Data Mining

Practical Machine Learning Tools and Techniques

Slides for Chapter 8, Data transformations

of *Data Mining* by I. H. Witten, E. Frank,
M. A. Hall, and C. J. Pal

Data transformations

- Attribute selection: Scheme-independent and scheme-specific
- Attribute discretization: Unsupervised, supervised, error- vs entropy-based, converse of discretization
- Projections: principal component analysis (PCA), random projections, partial least-squares, independent component analysis (ICA), linear discriminant analysis (LDA), text, time series
- Sampling: Reservoir sampling
- Dirty data: Data cleansing, robust regression, anomaly detection
- Transforming multiple classes to binary ones
  error-correcting codes, ensembles of nested dichotomies
- Calibrating class probabilities

Just apply a learner? NO!

- Scheme/parameter selection
  - Treat selection process as part of the learning process to avoid optimistic performance estimates
- Modifying the input:
  - Data engineering to make learning possible or easier
- Modifying the output
  - Converting multi-class problems into two-class ones
  - Re-calibrating probability estimates

Attribute selection

- Attribute selection is often important in practice
- For example, adding a random (i.e., irrelevant) attribute can significantly degrade C4.5's performance
  - Problem: C4.5's built-in attribute selection is based on smaller and smaller amounts of data
- Instance-based learning is particularly susceptible to irrelevant attributes
  - Number of training instances required increases exponentially with number of irrelevant attributes
- Exception: naïve Bayes can cope well with irrelevant attributes
- Note that relevant attributes can also be harmful if they mislead the learning algorithm

Scheme-independent attribute selection

- *Filter* approach to attribute selection: assess attributes based on general characteristics of the data
- In this approach, the attributes are selected in a manner that is independent of the target machine learning scheme
- One method: find smallest subset of attributes that separates data
- Another method: use a fast learning scheme that is different from the target learning scheme to find relevant attributes
  - E.g., use attributes selected by C4.5 and 1R, or coefficients of linear model, possibly applied recursively (*recursive feature elimination*)

Scheme-independent attribute selection

- Attribute weighting techniques based on instance-based learning can also be used for filtering
  - Original approach for doing this cannot find redundant attributes (but a fix has been suggested)
- Correlation-based Feature Selection (CFS):
  - Correlation between attributes measured by symmetric uncertainty:

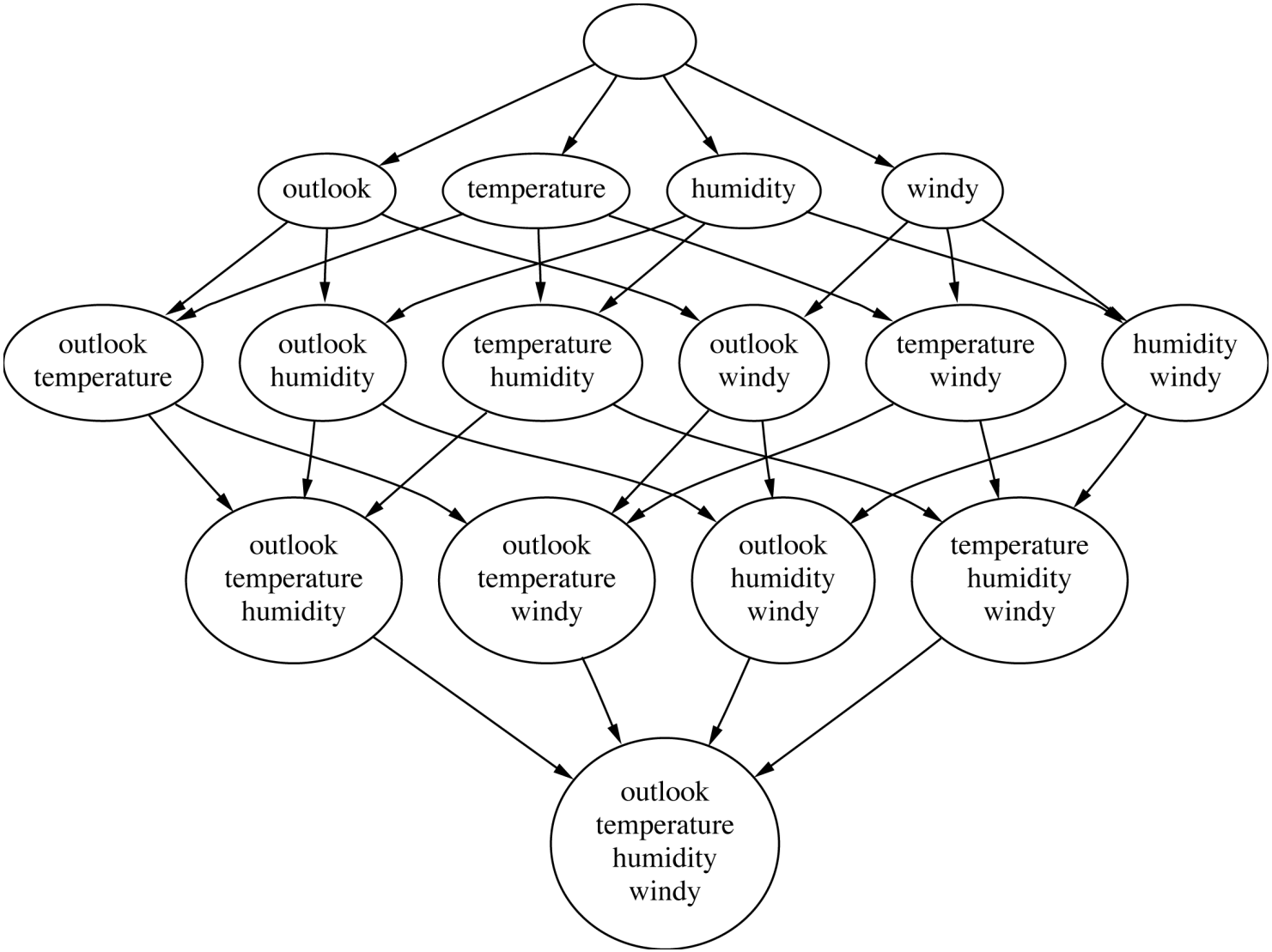
$$U(A,B)=2\frac{H(A)+H(B)-H(A,B)}{H(A)+H(B)}\in[0,1]$$

  - Goodness of subset of attributes measured by

$$\sum_j U(A_j,C)/\sqrt{(\sum_i\sum_j U(A_i,A_j))}$$

  breaking ties in favour of smaller subsets

Attribute subsets for weather data



Searching the attribute space

- Number of attribute subsets is exponential in the number of attributes
- Common greedy approaches:
  - forward selection
  - backward elimination
- More sophisticated strategies:
  - Bidirectional search
  - Best-first search: can find optimum solution
  - Beam search: approximation to best-first search
  - Genetic algorithms

Scheme-specific selection

- *Wrapper* approach to attribute selection: attributes are selected with target scheme in the loop
- Implement "wrapper" around learning scheme

  Evaluation criterion: cross-validation performance
- Time consuming in general
  - greedy approach, k attributes ⇒ $k^2 \times$ time $(k+k-1+k-2+\ldots)$
  - prior ranking of attributes ⇒ linear in k
- Can use significance test to stop cross-validation for a subset early if it is unlikely to "win" (*race search*)

  Can be used with forward, backward selection, prior ranking, or special-purpose *schemata search*
- Scheme-specific attribute selection is essential for learning decision tables
- Efficient for decision tables and Naïve Bayes

Attribute discretization

- Discretization can be useful even if a learning algorithm can be run on numeric attributes directly
- Avoids normality assumption in Naïve Bayes and clustering
- Examples of discretization we have already encountered:
  - 1R: uses simple discretization scheme
  - C4.5 performs local discretization
- Global discretization can be advantageous because it is based on more data
- Apply learner to
  - k -valued discretized attribute or to
  - k – 1 binary attributes that code the cut points
- The latter approach often works better when learning decision trees or rule sets

Discretization: unsupervised

- *Unsupervised discretization*: determine intervals without knowing class labels

  When clustering, the only possible way!

- Two well-known strategies:
  - *Equal-interval binning*
  - *Equal-frequency binning*
    (also called *histogram equalization*)
- Unsupervised discretization is normally inferior to supervised schemes when applied in classification tasks

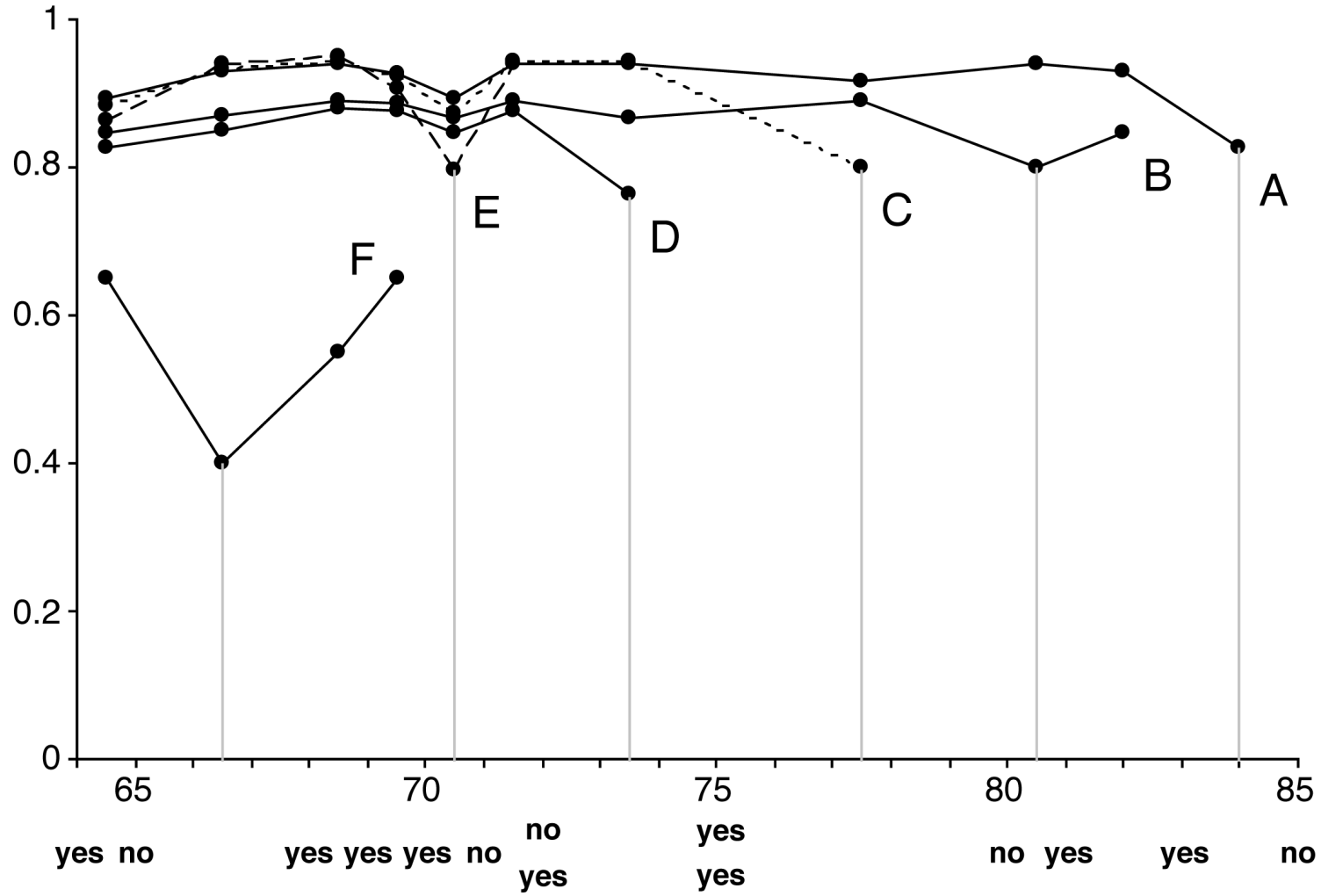
  But equal-frequency binning works well with naïve Bayes if the number of intervals is set to the square root of the size of dataset (*proportional k-interval discretization)*

Discretization: supervised

- Classic approach to *supervised* discretization is entropy-based
- This method builds a decision tree with pre-pruning on the attribute being discretized
  - Uses entropy as splitting criterion
  - Uses the minimum description length principle as the stopping criterion for pre-pruning
- Works well: still the state of the art
- To apply the minimum description length principle, the "theory" is
  - the splitting point (can be coded in $\log_2[N - 1]$ bits)
  - plus class distribution in each subset (a more involved expression)
- Description length is the number of bits needed for coding both the splitting point and the class distributions
- Compare description lengths before/after adding split

Example: temperature attribute

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



Formula for MDL stopping criterion

- Can be formulated in terms of the information gain
- Assume we have *N* instances
  - Original set: *k* classes, entropy *E*
  - First subset: k_1 classes, entropy E_1
  - Second subset: k_2 classes, entropy E_2

$$gain > \frac{\log_2(N-1)}{N} + \frac{\log_2(3^k-2) - kE + k_1E_1 + k_2E_2}{N}$$

- If the information gain is greater than the expression on the right, we continue splitting
- Results in *no* discretization intervals for the temperature attribute in the weather data

Supervised discretization: other methods

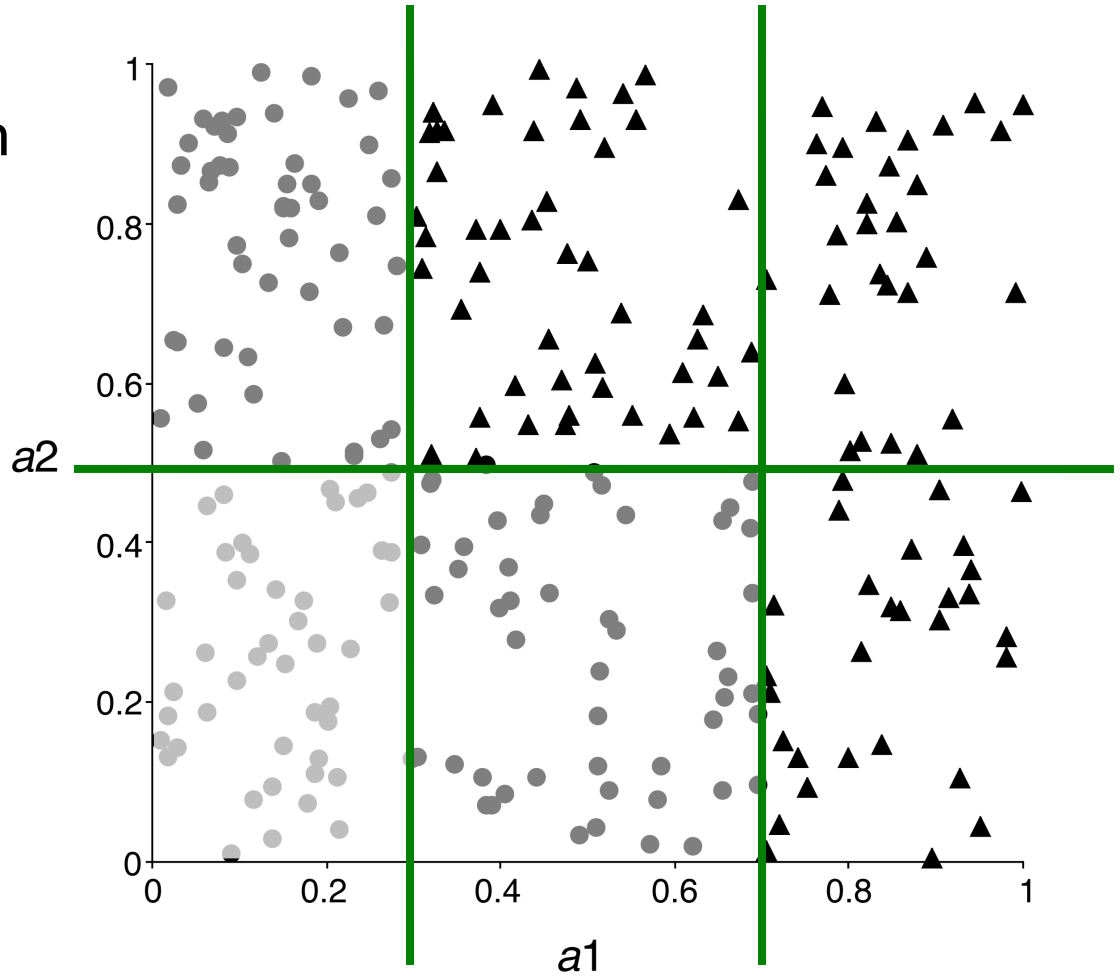
- Can replace top-down procedure by bottom-up method
- This bottom-up method has been applied in conjunction with the chi-squared test
  - Continue to merge intervals until they become significantly different
- Can use dynamic programming to find optimum *k*-way split for given additive criterion
  - Requires time quadratic in the number of instances
  - But can be done in linear time if error rate is used instead of entropy
  - However, using error rate is generally not a good idea when discretizing an attribute as we will see

Error-based vs. entropy-based

- Question:
  could the best discretization ever have two adjacent intervals with the same class?
- Wrong answer: No. For if so,
  - Collapse the two
  - Free up an interval
  - Use it somewhere else
  - (This is what error-based discretization will do)
- Right answer: Surprisingly, yes.
  - (and entropy-based discretization can do it)

Error-based vs. entropy-based

A 2-class,
2-attribute problem



Entropy-based discretization can detect change of class *distribution*

The converse of discretization

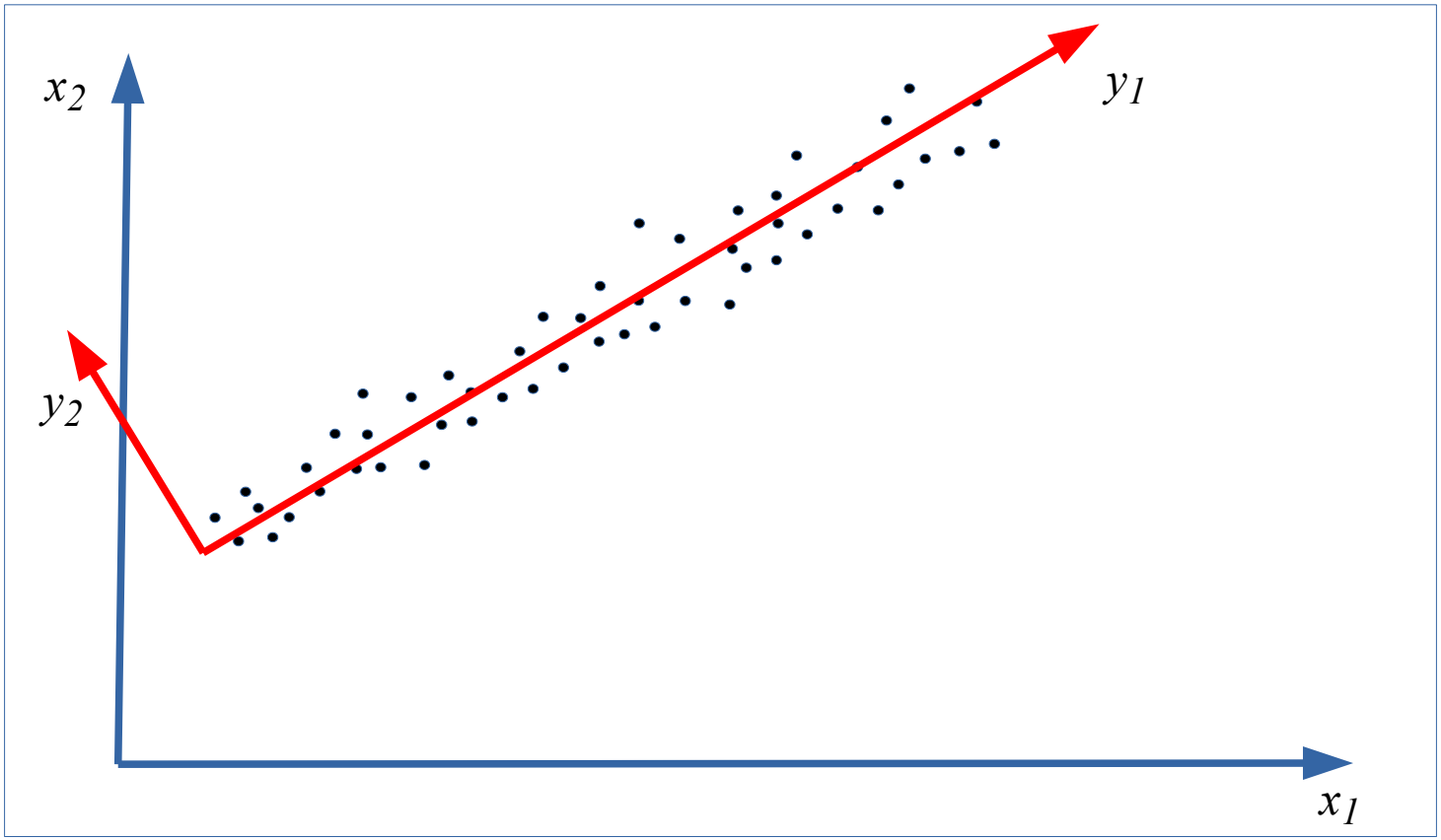
- Make nominal values into “numeric” ones
- Can use indicator attributes (as in IB1)
- Makes no use of potential ordering information if “nominal” attribute is actually ordinal
- Alternative: code an ordered nominal attribute into binary ones as in M5’
  - Inequalities are explicitly represented as binary attributes, e.g., *temperature* < *hot* in the *weather* data
  - Can be used for any ordered attribute
  - Better than coding ordering into an integer (which implies a metric)
- In general: can code binary partition of a set of attribute values as a binary attribute

Projections

- Simple transformations can often make a large difference in performance
- Example transformations (not necessarily for performance improvement):
  - Difference of two date attributes
  - Ratio of two numeric (ratio-scale) attributes
  - Concatenating the values of nominal attributes
  - Encoding cluster membership
  - Adding noise to data
  - Removing data randomly or selectively
  - Obfuscating the data

Principal component analysis

- Unsupervised method for identifying the important directions in a dataset
- We can then rotate the data into the (reduced) coordinate system that is given by those directions

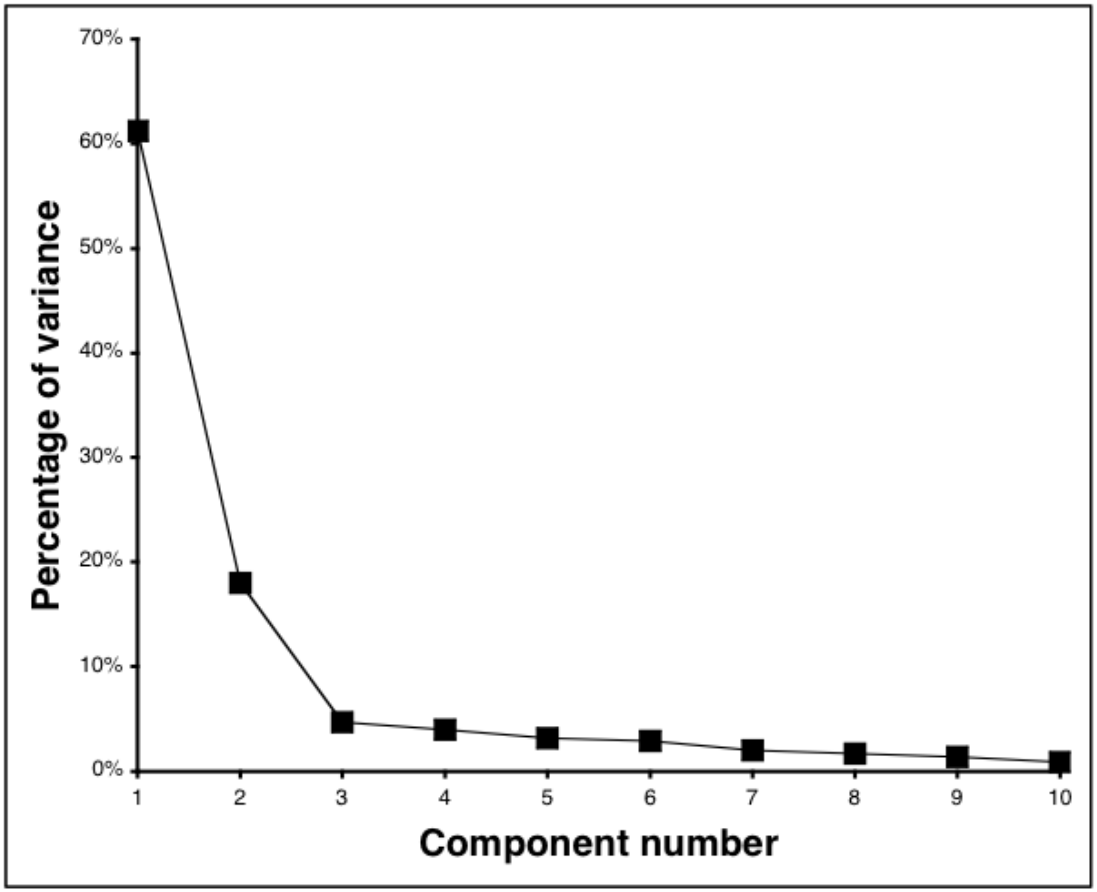


Principal component analysis (2)

- PCA is a method for *dimensionality reduction*
- Algorithm:
  1. Find direction (axis) of greatest variance
  2. Find direction of greatest variance that is perpendicular to previous direction and repeat
- Implementation: find eigenvectors of the covariance matrix of the data
  - Eigenvectors (sorted by eigenvalues) are the directions
  - Mathematical details are covered in chapter on “Probabilistic methods”

Example: 10-dimensional data

Axis	Variance	Cumulative
1	61.2%	61.2%
2	18.0%	79.2%
3	4.7%	83.9%
4	4.0%	87.9%
5	3.2%	91.1%
6	2.9%	94.0%
7	2.0%	96.0%
8	1.7%	97.7%
9	1.4%	99.1%
10	0.9%	100.0%



- Data is normally standardized or mean-centered for PCA
- Can also apply this recursively in a tree learner

Random projections

- PCA is nice but expensive: cubic in number of attributes
- Alternative: use random directions instead of principal components
- Surprising: random projections preserve distance relationships quite well (on average)
  - Can use them to apply *k*D-trees to high-dimensional data
  - Can improve stability by using ensemble of models based on different projections
- Different methods for generating random projection matrices have been proposed

Partial least-squares regression

- PCA is often used as a pre-processing step before applying a learning algorithm
  - When linear regression is applied, the resulting model is known as *principal components regression*
  - Output can be re-expressed in terms of the original attributes because PCA yields a linear transformation
- PCA is unsupervised and ignores the target attribute
- The *partial least-squares* (PLS) transformation differs from PCA in that it takes the class attribute into account
  - Finds directions that have high variance and are strongly correlated with the class
- Applying PLS as a pre-processing step for linear regression yields *partial least-squares regression*

An algorithm for PLS

1. Start with standardized input attributes
2. Attribute coefficients of the first PLS direction:
   - Compute the dot product between each attribute vector and the class vector in turn, this yields the coefficients
3. Coefficients for next PLS direction:
   - Replace attribute value by difference (residual) between the attribute's value and the prediction of that attribute from a simple regression based on the previous PLS direction
   - Compute the dot product between each attribute's residual vector and the class vector in turn, this yields the coefficients
4. Repeat from 3

PLS Example (2 attributes only)

Table 1.5 CPU Performance Data

	Main Memory (Kb)				Channels		
	Cycle Time (ns)	*Min*	*Max*	Cache (KB)	*Min*	*Max*	Performance
	MYCT	*MMIN*	*MMAX*	*CACH*	*CHMIN*	*CHMAX*	*PRP*
1	125	256	6000	256	16	128	198
2	29	8000	32,000	32	8	32	269
3	29	8000	32,000	32	8	32	220
4	29	8000	32,000	32	8	32	172
5	29	8000	16,000	32	8	16	132
...							
207	125	2000	8000	0	2	14	52
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

Table 7.1 First Five Instances from the CPU Performance Data

	(a)			(b)	(c)		
	chmin	**chmax**	**prp**	**pls 1**	**chmin**	**chmax**	**prp**
1	1.7889	1.7678	198	39.825	0.0436	0.0008	198
2	–0.4472	–0.3536	269	–7.925	–0.0999	–0.0019	269
3	–0.4472	–0.3536	220	–7.925	–0.0999	–0.0019	220
4	–0.4472	–0.3536	172	–7.925	–0.0999	–0.0019	172
5	–0.4472	–0.7071	132	–16.05	0.2562	0.005	132

(a) original values, (b) first partial least-squares direction, and (c) residuals from the first direction.

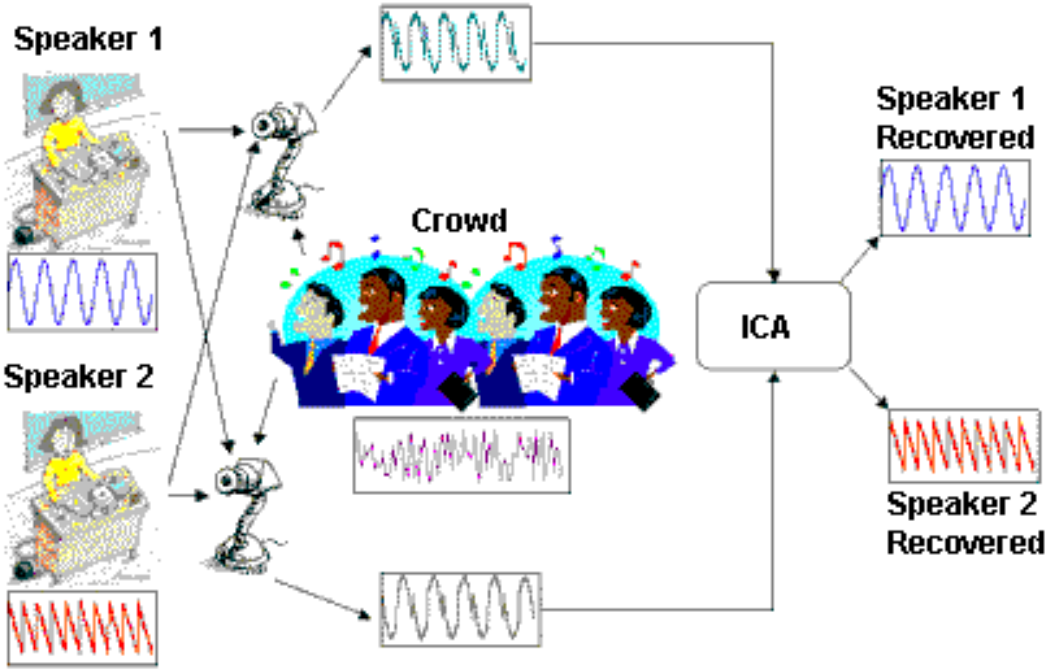
PLS Example (cont'd)

- PRP*CHMIN= -0.4472, PRP*CHMAX=22.981
  PLS1= -0.4472 CHMIN + 22.981 CHMAX
- Univariate Regression:
  CHMIN= 0.0438 PLS1
  CHMAX=0.0444 PLS1
- PLS2= -23.6002 CHMIN - 0.4593 CHMAX
- All attribute residuals are zero now

- Use PLS directions as input for linear regression:
  partial least squares regression model
- If all directions are used, result is the same as with original attributes

Independent component analysis (ICA)

- PCA finds a coordinate system for a feature space that captures the covariance of the data
- In contrast, *ICA* seeks a projection that decomposes the data into sources that are statistically independent
- Consider the "cocktail party problem," where people hear music and the voices of other people: the goal is to un-mix these signals
- ICA finds a linear projection of the mixed signal that gives the most statistically independent set of transformed variables



Correlation vs. statistical independence

- PCA is sometimes thought of as a method that seeks to transform correlated variables into linearly uncorrelated ones
- Important: correlation and statistical independence are two different criteria
  - Uncorrelated variables have correlation coefficients equal to zero – entries in a covariance matrix
  - Two variables *A* and *B* are considered independent when their joint probability is equal to the product of their *marginal* probabilities:

$$P(A,B) = P(A)P(B)$$

ICA and Mutual Information

- Mutual information (MI) measures the amount of info one can obtain from one random variable given another one
- It can be used as an alternative criterion for finding a projection of data
  - We can aim to minimize the mutual information between the dimensions of the data in a linearly transformed space
- Assume a model $\boldsymbol{s} = A\boldsymbol{x}$, where A is an orthogonal matrix, $\boldsymbol{x}$ is the input data and $\boldsymbol{s}$ is its decomposition into its sources
- Fact: minimizing the MI between the dimensions of s corresponds to finding a transformation matrix A so that
  a) the estimated probability distribution of the sources $p(s)$ is as far from Gaussian as possible and
  b) the estimates s are constrained to be uncorrelated

ICA & FastICA

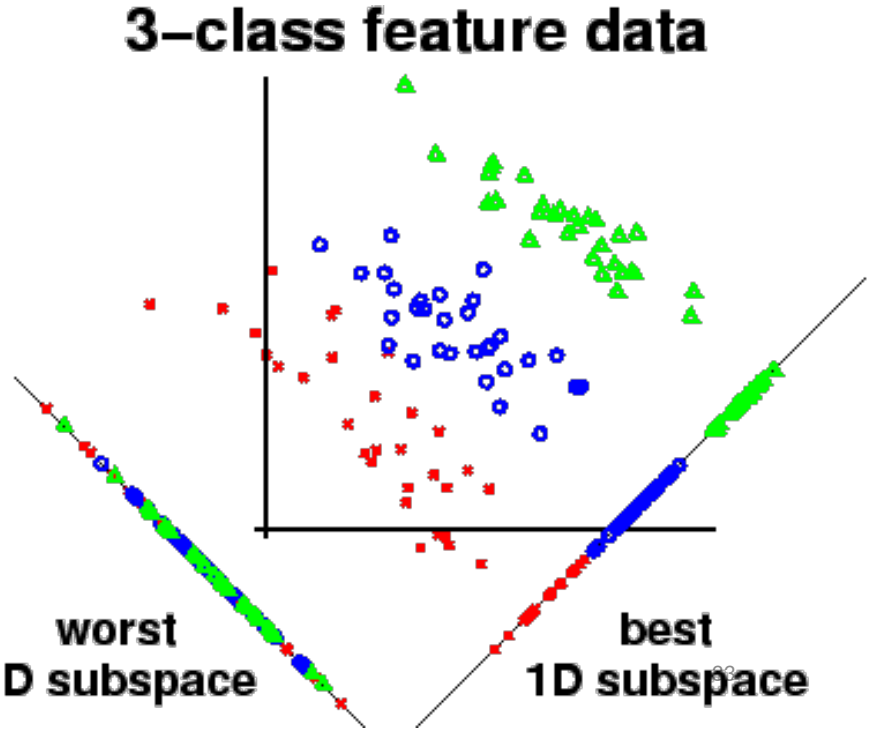
- A popular technique for performing independent component analysis is known as *fast ICA*
- Uses a quantity known as the *negentropy* $J(s) = H(z) - H(s)$, where
  - z is a Gaussian random variable with the same covariance matrix as s and
  - $H(.)$ is the "differential entropy," defined as

$$H(\mathbf{x}) = -\int p(\mathbf{x}) \log p(\mathbf{x}) d\mathbf{x}$$

- Negentropy measures the departure of s's distribution from the Gaussian distribution
- Fast ICA uses simple approximations to the negentropy allowing learning to be performed more quickly.

Linear discriminant analysis

- Linear discriminant analysis is another way of finding a linear transformation that reduces the number of dimensions
- Unlike PCA or ICA it uses labeled data: it is a supervised technique like PLS, but designed for classification
- Often used for dimensionality reduction prior to classification, but can be used as a classification technique itself
- When used for dimensionality reduction, it yields *(k-1)* dimensions for a k-class classification problem
- We will first discuss LDA-based classification (and, for completeness, QDA) before looking at data transformation



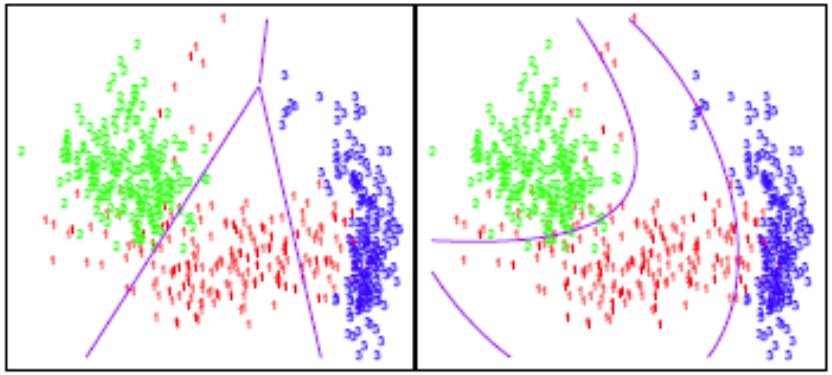
The LDA classifier

- Assumes data modeled by a multivariate Gaussian distribution for each class *c*, with mean μ_c and a common covariance Σ
- Assumption of *common* covariance matrix implies
  - the posterior distribution over the classes has a *linear* form and
  - there is a *linear* discriminant function for each class
- The linear discriminant classifier is computed as

$$y_c = \mathbf{x}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_c - 1/2\,\boldsymbol{\mu}_c^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_c + \log(n_c/n)$$

- where n_c is the number of examples of class *c* and *n* is the total number of examples
- Common variance matrix is obtained by pooling the covariance matrices from the different classes using a weighted average
- The data is classified by choosing the largest y_c

Quadratic discriminant analysis (QDA)



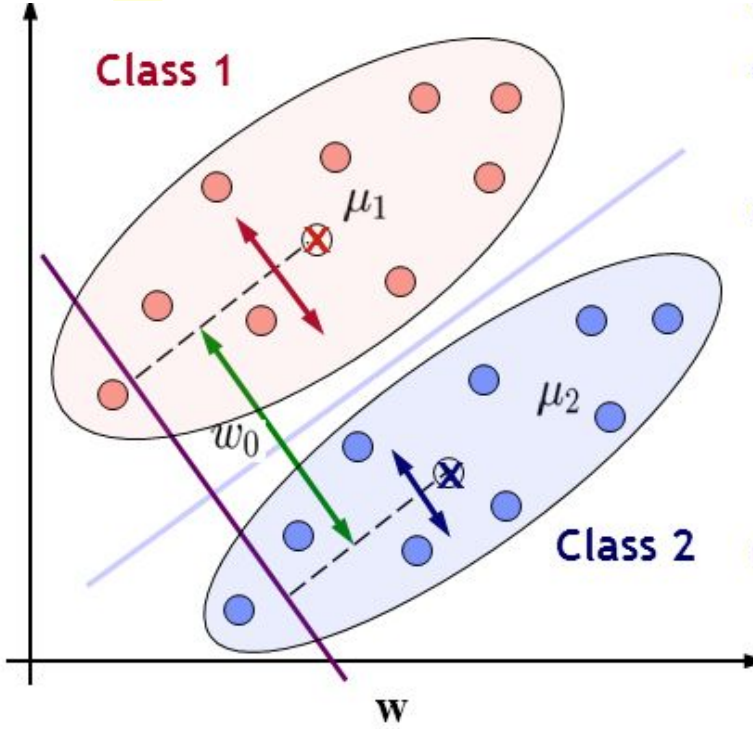
- The *QDA* classifier is obtained by giving each class its *own* covariance matrix Σ_c
- In QDA, the decision boundaries defined by the posterior over classes are described by quadratic equations
- The quadratic discriminant function for each class *c* is:

$$f_c(\mathbf{x}) = -\frac{1}{2}\log|\boldsymbol{\Sigma}_c| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_c)\boldsymbol{\Sigma}_c^{-1}(\mathbf{x}-\boldsymbol{\mu}_c)^{\mathrm{T}} + \log(n_c/n),$$

- These functions, as the ones for LDA, are produced by taking the log of the corresponding Gaussian model for each class
  - Constant terms can be ignored because such functions will be compared with one another

Fisher's linear discriminant analysis

- Maximize distances between classes
- Minimize distance within a class
- Considering the two-class case first
- We seek a projection vector ***a*** that can be used to compute scalar projections $y = \boldsymbol{a}\boldsymbol{x}$ for input vectors ***x***
- This vector is obtained by computing the means of each class, μ_1 and μ_2, and then computing two special matrices



- The *between-class scatter matrix* is calculated as

$$\mathbf{S}_B = (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^{\mathrm{T}}$$

(note the use of the outer product of two vectors here, which gives a matrix)

- The *within-class scatter matrix* is

$$\mathbf{S}_W = \sum_{i:c_i=1} (\mathbf{x}_i - \boldsymbol{\mu}_1)(\mathbf{x}_i - \boldsymbol{\mu}_1)^{\mathrm{T}} + \sum_{i:c_i=2} (\mathbf{x}_i - \boldsymbol{\mu}_2)(\mathbf{x}_i - \boldsymbol{\mu}_2)^{\mathrm{T}}$$

Fisher's LDA: the solution vector

- The solution vector a for FLDA is found by maximizing the "Rayleigh quotient"

$$J(\mathbf{a}) = \frac{\mathbf{a}^{\mathrm{T}}\mathbf{S}_B\mathbf{a}}{\mathbf{a}^{\mathrm{T}}\mathbf{S}_W\mathbf{a}}$$

- This leads to the solution

$$\mathbf{a} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$$

Multi-class FLDA

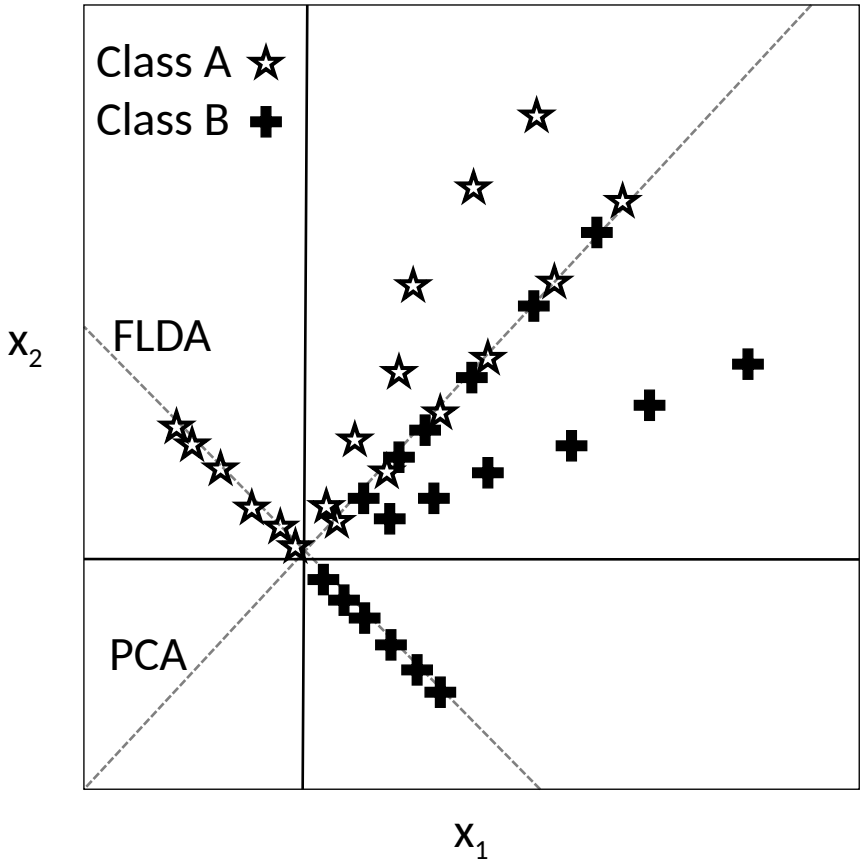
- FLDA can be generalized to multi-class problems
- Aim: projection *A* so that $\boldsymbol{y} = A^T\boldsymbol{x}$ yields points that are close when they are in the same class relative to the overall spread
- To do this, first compute both the means for each class and the global mean, and then compute the scatter matrices

$$\mathbf{S}_B = \sum_{c=1}^{C} n_c(\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^{\mathrm{T}} \qquad \mathbf{S}_W = \sum_{j=1}^{C} \sum_{i:c_i=j} (\mathbf{x}_i - \boldsymbol{\mu}_{c=j})(\mathbf{x}_i - \boldsymbol{\mu}_{c=j})^{\mathrm{T}}$$

- Then, find the projection matrix A that maximizes $J(\mathbf{A}) = \dfrac{\left|A^{\mathrm{T}}\mathbf{S}_B A\right|}{\left|A^{\mathrm{T}}\mathbf{S}_W A\right|}$

  Determinants are analogs of variances computed in multiple dimensions, along the principal directions of the scatter matrices, and multiplied together
- Solutions for finding *A* are based on solving a "generalized eigenvalue problem" for each column of the matrix *A*.

Fisher's LDA vs PCA



Adapted from Belhumeur, Hespanha & Kriegman, 1997

Text to attribute vectors

- Many data mining applications involve textual data (e.g., string attributes in ARFF)
- Standard transformation: convert string into bag of words by *tokenization*
- Attribute values are binary, word frequencies (f_{ij}), $\log(1+f_{ij})$, or TF $*$ IDF: $f_{ij} \log \frac{\#documents}{\#documents\ that\ include\ word\ i}$
- Many configuration options:
  - Only retain alphabetic sequences?
  - What should be used as delimiters?
  - Should words be converted to lowercase?
  - Should *stopwords* be ignored?
  - Should *hapax legomena* be included? Or even just the k most frequent words?

Time Series

Data Mining of Time Series

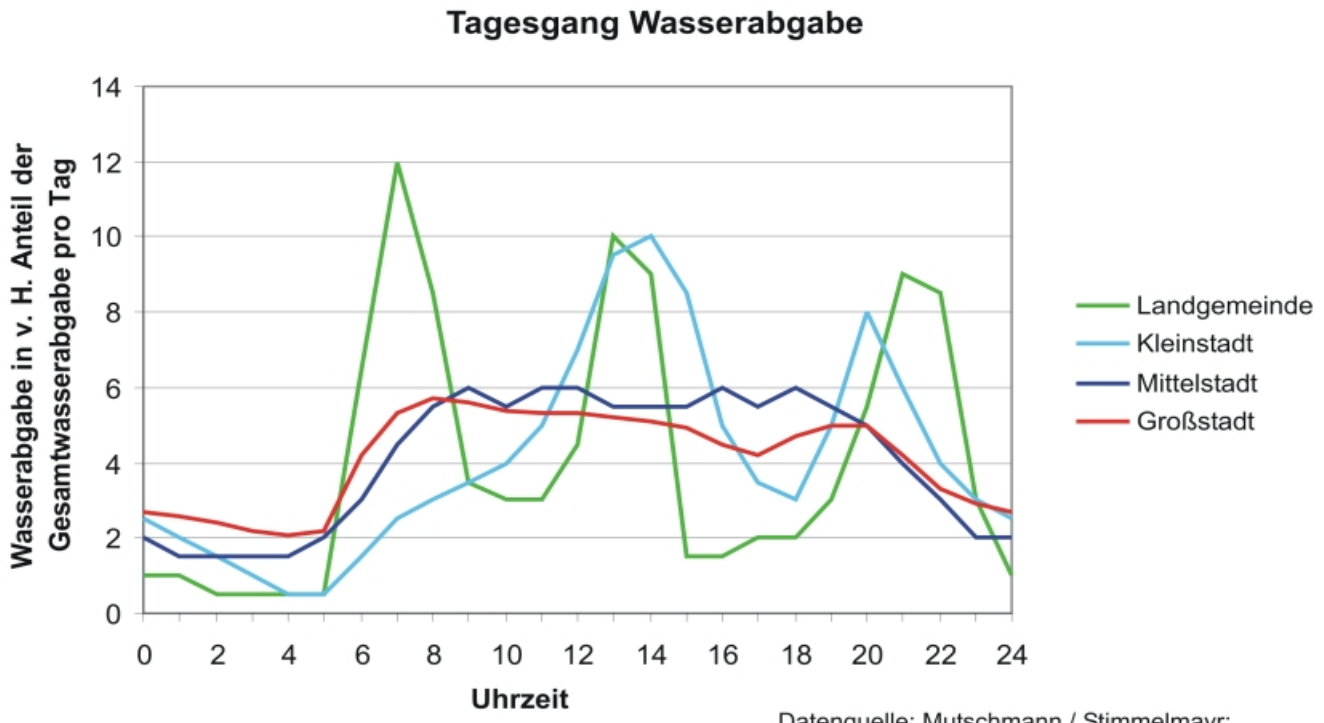
- Definition: Analysis of records where time is one of the relevant attributes
- More narrow interpretation: records generated in regular time intervals

Scale Levels of Time

- nominal: Emphasis on Periodicity
- ordinal: Sequence of events
- interval scaled: point in time, time intervals
- ration scale: time since start relevant

Time as Nominal Scale

- Emphasis on periodicity
  - Monday, Tuesday,,…/ January, February,…
  - often in combination with other scale levels



Datenquelle: Mutschmann / Stimmelmayr:
Taschenbuch der Wasserversorgung

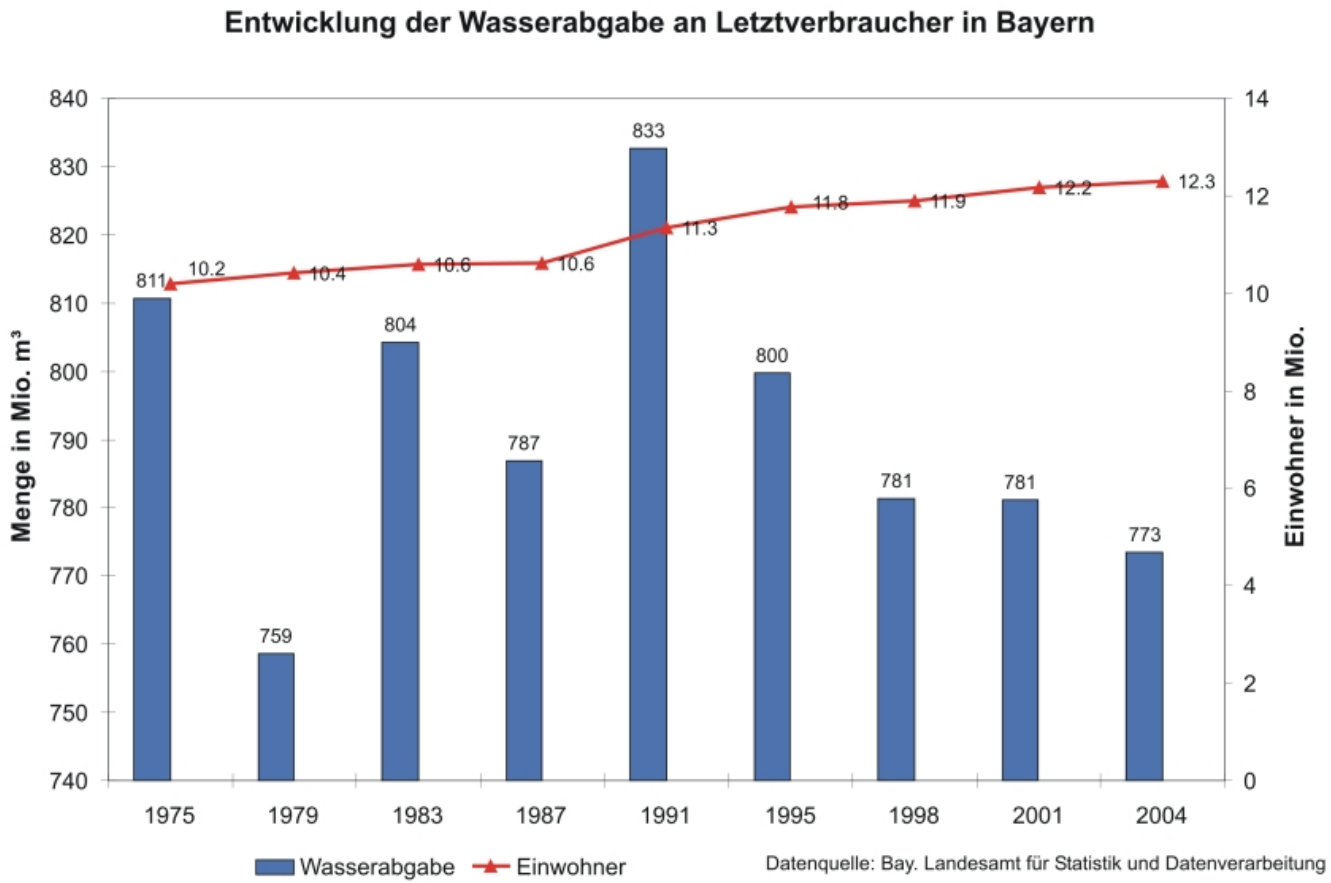
Time as Ordinal Scale

- Sequence of events
  - Transactions
  - News ticker
  - Logs
  - …

```
deleting .kde/share/apps/RecentDocuments/21link.pdf.desktop
deleting .kde/share/apps/RecentDocuments/21link.flat-2x2.pdf[3].desktop
deleting .kde/share/apps/RecentDocuments/21link.flat-2x2.pdf[2].desktop
deleting .kde/share/apps/RecentDocuments/21link.flat-2x2.pdf.desktop
.kde/share/apps/RecentDocuments/MiningSequentialPatterns.ppt.desktop
        214 100%    0.00kB/s    0:00:00 (xfer#38, to-check=1257/20694)
.kde/share/apps/RecentDocuments/
MiningSequentialPatterns2.ppt.desktop
        216 100%    2.48kB/s    0:00:00 (xfer#39, to-check=1256/20694)
.kde/share/apps/RecentDocuments/Zeitreihen.odp.desktop
        170 100%    1.95kB/s    0:00:00 (xfer#40, to-check=1255/20694)
.kde/share/apps/RecentDocuments/folien-infoseeking.pdf.desktop
        194 100%    2.23kB/s    0:00:00 (xfer#41, to-check=1254/20694)
.kde/share/apps/RecentDocuments/ieslides.sty.desktop
        152 100%    1.73kB/s    0:00:00 (xfer#42, to-check=1253/20694)
.kde/share/apps/RecentDocuments/server_log.gif.desktop
        140 100%    1.59kB/s    0:00:00 (xfer#43, to-check=1252/20694)
.kde/share/apps/RecentDocuments/tagesgang_wasser.jpg.desktop
        152 100%    1.73kB/s    0:00:00 (xfer#44, to-check=1251/20694)
.kde/share/apps/RecentDocuments/time_series_methods.pdf.desktop
        201 100%    2.28kB/s    0:00:00 (xfer#45, to-check=1250/20694)
.kde/share/apps/RecentDocuments/wachstumskurve-fichte.gif.desktop
        162 100%    1.84kB/s    0:00:00 (xfer#46, to-check=1249/20694)
.kde/share/apps/RecentDocuments/wasserabgabe-zeitintervall.jpg.desktop
        172 100%    1.95kB/s    0:00:00 (xfer#47, to-check=1248/20694)
.kde/share/apps/amarok/
.kde/share/apps/amarok/collection.db
    8107008 100%    4.72MB/s    0:00:01 (xfer#48, to-check=1247/20694)
.kde/share/apps/amarok/contextbrowser.html
       9166 100%   14.01kB/s    0:00:00 (xfer#49, to-check=1244/20694)
```

Time as Interval Scale

- Points in time, intervals
  - 9:00-10:00, 10:00-11:00,…



Time as Ratio Scale

- Time period since start relevant
  - Growth since start

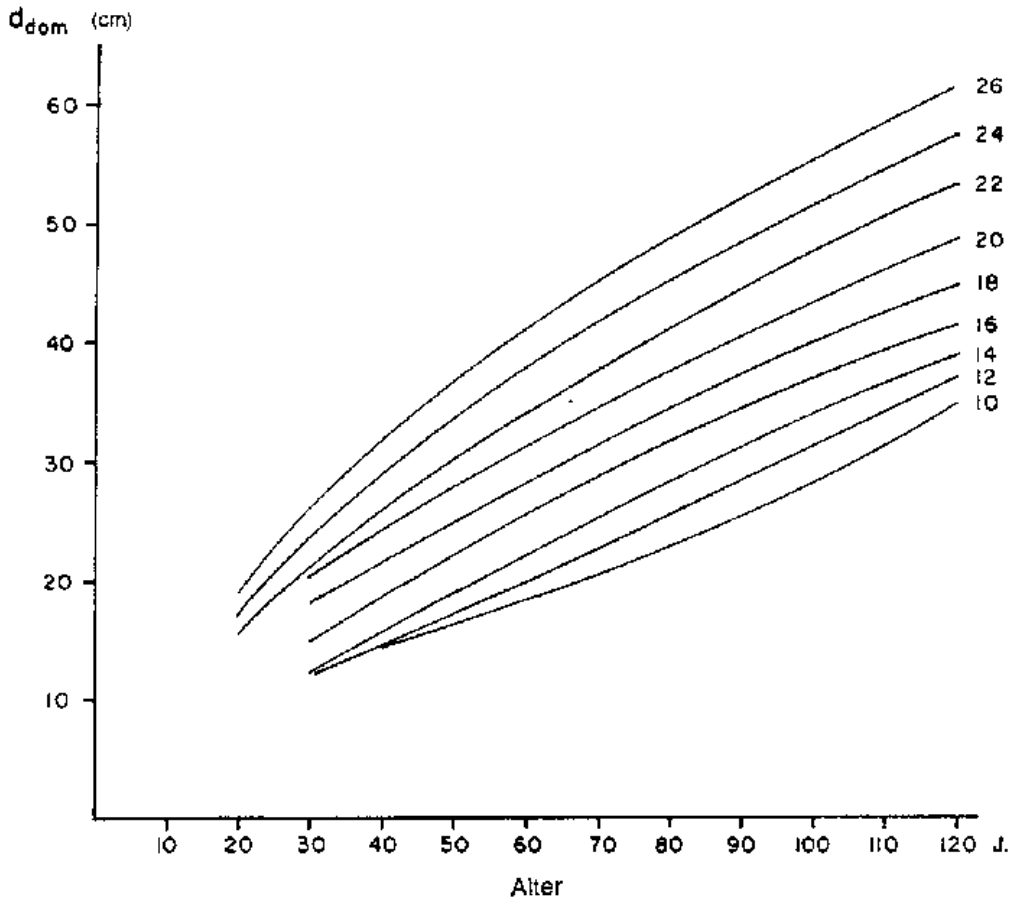


Fig.: Spruce growth for plants of different quality

Time series

- In time series data, each instance represents a different time step
- Some simple transformations:
  - Shift values from the past/future
  - Compute difference (*delta*) between instances (i.e., "derivative")
- In some datasets, samples are not regular but time is given by *timestamp* attribute
  - Need to normalize by step size when transforming
- Transformations need to be adapted if attributes represent different time steps

Sampling

- Sampling is typically a simple procedure
- What if training instances arrive one by one but we don't know the total number in advance?
  - Or perhaps there are so many that it is impractical to store them all before sampling?
- Is it possible to produce a uniformly random sample of a fixed size? Yes.
- Algorithm: *Reservoir sampling*
  - Fill the reservoir, of size *r*, with the first *r* instances to arrive
  - Subsequent, instances replace a randomly selected reservoir element with probability *r/i*, where *i* is the number of instances seen so far

Automatic data cleansing

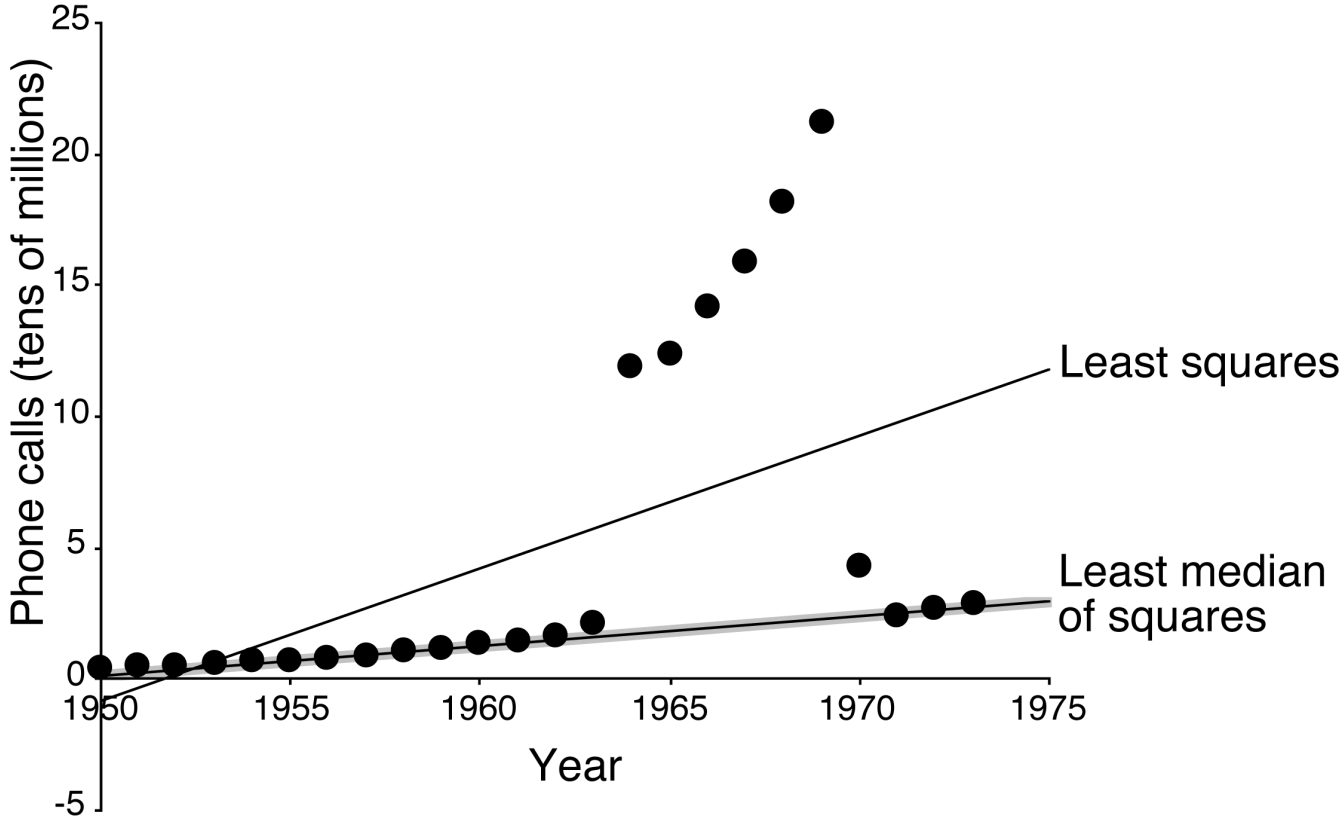
- To (potentially) improve a decision tree:
  - Remove misclassified instances, then re-learn!
- Better (of course!):
  - Human expert checks misclassified instances
- Attribute noise vs. class noise
  - Attribute noise should be left in the training set (i.e., do not train on clean set and test on dirty one)
  - Systematic class noise (e.g., one class substituted for another): leave in training set
  - Unsystematic class noise: eliminate from training set, if possible

Robust regression

- "Robust" statistical method → one that addresses problem of outliers
- Possible ways to make regression more robust:
  - Minimize absolute error, not squared error
  - Remove outliers (e.g., 10% of points farthest from the regression plane)
  - Minimize median instead of mean of squares (copes with outliers in x and y direction)
- *Least median of squares regression* finds the narrowest strip covering half the observations
  - Expensive to compute

Example: least median of squares

Number of international phone calls from Belgium, 1950–1973



Detecting anomalies

- Visualization can help to detect anomalies
- Automatic approach: apply committee of different learning schemes, e.g.,
  - decision tree
  - nearest-neighbor learner
  - linear discriminant function
- Conservative *consensus* approach: delete instances incorrectly classified by all of them
  - Problem: might sacrifice instances of small classes

One-Class Learning

- Usually training data is available for all classes
- Some problems exhibit only a single class at training time
- Test instances may belong to this class or a new class not present at training time
- This the problem of *one-class classification*
- Predict either *target* or *unknown*
- Note that, in practice, some one-class problems can be re-formulated into two-class ones by collecting negative data
- Other applications truly do not have negative data, e.g., password hardening (password typed in correct rhythm)

Outlier detection

- One-class classification is often used for *outlier/anomaly/ novelty* detection
- First, a one-class models is built from the dataset
- Then, outliers are defined as instances that are classified as *unknown*
- Another method: identify outliers as instances that lie beyond distance *d* from percentage *p* of training data
- Density estimation is a very useful approach for one-class classification and outlier detection
  - Estimate density of the target class and mark low probability test instances as outliers
  - Threshold can be adjusted to calibrate sensitivity

Using artificial data for one-class classification

- Can we apply standard multi-class techniques to obtain one-class classifiers?
- Yes: generate artificial data to represent the unknown non-target class
  - Can then apply any off-the-shelf multi-class classifier
  - Can tune rejection rate threshold if classifier produces probability estimates
- Too much artificial data will overwhelm the target class!
  - But: unproblematic if multi-class classifier produces accurate class probabilities and is not focused on misclassification error
- Generate uniformly random data?
  - Curse of dimensionality – as # attributes increases it becomes infeasible to generate enough data to get good coverage of the space

Generating artificial data

- Idea: generate data that is *close* to the target class
- *T* – target class, *A* – artificial class
- Generate artificial data using appropriate distribution *P*(*X* | *A*)
- Data no longer uniformly distributed -> must take this distribution into account when computing membership scores for the one-class model
- Want *P*(*X* | *T*), for any instance *X*; we know P(*X* | *A*)
- Train probability estimator *P*(*T* | *X*) on two classes *T* and *A*
- Then, rewrite Bayes' rule:

$$P[X|T] = \frac{(1-P[T])Pr[T|X]}{P[T](1-P[T|X])} P[X|A]$$

- For classification, choose a threshold to tune rejection rate
- How to choose P(*X* | *A*)? Apply a density estimator to the target class and use resulting function to model the artificial class

Transforming multiple classes to binary ones

- Some learning algorithms only work with two class problems
- Sophisticated multi-class variants exist in many cases but can be very slow or difficult to implement
- A common alternative is to transform multi-class problems into multiple two-class ones
- Simple methods:
  - Discriminate each class against the union of the others – *one-vs.-rest*
  - Build a classifier for every pair of classes – *pairwise classification*
- We will discuss *error-correcting output codes* and *ensembles of nested dichotomies*, which can often improve on these

Error-correcting output codes

- Multiclass problem → multiple binary problems

  Simple one-vs.rest scheme:
  One-per-class coding

- Idea: use *error-correcting codes* instead

  base classifiers predict
  1011111, true class = ??

- Use bit vectors (codes) sot that we have large *Hamming distance* between any pair of bit vectors:

  Can correct up to (d – 1)/2 single-bit errors

class	class vector
a	1000
b	0100
c	0010
d	0001

class	class vector
a	1111111
b	0000111
c	0011001
d	0101010

More on ECOCs

- Two optimization criteria for code matrix:
  1. *Row separation*: minimum distance between rows
  2. *Column separation*: minimum distance between columns (and columns' complements)
- Why is column separation important? Because if columns are identical, column classifiers will likely make the same errors
- Even if columns are not identical, error-correction is weakened if errors are correlated
- 3 classes → only 2^3 possible columns
  - (and 4 out of the 8 are complements)
  - Cannot achieve row and column separation
- ECOCs only work for problems with > 3 classes

Exhaustive ECOCs

- *Exhaustive* code for *k* classes:
  - Columns comprise every possible *k*-string …
  - … except for complements and all-zero/one strings
  - Each code word contains $2^{k-1} - 1$ bits
- Class 1: code word is all ones
- Class 2: 2^{k-2} zeroes followed by $2^{k-2} - 1$ ones
- Class *i* : alternating runs of 2^{k-i} 0s and 1s
  - last run is one short

Exhaustive code, *k* = 4

class	class vector
a	1111111
b	0000111
c	0011001
d	0101010

More on ECOCs

- #classes >> 10 → exhaustive codes infeasible

  Number of columns increases exponentially
- But: it turns out that random code words have good error-correcting properties on average!
- Alternatively, there are sophisticated methods for generating ECOCs with just a few columns
- Note: ECOCs do not work with NN classifiers because errors of column classifiers will be perfectly correlated

  But: works if different attribute subsets are used in nearest neighbor classifiers applied to predict each column/output bit

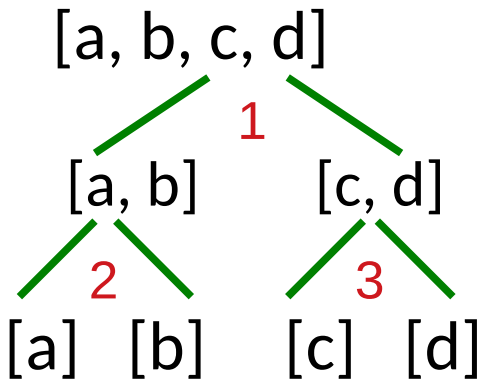
Ensembles of nested dichotomies

- ECOCs produce classifications, but what if we want class probability estimates as well?
  E.g., for cost-sensitive classification via minimum expected cost
- *Nested dichotomies* provide an alternative
- Also decompose multi-class problems to binary ones
- Work with two-class classifiers that can produce class probability estimates: yield *multi-class* probability estimates
- Recursively split the full set of *classes* into smaller and smaller subsets
- Set of instances is split into subsets corresponding to these subsets of classes
- Yields a binary tree of classes called a *nested dichotomy*

Example with four classes

Full set of classes:

Two disjoint subsets:



A two-class classifier is learned at each internal node of this tree

Nested dichotomy as a code matrix:

Class	Class vector
a	1 0 X
b	1 1 X
c	0 X 0
d	0 X 1

Probability estimation

- Suppose we want to compute P(a | x)?
  - Learn two class models for each of the three internal nodes
  - From the two-class model at the root:

    P({a, b} | x)
  - From the left-hand child of the root:

    P({a} | x, {a , b})
  - Using the chain rule:

    P({a} | x) = P({a} | x, {a , b}) $\times$ P({a, b} | x)
- Issues
  - Estimation errors for deep hierarchies
  - How to decide on hierarchical decomposition of classes?

Ensembles of nested dichotomies

- If there is no reason a priori to prefer any particular decomposition, then use them all
  - Impractical for any non-trivial number of classes
- Consider a subset by taking a random sample of possible tree structures
  - Implement caching of models for efficiency (since a given two class problem may occur in multiple trees)
  - Average probability estimates over the trees
  - Experiments show that this approach yields accurate multiclass classifiers
  - Can even improve the performance of methods that can already handle multiclass problems!

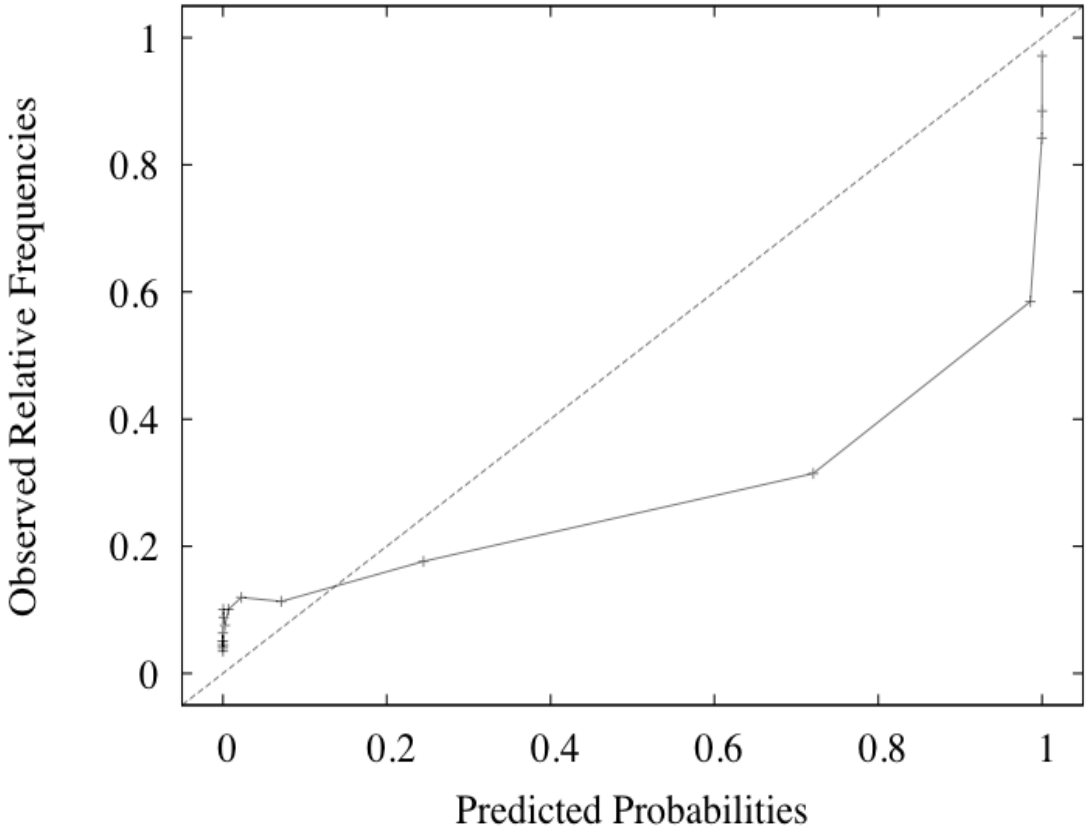
Calibrating class probabilities

- Class probability estimation is harder than classification:
  - Classification error is minimized as long as the correct class is predicted with maximum probability
  - Estimates that yield correct classification may be quite poor with respect to quadratic or informational loss
- But: it is often important to have accurate class probabilities
  - E.g. cost-sensitive prediction using the minimum expected cost method

Visualizing inaccurate probability estimates

- Consider a two class problem. Probabilities that are correct for classification may be:
  - Too optimistic – too close to either 0 or 1
  - Too pessimistic – not close enough to 0 or 1

Reliability diagram showing overoptimistic probability estimation for a two-class problem



Calibrating class probabilities

- Reliability diagram is generated by collecting predicted probabilities and relative class frequencies from a 10-fold cross-validation
- Predicted probabilities are discretized into 20 ranges via equal-frequency discretization
- We can use this for *calibration* of the probability estimates: correct bias by mapping observed curve to the diagonal
- Yields a discretization-based approach to the calibration of class probability estimates
- Discretization-based calibration is fast but determining an appropriate number of discretization intervals is not easy

Calibrating class probabilities

- Can view calibration as a function estimation problem
  - One input – estimated class probability – and one output – the calibrated probability
- Reasonable assumption in many cases: the function is piecewise constant and monotonically increasing
- Can use *isotonic regression*, which estimates a monotonically increasing piece-wise constant function:
  Minimizes squared error between observed class "probabilities" (0/1) and resulting calibrated class probabilities
- Alternatively, can use *logistic regression* to estimate the calibration function
  - Note: must use the *log-odds* of the estimated class probabilities as input
- Advantage: multiclass logistic regression can be used for calibration in the multiclass case

Weka implementations

- Attribute selection
  - CfsSubsetEval (correlation-based attribute subset evaluator)
  - ConsistencySubsetEval (measures class consistency for a given set of attributes, in the consistencySubsetEval package)
  - ClassifierSubsetEval (uses a classifier for evaluating subsets of attributes, in the classifierBasedAttributeSelection package)
  - SVMAttributeEval (ranks attributes according to the magnitude of the coefficients learned by an SVM, in the SVMAttributeEval package)
  - ReliefF (instance-based approach for ranking attributes)
  - WrapperSubsetEval (uses a classifier plus cross-validation)
  - GreedyStepwise (forward selection and backward elimination search)
  - LinearForwardSelection (forward selection with a sliding window of attribute choices at each step of the search, in the linearForwardSelection package)
  - BestFirst (search method that uses greedy hill-climbing with backtracking)
  - RaceSearch (uses the race search methodology, in the raceSearch package)
  - Ranker (ranks individual attributes according to their evaluation)

Weka implementations

- Learning decision tables: DecisionTable
- Discretization
  - Discretize (unsupervised and supervised versions)
  - PKIDiscretize (proportional k-interval discretization)
- Discriminant analysis for classification
  - LDA, FLDA, and QDA (in the discriminantAnalysis package)
- Discriminant analysis for dimensionality reduction
  - MultiClassFLDA (in the discriminantAnalysis package)
- PrincipalComponents and RandomProjection
- FastICA (independent component analysis, in the StudentFilters package)
- StringToWordVector (text to attribute vectors)
- PLSFilter (partial least squares transformation)
- Resampling and reservoir sampling

Weka implementations

- OneClassClassifier
  - Implements one-class classification using artificial data (available in the oneClassClassifier package)
- MultiClassClassifier
  - Includes several ways of handling multiclass problems with two-class classifiers, including error-correcting output codes
- END
  - Ensembles of nested dichotomies, in the ensemblesOfNestedDichotomies package
- Many other preprocessing tools are available:
  - Arithmetic operations; time-series operations; obfuscation; generating cluster membership values; adding noise; various conversions between numeric, binary, and nominal attributes; and various data cleansing operations

Further Reading and Bibliographic Notes

- Backward elimination, e.g., was introduced in (Marill & Green, 1963)
- Kittler (1978) surveys feature selection algorithms in pattern recognition.
- Best-first search and genetic algorithms are standard artificial intelligence techniques (Goldberg, 1989; Winston, 1992)
- John (1997) shows that the performance of decision tree learners can deteriorate when new attributes are added
- Langley and Sage (1997): the number of training instances must grow exponentially with the number of attributes in instance-based learning
- The idea of finding the smallest attribute set that carves up the instances uniquely is from Almuallin and Dietterich (1991, 1992)
- It was further developed by Liu and Setiono (1996)
- Kibler and Aha (1987) and Cardie (1993) both investigated the use of decision tree algorithms to identify features for nearest-neighbor learning
- Holmes and Nevill-Manning (1995) used 1R to order features for selection

Further Reading and Bibliographic Notes

- Kira and Rendell (1992) used instance-based methods to select features, leading to a scheme called RELIEF for Recursive Elimination of Features
- Gilad-Bachrach, Navot, and Tishby (2004) show how this scheme can be modified to work better with redundant attributes
- The correlation-based feature selection method is due to Hall (2000)
- The use of wrapper methods for feature selection is due to John, Kohavi, and Pfleger (1994) and Kohavi and John (1997)
- Genetic algorithms have been applied within a wrapper framework by Vafaie and DeJong (1992) and Cherkauer and Shavlik (1996)
- The selective naïve Bayes learning scheme is due to Langley and Sage (1994)
- Guyon, Weston, Barnhill, and Vapnik (2002) present and evaluate the recursive feature elimination scheme in conjunction with support vector machines
- The method of raced search was developed by Moore and Lee (1994)
- Gütlein, Frank, Hall, and Karwath (2009) show how to speed up scheme-specific selection for datasets with many attributes using simple ranking-based methods

Further Reading and Bibliographic Notes

- Dougherty, Kohavi, and Sahami (1995) show results comparing the entropy-based discretization method with equal-width binning and the 1R method
- Frank and Witten (1999) describe the effect of using the ordering information in discretized attributes
- Proportional k-interval discretization for Naive Bayes was proposed by Yang and Webb (2001)
- The entropy-based method for discretization, including the use of the MDL stopping criterion, was developed by Fayyad and Irani (1993)
- The bottom-up statistical method using the $\chi 2$ test is due to Kerber (1992)
- An extension to an automatically determined significance level is described by Liu and Setiono (1997)
- Fulton, Kasif, and Salzberg (1995) use dynamic programming for discretization and present a linear-time algorithm for error-based discretization
- The example used for showing the weakness of error-based discretization is adapted from Kohavi and Sahami (1996)

Further Reading and Bibliographic Notes

- Fradkin and Madigan (2003) analyze the performance of random projections
- The algorithm for partial least squares regression is from Hastie et al. (2009)
- The TFxIDF metric is described by Witten et al. (1999b)
- Hyvärinen and Oja (2000) created the fast ICA method
- Duda et al. (2001) and Murphy (2012) explain the algebra underlying FLDA
- Sugiyama (2007) presents a variant called “local Fisher discriminant analysis”
- John (1995) showed experiments on using C4.5 to filter its own training data
- The more conservative consensus filter is due to Brodley and Fried (1996)
- Rousseeuw and Leroy (1987) describe the least median of squares method and the telephone data
- Quinlan (1986) shows how removing attribute noise can be detrimental

Further Reading and Bibliographic Notes

- Barnett and Lewis (1994) address the general topic of outliers in data from a statistical point of view
- Pearson (2005) describes the statistical approach of fitting a distribution to the target data
- Schölkopf, Williamson, Smola, Shawe-Taylor, and Platt (2000) describe the use of support vector machines for novelty detection
- Abe, Zadrozny, and Langford (2006), amongst others, use artificial data as a second class
- Combining density estimation and class probability estimation using artificial data is suggested for unsupervised learning by Hastie et al. (2009)
- Hempstalk, Frank, and Witten (2008) describe it in the context of one-class classification
- Hempstalk and Frank (2008) discuss how to fairly compare one-class and multiclass classification when discriminating against new classes of data

Further Reading and Bibliographic Notes

- Vitter (1985) describes the algorithm for reservoir sampling we used, he called it method R; its computational complexity is O(#instances)
- Rifkin and Klautau (2004) show that the one-vs-rest method for multiclass classification can work well if appropriate parameter tuning is applied
- Friedman (1996) describes the technique of pairwise classification and Fürnkranz (2002) further analyzes it
- Hastie and Tibshirani (1998) extend it to estimate probabilities using pairwise coupling
- Fürnkranz (2003) evaluates pairwise classification as a technique for ensemble learning
- ECOCs for multi-class classification were proposed by Dietterich and Bakiri (1995); Ricci and Aha (1998) showed how to apply them to nearest neighbor classifiers
- Frank and Kramer (2004) introduce ensembles of nested dichotomies
- Dong, Frank, and Kramer (2005) considered balanced nested dichotomies to reduce training time

Further Reading and Bibliographic Notes

- Zadrozny and Elkan (2002) applied isotonic regression and logistic regression to the calibration of class probability estimates
- They also investigated how to deal with multiclass problems
- Niculescu-Mizil and Caruana (2005) compared a variant of logistic regression and isotonic regression
- They considered a large set of underlying class probability estimators
- Stout (2008) describes a linear-time algorithm for isotonic regression based on minimizing squared error (called the PAV algorithm)