.5`
    - Info([4,2],[5,3]) = (6/14)info([4,2]) + (8/14)info([5,3]) = 0.939 bits
- Split points are placed halfway between values
- All split points can be evaluated in one pass!

Avoiding repeated sorting

- Instances need to be sorted according to the values of the numeric attribute considered
  - Time complexity for sorting: $O(n \log n)$
- Does this have to be repeated at each node?
- No! Sort order from parent node can be used to derive sort order for children
  - Time complexity of derivation: $O(n)$
  - Only drawback: need to create and store an array of sorted indices for each numeric attribute

Notes on binary splits

- Information in nominal attributes is exhausted using one multi-way split on that attribute
- This is not the case for binary splits on numeric attributes
  - The same numeric attribute may be tested several times along a path in the decision tree
- Disadvantage: tree is relatively hard to read
- Possible remedies: pre-discretization of numeric attributes or multi-way splits instead of binary ones

Computing multi-way splits

- Simple and efficient way of generating multi-way splits: greedy algorithm
- Optimum multi-way splits (for additive criteria) can be found using dynamic programming in $O(n^2)$
    - Let IMP(k,i,j) be the impurity of the best split of values $x_i,\ldots,x_i$ into k sub-intervals
    - IMP(k,i,j) = $\mathrm{MIN}_{0<j<i}$\{IMP(k-1,1,j)+IMP(1,j+1,i)\}
    - IMP(k,1,N) gives us the best k-way split
- In practice, greedy algorithm works as well

Missing values

- C4.5 splits instances with missing values into pieces (with weights summing to 1)
  - A piece going down a particular branch receives a weight proportional to the popularity of the branch
- Info gain etc. can be used with fractional instances using sums of weights instead of counts
- During classification, the same procedure is used to split instances into pieces
  - Probability distributions are merged using weights

Pruning

- Pruning simplifies a decision tree to prevent overfitting to noise in the data
- Two main pruning strategies:

1. *Postpruning*: takes a fully-grown decision tree and discards unreliable parts
2. *Prepruning*: stops growing a branch when information becomes unreliable

- Postpruning preferred in practice because of *early stopping* in prepruning

Prepruning

- Usually based on statistical significance test
- Stops growing the tree when there is no *statistically significant* association between any attribute and the class at a particular node
- Most popular test: *chi-squared test*
- ID3 used chi-squared test in addition to information gain
  - Only statistically significant attributes where allowed to be selected by information gain procedure

Early stopping

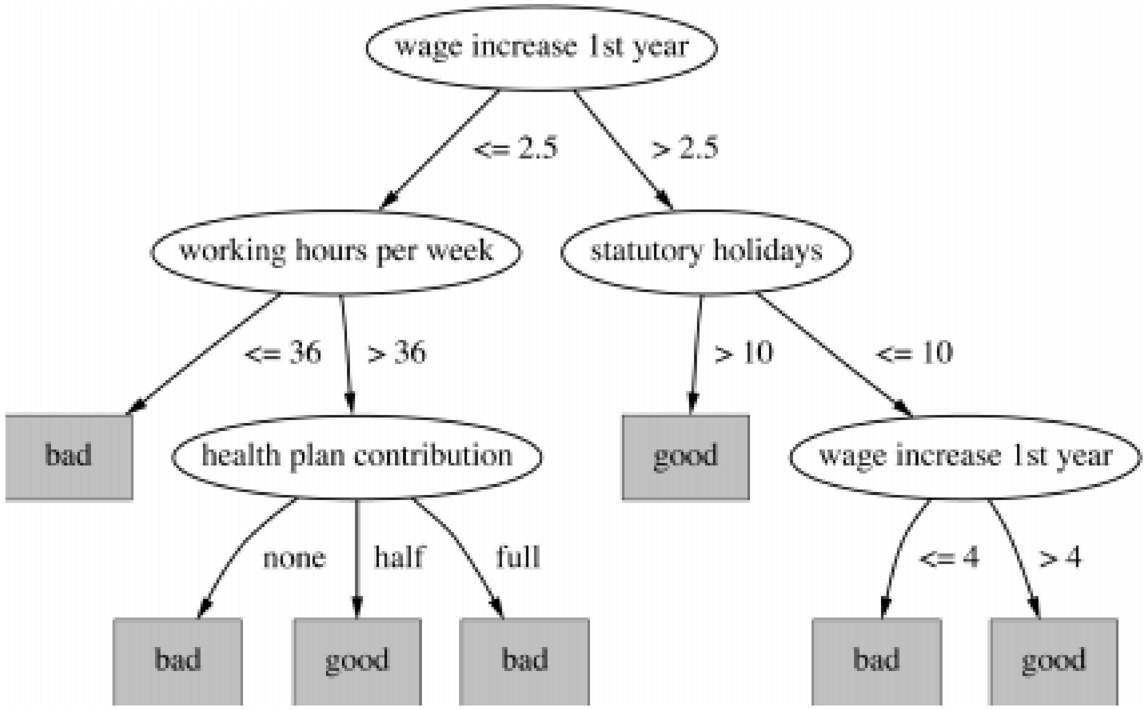
- Pre-pruning may suffer from early stopping: may stop the growth process prematurely
- Classic example: XOR/Parity-problem
  - No *individual* attribute exhibits any significant association to the class
  - Structure is only visible in fully expanded tree
  - Prepruning won't expand the root node
- But: XOR-type problems not common in practice
- And: prepruning faster than postpruning

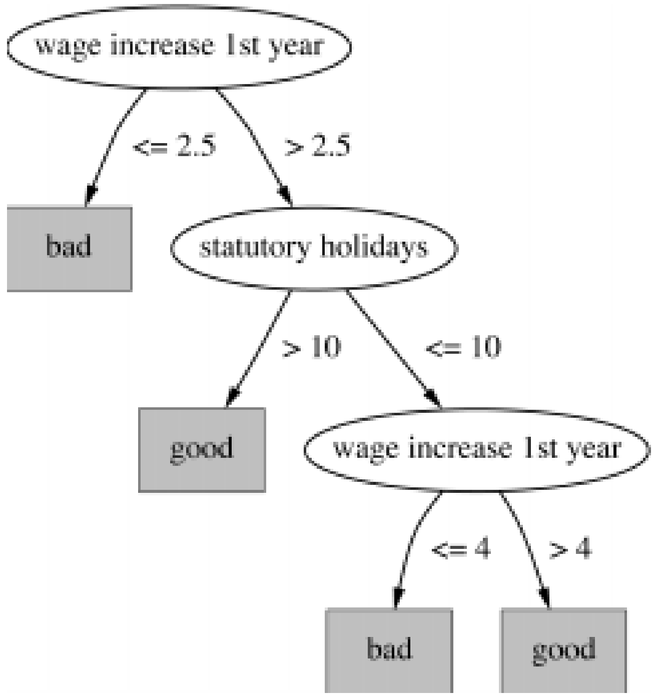
Postpruning

- Builds full tree first and prunes it afterwards
  - 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*
  2. *Subtree raising*
- Possible strategies: error estimation, significance testing, MDL principle

Subtree replacement

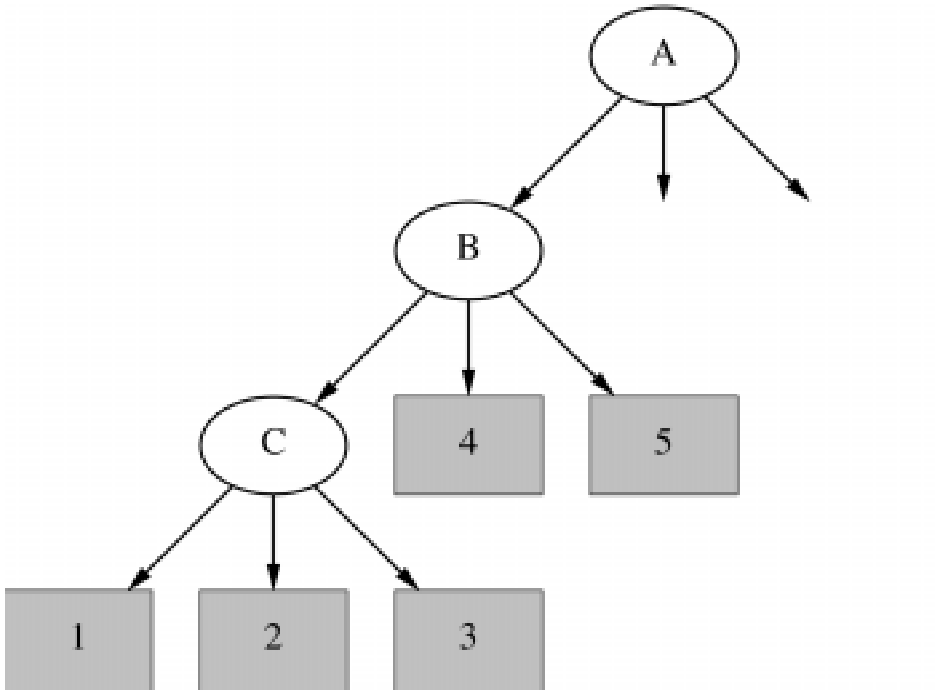
- *Bottom-up*: tree is considered for replacement once all its subtrees have been considered

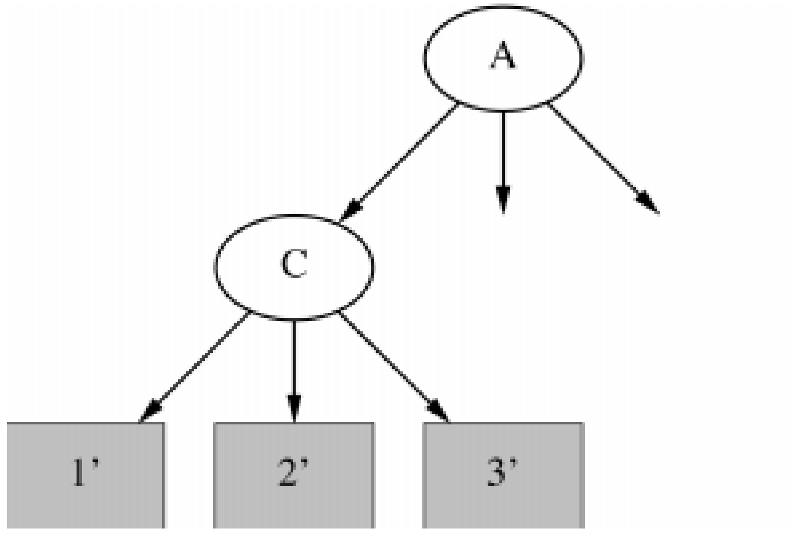




Subtree raising

- Deletes node and redistributes instances
- Slower than subtree replacement (Worthwhile?)





Estimating error rates

- Pruning operation is performed if this does not increase the estimated error
- Of course, error on the training data is not a useful estimator (would result in almost no pruning)
- One possibility: using hold-out set for pruning (*reduced-error pruning*)
- C4.5’s method: using upper limit of 25% confidence interval derived from the training data
  - Standard Bernoulli-process-based method

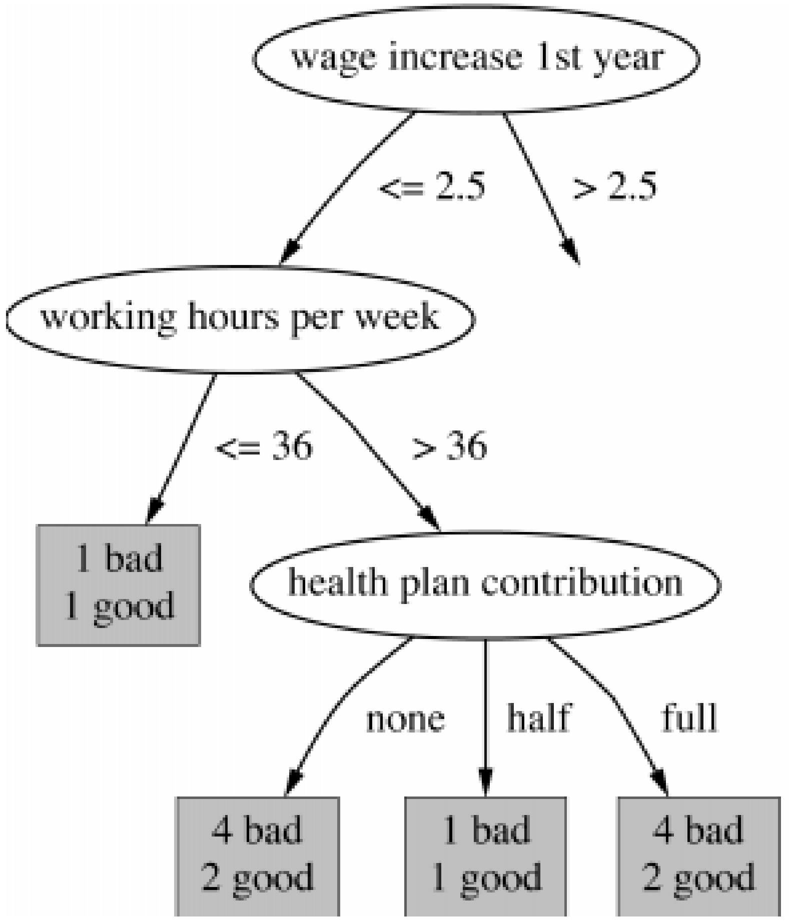
C4.5’s method

- Error estimate for subtree is weighted sum of error estimates for all its leaves
- Error estimate for a node:

$$e = \left(f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Big/ \left(1 + \frac{z^2}{N} \right)$$

- If *c* = 25% then *z* = 0.69 (from normal distribution)
- *f* is the error on the training data
- *N* is the number of instances covered by the leaf

Example



f=5/14
e=0.46

Combined using ratios 6:2:6 this gives 0.51

f=0.33
e=0.47

f=0.5
e=0.72

f=0.33
e=0.47

Complexity of tree induction

- Assume m attributes, n training instances and a tree depth of $O(\log n)$
- Cost for building a tree: $O(mn \log n)$
- Complexity of subtree replacement: $O(n)$
- Complexity of subtree raising: $O(n (\log n)^2)$
  - Every instance may have to be redistributed at every node between its leaf and the root: $O(n \log n)$
  - Cost for redistribution (on average): $O(\log n)$
- Total cost: $O(mn \log n) + O(n (\log n)^2)$

From trees to rules

- Simple way: one rule for each leaf
- C4.5rules greedily prunes conditions from each rule if this reduces their estimated error
    - This may produce duplicates which have to be removed subsequently
- Then it considers the rules for each class in turn and finds “good” subsets guided by MDL
- After that it ranks the subsets to avoid conflicts
- Finally rules are greedily removed if this decreases the error on the training data

C4.5: choices and options

- C4.5rules can be slow for large and noisy datasets
- Commercial version C5.0rules uses a different technique
  - Much faster and a bit more accurate
- C4.5 offers two parameters
  - The confidence value (default 25%): lower values incur heavier pruning
  - A threshold on the minimum number of instances in the two most popular branches (default 2)

Discussion

- TDIDT is probably the most extensively studied method of machine learning used in data mining
- Different criteria for attribute/test selection rarely make a large difference
- Different pruning methods mainly change the size of the resulting pruned tree
- C4.5 builds *univariate* decision trees
- Some TDITDT systems can build *multivariate* trees (e.g. CART)

Classification rules

- Common procedure: *separate-and-conquer*
- Differences:
  - Search method (e.g. greedy, beam search, ...)
  - Test selection criteria (e.g. accuracy, ...)
  - Pruning method (e.g. MDL, hold-out set, ...)
  - Stopping criterion (e.g. minimum accuracy)
  - Post-processing step
- Also: Decision list vs. one rule set for each class

Test selection criteria

- Accuracy: p/t
  - Attempts to produce rules that don't cover negative instances as quickly as possible
  - May produce rules with very small coverage
    - Special cases or noise?
- Information gain: $p[\log(p/t) - \log(P/T)]$
  - Puts more emphasis on number of positive instances covered
- These interact with the pruning mechanism used

Missing values, numeric attributes

- Common treatment of 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
- Note that 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

- Two main strategies:
  - *Incremental* pruning
  - *Global* pruning
- Other difference: pruning criterion
  - Error on hold-out set (*reduced-error pruning*)
  - Statistical significance
  - MDL principle
- Also: post-pruning vs. pre-pruning

INDUCT

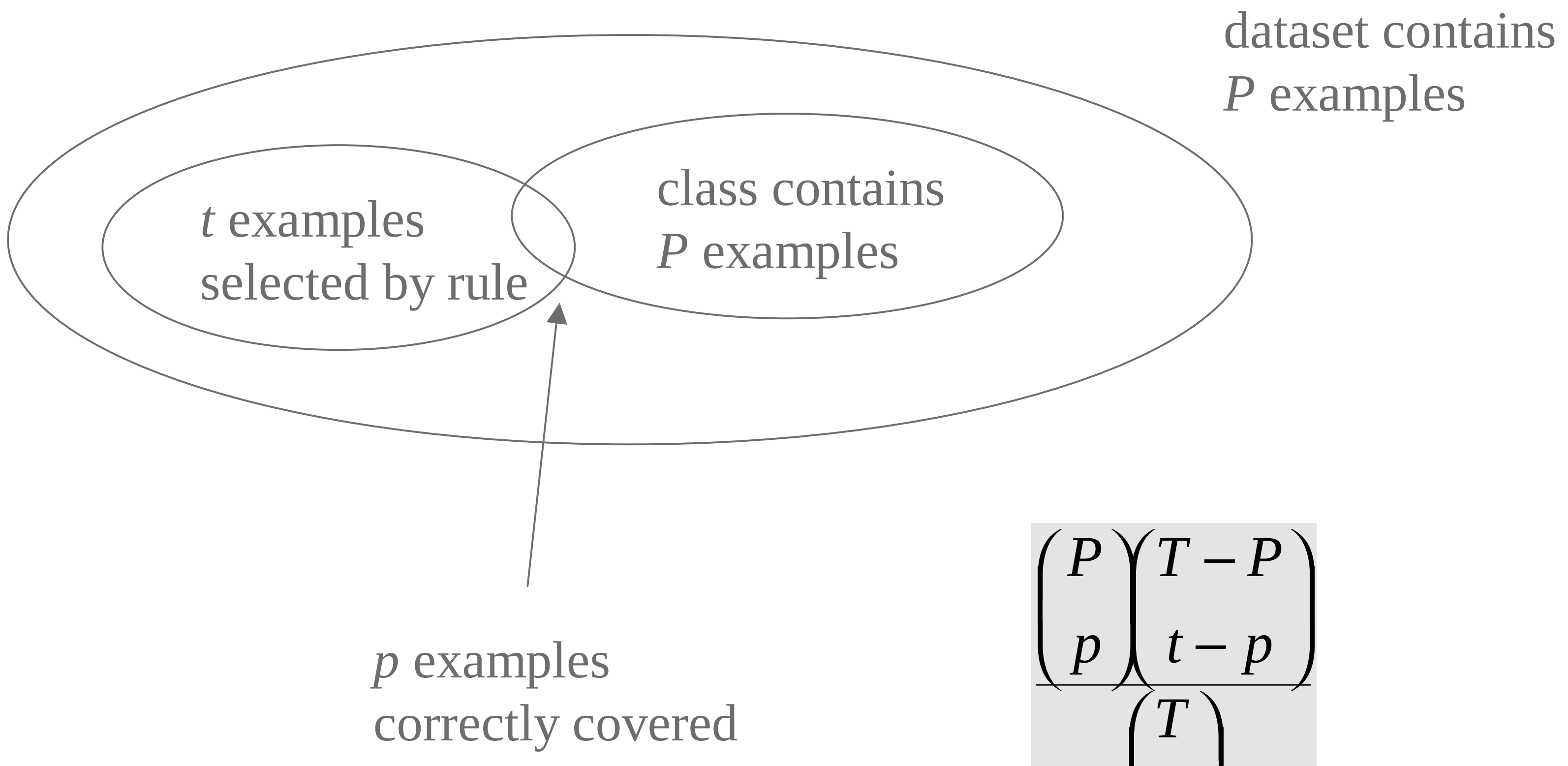
```
Initialize E to the instance set
Until E is empty do
  For each class C for which E contains an instance
    Use basic covering algorithm to create best perfect rule for C
    Calculate significance m(R) for rule and significance m(R-) for
         rule with final condition omitted
    If (m(R-)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

- Performs incremental pruning

Computing significance

- INDUCT’s significance measure for a rule:
  - Probability of completely random rule with same coverage performing at least as well
- Random rule R selects *t* cases at random from the dataset
- We want to know how likely it is that *p* of these belong to the correct class?
- This probability is given by the *hypergeometric* distribution

The hypergeometric probability



$$\frac{\binom{P}{p}\binom{T-P}{t-p}}{\binom{T}{t}}$$

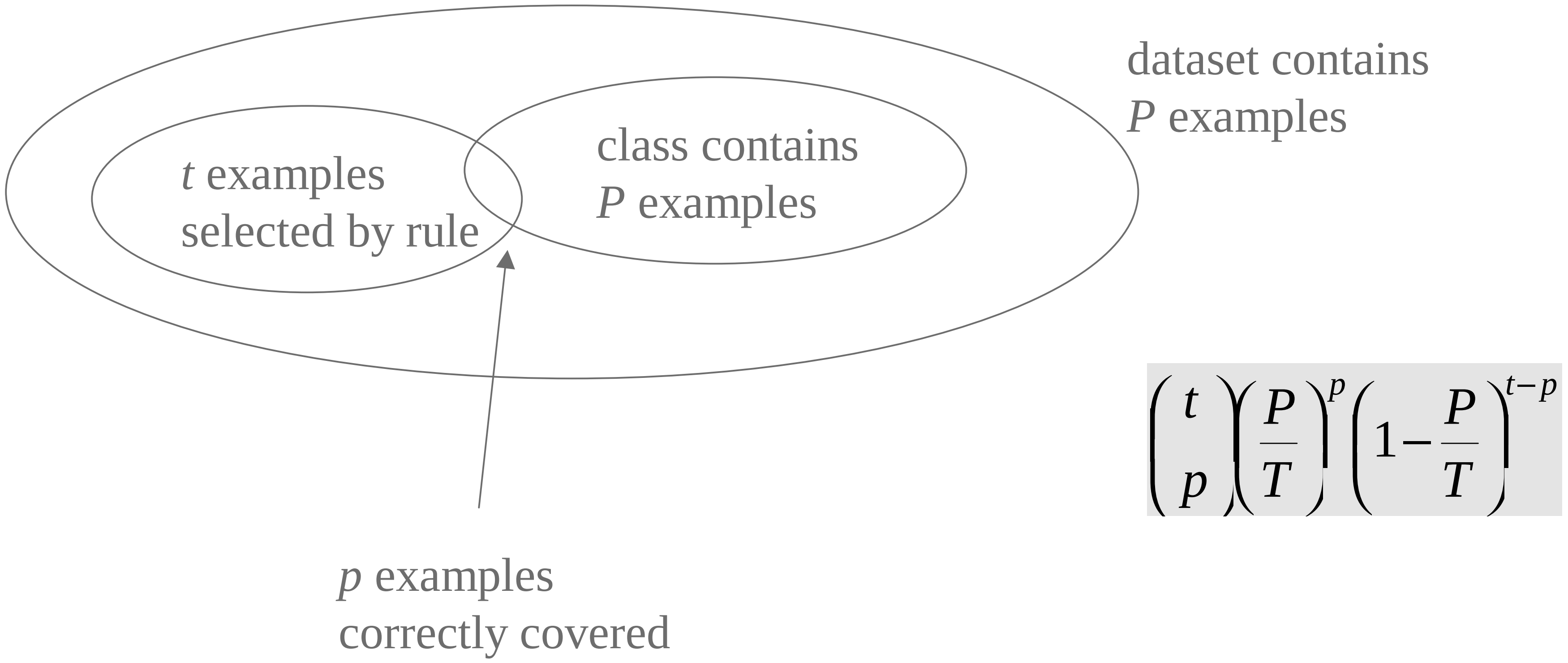
Computing significance II

- We want the probability that a random rule does *at least as well* (statistical significance of rule):

$$m(R) = \sum_{i=p}^{\min(t,P)} \frac{\binom{P}{p}\binom{T-P}{t-p}}{\binom{T}{t}}$$

The binomial distribution

- Approximation: can use sampling with replacement instead of samp. without replacement



$$\binom{t}{p}\left(\frac{P}{T}\right)^{p}\left(1-\frac{P}{T}\right)^{t-p}$$

Using a pruning set

- For measure to be valid in a statistical sense, it must be evaluated on data not used for training:
  - This requires a *growing set* and a *pruning set*
- *Reduced-error pruning* for rules builds a full unpruned rule set and simplifies it subsequently
- *Incremental reduced-error pruning* simplifies a rule immediately after it has been build
  - Can re-split data after rule has been pruned
- Stratification advantageous

Incremental reduced-error pruning

```
Initialize E to the instance set
Until E into Grow and Prune in the ratio 2:1
  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
  From the rules for the different classes, select the one that’s
      worth most (i.e. the one with the largest w(R))
  Print the rule
  Remove the instances covered by rule from E
Continue
```

Measures used in IREP

- $[p+(N-n)]/T$ (with N being the total #negatives)
  - Is counterintuitive:
    - $p = 2000$ and $n = 1000$ vs. $p = 1000$ and $n = 1$
- p/t
  - Problem: $p = 1$ and $t = 1$ vs. $p = 1000$ and $t = 1001$
- $(p-n)/t$
  - Has the same effect as success rate because it is equal to $2p/t-1$

Variations

- Generating rules for classes in order
  - Usually starting with the smallest class and leaving the largest class covered by the default rule
- Stopping criterion
  - Stop rule production if accuracy becomes too low
- Rule learner RIPPER:
  - Uses MDL-based stopping criterion
  - Employs post-processing step to modify rules guided by MDL criterion

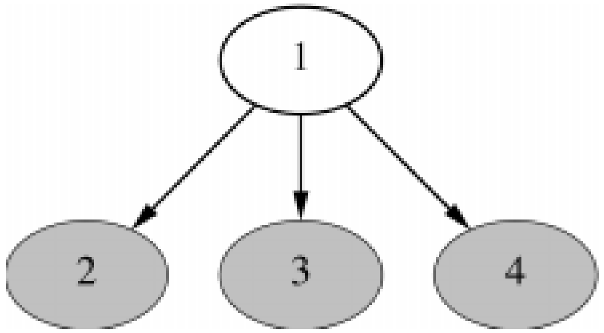
PART

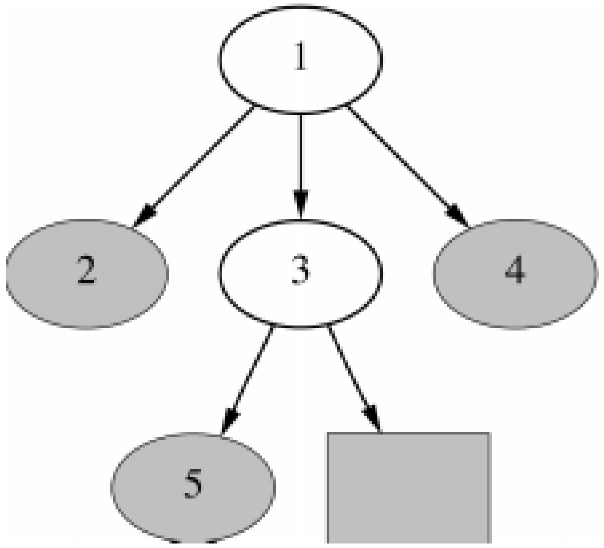
- Avoids global optimization step used in C4.5rules and RIPPER
- Generates an unrestricted decision list using basic separate-and-conquer procedure
- Builds a *partial* decision tree to obtain a rule
  - A rule is only pruned if all its implications are known
  - Prevents *hasty generalization*
- Uses C4.5's procedures to build a tree

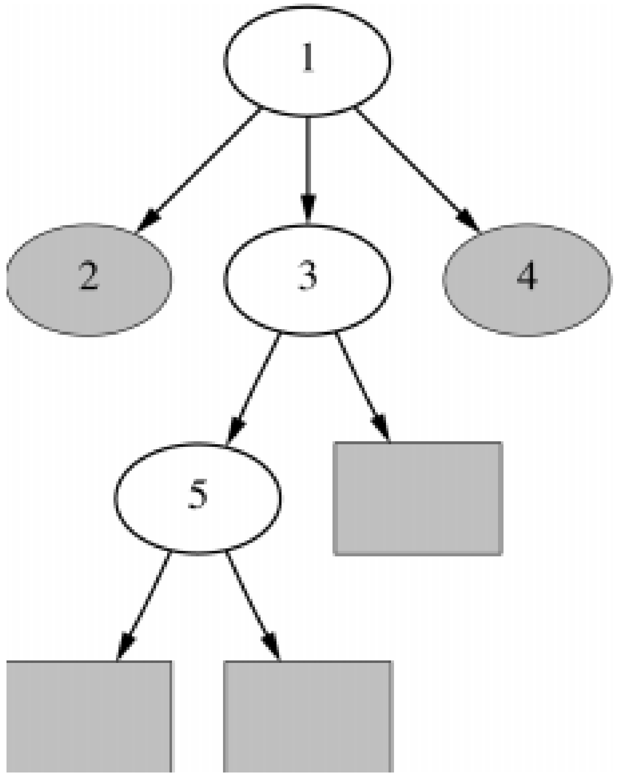
Building a partial tree

```
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 >= estimated error for node)
    undo expansion into subsets and make node a leaf
```

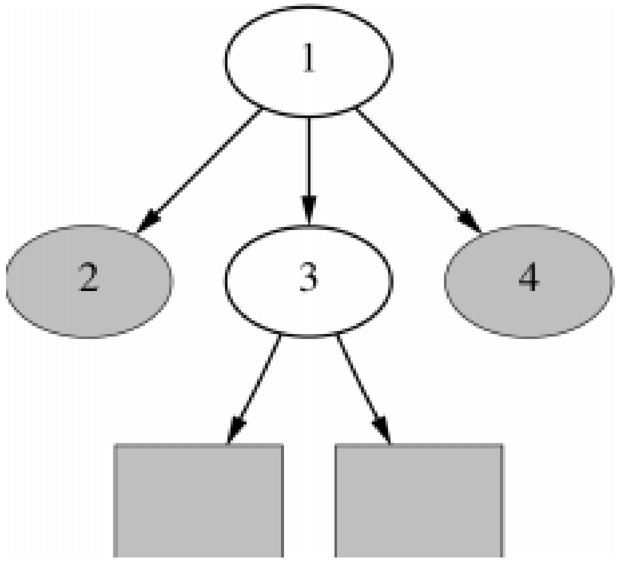
Example

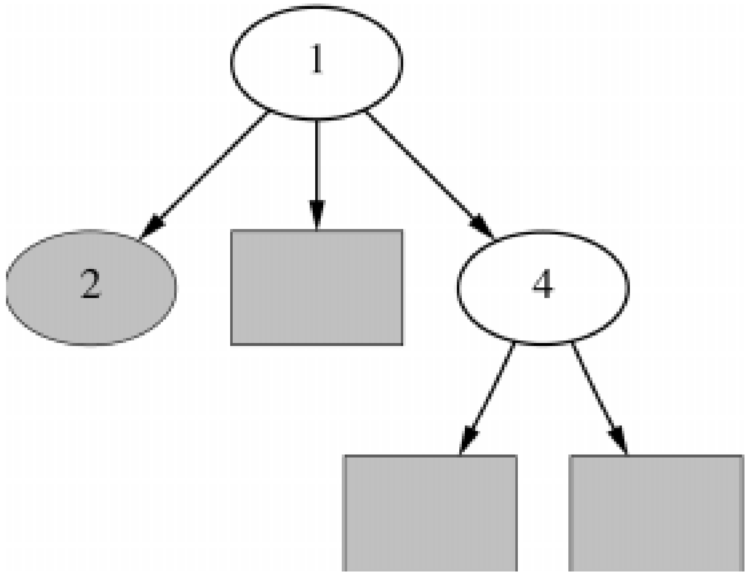






Example (continued)





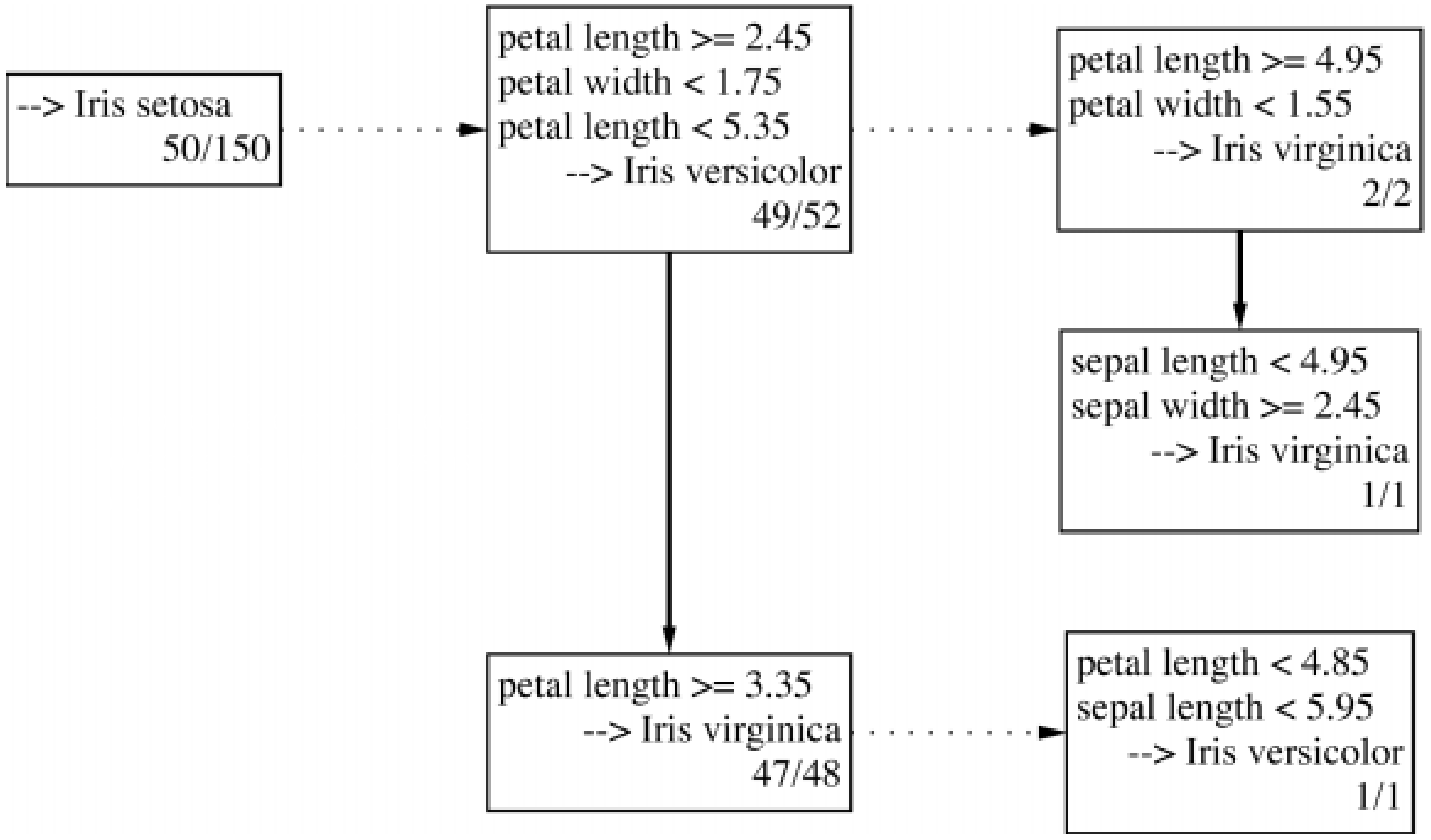
Notes on PART

- Leaf with maximum coverage is made into a rule
- Missing values are treated using C4.5’s procedure
  - I.e. instance is split into pieces
- Time complexity for generating a rule:
  - Worst case: same as for building a pruned tree
    - Occurs when data is noisy
  - Best case: same as for building a single rule
    - Occurs when data is noise free

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
- Finally we apply this method recursively to the two subsets produced by the rule
  - I.e. instances that are covered/not covered

Iris data example



Extending linear classification

- Linear classifiers can't model nonlinear class boundaries
- Simple trick to allow them to do that:
  - Map attributes into new space consisting of combinations of attribute values
  - E.g.: all products of *n* factors that can be constructed from the attributes
- Example with two attributes and *n* = 3:

$$x = w_1 a_1^{\ 3} + w_2 a_1^2 a_2 + w_3 a_1 a_2^2 + w_3 a_2^3$$

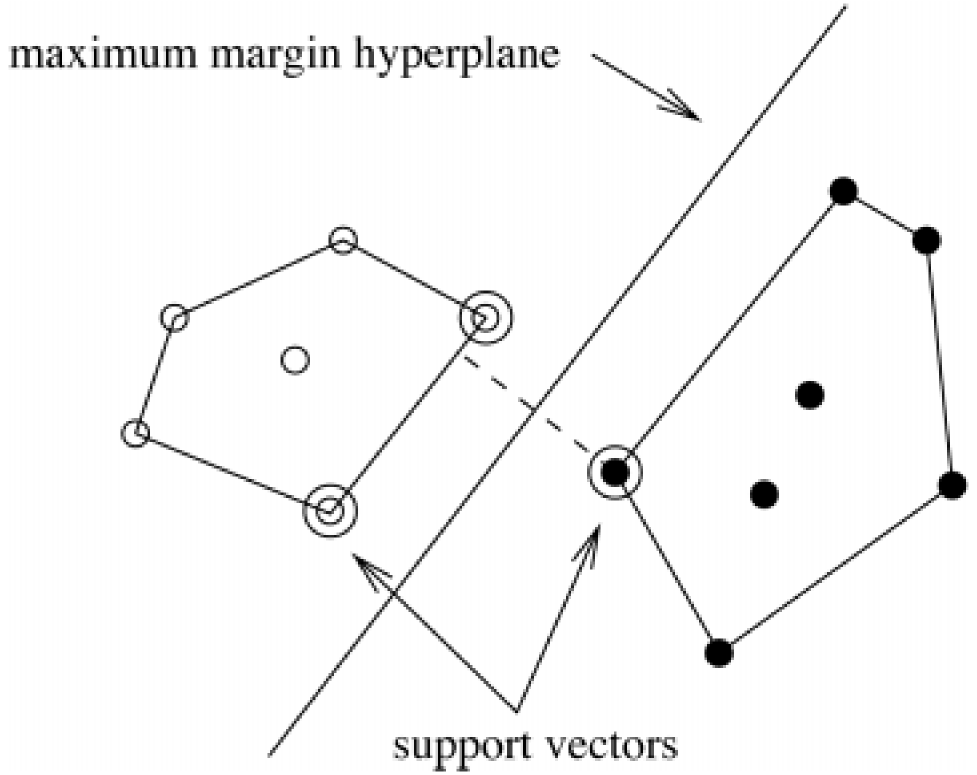
Problems with this approach

- 1st problem: speed
  - With 10 attributes and *n* = 5 we have to determine more than 2000 coefficients
  - Linear regression (with attribute selection) running time is cubic in the number of attributes
- 2nd problem: overfitting
  - Number of coefficients is large relative to the number of training instances
  - *Curse of dimensionality* kicks in

Support vector machines

- *Support vector machines* are algorithms for learning linear classifiers
- They are resilient to overfitting because they learn a particular linear decision boundary:
  - The *maximum margin hyperplane*
- They are fast in the nonlinear case
  - They employ a clever mathematical trick to avoid the creation of “pseudo-attributes”
  - The nonlinear space is created implicitly

The maximum margin hyperplane



Support vectors

- The instances closest to the maximum margin hyperplane are called *support vectors*
- Important observation: the support vectors define the maximum margin hyperplane!
  - All other instances can be deleted without changing the position and orientation of the hyperplane!
- This means the hyperplane $x = w_0 + w_1 a_1 + w_2 a_2$

  can be written as $$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$$

Finding support vectors

- Support vector: training instance for which $\alpha_i > 0$
- Determining all α_i and b is a constrained *quadratic optimization problem*
  - There are off-the-shelf tools for solving these problems
  - However, special-purpose algorithms are faster
    - Example: Platt's *sequential minimal optimization* algorithm (implemented in WEKA)
- Note: all this assumes separable data!

Nonlinear SVMs

- Same trick can be applied here: “pseudo attributes” representing attribute combinations
- Overfitting not (such) a (big) problem because the maximum margin hyperplane is stable
  - There are usually few support vectors relative to the size of the training set
- Computation time still seems to be a problem
  - Every time the dot product is computed we need to go through all the “pseudo attributes”

A mathematical trick

- We can avoid computing the “pseudo attributes”!
- We can compute the dot product before the nonlinear mapping is performed
- Example: instead of computing $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i \mathbf{a}(i) \bullet \mathbf{a}$

  we can compute $x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i (\mathbf{a}(i) \bullet \mathbf{a})^n$
- This corresponds to a map into the instance space spanned by all products of *n* attributes

Other kernel functions

- The mapping is performed by the kernel function
- We can use *kernel functions* other than the *polynomial kernel* from above

$$x = b + \sum_{i \text{ is supp. vector}} \alpha_i y_i K(\mathbf{a}(i) \bullet \mathbf{a})$$

- Only requirement: $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \bullet \phi(\mathbf{x}_j)$
- Examples: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \bullet \mathbf{x}_j + 1)^d$

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-(\mathbf{x}_i - \mathbf{x}_j)^2}{2\sigma^2}} \qquad K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta \mathbf{x}_i \bullet \mathbf{x}_j + b)$$

Noise

- So far we have assumed that the data is separable (in original or transformed space)
- SVMs can be applied to noisy data by introducing a “noise” parameter *C*
- *C* bounds the influence of any one training instance on the decision boundary
  - Corresponding constraint: $0 \le \alpha_i \le C$
- Still a quadratic optimization problem
- *C* has to be found by experimentation

Sparse data

- SVM algorithms can be sped up dramatically if the data is *sparse* (i.e. many values are 0)
- Why? Because they compute lots and lots of dot products
- With sparse data dot products can be computed very efficiently
  - We just need to iterate over the values that are non-zero
- SVMs can process sparse datasets with tens of thousands of attributes

Applications

- Machine vision: e.g face identification
  - Outperforms alternative approaches (1.5% error)
- Handwritten digit recognition: USPS data
  - Comparable to best alternative (0.8% error)
- Bioinformatics: e.g. prediction of protein secondary structure
- Text classifiation
- Algorithm can be modified to deal with numeric prediction problems

Instance-based learning

- Practical problems of 1-NN scheme:
  - Slow (but: fast tree-based approaches exist)
    - Remedy: removing irrelevant data
  - Noise (but: *k*-NN copes quite well with noise)
    - Remedy: removing noisy instances
  - All attributes deemed equally important
    - Remedy: attribute weighting (or simply selection)
  - Doesn’t perform explicit generalization
    - Remedy: rule-based NN approach

Edited NN

- Edited NN classifiers discard some of the training instances before making predictions
- Saves memory and speeds-up classification
- IB2: incremental NN learner that only incorporates misclassified instances into the classifier
  - Problem: noisy data gets incorporated
- Other approach: Voronoi-diagram-based
  - Problem: computationally expensive
  - Approximations exist

Dealing with noise

- Excellent way: cross-validation-based *k*-NN classifier (but slow)
- Different approach: discarding instances that don't perform well by keeping success records (IB3)
  - Computes confidence interval for instance's success rate and for default accuracy of its class
  - If lower limit of first interval is above upper limit of second one, instance is *accepted* (IB3: 5%-level)
  - If upper limit of first interval is below lower limit of second one, instance is *rejected* (IB3: 12.5%-level)

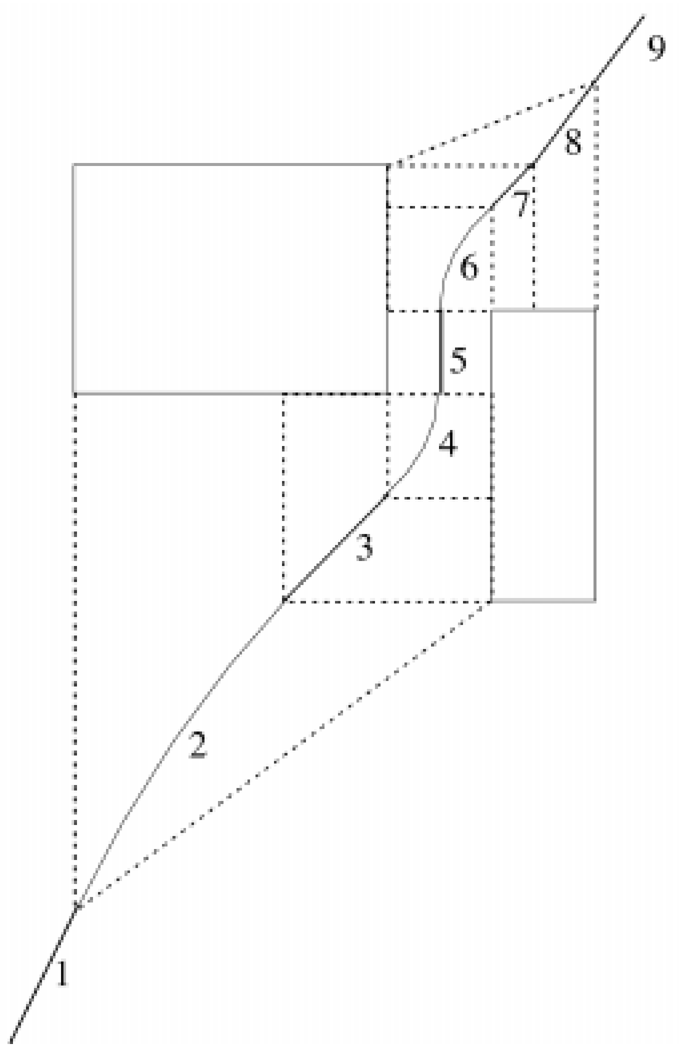
Weighting attributes

- Problem: irrelevant attributes
- Simple solution: attribute selection
- More sophisticated: attribute weighting
  - Class-specific weights may be used (can result in unclassified instances and multiple classifications)
- Euclidean d. w. weights: $\sqrt{w_1{}^2(x_1 - y_1)^2 + \ldots + w_n{}^2(x_n - y_n)^2}$
- Updating of weights based on nearest neighbor
  - Class correct/incorrect: weight increased/decreased
  - $|x_i$-$y_i|$ small/large: amount large/small

Generalized exemplars

- Instances can be generalized into *hyperrectangles*
  - Online version incrementally modifies rectangles
  - Offline version tries to find small set of rectangles covering given set of instances
- Important design decisions:
  - Overlapping rectangles allowed?
    - Conflict resolution required
  - Nested rectangles allowed?
  - Distance for instances that are not covered?

An example



Generalized distance functions

- K*: distance is measured as probability of transforming instance A into B by chance
  - Has to average over all transformation paths (by weighting paths according their probability)
  - Requirement: set of elementary transformation operations (and way of measuring the probabilities)
- Uniform way of dealing with different types of attributes
- Can easily be generalized to compute the distance between sets of instances

Numeric prediction

- Counterparts exist for all schemes that we previously discussed
    - Decision trees, rule learners, SVMs, etc.
- All classification schemes can be applied to regression problems using discretization
    - Prediction: weighted average of intervals’ midpoints (weighted according to class probabilities)
- Regression more difficult than classification (i.e. percent correct vs. mean squared error)

Regression trees

- Differences to decision trees:
  - Splitting criterion: minimizing intra-subset variation
  - Pruning criterion: based on numeric error measure
  - Leaf node predicts average class values of training instances reaching that node
- Can approximate piecewise constant functions
- Easy to interpret
- More sophisticated version: *model trees*

Model trees

- Regression trees with linear regression functions at each node
- Linear regression applied to instances that reach a node after full regression tree has been built
- Only a subset of the attributes is used for LR
  - Attributes occurring in subtree (+maybe attributes occurring in path to the root)
- Fast: overhead for LR not large because usually only a small subset of attributes is used in tree

Smoothing

- Naïve method for prediction outputs value of LR for corresponding leaf node
- Performance can be improved by *smoothing* predictions using internal LR models
  - Predicted value is weighted average of LR models along path from root to leaf
- Smoothing formula: $p' = \frac{np + kq}{n + k}$
- Same effect can be achieved by incorporating the internal models into the leaf nodes

Building the tree

- Splitting criterion: *standard deviation reduction*

$$SDR = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

- Termination criteria (important when building trees for numeric prediction):
  - Standard deviation becomes smaller than certain fraction of sd for full training set (e.g. 5%)
  - Too few instances remain (e.g. less than four)

Pruning

- Pruning is based on estimated absolute error of LR models
- Heuristic estimate: $\frac{n+\nu}{n-\nu} \times \text{average\_absolute\_error}$
- LR models are pruned by greedily removing terms to minimize the estimated error
- Model trees allow for heavy pruning: often a single LR model can replace a whole subtree
- Pruning proceeds bottom up: error for LR model at internal node is compared to error for subtree

Nominal attributes

- Nominal attributes are converted into binary attributes (that can be treated as numeric ones)
  - Nominal values are sorted using average class val.
  - If there are *k* values, *k*-1 binary attributes are generated
    - The *i*th binary attribute is 0 if an instance's value is one of the first *i* in the ordering, 1 otherwise
- It can be proven that the best split on one of the new attributes is the best binary split on original
- But M5‘ only does the conversion once

Missing values

- Modified splitting criterion: $SDR = \frac{m}{|T|} \times \left[sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i) \right]$

- Procedure for deciding into which subset the instance goes: s*urrogate splitting*
  - Choose attribute for splitting that is most highly correlated with original attribute
  - Problem: complex and time-consuming
  - Simple solution: always use the class
- Testing: replace missing value with average

Surrogate splitting based on class

- Instances with known values are used to compute split point
- Given the split point, instances can be divided into two subsets *L* and *R*
- Assume *L* has smaller average class value than *R*
- Let *m* be the average of the two averages
- Then, if an instance with a missing value has class value smaller than *m* it goes into *L*, otherwise into *R*
- After full tree has been built, missing values are replaced with average values from corresponding leaf nodes

Pseudo-code for M5’

- Four methods:
  - Main method: *MakeModelTree*()
  - Method for splitting: *split*()
  - Method for pruning: *prune*()
  - Method that computes error: *subtreeError*()
- We’ll briefly look at each method in turn
- Linear regression method is assumed to perform attribute subset selection based on error

MakeModelTree()

```
MakeModelTree (instances)
{
  SD = sd(instances)
  for each k-valued nominal attribute
    convert into k-1 synthetic binary attributes
  root = newNode
  root.instances = instances
  split(root)
  prune(root)
  printTree(root)
}
```

split()

```
split(node)
{
  if sizeof(node.instances) < 4 or
    sd(node.instances) < 0.05*SD
    node.type = LEAF
  else
    node.type = INTERIOR
    for each attribute
      for all possible split positions of the attribute
        calculate the attribute’s SDR
    node.attribute = attribute with maximum SDR
    split(node.left)
    split(node.right)
}
```

prune()

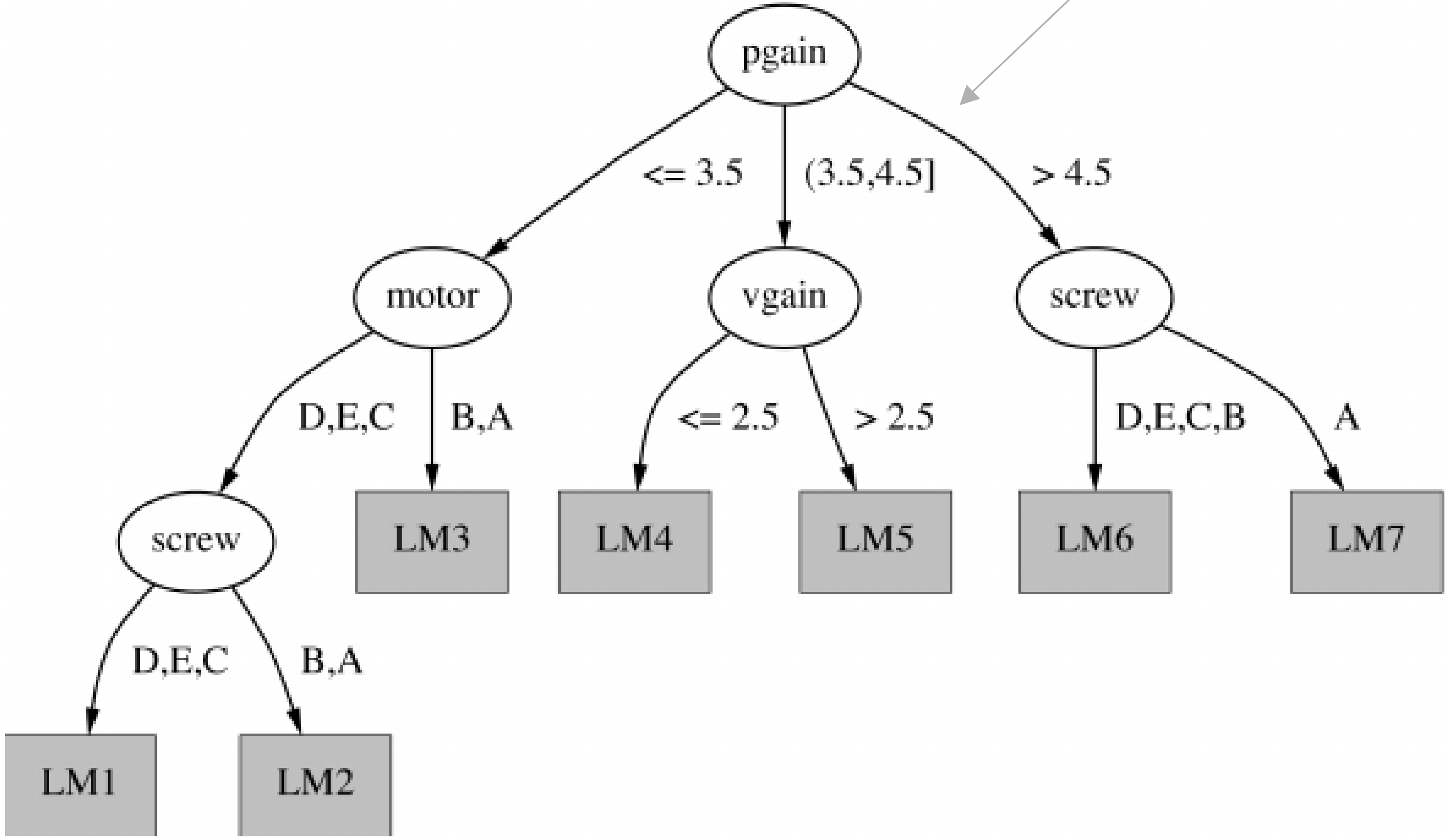
```
prune(node)
{
  if node = INTERIOR then
    prune(node.leftChild)
    prune(node.rightChild)
    node.model = linearRegression(node)
    if subtreeError(node) > error(node) then
      node.type = LEAF
}
```

subtreeError()

```
subtreeError(node)
{
  l = node.left; r = node.right
  if node = INTERIOR then
    return (sizeof(l.instances)*subtreeError(l)
            + sizeof(r.instances)*subtreeError(r))
                /sizeof(node.instances)
  else return error(node)
}
```

Model tree for servo data

Result of merging



Locally weighted regression

- Numeric prediction using instance-based learning combined with linear regression
- Lazy learning scheme: linear regression function is computed at prediction time
- Training instances are weighted according to distance to test instance
- Requires a weighted version of linear regression
- Advantage: nonlinear approximation, incremental
- Disadvantage: slow

Design decisions

- Type of weighting function:
  - Inverse of Euclidean distance
  - Gaussian kernel applied to Euclidean distance
  - Triangular kernel used the same way, etc.
- More important: *smoothing parameter* used to scale the distance function
  - Distance is multiplied by inverse of this parameter
  - Possible choice: distance of *k*th nearest training instance (makes it data dependent)

Discussion

- Regression trees were introduced in CART
- Quinlan proposed the M5 model tree inducer
- M5’: slightly improved version that’s publicly available
- Quinlan also investigated combining instance-based learning with M5
- CUBIST: Quinlan’s commercial rule learner for numeric prediction
- Interesting comparison: Neural nets vs. M5

Clustering

- *Unsupervised*: no target value to be predicted
- Differences between models/algorithms:
  - Exclusive vs. overlapping
  - Deterministic vs. probabilistic
  - Hierarchical vs. flat
  - Incremental vs. batch learning
- Evaluation problematic: usually done by inspection
- But: if clustering is treated as a density estimation problem, then it can be evaluated on test data!

Hierarchical clustering

- Bottom up: at each step join the two closest clusters (starting with single-instance clusters)
  - Design decision: distance between clusters
    - E.g. two closest instances in clusters vs. distance between means
- Top down: find two clusters and then proceed recursively for the two subsets
  - Can be very fast
- Both methods produce a dendrogram

The *k*-means algorithm

- Clusters the data into *k* groups where *k* is predefined
- 1st step: cluster centers are chosen (e.g. at random)
- 2nd step: instances are assigned to clusters based on their distance to the cluster centers
- 3rd step: *centroids* of clusters are computed
- 4th step: go to 1st step until convergence

Discussion

- Result can vary significantly based on initial choice of seeds
- Algorithm can get trapped in a local minimum
  - Example: four instances at the vertices of a two-dimensional rectangle
    - Local minimum: two cluster centers at the midpoints of the rectangle's long sides
- Simple way to increase chance of finding a global optimum: restart with different random seeds

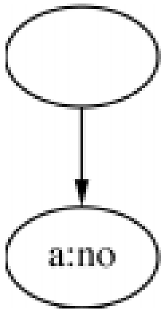
Incremental clustering

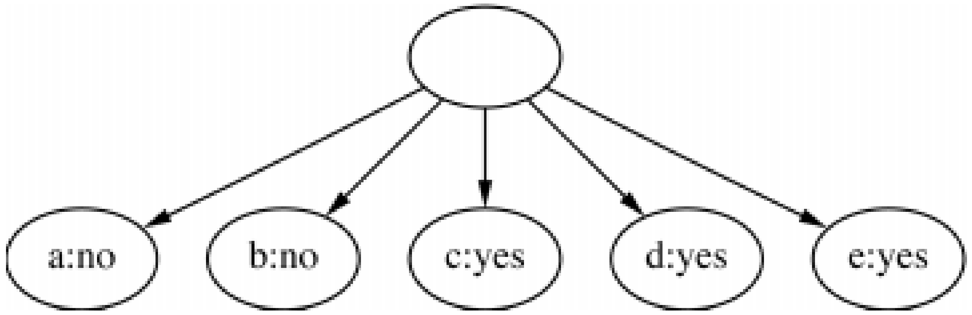
- COBWEB/CLASSIT: incrementally forms a hierarchy of clusters
- In the beginning tree consists of empty root node
- Instances are added one by one, and the tree is updated appropriately at each stage
- Updating involves finding the right leaf for an instance (possibly restructuring the tree)
- Updating decisions are based on *category utility*

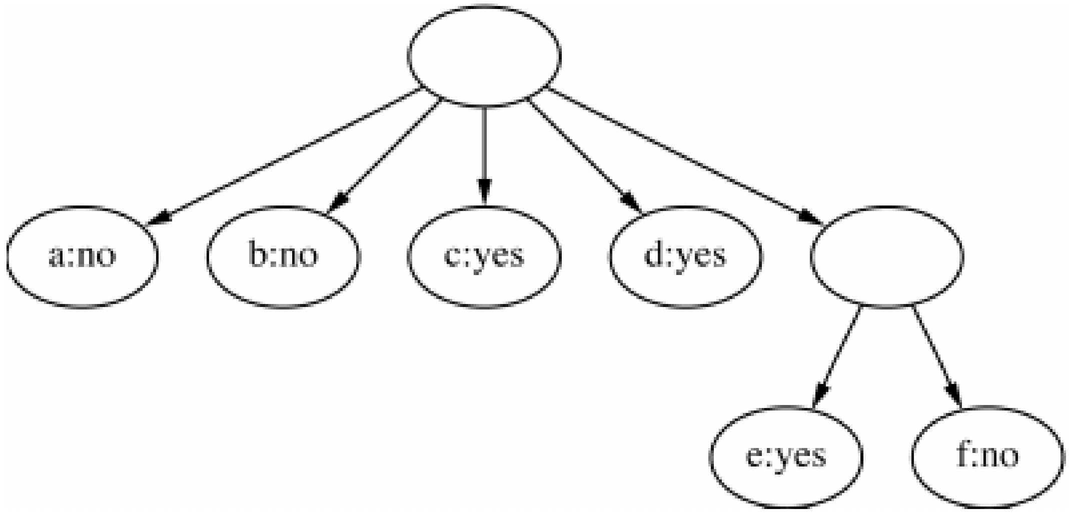
Clustering the weather data

ID code	Outlook	Temp.	Humidity	Windy
A	Sunny	Hot	High	False
B	Sunny	Hot	High	True
C	Overcast	Hot	High	False
D	Rainy	Mild	High	False
E	Rainy	Cool	Normal	False
F	Rainy	Cool	Normal	True
G	Overcast	Cool	Normal	True
H	Sunny	Mild	High	False
I	Sunny	Cool	Normal	False
J	Rainy	Mild	Normal	False
K	Sunny	Mild	Normal	True
L	Overcast	Mild	High	True
M	Overcast	Hot	Normal	False
N	Rainy	Mild	High	True

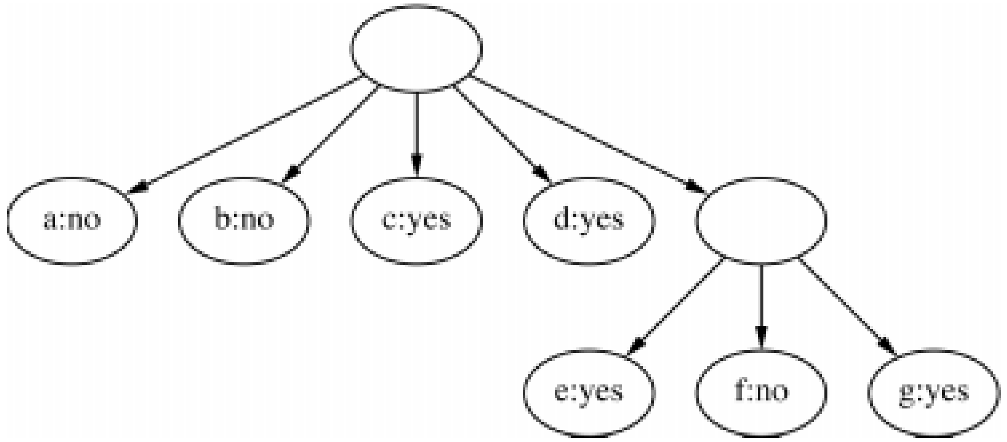
Steps 1-3



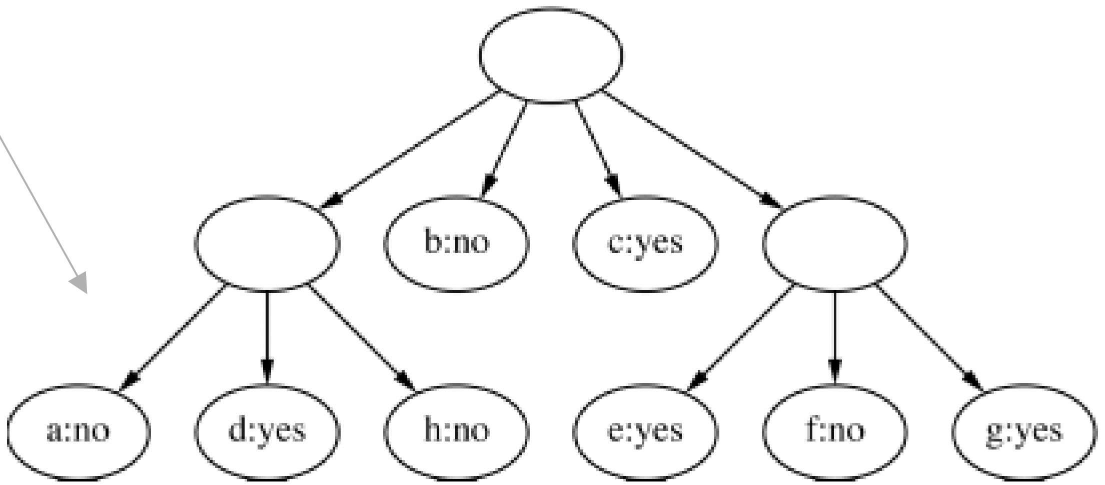




Steps 3-4

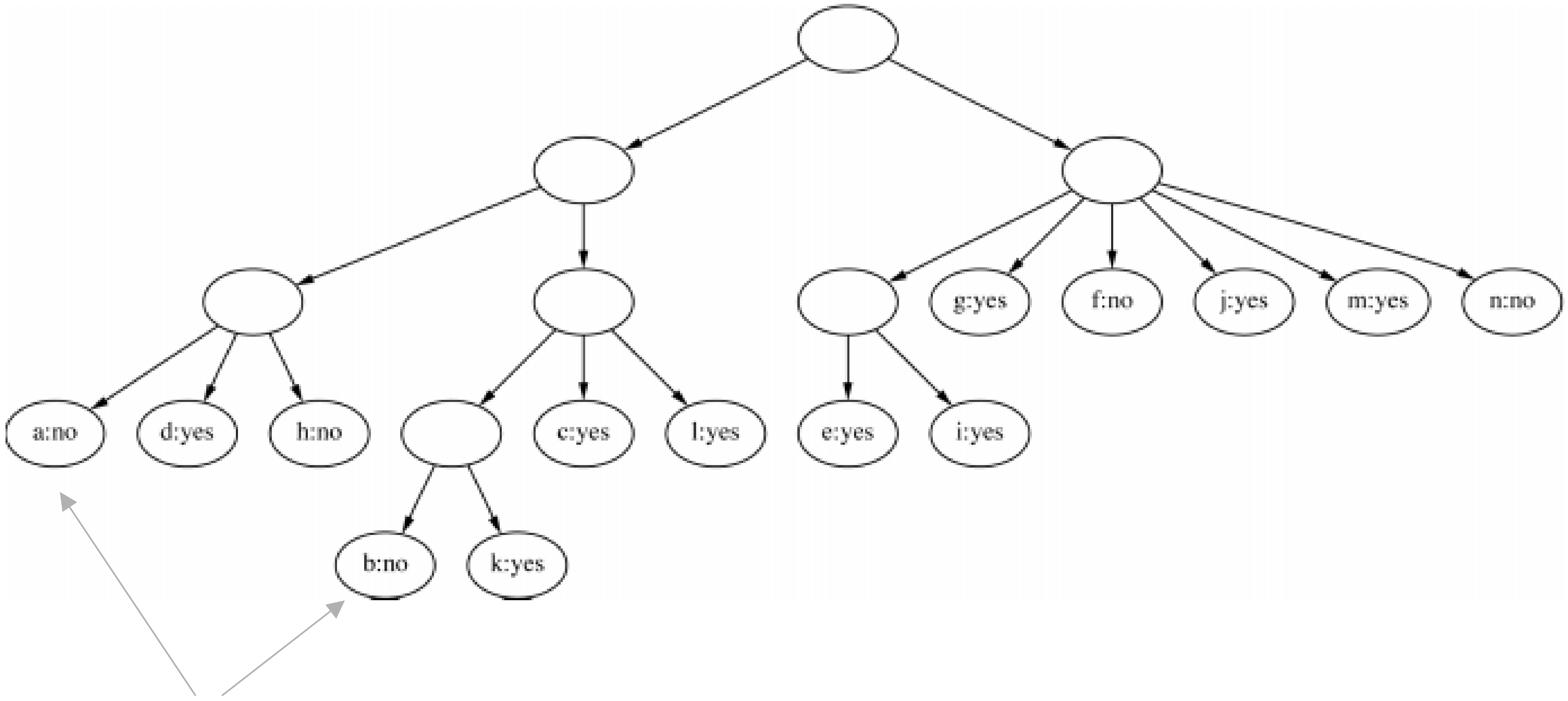


Best host and runner-up have been *merged*



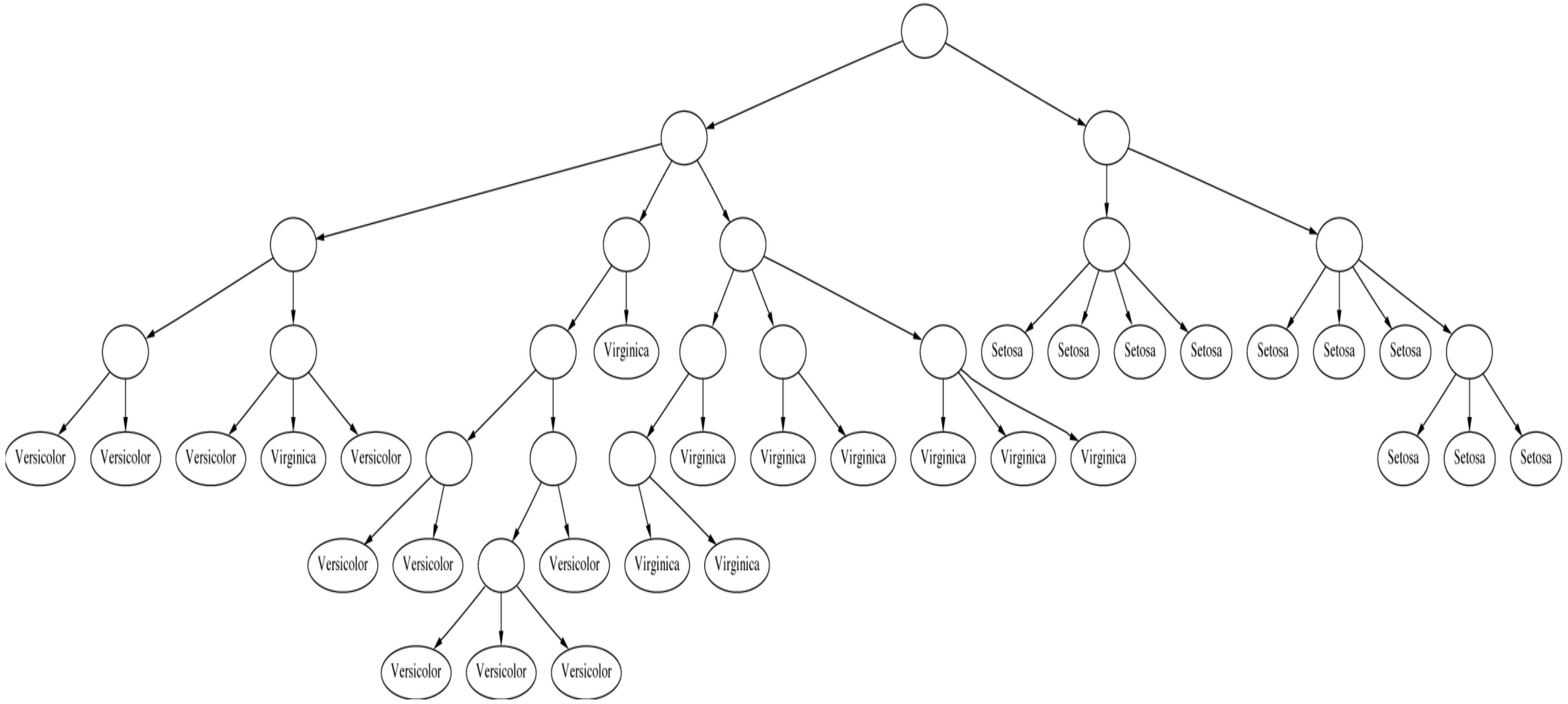
Note: *splitting* the best host is considered if merging doesn’t help

The final hierarchy

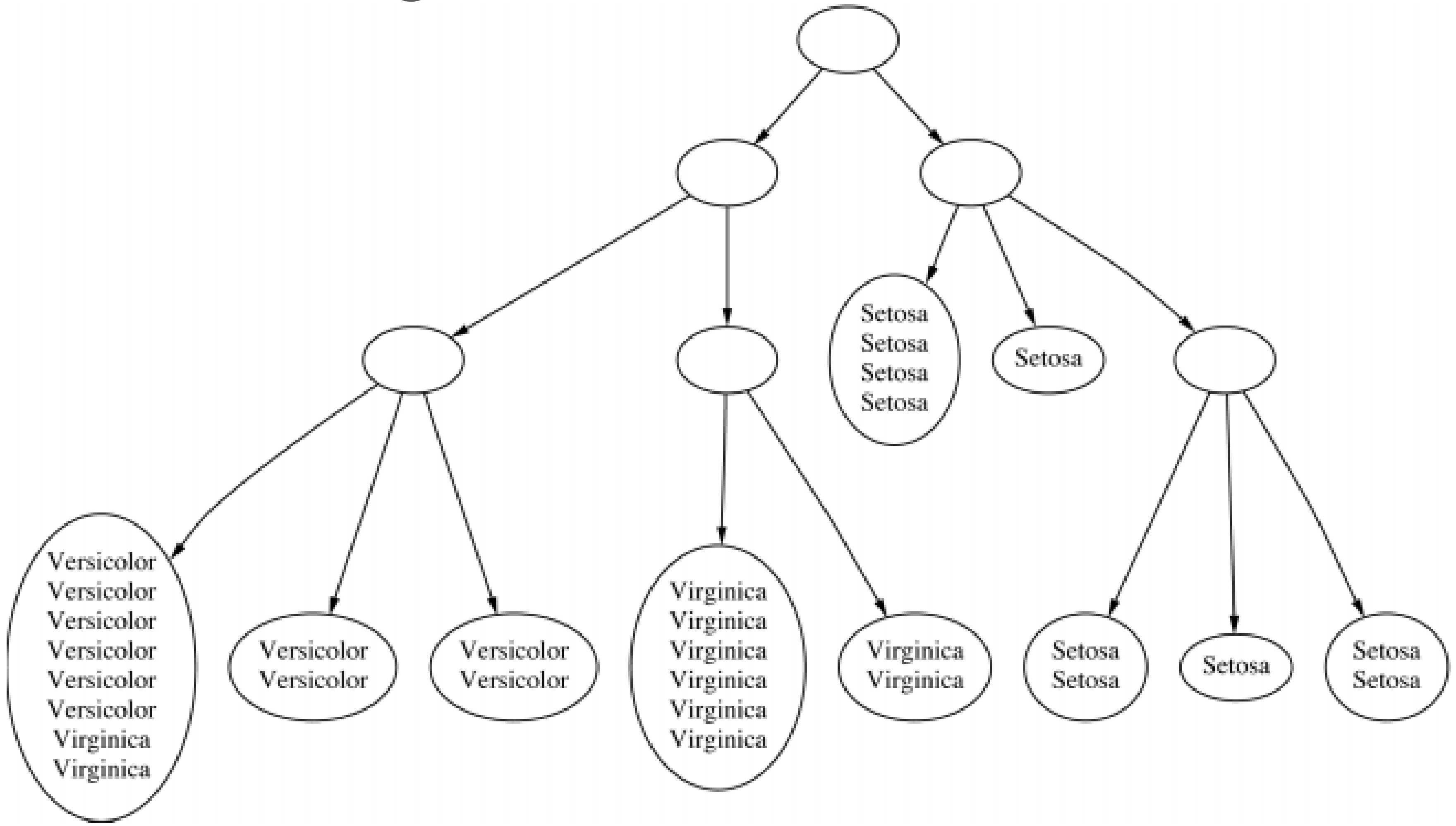


a and b are actually very similar

Clustering (parts) of the iris data



Clustering the iris data with cutoff



Category utility

- Category utility is a kind of quadratic loss function defined on conditional probabilities:

$$CU(C_1, C_2, ..., C_k) = \frac{\sum_l \Pr[C_l] \sum_i \sum_j (\Pr[a_i = v_{ij} \mid C_l]^2 - \Pr[a_i = v_{ij}]^2)}{k}$$

- If every instance gets put into a different category the numerator becomes (m = #attributes):

$$m - \Pr[a_i = v_{ij}]^2 \longleftarrow \textit{maximum}$$

Numeric attributes

- We assume normal distribution: $$f(a) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(a-\mu)^2}{2\sigma^2}}$$
- Then we get: $$\sum_j \Pr[a_i = v_{ij}]^2 \Leftrightarrow \int f(a_i)^2 da_i = \frac{1}{2\sqrt{\pi}\sigma_i}$$
- Thus $$CU = \frac{\sum_l \Pr[C_l] \sum_i \sum_j (\Pr[a_i = v_{ij} \mid C_l]^2 - \Pr[a_i = v_{ij}]^2)}{k}$$

  is $$CU = \frac{\sum_l \Pr[C_l] \frac{1}{2\sqrt{\pi}} \sum_i \left(\frac{1}{\sigma_{il}} - \frac{1}{\sigma_i} \right)}{k}$$
- *Acuity* parameter: prespecified minimum variance

Probability-based clustering

- Problems with above heuristic approach:
  - Division by *k?*
  - Order of examples?
  - Are restructuring operations sufficient?
  - Is result at least *local* minimum of category utility?
- From a probabilistic perspective, we want to find the most likely clusters given the data
- Also: instance only has certain probability of belonging to a particular cluster

Finite mixtures

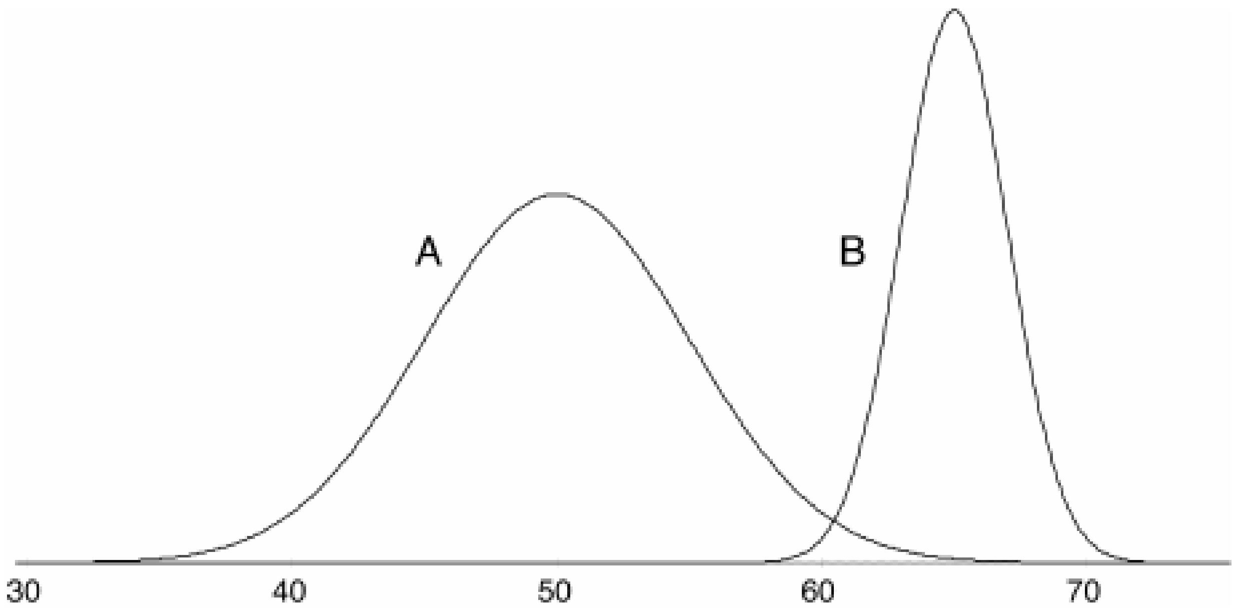
- Probabilistic clustering algorithms model the data using a *mixture* of distributions
- Each cluster is represented by one distribution
  - The distribution governs the probabilities of attributes values in the corresponding cluster
- They are called *finite mixtures* because there is only a finite number of clusters being represented
- Usually individual distributions are normal distribut.
- Distributions are combined using cluster weights

A two-class mixture model

data

A	51	B	62	B	64	A	48	A	39	A	51
A	43	A	47	A	51	B	64	B	62	A	48
B	62	A	52	A	52	A	51	B	64	B	64
B	64	B	64	B	62	B	63	A	52	A	42
A	45	A	51	A	49	A	43	B	63	A	48
A	42	B	65	A	48	B	65	B	64	A	41
A	46	A	48	B	62	B	66	A	48		
A	45	A	49	A	43	B	65	B	64		
A	45	A	46	A	40	A	46	A	48		

model



$\mu_A=50$, $\sigma_A=5$, $p_A=0.6$ $\mu_B=65$, $\sigma_B=2$, $p_B=0.4$

Using the mixture model

- The probability of an instance *x* belonging to cluster *A* is:

$$\Pr[A \mid x] = \frac{\Pr[x \mid A]\Pr[A]}{\Pr[x]} = \frac{f(x;\mu_A,\sigma_A)\,p_A}{\Pr[x]}$$

with

$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

- The *likelihood* of an instance given the clusters is:

$$\Pr[x \mid \text{the distributions}] = \sum_i \Pr[x \mid \text{cluster}_\text{i}]\Pr[\text{cluster}_\text{i}]$$

Learning the clusters

- Assume we know that there are *k* clusters
- To learn the clusters we need to determine their parameters
  - I.e. their means and standard deviations
- We actually have a performance criterion: the likelihood of the training data given the clusters
- Fortunately, there exists an algorithm that finds a local maximum of the likelihood

The EM algorithm

- *EM algorithm*: expectation-maximization algorithm
  - Generalization of *k*-means to probabilistic setting
- Similar iterative procedure:
  1. Calculate cluster probability for each instance (expectation step)
  2. Estimate distribution parameters based on the cluster probabilities (maximization step)
- Cluster probabilities are stored as instance weights

More on EM

- Estimating parameters from weighted instances:

$$\mu_A = \frac{w_1 x_1 + w_2 x_2 + \ldots + w_n x_n}{w_1 + w_2 + \ldots + w_n}$$

$$\sigma_A{}^2 = \frac{w_1 (x_1 - \mu)^2 + w_2 (x_2 - \mu)^2 + \ldots + w_n (x_n - \mu)^2}{w_1 + w_2 + \ldots + w_n}$$

- Procedure stops when log-likelihood saturates
- Log-likelihood:

$$\sum_i \log(p_A \Pr[x_i \mid A] + p_B \Pr[x_i \mid B])$$

Extending the mixture model

- Using more then two distributions: easy
- Several attributes: easy if independence is assumed
- Correlated attributes: difficult
  - Modeled jointly using a bivariate normal distribution with a (symmetric) covariance matrix
  - With n attributes this requires estimating $n+n(n+1)/2$ parameters
- Nominal attributes: easy if independent

More on extensions

- Correlated nominal attributes: difficult
  - Two correlated attributes result in $v_1 \, v_2$ parameters
- Missing values: easy
- Distributions other than the normal distribution can be used:
  - “log-normal” if predetermined minimum is given
  - “log-odds” if bounded from above and below
  - Poisson for attributes that are integer counts
- Cross-validation can be used to estimate *k*!!

Bayesian clustering

- Problem: overfitting possible if number of parameters gets large
- *Bayesian approach*: every parameter has a prior probability distribution
    - Gets incorporated into the overall likelihood figure and thereby penalizes introduction of parameters
- Example: Laplace estimator for nominal attributes
- Can also have prior on number of clusters!
- Actual implementation: NASA’s AUTOCLASS

Discussion

- Clusters can be interpreted by using supervised learning in a post-processing step
- Can be used to fill in missing values
- May be advantageous to make attributes more independent in pre-processing step
  - I.e. using *principal component analysis*
- Big advantage of probabilistic clustering schemes:
  - Likelihood of data can be estimated and used to compare different clustering models