

Audio signal classification

ISMIR Graduate School, October 4th-9th, 2004



Contents:

- Introduction to pattern classification
- Features
- Feature selection
- Classification methods

1 Introduction to pattern classification

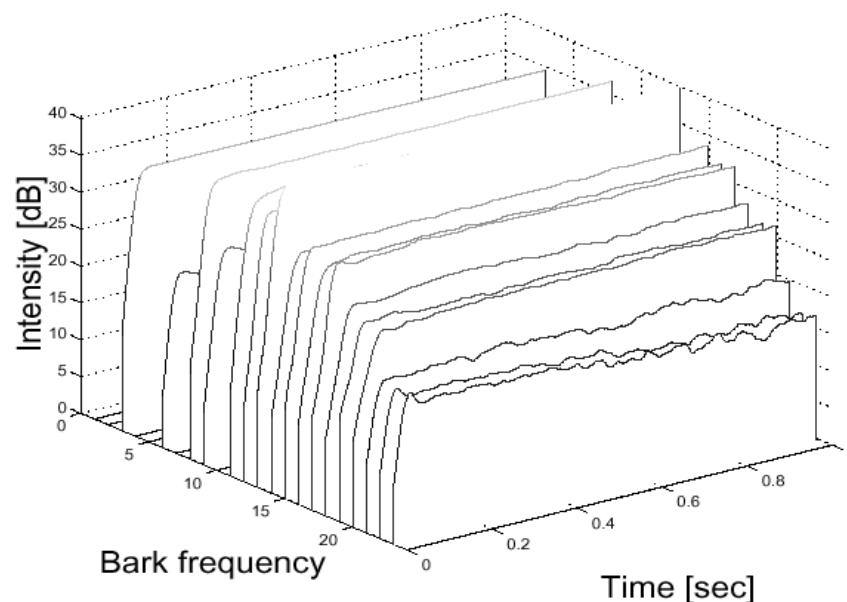
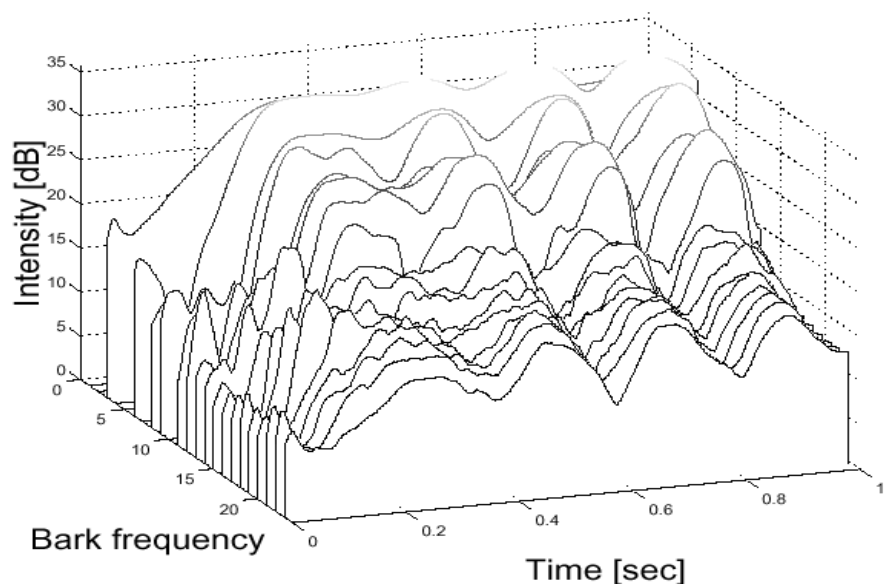
- Refresher of the basic concepts of pattern classification:
<http://rii.ricoh.com/~stork/DHSch1.ppt>
 - (See also <http://rii.ricoh.com/~stork/DHS.html>)

Audio signal classification

- Concrete problems
 - musical instrument classification
 - musical genre classification
 - percussive instrument transcription
 - music segmentation
 - speaker recognition, language recognition, sound effects retrieval, context awareness, video segmentation using audio,...
- Closely related to *sound source recognition* in humans
 - includes segmentation (perceptual sound separation) in polyphonic signals
- Many efficient methods have been developed in the speech / speaker recognition field

Example: musical instrument recognition

- Different acoustic properties of sound sources make them recognizable
 - properties result from sound production mechanism
- But: even a single source produces varying sounds
 - we must find something characteristic to the source, not sound events alone:
source invariants
- Examples below: flute (left) and clarinet (right) – what to measure?
[Eronen&Klapuri,2000]



Typical classification system

Simplified:

A. Feature extraction

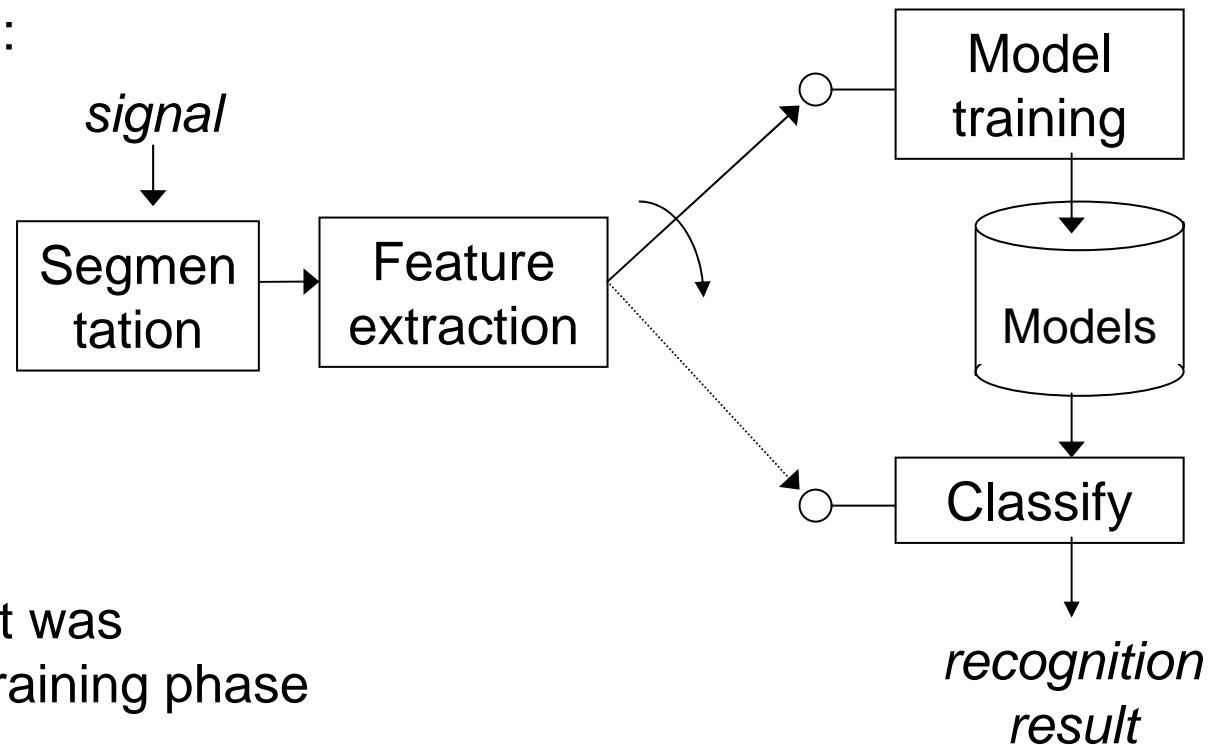
B. Classification (using *models*)

1. training phase:

learn from examples,
collect statistics
for feature
values

2. classify new
instances

based on what was
learnt in the training phase



2 Feature extraction

- Feature extraction is inevitable
 - a time-domain signal as such contains too much irrelevant data to use it directly for classification



- Using the right features is crucial for successful classification
- Good features simplify the design of a classifier whereas lousy features (with little discriminating power) can hardly be compensated with any classifier

2.1 Spectral features

- = Features that characterize the short-time spectrum
- Most successful features for audio classification (so far)
- Also the most general-purpose features for different problems
 - in contrast, *temporal* features are typically different for instrument recognition, genre recognition, or speaker recognition, for example

In extracting spectral features

- Phase spectrum can be discarded!
 - 50% reduction of information
- Spectral fine structure can be discarded (in almost all cases)
 - even more reduction of irrelevant information
- Retain only the *coarse spectral energy distribution*
 - most important for general audio classification
 - basis for speech and speaker recognition

Spectral features

Cepstral coefficients

- *Cepstral coefficients* $c(k)$ are a very convenient way to model spectral energy distribution

$$c(k) = IDFT \{ \log |DFT \{ x(n) \}| \}$$

where DFT denotes the Fourier-transform and $IDFT$ its inverse

- In Matlab

```
% see "type rceps"
```

```
c = real( ifft( log( abs( fft(x) ) ) ) );
```

- only real part, `real()`, is taken because numerical precision produces infinitesimally small imaginary part

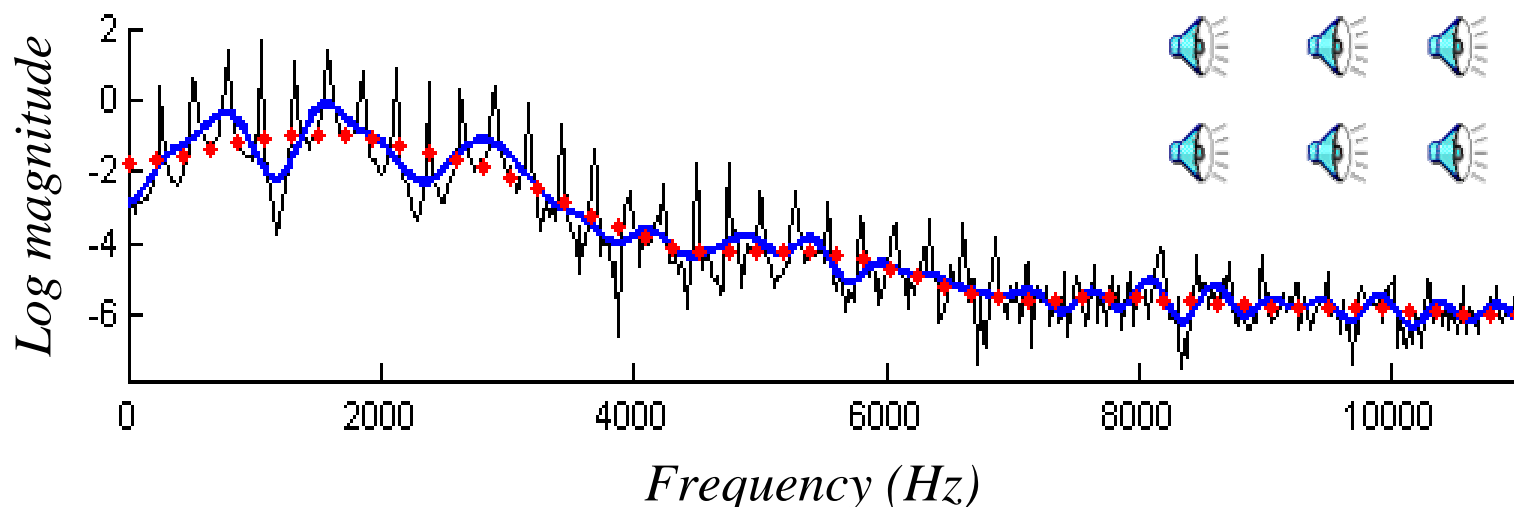
- Cepstrum coefficients are calculated in short frames over time

- can be further modeled by calculating e.g. the mean and variance of each coefficient over time

Spectral features

Cepstral coefficients

- Only the first M cepstrum coefficients are used as features
 - all coefficients model the precise spectrum
 - coarse spectral shape is modeled by the first coefficients
 - precision is selected by the number of coefficients taken
 - the first coefficient (energy) is usually discarded
- Usually $M = f_s / (2000 \text{ Hz})$ is a good first guess for M
- Figure: piano spectrum (thin black line) and spectrum modeled with the first 50 coeffs (thick blue line) or 10 coeffs (red broken line)



Spectral features

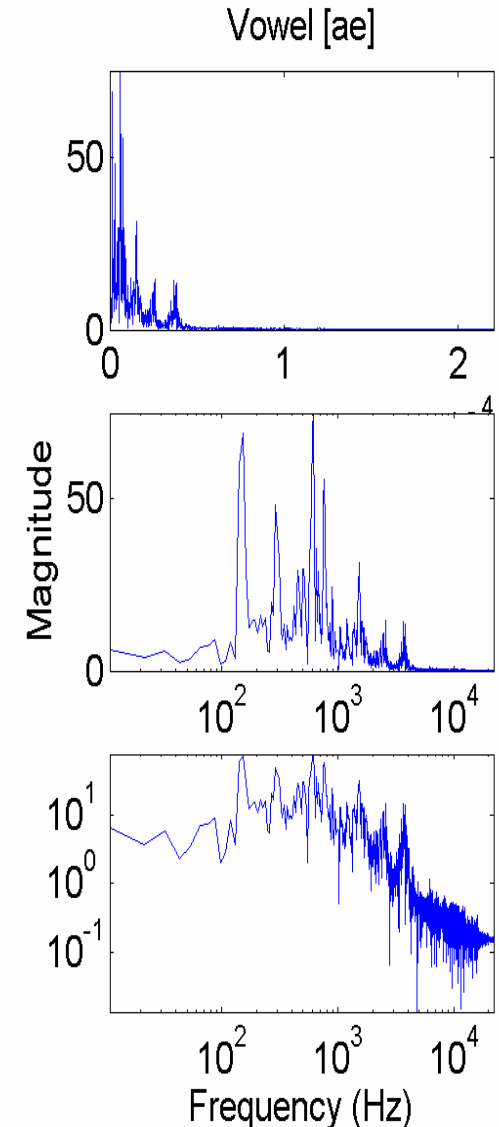
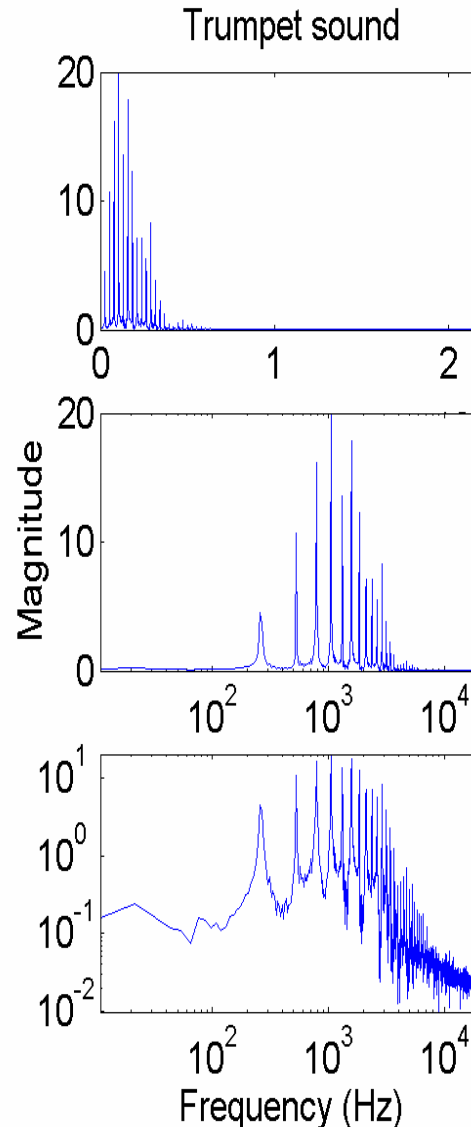
Cepstral coefficients

- A drawback of the cepstral coefficients: *linear frequency scale*
- Perceptually, the frequency ranges 100–200Hz and 10kHz – 20kHz should be approximately equally important
 - the standard cepstral coefficients do not take this into account
 - *logarithmic frequency scale* would be better
- Why mimic perception?
 - typically we want to classify sounds according to perceptual (dis)similarity
 - perceptually relevant features often lead to robust classification, too
- Desirable for features:
 - small change in feature vector → small perceptual change (and vice versa)
 - Mel-frequency cepstral coefficients fulfill this criterion

Spectral features

Frequency and magnitude warping

- Linear scale
 - usually hard to "see" anything
- Log-frequency
 - each octave is approximately equally important perceptually
- Log-magnitude
 - perceived change from 50dB to 60dB about the same as from 60dB to 70dB



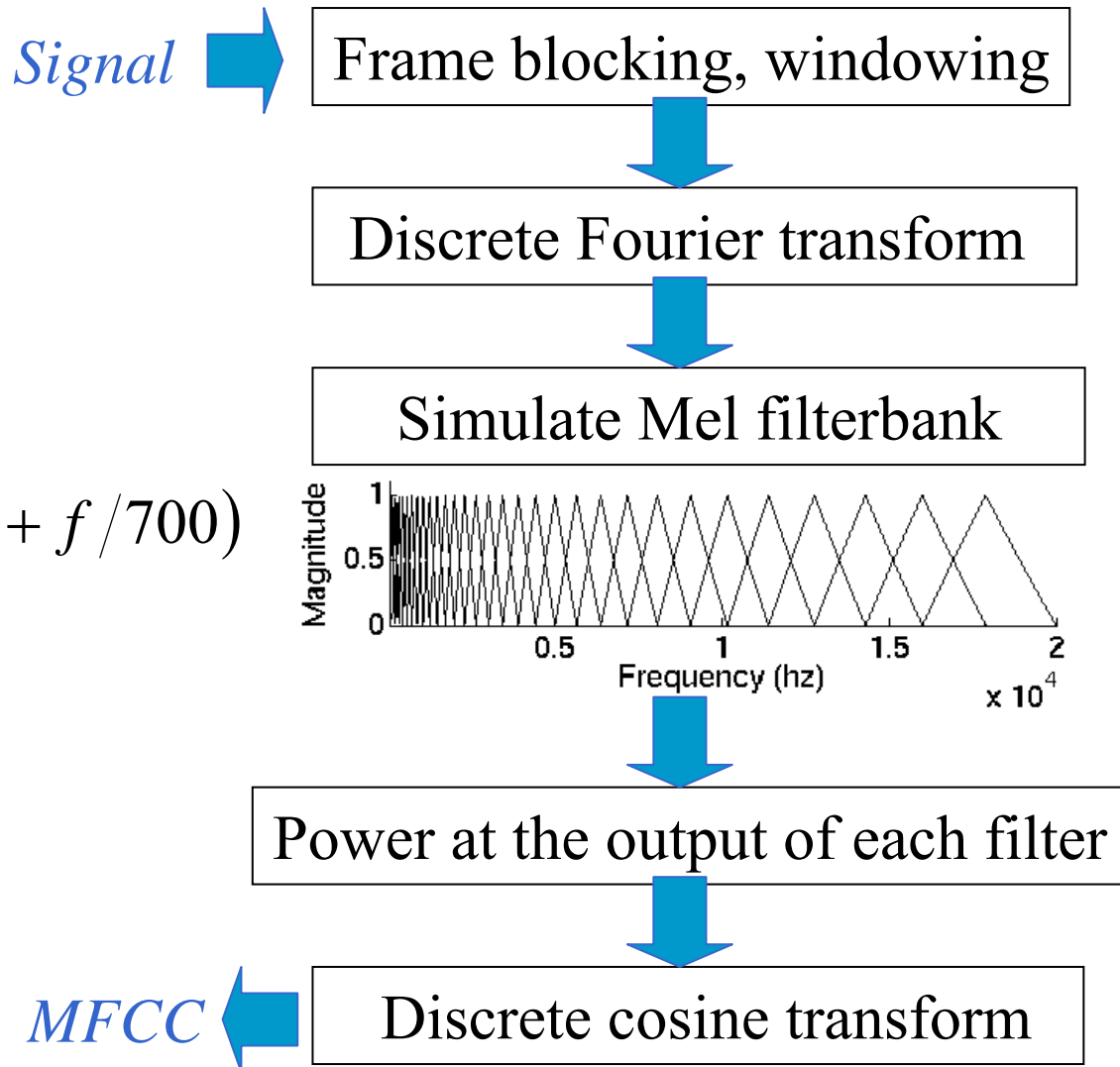
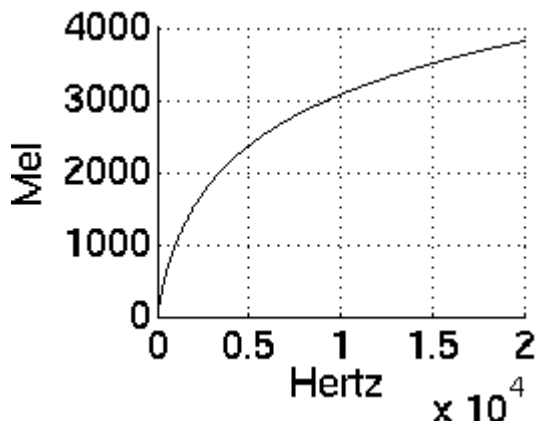
Spectral features

Mel-frequency cepstral coefficients

- Improves over the standard cepstral coefficients
- **Figure:** calculations in practice

- Mel frequency scale:

$$Mel(f) = 2595 \log_{10}(1 + f/700)$$



Spectral features

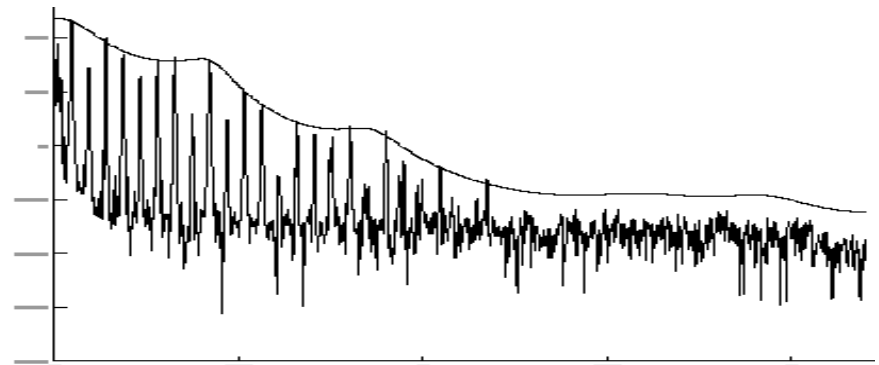
Mel-frequency cepstral coefficients

- Some reasons why MFCCs are successful:
 - Mel-frequency scale
 - Log of magnitudes

*Large change in MFCC vector
↔ large perceptual change*

 - Discrete cosine transform

*Drop spectral fine structure,
decorrelate the features*
- Δ MFCCs and other additional features can be catenated to the feature vector
- **Figure:** another way to model spectral energy distribution:
 - low-order all-pole filter



Spectral features

Other spectral features

- Spectral centroid (correlates with brightness)

$$SC = \left(\sum_{k=1}^K k \times |X(k)|^2 \right) / \left(\sum_{k=1}^K |X(k)|^2 \right)$$

where $|X(k)|$ and $f(k)$ are the magnitude and frequency of component k

- Bandwidth


$$BW = \sqrt{\left(\sum_{k=1}^K (k - SC)^2 |X(k)|^2 \right) / \left(\sum_{k=1}^K |X(k)|^2 \right)}$$

- More specific ones... for example for harmonic sounds:
 - spectral irregularity: standard deviation of harmonic amplitudes from spectral envelope
 - even and odd harmonic content (open vs. closed acoustic pipes)

2.2 Temporal features

- Characterize the temporal evolution of an audio signal
- Temporal features tend to be more task-specific
- Middle-level representation for feature extraction:
 - power envelope of the signal sampled at 100Hz...1kHz rate
 - or: power envelopes of the signal at 3...40 subbands
- *Note*: also these drop the phase spectrum and all spectral fine structure!

Temporal features

- Musical instrument classification:
 - *rise time*: time interval between the onset and instant of maximal amplitude
 - *onset asynchrony* at different frequencies
 - *frequency modulation*: amplitude and rate (vibrato: 4–8Hz) 
 - *amplitude modulation*: amplitude and rate
 - tremolo, roughness (fast amplitude modulation at 15-300 Hz rate)
- General audio classification
 - e.g. amplitude modulation
 - also Δ MFCCs can be seen as temporal features

2.3 Features calculated in the time domain

- Sometimes (computationally) ultra-light features are needed and the Fourier transform is avoided

- Zero-crossing rate

$$ZCR = \frac{1}{N} \sum_{n=2}^N |sign(x(n)) - sign(x(n-1))| \quad sign(x) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases}$$

- **Figure:** ZCR correlates strongly with spectral centroid (~ brightness)

spectral centroid

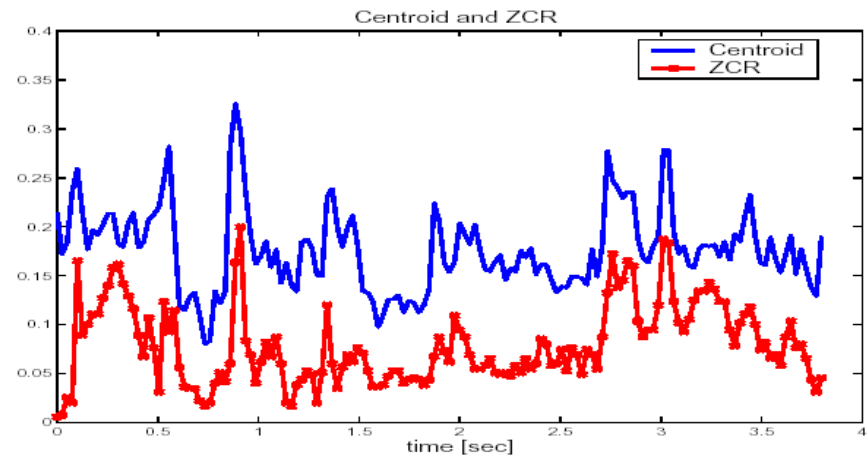
zero-crossing rate

[Peltonen, MSc thesis, 2001]

- Short-time energy

$$STE = \frac{1}{N} \sum_{n=1}^N x(n)^2$$

- lousy feature as such, but different statistics of STE are useful



More features...

- **Table:** some features for musical instrument recognition [Eronen,Klapuri,2000]
- One can measure many things...

Table 1: Feature descriptions	
1	Rise time, i.e., the duration of attack
2	Slope of line fitted into rms-energy curve after attack
3	Mean square error of line fit in 2
4	Decay time
5	Time between the end of attack and the maximum of rms-energy
6	Crest factor, i.e., max / rms of amplitude
7	Maximum of normalized spectral centroid
8	Mean of normalized spectral centroid
9	Mean of spectral centroid
10	Standard deviation of spectral centroid
11	Standard deviation of normalized spectral centroid
12	Frequency of amplitude modulation, range 4-8Hz
13	Strength of amplitude modulation, range 4-8Hz
14	Heuristic strength of the amplitude modulation in range 4-8Hz
15	Frequency of amplitude modulation, range 10-40Hz
16	Strength of amplitude modulation, range 10-40Hz
17	Standard deviation of rise times at each Bark band
18	Mean error of the fit between each of steady state intensities and mean steady state intensity
19	Mean error of fit between each of onset intensities and mean onset intensity
20	Overall variation of intensities at each band
21	Fundamental frequency
22	Standard deviation of fundamental frequency
23-33	Average cepstral coefficients during onset
34-44	Average cepstral coefficients after onset

3 Feature selection¹

¹) This section is based on lecture notes of Antti Eronen

- Let us consider methods to *select a set of features from a larger set of available features*
- The number of features at the disposal of the designer is usually very large (tens or even hundreds)
- Reasons for reducing the number of features to a sufficient minimum:
 - less features → simpler models → less training data needed (remember the “*curse of dimensionality*”)
 - amount of training data: the higher the ratio of the number of training patterns N to the number of free classifier parameters, the better the *generalisation properties* of the resulting classifier
 - computational complexity, correlating (redundant) features

Feature selection

- Feature selection (or, *reduction*) problem:
 - reduce the dimension of the feature vectors while preserving as much class discriminatory information as possible
- Good features result in *large between-class distance* and *small within-class variance*
- Approaches for finding good sets of features:
 - examine features *individually* and discard those with little discrimination ability
 - a better way is to examine the *features in combinations*
 - linear (or nonlinear) *transformation* of the feature vector

3.1 Data normalization

- Features with large values may have a larger influence in the cost function than features with small values (e.g. Euclidean distance)
- For N available data points of the d th feature ($d = 1, 2, \dots, D$);

mean:

$$m_d = \frac{1}{N} \sum_{n=1}^N x_{nd}$$

variance:

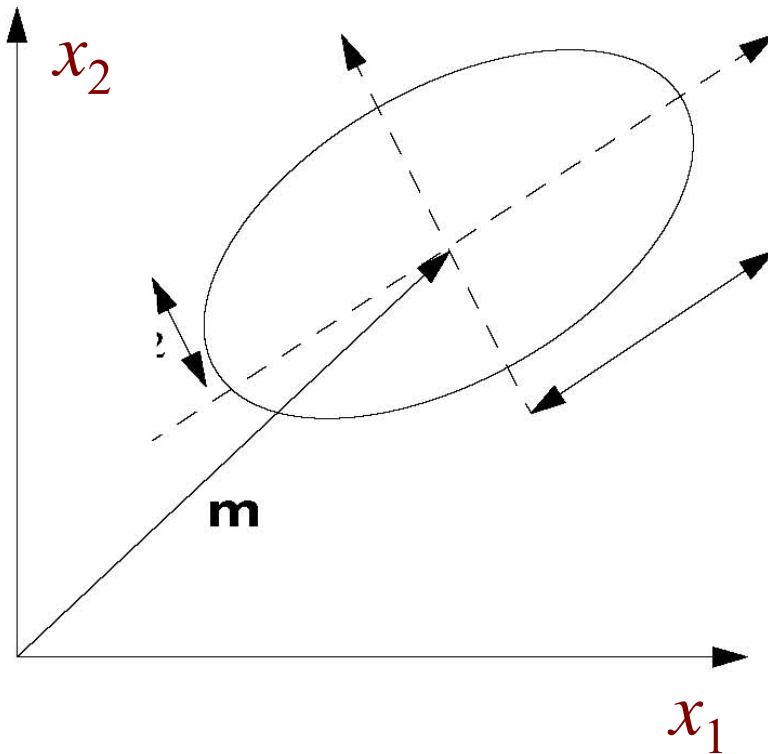
$$\sigma_d^2 = \frac{1}{N-1} \sum_{n=1}^N (x_{nd} - m_d)^2$$

*normalized
feature:*

$$\hat{x}_{nd} = \frac{x_{nd} - m_d}{\sigma_d}$$

- The resulting normalised features have zero mean and unit variance
 - if desired, feature weighting can be performed separately after normalisation.

Mean? Variance? Correlation?



3.2 Principal component analysis

- Idea: *find a linear transform A* such that when applied to the feature vectors x , the resulting new features y are *uncorrelated*.
→ the covariance matrix of the feature vectors y is diagonal.
- Define the covariance matrix of x by (here we use the training data!)

$$C_x = E\{(\mathbf{x} - \mathbf{m})(\mathbf{x} - \mathbf{m})^T\}.$$

- Because C_x is real and symmetric, it is always possible to find a set of D orthonormal *eigenvectors*, and hence it can be diagonalised:

$$A C_x A^T = \Lambda$$

Above, A is an D by D matrix whose rows are formed from the eigenvectors of C_x and Λ is a diagonal matrix with the corresponding eigenvalues on its diagonal

- Now the transform can be written as

$$\mathbf{y} = A(\mathbf{x} - \mathbf{m})$$

where \mathbf{m} is the mean of the feature vectors x .

Principal component analysis

Eigenvectors

- Recall the eigenvector equations:

$$C e_d = \lambda_d e_d$$

$$(C - \lambda_d I) e_d = 0$$

where e_d are the eigenvectors and λ_d are the eigenvalues

- In Matlab: `[A, lambdas]=eig(Cx)`
- The transform matrix A in $\mathbf{y} = A(\mathbf{x} - \mathbf{m})$ consists of the eigenvectors of C_x : $A = [e_1, e_2, \dots, e_D]^T$
 - Since C_x is real and symmetric, A is orthonormal ($AA^T = I$, $A^T = A^{-1}$)

Principal component analysis

Transformed features

- The transformed feature vectors $y = A(x - m)$ have **zero mean** and their covariance matrix (ignoring the mean terms):

$$C_y = E\{ A(x - m) (A(x - m))^T \} = AC_x A^T = \Lambda$$

i.e. C_y is diagonal and the features are thus **uncorrelated**.

- Furthermore, let us denote by $\Lambda^{1/2}$ the diagonal matrix whose elements are the square roots of the eigenvalues of C_x . Then the transformed vectors

$$y = \Lambda^{-1/2} A (x - m)$$

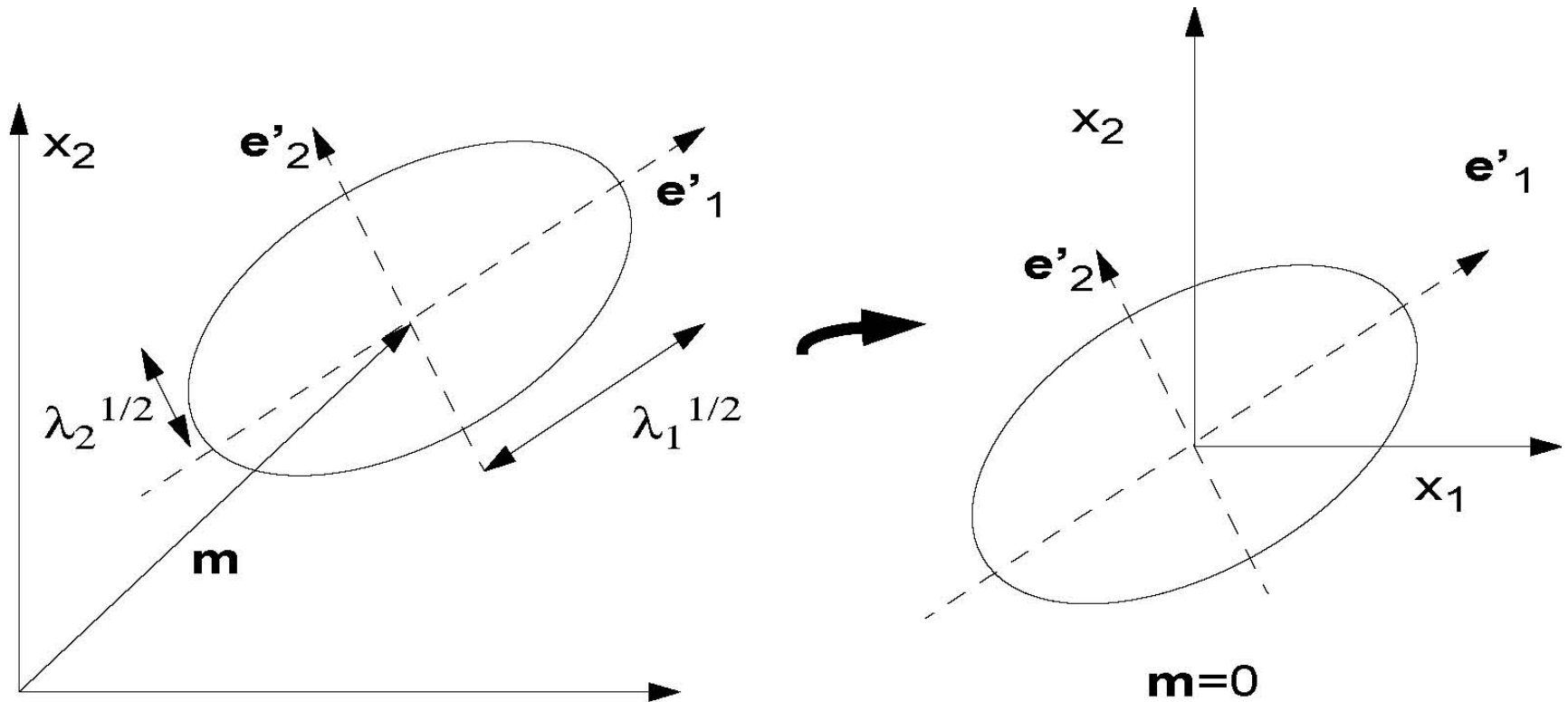
have uncorrelated elements with unit variance:

$$C_y = (\Lambda^{-1/2} A) C_x (\Lambda^{-1/2} A)^T = \dots = I.$$

Principal component analysis

Transformed features

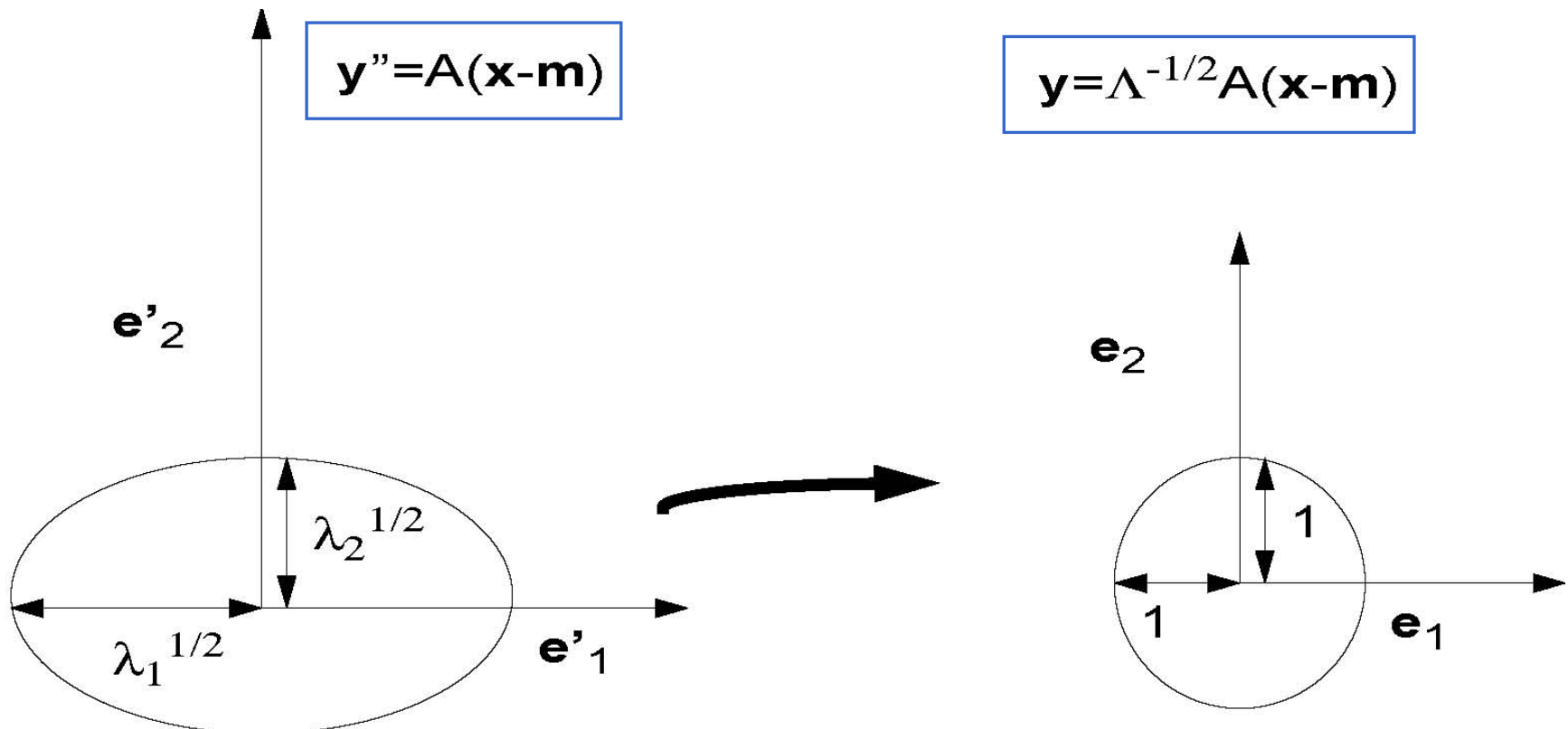
- Mean removal: $y' = x - m$



Principal component analysis

Transformed features

- Left: decorrelated features \rightarrow basis vectors orthogonal $\mathbf{e}_1^T \mathbf{e}_2 = 0$,
- Right: variance scaling \rightarrow basis vectors orthonormal $\mathbf{e}_1^T \mathbf{e}_1 = \mathbf{e}_2^T \mathbf{e}_2 = 1$



Principal component analysis

- The transformation can be used to *reduce the amount of needed data*
 - order the eigenvectors and values so that the first row of A corresponds to the largest eigenvalue and the last row to the smallest eigenvalue
 - take only the first M principal components to create an M by D matrix A for the data projection

$$\hat{\mathbf{x}} = A_M (\mathbf{x} - \mathbf{m})$$

- We obtain an approximation for \mathbf{x} , which essentially is the projection of \mathbf{x} onto the subspace spanned by the M orthonormal eigenvectors.
- This projection is optimal in the sense that it minimises the mean square error (MSE)

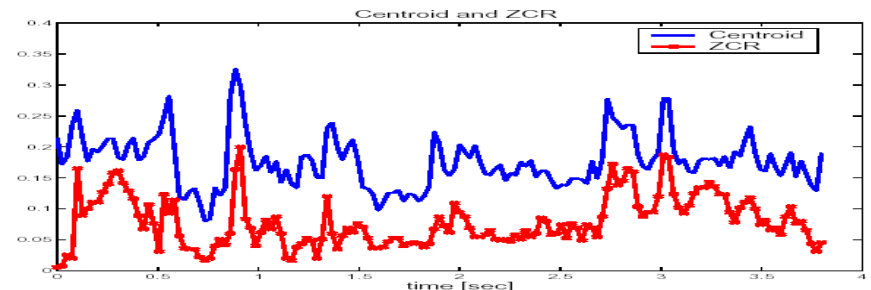
$$E \left\{ \|\hat{\mathbf{x}} - \mathbf{x}\|^2 \right\}$$

for any approximation with M components

Principal component analysis

Practical use of PCA

- PCA is useful for preprocessing features before classification
 - Note: **PCA does not take different classes into account** – it only considers the properties of different *features*
- If two features x_i and x_j are redundant, then one eigenvalue in A is very small and one dimension can be dropped
 - ***we do not need to choose between two correlating features!***: it is better to do the linear transform and then drop the least significant dimension
 - both of the correlating features are utilized
- You need to scale the feature variances before eigenanalysis, since eigenvalues are proportional to the numerical range of the features
 - procedure: 1. normalize → PCA → are there unnecessary dimensions?
- Example of correlating features:
 - ZCR and spectral centroid



3.3 Class separability measures

- PCA does not take different classes into account
 - features remaining after PCA are efficient in characterising sounds but do not necessarily discriminate between different classes
- Now we will look at a set of simple criteria that measure the *discriminating properties* of feature vectors
- *Within-class scatter matrix* (K different classes)

$$S_w = \sum_{k=1}^K p(\omega_k) C_k$$

where $C_k = E((\mathbf{x} - \mathbf{m}_k)(\mathbf{x} - \mathbf{m}_k)^T)$ is the covariance matrix for class k and $p(\omega_k)$ the prior probability of the class ω_k , i.e., $p(\omega_k) = N_k/N$, where N_k is the number of samples from class ω_k (of total N samples).

- $\text{trace}\{S_w\}$, i.e. the sum of the diagonal elements of S_w , is a *measure of the average variance of the features over all classes*.

Class separability measures

- *Between-class scatter matrix*

$$S_b = \sum_{k=1}^K p(\omega_k)(\mathbf{m}_k - \mathbf{m}_0)(\mathbf{m}_k - \mathbf{m}_0)^T$$

where \mathbf{m}_k is the mean of class k and \mathbf{m}_0 is the global mean vector

$$\mathbf{m}_0 = \sum_{k=1}^K p(\omega_k)\mathbf{m}_k$$

- $\text{trace}\{S_b\}$ is a *measure of the average distance of the mean of each class from the global mean value over all classes*

Class separability measures

- Now we can define a criterion

$$J_1 = \frac{\text{trace}\{S_b\}}{\text{trace}\{S_w\}}$$

- It obtains large values when
 - samples in the D -dimensional space are well clustered around their mean within each class (small $\text{trace}\{S_w\}$)
 - the clusters of the different classes are well separated (large $\text{trace}\{S_b\}$)

3.4 Feature subset selection

- Problem: how to select a subset of M features from the D originally available so that
 1. we reduce the dimensionality of the feature vector ($M < D$)
 2. we optimize the desired class separability criterion
- To find the “optimal” subset of features, we should form all possible combinations of M features out of the D originally available
 - the best combination is then selected according to any desired class separability measure J
- In practice, it is *not possible to evaluate all the possible feature combinations!*

Sequential backward selection (SBS)

- Particularly suitable for discarding a few worst features:
 1. Choose a class separability criterion J , and calculate its value for the feature vector which consists of *all* available features (\rightarrow length D)
 2. Eliminate one feature, and for each possible resulting combinations (of length $D-1$) compute J . Select the best.
 3. Continue this for the remaining features, and stop when you have obtained the desired dimension M .
- This is a *suboptimal* search procedure, since nobody can guarantee that the optimal $r-1$ dimensional vector has to originate from the optimal r dimensional one!

Feature *vector* selection

Sequential forward selection (SFS)

- Particularly suitable for finding a few "golden" features:
 1. Compute criterion J value for all individual features. Select the best.
 2. Form all possible two-dimensional vectors that contain the winner from the previous step. Calculate the criterion for each vector and select the best.
 3. Continue adding features one at time, taking always the one that results in the largest value of the criterion J .
 4. Stop when the desired vector dimension M is reached.
- Both SBS and SFS suffer from the *nesting effect*: once a feature is discarded in SBS (selected in SFS), it cannot be reconsidered again (discarded in SFS).

3.5 Linear transforms

- Feature “generation” by combining all the D features to obtain M (with $M < D$) features
- Define a mapping A that transforms the original feature vectors x to M -dimensional feature vectors such that

$$y = Ax$$

- Objectives:
 1. reduce the dimensionality of the feature vector
 2. optimize the desired class separability criterion
- Note the similarity with PCA – the difference is that here we consider the *class separability*

Linear transforms

Linear discriminant analysis (LDA)

- Choose A to optimise

$$J_2 = \text{trace}\{S_w^{-1} S_b\}$$

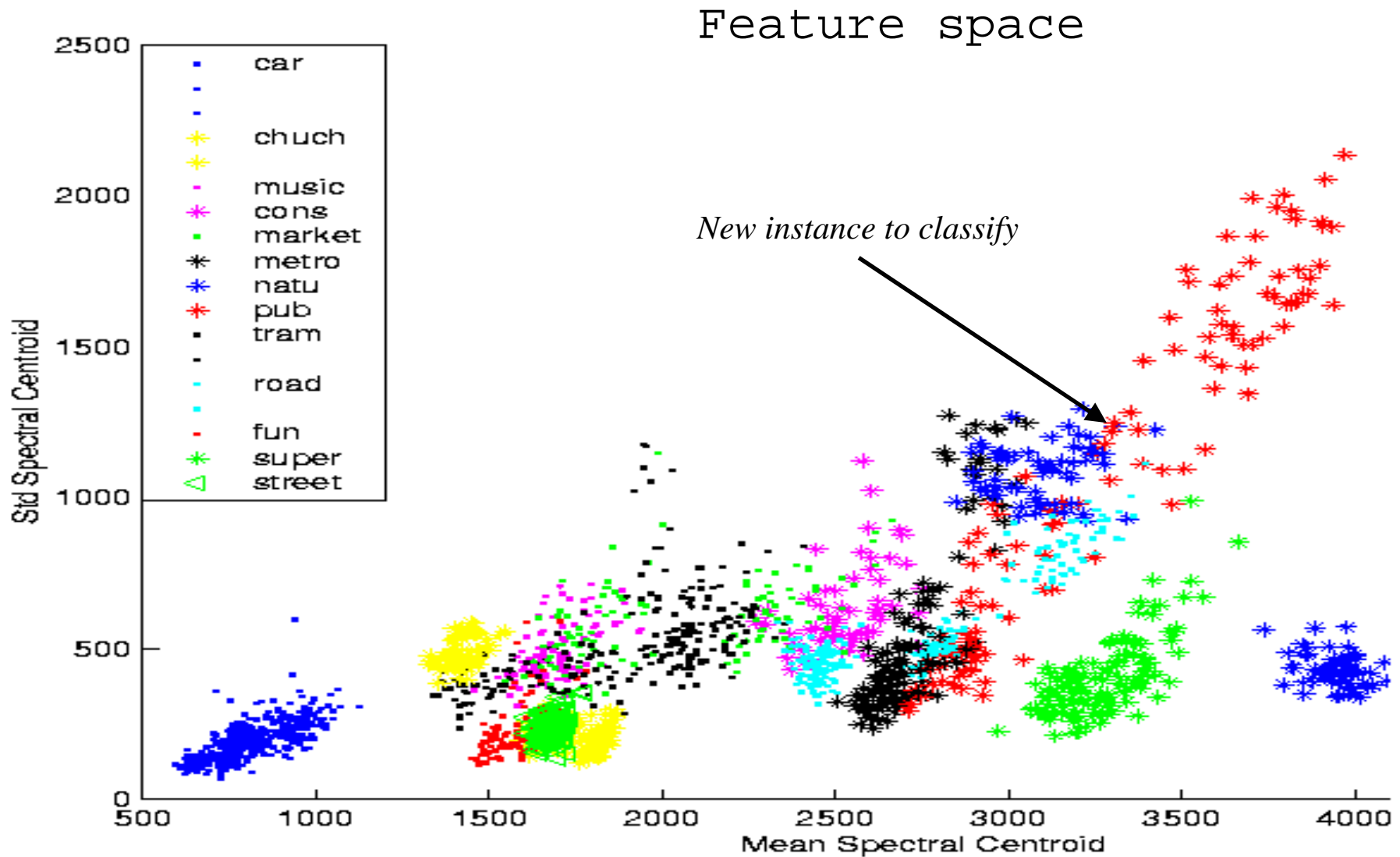
- Very similar to PCA. The difference is that here the eigenanalysis is performed for the matrix $S_w^{-1} S_b$ instead of the global covariance.
- The rows of the transform matrix are obtained by choosing the M largest eigenvectors of $S_w^{-1} S_b$
- Leads to a feature vector dimension $M \leq K-1$ where K is the number of classes
 - this is because the rank of $S_w^{-1} S_b$ is $K-1$

4 Classification methods

- Goal is to classify previously unseen instances after the classifier has learned the training data
- *Supervised* vs. *unsupervised* classification
 - supervised: the classes of training instances are told during learning
 - unsupervised: *clustering* into hitherto unknown classes
- *Supervised* classification is the focus here
- Example on the next page
 - data is represented as M -dimensional feature vectors (here $M=2$)
 - 18 different classes
 - several training samples from each class

Classification methods

Example



4.1 Classification by distance functions

- Minimum distance classification
 - calculate the distance $D(x, y_i)$ between the unknown sample x and all the training samples y_i from all classes
 - for example the Euclidean distance $D_E(x, y) = \sqrt{(x - y)^T (x - y)}$
 - choose the class according to the closest training sample
- ***k*-nearest neighbour** (*k*-NN) classifier
 - pick k nearest neighbours to x and then choose the class which was most often picked
- These are a ***lazy classifiers***
 - *training* is trivial: just store the training samples y_i
 - *classification* gets very complex with a lot of training data
→ must measure distance to all training samples
- Computational efficiency can be improved by storing only a sufficient number of ***class prototypes*** for each class

Classification by distance functions

Distance metrics

- Choice of the distance metric is very important

- *Euclidean* distance metric:

$$D_E(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})} = \sqrt{\sum_d (x_d - y_d)^2}$$

sqrt is order-preserving, thus it is equivalent to minimize

$$D_E^2(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})$$

- *Mahalanobis* distance between \mathbf{x} and \mathbf{y}

$$D_M(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T C^{-1} (\mathbf{x} - \mathbf{y})$$

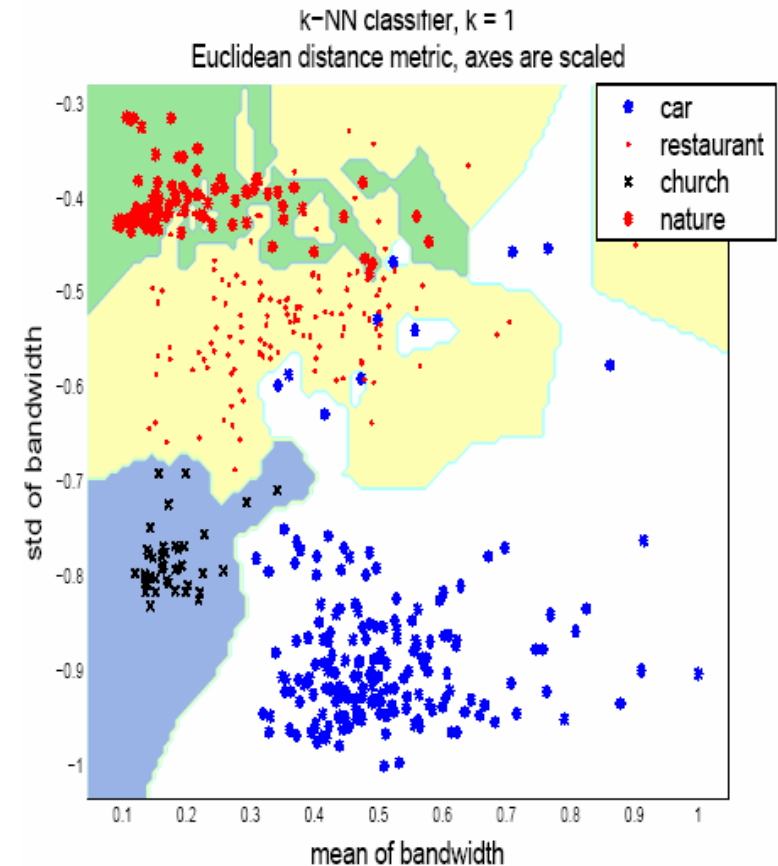
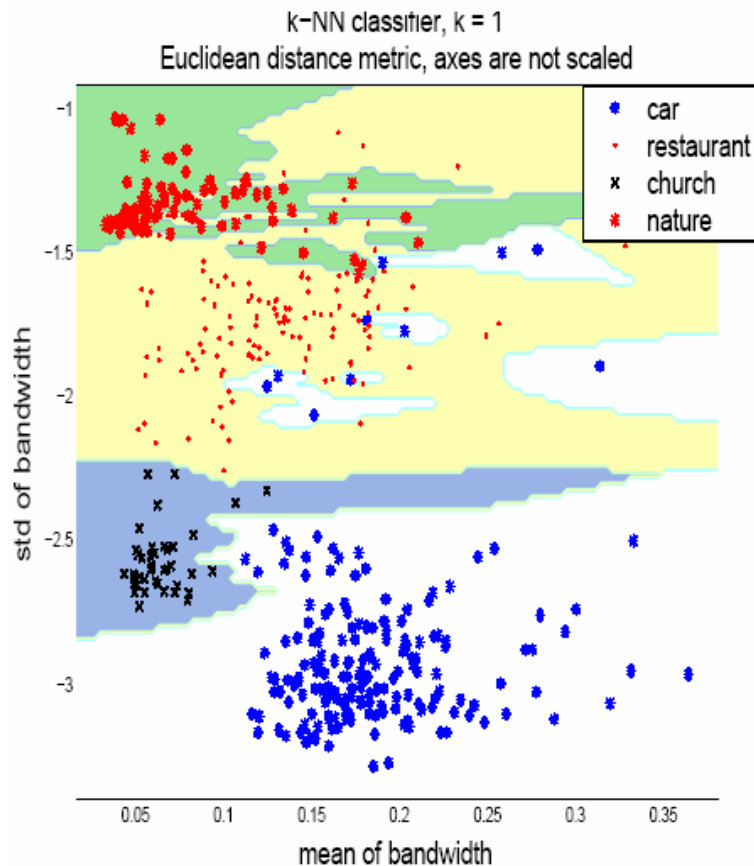
where C is the covariance matrix of training data.

- Mahalanobis distance D_M is generally a good choice
- Comment: k -NN is a decent classifier yet very easy to implement

Classification by distance functions

Example: decision boundaries for 1-NN classifier

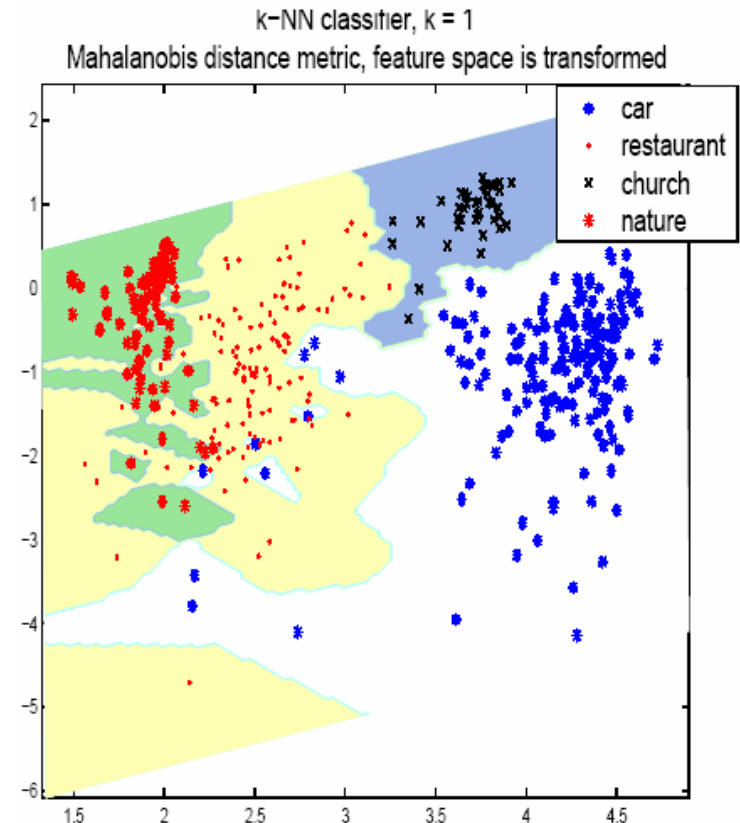
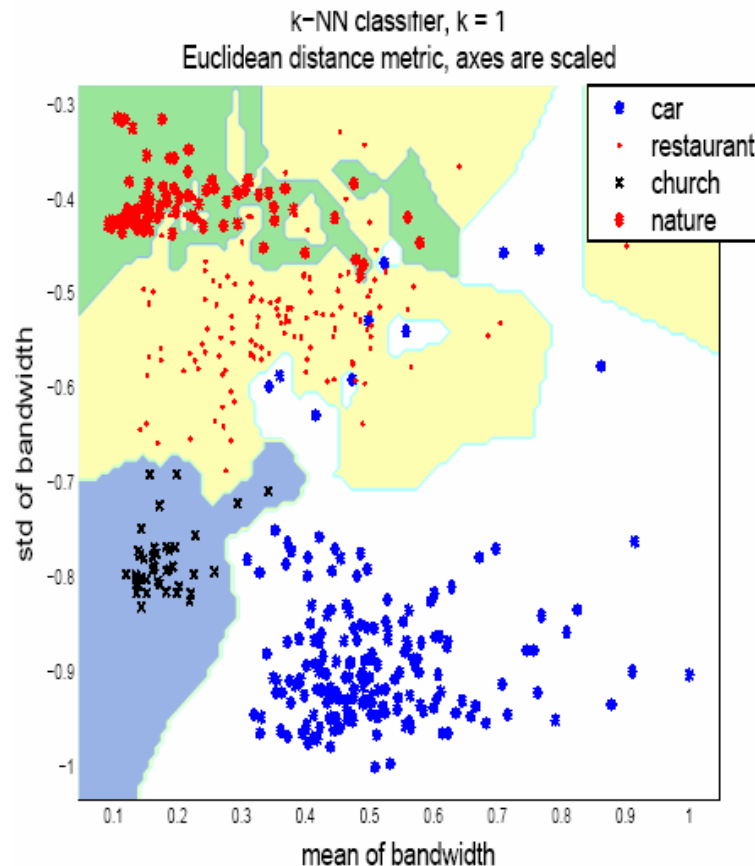
- Left: Euclidean distance, axes not scaled
→ undesired weighting (elongated horizontal contours)
- Right: Euclidean distance, axes scaled [[Peltonen, MSc thesis, 2001](#)]



Classification by distance functions

Example: decision boundaries for 1-NN classifier

- Left: Euclidean distance, axes scaled
- Right: Mahalanobis distance scales the axes and rotates the feature space to decorrelates the features [[Peltonen, MSc thesis, 2001](#)]



4.2 Statistical classification

- Idea: interpret the feature vector \mathbf{x} as a random variable whose distribution depends on the class
- Bayes' formula – a *very* important tool:

$$P(\omega_i|\mathbf{x}) = \frac{P(\mathbf{x}|\omega_i)P(\omega_i)}{P(\mathbf{x})}$$

where $P(\omega_i|\mathbf{x})$ denotes the probability of class ω_i given an observed feature vector \mathbf{x}

- **Note:** we do not know $P(\omega_i|\mathbf{x})$, but $P(\mathbf{x}/\omega_i)$ can be estimated from the training data
- Moreover, since $P(\mathbf{x})$ is the same for all classes, we can perform classification based on (MAP classifier):

$$\omega_i = \arg \max_i \{P(\mathbf{x}|\omega_i)P(\omega_i)\}$$

- A remaining task is to parametrize and learn $P(\mathbf{x}/\omega_i)$ and to define (or learn) the prior probabilities $P(\omega_i)$

Statistical classification

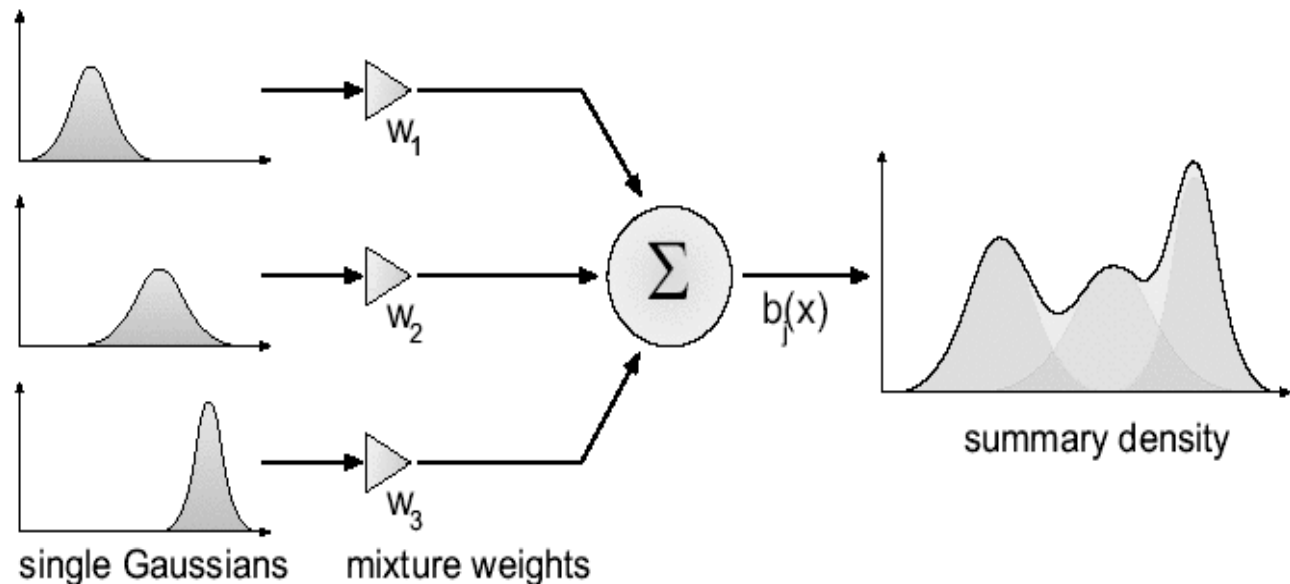
Gaussian mixture model (GMM)

- GMMs are very convenient for representing $P(\mathbf{x}|\omega_i)$, i.e., the multidimensional distribution of features for the class ω_i
- Weighted sum of multidimensional Gaussian distributions

$$P(\mathbf{x}|\omega_i) = \sum_{q=1}^Q w_{i,q} N(\mathbf{x}; \boldsymbol{\mu}_{i,q}, \Sigma_{i,q})$$

where $w_{i,q}$ are the weights and $N(\mathbf{x}, \boldsymbol{\mu}, \Sigma)$ is a Gaussian distribution

- **Figure:**
GMM in one dimension
[Heittola,
MSc thesis, 2004]



Statistical classification

Gaussian mixture model (GMM)

- GMMs can fit *any* distribution given a sufficient number of Gaussians
 - successful generalization requires limiting the number of components

$$P(\mathbf{x}|\omega_i) = \sum_{q=1}^Q w_{i,q} N(\mathbf{x}; \boldsymbol{\mu}_{i,q}, \Sigma_{i,q})$$

$$N(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

- GMM is a *parametric model*
 - parameters: weights $w_{i,q}$, means $\boldsymbol{\mu}_{i,q}$, covariance matrices $\Sigma_{i,q}$
 - diagonal covariance matrices can be used if features are decorrelated
→ *much less* parameters
- Parameters of the GMM of each class are estimated to maximize $P(\mathbf{x}|\omega_i)$, i.e., probability of the training data for each class ω_i
 - iterative EM algorithm
- Toolboxes: insert training data → get the model → use to classify new

Statistical classification

MAP classification

- After having learned $P(\mathbf{x}/\omega_i)$ and $P(\omega_i)$ for each class, maximum *a posteriori* classification of a new instance \mathbf{x} :

$$\omega_i = \arg \max_i \{P(\mathbf{x}|\omega_i)P(\omega_i)\}$$

- Usually we have a sequence of feature vectors $\mathbf{x}_{1:T}$ and

$$\omega_i = \arg \max_i \left\{ \prod_{t=1}^T P(\mathbf{x}_t|\omega_i)P(\omega_i) \right\}$$

since logarithm is order-preserving we can maximize

$$\begin{aligned} \omega_i &= \arg \max_i \left\{ \log \left(\prod_{t=1}^T P(\mathbf{x}_t|\omega_i)P(\omega_i) \right) \right\} \\ &= \arg \max_i \left\{ \sum_{t=1}^T \log [P(\mathbf{x}_t|\omega_i)P(\omega_i)] \right\} \end{aligned}$$

Statistical classification

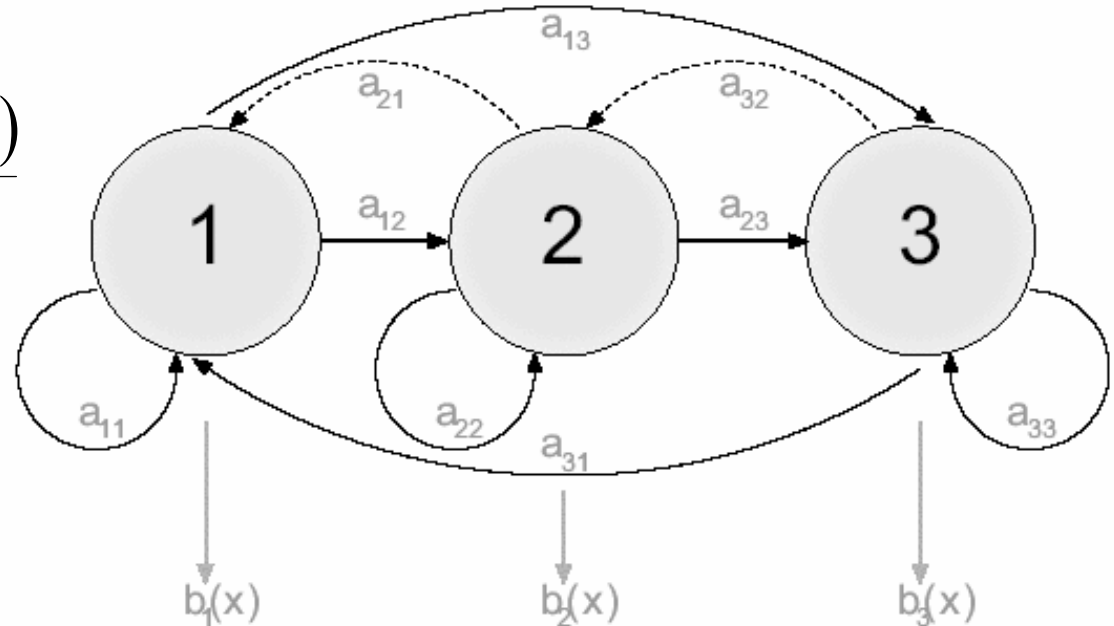
Hidden Markov model (HMM)

- HMMs account for the time-varying nature of sounds
 - probability of **observation** x in state j : typically GMM model $b_j(x)$
 - transition probabilities btw states: matrix A with $a_{ij} = P(q_t=i/q_{t-1}=j)$
 - the **state variable is hidden**, and usually not of interest in classification
- Train own HMM for each class
- HMMs allow computing

$$P(\omega_i | \mathbf{x}_{1:T}) = \frac{P(\mathbf{x}_{1:T} | \omega_i) P(\omega_i)}{P(\mathbf{x}_{1:T})}$$

where $\mathbf{x}_{1:T}$ is
an observation
sequence

- Figure: [Heittola,
MSc thesis, 2004]



Statistical classification

Hidden Markov model (HMM)

- Hidden Markov models are a generalization over GMMs
 - GMM = HMM with only one state
- HMMs do not necessarily perform better than GMMs
 - *only if some temporal information is truly learned to the state transition probability matrix, then HMMs are useful* → e.g. speech
- Example: genre classification, general audio classification
 - using GMMs: (1) extract feature vectors from training data and (2) model the distribution of the features with a GMM
 - using HMMs: extract features and train a HMM for each class
 - my subjective experience: certain features are characteristic to certain genres and can be learned BUT the temporal order of the feature vectors is often very irregular → difficult to learn → transition probabilities do not contain useful information → GMMs perform almost equally well