Pre-Processing

- Databases are typically not made to support analysis with a data mining algorithm
  - pre-processing of data is necessary
- Pre-processing techniques:
  - **Data Cleaning:** remove inconsistencies from the data
  - **Feature Engineering:** find the right features/attribute set
    - *Feature Subset Selection*: select appropriate feature subsets
    - *Feature Transformation*: bring attributes into a suitable form (e.g., discretization)
    - *Feature Construction*: construct derived features
  - **Sampling:**
    - select appropriate subsets of the data

Unsupervised vs. Supervised Pre-processing

- Unsupervised
  - do not use information about the learning task
    - only prior information (from knowledge about the data)
    - and information about the distribution of the training data
- Supervised
  - use information about the learning task
    - e.g.: look at relation of an attribute to class attribute
- WARNING:
    - pre-processing may only use information from training data!
      - compute pre-processing model from training data
      - apply the model to training and test data
      - otherwise information from test data may be captured in the pre-processing step → biased evaluation
    - in particular: apply pre-processing to every fold in cross-validation

Feature Subset Selection

- Databases are typically not collected with data mining in mind
- Many features may be
  - irrelevant
  - uninteresting
  - redundant
- Removing them can
  - increase efficiency
  - improve accuracy
  - prevent overfitting
- Feature Subsect Selection techniques try to determine appropriate features automatically

Unsupervised FSS

- Using domain knowledge
  - some features may be known to be irrelevant, uninteresting or redundant
- Random Sampling
  - select a random sample of the feature
  - may be appropriate in the case of many weakly relevant features and/or in connection with ensemble methods

Supervised FSS

- Filter approaches:
  - compute some measure for estimating the ability to discriminate between classes
  - typically measure feature weight and select the best n features
  - problems
    - redundant features (correlated features will all have similar weights)
    - dependent features (some features may only be important in combination (e.g., XOR/parity problems).
- Wrapper approaches
  - search through the space of all possible feature subsets
  - each search subset is tried with the learning algorithm

Supervised FSS: Filters

- foreach attribute A
  - $W[A]$ = feature weight according to some measure of discrimination
    - e.g., decision tree splitting criteria (entropy/information gain, gini-index, ...)
- select the n features with highest $W[A]$

Basic idea:
- a good attribute should discriminate between the different classes
- use a measure of discrimination to determine which attributes to select

Disadvantage:
- quality of each attribute is measured in isolation
- some attributes may only be useful in combination with others

RELIEF
(Kira & Rendell, ICML-92)

Basic idea:

- in a local neighborhood around an example R a good attribute A should
  - allow to discriminate R from all examples of different classes (the set of *misses*)
    - therefore the probability that the attribute has the same value for R and a miss M should be low
  - have the same value for all examples of the same class as R (the set of *hits*)
    - therefore the probability that the attribute has the same value for R and a hit H should be high

$\rightarrow$ try to estimate and maximize $W[A] = P(a_R \neq a_M) - P(a_R \neq a_H)$

where a_X is the value of attribute A in example X

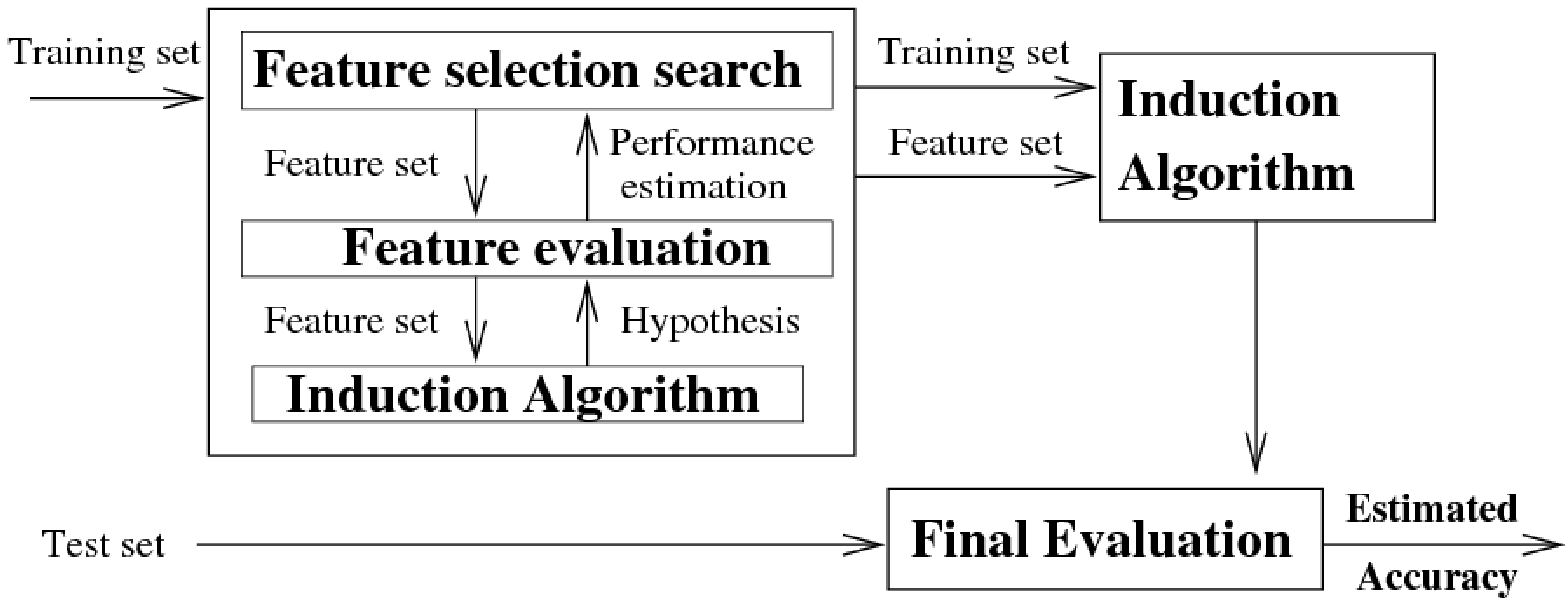
RELIEF
(Kira & Rendell, ICML-92)

- set all attribute weights $W[A] = 0.0$
- for $i = 1$ to m (← user-settable parameter)
  - select a random example R
  - find
    - H: nearest neighbor of same class (*near hit*)
    - M: nearest neigbor of different class (*near miss*)
  - for each attribute A
    - $W[A] \leftarrow W[A] - \frac{d(A, H, R)}{m} + \frac{d(A, M, R)}{m}$

    where $d(A,X,Y)$ is the distance in attribute A between examples X and Y (normalized to [0,1]-range).

FSS: Wrapper Approach
(John, Kohavi, Pfleger, ICML-94)

- Wrapper Approach:
  - try a feature subset with the learner
  - improve it by modifying the feature sets based on the result
  - repeat



The induction algorithm itself is used as a "black box" by the subset selection algorithm.

FSS: Wrapper Approach

- Forward selection:

1. start with empty feature set F
2. for each attribute A
   a) $F = F \cup \{A\}$
   b) Estimate Accuracy of Learning algorithm on F
   c) $F = F \setminus \{A\}$
3. $F = F \cup$ {attribute with highest estimated accuracy}
4. goto 2. unless estimated accuracy decreases significantly

- Backward elimination:
  - start with full feature set F
  - try to remove attributes
- Bi-directional search is also possible

Example: Forward Search

Attrs: current set of attributes
Est: accuracy estimated by wrapper
Real: „real“ accuracy

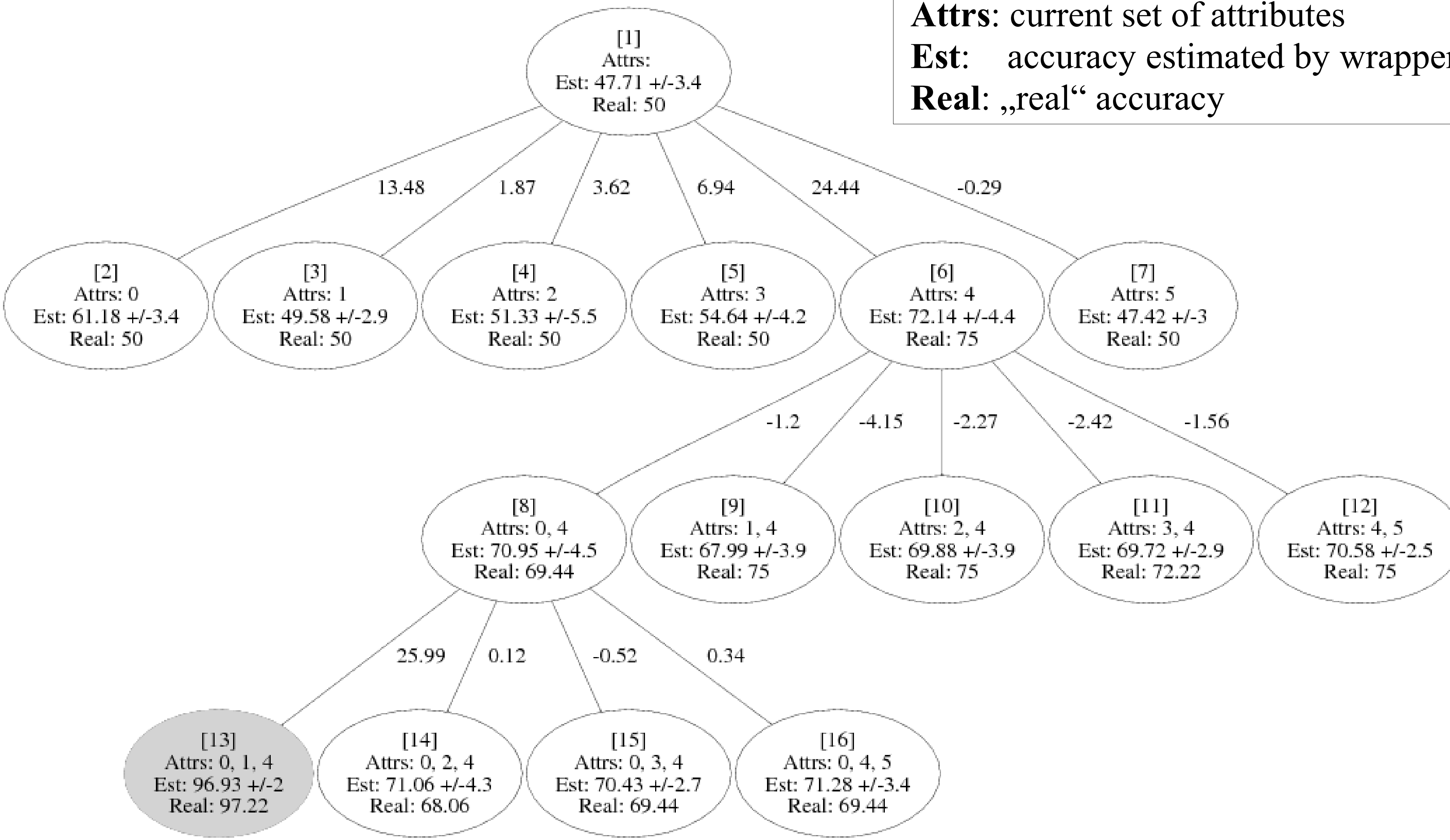
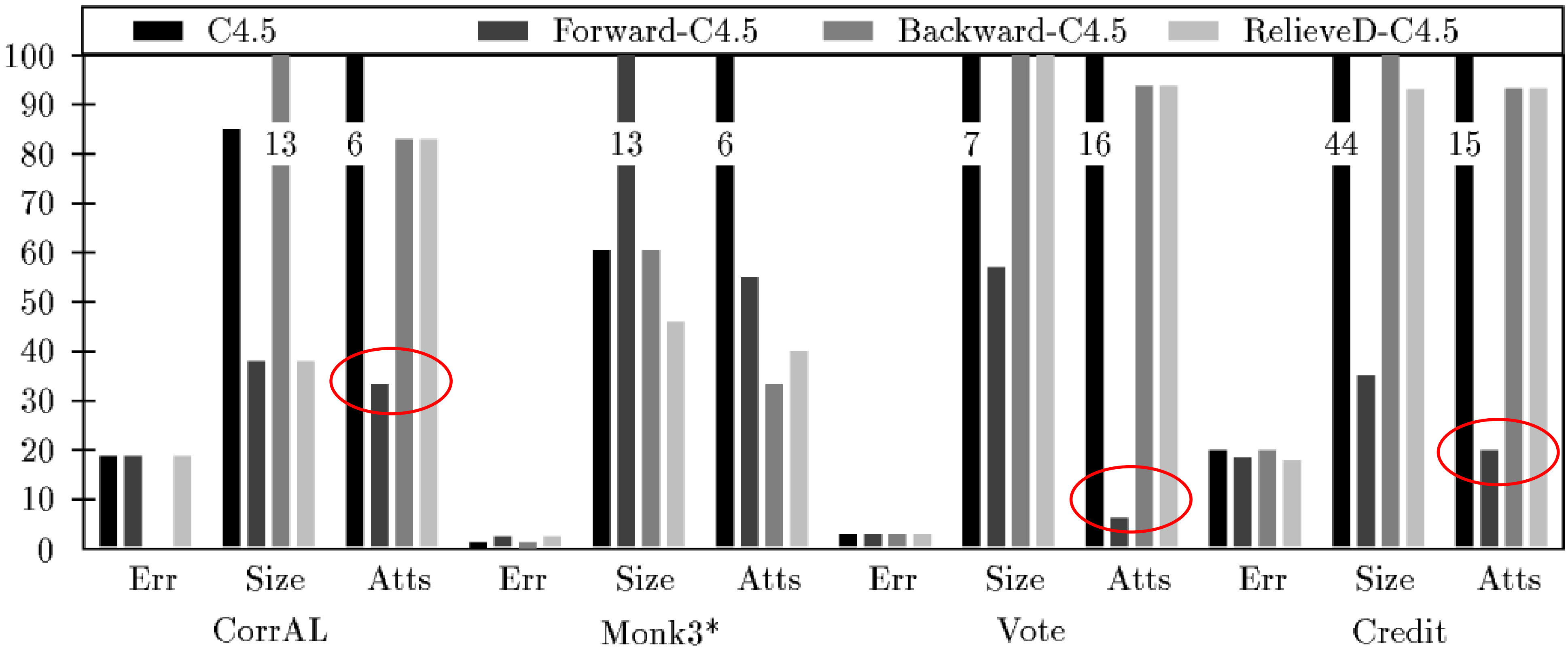


Figure by John, Kohavi & Pfleger

Comparison Wrapper / Relief

Note: RelieveD is a version of Relief that uses all examples instead of a random sample



- on these datasets:
  - forward selection reduces attributes w/o error increase
- in general, it may also reduce error

Figure by John, Kohavi & Pfleger

Properties

- Advantage:
  - find feature set that is tailored to learning algorithm
  - considers combinations of features, not only individual feature weights
  - can eliminate redundant features
    (picks only as many as the algorithm needs)

- Disadvantage:
  - very inefficient: many learning cycles necessary

Feature Transformation

- bring features into a usable form
- numerization
  - some algorithms can only use numeric data
  - nominal → binary
    - a nominal attribute with n values is converted into n binary attributes
  - binary → numeric
    - binary features may be viewed as special cases of numeric attributes with two values
- discretization
  - some algorithms can only use categorical data
    - transform numeric attributes into a number of (ordered) categorical values

Discretization

- Supervised vs. Unsupervised:
  - Unsupervised:
    - only look at the distribution of values of the attribute
  - Supervised:
    - also consider the relation of attribute values to class values
- Merging vs. Splitting:
  - Merging (bottom-up discretization):
    - Start with a set of intervals (e.g., each point is an interval) and successively combine neighboring intervals
  - Splitting (top-down discretization):
    - Start with a single interval and successively split the interval into sub-intervals

Unsupervised Discretization

- domain-dependent:
  - suitable discretizations are often known
  - age (0-18) →
    baby (0-3), child (3-6), school child (6-10), teenager (11-18)
- equal-width:
  - divide value range into a number of intervals with equal width
  - age (0,18) → (0-3, 4-7, 8-11, 12-15, 16-18)
- equal-frequency:
  - divide value range into a number of intervals so that (approximately) the same number of datapoints are in each interval
  - e.g., N = 5: each interval will contain 20% of the training data
  - good for non-uniform distributions (e.g., salary)

Supervised Discretization: Chi-Merge (Kerber, AAAI-92)

Basic Idea: merge neighboring intervals if the class information is independent of the interval an example belongs to

- initialization:
  - sort examples according to feature value
  - construct one interval for each value
- interval merging:
  - compute χ^2 value for each pair of adjacent intervals

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{c} \frac{(A_{ij} - E_{ij})^2}{E_{ij}}$$

  A_{ij} = number of examples in i-th interval that are of class j
  E_{ij} = expected number of examples in i-th interval that are of class j
  = no. of examples in i-th interval * fraction of (all) examples of class j

  - merge those with lowest χ^2 value
- stop
  - when the χ^2 values of all pairs exceed a significance threshold

Supervised Discretization: Entropy-Split (Fayyad & Irani, IJCAI-93)

Basic Idea: grow a decision tree using a single numeric attribute and use the value ranges in the leaves as ordinal values

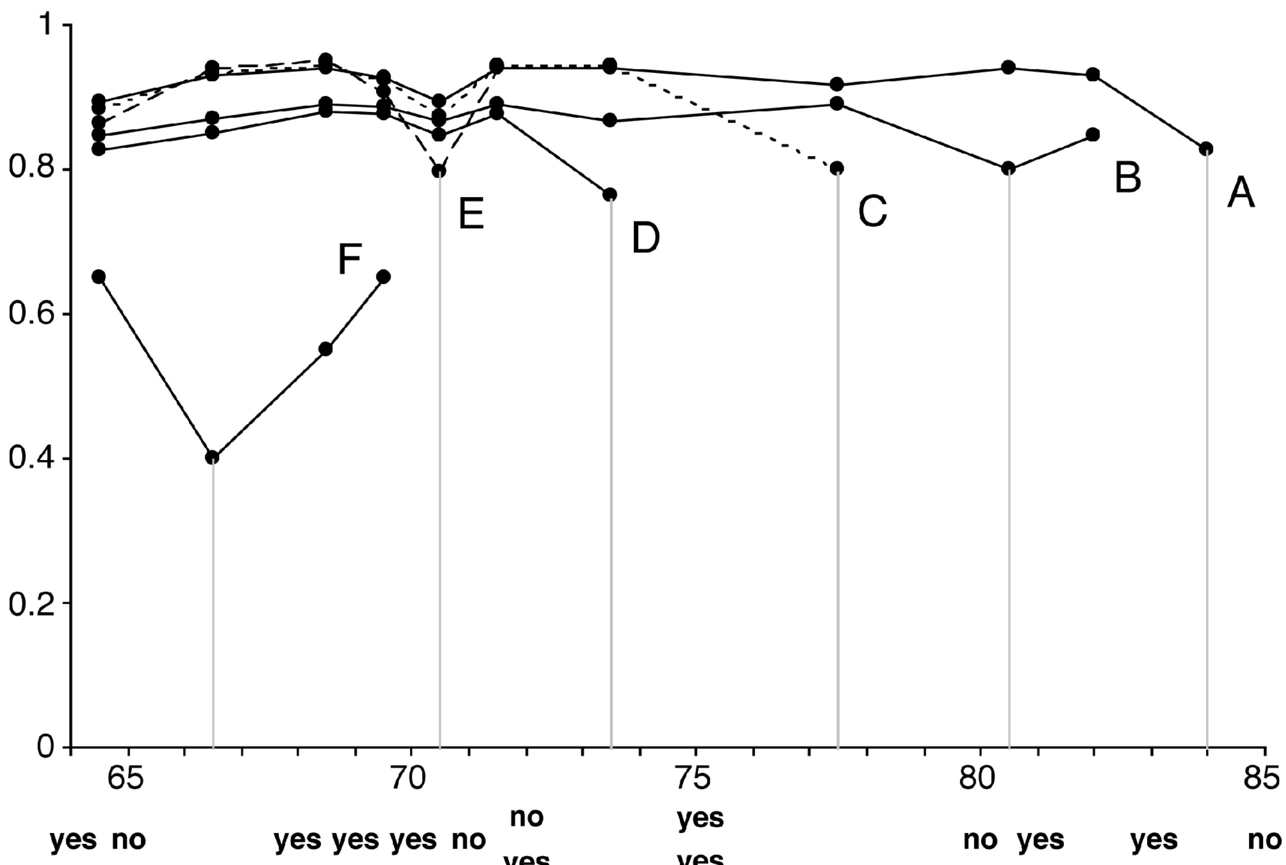
- initialization:
  - initialize intervals with a single interval covering all examples S
  - sort all examples according to the attribute value
  - initialize the set of possible split points
    - simple: all values
    - more efficient: only between class changes in sorted list
- interval splitting:
  - select split point with the minimum weighted entropy

$$T_{max} = \arg\min_{T} \left(\frac{|S_{A<T}|}{|S|} Entropy(S_{A<T}) + \frac{|S_{A\geq T}|}{|S|} Entropy(S_{A\geq T}) \right)$$

  - recursively apply Entropy-Split to $S_{A<T_{max}}$ and $S_{A\geq T_{max}}$
- stop
  - when a given number of splits is achieved
  - or when splitting would yield too small intervals
  - or MDL-based stopping criterion (Fayyad & Irani, 1993)

Example

Temperature	64	65	68	69	70	71	72	72	75	75	80	81	83	85
Play	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	Yes	Yes	No



Unsupervised Feature Construction

- based on domain knowledge
  - Example: Body Mass Index

$$BMI = \frac{weight(kg)}{height(m)^2}$$

- automatic
  - Examples:
    - kernel functions
      - may be viewed as feature construction modules
      - → support vector machines
    - principal components analysis
      - transforms an n-dimensional space into a lower-dimensional subspace w/o losing much information
    - GLEM:
      - uses an Apriori -like algorithms to compute all conjunctive combinations of basic features that occur at least n times
      - application to constructing evaluation functions for game Othello

Supervised Feature Construction

- use the class information to construct features that help to solve the classification problem

- Examples:
  - Wrapper approach
    - use rule or decision tree learning algorithm
    - observe frequently co-occurring features or feature values
    - encode them as separate features
  - Neural Network
    - nodes in hidden layers may be interpreted as constructed features

Scalability

- databases are often too big for machine learning algorithms
  - ML algorithms require frequent counting operations and multi-dimensional access to data
  - only feasible for data that can be held in main memory
- two strategies to make DM algorithms scalable
  - design algorithms that are explicitly targetted towards minimizing the number of database operations (e.g., Apriori)
  - use sampling to work on subsets of the data

Sampling

- Random Sampling
  - Select a random subset of the data
- Progressive Sampling
  - start with a small sample
  - increase sample size
    - arithmetic: increase sample size by fixed number of examples
    - geometric: multiply size with a fixed number (e.g., double size)
  - stop when convergence is detected
- Sequential sampling
  - rule out solution candidates based on significant differences

Windowing

- Idea:
  - focus the learner on the parts of the search space that are not yet correctly covered
- Algorithm:

1. Initialize the window with a random subsample of the available data
2. Learn a theory from the current window
3. If the learned theory correctly classifies all examples (including those outside of the window), return the theory
4. Add some mis-classified examples to the window and goto 2.

- Properties:
  - may learn a good theory from a subset of the data
  - problems with noisy data