DM & ML
Industrial Strength Algorithms

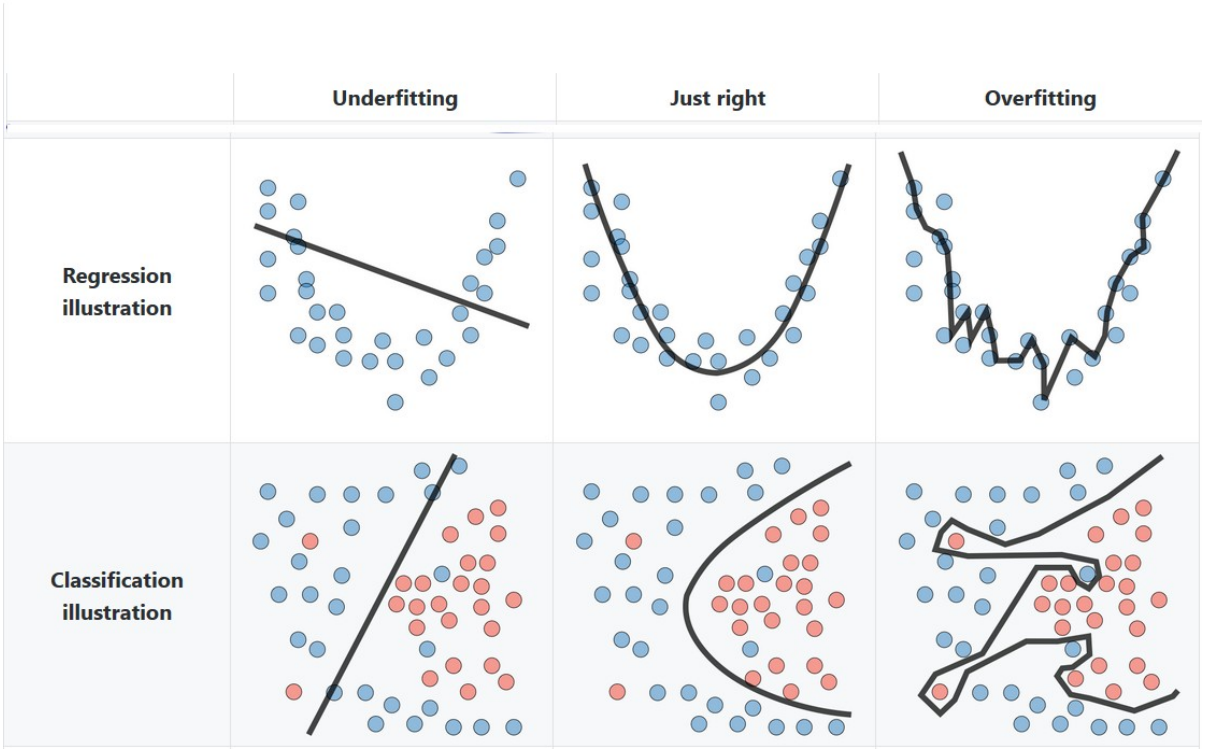
Gergely Lukács
Pázmány Péter Catholic University
Faculty of Information Technology
and Bionics
Budapest, Hungary
lukacs@itk.ppke.hu

Industrial-strength algorithms

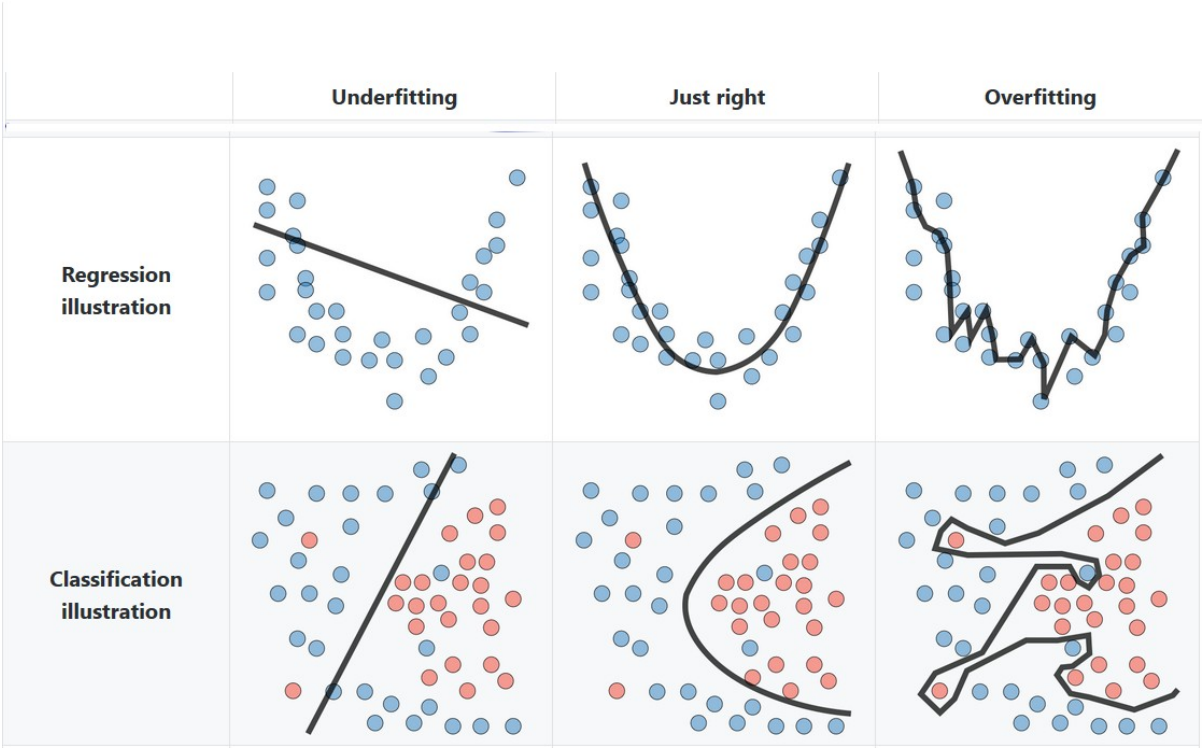
- Requirements for an algorithm to be useful in a wide range of real-world applications
  - Is robust in the presence of ***noise*** (avoids *overfitting*)
  - Can deal with numeric attributes
  - Doesn’t fall over when missing values are present
  - Scalable
- Basic schemes need to be extended to fulfil these requirements

Underfitting, overfitting

- Data: underlying pattern („concept”) + noise
- Goal: describe pattern, ignore noise
  - generalizes well to unseen data
- Two possible problems:



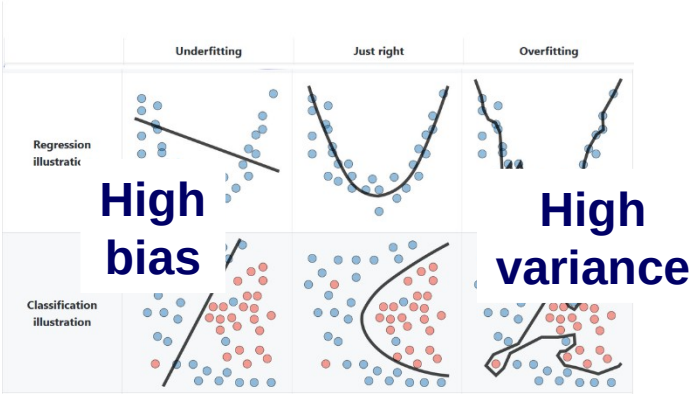
Underfitting, overfitting



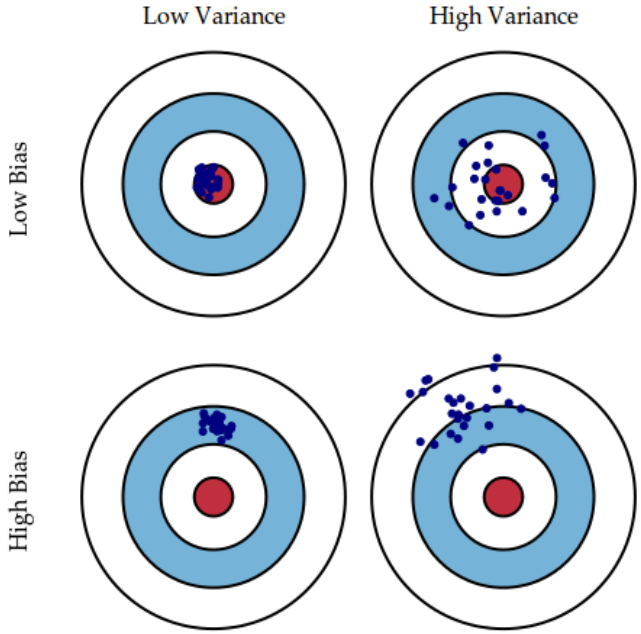
- generalize badly
  - did not learn pattern (well)
  - overly simple model
- represent training set badly
- bad performance on unseen data

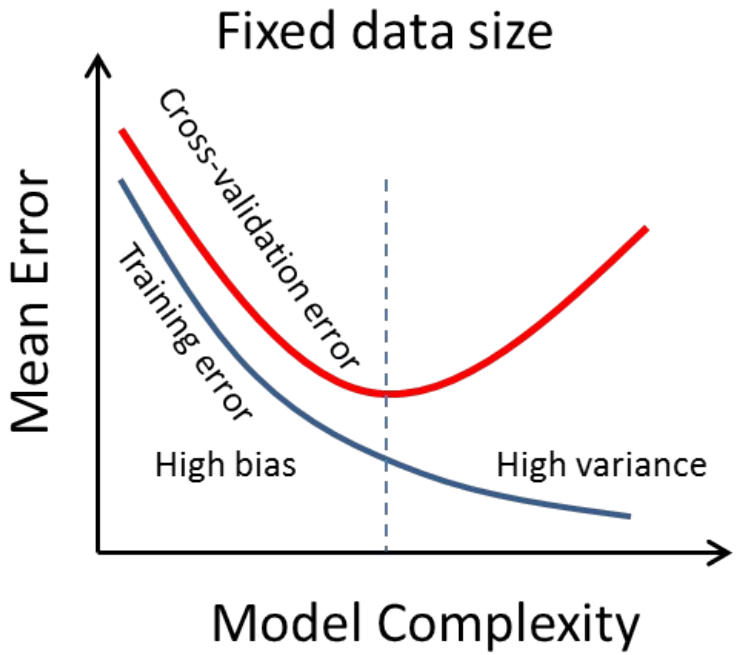
- generalize badly
  - learned noise
  - overly complex model
- represent training set well
- bad performance on unseen data

Bias, variance



Illustrating predictions





Decision trees, C4.5

Decision trees

- Extending ID3
  - numeric attributes: pretty straightforward
  - missing values: a bit trickier
  - stability for noisy data: requires sophisticated ***pruning*** mechanism
- End result of these modifications: C4.5 (Weka: J48)
  - Best-known and (probably) most widely-used learning algorithm

Numeric attributes

- Standard method: binary splits (e.g. 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
- Select attribute – nominal or numeric – with highest info gain for node

Numeric attributes: 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	Yes	No	Yes	Yes	No	Yes	Yes	No

  - Split points are placed halfway between values
  - E.g. 71.5
    4 „yes”es and 2 „no”s for `temperature < 71.5`,
    5 „yes”es and 3 „no”s for `temperature ≥ 71.5`
    - Info([4,2],[5,3]) = (6/14)info([4,2]) + (8/14)info([5,3]) = 0.939 bits
  - Calculation for all (!) split-points, selecting best split-point

(Numeric attributes: efficient execution)

- Instances need to be sorted according to the values of the numeric attribute considered
- All split points can be evaluated in one pass!

- Does sorting have to be repeated at each node?
- No! Sort order from parent node can be used to derive sort order for children
  - Only drawback: need to create and store an array of sorted indices for each numeric attribute

Numeric attributes: Notes on binary splits

- Information in nominal attributes is exhausted using one multi-way split on that attribute
  - Nominal attribute is tested (at most) once on any path in the tree
- 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
  - multi-way splits instead of binary ones

Missing values

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

Tree overfitting

- ID3 stop criterion
  - Leaf node pure
  - No more attribute to select
- Tends to overfit
- Lower levels (close to leaf nodes) decided on small number of training instances

Pruning

- Simplifying decision tree
  to prevent overfitting
- *Postpruning*:
  takes a fully-grown decision tree and discards unreliable parts
- *Prepruning*:
  stops growing a branch when no further improvement (information becomes unreliable)

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
- Problem: **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
- Prepruning faster than postpruning

Postpruning

- Problem: some subtrees might be due to chance effects/noise
- First, build full tree
- Then, prune (=simplify) it
  - Fully-grown tree shows all attribute interactions
- Postpruning preferred in practice because of *early stopping* in prepruning

Postpruning in C4.5: pruning operations, pruning strategy

Pruning operations: **How** to simplify the tree

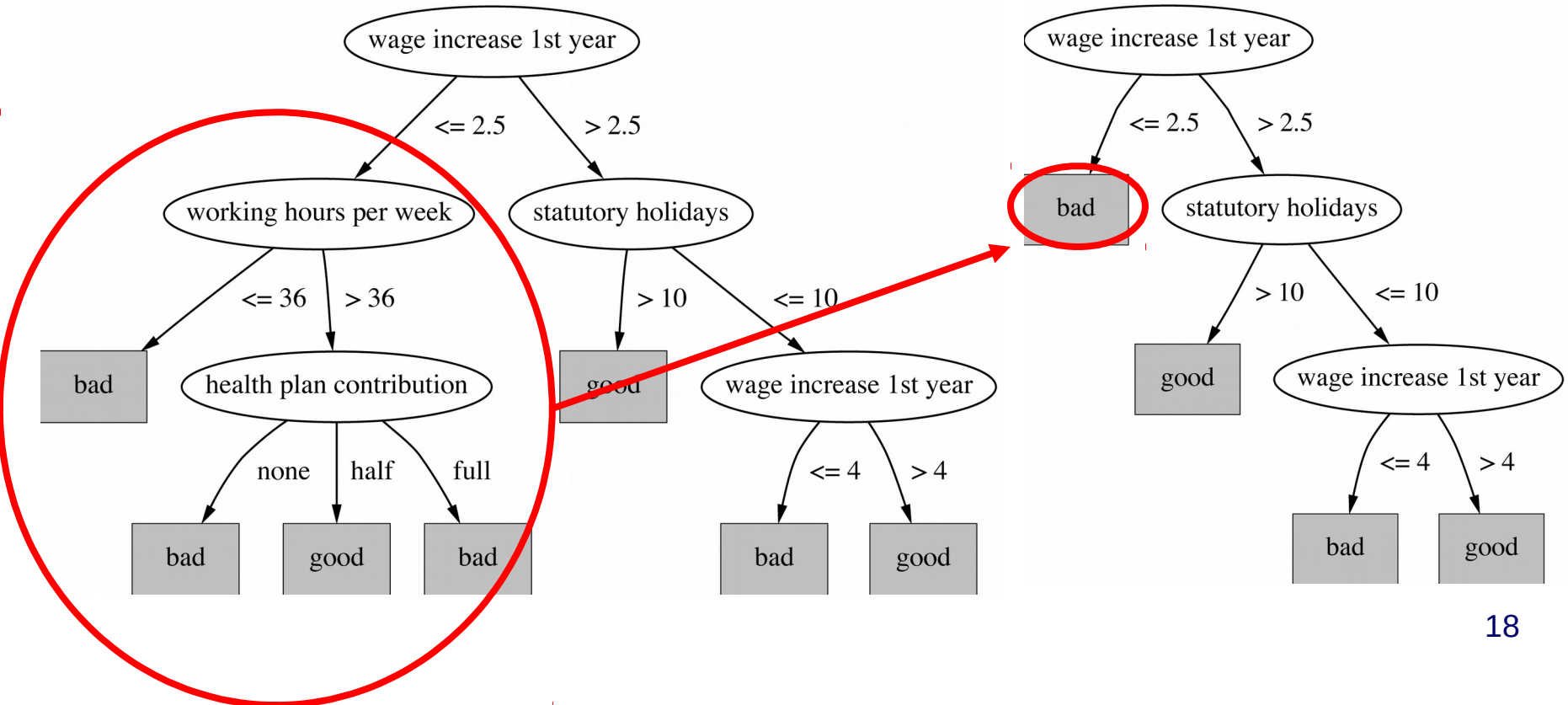
1. Subtree replacement
2. Subtree raising

Pruning strategy: ***W*hen** to apply a pruning operation

- error estimation

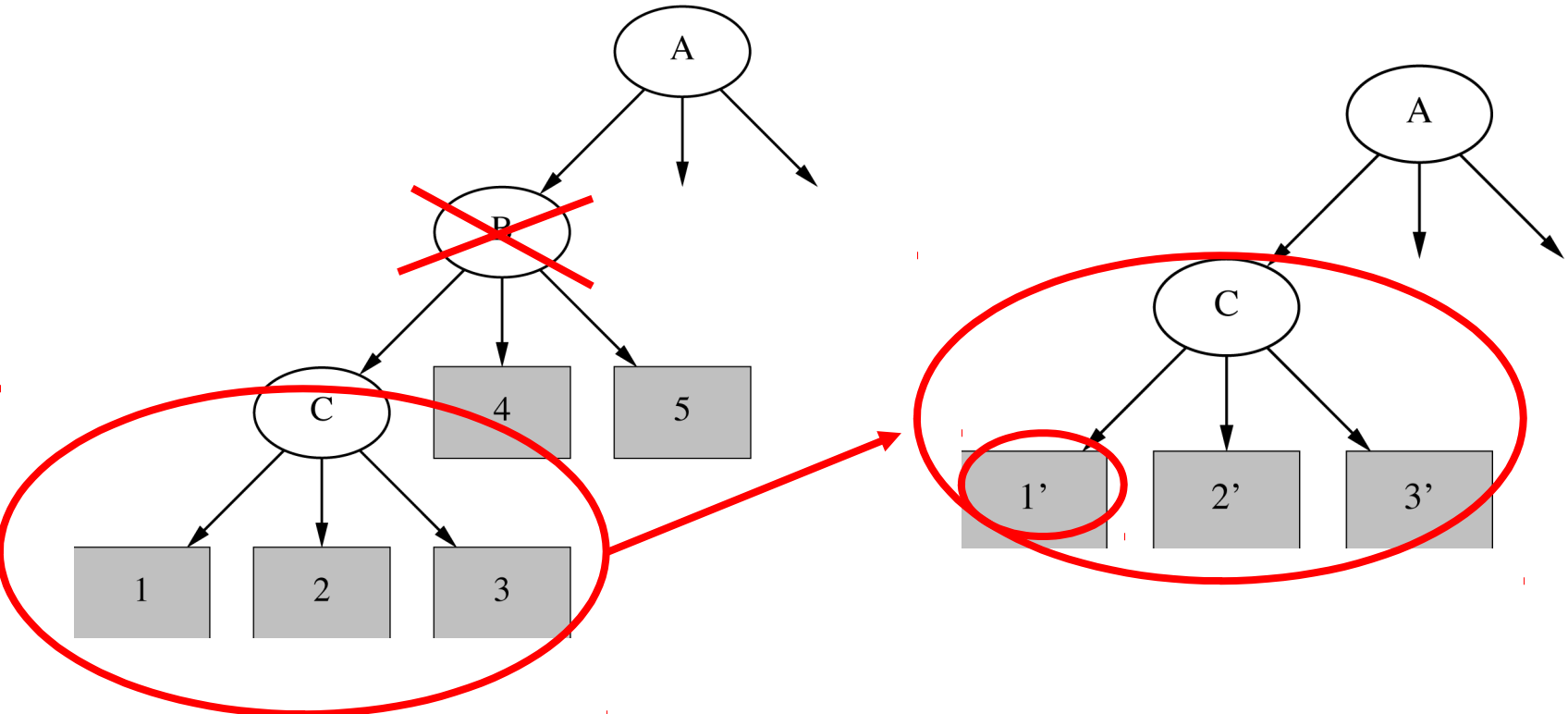
Pruning operation: Subtree replacement

- Select some subtree and replace it with single leaf
- Bottom-up: tree is considered for replacement once all its subtrees have been considered

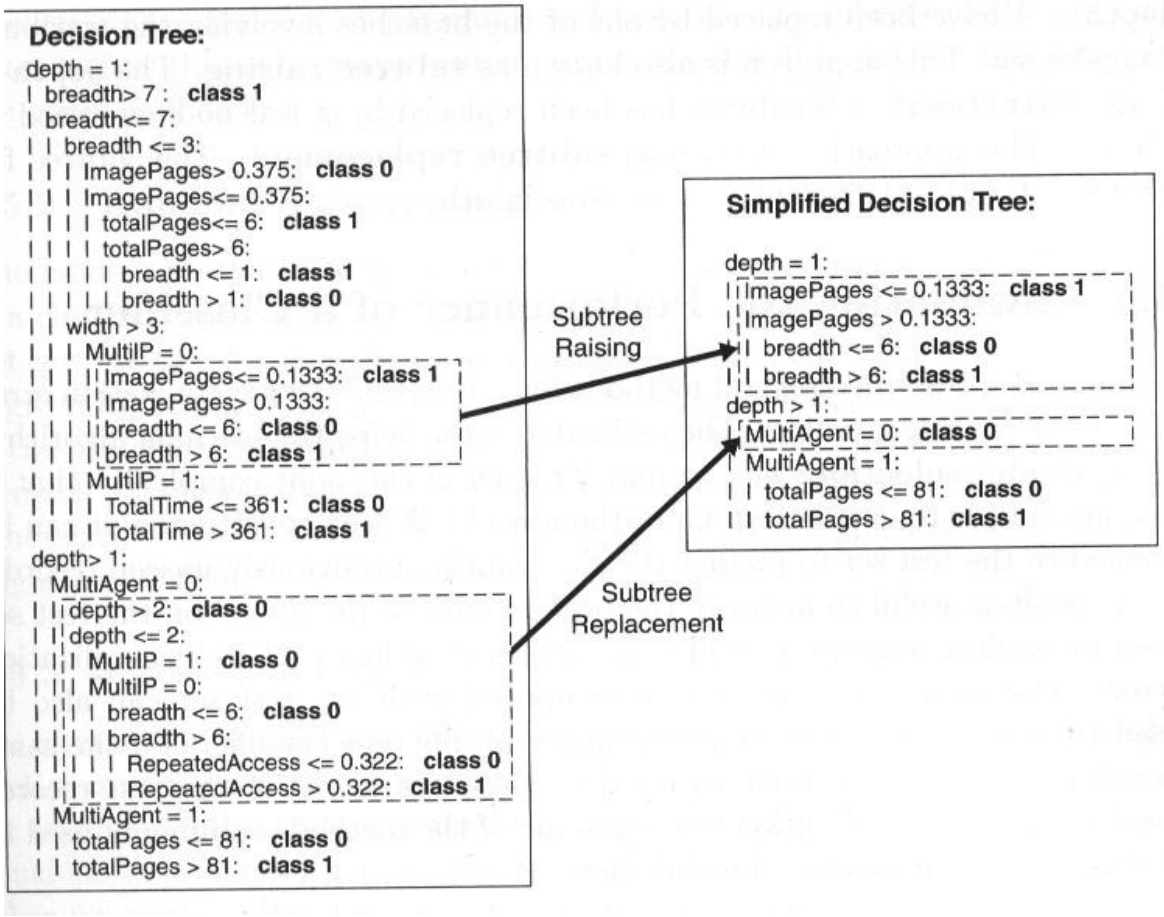


Pruning operation: Subtree raising

- Deletes node and redistributes instances
- Slower than subtree replacement
  - Is it useful? C4.5 does it that way



Postpruning Example



Pruning strategy: Estimating error rates

- Prune only if it does not increase the estimated error

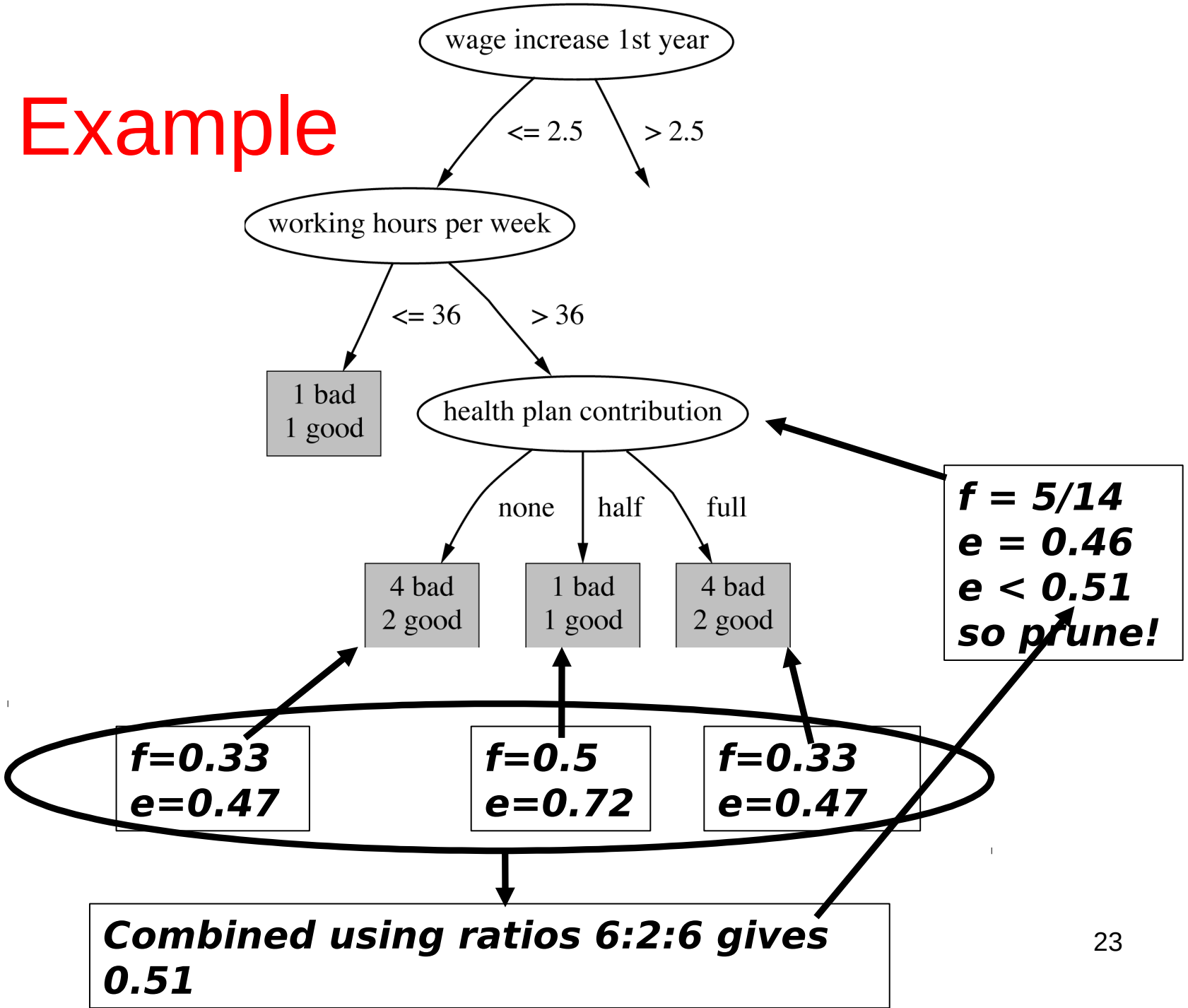
- Methods
  - Error on the training data is NOT a useful estimator *(would result in almost no pruning)*
  - Use hold-out set (validation set) for pruning
    - Disadvantage: amount of data used for tree construction smaller
  - C4.5's method
    - Derive confidence interval from training data
      - (Also considers amount of training data at this point in the tree)
    - Use a heuristic limit, derived from this, for pruning
    - Standard Bernoulli-process-based method
    - Shaky statistical assumptions (based on training data)

C4.5's method

- Error estimate for subtree is weighted sum of error estimates for all its leaves
- Error estimate for a node:
  - *f:* error on the training data
  - *N:* the number of instances covered by the leaf
  - z: related to confidence level
    - If *c* = 25% (c: parameter of C4.5) then *z* = 0.69 (from normal distribution)

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

Example


wage increase 1st year
<= 2.5
> 2.5
working hours per week
<= 36
> 36
1 bad
1 good
health plan contribution
none
half
full
4 bad
2 good
1 bad
1 good
4 bad
2 good
f = 5/14
e = 0.46
e < 0.51
so prune!
f=0.33
e=0.47
f=0.5
e=0.72
f=0.33
e=0.47
Combined using ratios 6:2:6 gives 0.51

C4.5: choices and options

- C4.5 can be slow for large and noisy datasets
- 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)

CART (Classification and Regression Trees)

- Breiman, Friedman, Olshen, Stone, 80's.
- Gini index
- Cost-complexity postpruning
- …
- Handles outliers
- Binary trees

GINI Impurity

- how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset

$$\mathrm{I}_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k$$

  - J classes
  - p_i fraction of items labelled with i in the set

Cost-complexity pruning

- Error relative to size of subtree
  - Increase in error (α): average error increase per leaf of subtree
- Pruning generates a sequence of successively smaller trees
  - Each candidate tree in the sequence corresponds to one particular threshold value, α_i
- Which tree to choose as the final model?
  - Use either a hold-out set or cross-validation to estimate the error of each

(Comparison of Classification Tree Methods)

TABLE 1 | Comparison of Classification Tree Methods. A Check Mark Indicates Presence of a Feature

Feature	C4.5	CART	CHAID	CRUISE	GUIDE	QUEST
Unbiased Splits				√	√	√
Split Type	*u*	*u,l*	*u*	*u,l*	*u,l*	*u,l*
Branches/Split	≥2	2	≥2	≥2	2	2
Interaction Tests				√	√	
Pruning	√	√		√	√	√
User-specified Costs		√	√	√	√	√
User-specified Priors		√		√	√	√
Variable Ranking		√			√	
Node Models	*c*	*c*	*c*	*c,d*	*c,k,n*	*c*
Bagging & Ensembles					√	
Missing Values	*w*	*s*	*b*	*i,s*	*m*	*i*

b, missing value branch; *c*, constant model; *d*, discriminant model; *i*, missing value imputation; *k*, kernel density model; *l*, linear splits; *m*, missing value category; *n*, nearest neighbor model; *u*, univariate splits; *s*, surrogate splits; *w*, probability weights

Discussion

- Top-down induction of decision trees 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 (one attribute per test)
  - Some systems (e.g. CART) can build *multivariate* trees (more than one attributes per test allowed)

Regularized linear regression

Regularization

- discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.
- minimizing error on data + **complexity of model**

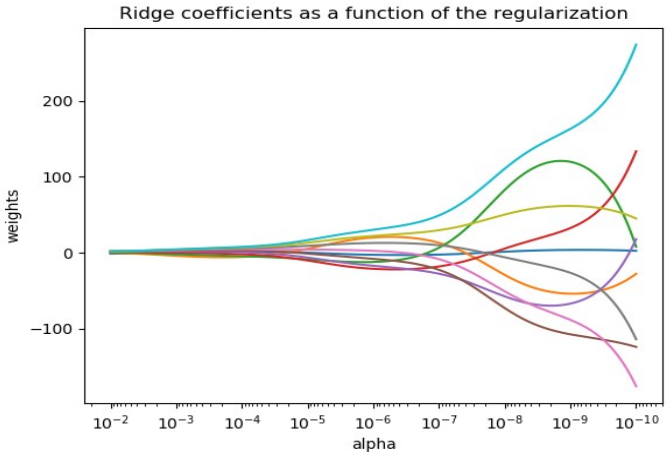
Ridge Regression

- Ordinary Linear Regression
  - Minimizes Least Square of Error, LS

- Ridge regression
  - Minimization objective = LS + α * (sum of square of coefficients)
  - L2 regularization
  - alpha >=0: complexity parameter

Ridge Regression 2

- α>=0: complexity parameter controls amount of „shrinkage”: the larger, the greater the amount of „shrinkage”
- Cross validation



- Coefficients become also more robust to collinearity

Multicollinearity

- Input variables that are correlated with other input variables in the model
- Coefficient estimates are unstable

- Doesn’t affect the overall fit of the model or produce bad predictions
  but interferes in determining the precise effect of each input variable

Lasso regression

- Lasso: *Least Absolute Shrinkage and Selection Operator*
- Minimization objective = LS + α * (sum of absolute value of coefficients)
- L1 regularization

- Prefers solutions with fewer parameter values, reducing the number of variables

Ridge vs Lasso

- model complexity reduction
- overfitting prevention
- **Ridge:**
  - exorbitantly high #features (~ millions): computational challenges
  - highly correlated features OK (includes all)
- **Lasso:**
  - shrinking coefficients+ feature selection as well (coefficients become exactly zero)
  - #features are in millions or more
  - highly correlated features: arbitrary selection

Instance Based Learning, (Nearest Neighbour)

Problem 0: Different Attribute Domains

(some) normalisation of attributes!

Problem 1: slow predictions (+ storage place)

Solutions

- Data structure: multidimensional tree
  - kD tree
  - Ball tree
- Do not store all instances
  - saves memory, speeds up classification
  - Simple approach: Incremental model building, only incorporating misclassified instances has problems:
    - noisy data gets incorporated
    - discarding instances too early

Problem 2: noisy data

Solutions

- k-NN
  - Voting (averaging)
  - selection of K?
    - the more noise, the larger k
    - Cross-validation for good k
  - Weighing with distance
- Radius-neighbors
  (for non-uniformly sampled data)
- Ignore or remove noisy instances
  Monitoring performance of instances
    - Two thresholds: ignoring for predictions, discarding

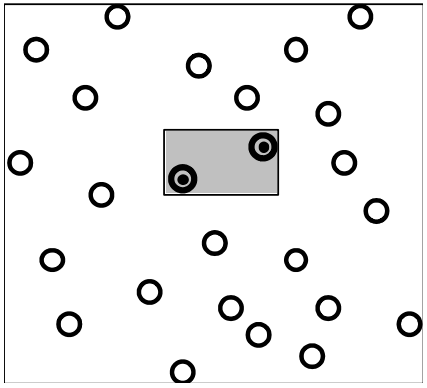
Problem 3: All attributes equally important?

Solutions

- attribute selection (s. later; irrelevant attributes!)
- attribute weighting
  - 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: weight change large/small
    - Class specific weights
      - weights, weighing per class
      - can result in unclassified instances and multiple classifications

Problem 4: Black box

- Solution
  - Rectangular generalisations
  - In fact: hybrid representation
    - Rectangulars: rules
    - Outside of rectangulars: nearest-neighbour



Discussion

- Simple method
- No generalisation
- Often good predictions!
  - irregular, flexible decision boundary

Numeric prediction with local linear models

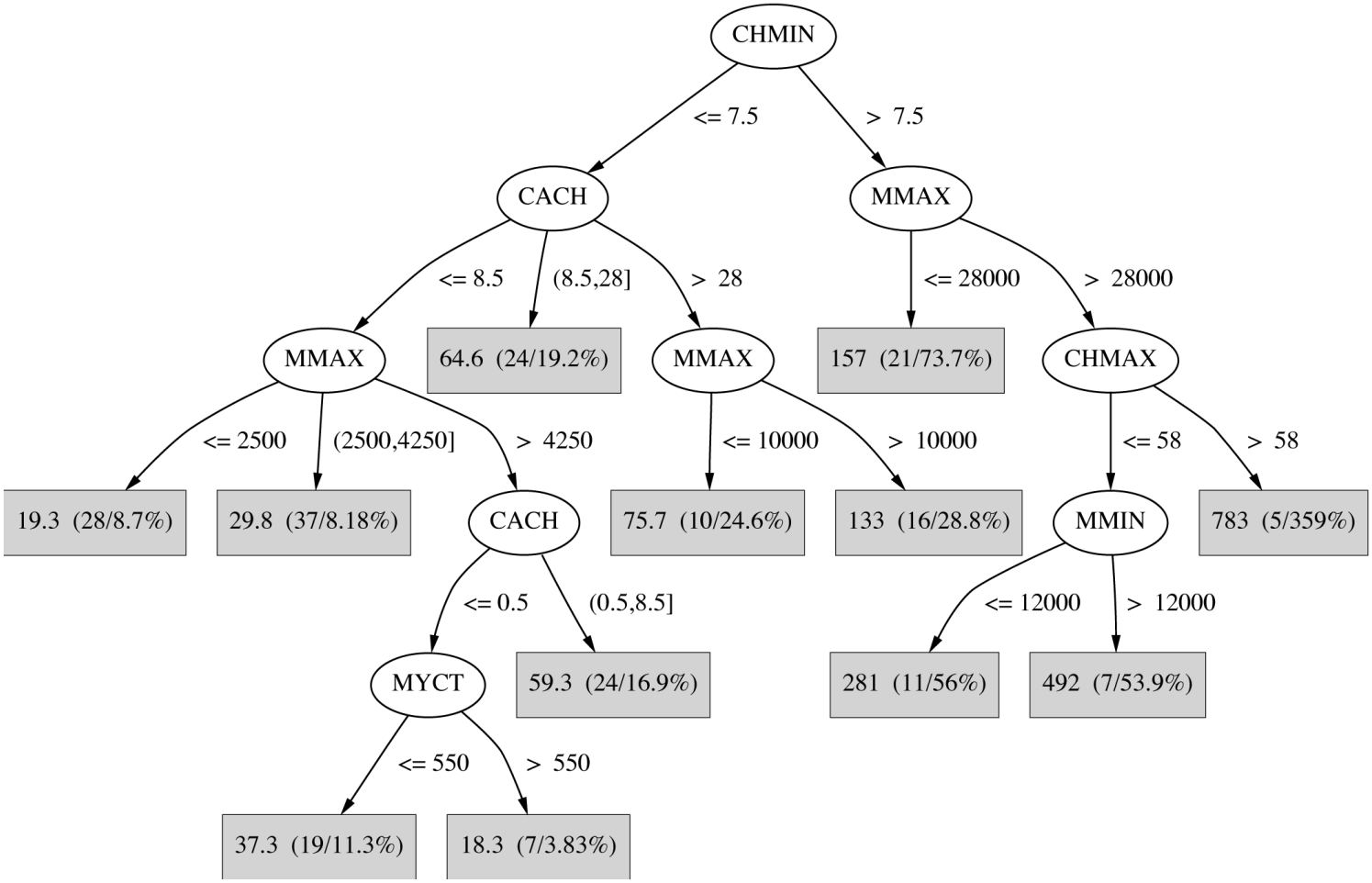
Numeric prediction

- Counterparts exist for (almost all) classification algorithms
- All classification algorithms can be applied to regression problems using discretization (almost)
  - 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

- Leaf node predicts average values of training instances reaching that node
- Easy to interpret

Regression tree for the CPU data

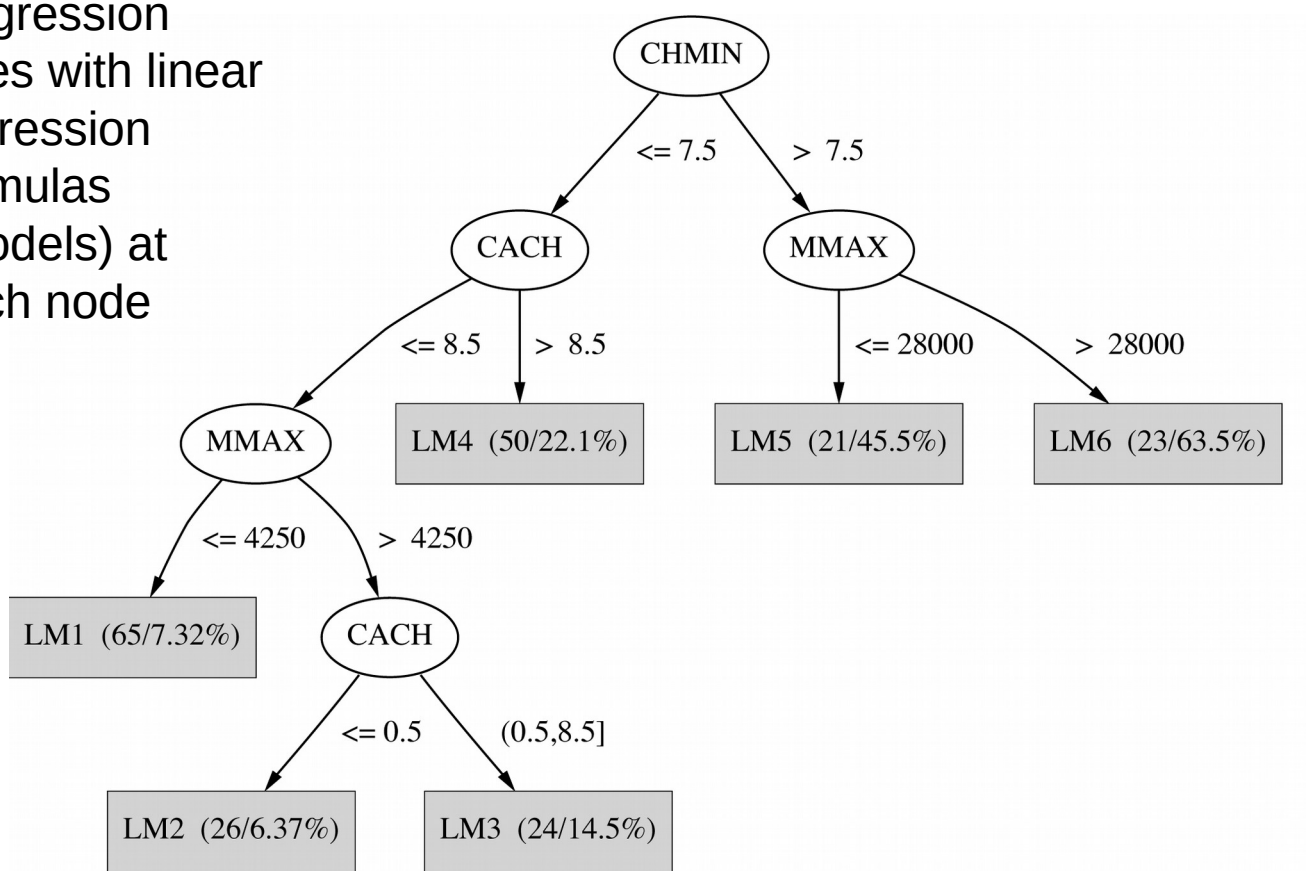


Building the tree

- Splitting criterion: *standard deviation reduction*
  - minimizing intra-subset variation
- Termination criteria (important when building trees for numeric prediction):
  - Standard deviation becomes smaller than certain fraction of s. d. for full training set (e.g. 5%)
  - Too few instances remain (e.g. less than four)
- Pruning criterion: based on numeric error measure

Model tree for the CPU data

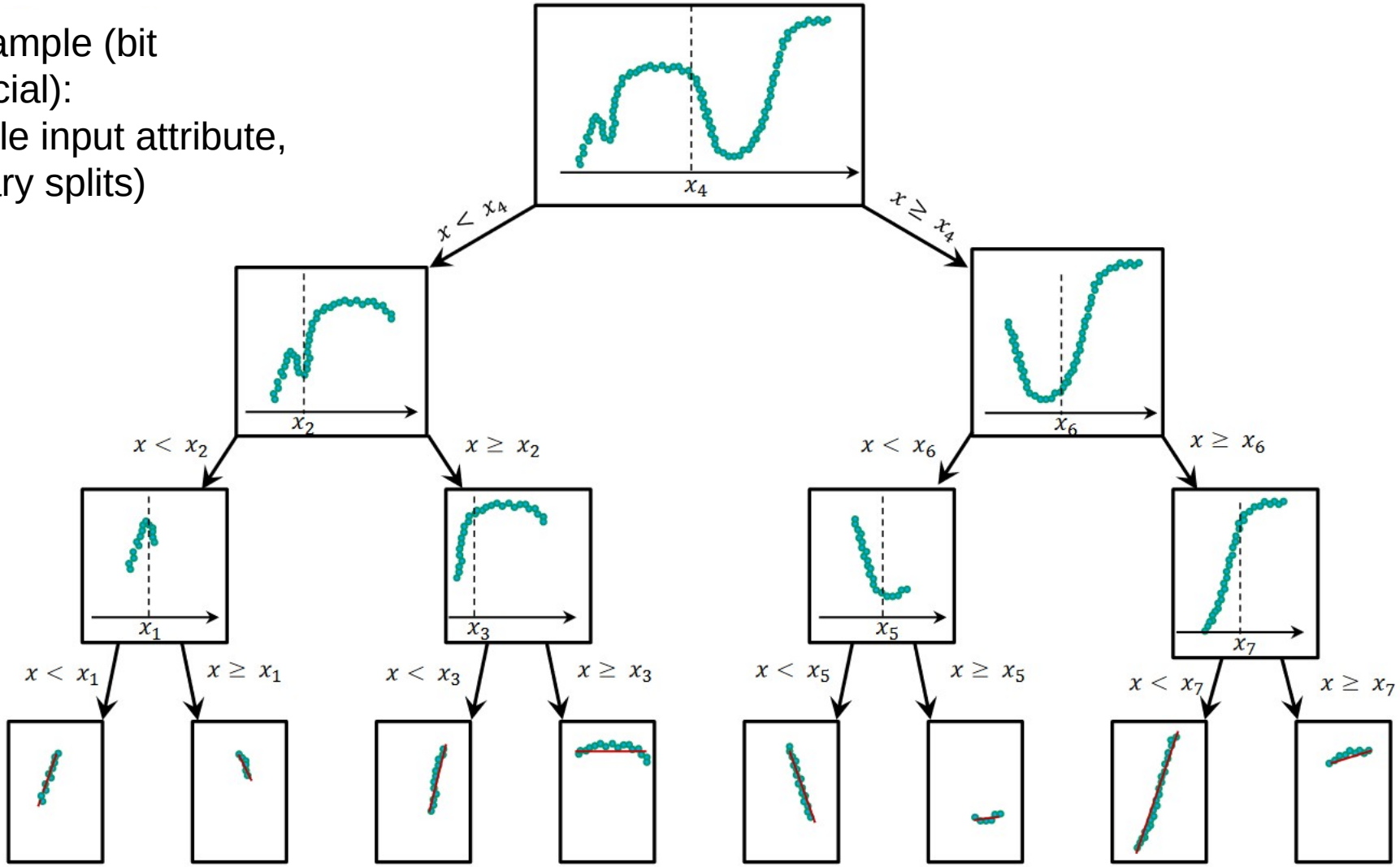
Regression trees with linear regression formulas (models) at each node



Linear Regression Functions

- LM1 PRP = 8.29 + 0.004 MMAX + 2.77 CHMIN
- LM2 PRP = 20.3 + 0.004 MMIN – 3.00 CHMIN
  + 0.946 CHMAX
- etc.

(Example (bit special):
single input attribute, binary splits)



Building model trees

- 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

- Smoothing: factor in ancestor's predictions
  - Smoothing formula: $p' = \frac{np + kq}{n + k}$
  - Same effect can be achieved by incorporating ancestor models into the leaves
- Need linear regression function at each *node*

Pruning

- Pruning is based on estimated absolute error of LR models
- 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

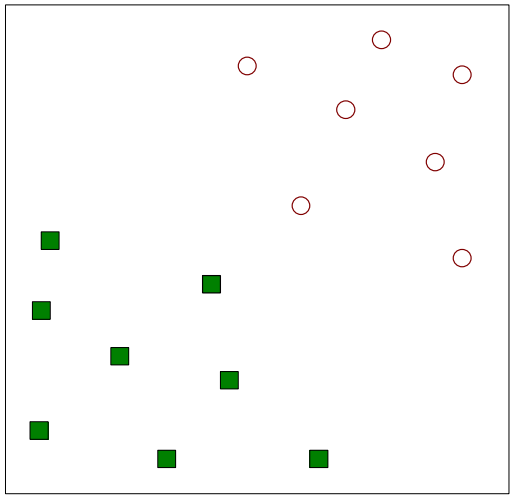
Nominal attributes

- Nominal attributes are converted into binary attributes (that can be treated as numeric ones)
  - Nominal values are sorted using average class values
  - If there are *k* values, *k*-1 so-called synthetic 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

Support Vector Machine (SVM)

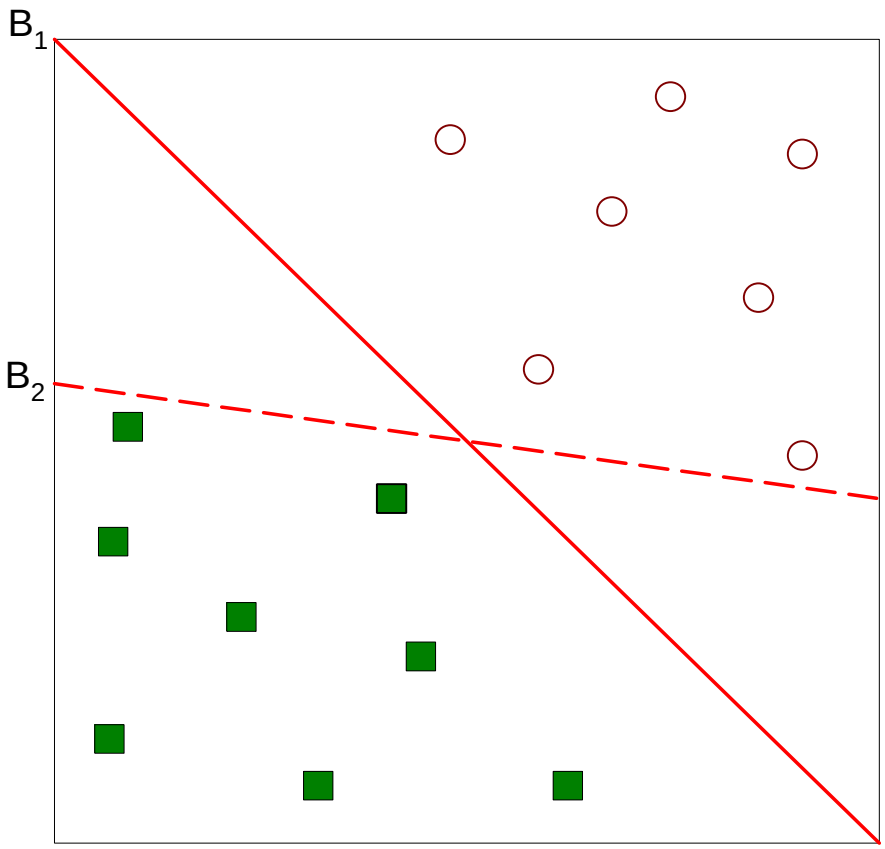
Support vector machines

- *Support vector machines* are algorithms for learning linear classifiers (for binary classification)



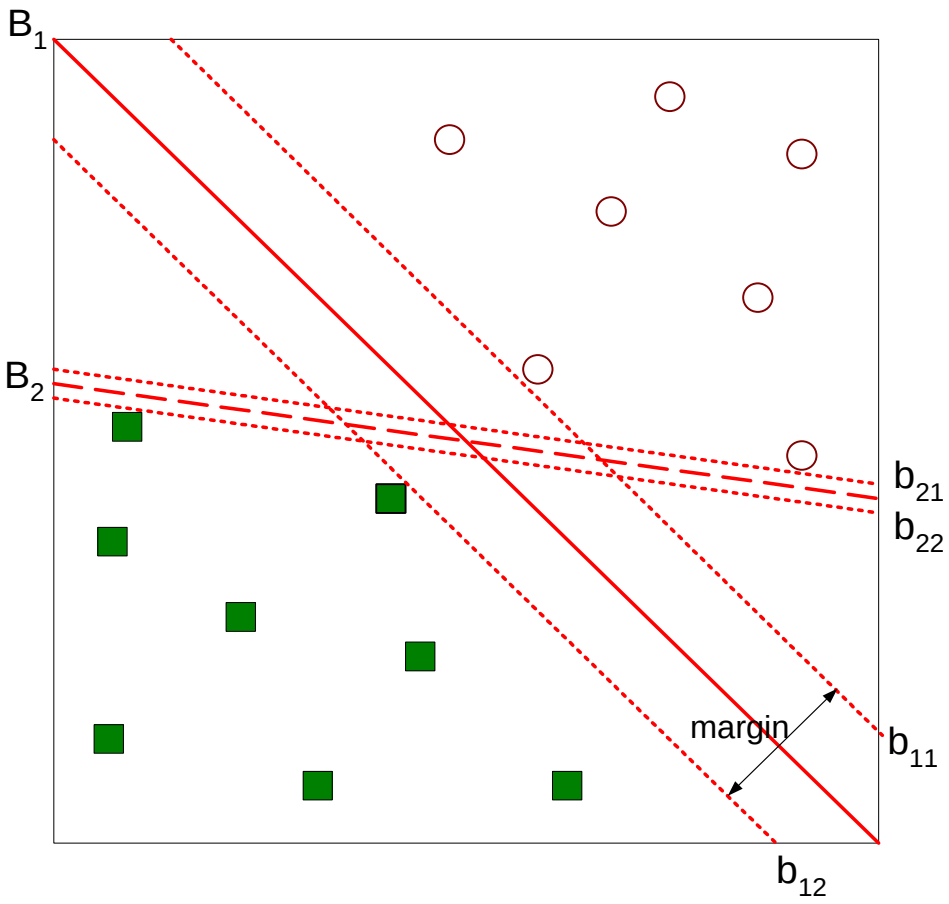
Find a linear hyperplane (decision boundary) that will separate the data

Many linear decision boundaries



- Which one is better? B1 or B2?
- How do you define better?

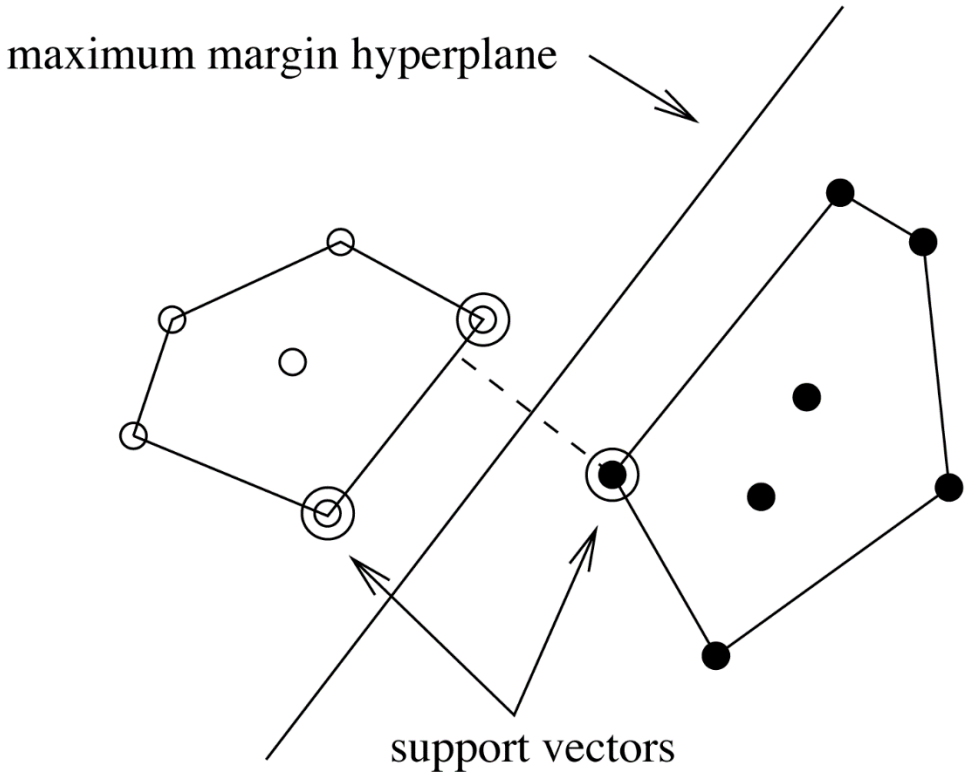
Maximum Margin Hyperplane



Find hyperplane maximizes the margin => B1 is better than B2

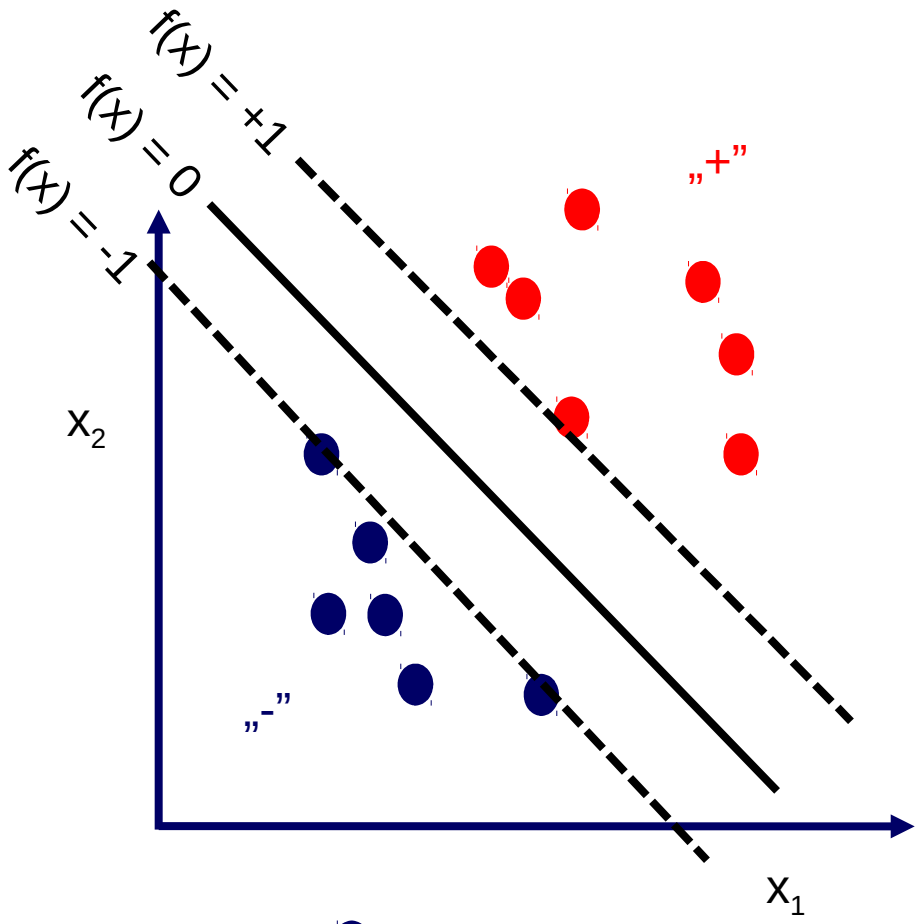
Maximum margin hyperplane

Support Vectors



- Only a few instances are important for the maximum margin hyperplane: the closest ones
  - Those are called *support vectors*
  - All other instances can be deleted without changing the decision boundary

Finding the Maximum Margin Hyperplane



$f(x) = b+w_1*x_{12}+w_{12}*x_{12}+\ldots = 0$

$b+ \mathbf{w}\cdot\mathbf{x} = 0$

Decision rule:

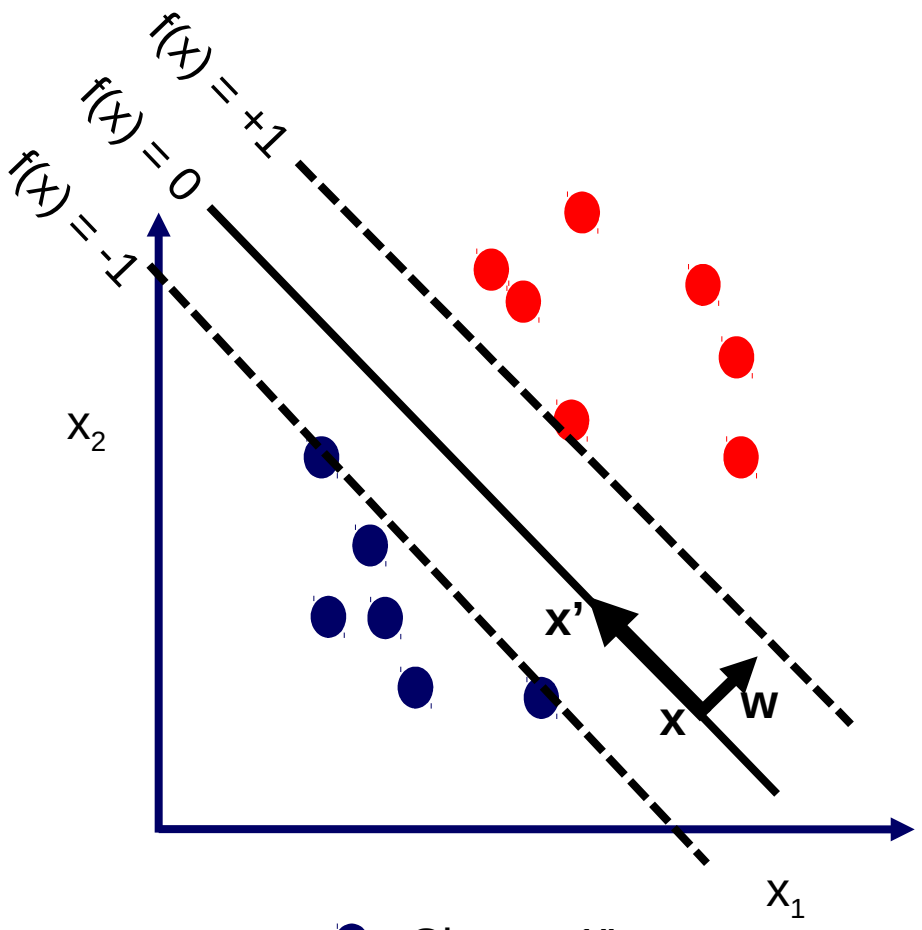
$b+ \mathbf{w} \cdot \mathbf{x} > 0$ **then „+”**

$b+ \mathbf{w} \cdot \mathbf{x} < 0$ **then „-”**

Scale invariant, fixing scale:
for support vectors: -1, +1

Class „+” Class „-”

Margin width 1



$f(x) = b + \mathbf{w}*\mathbf{x} = 0$

w orthogonal to hyperplane:

$b+\mathbf{w}\cdot\mathbf{x}=0$ and $b+\mathbf{w}\cdot\mathbf{x}'=0$

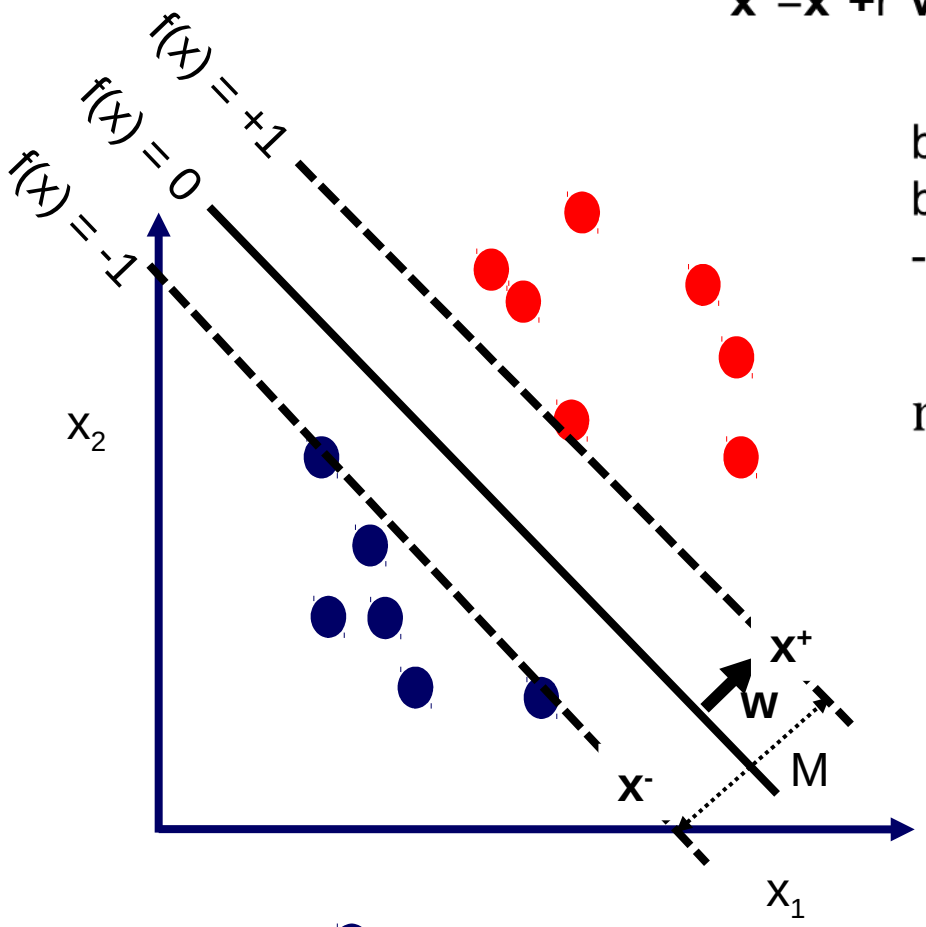
$\mathbf{w}\cdot(\mathbf{x}'-\mathbf{x}) = 0$

Class „+1”

Class „-1”

Margin width 2

Any $\mathbf{x}^-$ such that $f(\mathbf{x}^-) = b+\mathbf{w} \cdot \mathbf{x}^- = -1$
$\mathbf{x}^+$ closest point with $f(\mathbf{x}^+) = b+\mathbf{w}\cdot\mathbf{x}^+ = +1$
$\mathbf{x}^+=\mathbf{x}^- +r*\mathbf{w}$



$b+\mathbf{w} \cdot \mathbf{x}^+ = +1$
$b+\mathbf{w} \cdot(\mathbf{x}^- +r*\mathbf{w}) = +1$
$-1 + r\,||\mathbf{w}||^2 = +1$

$$r = \frac{2}{||\boldsymbol{w}||^2}$$

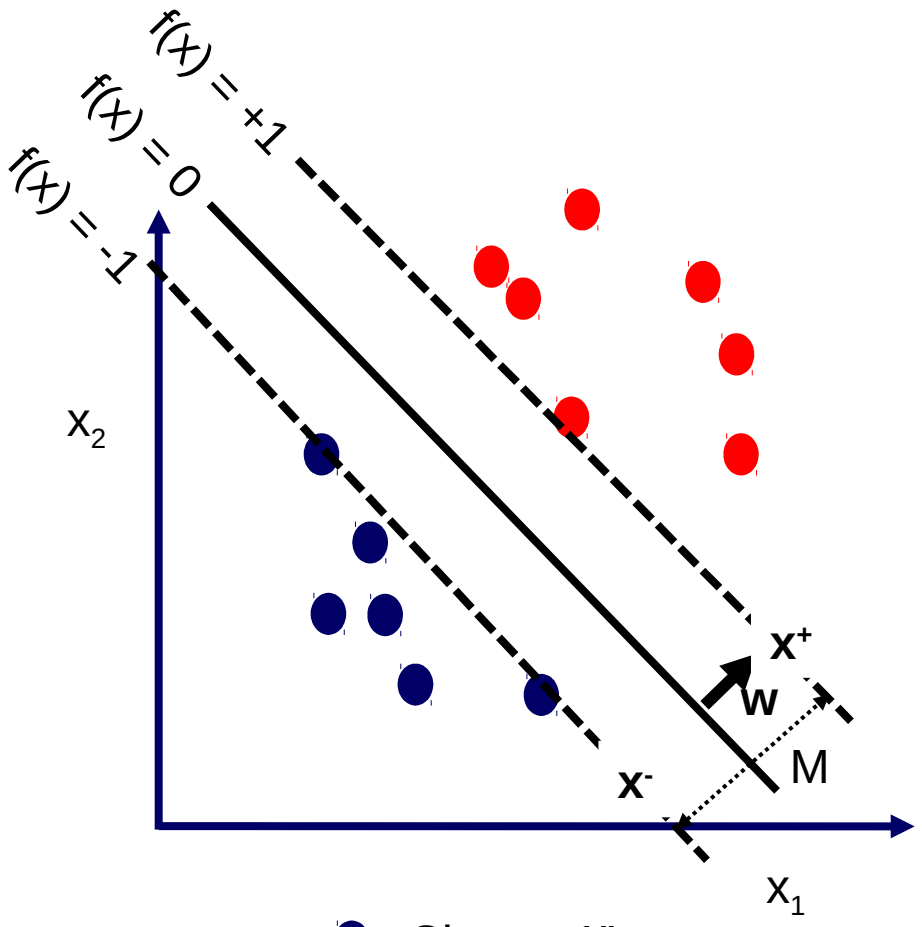
$$M = \left|\left|\mathbf{x}^+ + \mathbf{x}^-\right|\right| = \|r\mathbf{w}\| =$$

$$= \frac{2}{||\mathbf{w}||^2}||\boldsymbol{w}|| = \frac{2}{||\boldsymbol{w}||}$$

Class „+” Class „-”

Maximum Margin

$$\mathbf{w}^* = \text{argmax}_{\mathbf{w}} \frac{2}{||\boldsymbol{w}||} \qquad \mathbf{w}^* = \min_{\mathbf{w}} ||\boldsymbol{w}|| \qquad \mathbf{w}^* = \min_{\mathbf{w}} \frac{1}{2} * ||\boldsymbol{w}||^2$$



Such that m (number of instances) constraints satisfied:

$$y_i = +1: \quad \mathbf{w} \cdot \mathbf{x_i} + \mathrm{b} \geq +1$$

$$y_i = -1: \quad \mathbf{w} \cdot \mathbf{x_i} + \mathrm{b} \leq -1$$

Compact form of all constraints:

$$y_i * (\mathbf{w} \cdot \mathbf{x_i} + \mathrm{b}) - 1 \geq 0$$

Primal problem

Constrained quadratic optimization

Class „+1”

Class „-1”

Lagrangian optimization

$\mathbf{w}^* = \min_{\mathbf{w}} \frac{1}{2} * ||\boldsymbol{w}||^2$ with constraints $y_i * (\mathbf{w} \cdot \mathbf{x_i} + b) - 1 \geq 0$

Lagrange multiplier:
searching the extreme value of some function under some constraints
->
searching the extreme value (no constraints)

$L = \frac{1}{2} * \|\mathbf{w}\|^2 - \sum_i \alpha_i [y_i * (\mathbf{w} \cdot \mathbf{x_i} + b) - 1]$

$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i (y_i * \mathbf{x_i}) = 0$

$$\mathbf{w} = \sum_i \alpha_i (y_i * \mathbf{x_i})$$

$\frac{\partial L}{\partial b} = -\sum_i \alpha_i * y_i = 0$

$\sum_i \alpha_i * y_i = 0$

Lagrangian optimization

$$\mathbf{w} = \sum_i \alpha_i (y_i * \mathbf{x}_i)$$

$$\sum_i \alpha_i * y_i = 0$$

$$L = \tfrac{1}{2} * \|\mathbf{w}\|^2 - \sum_i \alpha_i [y_i * (\mathbf{w} \cdot \mathbf{x}_i + b) - 1]$$

$$L = \tfrac{1}{2} * \left(\sum_i \alpha_i * y_i * \mathbf{x}_i\right) \cdot \left(\sum_j \alpha_j * y_j * \mathbf{x}_j\right) - \sum_i \alpha_i * y_i * \mathbf{x}_i \cdot \left(\sum_j \alpha_j * y_j * \mathbf{x}_j\right) - \sum_i \alpha_i * y_i * b + \sum_i \alpha_i$$

$$L = \sum_i \alpha_i - \frac{1}{2} * \left(\sum_i \sum_j \alpha_i * \alpha_j * y_i * y_j * \mathbf{x}_i \cdot \mathbf{x}_j\right)$$

Extreme value search of this.

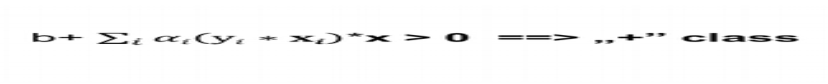
Lagrangian dual of original problem

Optimization only depends on dot products of pairs of instances!!!

- Number of dimensions: number of instances (original: number of attributes) – will become important for nonlinear case
- No local extreme values: optimal parameters can be found!

Decision Rule transformed

b+ **$\mathbf{w \cdot x > 0}$ ==> „+” class**

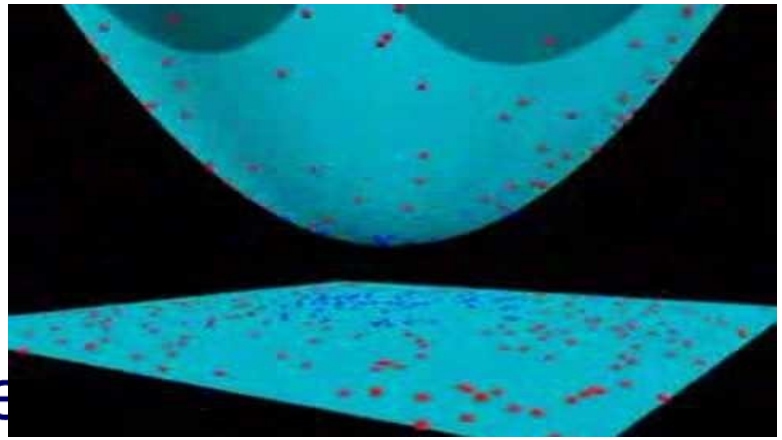


Decision rule also only depends on dot product!

„Everything” depens only on the dot products!!!

Nonlinear SVMs

- Linearly non separable data?
- Nonlinear transformation: $\Phi(x)$
  - New space: classes linearly separable
  - Demo: SVM with polynomial kernel visualization:



- Computation time

Kernel trick

- $\boldsymbol{\Phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\Phi}(\boldsymbol{x}_j)$ – only dot products needed!
- Avoid computing new attributes explicitely! Use a *Kernel function*!
  - $\mathrm{K}(\mathbf{x_i}; \mathbf{x_j}) = \boldsymbol{\Phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\Phi}(\boldsymbol{x}_j)$

Kernel Functions

- Polynomial kernel

$$x = b + \sum_{\text{i is supp. vector}} \alpha_i y_i (\vec{a}(i) \cdot \vec{a})^n$$

- $$K(\vec{x}_i, \vec{x}_j) = \tanh(\beta \vec{x}_i \cdot \vec{x}_j + b)$$

$$K(\vec{x}_i, \vec{x}_j) = \exp\left(\frac{-(\vec{x}_i - \vec{x}_j)^2}{2\sigma^2}\right)$$

- **How to select kernel for SVM?**
  - https://stats.stackexchange.com/questions/18030/how-to-select-kernel-for-svm
  - Prior, expert knowledge
    (automatic selection, tuning: very easy to overfit!)

Noise

- Have assumed that the data is separable (in original or transformed space)
  - Few noisy instances can cause big problems!
- Can apply SVMs to noisy data by introducing a *regularization parameter C*
- *C* bounds the influence of any one training instance on the decision boundary
- Corresponding constraint: $0 =< \alpha_i =< C$
- Still a quadratic optimization problem
- Have to determine *C* by experimentation

Advantages

- maximizes margin: (some) robustness
- **support for kernels – computable approach for nonlinear problems**
- regularization parameter to reduce overfitting
- Kernel: you can build in expert knowledge about the problem via engineering the kernel
- SVM is defined by a convex optimization problem (no local minima) for which there are efficient methods (e.g. SMO).

Disadvantages

- choosing kernel function?
- hyperparameter selection for avoiding overfitting?
- high algorithmic complexity and extensive memory requirements of the required quadratic programming in large-scale tasks

Sparse data

- SVM algorithms speed up dramatically if the data is *sparse* (i.e. many values are 0)
  - lots and lots of dot products: for sparse data very efficient, iterate only over non-zero values
- SVMs can process sparse datasets with 10,000s 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

Support vector regression

- Maximum margin hyperplane only applies to classification
- However, idea of support vectors and kernel functions can be used for regression
- Basic method same as in linear regression: want to minimize error. Differences
  - ignore errors smaller than ε (user specified parameter)
  - use absolute error instead of squared error
  - simultaneously aim to maximize flatness of function (= avoid overfitting)

History …

- **Vladimir Vapnik**

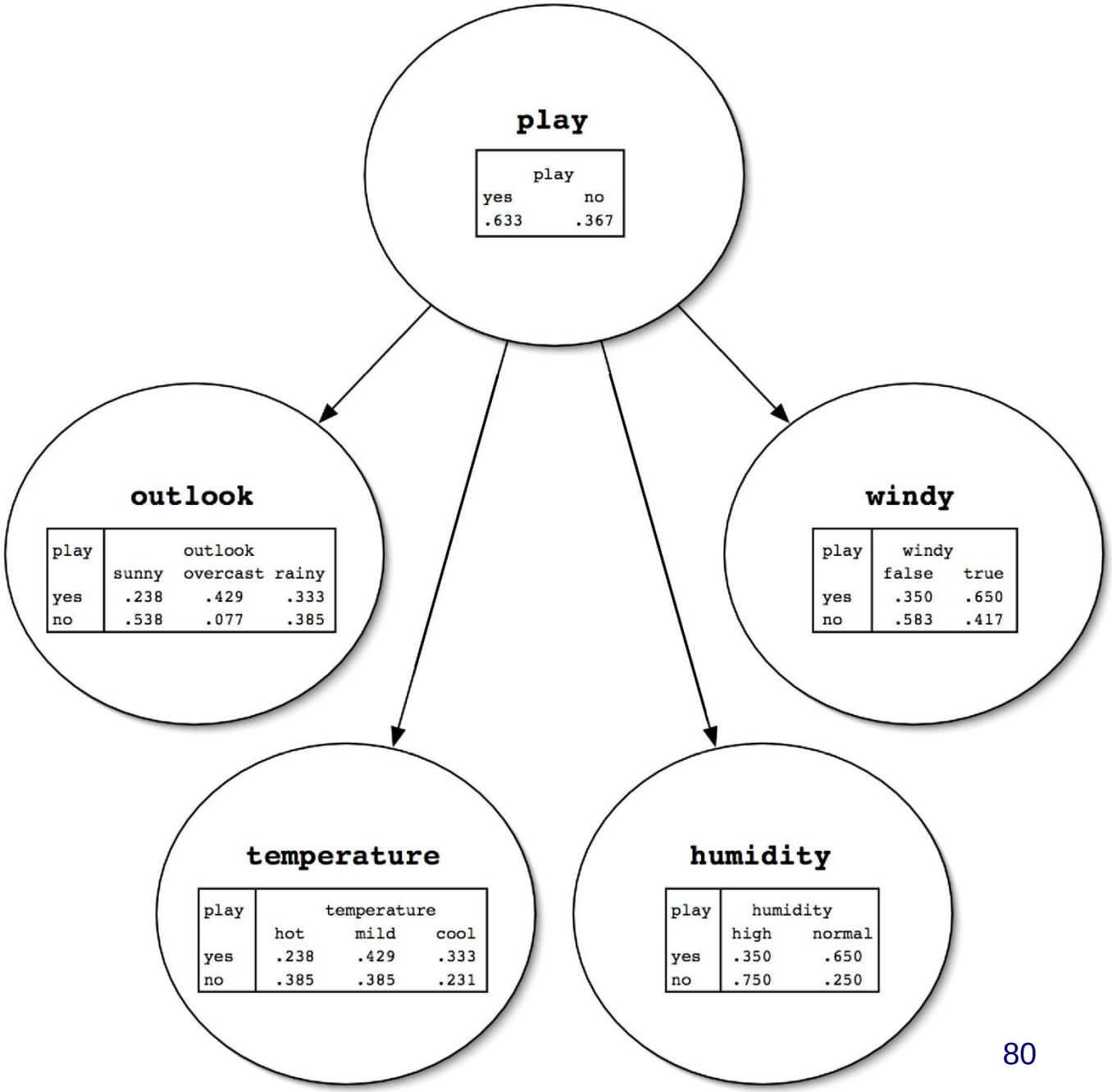
- https://youtu.be/_PwhiWxHK8o?t=2779

Bayesian Networks

From naïve Bayes to Bayesian Networks

- Naïve Bayes assumes:
  attributes conditionally independent given the class
- Doesn’t hold in practice but classification accuracy often high
- However: sometimes performance much worse than e.g. decision tree
- Can we eliminate the assumption?

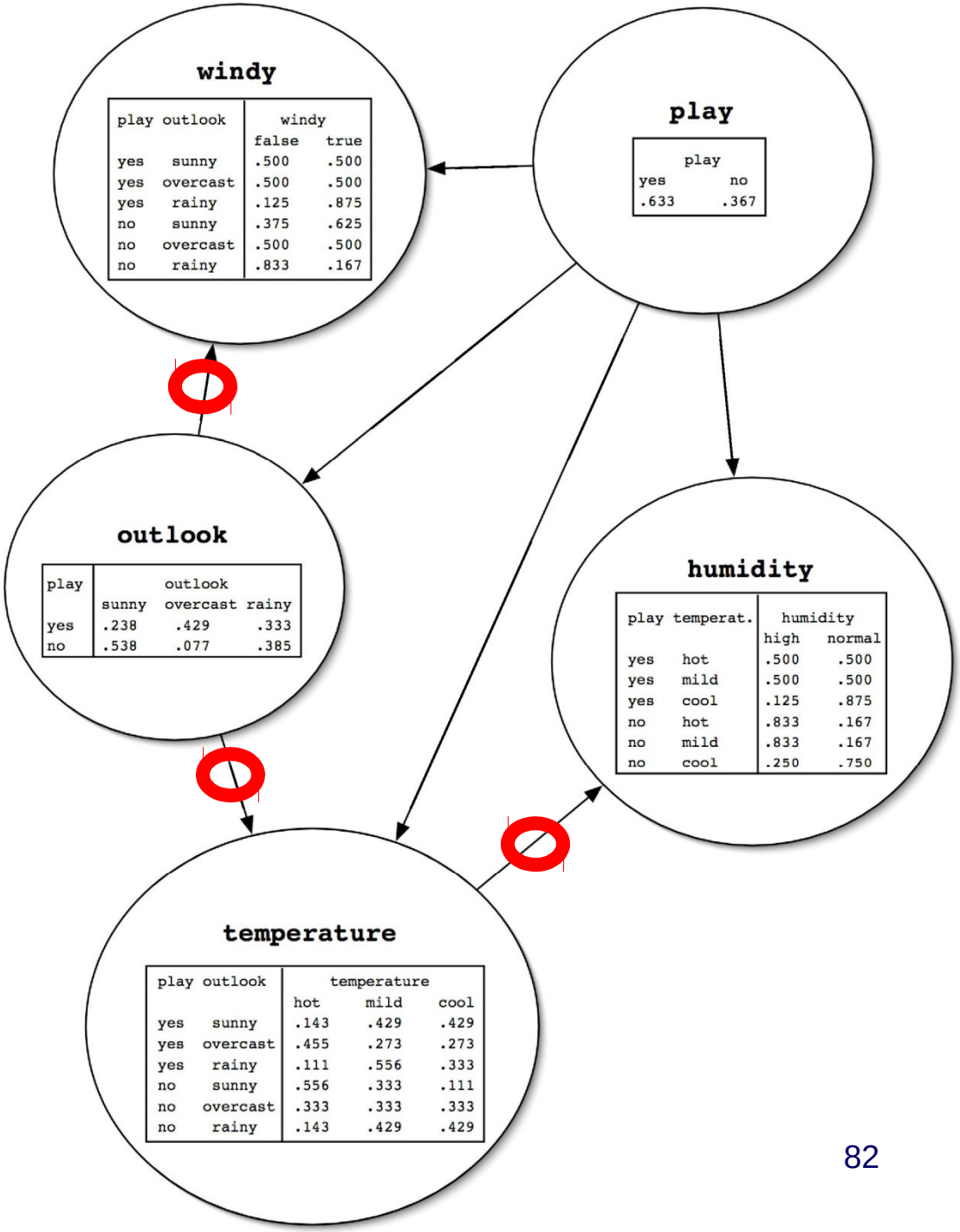
Weather data



Bayesian networks

- Graphical models that can represent any probability distribution
- Graphical representation:
  - ***directed***
  - ***acyclic graph***,
  - one node for each attribute
- Overall probability distribution factorized into conditional distributions
- Graph's nodes hold conditional distributions

Weather data

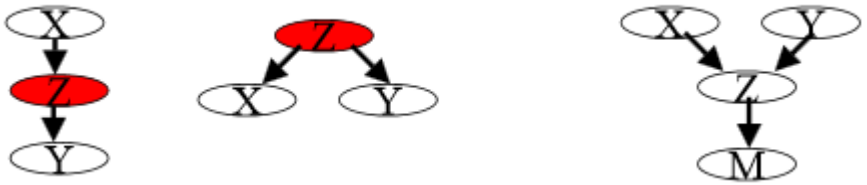


Causality – be cautious!

- Discussions on causality on solid mathematical basis
- Distinguishing causation from correlation?
- **Sometimes**…
  - At least three variables
  - One of the variables: „virtual control” for the relationship between the other two

(d-separation)

- Nodes X and Y are d-separated if on any (undirected) path between X and Y there is some variable Z such that is either
  - Z is in a serial or diverging connection and Z is known, or
  - Z is in a converging connection and neither Z nor any of Z's descendants are known



- Nodes X and Y are called **d-connected** if they are not d-separated (there exists an undirected path between X and Y not d-separated by any node or a set of nodes)
- If nodes X and Y are d-separated by Z, then X and Y are conditionally independent given Z

Prediction: Computing the class probabilities

- Two steps:

  computing a product of probabilities for each class

  1. For each class value
     1. Take all attribute values and class value
     2. Look up corresponding entries in conditional probability distribution tables
     3. Take the product of all probabilities

  normalization

  2. Divide the product for each class by the sum of the products

Why can we do this? (Part I)

- Single assumption: values of a node's parents completely determine probability distribution for current node

- Means that node/attribute is conditionally independent of other ancestors given parents

Why can we do this? (Part II)

- Chain rule from probability theory:

- Because of our assumption from the previous slide:

Learning Bayesian Networks

- Known structure, full observability:
  - Maximum likelihood estimation
- Unknown structure, full observability:
  - Search through model space
- (Known structure, partial observability)
- (Unknown structure, partial observability)

Known structure, full observability

- Conditional probability distributions for parameters,
  maximising the likelihood of the training data
  - Log likelihood:
    decomposes according to the structure
    maximising node by node
- Counting (for nominal distribution),
  Gaussian: mean + variance

Unknown structure, full observability

- Method for ***evaluating the goodness of a given network***
  - Log-likelihood
- S***earching for a good network (structure)***
  - I.e. searching through sets of edges because nodes are fixed

Goodness of given network

- Maximizing (log-)likelihood of training data
- However, addig more edges improves this, as it fits the training data more closely
  - Overfitting has to be avoided!

Avoiding overfitting

- **Cross-validation** or
-
- **Penalty for complexity** of the network
  - AIC (Akaike Information Criterion) measure:

    $$AICscore = - LL + K$$
  - MDL (Minimum Description Length) measure:

    $$MDLscore = -LL + K/2 \log N$$

    LL: log-likelihood (log of probability of data),
    K: number of free parameters,
    N: #instances

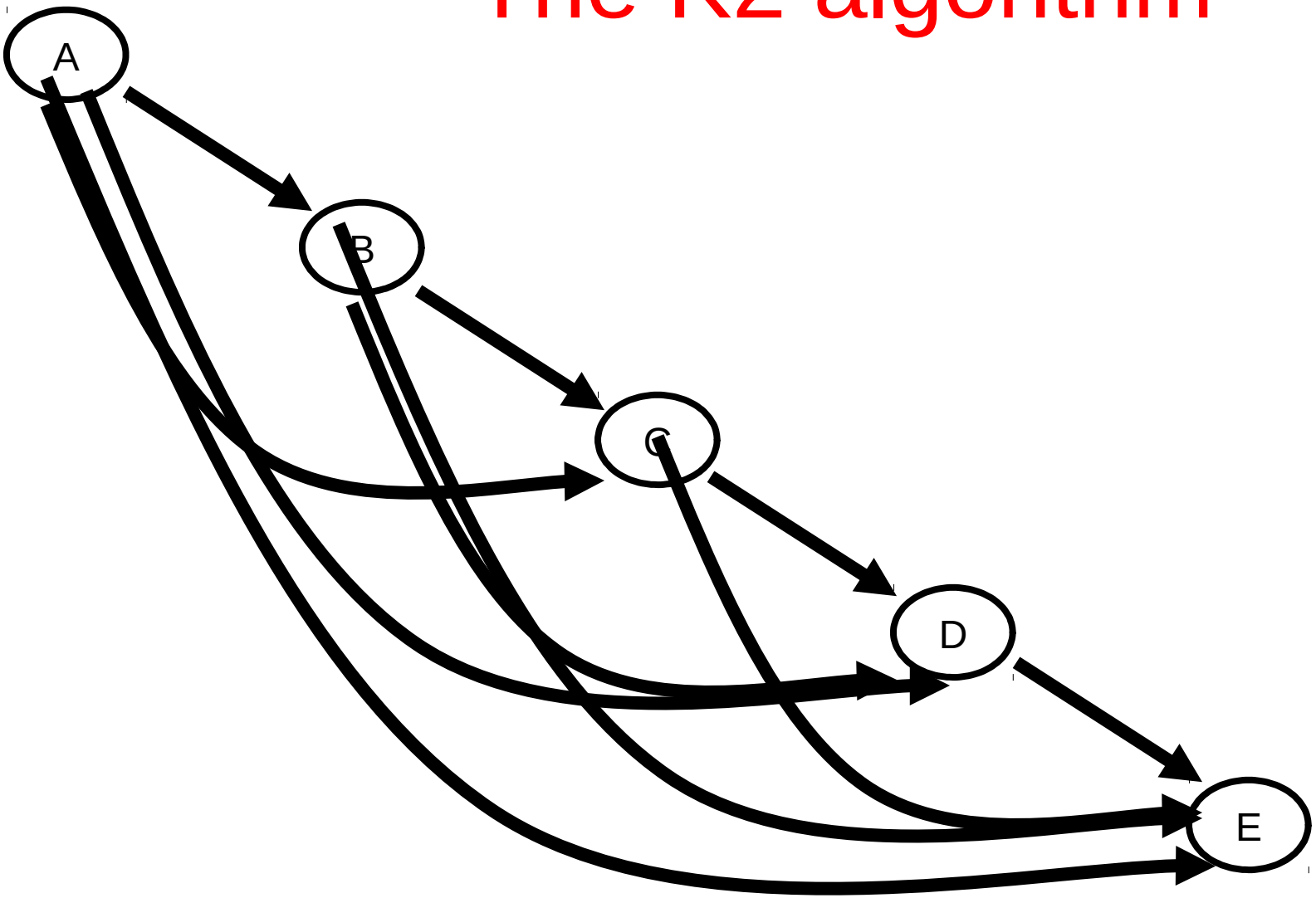
Searching for a good structure

- NP-hard
  - (4 nodes: 453 dags; 10 nodes O(10E18) dags)
- If ordering of nodes given: simpler
  - Learning parent set per node, independently (score, also AIC and MDL decomposable)
  - Can optimize node by adding or removing edges from other nodes

The K2 algorithm

- Starts with given ordering of nodes (attributes)
- Processes each node in turn
- Greedily tries adding edges from previous nodes to current node
- Moves to next node when current node can’t be optimized further
- Result depends on initial order

The K2 algorithm

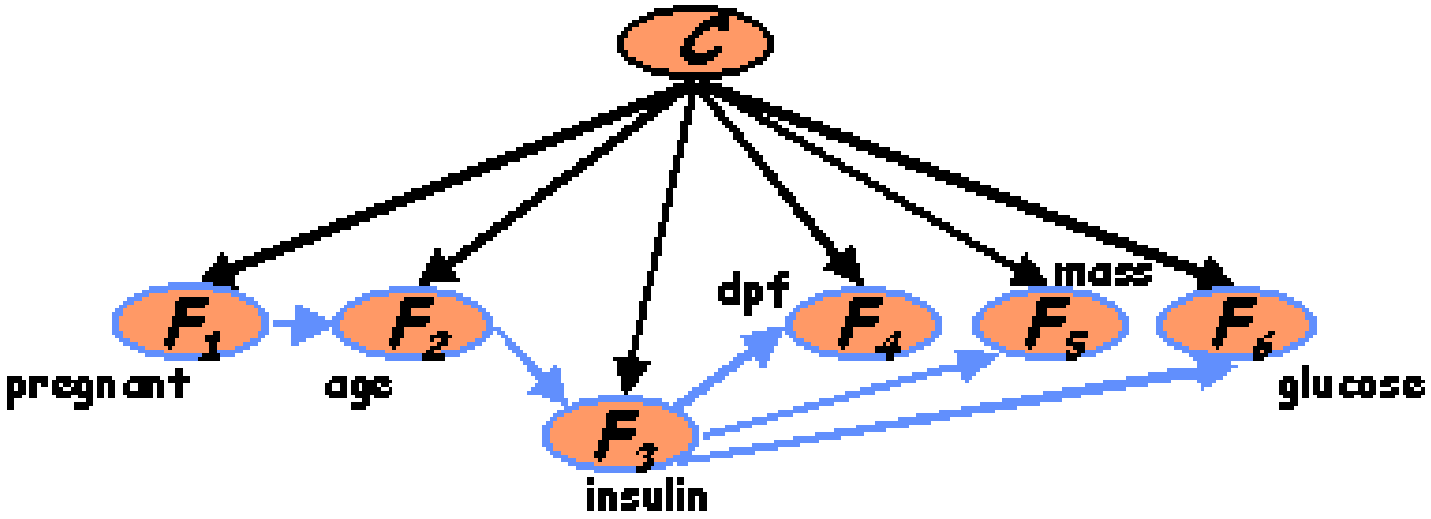


Other algorithms

- Extending K2
  - no preset order of nodes
  - greedily adding or deleting edges between any pair of nodes
  - further step: considering inverting the direction of edges
- TAN (Tree Augmented Naïve Bayes):
  - Starts with naïve Bayes
  - Considers adding second parent to each node (apart from class node)
  - Efficient algorithm exists

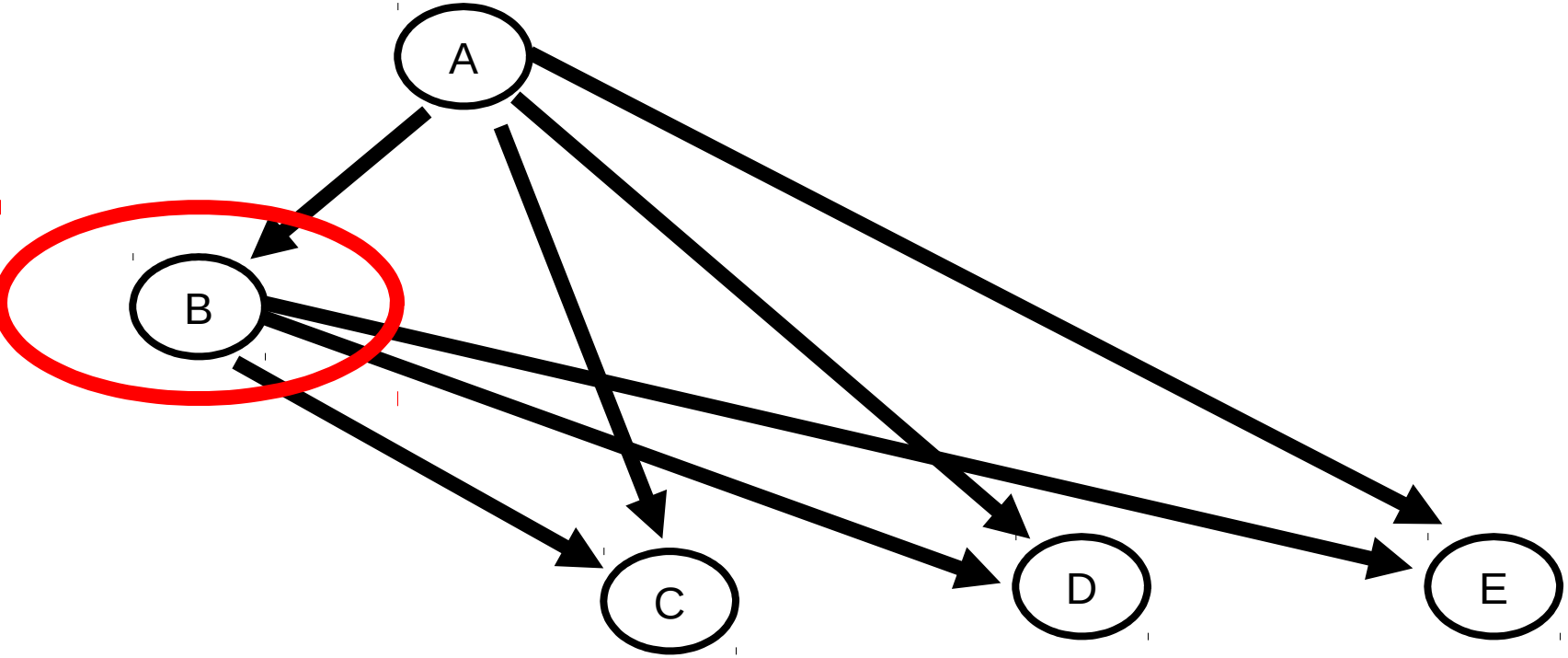
TAN - Example

- ***One-dependence estimator***



TAN – Special case

- ***Superparent one-dependence estimator***

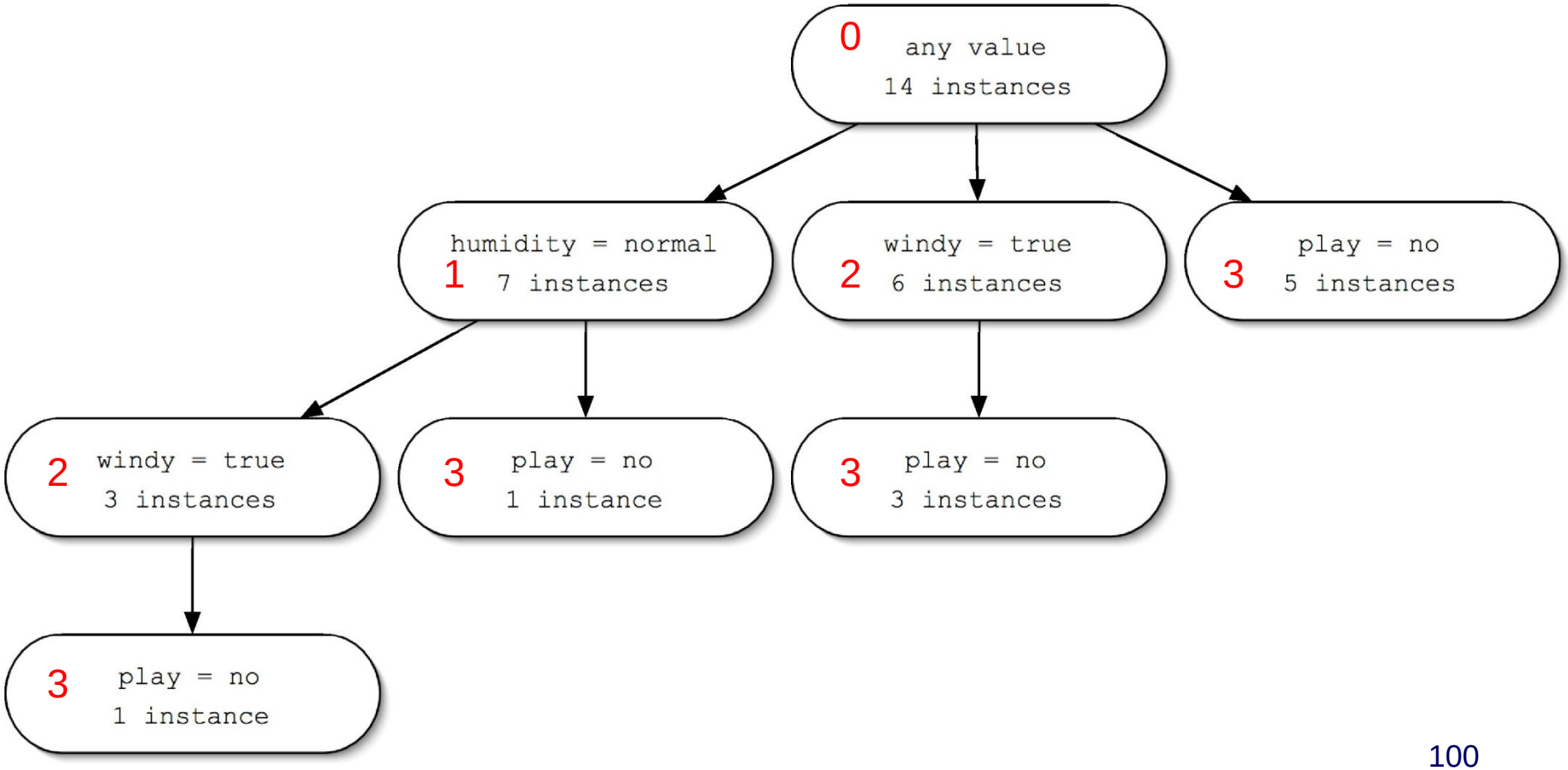


(Data structures for fast learning)

- Learning Bayes nets involves a lot of counting for computing conditional probabilities
- Naïve strategy for storing counts: hash table
  - Runs into memory problems very quickly
- More sophisticated strategy: *all-dimensions (AD) tree*
  - Analogous to *k*D-tree for numeric data
  - Counts in tree
  - Redundancy is eliminated
  - Only makes sense to use it for large datasets

(AD tree example)

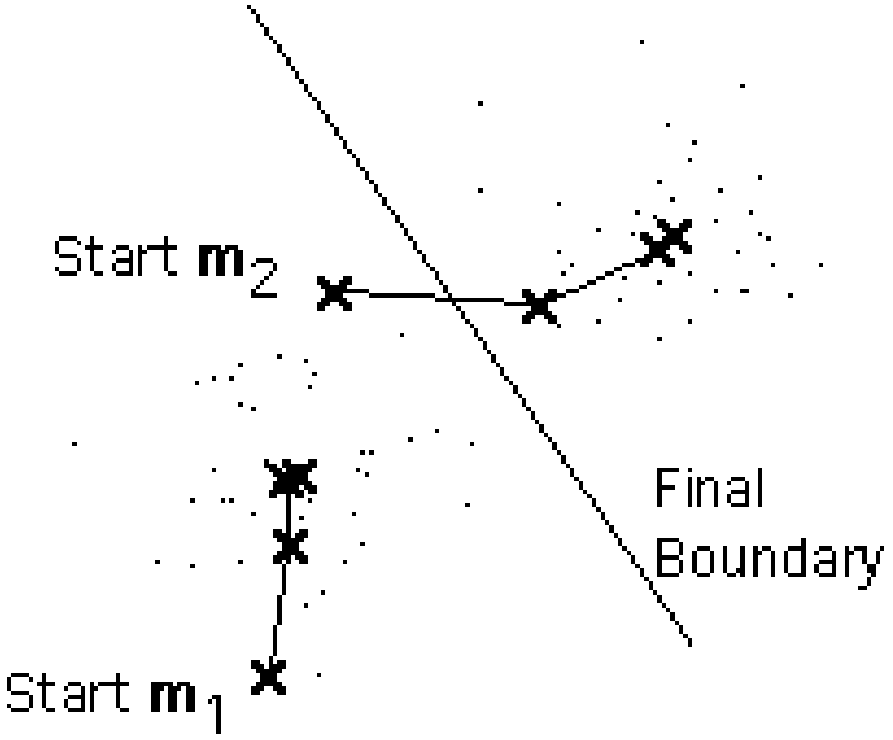
1 humidity	2 windy	3 play	count
high	true	yes	1
high	true	no	2
high	false	yes	2
high	false	no	2
normal	true	yes	2
normal	true	no	1
normal	false	yes	4
normal	false	no	0



Clustering

K-Means

- Example: k = 2



- http://syskall.com/kmeans.js/

K-Means

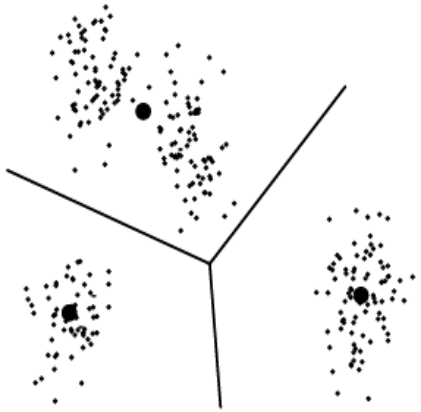
- Number of clusters ?
- Minimizes within-cluster sum-of-squares (inertia). Good for clusters that are
  - convex
  - isotropic (all directions are uniform)
- High dimensional spaces?
  - Curse of dimensionality
  - Principal Component Analysis

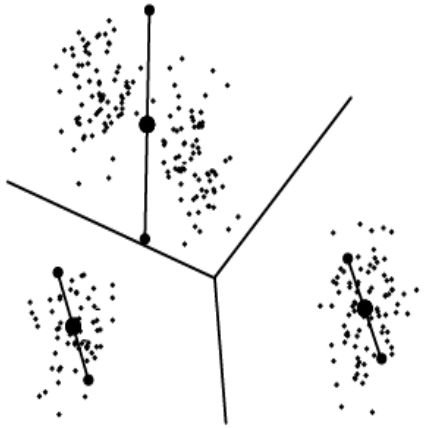
K-Means Number of Clusters/1

- K=1, …n (small)
  - minimizing total squared distance to cluster centers
  - penalizing solutions with many clusters (eg. using an MDL criterion)

K-Means Number of Clusters/1
Recursive k-means

- k-means (k = 2) recursively
- seeds for subclusters can be chosen by seeding along direction of greatest variance in cluster (one standard deviation away in each direction from cluster center of parent cluster)
- use stopping criterion (eg. MDL)
- *X*-means algorithm (using Bayesian Information Criterion instead of MDL)





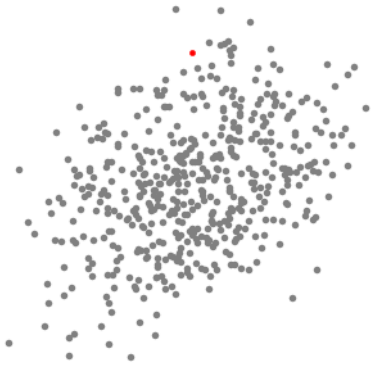
Hierarchical clusters (top-down)

K-Means Scaling

- Scaling
  - Goal: reduce computation time, without affecting quality too much
- *Mini-batch*
  - Randomly sampled subset of the data

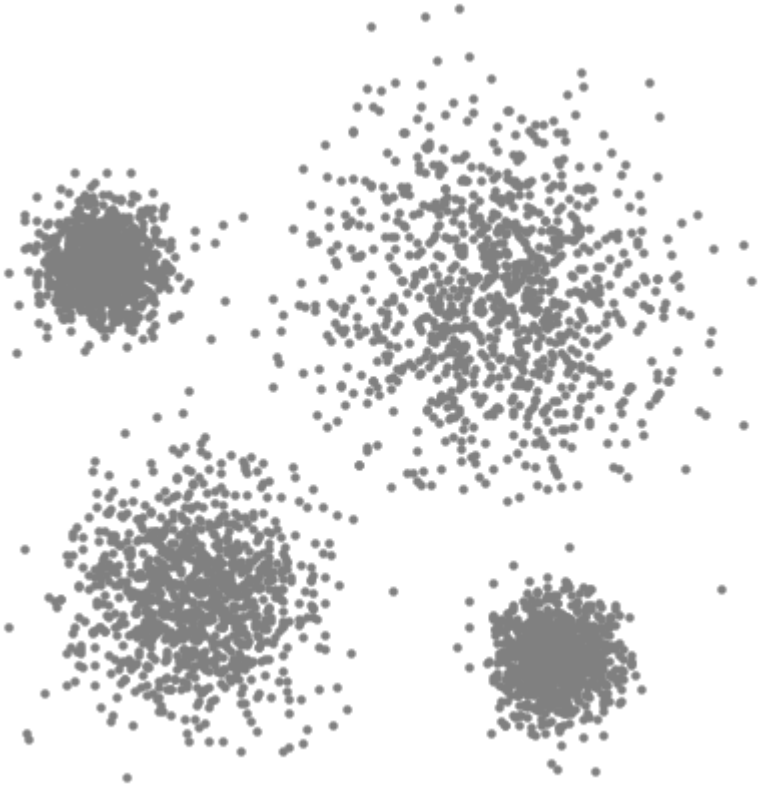
Mean-Shift

- Sliding-window, for finding dense areas of instances
  - Centroid-based: center points of instances within sliding window
  - Shift towards regions of higher density (mean of points in the window)
  - Stop when density can’t be increased



Mean-Shift/2

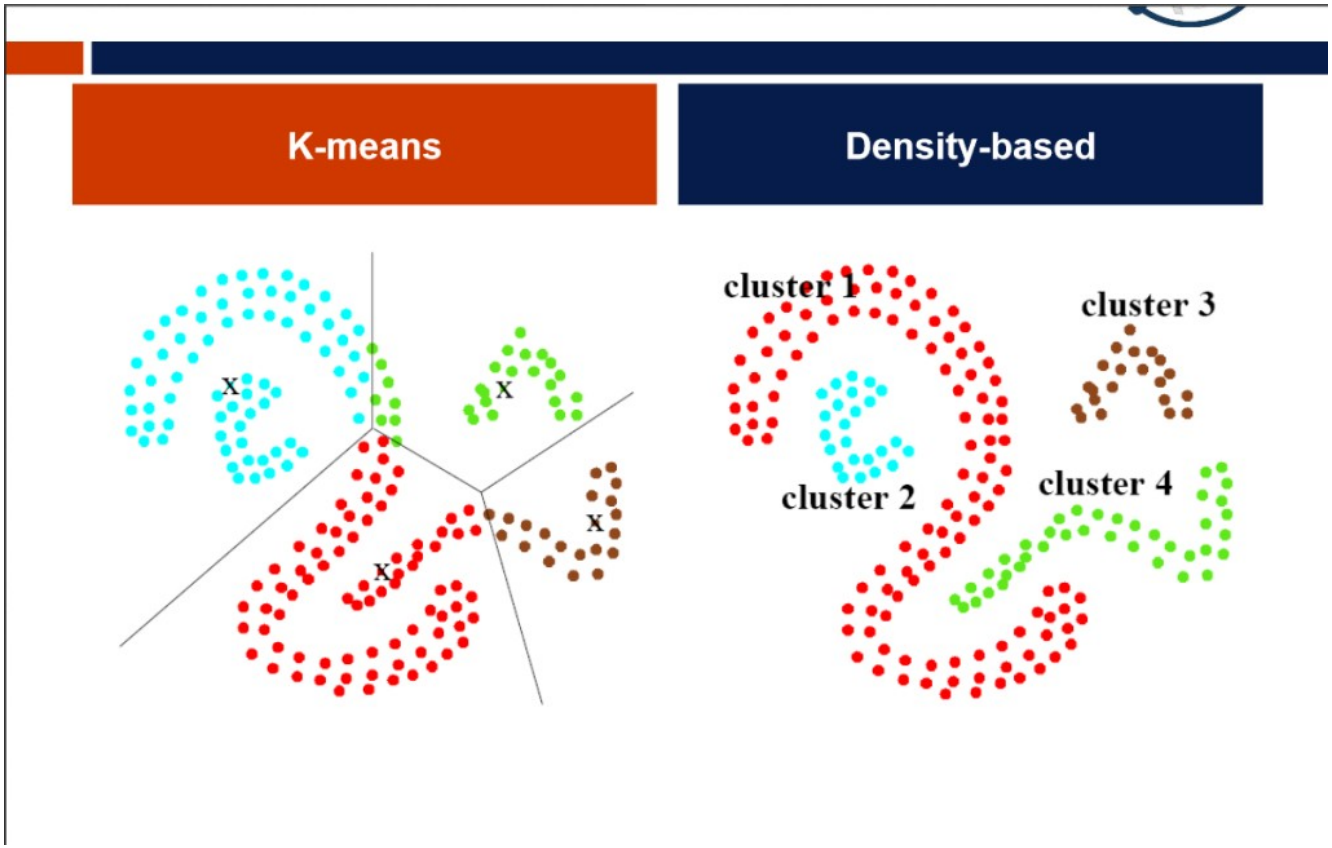
- Many sliding windows
  - All points lie within a window
- Overlapping windows merged
  - Automatically sets number of clusters



Mean-shift/3

- Many clusters
- Uneven cluster size
- Scalability bad

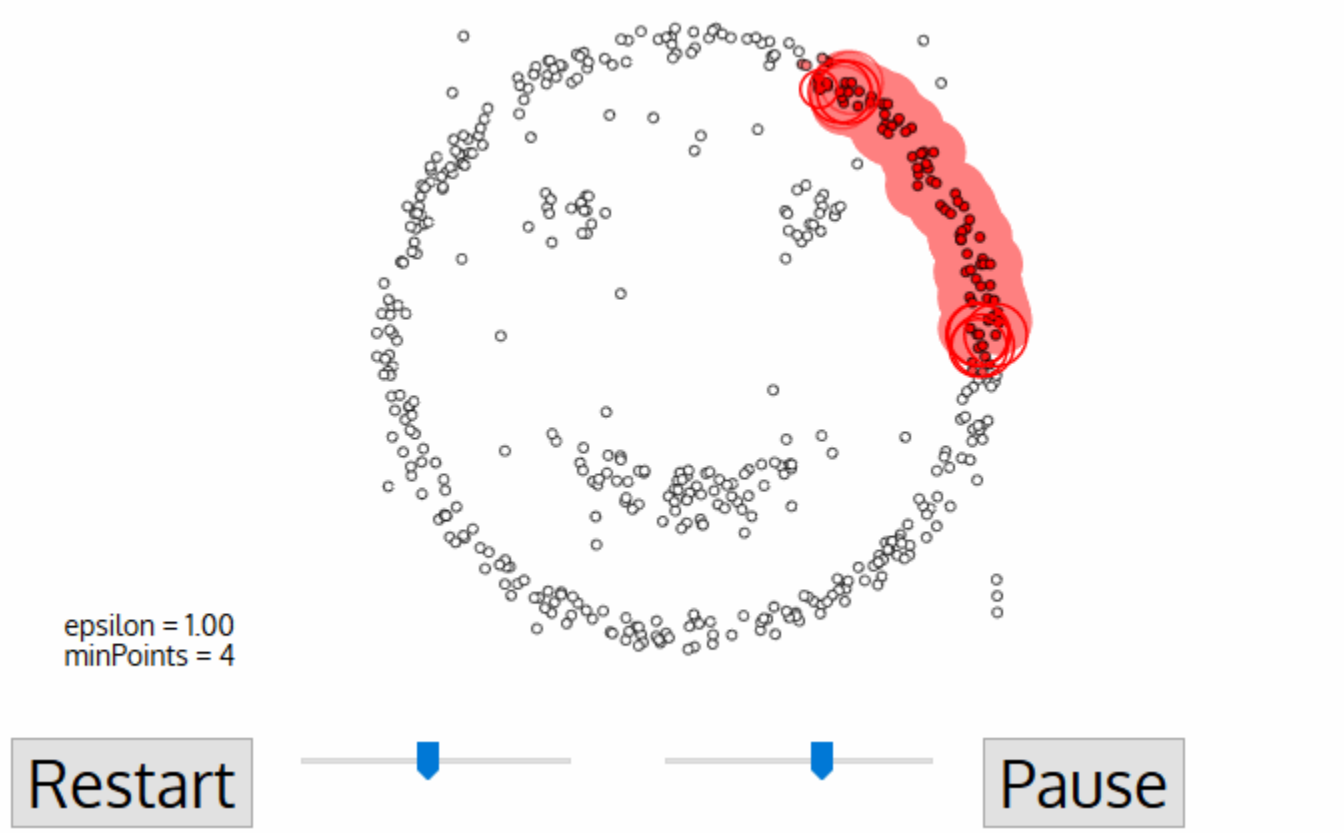
Density-based clustering



DBSCAN: Density-Based Spatial Clustering of Applications with Noise

- Arbitrary starting point (not yet visited)
  - Neighborhood (epsilon)
- Point: marked as visited
- If sufficient number of points (minPoints):
  - clustering process starts
    - For all points in neighborhood repeat step
  - Otherwise: noise
- New unvisited point ….

DBSCAN / 2



DBSCAN / 3

- Arbitrarily sized and shaped clusters
- Number of clusters flexible
- Outliers identified as noise

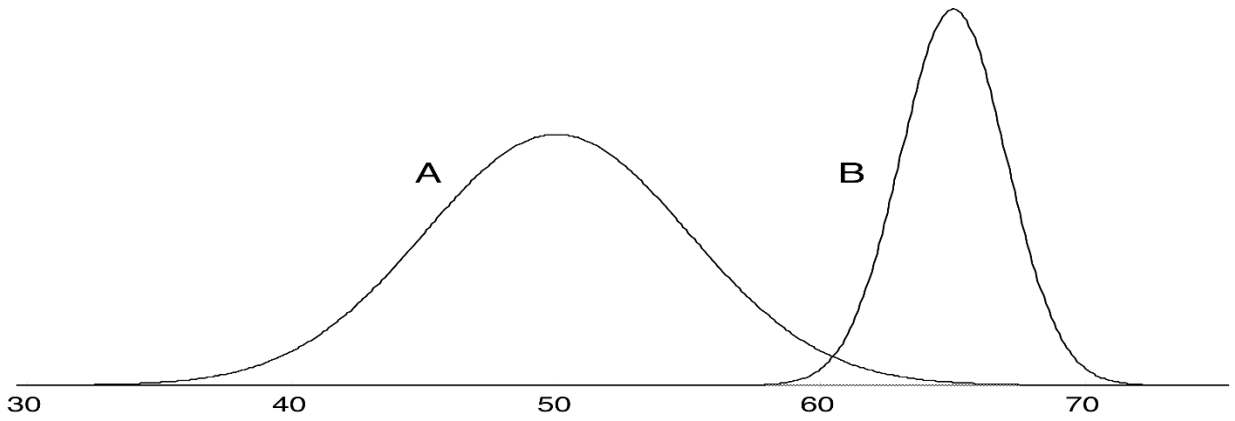
- Clustes of varying density?

Probability-based clustering

- instance belongs to a particular cluster *with a certain probability*
- Probabilistic perspective ⇒
  seek the *most likely* clusters given the data

- Finite mixtures
  - Model data using a *mixture* of distributions
  - One cluster, one distribution
    - governs probabilities of attribute values in that cluster
  - *Finite mixtures* : finite number of clusters
  - Individual distributions are normal (usually)

Gaussian Mixture Model (GMM)



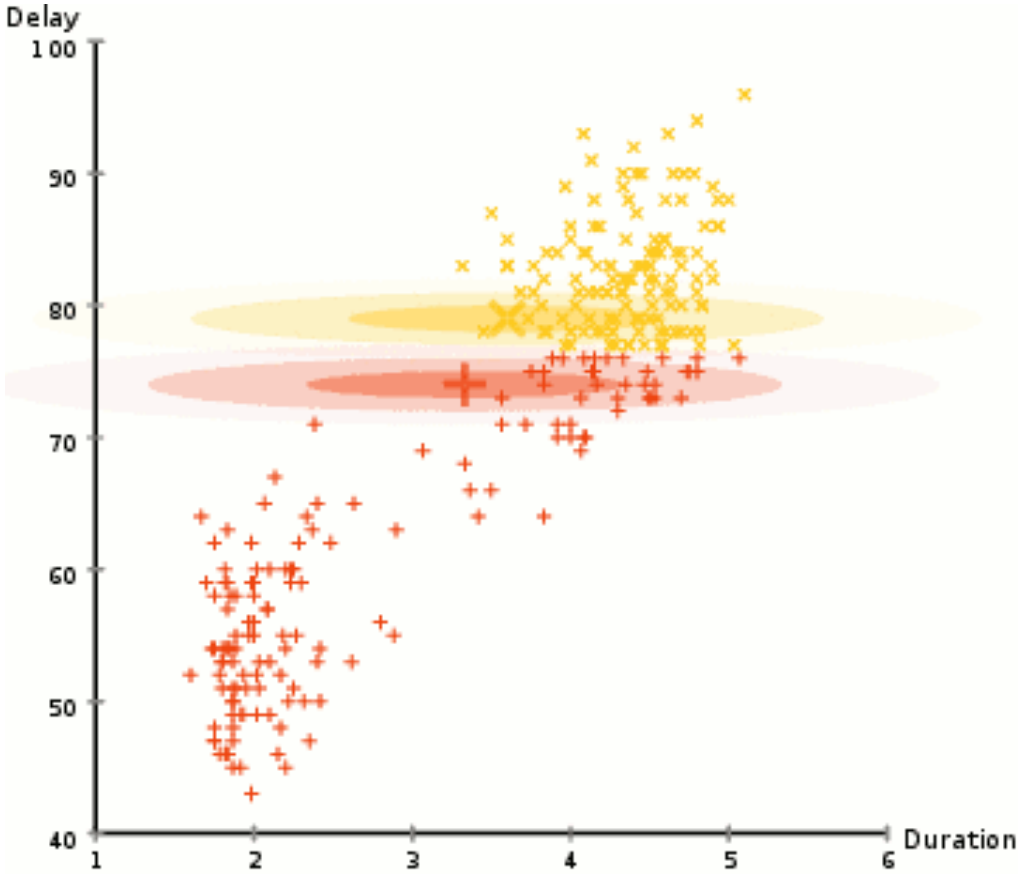
μ_A=50, σ_A =5, p_A=0.6 μ_B=65, σ_B =2, p_B=0.4

(probability that instance x belongs to cluster A….)

Expectation–Maximization (EM) Clustering using GMM

- EM = Expectation-Maximization
  - Generalize *k*-means to probabilistic setting
- Iterative procedure:
  - E “expectation” step:
    Calculate cluster probability for each instance
    (improving mapping of instances to clusters)
  - M “maximization” step:
    Estimate distribution parameters from cluster probabilities
    (adopting, improving clusters)
    probability of training data given the clusters
- Stop when improvement is negligible
  - finds a local maximum of the likelihood

EM Clustering using GMM



Hierarchical clustering

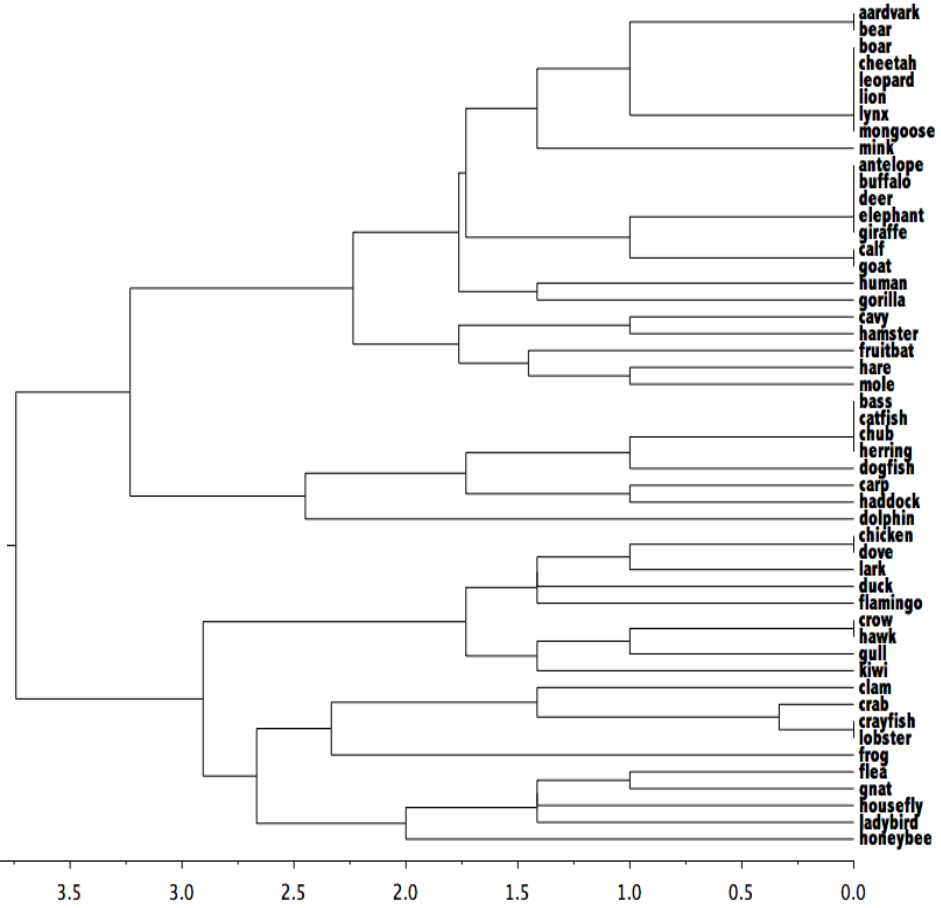
- Top-down:
  - recursively splitting up large clusters into smaller ones
    e.g. recursive k-means
- Bottom-up (hierarchical agglomerative clustering):
  - Each instance as a single cluster
  - Successvely merge (agglomerate) pairs of clusters

Agglomerative clustering (bottom-up)

- Simple algorithm
  - Requires a ***distance/similarity measure***
  - Start by considering each instance to be a cluster
  - Find the two closest clusters and merge them
  - Continue merging until only one cluster is left
  - The record of mergings forms a hierarchical clustering structure

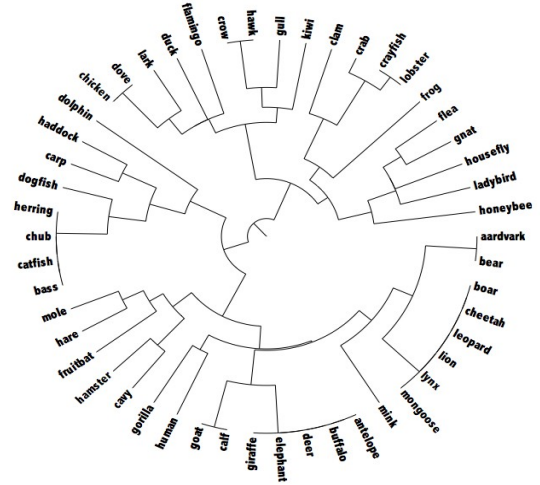
Example hierarchical clustering

- 50 examples of different creatures from the zoo data
  - 1 numeric attribute (number of legs: 0-6 scaled to 0-1)
  - 15 Boolean attributes (has feathers, lays eggs,…)



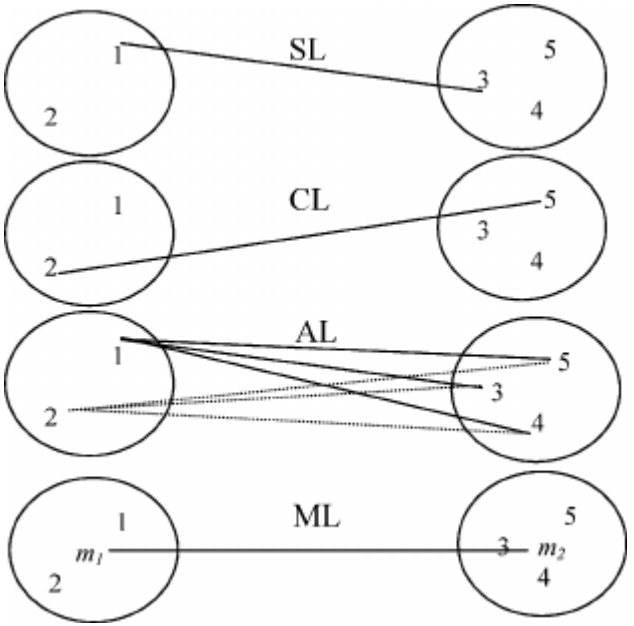
Dendogram

Height of each node in the dendogram can be made proportional to the dissimilarity between its children



Distance measures

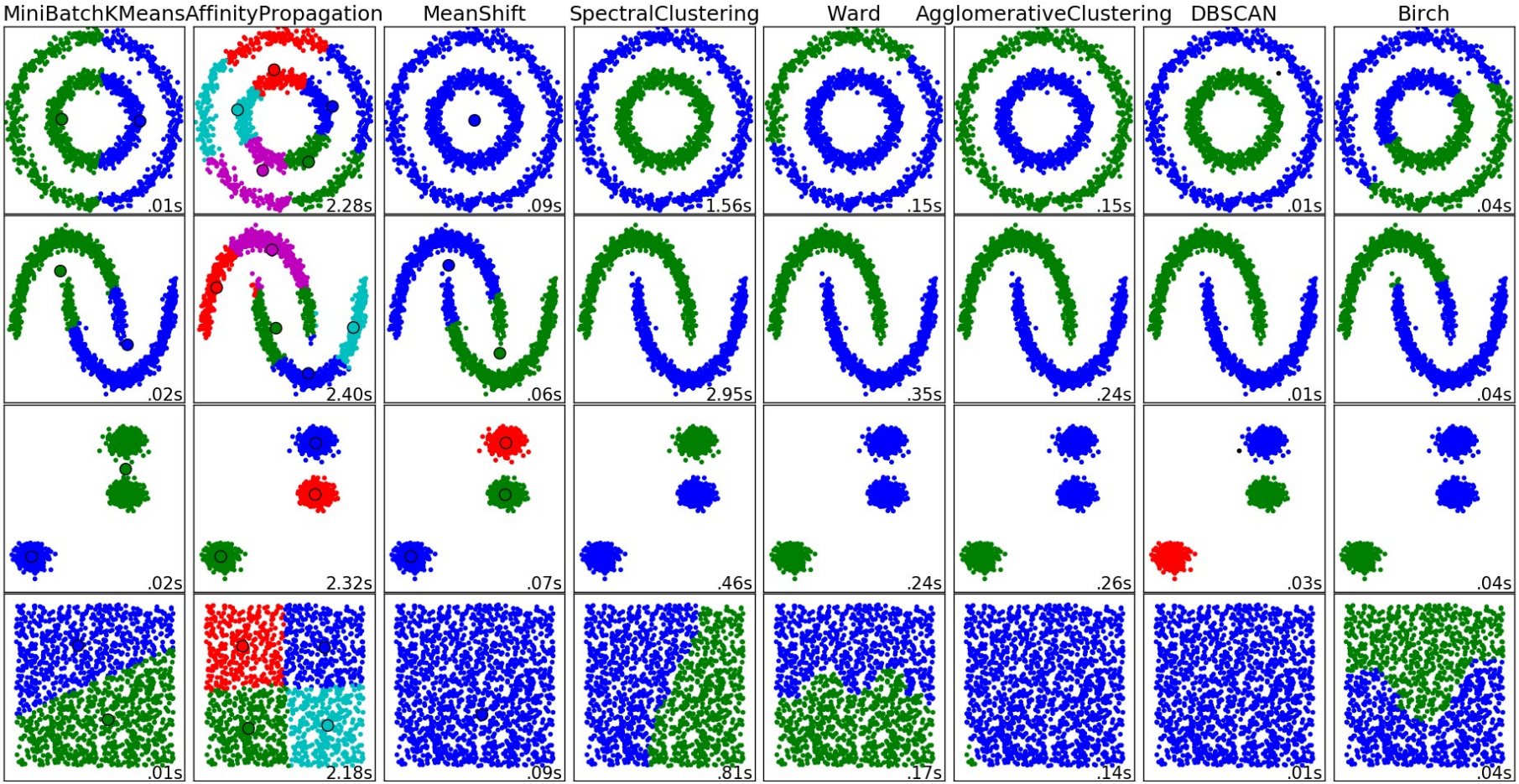
- *Single linkage* (minimum distance)
  - outliers?!
- *Complete linkage* (maximum distance)
  - outliers?!
- *Average linkage*
  - a lot of calculations!
- *Centroid* (mean) *linkage*



For compact and well separated clusters – not much difference

Python, sci-kit learn, clustering methods overview

(http://scikit-learn.org/stable/modules/clustering.html)



Clustering …

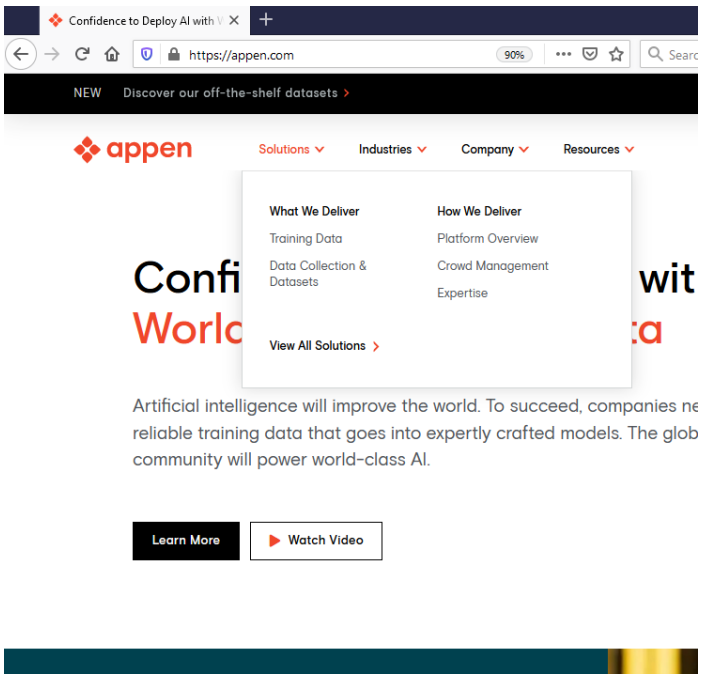
- Iterative clustering
- Quality measures for clustering
- Clustering nodes in a graph

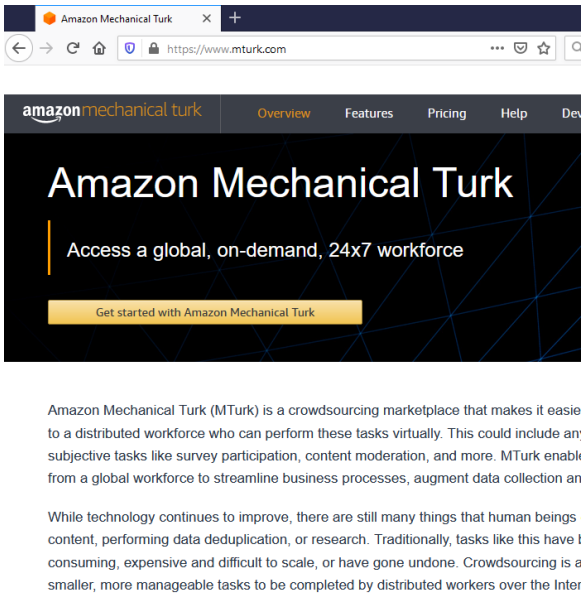
Combining clustering and classification

Semisupervised learning

- *Semisupervised learning*: attempts to use unlabeled data as well as labeled data
  - The aim is to improve classification performance
- Why try to do this? Unlabeled data is often plentiful and labeling data can be expensive
  - Web mining: classifying web pages
  - Text mining: identifying names in text
  - Video mining: classifying people in the news
- Leveraging the large pool of unlabeled examples would be very attractive

Labelling data is expensive! Crowdsourcing (Appen, Amazon Mechanical Turk, Figure Eight,…)





Fun staff with Hungarian connection:The Turk, Farkas Kempelen (https://en.wikipedia.org/wiki/The_Turk)

Clustering + classification

- Classification problem
  - Possible to train classifier directly
- Improving classifier
  - Clustering first
    - Groups similar instances
  - Including cluster value as new attribute

Summary/concepts/questions

- Overfitting, underfitting
- C4.5
  - Handling of numeric attributes
  - Handling of missing values (learning, classification)
  - Pruning
    - pre-pruning, post-pruning (subtree replacement, subtree raising)
    - Error estimation
  - Reduced Error Pruning
- CART
  - GINI Index
  - Cost-complexitiy pruning
- Regularized linear regression
  - Ridge
  - Lasso
- Regression tree learning
- Model tree learning
  - Smoothing

Summary/concepts/questions 2

- Instance based learning
  - performance
  - noise
  - attribute weighing and selection
  - generalisation
- Bayesian networks
  - d-separation
  - Prediction with Bayesian network
  - Learning Bayesian Networks
    - Known structure, full observability
    - Unknown structure, full observability
      - Evaluating the goodness of a given network (maximum likelihood + preventing overfitting (AIC, MDL)
      - Searching through space of possible networks (K2, TAN, Superparent one-dependence estimator)
- Support Vector Machine
  - Maximum margin hyperplane
  - nonlinear case, Kernel function
  - Regularization parameter

Summary/concepts/questions 3

- clustering
  - K-Means
  - Mean-shift
  - DBSCAN (density-based)
  - EM Clustering using GMM (probabilistic)
  - Hierarchical
    - Top-down
    - Bottom-up (agglomerative)
- Semisupervised learning