Pattern Recognition

Michael Prinzinger

29. November 2006

Inhaltsverzeichnis

Preface

This script is intended to be a summary for students of Pattern Recognition. It tries to capture all the topics discussed in lectures on Pattern Recognition I & II. It often attempts to make approaches different to those of the lectures. Hopefully these approaches might be more suitable for students and enrich the topics with examples and illustrations. It is not, however, a complete preparation for the diplomacy/master exam in Pattern Recognition, but rather a repetition, a summary and a source for alternative explanations. You additionally should read the books recommended in the lecture, the script of your professor and above all regularly attend to the lectures.

Extra Chapters

Basic Methods

In addition to the big topics of the field, there is a chapter dedicated to typical methods and terms used quite often in Pattern Recognition, like ML-Estimation or the Newton Method.

FAQ

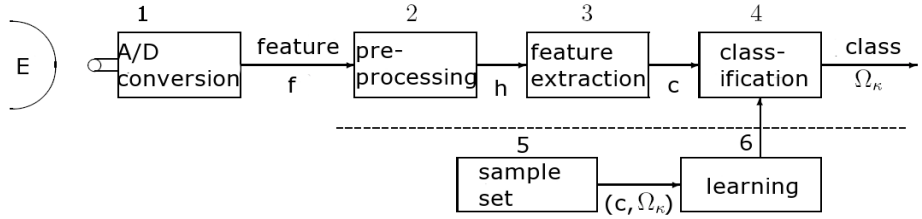
Last but not least there's a section with frequently asked questions. I think reading this part especially helps to get a good feeling and understanding of the big picture in Pattern Recognition.

Notation

section Sections enclosed with two stars are to be seen as an optional extension and probably not necessary for the exam.

Copyright

The Big Picture



1. A/D Conversion: signal f
2. Normalization: normalized signal h
3. Feature Extraction: feature vector $\vec{c}$
4. Classification: class number Ω_λ
5. (Labelled) Sample set: $(\vec{c}, \Omega_k)$
6. Learning / Training

1 A/D Conversion

Before we can deal with the actual classification of features, we have to get a digital representation of the analogue world data in our computer. In principle we have to store several equidistant sample evaluations of the signal and build a function out of them approximating the real signal als close as possible.

1.1 Nyquist Sampling-Theorem

Nyquist's Theorem tells us, under which conditions a digital signal can be reconstructed without loss of information. The first version says that the frequence of the sampling signal f_s must be at least twice as high as the maxmial occuring frequency f_H of the signal to sample f:

$$f_s > 2 \cdot f_H$$

The second version uses the bandwith B of the signal x (Bandlimit $[-B_x, B_x]$) effectively saying the same thing.

$$\Delta x < \frac{1}{2B_x}$$

$2B$ is also called the **Nyquist Rate**.

Abbildung 1: Proof Of The Nyquist Theorem

1. compute $f(x)$ given FT^{-1} (inverse integral).
2. Think of $F(\xi)$ as a periodic function continued by thought (because in reality we only look at one period) and develop $F(\xi)$ as Fourier Series.

Remarks:

- Because of the bandlimit, a development of the Fourier Series is possible. Determine it's coefficients.
- Insert $F(\xi)$ in the inverse integral of $f(x)$
- Write the result as a sum:

$$f(x) = \sum_{j=-\infty}^{\infty} f_j \frac{\sin\left[2\pi B_x (x - j\Delta x)\right]}{2\pi B_x (x - j\Delta x)}$$

- A complete reconstruction of the signal $f(x)$ requires $\Delta x = \frac{1}{2B_x}$ sample steps and ∞ many sample values (see the sum above).

Reconstruction

If a signal was sampled in respect to the Nyquist Theorem, it can be reconstructed by interpolating f as an infinite series (e.g. Fourier Series) with the **sinc function**:

$$f(x) = \sum_{j=-\infty}^{\infty} f_j \cdot \operatorname{sinc}(\pi(x-j)) = \sum_{j=-\infty}^{\infty} f_j \cdot \frac{\sin(\pi(x-j))}{\pi(x-j)}$$

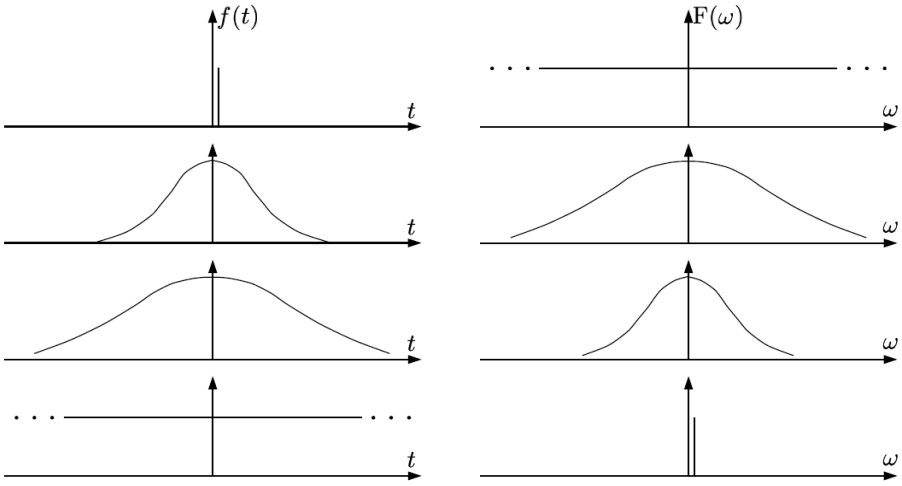
Space/Time Frequency Trade-Off

This exact reconstruction is however not in our reach. This is because a signal is either limited in the spatial resp. time domain or in the frequency resp. Fourier domain. This corresponds to a trade-off between storage requirements and computational efficiency and precision, i.e. the number of sample values. This trade-off can be formulated as

$$\Delta x \Delta \xi \geq \frac{1}{4\pi}$$

and illustrated as

Abbildung 2: Time/Space Frequency Trade-Off



The more narrow a function in the spatial domain, the broader it becomes in the frequency domain and vice versa.

1.2 Quantization

Quantization is another term for the more colloquial "rounding of numbers". A quantization function is defined as

$$Q(x) = \frac{\lfloor 10^{m-1}x + 0.5 \rfloor}{10^{m-1}}$$

where m is the level of precision.

Example

Rounding to the next integer corresponds to $m = 1$.
For example we will round $x = 4.564$:

$$Q(4.564) = \frac{\lfloor 4.564 + 0.5 \rfloor}{1} = 5$$

Rounding with floating point precision 1, $m = 2$

$$Q(4.564) = \frac{\lfloor 10 \cdot 4.564 + 0.5 \rfloor}{10} = \frac{\lfloor 46.14 \rfloor}{10} = 4.6$$

Signal-To-Noise-Ratio (SNR)

No matter how big we choose m, we will always loose precision quantizing a signal. After all that is what we want, since we do not have unlimited precision in computer architecture. Shanon saw this loss of precision as a kind of noise and applied the Signal-To-Noise-Ratio from information theory to the quantization of signals:

signal-to-noise-ratio

$$r = \frac{E\left\{f_j^2\right\}}{E\left\{n_j^2\right\}} dB$$

where f_j are the sample values and n_j the noise ($n_j = f_j - \hat{f}_j$, where $\hat{f}_j$ is f_j after quantization).

It can be prooven, that for every Bit of quantization, the SNR improves by 6 dB (x dB $:= 10 \log 10\,(x)$).

Abbildung 3: Proof For The Signal-To-Noise-Ratio Improvement
For this proof we assume 8 quantization steps.
$E\left\{n_j^2\right\} = \int_{-\frac{s}{2}}^{+\frac{s}{2}} \frac{1}{s} \cdot n^2 dn = \frac{s^2}{12}$
$E\left\{f_j^2\right\} = \sigma_f^2$
$s = \frac{8\sigma_f}{2^B}$ for $f_{\min} = -4\sigma_f$ and $f_{\max} = 4\sigma_f$

Quantization Error

For optimization and an objective function it is necessary to have a closed form for the average quadratic error that occurs during quantization:

$$\mathrm{ERR} = \sum_{\nu=1}^{L} \int_{a_\nu}^{a_{\nu+1}} (f - b_\nu)^2 \, p(f) \, df$$

where L is the number of quantization steps and a_ν, b_ν are determined by the assignment of sample values a_ν to quantization values b_ν:

$$a_v = \frac{b_{\nu-1} + b_\nu}{2}; \nu = 2, 3, \ldots L = 2^B$$

$$b_\nu = \frac{\int_{a_\nu}^{a_{\nu+1}} f \cdot p(f)\, df}{\int_{a_\nu}^{a_{\nu+1}} p(f)\, df}; \nu = 1, 2, 3, \ldots L$$

Application

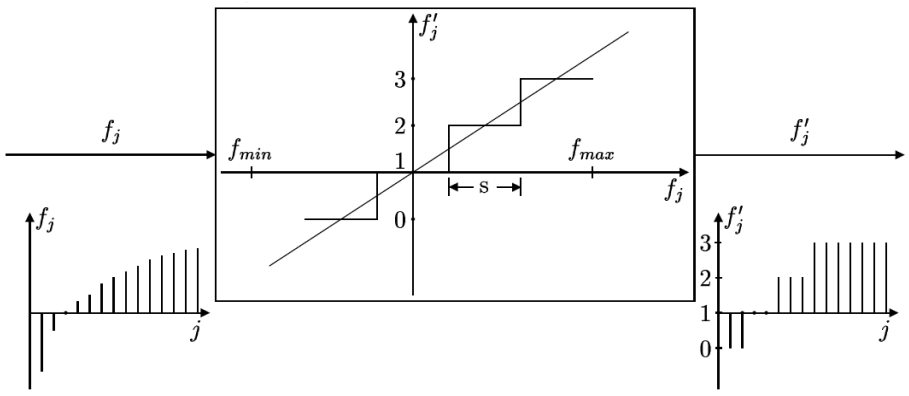
speech 8 Bit

pictures 8 to 12 Bit

audio/music 16 Bit

1.3 Characteristic Curve

Abbildung 4: Characteristic Curve



At the bottom left corner you can see the sample values of the original signal. The middle corresponds to the process of quantization. The line averaging the quantization is called **Characteristic Curve**. In this case it is linear, but it doesn't have to be.
On the right you can see the signal resulting after quantization has been performed.

The characteristic curve of the quantization is optimal, if the function values are uniformly distributed.

$p(f)$ is constant, if $f(x)$ is uniform. Assuming uniform distribution, b_ν becomes:

$$b_\nu = \frac{\left[\frac{1}{2} \cdot f^2 \cdot p\right]_{a_\nu}^{a_{\nu+1}}}{[f \cdot p]_{a_\nu}^{a_{\nu+1}}} = \frac{\frac{1}{2} \cdot p\left(a_{\nu+1}^2 - a_\nu^2\right)}{p(a_{\nu+1} - a_\nu)} = -\frac{1}{2} \cdot p(a_{\nu+1} + a_\nu)$$

Now our aim is of course to minimize the quantization error:

$$\mathrm{ERR} = \sum_{\nu=1}^{L} \int_{a_\nu}^{a_{\nu+1}} (f - b_\nu)^2 \, p(f) \, df$$

Due to the uniform distribution assumption, p becomes

$$p(n) = \begin{cases} \frac{1}{s} & \text{for } -\frac{s}{2} \leq n \leq \frac{s}{2} \\ 0 & \text{otherwise} \end{cases}$$

and the variance σ_f becomes

$$E\{n_j^2\} = \int_{-\frac{s}{2}}^{+\frac{s}{2}} \frac{1}{s} \cdot n^2 dn = \frac{s^2}{12}$$

$$\sigma_f = \sqrt{E\{f_j^2\}}$$

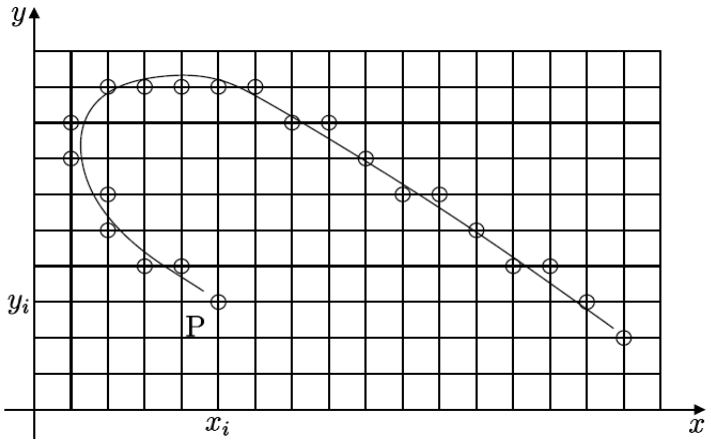
Abbildung 5: Proof For An Optimal Quantization Curve
compute the partial derivatives of ERR for a_ν and b_ν and solve them for 0.

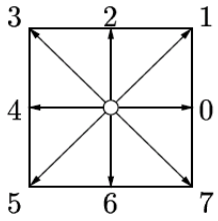
1.4 Alternatives

Instead of quantizing single sample values, as we did, other quantization methods were proposed. In general the choice of the quantization depends on the application, the features that will be extracted and the classification. So no method is superior to the others. Other methods for quantization are:

- **Vector Quantization**: Instead of quantizing single sample values, we take a vector of sample values and deal with the whole vector.
- **Run-Length Encoding**: This effectively means to represent binary black/white images as a series of 0/1 values indicating the color value. This allows for a very compact representation of images.
- **Chain Coding**: For images that only contain a single distinct feature, like a curve, a graph or a line, we can store a chain of direction values (see Figure below) following the course of the curve and later restore this curve on a solely white background.

Abbildung 6: Chain Coding





2 Preprocessing

The signal is on the computer. But the signal can not yet be used for classification, since it usually:

- contains noise and superflous data
- is too large
- is not yet adapted to / suitable for the application
- differs too much from training samples (it is not normalized)

Therefore depending on the application, we perform two steps before proceeding.

1. Filtering: We try to apply filters to remove the noise and/or unnecessary information.
2. Normalization: We try to find a normalized form of our samples, and normalize incoming signals to this form.

2.1 Filter

A filter is usually applied to remove noise from a signal. Another application is to remove unwanted data, or to scale down signals, like high- or low-pass filters do. Apart from that a general improvement of quality or the emphasis of certain features can be the intention when applying filters.

2.1.1 Binarization

The goal of Binarization is to reduce the number of values of a signal, usually of a picture, to two. With this procedure much information gets lost, but the signal becomes much smaller and noise is binarized away.

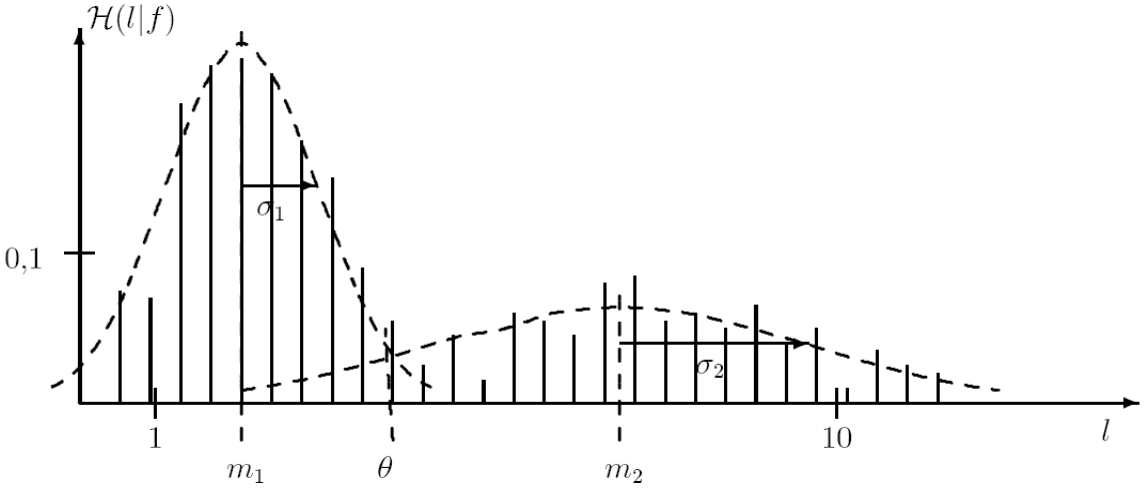
Iterative Clustering

Algorithm 1 Iterative Clustering

1. Randomly choose two vectors as representatives v_1 and v_2.
2. Choose a metric and associate each vector either to v_1 or to v_2 using an Nearest Neighbour classifier (see 4.4.3).
3. Compute the mean vectors μ for both vector sets and set them as the new representatives: $v_1 = \mu_1$ and $v_2 = \mu_2$.
4. ↻ until the resulting sets do not change anymore.

Intersection Of Two Gaussian Distributions

Abbildung 7: Intersection of two gaussian distributions (binarization)



A grey value histogram is approximated by the sum of two gaussian distributions.

The idea is to approximate a histogram of values of grey with a sum of two guassian distributions:

$$p(f) = \alpha\mathcal{N}\left(f, \mu_1, \sigma_1^2\right) + (1-\alpha)\mathcal{N}\left(f, \mu_2, \sigma_2^2\right)$$

$$p(f) = \alpha\frac{1}{\sigma_1\sqrt{2\pi}}e^{\left(-\frac{1}{2}\left(\frac{f-\mu_1}{\sigma_1}\right)^2\right)} + (1-\alpha)\frac{1}{\sigma_2\sqrt{2\pi}}e^{\left(-\frac{1}{2}\left(\frac{f-\mu_2}{\sigma_2}\right)^2\right)}$$

where α represents a weight for scaling. The weights must sum up to 1 to fullfill the stochastic criterion (see 10.3), therefore we choose $(1-\alpha)$ for the second weight.

The weight α and the density parameters $\mu_1, \mu_2, \sigma_1, \sigma_2$ can be estimated by Maximum Likelihood Estimation (see 8.1).

Application

- edge detection
- optical character recognition
- object detection (decide for every pixel, if it belongs to the object or background)

2.1.2 Filters And Convolution

A good idea is to make use of the convolution theorem (see 9.1) to apply a linear filter, which can be convoluted with the original signal to remove noise:

$$f = s \oplus g + \eta$$

where f is the signal, g is a linear system to convolute with, s is an ideal signal without noise and η an additive noise term. Our goal is the ideal signal s. We approximate it by looking for a convolution term γ with

$$f \oplus \gamma = \hat{s}$$

and $\hat{s} \simeq s$.

To make efficient use of the convolution theorem we use Fourier Transforms for applying such filter kernels (see 9.5).

2.1.3 Bluring (Box Filter, Mittelwertfilter)

Abbildung 8: Bluring

For every value of the signal: Sum over the value itself and it's neighbours and divide by the number of values used. E.g. for a 1D signal, that gives:

$$f_i = \frac{f_{i+1} + f_i + f_{i-1}}{3}$$

This is a intuitive and efficient method to defeat noise, yet the disadvantages are that everything gets blurred: transitions and striking features get lost.

2.1.4 Gaussian Filter

A gaussian filter is a weighted Box Filter (Mittelwertfilter). It works by moving a gaussian bell over the signal. For preseverving transitions and striking features, we could for example weight the original value twice as much and still reduce the noise significantly:

$$f_i = \frac{f_{i+1} + 2 \cdot f_i + f_{i-1}}{4}$$

The gaussian filter and the Box Filter are filter kernels, that are best applied using convolution (see above).

2.1.5 Kausal Filter

Kausal means that the system response does not depend on a time before the input.

Kausal filter are recursive filters:

$$h_j = \sum_{\mu=0}^{m-1} a_\mu f_{j-\mu} - \sum_{\mu=1}^{n} b_\mu h_{j-\mu}$$

estimate a_μ and b_μ to approximate the convolution term γ.

2.1.6 Homomorph Filter

If the noise is not additive, as has been assume above, we have to apply a transformation T:

$$h = T_c^{-1} \{T_L \{T_c \{f\}\}\}$$

Algorithm 2 homomorph transformation

1. transform non-additive operation into an additive one by applying T_c.
2. apply the linear/kausal filter T_L.
3. apply the inverse of the transformation T_c^{-1} to transform the signal back.

For example T can be an approximation of the signal by sums (e.g. Taylor Series).

2.1.7 Erosion

Erosion is a "shrinking" of the signal. Assuming a 2D signal, i.e. a picture, each pixel is set to the minimum of it's neighbourhood:

$$\begin{matrix} f_1 & f_2 & f_3 \\ f_3 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{matrix} \rightarrow f_5 = \min\{f_1, \ldots, f_9\}$$

2.1.8 Dilatation

Dilatation acts just like Erosion, but instead of shrinking the signal, Dilatation "blows it up". Each pixel is thus set to the maximum of it's neighbourhood:

$$\begin{array}{ccc} f_1 & f_2 & f_3 \\ f_3 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{array} \rightarrow f_5 = \max\{f_1, \ldots, f_9\}$$

2.1.9 Edge Detection

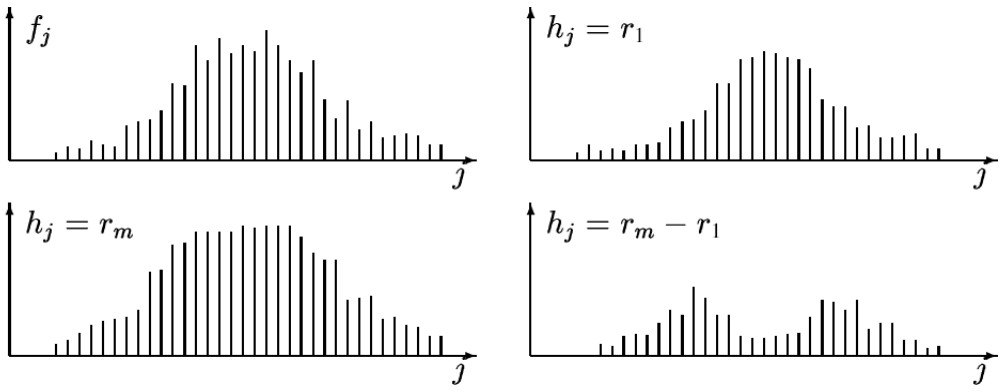
Combining Erosion and Dilatation, we get an efficient method for detecting edges:

Algorithm 3 Edge Detection With Erosion And Dilatation

given signal/picture f

1. Dilatation $(f) = \hat{f}$ blows the picture up
2. Erosion $(f) = f'$ cuts the picture (comprimizes it)
3. Computing $\hat{f} - f'$ leaves only the edges, because the edges are the only thing Dilatation and Erosion differ in.

Abbildung 9: Erosion and Dilatation



This first picture (top left) shows the sample values of the signal $\vec{f}$, whose values are ordered by size $r_1 \leq r_2 \leq \cdots \leq r_m$.
The second picture (top right) shows the result of applying Erosion to the signal.
The third picture (bottom left) shows the result of applying Dilatation to the signal.
The final picture (bottom right) shows the result of edge detection by consecutively applying first Dilatation and then Erosion.

2.1.10 Median

The Median is similar to the Box Filter (Mittelwertfilter), but lacks the disadvantages of it's 1D correspondancy. Each pixel of the picture is set to the medium value of it's neighbourhood:

$$\begin{array}{lll} f_1 & f_2 & f_3 \\ f_3 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{array} \rightarrow f_5 = \text{medium}\{f_1, \ldots, f_9\}$$

By this slight change (taking the medium and not the mean), the Median has the following properties:

- preserves edges
- removes noise
- efficiently implemented using dynamical programming (reuse of internim results)

2.1.11 Sobel Filter

The sobel filer calculates the gradient of the image intensity at each point. This gives the level of change in the picture, what (high values) excatly corresponds to the edges.

The gradient is vertically and horizontally approximated by the following filter kernels:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \cdot A$$

for x direction and

$$G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \cdot A$$

for y direction, where A is a approximation of the signal f.

Combining both gradients we get

$$G = \sqrt{G_x^2 + G_y^2}$$

The direction of the gradient (important to find the edges) is

$$\arctan\left(\frac{G_y}{G_x}\right)$$

2.1.12 La Place Filter

The La Place filter subtracts for each pixel x the brightness of each neighbouring pixel from the central pixel x. Then it normalizies every pixel to avoid the *white eagle on white background phenomena* (see 8.23).

This procedure means that regions with a constant intensity are reduced to a color value of zero, whilst a high variance of intensity within a region results in a larger color value. For normalization usually a grey value of 128 is added to each pixel for better visibility. The La Place kernel looks like that:

$$\text{La Place: } \begin{pmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

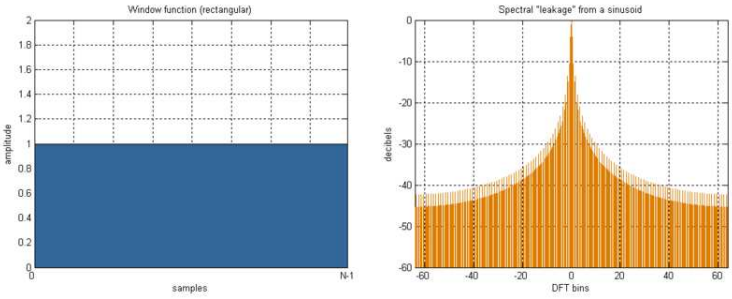
2.1.13 Window Functions

Another kind of filters are window functions that are moved across the signal.

Types Of Window Functions

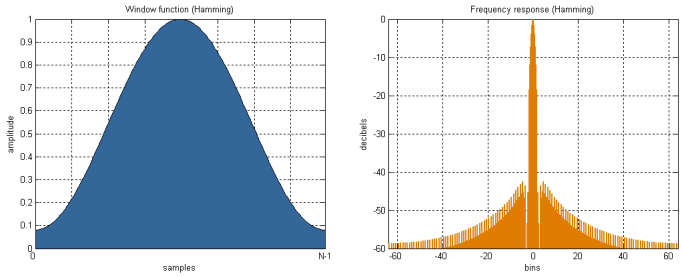
Rectangle $a_\nu = 1$

Abbildung 10: Rectangle Window Function



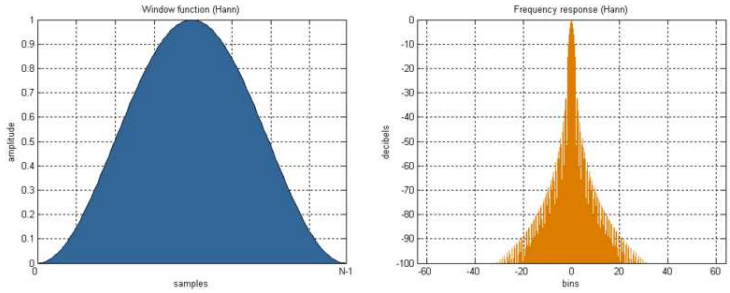
Hamming $a_\nu = 0.54 - 0.46 \cdot \cos \frac{2\pi}{M-1}$

Abbildung 11: Hamming Window Function



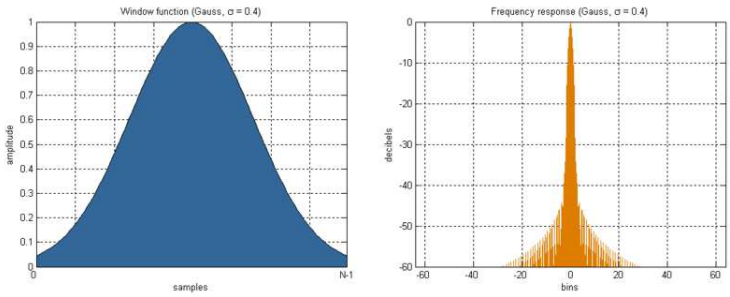
Hanning $a_\nu = 0.5\left(1 - \cos\frac{2\pi}{M-1}\right)$

Abbildung 12: Hanning Window Function



Gaus $a_\nu = \frac{1}{\sqrt{2\pi}\cdot\sigma} \cdot e^{-\frac{1}{2}\cdot\frac{x^2}{\sigma^2}}$

Abbildung 13: Gauss Window Function

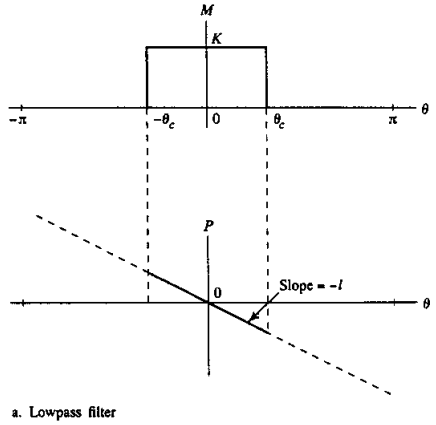


A problem of window functions is the loss of time/frequency information, since their effects are local. Thus they are not suitable for e.g. digital image processing.

2.1.14 Low-Pass, High-Pass, Band-Pass Filter

low-pass A low-pass filter cuts off frequencies below a certain threshold and only leaves through high frequencies.
We have an ideal **low-pass** filter if the Fourier Transform corresponds to a rectangle. It cuts edges off in a very clean way.

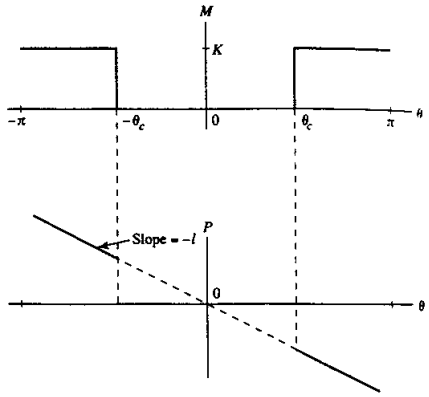
Abbildung 14: a) Ideal Low-Pass Filter



a. Lowpass filter

high-pass A **high-pass** filter cuts off frequencies above a certain threshold and only lets through low frequencies.

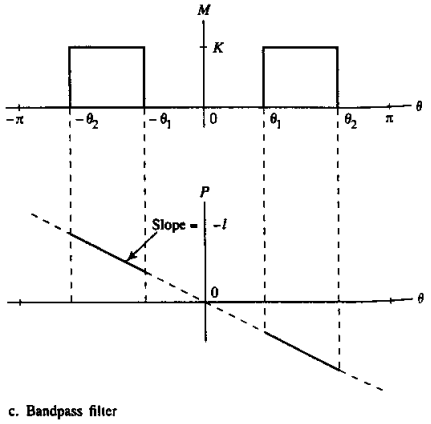
Abbildung 15: b) Ideal High-Pass Filter



b. Highpass filter

band-pass A **band-pass** filter has thresholds for both high and low frequencies.

Abbildung 16: c) Ideal Band-Pass Filter



Both low- and high-pass filters halve the band-width of the signal, that means, according to the Sampling Theorem (see 1.1), we only need half as many sample points to represent the signal.

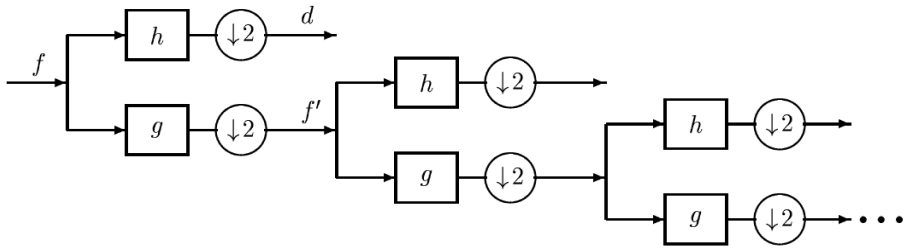
2.1.15 Resolution Hierachy

Since edge detection and other algorithms take quite long to process pictures having a high resolution, it might be better to reduce the resolution of the image and process the smaller version. Low- and high-pass filters can be used to downsample a picture in this way. Yet every level of resolution lost, could mean the loss of essential information. Therefore the idea of resolution hierachy is to downsample the picture, yet preserve the possibility to fall back to a higher resolution without lots of space or time usage.

The idea is to use a low-pass filter g to downsample a picture to f' while parallely applying a high-pass filter h, which returns excatly the part, the low-pass filter downsampled away d. We keep this part and use it later on in case we have to fall back to a higher level of resolution.

Hierachy

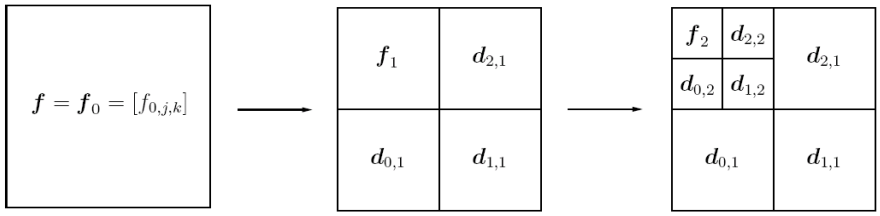
Abbildung 17: Resolution Hierachy (Principle)



Partition

This partition can be done very efficiently using wavlets (see 3.1.11):

Abbildung 18: Resolution Hierachy (Wavelet Partition)



Reconstruction

The reconstruction is rather straight forward, just add the removed high-pass parts back to the signal using again wavelets. Since there will be some rounding errors, just set values close to zero to zero and you should not run into problems.

Note The wavelet theory (see 3.1.11) has prooven that this method even works with non ideal filters (see 2.1.14).

2.2 Normalization

Normalization is not optional but compulsory.

Bear in mind that normalization should always aim towards the maxims of pattern recognition (see 8.25). Apart from that common normalization affects:

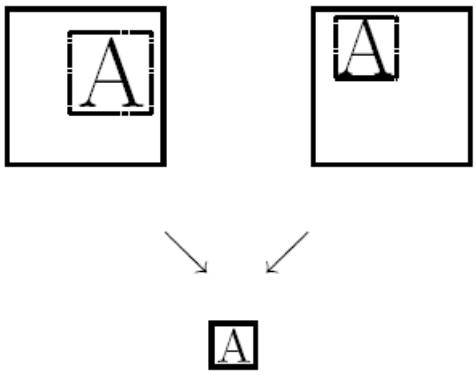
- rotation (pay attention to the 6/9 problem)
- location
- lighting

- intensity
- length (speech)
- size

In the follwing section the initial pattern is denoted by $f(x, y)$ and the transformed one by $h(x', y')$.

2.2.1 Scaling

Abbildung 19: Scaling Of The Letter A



In this picture a scanned image of the letter 'A' is scaled down and translated to a normalized reference version.

Scaling a picture can be done in three steps:

Algorithm 4 Scaling

1. Get the scaling factor using intercept theorems.
2. Resample according to the sampling theorem (1.1) and the sinc function:

$$f(x) = \sum_{j=-\infty}^{\infty} f_j \operatorname{sinc}(\pi(x - j))$$

3. Interpolate:

$$f(x, y) = \sum_{j=1}^{N} \sum_{k=1}^{N} f_{jk} g(x - j, y - k)$$

where $g(x - j, y - k)$ is an interpolation function.

Interpolation can be a simple Nearest Neighbour Interpolation or bilinear interpolation with a weighted mid point (the close a point lies to the interpolation point, the more it contributes). Bilinear interpoltaion usually works just fine.

Application

- **Letter Recognition**: Find the closest bounding box around the letter and scale this rectangle.
- **Speech Recognition**: Try to find start and ending points of words (difficult to locate! requires at least word lexica) and scale this interval.

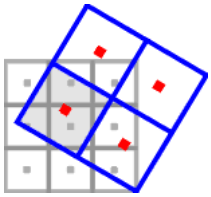
2.2.2 Translation

Usually scaling includes translation, if not it is essentially done in the same way: Get some kind of bounding box or interval and move it to the origin.

2.2.3 Rotation

Rotation works just like scaling, but remember always to sample and resample the original picture and not the one that is being rotated.

Abbildung 20: Rotation without resampling



As you can see in the image above, a rotation whithout resampling, where the function values are taken from the image, that is being rotated, leads to wrong values.

2.2.4 Energy

The energy of a pattern is for example the intensity of an image or the wave enegery of sound waves. It is defined as:

$$A_j = \sum_{\nu=1}^{M} |\omega_\nu f_{j-\nu}|$$

where M is the number of sample values and ω_ν is a window function (see 2.1.13) to blind out important signal values.

The normalization is then performed by:

$$h_k = \frac{f_k}{A_j}$$

for 1D or

$$h_k = \frac{f_k}{\sqrt{A_j}}$$

for 2D.

2.2.5 Moments

Mathematical moments (see 10.6) can be excelentely used for normalization.

Center Of Gravity (german Schwerpunkt)

We take the moments: $x = \frac{x^p}{x_0}$ and $y = \frac{y^q}{y_0}$ and get the center of gravity g by:

$$g_x = \frac{\int_1^n x \cdot f(x,y)\, dxdy}{\int_1^n f(x,y)\, dxdy}$$

$$g_y = \frac{\int_1^n y \cdot f(x,y)\, dxdy}{\int_1^n f(x,y)\, dxdy}$$

Translation

We translate by $x_s = \frac{m_{10}}{m_{00}}$ and $y_s = \frac{m_{01}}{m_{00}}$.

Then the resulting translated points are simply: $x' = x - x_s$ and $y' = y - y_s$. And finally

$$h(x', y') = \frac{f(x,y)}{m_{00}}$$

The central moments must have the following values: $\mu_{00} = 1$ and $\mu_{01} = \mu_{10} = 0$.

Scaling

Signals can be scaled using the second moments. We replace $f(x,y)$ by $h(x', y')$ and choose the focus point r as $r = \sqrt{m_{02} + m_{20}}$ and get the scaled points: $x' = \frac{x}{r}$ and $y' = \frac{y}{r}$. And again

$$h(x', y') = r^2 f(x,y)$$

The second central moments must fullfill the postulate: $\mu_{20} + \mu_{02} = 1$ where $\mu_{20} > \mu_{02}$.

Rotation

We use a standard rotation matrix and get our rotated points with:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

where angle α is choosen as

$$\tan(2\alpha) = \frac{2 \cdot m_{11}}{m_{20} - m_{02}}$$

and h is simply

$$h(x', y') = f(x, y)$$

with central moment $\mu_{11} = 0$.

Reflection

Last but not least we want to reflect the signal and choose a constant $\beta \in \{1, -1\}$, such that $x' = \beta x$ and $y' = y$. That gives us for h

$$h(x', y') = f(x, y)$$

with central moment $\mu_{12} > 0$ and $\mu_{21} > 0$.

Summing up, after applying all this operations we have the following central moments:

$\mu_{00} = 1$, $\mu_{10} = \mu_{01} = \mu_{11} = 0$, $\mu_{20} + \mu_{02} = 1$ with $\mu_{20} > \mu_{02}$ and finally $\mu_{21} > 0$ and $\mu_{12} > 0$.

3 Feature Extraction

After normalization, we need to find features in signals, upon which we can make a classification.
Of course taking the whole signal as one single feature would be possible, but wouldn't work in most cases. Imagine for examples two pictures taken by a camera right after each other: Looking at them pixel by pixel, they already show such a great variance, that it could not be guaranteed that they will even be classified to the same class. Apart from that we would have an enourmous big search space and algorithms would need quite long for classification.

Therefore maxims for features are:

- take not too few and not too many single features (see 8.18)
- prefer features invariant to certain effects (e.g. invariance to rotation)
- look for features that are characteristic for the object you want to classify (e.g. LPC for speech 3.1.7, the radius for circles, the color for flowers and so on)
- look for features that are very similar within a class but very different to other classes' features (see 8.25)

3.1 Feature Computation

We distinguish three types of features:

1. heuristical features
2. spectral and cepstral features
3. analytical features

The first category describes merely heuristic features, i.e. quantities that have prooven to be good features in general, which though are not related to the signals, we want to classifiy, nor to our application. They most commonly are used because of a fast and easy computation, very valuable properties like invariances and simply because they have prooven to work in many cases very well.
The second category is still kind of heuristical, but instead of merely using informations from the direct signal we focus on the signal's spectrum and cepstrum (see 3.1.12).
The third category however uses analytical tools to find features suitable for a special kind of signals. The big disadvantage of this approach is, that it usually takes quite long to determine adapted features and is not always easy.

[1] Heuristic Features

3.1.1 Development To Orthogonal Bases

A first idea for features is to build an orthogonal basis to the subspace our signal f is located in and represent our signal as a linear combination of the orthogonal basis vectors $\vec{\varphi_0}, \ldots, \vec{\varphi_n}$ of this space. The features are then the factors of the linear combinations $c_0, \ldots, c_n$. The advantage is that this base is canonical, i.e. unique, which gives a good measure for comparison respectively classification.

For building an orthogonal basis we can apply Gram-Schmidts-Orthogonalization-Process (see 9.11).

$$\vec{f} = (\vec{\varphi_0}\vec{\varphi_1} \ldots \vec{\varphi_{n-1}}) \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix}$$

shuffling to get the features in front gives

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = (\vec{\varphi_0}\vec{\varphi_1} \ldots \vec{\varphi_{n-1}})^{-1} \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_{n-1} \end{pmatrix}$$

the matrix $(\vec{\varphi_0}\vec{\varphi_1} \ldots \vec{\varphi_{n-1}})$ is probably not invertable, therefore we use the Pseudo Inverse to get the above shuffling

$$\vec{c} = \left(A^T \cdot A\right)^{-1} \cdot A^T \cdot \vec{f}$$

Note Instead of an orthogonal basis, we can also build an orthonormal basis using Gram-Schmidts-Orthonormalization (see 9.12). The advantage is, that the basis and the linear combinations are further contracted. The disadvantage is that certain features could get normalized away by this. So you have to decide per application, which basis to take.

Properties

- heuristical method (not orientated on the real data)
- orthogonal / orthonormal

Pro/Contra

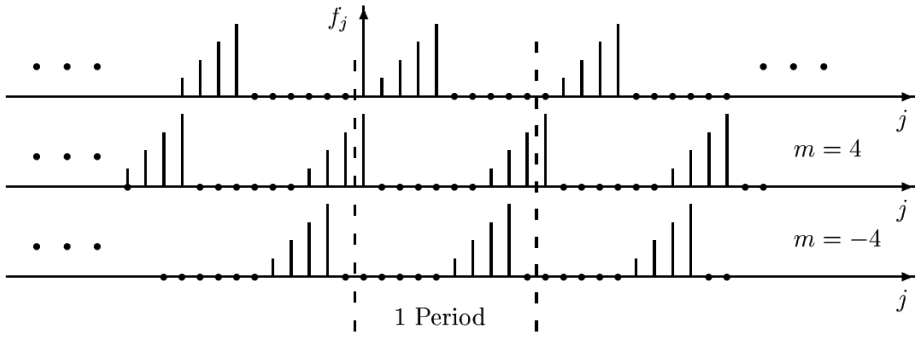
\+ the feature space is much smaller than the signal space, yet by using linear combination with orthogonal basis vectors, we do nothing else but a projection into the image space.

\+ the coefficients of orthogonal resp. orthonormal bases are unique.

3.1.2 Discrete Fourier Transform (DFT)

A different approach is, to transform the signal with discrete fourier transform into the frequency space. The big advantage of this transformation is, that it is invariant to translations (i.e. $f(\vec{x}) = f(\vec{x} + \vec{t})$).

Abbildung 21: Translation Invariance Of A Periodic Function



$$F = DFT\{f\}$$

the contionous form is

$$FT(\{f(x,y)\}) = F(\xi,\eta) = \int\int_{-\infty}^{\infty} f(x,y) \cdot \exp^{-i(\xi x + \eta y)}\, dxdy$$

and the discrete form

$$DFT(\{f(x,y)\}) = F(\xi,\eta) = \sum_{j=0}^{M_x-1} \sum_{k=0}^{M_y-1} f(j,k) \cdot \exp^{-i2\pi\left(\frac{\xi j}{M_x} + \frac{\eta k}{M_y}\right)}$$

where M is the number of sample values.

Looking closer, this is resembles an orthogonal basis $F(\xi) = \varphi_\xi^T \vec{f}$ where $\varphi_\xi^T \varphi_\eta = 0$.

Properties

- heuristical method (not orientated on the real data)

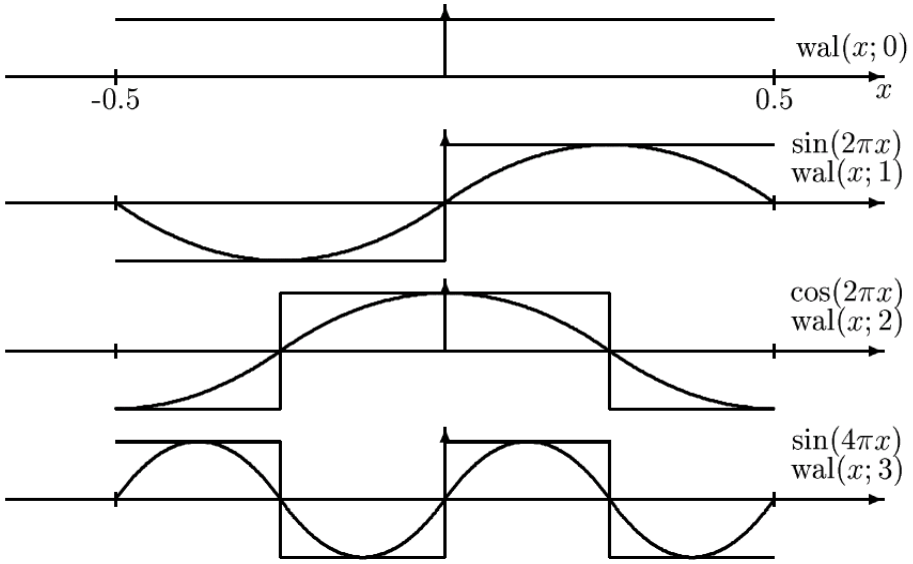
Pro/Contra

\+ translation invariant: $f(\vec{x}) = f(\vec{x} + \vec{t})$

3.1.3 Walsh Transformation

Remember Euler's formula $e^{ix} = \cos\alpha + i \cdot \sin\alpha$. In accordance to that the Walsh Transformation uses discrete even and odd fuctions, which approximate sin and cos, to represent a signal.

Abbildung 22: Walsh Transformation



some walsh functions in comparison to the approximated trigonometric functions

We define the walsh transformation as:

$$\text{wal}\,(x,0) = \begin{cases} 1 & \text{if } -\frac{1}{2} \le x \le \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{wal}\,(x,2j+p) = (-1)^{\frac{j}{2}+p}\left(\text{wal}\left(2\left(x+\frac{1}{4}\right),j\right) + (-1)^{j+p}\,\text{wal}\left(2\left(x-\frac{1}{4}\right),j\right)\right)$$

3.1.4 Walsh-Hadamard Transform

Hadamard matrices looks like this:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H_4 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

$$H_8 = \begin{pmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{pmatrix}$$

$$\vdots$$

Combined with the idea from Walsh, they can be used to efficiently compute features:

$$\vec{c} = H_M \vec{f} = \mathrm{WHT}\left\{\vec{f}\right\}$$

where $\vec{f}$ is a feature vector with M components. The inverse Walsh-Hadamard Transformation is accordingly:

$$\vec{f} = \frac{1}{M} H_M \cdot \vec{c} = \mathrm{WHT}^{-1}\left\{\vec{c}\right\}$$

The Walsh-Hadamard Transform requires only addition and subtraction.

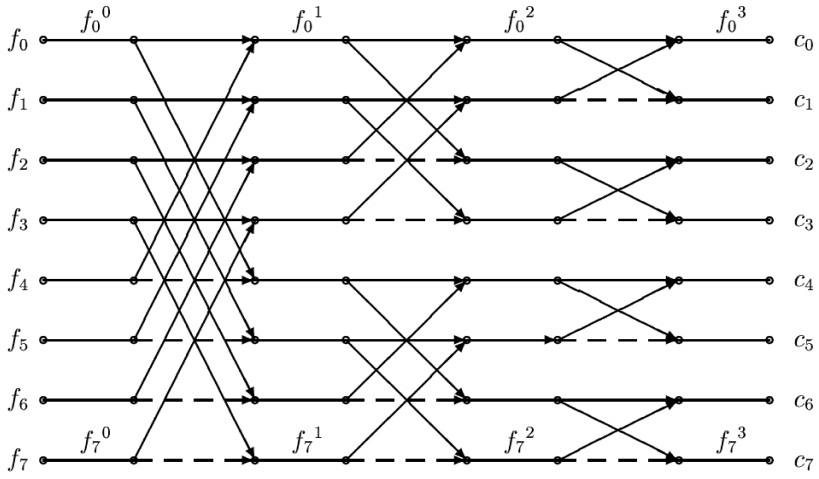
Properties

- $O\left(M^2\right)$

3.1.5 Fast Walsh-Hadamard Transform

This algorithm is very similar to the Fast Fourier Transform (see 9.5.4) and is also based on decomposing the the transform using factorization.

Algorithm 5 Fast Walsh-Hadamard Transform

1. Partition H_8 into four H_4 matrices and multiply them with the corresponding function values of f.
2. Partition the H_4 matrices further into 16 H_2 matrices.
3. 

   unbroken addition / multiplication with $(+1)$

   dashed subtraction / multiplication with (-1)

Example

$$\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_8 \end{pmatrix} = H_8 \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_8 \end{pmatrix}$$

is decomposed to

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = H_4 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} + H_4 \begin{pmatrix} f_5 \\ f_6 \\ f_7 \\ f_8 \end{pmatrix}$$

and

$$\begin{pmatrix} c_5 \\ c_6 \\ c_7 \\ c_8 \end{pmatrix} = H_4 \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} - H_4 \begin{pmatrix} f_5 \\ f_6 \\ f_7 \\ f_8 \end{pmatrix}$$

and so on.

Properties

- heuristical method (not orientated on the real data)
- $O(M \log M)$
- orthonormal:
  $\int_{-\pi}^{\pi} \cos(kx) \cdot \sin(jx)\, dx = 0$ if $j \neq 0$
  $\int_{-\frac{1}{2}}^{\frac{1}{2}} \text{wal}(x,j) \cdot \text{wal}(x,k) = \begin{cases} 0 & \text{if } j \neq k \\ 1 & \text{if } (j = k) \end{cases}$

Pro/Contra (Fast Walsh-Hadarmad Transform)

\+ very fast

\+ only operations are addition and subtraction

\+ only real values

\- the resulting order of the features is arbitrary

3.1.6 Haar Transformation

Haar Transformation is almost the same as the Walsh Transformation, it also uses 0/1 values. The difference is that instead of the Walsh basis functions, we define very simple Haar basis functions that includes scaling by including a time dependent component. We then approximate our signal by a linear combinations of Haar basis functions of the following type:

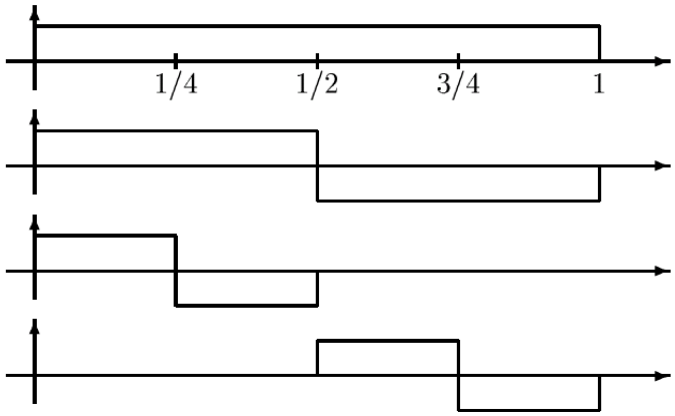
$$h_{00}(x) = \frac{1}{\sqrt{N}}$$

$$h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} +2^{\frac{p}{2}} & : \frac{q-1}{2^p} \leq x \leq \frac{q-\frac{1}{2}}{2^p} \\ -2^{\frac{p}{2}} & : \frac{q-\frac{1}{2}}{2^p} \leq x \leq \frac{q}{2^p} \\ 0 & : \text{otherwise for } x \in [0;1] \end{cases}$$

where p is the degree of the function and $q = \begin{cases} 0, 1 & : p = 0 \\ 2^0, 2^1, \ldots, 2^p & : p \neq 0 \end{cases}$.

Abbildung 23: Haar Basis Functions

p	q	k	h_{pq}
0	0	0	h_{00}
0	1	1	h_{01}
1	1	2	h_{11}
1	2	3	h_{12}



Forward Transformation

Algorithm 6 Haar Forward Transformation

---

1. Place the discrete signal perpendicular, to make it vektorwertig:

$$f_p = \left(\ldots, \begin{pmatrix} f_{-2} \\ f_{-1} \end{pmatrix}, \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}, \begin{pmatrix} f_2 \\ f_3 \end{pmatrix}, \ldots \right)$$

2. Multiply this function with the Haar-Transformation Matrix

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H \cdot f_p = \left(\ldots, \begin{pmatrix} s_{-1} \\ d_{-1} \end{pmatrix}, \begin{pmatrix} s_0 \\ d_0 \end{pmatrix}, \begin{pmatrix} s_1 \\ d_1 \end{pmatrix}, \ldots \right)$$

where $s_k = \frac{f_{2k+1}+f_{2k}}{\sqrt{2}}$ and $d_k = \frac{f_{2k+1}-f_{2k}}{\sqrt{2}}$

3.1.7 Linear Predictive Coding (LPC)

The goal of Linear Predictive Coding is to find a set of parameters $a_1 \ldots a_n$, which allow to predict the $n+1^{\text{th}}$ value as linear combination of the $1 \ldots n$ previous function values. Concerning feature computation, we can use the coefficients $a_1 \ldots a_n$ as feature vector.

$$\hat{f}_n = -\sum_{\mu=1}^{m} a_\mu f_{n-\mu}$$

The error of this approximation is given by

$$\mathrm{ERR} = \sum_{n=1}^{n} \left(f_n - \hat{f}_n \right)^2 = \sum_{\mu=0}^{m} \sum_{\nu=0}^{m} a_\mu a_\nu r_{|\mu-\nu|} = \sum_{\mu=0}^{m} a_\mu r_\mu$$

with

$$r_{|\mu-\nu|} = \sum_n f_n f_{n+|\mu-\nu|} = \sum_n f_{n-\mu} f_{n-\nu}$$

Algorithm 7 Linear Predictive Coding / Model Spectrum

LPC can be used to efficiently calculate a model spectrum:

1. First we calculate the LPC coefficients $\{a_0, \ldots, a_m\}$ (the convolution kernel)

$$\hat{f}_n = \sum_{k=n-m}^{n} a_k f_{n-k}$$

2. We apply Discrete Fourier Transfer to these coefficients

$$\mathrm{DFT}\,(a_0, \ldots, a_n) \rightarrow \mathrm{FT}\,(\vec{a})$$

3. Our model spectrum is

$$|\mathrm{FT}\,(a_0)| \ldots |\mathrm{FT}\,(a_n)|$$

LPC Coeffecients

The LPC coefficients can be computed using the Pseudo Inverse. We are starting with:

$$\begin{pmatrix} \hat{f}_2 \\ \hat{f}_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} f_1 f_0 f_{-1} \cdots f_{l-k} \\ \vdots \\ \vdots \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}$$

we call the midpart the matrix A:

$$\hat{f} = -A \cdot \vec{a}$$

Now we build the Pseudoinverse from A and shuffle:

$$A^T \hat{f} = -A^T \cdot A \cdot \vec{a}$$

$$-\vec{a} = \left(A^T A \right)^{-1} A^T \cdot \hat{f}$$

Properties

- practise has shown that LPC parameters are ideally suited for modelling the human speech tract.
- the error introduced above can be used as additional featrure c_{n+1}.
- further possibilities for additional featrures are coefficients of the DFT.

Practice

- for speech modelling, usually 10-15 parameters are sufficient (1 feature per 10-25 ms time frame)
- in general we take: sample frequency (in KHZ) + 4 or 5 parameters

3.1.8 Moments

In chapter two we used moments (see 10.6) for normalization, yet they also make very good features, since they show many invariances. For example the center of gravity is always the same, no matter the rotation or translation a object has. A central geometrical moment is defined as

continous $m_{pq} = \int \int_{-\infty}^{\infty} x^p y^q f(x,y)\, dxdy$

discrete $\mu_{pq} = \sum_{j=0}^{M-1} \sum_{k=0}^{M-1} (x_j - x_s)^p (y_k - y_s)^q f_{jk} \Delta x \Delta y$

where (x_s, y_s) is the center of gravity of the object.

Given this defintion, we can build features out of those moments:

[A] Central Moments

we choose the feature vector:
$c_1 = \mu_{20} + \mu_{02}$
$c_2 = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2$
because this feature vector has the following invaraiances:

- invariant to rotations
- invariant to translations

[B] Legendre Moments

Legendre Moments are defined as:

$$\lambda_{pq} = \frac{(2p+1)(2q+1)}{4} \int \int_{-1}^{+1} \mathcal{P}_{p(x)} \mathcal{P}_{q(y)} f(x,y)\, dxdy$$

where $\mathcal{P}$ is the so called Legendre Polynom. The advantage of these moments is, that there is an efficient recursive computation procedure for them.

[C] Zernike Moments

$$c_{pq} = \frac{p+1}{\pi} \int_0^1 \int_{-\pi}^{+\pi} R_{pq}(r) \exp^{[-iq\phi]} f(r,\phi) \cdot r \, dr d\phi$$

$$R_{pq}(r) = \sum_{s=0}^{\frac{(p+|q|)}{2}} (-1)^s \frac{(p-s)!}{s! \left(\frac{(p+|q|)}{2} - s\right)! \left(\frac{(p-|q|)}{2} - s\right)!} r^{(p-2s)}$$

$$r = \sqrt{x^2 + y^2}$$

$$\phi = \tan^{-1}\left(\frac{y}{x}\right)$$

where the parameters underly the conditions $-1 < x, y < 1$, $n > 0$ and $0 \leq |m| \leq n$.

The Zernike Moments also have an efficient recurisve computation procedure and c can be partitioned in $\Re$ and $\Im$.

3.1.9 Feature Filters

The idea of feature filters is to find a linear system g, which allows for a good differentiation of two patterns f_0, f_1 belonging to different classes. That means after the filter was applied, f_0 and f_1 should be as far apart from each other as possible.

We imagine a typical signal/noise model, where noise is added to an ideal signal:

where s denotes the ideal pattern/signal and n the noise.

Now we say that:

$f_0 = n$

$f_1 = s + n$

We start with computing the noise energy of the signal: $\Sigma (f_1)^2 = \vec{f}^T \vec{f}$.
Furthermore we use convolution theory (see 9.1) and move a indicator signal over our pattern, which makes 'peep' (1), when the optimal signal has been found.

noise energy $P_n = E\left\{ \left(g^T \cdot n\right) \left(g^T \cdot n\right)^T \right\} = g^T \Sigma_n g$

signal energy $P_s = \left(g^T \cdot s\right)^2$

The covariance matrix Σ can be estimated using Maximum Likelihood Estimation(see 8.1). The next step is, to have a closer look at the Signal-To-Noise ratio:

$$\text{SNR} = \frac{P_s}{P_n} = \frac{\left(g^T \cdot f\right)^2}{g^T \Sigma_n g}$$

we choose g in accordance to maximizing the SNR:

$$g* = \max_g \text{SNR}$$

$$g* = \frac{g^T \Sigma_n g}{g^T s} \cdot \Sigma_n^{-1} \cdot s = \alpha \Sigma_n^{-1} \cdot s$$

In $\frac{P_s}{P_n}$, α drops out (so we can choose an arbitrary value for it, e.g. $\alpha = 1$) and we get the adapted feature filter:

$$g* = \Sigma_n^{-1} \cdot s$$

3.1.10 Short Term Fourier Transformation

The problem with Fourier Transformation is that information about time gets lost, because of the translation invariance. Yet with time dependent signals (e.g. speech signals) time information is fundamental. The idea of short term fourier transform is to use time windows (see 2.1.13) for the fourier transform.
In speech recognition these windows have shown through practical application, that 10-25 ms is a good width for such windows. Yet in general this is a problem, because the window resolution has to be:

- small enough, to allow for a good resolution in time
- high enough, to allow fot a good resolution in frequency

This trade-off is called the **uncertainty principle**:

$$\Delta t \Delta w \geq \frac{1}{4\pi}$$

where w is the signal's bandwidth and t the time spread.

Carefully choosing our window resolution in accordance to the uncertainty principle, we get the window function ω and can give a defintion of a Fourier Transform based on window functions:

$$\mathrm{STFT}\left(\tau, \omega\right) = \int f\left(t\right) \cdot \omega\left(t - \tau\right) e^{-2\pi i \omega t} dt$$

Properties

- evenly distributed partition of the time/frequence area
- preserves time information

3.1.11 Wavelets

Waveltes are a set of waves on the time-axis. A basis wave type, called **Motherwavelet,** is defined and **Childrenwavelets** are derived from this basic type by operations like scaling (α) and translation (τ). The Short Time Fourier Transform above is a special kind of wavelet. That means just like the STFT, wavlets preserve information about the time spread.

Wavelet Transformation

The Wavelet Transformation is given by:

$$WT\left(\tau, \alpha\right) = \frac{1}{\sqrt{\alpha}} \int_{-\infty}^{\infty} f\left(t\right) \oplus \psi\left(\frac{t - \tau}{\alpha}\right) dt$$

and the Inverse:

$$f(t) = \frac{1}{c_\psi} \int \int_{-\infty}^{\infty} WT(\tau, \alpha) \frac{1}{\sqrt{\alpha}} \psi \left(\frac{t-\tau}{\alpha} \right) \frac{d\tau d\alpha}{\alpha^2}$$

where α is the parameter for scaling mentioned above and τ the parameter for translations.

Features

Having the Wavelet Transform $\Psi(\omega)$ we can take the features, just as we did with the Fourier Transformation (see 3.1.2):

$$c_\psi = \int_{-\infty}^{\infty} \frac{|\Psi(\omega)|}{|\omega|} d\omega$$

where $\Psi(\omega)$ is the Wavelet Transform[1] of the basis wavelet $\psi(t)$.

Wavelets $\psi(t)$ underly the condition: $\int_{-\infty}^{\infty} \frac{|\Psi(\omega)|}{|\omega|} d\omega < \infty$, that means $\Psi(0) = 0$, which is equivalent to $\int_{-\infty}^{\infty} \psi(t)\, dt = 0$.
Other possible features are, like with the Fourier Series, the coefficients of the wavlet series.

Wavelet Series

Like the Fourier Series, the summation over wavelets is called Wavelet Series:

$$f(t) = \sum_{i=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} c_{ik} \psi_{ik}(t)$$

where

$$c_{ik} = \int_{-\infty}^{\infty} f(t)\, \psi_{ik}(t)\, dt$$

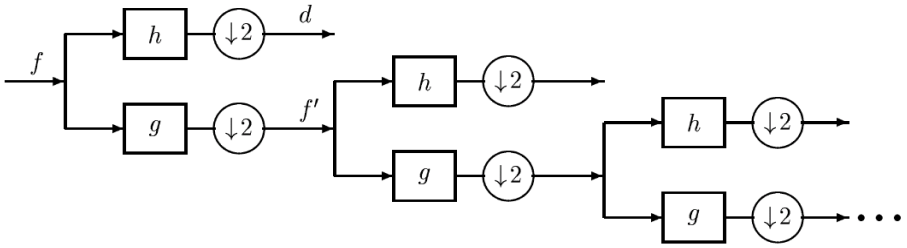
are the wavelet coefficients, which can be taken as additional features.

Resolution Hierachy

As mentioned in the chapter about filters, Wavelets are excellent to use for building high pass (h) / low pass (g) filters for resolution hierachies (see 2.1.15). Every application of a low/high pass filter reduces the bandwith by $\frac{1}{2}$ ($\downarrow 2$). The signal d resulting from the application of the high pass filter, is kept fpr restoring the resolution, while the signal f' is the original signal of a lower resolution. The part that has been removed from f to get f' is exactly d. According to the Nyquist Theorem (see 1.1) having $\frac{1}{2}$ the original bandwith, we only need half as many sample points resulting in a picture of reduced resolution.

[1] Since Wavelet Transforms also transform the signal into the frequency or Fourier space, $\Psi(\omega)$ ist often also referred to as the Foruier Transform.

Abbildung 24: Resolution Hierachy



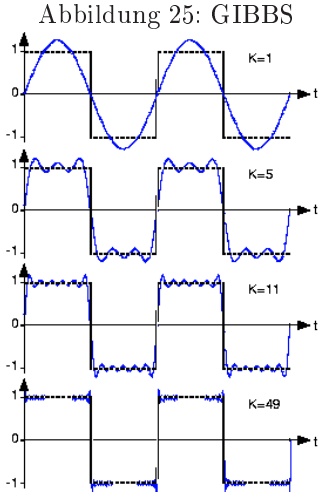
Again the basic idea of resolution hierachies is, that:

- a low resolution allows for a fast computation of features
- a high resolution allows for a exact computation of features

GIBBS

All window functions, like wavelets or short time fourier transformation, have the problem of **GIBBS**, swings over or under the top. That is, because they are approximating a continous function, and at some spots discontinouities occur. At these spots we the phenomena of GIBBS occur (see illustration below).

Abbildung 25: GIBBS



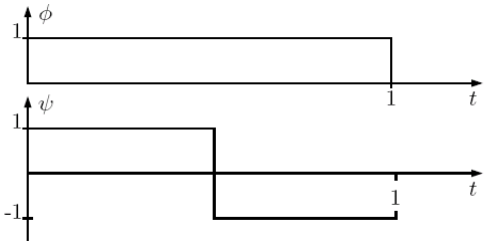
The image shows a Fourier series approximation of a signal. The number of terms in the Fourier sum is indicated in each plot, and the square wave is shown as a dashed line over two periods. With increasing number of summands the GIBBS' excess over resp. under the top is reduced but not their number.

STFT sets those swings simply to 0, Wavelet Theory models them with gaussian bells (GIBBS $\rightarrow \mathcal{N}$).

Example

We illustrate the realization of a resolution hierachy by using simple Haar mother wavelets $\phi(t), \psi(t)$ and child wavelets $\phi_{3,j}, \phi_{2,j}, \psi_{2,j}$ where the first index corresponds to the scaling parameter α and the second one to the translation τ:

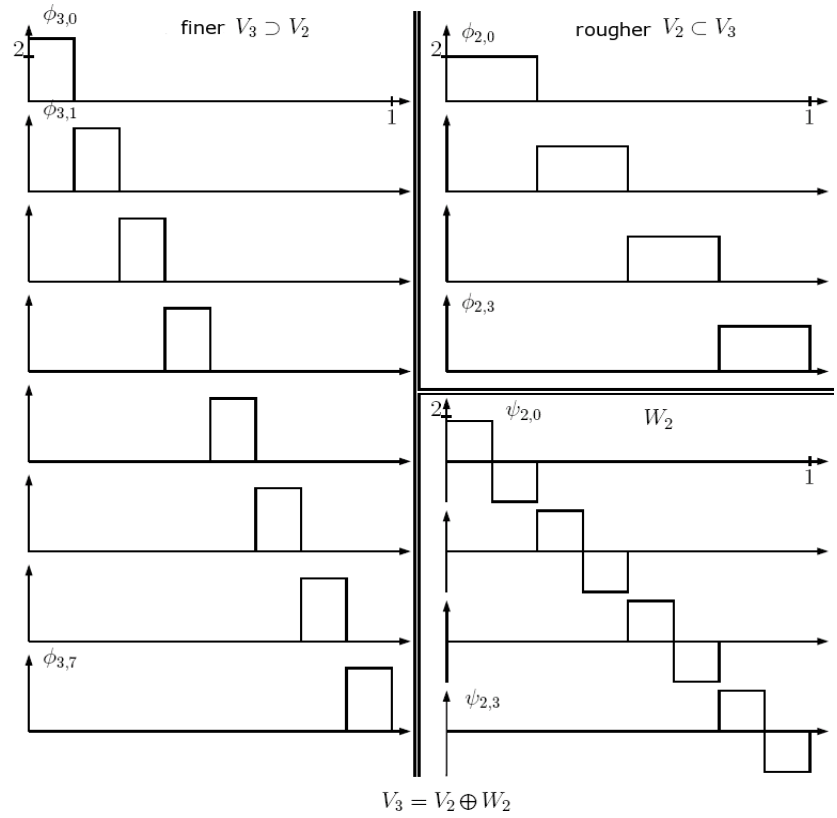
The mother wavlets are defines as:



$$\phi_{j,k} = 2^{j/2}(2^j t - k)$$

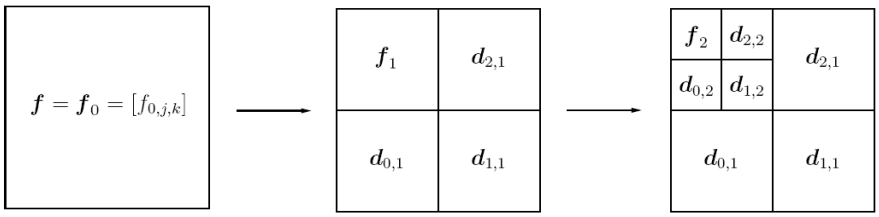
$$\psi_{j,k} = 2^{j/2}\psi(2^j t - k)$$

Using these wavelets we can build a resolution hierachy:



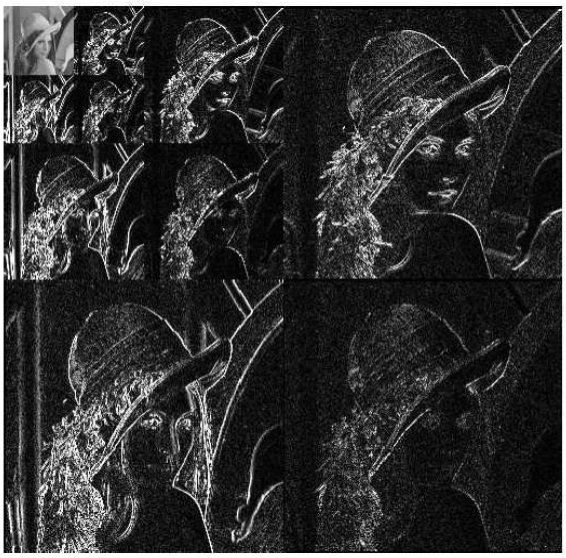
2D

Dealing with images the above process can be illustrated very nicely:



where f is the original picture, f_1 and f_2 are the results of low-pass filtering f to a lower resolution and $d_{i,j}$ are the parts we low-pass filtered away, that are gained by applying a high-pass filter. Because of the second dimension, we get three different such parts, by consecutively applying a 1D Wavelet Transform to the rows and then to the columns (or vice versa).

Example



Properties

- like the Fourier Transform, the Wavelet Transform has a band limit. At some point the FT is 0.
- the impulse respone determines the filter kernel (impulse respone:$= f(t)$)
- the short time fourier transformation is a form of wavelets (see 3.1.10)
- The afore mentioned Haar basis functions are also a form of wavlets (see 3.1.6)

Pro/Contra

+ preserves information about the time

+ Using Wavelets we get a hierachy in the spatial space and not only in the frequency space.

+ allows the modelling of GIBBS

3.1.12 MEL-Cepstrum

A **cepstrum** is the result of taking the Fourier Transform (FT) of the decibel spectrum as if it were a signal. Its name was derived by reversing the first four letters of "spectrum".

Features are won by a short term analysis of window functions (Rectangle, Hamming, Hanning, see 2.1.13). One feature per time window is extracted by the following procedure:

Algorithm 8 MEL-Cepstrum

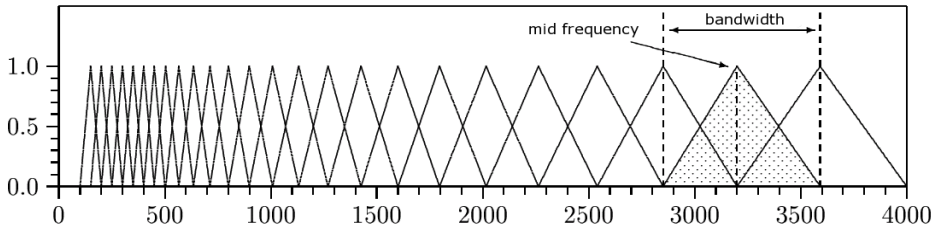
---

1. take the absolute values of the Discrete Fourier Analysis coefficients
2. take the logarithms
3. calculate LPC coefficients with the resulting function values
4. build the model spectrum (see 7)
5. partition into **MEL-frequency-components**: 32 Δ-Filter building frequency groups
6. apply Fourier Transfrom to the model spectrum to get the cepstrum and according coefficients
   calculate MEL-cepstrum coefficients by cos-transformations
7. features are time sequences of these cepstrum coefficients

---

Δ-Filter

Abbildung 26: Δ-Filter



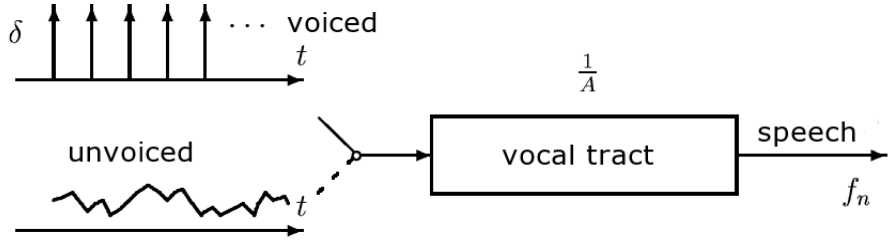
a triangle filter bank consiting of seven filters linearily stepped by mid frequencies

One speech signal is partitioned/filtered into 32 filtered speech signals. They become increasingly longer parallel too having less features (see image).

Those filter summarize over the coefficients of the spectrum.

MEL cepstrum coefficients

Abbildung 27: Speech Production



The computation of the MEL cepstrum coefficients is modelling the system of human speech production (see image):
A stimulation signal is convoluted with the impuls answer of the vocal tract. This convolution becomes an additive operator, when moving to the cepstrum. That means we may apply the logarithm, and therefore we can use quadratic Fourier Coefficients.

Properties

- preservation of time information by the use of window partitioning
- if the number of coefficients is too high, it can be reduced by applying PCA (see 3.1.14).
- An additional feature often combine with the mel coefficients, is the auditory level (german "Lautheit").

Pro/Contra

+ almost exact modelling of the human hearing system
+ very well suited for speech recognition
- not suited for image processing (because of the separation)

Application

- a practical number of coefficients is 20-30

[3] Analyitcal Features The goal of analytical feature computation is to find a transformation ϕ, that allows for the computation of features $\vec{c}$, which optimally describe a signal $\vec{f}$:

$$\vec{c} = \phi \cdot \vec{f}$$

where $\phi \in \mathbb{R}^{N \times M}$ with N being the dimension of the features and M being the dimension of the signal.

Criterions Criterions for an optimal description are:

- the maxims of pattern recognition (see 8.25)
- a low error rate of the classifier

The maxims can be represented mathematically by four criterions of compactness:

Compactness Criterions

1. The first criterion is the average distance (quadratic) of all features to each other feature:

$$S_1 = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (c_i - c_j)^T (c_i - c_j)$$

   S_1 does not require labeled samples (features labeled with class numbers).

2. The second criterion is the average distance (quadratic) of all features with one class κ to all features from other classes $\lambda \neq \kappa$:

$$S_2 = \frac{2}{K(K-1)} \sum_{\kappa=2}^{K} \sum_{\lambda=1}^{K-1} \frac{1}{N_\kappa N_\lambda} \sum_{i=1}^{N_\kappa} \sum_{j=1}^{N_\lambda} (c_{\kappa i} - c_{\lambda j})^T (c_{\kappa i} - c_{\lambda j})$$

   S_2 does require labeled samples. It corresponds to the first maxim of pattern recognition.

3. The third criterion is the average distance (quadratic) of all features with one class κ to other features of the same class:

$$S_3 = \frac{1}{K} \sum_{\kappa=1}^{K} \frac{1}{N_\kappa^2} \sum_{i=1}^{N_\kappa} \sum_{j=1}^{N_\kappa} (c_{\kappa i} - c_{\kappa j})^T (c_{\kappa i} - c_{\kappa j})$$

   S_4 also requires labeled samples. It corresponds to the second maxim of pattern recognition.

4. The fourth criterion is S_2 with the Lagrange constraint S_3 (see 8.10):

$$S_4 = S_2 + \theta \cdot S_3$$

3.1.13 Problem Dependent Series Development

As stated above, our goal with analytical feature computation methods is to find a transformation ϕ, such that feature vector $\vec{c}$ optimally represents signal $\vec{f}$.

$$\vec{c} = \phi \cdot \vec{f}$$

For Problem Dependent Series Development we choose ϕ to maximize S_1, the distance from all features to all other features, and define the objective function:

$$\hat{\phi} = \text{argmax}_\phi \, S_1(\phi)$$

$$\hat{\phi} = \text{argmax}_\phi \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\phi\vec{f_i} - \phi\vec{f_j}\right)^T \left(\phi\vec{f_i} - \phi\vec{f_j}\right) + \lambda\left(\|\phi\|_2 - 1\right)$$

The Lagrange constraint (see 8.10) at the end is necessary, because without we would optimize S_1 to ∞.

Look at the vector product in the middle: We are able to proove that the row vectors of ϕ correspond to the eigenvectors of the kernel matrix (see below).

Of course S_1 can be replaced by any other criterion of compactness. In the following section we will show exemplarily how to find ϕ choosing the criterion S_1.

Sloving For ϕ Optimizing Criterion S_1

Spread Maximization is defined as:

$$S_1(\phi) = \frac{1}{K} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\vec{c_i} - \phi\vec{f_j}\right)^T \left(\vec{c_i} - \phi\vec{f_j}\right)$$

thus maximizing for ϕ ($\hat{\phi} = \text{argmax}\,\phi$) corresponds to the maximazition of

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\phi\vec{f_i} - \phi\vec{f_j}\right)^T \left(\phi\vec{f_i} - \phi\vec{f_j}\right)$$

Douh! That's quadratic in ϕ and that's bad!

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\phi\left(\vec{f_i} - \vec{f_j}\right)\right)^T \left(\phi\left(\vec{f_i} - \vec{f_j}\right)\right)$$

we substitute $\left(\vec{f_i} - \vec{f_j}\right)$ by $\vec{g_{ij}}$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \vec{g}_{ij}^T \phi^T \phi \vec{g}_{ij}$$

applying some matrix properties (see 9.9), we get

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \text{trace}\left(\phi^T \phi \vec{g}_{ij} \vec{g}_{ij}^T\right)$$

$\vec{g}_{ij}\vec{g}_{ij}^T$ is called **Measurement Matrix** (ME).

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \text{trace}\left(\phi^T \phi \, \text{ME}_{ij}\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \text{trace}\left(\text{ME}_{ij}^T \, \phi^T \phi\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \text{trace}\left(\text{ME}_{ij}^T \begin{pmatrix} \vec{t}_1 & \cdots & \vec{t}_N \end{pmatrix} \cdot \begin{pmatrix} \vec{t}_1 \\ \vdots \\ \vec{t}_N \end{pmatrix}\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \text{trace}\left(\text{ME}_{ij}^T \sum_{k=1}^{N} \vec{t}_k \cdot \vec{t}_k^T\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \text{trace}\left(\text{ME}_{ij}^T \, \vec{t}_k \cdot \vec{t}_k^T\right)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} \vec{t}_k^T \, \text{ME}_{ij}^T \, \vec{t}_k$$

$$= \sum_{k=1}^{N} \vec{t}_k^T \left[\sum_{i=1}^{N} \sum_{j=1}^{N} \text{ME}_{ij}^T\right] \vec{t}_k \rightarrow \text{maximize}$$

$\sum_{i=1}^{N} \sum_{j=1}^{N} \text{ME}_{ij}^T$ is called **Kernel Matrix** Q. Now we can maximize this term by partial derivation with Lagrange constraints. We have shown that the rows of ϕ ($\vec{t}_k$) are the Eigenvectors of Q ($Q \cdot \vec{t}_k = \lambda \vec{t}_k$).

A crucial advantage gained by the usage of Kernel Matrices is that the optimization of S_i is reduced to solving an eigenvector, eigenvalue problem.

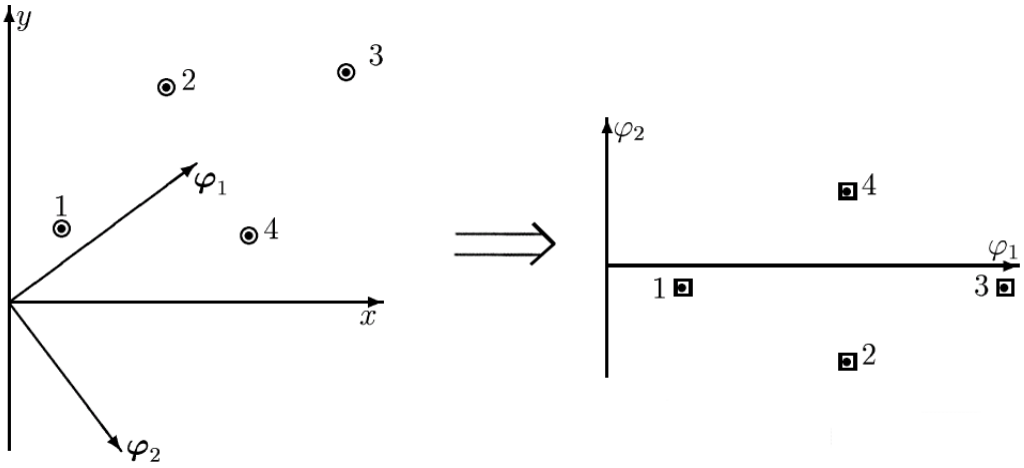
A crucial disadvantage is, that this matrix is simply too huge to be solved for eigenvectors (see 3.1.14 for an example). But some bright man discovered a way to simplify this eigenvector computation by factorizing Q and computing **implicit eigenvectors** of the resulting factorization, which in fact correspond to the eigenvectors of Q: see 8.13.

Properties

- since the resulting vectors $\vec{t}_k$ depend on the features, respectively the samples, this method was called problem dependent series development.

3.1.14 Principle Component Analysis (PCA)

Abbildung 28: Principle Component Analysis



φ_1, φ_2 denote the new found axis, that is seperating the features: $1 \dots 4$ best.

Theory

In PCA we try to find a Transformation T that maximizes the distance between any two features: $T := \vec{a}^T \in \mathbb{R}^n$

What we do is, we look for an axis in the feature space that seperates the features best. The eigenvector corresponding to the biggest eigenvalue (absolute values) of the covariance matrix corresponds to the "most" information in the features. Then we use this axis as a new coordinate system (sub-space) and project the features onto it.

Abbildung 29: Principle Component Analysis Theory

In a theoretical point of view, PCA corresponds to an orthogonal regression L of features $L(\vec{c}) = (c_1 \dots c_p) \curvearrowright \vec{y}(y_1 \dots y_q)$ where $q \ll p$.
$c_i :=$ feature
$\bar{c} :=$ mean value
$\vec{u} :=$ direction vector
$y_i := \vec{u}^T \cdot (c_i - \bar{c})$
use Pytagoras: $z_i^2 = (c_i - \bar{c})^2 - y_i^2$

The goal is now to minimize $\vec{u}$, which corresponds to a maximization of y_i^2:

$$\sum y_i^2 = \sum \vec{u}^T (\vec{c}_i - \bar{c}) (\vec{c}_i - \bar{c})^T \cdot \vec{u}$$

$$L\left(\vec{u}^T\right) = u^T \left[\sum_{i,j} \left(\vec{c}_i - \vec{c}_j\right)^T \left(\vec{c}_i - \vec{c}_j\right)\right] \vec{u} \rightarrow \max$$

$\left(\vec{c}_i - \vec{c}_j\right)^T \left(\vec{c}_i - \vec{c}_j\right)$ is the covariance matrix Σ.

We add the Lagrange constraint $\|\vec{u}\|_2 = 1$ to avoid infinite spread:

$$\vec{u}^T S \vec{u} - \lambda \vec{u}^T \vec{u} = 1$$

and apply the Lagrange Multiplyer Method:

$$\text{argmax}_{\vec{u}^T} \left\{\vec{u}^T \Sigma \vec{u} - \lambda \left(\vec{u}^T \vec{u} = 1\right)\right\}$$

deriving to $\vec{u}^T$ draws

$$2\Sigma\vec{u} - 2\lambda\vec{u} = 0$$

finally we try to solve the resulting eigenvector, eigenvalue problem:

$$\Sigma\vec{u} = \lambda\vec{u}$$

Algorithm

Algorithm 9 PCA

---

1. calculate covariance matrix Σ of the features x_i:

$$\Sigma = \frac{1}{N^2} \cdot \sum_i^N \sum_j^N \left(\vec{x}_i - \vec{x}_j\right)^T \left(\vec{x}_i - \vec{x}_j\right)$$

2. calculate eigenvalues and eigenvectors of Σ.

3. find the eigenvector which corresponds to the biggest eigenvalue (absolute values).

4. use this vector as first row of the transformation T.

$$T = \begin{pmatrix} x_1 & y_1 & \cdots \\ x_2 & y_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

   where vector 1 $(x_1 y_1 \cdots)$ is the eigenvector corresponding to the biggest eigenvalue (absolute values) and vector 2 $(x_2 y_2 \cdots)$ the eigenvector corresponding to the second biggest eigenvalue, and so on.

5. get the features: $c_i = T\left(x_i\right)$

Example

Example (memory requirements)

Let's take X-Ray images with resolution 1024×1024.
Our covariance matrix is $\Sigma = (\vec{c}_i - \vec{c}_j)^T (\vec{c}_i - \vec{c}_j), \Sigma \in \mathbb{R}^{2^{20} \times 2^{20}}$
that means in Byte (B): $2^{20} \cdot 2^{20} \cdot 4B = 2^{30} \cdot 4KB = 2^{20} \cdot 4MB = 2^{10} \cdot 4GB = 4TB$
Σ should be kept in RAM to work efficiently!

Properties

- reduces the dimension
- maximizes the distance between any two feature vectors
- non-supervised learning
- instead of the covariance matrix, you can also take the kernel matrix (see 10.1)
- PCA is related to linear regression (see 4.3.3), as has been illustrated in the picture at the beginning of this section

Pro/Contra

\+ no requirements on the features and samples

\- Adidas-Problem (see 8.24)

\- huge memory requirements (for a solution, see 8.13)

\- not suitable for image processing, because of the memory requirements (see example above)

\- not suitable for modelling rotations, take geometrical moments instead (see 3.1.8)

3.1.15 Linear Discriminant Analysis (LDA)

LDA is an extension to PCA. LDA resembles gaussian classifiers, with the difference, that it assumes the covariance matrices of all classes to be equal. This is a form of Parameter Tying.

1 Parameter Tying

In LDA we assume all covariance matrices to be equal: $\forall_\kappa \Sigma_\kappa = \Sigma$ (see 8.14)
As a consequence the decision boundary becomes linear in the components of $\vec{c}$:
Start:

$$\lambda = \text{argmax}_\lambda \log p(\Omega_\lambda) - \frac{1}{2}\log\det(2\pi\Sigma_\lambda) - \frac{1}{2}(\vec{c}-\vec{\mu}_\lambda)^T \Sigma_\lambda^{-1}(\vec{c}-\vec{\mu}_\lambda)$$

$\log\det(2\pi\Sigma_\lambda)$ is constant, and can be left out for maximization.

$$\lambda = \text{argmax}_\lambda \log p(\Omega_\lambda) - \frac{1}{2}\vec{c}^T\Sigma^{-1}\vec{c} - \frac{1}{2}\vec{\mu}_\lambda^T\Sigma^{-1}\vec{\mu}_\lambda + \vec{c}^T\Sigma^{-1}\vec{\mu}_\lambda$$

$\frac{1}{2}\vec{c}^T\Sigma^{-1}\vec{c}$ is also constant, what remains is:

$$\lambda = \text{argmax}_\lambda \log p(\Omega_\lambda) - \frac{1}{2}\vec{\mu}_\lambda^T\Sigma^{-1}\vec{\mu}_\lambda + \vec{c}^T\Sigma^{-1}\vec{\mu}_\lambda$$

which is in fact linear. □

(For classification a linear function can be built out of that: $\alpha_{0\kappa} + \alpha_\kappa^T\vec{c}$)

2 Data Normalization

We can choose to not only set the covariance matrices equal, but to normalize them to 1. For that we use SVD and decompose Σ to $\Sigma = (UDU^T)$. We then add the decomposed Σ components to the rest of the decision boundary and get a new covariance matrix equal to 1:

$$(\vec{c}-\vec{\mu})^T\Sigma^{-1}(\vec{c}-\vec{\mu}) = \left[(\vec{c}-\vec{\mu})^T UD^{-\frac{1}{2}}\right] 1 \left[D^{-\frac{1}{2}}U(\vec{c}-\vec{\mu})\right]$$

By that we are moving towards a Nearest Neighbour Classifier with all covariances equal to 1 and the mean corresponding to the neighbours. Still we have additional a-priori knowledge.

Algorithm

Algorithm 10 LDA

given: $M = \begin{pmatrix} \vec{M}_1 \\ \vec{M}_2 \\ \vdots \\ \vec{M}_k \end{pmatrix}$, k classes

1. compute the common covariance matrix of all classes: $\forall_\kappa \Sigma_\kappa = \Sigma$
   e.g. compute all Σ_λ and take the average of all:
   $$\Sigma = \frac{\sum_{\kappa=1}^{K} \sum_{i=1}^{N} (\vec{c}_i - \vec{\mu}_\kappa)(\vec{c}_i - \vec{\mu}_\kappa)^T}{N - K}$$
2. normalize the covariance matrix to $\Sigma = 1$ by transforming the features with SVS decomposition.
   $\Sigma = UDU^T$ and $T = UD^{-\frac{1}{2}}$.
3. compute normalized mean vectors: $\mu^* = \mu \cdot U \cdot D^{-\frac{1}{2}}$.
4. apply 3.1.14 to the normalized mean vectors μ^*.

Properties

- assumes gaussian distribution and equal covariance matrices
- these two assumptions result in a linear decision boundary
- reduces the dimension (limitation to the subsapce spanned by the mean vectors μ)
- maximizes the inter class distance
- minimizes the intra class distance
- supervised learning
- Bias-Variance Trade-Off (see 6.1): We take the bias of a linear decision boundary, because it can be computed with a much lower variance.
- $O\left(N \cdot p^2 + p^3\right)$ where p is the number of predictors and N the number of features

Pro/Contra

\+ solves the Addidas problem because of the normalization.

+ decision boundaries created by LDA are linear leading to easy to implement decision rules

+ its simplicity gives a natual low-dimensional view of the data

+ it generally has a very low variance

- requires labelled training vectors $(\vec{c}_i, \Omega_\lambda)$

- for every feature, we want to classify, the Mahalonobis distance has to be computed

- classes are not always linear seperable, QDA can help you there, but a method to produce a general irregular decission boundary would be preferable

- LDA uses one prototype (the centroid) and a common covariance matrix to model the spread of a class, this is quite often insufficient. Often several prototypes are more appropriate. A solution for this issue is MDA.

Quadratic Discriminant Analysis (QDA)

Just let those covariance matrices be different and use other tricks to gain efficiency.

$$\delta_\kappa(\vec{c}) = \log p(\kappa) - \frac{1}{2}\log|\Sigma_\kappa| - \frac{1}{2}(\vec{c} - \vec{\mu}_\kappa)^T \Sigma_\kappa^{-1}(\vec{c} - \vec{\mu}_\kappa)$$

Regularized Discriminant Analysis (RDA Friedmann 1989)

Only assume equality in some components (Σ) of the covariance matrices and allow others class dependent (Σ_κ) to vary.

$$\Sigma_\kappa(\alpha) = \alpha\Sigma_\kappa + (1-\alpha)\Sigma$$

Flexible Discriminant Analysis (FDA)

FDA follows a different approach than LDA. It formulates the LDA problem as a problem of regression making it much more general and flexible.

$$\text{argmin}_{\alpha,f} \sum_{i=1}^{N} \left(f(g_i) - \vec{c}_i^T \alpha\right)^2$$

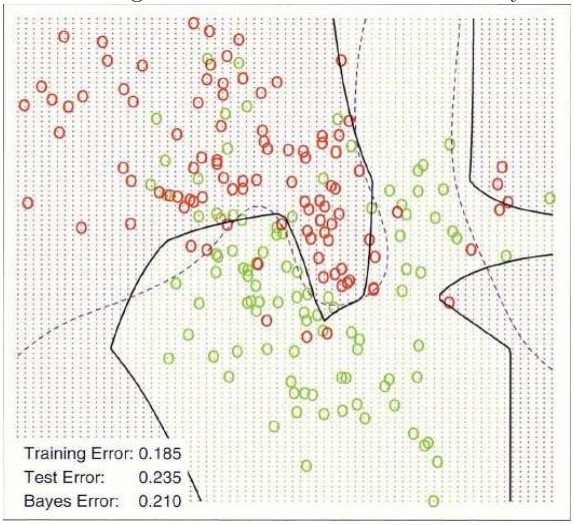
where $f : \mathcal{G} \to \mathbb{R}^1$ is a function assigning scores to functions.

This can also be minimized by minimizing the average squarrd residual:

$$\text{ASR} = \frac{1}{N}\sum_{l=1}^{L}\left[\sum_{i=1}^{L}\left(f_l(g_i) - \vec{c}_i^T \alpha_l\right)^2\right]$$

That means LDA can be performed by a sequence of linear regressions, followed by classifying to the closest class centroid. We can choose any regression

Abbildung 30: Flexible Discriminant Analysis



FDA with the MARS regression method.

method with FDA and exploit its advantages combined with the advantages of LDA.

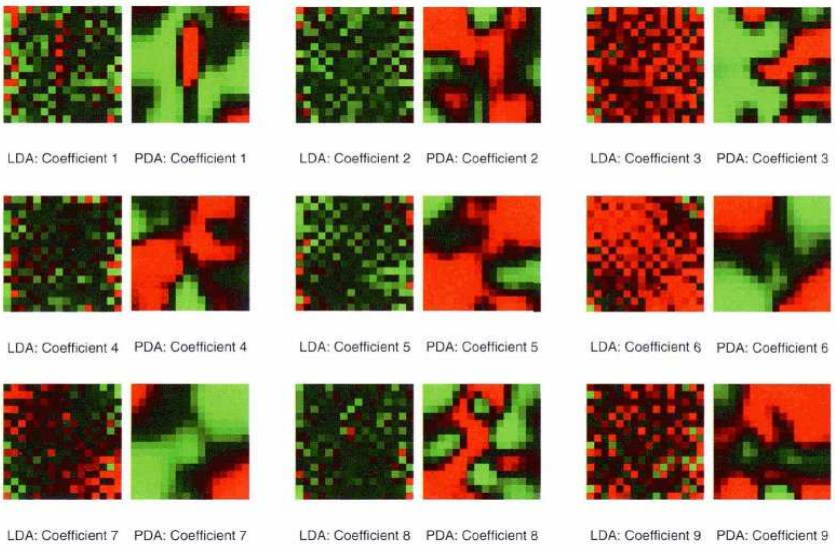
Properties

- runtime behaviout depends on the regression methods of choice. E.g. additive models and MARS are linear in $O(N)$, others like splines however run in $O\left(N^3\right)$.

Penalized Discriminant Analysis (PDA)

In the case of a too high feature dimension, we might apply PDA, penalizing features to be smooth or otherwise coherent in the spatial domain (This applies usually for images). PDA uses a penalized form of the Mahalanobis Distance (see 3.2.8) to evalute features. Furthermore the classification subspace is decomposed using a penalized metric. This penalized evaluation gives less weight too "rough" features and more too "smooth" ones.

Abbildung 31: Penalized Discitriminant Analysis



The images appear in pairs, and represent the nine discriminant coeficient functions for the digit recognition problem. The left member of each pair is the LDA coeficient, while the right member is the PDA coeficient, regularized to enforce spatial smoothness.

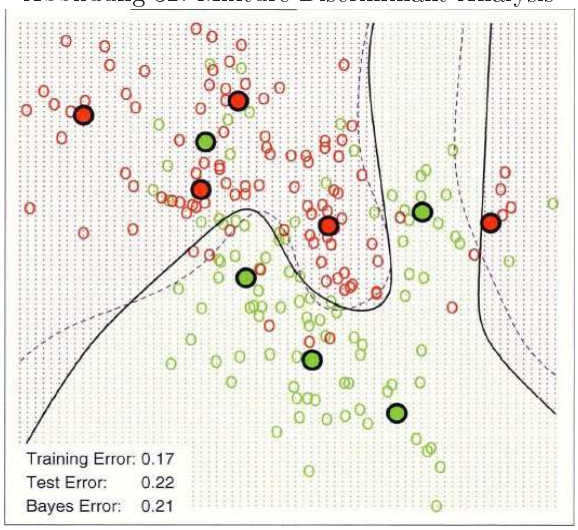
Properties

- PDA is very similar to FDA using penalized regression methods.
- Of course penalizing "roughness" makes not always sense. Therefore PDA is mainaly a classification method for image processing.
- $O\left(N \cdot p^2 + p^3\right)$ where p is the number of predictors and N the number of features

Mixture Discriminant Analysis (MDA)

MDA overcomes the issue that one single prototype per class (the class centroid) is often insufficient. Instead each class is modelled with a mixture of two to three gaussians giving the class a total of two to three prototypes and combinations between them. Because of the mixture densities ML is inefficient for estimating the estimates for the mixture distributions, instead we use the EM Algorithm (see 8.3).

Abbildung 32: Mixture Discriminant Analysis



MDA with two classes and five mixture centers per class

Properties

- This is connection of LDA and Mixture Densities (see 4.2.3).

3.1.16 Fisher Transform

Idea Find a linear mapping $\vec{a}^T$ of the features $\vec{c}$: $\vec{c}' = \vec{a}^T\vec{c}$, such that the inter class distance is maximized and the intra class distance is minimized. It corresponds to a new combination of the criterions of compactness (see3.1.12):

$$S_5 = \frac{S_2}{S_3}$$

Theory

The objective function, called Rayleigh-Quotient, is according to S_5:

$$\vec{a}^T = \text{argmax}_{\vec{a}} \frac{\vec{a}^T A\vec{a}}{\vec{a}^T B\vec{a}}$$

where A denotes the inter class covariance and B denotes the intra class covariance. This corresponds to a maximization of $\vec{a}^T A\vec{a}$ subject to $\vec{a}^T B\vec{a} = \text{const}$, to keep the intra class distance constant, while maximizing the inter class distance. (See similarity to 3.1.15). This is a generalized eigenvalue problem, with $\vec{a}$ given as the eigenvector corresponding to the largest eigenvalue of $B^{-1}A$.

Apply the Lagrange Multiplyer Method to solve this problem by deriviations.

$$\delta(\vec{c}) = \left(\vec{c} - \frac{1}{2}(\mu_{\Omega_1} + \mu_{\Omega_2})\right)^T \Sigma^{-1}(\mu_{\Omega_1} - \mu_{\Omega_2})$$

Properties

- Fisher Transform reaches the same result as LDA (see 3.1.15), with a different approach
- Fisher Transform is also closely related to linear regression (see 4.3.3): We get the same results if we take the $\vec{y}$ from linear regression and take $\vec{a}$ from $\vec{y}^T\vec{y}$ using eigenvalue decomposition as above.

Pro/Contra

\+ simple classification due to a linear function (LDA requires calculation of Mahalonobis Distances (see 3.2.8) for every feature

3.1.17 Minimum Distance Classifier

The goal of Minimum Distance Classifiers is to find a feature transformation B, that minimizes the classified distances between feature vector and class. Therefore it is not, as the previous sections were, independent of the chosen classifier. Note, that we will use some terms that are later introduced in 4.2.1.

Having this goal in mind, we define a new criterion of compactness S_6:

$$S_6 = \sum_{\kappa=1}^{K} p(\Omega_\kappa) \frac{N}{u_\kappa(\vec{c})}$$

u_κ is measure for distance called Prüfgröße:

$$u_\kappa(\vec{c}) = (\vec{c} - \vec{\mu}_\kappa)^T \Sigma_\kappa^{-1} (\vec{c} - \vec{\mu}_\kappa) + \gamma_\kappa$$

where γ_κ is a class specific constant. The classifier will decide for the class Ω_κ, where u_κ is minimal. u_k can be simplified by dropping the class specific constant γ_κ. u_κ measures the distance or rather deviation between the feature $\vec{c}$ and the mean value of a class $\vec{\mu}_\kappa$ respecting the class covariance Σ_κ.

We look for a transformation B minimizing S_6:

$$\vec{c}' = B\vec{c}$$

$$\left(B \cdot \Sigma_\kappa B^T\right)^{-1} = B^{T^{-1}} \Sigma_\kappa^{-1} B^{-1}$$

This time we try to minimize the defined distances using statistics. Therefore we try to derive an euclidean distance measure from a probability density function (see 10.1).

$$p(\vec{c} \mid \Omega_\kappa) \rightarrow \text{euclidean distance}$$

We assume gaussian distribution: $p(\vec{c} \mid \Omega_\kappa) = \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$:

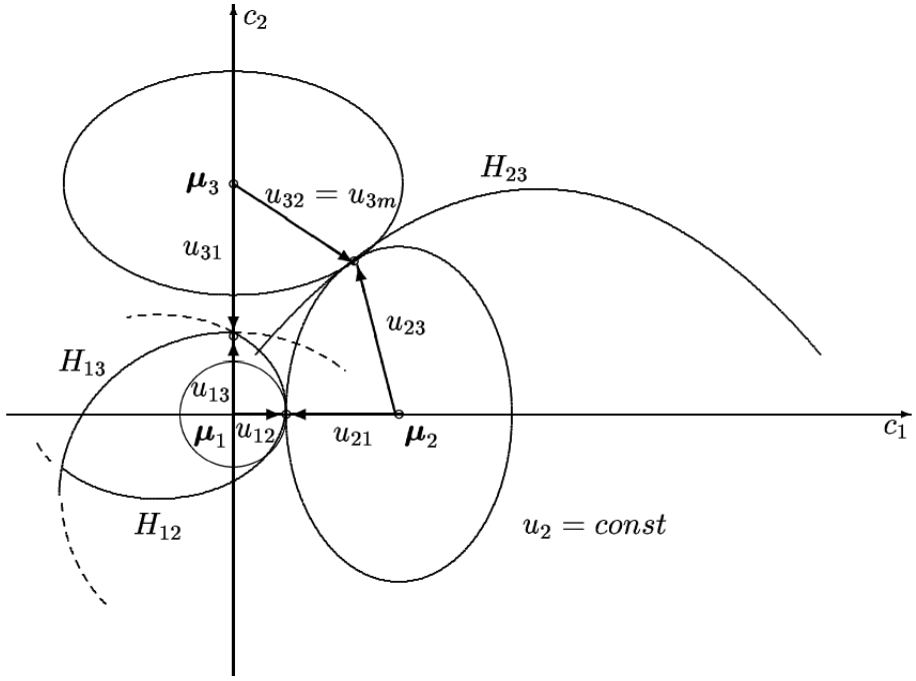
$$\text{argmax}_\kappa \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa) = \text{argmax}_\kappa \log \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$$

a maximizing of $\mathcal{N}$ corresponds to a minimization of the exponent:

$$\text{argmin}_\kappa (\vec{c} - \vec{\mu}_\kappa)^T \Sigma_\kappa^{-1} (\vec{c} - \vec{\mu}_\kappa)$$

After minimizing, we take the result as distance and optimize B accordingly, for example by using Corrdinate Descent (see 8.7).

Abbildung 33: Minimum Distance Classifer



As can be seen, we indeed use the distance measure $u_{\kappa\lambda}$ as an euclidean measure of distance, used to define the class borders $H_{\kappa\lambda}$. μ_κ denotes the mean value of the corresponding class.

Principle

This works, because under three circumstances our pdf exactly corresponds to the euclidean distance:

- Every covariance matrix is equal to the identity matrix: $\forall_\kappa : \Sigma_\kappa = \text{Id}$
- The prior probability of the classes is uniformly distributed: $\forall_{\kappa,\lambda} p(\Omega_\kappa) = p(\Omega_\lambda)$

- The pdf is gaussian: $p(c \mid \Omega_\kappa) = \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$

Then the optimization above is reduced to the minimization of $\text{argmin}_\kappa (\vec{c} - \vec{\mu}_\kappa)^T (\vec{c} - \vec{\mu}_\kappa)$ which is in fact the euclidean distance.
This means, even if not all of those circumstances are given, the pdf is stil closely related to the euclidean concept of distance and instead of maximizing a pdf, we can minimize distances.

Pro/Contra

- requires gaussian distribution in the classes. It does work without, but with a much lower performance.

See Also

- Optimal Classifier, Bayes (maximization of pdfs): 4.2.1
- How are distances and probabilities related: 12

3.1.18 Sammon-Transformation

Sammon's idea was to find a projection of the feature space into 2D, like a topographical map. This has not only the advantage of simpler classification, but also that the pattern recognition process can be illustrated and explained to the user, e.g. a patient at the hospital. The important thing is that the desired transformation T must preserve topographical properties.

So again we look for a transformation T

$$\vec{c}' = T\{\vec{c}\}$$

and minimize the **Sammon Criterion** S_s

$$\hat{T}_\theta = \text{argmin}_\theta S_s(\theta)$$

where argmin must be preserve topograhical properties.

To minimize S_s we use gradient decent (see 8.6).

Pro/Contra

+ good visualization, especially for non-technical users (e.g. patients at the hospital)
+ reduction of dimension
+ easy classification
- lost information due to the projection

3.2 Feature Evaluation And Selection

Finding many features is no problem at all, in the last section we introduced various approaches of possible features, yet classification with so many features is a problem. That is above all because of the Curse Of Dimensionality (see 8.18). and because many features lead to computational expensive classifications. That is why in this chapter we will discuss some methods with that, given n different features, we will be able to find the $n - x$ best ones. This gives us two basic taks:

1. finding a measure how good features are
2. developing search strategies for finding the best ones, using the found measure from

Also keep in mind that parameter tying (see 8.14), i.e. combining two features into one, as performed in for example PCA, is always a valiant alternative to abolishing features completely.

Pro/Contra

\+ Avoiding the Curse Of Dimensionality

\+ Reducing the search space

\- Assumes that all features are statistically independent (execpt for gaussian distribution)

\- Geometrical moments above second order can not be evaluated

\- Most measures assume gaussian distribution and are very difficult to calculate, having a different distribution (i.e. the probability denisty functions of the classes have to be estimated resp. approximated by gaussians)

[A] Measures

3.2.1 Error Rate

The most intutive measure is the following one:

$$\hat{p}_f = \frac{\text{number of correctly classified patterns}}{\text{number of wrong classified patterns}}$$

We use the estimation indicator $\hat{}$, because p_f is estimated from a training set. Of course the error rate additionally depends on the classifier and not merely on the features.

Pro/Contra

+ very good

- very expensive

- depends on the chosen classifer

3.2.2 Bayes Distance

The Bayes Distance works with quadratic a-posteriori probabilities as measure respecting the piror probability of the feature occuring:

$$\mathcal{B} = \int_{R_c} \sum_{\kappa=1}^{K} p^2 \left(\Omega_\kappa \mid \vec{c}\right) p\left(\vec{c}\right) d\vec{c}$$

with

$$p\left(\vec{c}\right) = \sum_{\kappa=1}^{K} p\left(\Omega_\kappa\right) p\left(\vec{c} \mid \Omega_\kappa\right)$$

The bigger the Bayes Distance $\mathcal{B}$, the better the feature $\vec{c}$.

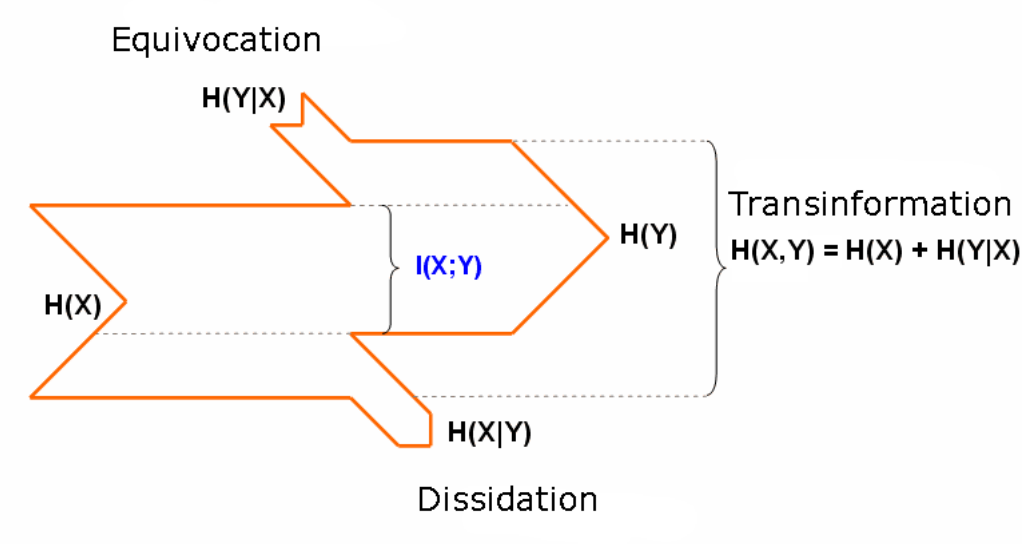
Pro/Contra

+ very good

- very expensive

3.2.3 Equivocation (Conditional Entropy)

Abbildung 34: Equivocation



As you can see the term reffered to as equivocation enters the signal, while the dissidation gets lost. The Transinfromation denoted by $I(X;Y)$ is the amount of original information that is contained in the final signal.

The Equivocation H is a measure for the transinformation(see 11):

$$H = \int_{R_c} \left(-\sum_{\kappa=1}^{K} p(\Omega_\kappa)\, p(\Omega_\kappa \mid \vec{c}) \log p(\Omega_\kappa \mid \vec{c}) \right) p(\vec{c})\, d\vec{c}$$

where R_c denotes the complete feature space.

Pro/Contra

\+ very good

\- very expensive

3.2.4 Kuhlbach-Leibler-Divergence

The Kuhlbach-Leibler-Divergence also models the transinformation (see 11), it is based on similarities of distributions:

$$Kl(p,q) = \int p(x) \log \frac{p(x)}{q(x)} dx$$

maximizing the Kulbach-Leibler similarity corresponds to
maximizing the transinformation corresponds to
maximizing the **statistical dependency**:

$$\max \int p\left(\vec{c}, \vec{c}'\right) \log \frac{p\left(\vec{c}, \vec{c}'\right)}{p\left(\vec{c}\right) \cdot p\left(\vec{c}'\right)} d\vec{c} d\vec{c}'$$

Properties

- $Kl\left(p, q\right) \neq Kl\left(q, p\right)$ The Kulbach-Leibler-Divergence is not symmetric
- The information is the better the more statistical dependent features $\vec{c}$ are.

3.2.5 Bhattacharyya Distance

The Bhattacharyya distance measures the similarity of two discrete probability distributions:

$$G_{\kappa\lambda}^{B} = -\ln\left(\int \sqrt{p\left(\vec{c} \mid \Omega_\kappa\right) p\left(\vec{c} \mid \Omega_\lambda\right)} dc\right)$$

assuming gaussian distribution, we can find a closed formula:

$$G_{\kappa\lambda}^{B} = \frac{1}{8}\left(\vec{\mu}_\kappa - \vec{\mu}_\lambda\right)^T \left(\frac{\Sigma_\kappa + \Sigma_\lambda}{2}\right)^{-1} \left(\vec{\mu}_\kappa - \vec{\mu}_\lambda\right) + \frac{1}{2} \ln \frac{\left|\frac{1}{2}\left(\Sigma_\kappa + \Sigma_\lambda\right)\right|}{\sqrt{\left|\Sigma_\kappa\right| \left|\Sigma_\lambda\right|}}$$

Example

given

$\vec{c}$	Ω_1	Ω_2
1	0	2
2	0	3
3	1	2
4	4	0
5	3	0

the Bhattacharyya Distance is:

$$G_{1,2}^{B} = -\ln\left(\sqrt{p\left(\vec{c}_3 \mid \Omega_1\right) p\left(\vec{c}_3 \mid \Omega_2\right)} + 0 + 0 + 0 + 0\right)$$

$$G_{1,2}^{B} = -\ln\sqrt{\frac{1}{8} \cdot \frac{2}{8}} = -\ln\sqrt{\frac{2}{64}} = 1.7328$$

Properties

- monotonic
- relates to two classes

3.2.6 Divergence

$$E\left\{\log\frac{p\left(\vec{c}\mid\Omega_\kappa\right)}{p\left(\vec{c}\mid\Omega_\lambda\right)}\mid\Omega_\kappa\right\}-E\left\{\log\frac{p\left(\vec{c}\mid\Omega_\kappa\right)}{p\left(\vec{c}\mid\Omega_\lambda\right)}\mid\Omega_\lambda\right\}$$

$$G_{\kappa\lambda}^{D}=\int\left(\ln p\left(\vec{c}\mid\Omega_\kappa\right)-\ln p\left(\vec{c}\mid\Omega_\lambda\right)\right)\left(p\left(\vec{c}\mid\Omega_\kappa\right)-\ln p\left(\vec{c}\mid\Omega_\lambda\right)\right)d\vec{c}$$

assuming gaussian distribution:

$$G_{\kappa\lambda}^{D}=\frac{1}{2}\left(\vec{\mu}_\kappa-\vec{\mu}_\lambda\right)^T\left(\Sigma_\kappa^{-1}+\Sigma_\lambda^{-1}\right)\left(\vec{\mu}_\kappa-\vec{\mu}_\lambda\right)+\frac{1}{2}\mathrm{Sp}\left(\Sigma_\kappa^{-1}\Sigma_\lambda+\Sigma_\lambda^{-1}\Sigma_\kappa-2\cdot\mathrm{Id}\right)$$

Properties

- monotonic
- relates to two classes

3.2.7 Transinformation

After talking about measures modelling the transinformation, we can take it itself as a distance measure:

$$G^T=\sum_{\kappa=1}^{K}p\left(\Omega_\kappa\right)\cdot p\left(\vec{c}\mid\Omega_\kappa\right)\cdot\log\frac{p\left(\vec{c}\mid\Omega_\kappa\right)}{p\left(\vec{c}\right)}$$

A good illustration of what transinformation means can be found here: 11.

Example

$\vec{c}$	1	2	3	4	5
$p\left(\vec{c}\mid\Omega_1\right)$	0	0	$\frac{1}{8}$	$\frac{4}{8}$	$\frac{3}{8}$
$p\left(\vec{c}\mid\Omega_2\right)$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{2}{8}$	0	0
$p\left(\vec{c}\right)$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{4}{16}$	$\frac{3}{16}$

$$G^T=\frac{1}{2}\cdot\left[\frac{1}{8}\cdot\log\frac{\frac{1}{8}}{\frac{3}{16}}+\frac{4}{8}\log\frac{\frac{4}{8}}{\frac{4}{16}}+\frac{3}{8}\log\frac{\frac{3}{8}}{\frac{3}{16}}+\frac{3}{8}\log\frac{\frac{3}{8}}{\frac{3}{16}}+\frac{3}{8}\log\frac{\frac{3}{8}}{\frac{3}{16}}+\frac{2}{8}\log\frac{\frac{2}{8}}{\frac{3}{16}}\right]$$

$$G^T=\frac{1}{2}\cdot 0.49=0.2492$$

Properties

- relates to all classes
- the transinformation can be estimated using Parzen Windowing (see 4.4.2)

3.2.8 Mahalanobis Distance

The Mahalanobis Distance is a distance measure for the special case, that all covariance matrices are equal:

$$\forall_{\kappa,\lambda} : \Sigma_\kappa = \Sigma_\lambda$$

For this case it, Mahalonibis gives a good measure for the quality of features:

$$G^M_{\kappa\lambda} = (\vec{\mu}_\kappa - \vec{\mu}_\lambda)^T \, \Sigma^{-1} \, (\vec{\mu}_\kappa - \vec{\mu}_\lambda)$$

in the special special case, that we only have one feature, we get:

$$G^\nu_{\kappa\lambda\nu} = 2 \cdot \frac{(\vec{\mu}_{\kappa\nu} - \vec{\mu}_{\lambda\nu})^2}{\sigma^2_{\kappa\nu}\sigma^2_{\lambda\nu}}$$

respectively

$$G^\nu_{\kappa\lambda\nu} = 2 \cdot \frac{(\vec{\mu}_{\kappa\nu} - \vec{\mu}_{\lambda\nu})^2}{\sigma^2}$$

because we assumed the covariances to be equal $\Sigma_\kappa = \Sigma_\lambda$.

Properties

- Given equal covariance matrices the Mahalanobis Distance $G^M_{\kappa\lambda}$ corresponds to:
  - the Bhattacharyya Distance: $G^M_{\kappa\lambda} = 8 \cdot G^B_{\kappa\lambda}$
  - the Divergence: $G^M_{\kappa\lambda} = G^D_{\kappa\lambda}$
- the Mahalanobis Distance is a generalization of the euclidean distance, in fact if the covariance matrices euqal the identity matrix $\Sigma = \mathrm{Id}$, Mahalanobis Distance equals the euclidean distance.
- In case of differing covariances, we can try Parameter Tying and setting all covariance matrices equal (see 8.14).
- monotonic

[B] Search Strategies

3.2.9 Single Best Evaluated Features

After having a measure to evaluate features, the "Single Best Evaluated Features" method simply evaluates all features and chooses the best ones.

Pro/Contra

+ easy, intuitive and fast

- does not respect interrelations between features

- does not respect hard-to-classify and easy-to-classify patterns (It might be better to concentrate on the difficult ones and choose the features accordingly)

3.2.10 Best Features Relatively To Other Features

A feature might get a good evaluation singled out, yet in a set with other features it might be not as good.

Algorithm 11 Selecting Features In Respect To Other Features

---

1. choose a measure (e.g. Mahalanobis Distance)
2. choose the single best evaluated feature (see 3.2.9) c_1 and set the selected feature set $S = \{c_1\}$.
3. ↻ compute the measure for the already selected feature set $S = \{c_1 \dots c_k\} \cup \{c_i\}$ for index i passing all not yet selected features.
4. select the c_i maximizing S and add it to $S = S \cup \{c_i\}$.
5. continue at 3, as long as the desired number of features has not been reached.

Pro/Contra

+ does respect interrelations between features

- does not respect hard-to-classify and easy-to-classify patterns (It might be better to concentrate on the difficult ones and choose the features accordingly)

3.2.11 Features For Difficult Patterns Search

A good strategy is not to focus on patterns that are simple to classify, but to look closer at the difficult ones. Accordingly "Features For Difficult Patterns Search" proposes to choose features that make classification of the most difficult patterns easier.

Pro/Contra

+ does respect hard-to-classify patterns

- prior knowledge about the difficulty of the classifications of different patterns is required

- does not respect a-priori probabilities of difficult patterns, that means a difficult pattern that almost never arises, might greatly influence the feature vector

3.2.12 (l, r)-Search

Start with a feature set S containing a couple of single best evaluated features. Then parallely add the l best evaluated features to S while removing the r worst evaluated features, until the desired number of features n' has been reached.

Algorithm 12 (l,r)-Search

We want to find the n' best features of a total number of features n.

Initialization: Choose values for l and r (with $l \neq r$).

1. If $l > r$: Initialize an empty set S, with currently chosen features.
   If $l < r$: Initialize S with all n features.

2. $\circlearrowleft$ l times:
   Choose from all features $\vec{c} \notin S$ the best evaluated feature and add it to S.
   If S contains n' features, end the search here and return S.

3. $\circlearrowleft$ r times:
   Choose from S the worst evaluated feature and remove it.
   If S contains n' features, end the search here and return S.

4. back to 2.

Properties

- optimal method for non-monotonic measurement functions

Pro/Contra

- always chooses single best/worst evaluated features

3.2.13 Floating Serach

Floating Search is very similar to (l,r)-search, yet we abolish l and r and rather look on how much features contribute to a set, rather than choose the single best evaluated ones.

Algorithm 13 Floating Search

Again we want to choose n' features out of a total number of n features.

1. Initialize S with the single best evaluated feature.
2. Choose the feature best in respect to all remaining features and add it to S.
   If S contains n' features, end the search here and return S.
3. Choose the feature $\vec{c}$ that contains the least amount to the evaluation of S.
   If the evaluation of S rises by eleminating $\vec{c}$, eleminate it and continue with step 3.
   Otherwise keep it and continue with step 2.

Properties

- does not require monotonic measurement functions

Pro/Contra

\+ does respect interrelations between features

3.2.14 Branch & Bound Search

Precondition for branch & bound is a monotonic measure function (e.g. Mahalanobis Distance, Divergence, Bhattacharyya Distance). Having a monotonic measure function means that the evaluation of S can never be bigger than the evaluation of $S \cup \{\vec{c}\}$ no matter what feature $\vec{c}$ represents ($G_1^j \leq G_1^{j+1}$). The goal of Branch & Bound is not to find the best features, but to find features which can be removed from the feature set with the least possible loss of information.

Algorithm 14 Branch & Bound Search

1. The first level contains $n + 1 - x$ nodes, ordered ($<$) by a measurement function, where x is the number of nodes you want to remove from your feature set.

2. Start with the biggest node of the current level, the one on the right hand border.

3. Evaluate all nodes whose index succeeds this nodes to build the next level.

4. ... repeat 2 and 3 untill you have removed x nodes. Then set the current optimal path to this sequence of nodes, if their evaluation overpasses the current optimal path's evaluation.

5. ↻ Iterate over all paths that have not been visited yet.
   → skip branches whose current evaluation is already worse than the current optimal path.

6. End when all paths have been processed.

Example

Our goal is to eliminate three out of ten features, thus we need to choose three different measures for each level of eleimination. We choose very simple ones, based on the index of the featrure to keep the example simple:
$G^9 = 10 + i$
$G^8 = 10 - \frac{1}{2}(i + j)$
$G^7 = 10 - \frac{1}{2}(i + j + k)$
Start

i	1	2	3	4	5	6	7	8
G^9	11	12	13	14	15	16	17	18

We stop at feature 8, because we have to keep at least two features for the other two levels, we want to remove three features after all. So our first feature to choose is $i = 8$, Let us continue with j.

j	9							
G^8	1,5							

We already have to stop after the first feature, to leave one for the last level. We choose $j = 9$ and continue with k

k	10							
G^7	-3,5							

We choose $k = 10$ and have our first current optimal path 8-9-10 with an evaluation of $G = 16$. Now we do backtracking and choose the next untouched branch. We reckognize that this is at the first level. We choose the second best evaluated feature $i = 7$ and continue at level 2.

j	8	9						
G^8	2,5	2						

we choose $i = 8$, because it got the best evaluation at this level and continue to level three

k	9	10						
G^7	-2	-3,5						

We choose $k = 9$ having the best evaluation in this level and get a new current optimal path: 7-8-9 with evaluation $G = 17.5$. We continue to the next unvisited branch at $j = 9$ and so on, until we visited every possible path. If you want to do it till then end, the best path results to be 1-2-3 with an evaluation of $G = 26.5$. Thus we remove these three features from our set, since speaking in Branch & Bound the best evaluations corresponds to the worst features.

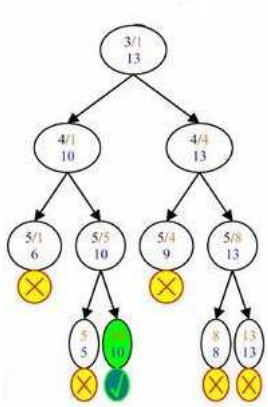
Note that usually we would not have to visit all paths. As soon we we'd encounter that the evaluation of the first branch node is worse than our current optimal path, we can skip this whole branch.
E.g. We are at index $i = 8$ and have an current optimal path of 17.5. Now via backtracking we would have to check the branched starting with index $i = 7$. Yet the node at 7 has an evaluation of 17. According to the rules of monotony, removing features (i.e. going down one level) can only worsen the evaluation, so we could never top 17.5 and only get worse. The result we draw from that is, that we can skip all branches rooting in $i = 7$ as well as for all indices smaller than $i = 7$, since we ordered them by evaluation from left to right ($<$). So in fact we are done at this point and can finish the algorithm having found the optimal path. This example illustrates, that branch&bound can in fact be pretty fast and usually never has to visit all branches (the worst case).

Properties

- optimal method for monotonic measurement functions

the worst case runtime usually is never reached. On the contrary, because of the monotony huge branches of the search tree can be skipped.

Abbildung 35: Branch&Bound



This part of a branch & bound search tree has been taken somewhere out of an application. Yet it illustrates, that whole branches can be skipped, if the weight total is already worse than that of the parent node.

3.2.15 Parameter Tying

The idea of parameter tying is to choose one parameter that is actually modelling two. The hope with this method is, that because of interrelations (e.g. statistical dependency of random variables) both parameters can be expressed by one, without loosing to much information of any one of them.

Concerning feature selection, we might decide to tie two features that were evaluated very badly, since the information loss would be minimal or choose to tie two parameters that were evaluated very similar, depending on the measure (e.g. measures basing on statistical dependency: Equivocation, Kuhlbach-Leibler Distance, Bhattacharyya Distance, Transinformation, etc.). For more details on Parameter Tying see 8.14.

3.2.16 Dynamic Programming

Use the methods of Dynamic Programming in respect to Bellman's Optimality Principle (see 8.19) to choose a "sequence" of good features.

3.2.17 Genetic Algorithms

The field of genetic algorithms can be applied to choose a set of good features. For more information about genetic algorithms, check e.g. Wikipedia / Genetic Algorithms `http://en.wikipedia.org/wiki/Genetic_algorithms`.

4 Classification

4.1 Introduction

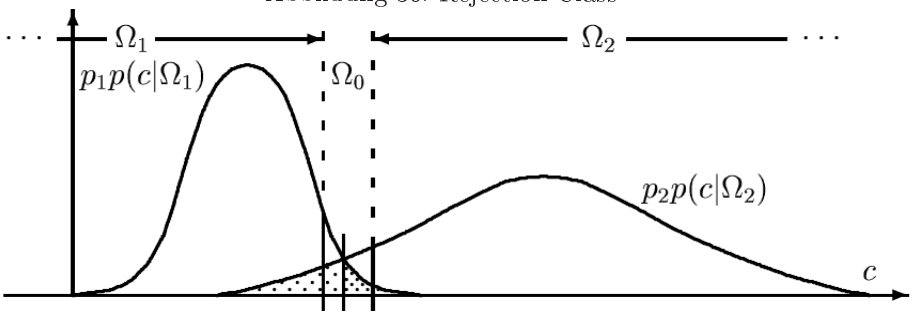
Last but not least we want to assign our features to a class. This is the job of a classifier. Restricting to two classes, we can formulate it like this:

4.1.1 Definitions

classifier $\delta(\vec{c}) = \begin{cases} -1 & \text{if } \vec{c} \text{ belongs to } \Omega_1 \\ +1 & \text{if } \vec{c} \text{ belongs to } \Omega_2 \end{cases}$

This can be expanded for any number of classes. In some cases we even introduce a special class Ω_0 used for rejection, we call it the **rejection class**.

Abbildung 36: Rejection Class



If the classification cannot decide whether to assign $\vec{c}$ to Ω_1 or Ω_2, e.g. because of an overlapping as in the picture above, $\vec{c}$ is assigned to the rejection class Ω_0.

rejection class if a feature cannot savely be asserted to any class, we assert it to a special rejection class Ω_0.

Most classifier need a set of labelled samples for training or even afterwards for comparrison. So we need a set of sample features, which we already know the corresponding class number of. We call it the **training set**:

training set a set of features labelled with the class they belong to $\left\{\left(\vec{c}_1, \Omega_{\kappa(1)}\right), \ldots, \left(\vec{c}_n, \Omega_{\kappa(n)}\right)\right\}$ where $\kappa \to \{1, \ldots, K\}$ where K is the number of classes.

No classifier will always make only correct decissions, so we will get misclassifications. We introduce costs for this case. In real world applications it might be usefull to also have costs for correct classifications, which naturally have to be less than the wrong classification for the system to work correctly.

cost function $r(\Omega_\kappa \mid \Omega_\lambda) = \begin{cases} r_{c,\kappa} & \text{if } \Omega_\kappa = \Omega_\lambda \\ r_{0,\kappa} & \text{if } \Omega_\kappa = \Omega_0 \\ r_{f,\kappa} & \text{otherwise} \end{cases}$

where $r_{c,\kappa}$ is the cost of class Ω_κ for a correct classification, $r_{0,\kappa}$ for rejec-

tion and $r_{f,\kappa}$ for misclassification. Furthermore we have $0 \leq r_{c,\kappa} < r_{0,\kappa} < r_{f,\kappa}$.

The easiest cost function however simply assigns costs 0 for correct and costs 1 for misclassifications. We therefore call it the 0/1-cost function:

0/1 cost function $r_{0/1}(\Omega_\kappa \mid \Omega_\lambda) = \begin{cases} 0 & \text{if } \Omega_\kappa = \Omega_\lambda \\ 1 & \text{if } \Omega_\kappa \neq \Omega_\lambda \end{cases}$

Based on this cost concept we introduce the concept of risk.

risk the risk V is simply the average over all costs: $V(\delta) = \sum_\lambda \sum_\kappa p(\Omega_\kappa) \cdot p(\Omega_\lambda \mid \Omega_\kappa) \cdot r(\Omega_\lambda \mid \Omega_\kappa)$
where we call $p(\Omega_\lambda \mid \Omega_\kappa)$ the **confusion probability**:

$$p(\Omega_\lambda \mid \Omega_\kappa) = \int p(\vec{c} \mid \Omega_\kappa)\, \delta(\Omega_\lambda \mid \vec{c})\, d\vec{c}$$

We define an additional **Prüfgröße** $u_\lambda(\vec{c})$ for comparissons.

Prüfgröße $u_\lambda(\vec{c}) = \sum_{\kappa=1}^{K} r(\Omega_\lambda \mid \Omega_\kappa) \cdot p(\Omega_\kappa) \cdot p(\vec{c} \mid \Omega_\kappa)$

Finally you should remeber that every good classifier approximates the Baysean Classifier (see 4.2.1).

4.1.2 Criterions For Classifier

There are various goals when designing a classifier, that can be optimized. In the following we will concentrate on the first two: the risk and the error probability and find an optimal classifier minimizing the risk. But of course any other criterion could be used for finding a corresponding optimal classifier.

- error probability
- risk
- runtime behaviour
- trainability
- robustness
- generalization
- size
- maintenance
- the margin between classes (see SVM 4.7)

4.1.3 Variable Overview

Tabelle 1: Classifier Variable Lookup Table

Variable	Interpretation	Definition
Ω_λ	class with number λ	\
λ, κ	indices for class numbers	\
K	the number of classes	\
$\vec{c}$	feature vector	\
N	the number of features	\
δ	decission rule	$\lambda = \text{argmax}_\kappa \, p\left(\Omega_\kappa \mid \vec{c}\right)$
r	costs for decissions	$r\left(\Omega_\kappa \mid \Omega_\lambda\right) = \begin{cases} r_{c,\kappa} & \text{if } \Omega_\kappa = \Omega_\lambda \\ r_{0,\kappa} & \text{if } \Omega_\kappa = \Omega_0 \\ r_{f,\kappa} & \text{otherwise} \end{cases}$
$V\left(\delta\right)$	risk	$V\left(\delta\right) = \sum_\lambda^K \sum_\kappa^K p\left(\Omega_\kappa\right) \cdot p\left(\Omega_\lambda \mid \Omega_\kappa\right) \cdot r\left(\Omega_\lambda \mid \Omega_\kappa\right)$
$u_\lambda\left(\vec{c}\right)$	Prüfgröße	$u_\lambda\left(\vec{c}\right) = \sum_\kappa^K r\left(\Omega_\lambda \mid \Omega_\kappa\right) \cdot p\left(\Omega_\kappa\right) \cdot p\left(\vec{c} \mid \Omega_\kappa\right)$

4.2 Statistical Classifiers

We start this chatpter by listing some arguments for using statistical models:

- Signals characterize a statistical process
- We can use the optimal Bayes classifier
- We can make use of a known statistical model for the classification
- With mixtures we can also combine different distributions respectively approximate difficult distributions with simpler ones
- We can use marginalization to get rid of unwanted parameters (e.g. the third dimension in 2D pictures of 3D objects)
- We can use methods like, ML, MAP and EM to estimate parameters.

Statistical classifiers are espeically useful in the following cases:

- when features are classwise approximately gaussian distributed
- when features are classwise approximately statistically independent
- the probability denisty function of the classes is known
- a mixture of gaussians to approximate the pdf can be estimated

Applying a statistical classifier is composed of many different steps, that can be summatrized like this:

Algorithm 15 Applying A Statistical Classifier

1. Preliminary
   - collect a representative set of labeled sample patterns
   - preprocessing
   - feature computation
   - feature selection (decide for a fixed number of good features)
   - partition the labeled sample set into two parts for training and testing
2. Learning/Training Phase
   - estimate a-priori probabilities for the classes $p(\Omega_\kappa)$
   - estimate the probability density functions of the classes $p(\vec{c} \mid \Omega_\kappa)$
3. Workphase
   - get a new pattern, preprocess it and extract feature vector $\vec{c}$
   - compute Prüfgrößen $u_\kappa(\vec{c})$ for each class
   - decide for the class corresponding to the highest Prüfgröße
4. Fine Tuning
   - estimate the error rate ERR
   - tune step 1, repeat step 2 and 3 until a reasonable error rate has been stabilized

   (for both see Chapter 6)

Furthermore refer to Chapter 10 to look up some statistical terms and properties, that will be used in this chapter.

4.2.1 Optimal Classifier (Bayes)

Those who refuse Bayes techniques only show - by arguments used - their ignorance about what Bayes techniques are (JAYNES).

In this chapter we will try to find the optimal classifier, i.e. a classifier δ^* with minimal risk:

$$V(\delta^*) = \min_{\delta} V(\delta)$$

our risk looks like this:

$$V(\delta) = \sum_{\kappa=0}^{K} p(\Omega_\kappa) \cdot \sum_{\lambda=1}^{K} r(\Omega_\lambda \mid \Omega_\kappa) \int_{R_c} p(\vec{c} \mid \Omega_\kappa) \cdot \delta(\Omega_\kappa \mid \vec{c})\, dc$$

or using the Prüfgröße from above:

$$V(\delta) = \int_{R_c} \sum_{\kappa=0}^{K} u_\kappa(\vec{c}) \cdot \delta(\Omega_\kappa \mid \vec{c})\, dc$$

where R_c is the feature space and $\vec{c} \in R_c$. We see that minimizing $V(\delta)$ corresponds to a minimization of the Prüfgröße $u_\kappa(\vec{c})$:

$$\min_\delta V(\delta) \triangleq \min_\kappa u_\kappa(\vec{c})$$

integrating over R_c:

$$\sum_{\kappa=0}^{K} u_\kappa(\vec{c}) \cdot \delta(\Omega_\kappa \mid \vec{c}) \geq \sum_{\kappa=0}^{K} \min_\kappa u_\kappa(\vec{c}) \cdot \delta(\Omega_\kappa \mid \vec{c})$$

since $\delta(\Omega_\kappa \mid \vec{c})$ is either 0 or 1, it only depends on $\min_\kappa u_\kappa(\vec{c})$.

Now we formulate the **optimal decission rule**:

$$\delta^*(\Omega_\lambda \mid \vec{c}) = \begin{cases} 1 & \text{if } u_\lambda(\vec{c}) = \min_\kappa u_\kappa(\vec{c}) \\ 0 & \text{if } \lambda \neq \kappa \end{cases}$$

Algorithm 16 Finding An Optimal Classifier

1. compute the risk $V(\delta)$: confusion probability + costs.
2. minimize the risk in respect to the decission rule δ.

Warning It is not sufficient to optimize the error probability unreflected. E.g. think about a medicinical application to recognize tumors. The probability for a tumor is only 0.001%, so the optimization would simply always say "No" to get a very low error probability, but that is not what we wanted. The solution is to optimize in respect to certain Lagrange constraints (see 8.10).

In Summa The optimal classifier, minimizing the risk $V(\delta)$, computes the $K+1$ Prüfgrößen $u_\kappa(\vec{c})$ and decides fpr the class Ω_λ, whose Prüfgröße shows the minimal value.

Baysean Classifier

The Baysean Classifier is such an optimal classifier. It's principle to minimize the Prüfgröße is to maximize the a-posteriori probabilities. The Bayes Classifier has three preconditions:

- no rejection class Ω_0

- the optimal decission rule $\delta^* (\Omega_\kappa \mid \vec{c}) = \begin{cases} 1 & \text{if } u_\kappa (\vec{c}) = \min_\kappa u_\kappa (\vec{c}) \\ 0 & \text{if } \lambda \neq \kappa \end{cases}$
- a $0/1$ cost function $r_{0/1}$

A feature is assigned to the class which maximizes the a-posteriori probability:

$$\lambda = \text{argmax}_\kappa \, p (\Omega_\kappa \mid \vec{c})$$

Theory

1. abolish the cost factor, by mkaing the cost independent of the class
   false classification: $r (\Omega_\kappa \mid \Omega_\lambda) = r_f$
   correct classification $r (\Omega_\kappa \mid \Omega_\kappa) = r_c$
   rejection: $r (\Omega_\kappa \mid \Omega_0) = r_0$.
2. update Prüfgröße
   $u_\lambda (\vec{c}) = \sum_{\kappa=1}^{K} r \cdot p (\Omega_\kappa) \cdot p (\vec{c} \mid \Omega_\kappa)$
3. update risk
   $V (\delta) = \int_{R_c} \sum_{\kappa=1}^{K} u_\kappa (\vec{c}) \cdot \delta (\Omega_\kappa \mid \vec{c}) \, d\vec{c}$ becomes aplying the new cost function
   $V (\delta) = p_c \cdot r_c + p_f \cdot r_f + p_0 \cdot r_0$
   Because of the $0/1$ cost function and the precondition of having no rejection class, $p_c \cdot r_c = p_0 \cdot r_0 = 0$.
   That means the Prüfgröße is simplified to:

   $$u_\lambda (\vec{c}) = \sum_{\substack{\kappa = 1 \\ \kappa \neq \lambda}}^{K} p (\Omega_\kappa) \cdot p (\vec{c} \mid \Omega_\kappa)$$

   This is going to be minimal, if the element removed from the sum ($\kappa = \lambda$) is the biggest one. Since it is set to zero because of $r_c = 0$.
4. maximize a-posteriori probability
   $\kappa = \text{argmax}_\kappa \, p (\Omega_\kappa \mid \vec{c})$, aplying the rule of Bayes (see 10.3)
   $\kappa = \text{argmax}_\kappa \, \frac{p(\vec{c} \mid \Omega_\kappa) p(\Omega_\kappa)}{p(\vec{c})}$, $p (\vec{c})$ is constant (respectively independent of maximizing over κ) and thus unimportant for optimization
   $\kappa = \text{argmax}_\kappa \, p (\Omega_\kappa) \, p (\vec{c} \mid \Omega_\kappa)$, because of reasons of numerical robustness, we scale everything down using the log-likelihood:
   $\kappa = \text{argmax}_\kappa \log p (\Omega_\kappa) \log p (\vec{c} \mid \Omega_\kappa)$

$p(\vec{c} \mid \Omega_\kappa)$ is the probability density function of class Ω_κ.

Most commonly taken for that is gaussian distribution:

$$p(\vec{c} \mid \Omega_\kappa) = \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$$

$p(\vec{c} \mid \Omega_\kappa)$ is they key. If however we have no information about the density, there are some approaches to estimate it: 10.5.

In Summa

The BAYES-classifier, minimizing the confusion probability under forced decissions (no rejection class), computes k a-posteriori probabilities and decides for the class showing the maximal a-posteriori probability.

Properties

- 0/1 cost/loss function: $r(\Omega_\kappa, \Omega_\lambda) = \begin{cases} 0 & \text{if } \Omega_\lambda = \Omega_\kappa \\ 1 & \text{if } \Omega_\lambda \neq \Omega_\kappa \end{cases}$
- no rejection class Ω_0
- requires probability information: a-priori, a-posteriori, class dependent denisty
- optimal decision rule: $\lambda = \text{argmax}_\lambda\, p(\Omega_\lambda \mid \vec{c})$ (choose the class with the highest a-posteriori probability)
  Proof:

  $$\lambda = \min \sum_{\lambda}^{\kappa} r(\Omega_\kappa, \Omega_\lambda), p(\Omega_\kappa \mid \vec{c}) = \text{argmin}_\lambda\, (1 - p(\Omega_\lambda \mid \vec{c})) = \text{argmax}_\lambda\, p(\Omega_\lambda \mid \vec{c})$$

- the computation of the Prüfgröße $u_\kappa(\vec{c})$ is numerically a computation of K scalar products
- minimizing p_f equals maximizing p_c, since $p_c + p_f + p_0 = 1$

Pro/Contra

\+ optimal. There is no better one.

\- requires probability density information for the field of application.

\- costs are independent of the classes (e.g. the costs for classifying a healthy patient ill are the same for classfying a ill patient healthy)

\- no rejection class

4.2.2 Gaussian Classifier

Assumption $p(\vec{c} \mid \Omega_\kappa) = \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$

Method

1. a-priori probability: $p(\Omega_\kappa) = \frac{\#\vec{c}_i}{\#\Omega_\kappa}$
2. use ML, MAP to determine $\vec{\mu}_\kappa, \Sigma_\kappa$:

$$\hat{\mu}_\kappa = \frac{1}{N_\kappa} \sum_{j=1}^{N_\kappa} \vec{c}_{j(\kappa)}$$

where $\vec{c}_{j(\kappa)}$ is a feature associated to class Ω_κ and

$$\hat{\Sigma}_\kappa = \frac{1}{N_\kappa} \sum_{j=1}^{N_\kappa} \left(\vec{c}_{j(\kappa)} - \hat{\mu}_\kappa\right) \left(\vec{c}_{j(\kappa)} - \hat{\mu}_\kappa\right)^T$$

Decision Boundary

from Bayes: $\lambda = \operatorname{argmax}_\lambda p(\Omega_\lambda) \cdot \mathcal{N}(\vec{c}, \vec{\mu}_\lambda, \Sigma_\lambda)$
log likelyhood: $\lambda = \operatorname{argmax}_\lambda \log p(\Omega_\lambda) + \log \mathcal{N}(\vec{c}, \vec{\mu}_\lambda, \Sigma_\lambda)$
einsetzten: $\lambda = \operatorname{argmax}_\lambda \log p(\Omega_\lambda) - \frac{1}{2} \log \det(2\pi\Sigma_\lambda) - \frac{1}{2} (\vec{c} - \vec{\mu}_\lambda)^T \Sigma_\lambda^{-1} (\vec{c} - \vec{\mu}_\lambda)$
As you can see the first part is independent of the observation and the second part (the former exp part) resembles the Mahalonobis distance.

Pro/Contra

- quadratic in components of $\vec{c}$ (see also 3.1.15)

\+ linear decision boundary if $\forall\kappa : \Sigma\kappa = \Sigma$

\+ reasonable memory requirements (information can be derived from the paramertic family and statistical knowledge)

Tricks

the coefficients of a quadratic function (here decision boundary) can be estimated linearly.

1. Decision Boundary: $\hat{\delta}_\kappa(\overrightarrow{c}) = \sum_{\substack{i,j \\ i \le j}} a_{ij} c_i c_j$
2. Least Square: $\left(\sum_{\substack{i,j \\ i \le j}} a_{ij} c_i c_j - \hat{\delta}_\kappa(\overrightarrow{c}) \right)^2 \to \min$

3. Matrix Form: $\begin{pmatrix} c_0^1 c_0^1 & c_0^1 c_1^1 & \dots \\ c_0^2 c_0^2 & c_0^2 c_1^2 & \dots \\ \vdots & \ddots & \vdots \\ c_0^d c_0^d & \dots & c_d^d c_d^d \end{pmatrix} \begin{pmatrix} a_{00} \\ a_{01} \\ \vdots \\ a_{dd} \end{pmatrix} = \begin{pmatrix} \delta_\kappa \left(\overrightarrow{c^1} \right) \\ \delta_\kappa \left(\overrightarrow{c^2} \right) \\ \vdots \\ \delta_\kappa \left(\overrightarrow{c^d} \right) \end{pmatrix}$

4.2.3 Mixture Densities

Idea Use a linear combination of gaussians (or other standard denisties) to approximate the real distribution.

$$p\left(\vec{c} \mid \Omega_\kappa\right) = \sum_{e=1}^{M} p\left(\Omega_e\right) \mathcal{N}\left(\vec{c}; \vec{\mu}_e; \Sigma_e\right)$$

where M is the number of gaussians. A mixture density is thus defined as a weighted sum of standard densities. In our case we have chosen a convex combination of gaussians to illustrate the principle. Of course this mixture has to fulfill the stochastic criterion (10.3). We can verify this by Integration:

$$\int p\left(\vec{c} \mid \Omega_\kappa\right) d\vec{c} = \sum_{e=1}^{M} p\left(\Omega_e\right) \int \mathcal{N}\left(\vec{c}; \vec{\mu}_e; \Sigma_e\right) d\vec{c} = \sum_{e=1}^{M} p\left(\Omega_e\right) = 1$$

Parameter Estimation

[A] Maximum Likelihood (see 8.1)

$$\hat{\theta} - \operatorname{argmax}_\theta \prod_{m=1}^{N} p\left(\vec{c}_m; \theta\right) = \operatorname{argmax}_\theta \sum_{m=1}^{N} \log p\left(\vec{c}_m; \theta\right)$$

We have to seperately estimate the parameters for every density of the mixture.

Example

given $\vec{c} \in \mathbb{R}^{500}, M = 300$
we would have to estimate

$$\text{numberof } p_{1;l} + \text{numberof } \vec{\mu}_{1;l} \cdot \text{dof}\left(\vec{\mu}_{1;l}\right) + \text{numberof } \Sigma_{1;l} \cdot \text{dof}\left(\Sigma_{1;l}\right) =$$

$$= 300 + 300 \cdot 500 + 300 \cdot \frac{501 \cdot 500}{2} = 37725500$$

since ML already takes quite a while for a few paramters, we should not even consider trying to estimate this amount of parameters by ML, and rather use the related EM-Algorithm. A second problem, and argument to take EM, is the high dimension of the feature space, which quickly leads to the Curse Of Dimension (8.18).

[B] Expectation Maximization (see 8.3)

1. what we need:

   - a training set $\{\vec{c}_1, \ldots, \vec{c}_n\}$
   - a probability denisty function $p(\vec{c}; \mathcal{B}) = \sum_{l=1}^{M} p(l) \mathcal{N}(\vec{c_i}; \vec{\mu}_l, \Sigma_l)$
   - the parameter vector to estimate $\mathcal{B} = \{p_1 \ldots p_M, \vec{\mu}_1 \ldots \vec{\mu}_M, \Sigma_1 \ldots \Sigma_M\}$
   - observable information(X): $\vec{c}_i$ the feature vectors
   - hidden information (Y):l the indices of the convex combination
   - the Q-function:

     $$Q = \sum_{l=1}^{M} \sum_{j=0}^{N} \frac{p(l)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu}_l^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p(k)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu}_k^{(i)}, \Sigma_k^{(i)}\right)} \cdot \log\left(\hat{p}(l)^{(i+1)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu}_l^{(i+1)}, \Sigma_l^{(i+1)}\right)\right)$$

     where l corresponds to one part of the convex combination, j sums over all features and i is the current step of the iteration (the i-th estimate for $\hat{\mathcal{B}}$).

2. start:

   Estimation Step: $p(l)$

   $$\frac{\partial Q}{\partial \hat{p}(l)^{(i+1)}} = \sum_{l=1}^{M} \sum_{j=0}^{N} \frac{\hat{p}(l)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_l}^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p(k)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)} \cdot \frac{1}{\hat{p}(l)^{(i+1)}} + \eta$$

   Remark 1: $\sum_{l=1}^{M}$ can be dropped, since for estimation one parameter is sufficient
   Remark 2: $\sum_{k=1}^{M} p(k)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)$ must sum up to one (see 10.3), therefore we add the condition: $+\eta\left(\sum_k^M p_k^{(i+1)} - 1\right)$ which becomes $+\eta$ after the partial derivitaion performed above. Then

   $$\frac{\partial Q}{\partial \hat{p}(l)^{(i+1)}} = \sum_{j=0}^{N} \frac{\hat{p}(l)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_l}^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p(k)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)} \cdot \frac{1}{\hat{p}(l)^{(i+1)}} + \eta = 0$$

   is our term for optimization.

3. compute η:

   $$\eta : \sum_{j=0}^{N} \frac{\hat{p}(l)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_l}^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p(k)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)} = -\eta \cdot \hat{p}(l)^{(i+1)}$$

multiplying both sides with $\sum_{l=1}^{M}$

$$\sum_{j=0}^{N} \frac{\sum_{l=1}^{M} \hat{p}\left(l\right)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_l}^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p\left(k\right)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)} = -\eta \cdot \sum_{l=1}^{M} \hat{p}\left(l\right)^{(i+1)}$$

reducing the fraction

$$\sum_{i=1}^{N} 1 = -\eta$$

and

$$\eta = -N$$

inserting this result into our optimization term

$$\sum_{j=0}^{N} \frac{\hat{p}\left(l\right)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_l}^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p\left(k\right)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)} = -N \cdot \hat{p}\left(l\right)^{(i+1)}$$

4. now we can give a closed formula:

$$\hat{p}\left(l\right)^{(i+1)} = \frac{1}{N} \sum_{j=0}^{N} \frac{\hat{p}\left(l\right)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_l}^{(i)}, \Sigma_l^{(i)}\right)}{\sum_{k=1}^{M} p\left(k\right)^{(i)} \cdot \mathcal{N}\left(\vec{c}_j; \vec{\mu_k}^{(i)}, \Sigma_k^{(i)}\right)}$$

Estimation of μ and Σ

(see also 9.8)

remember that:

$$\mathcal{N}\left(\vec{c}_i; \vec{\mu_l}, \Sigma_l\right) = \frac{1}{\left(2\pi\right)^{\frac{d}{2}} \sqrt{\det\left(\Sigma_l\right)}} \cdot \exp^{-\frac{1}{2}\left(\vec{c}_j - \vec{\mu}_l\right)^T \Sigma_l^{-1} \left(\vec{c}_j - \vec{\mu}_l\right)}$$

1. Insertion I: Vector Differential

   The gradient: $\nabla_{\vec{x}} F\left(\vec{x}\right) = \left(\frac{\partial F(\vec{x})}{\partial \vec{x}_n}\right)_{1 \leq k \leq n}, \vec{x} \in \mathbb{R}^n$

   (a) LinAlg I: consider

   $$\nabla_{\vec{y}} \left(\vec{y}^T A \vec{y}\right), A \in \mathbb{R}^{n \times n}, \vec{y} \in \mathbb{R}^n$$

   $$\nabla_{\vec{y}} \left(\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}^T \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right) = \frac{\partial}{\partial_{y_k}} \left[\sum_j \left(\sum_i y_i a_{ij} \right) y_j \right]$$

   applying product rule

   $$\sum_j a_{kj} y_j + \sum_i y_i a_{ik} = \left(A\vec{y} + A^T \vec{y}\right)_k$$

   that means:

   $$\nabla_{\vec{y}} \left(\vec{y}^T A \vec{y}\right) = \left(A\vec{y} + A^T \vec{y}\right)$$

(b) LinAlg II: consider

$$\nabla_{\vec{y}}\left(\left(\vec{x}-\vec{y}\right)^T A\left(\vec{x}-\vec{y}\right)\right) = -\left(A\left(\vec{x}-\vec{y}\right)+A^T\left(\vec{x}-\vec{y}\right)\right)$$

(c) estimating $\hat{\vec{\mu}}_l^{(i+1)}$

$$\nabla_{\mu_l} Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) = \sum_l^M \sum_j^N \frac{\hat{p}_l^{(i)} \mathcal{N}\left(\vec{c}_j, \vec{\mu}_l^{(i)}, \Sigma_l^{(i)}\right)}{\sum_k^M \hat{p}_k^{(i)} \mathcal{N}\left(\vec{c}_j, \vec{\mu}_k^{(i)}, \Sigma_k^{(i)}\right)} \cdot \nabla_{\mu_l}\left[-\frac{1}{2}\left(\vec{c}_j-\vec{\mu}_l\right)^T \Sigma_l^{-1}\left(\vec{c}_j-\vec{\mu}_l\right)\right]$$

we define F_j as

$$F_j = \sum_l^M \frac{\hat{p}_l^{(i)} \mathcal{N}\left(\vec{c}_j, \vec{\mu}_l^{(i)}, \Sigma_l^{(i)}\right)}{\sum_k^M \hat{p}_k^{(i)} \mathcal{N}\left(\vec{c}_j, \vec{\mu}_k^{(i)}, \Sigma_k^{(i)}\right)}$$

and get

$$\sum_j^N F_j \cdot \Sigma_l^{-1}\left(\vec{c}_j-\vec{\mu}_l\right) \doteq 0$$

solving for μ wet get our closed formula:

$$\hat{\vec{\mu}}_l^{(i+1)} = \frac{\sum_j^N F_j \cdot \vec{c}_j}{\sum_m^N F_m}$$

2. Insertion II: Matrix Differential
The gradient: $\nabla_A F(A) = \left(\frac{\partial F(A)}{\partial a_{i,j}}\right)_{1\leq i,j\leq n}, A \in \mathbb{R}^{n\times n}$

(a) LinAlg I: consider

$$\nabla_A A^{-1}$$

$$\nabla_A\left(AA^{-1}\right) = \left(\nabla_A A\right)A^{-1} + A\left(\nabla_A A^{-1}\right) = 0$$

$$-A\left(\nabla_A A^{-1}\right) = \left(\nabla_A A\right)A^{-1}$$

$$\nabla_A A^{-1} = -A^{-1}\left(\nabla_A A\right)A^{-1}$$

(b) LinAlg II: consider

$$\nabla_A \det(A)$$

use the adjoint matrix $A_{i,j}^\sharp \in \mathbb{R}^{(n-1)\times(n-1)}$ (leave out i^{th} row and j^{th} column, see 9.7) and the LaPlace Extension Formula for solving determinants:

$$\det(A) = \sum_{k\neq i}^n a_{i,k} \det\left(A_{i,j}^\sharp\right), i \in \{1,2,\ldots,n\}$$

partially dervied to $a_{i,j}$

$$\frac{\partial \det(A)}{\partial a_{i,j}} = \det\left(A^{\sharp}_{i,j}\right) = a^{\complement}_{i,j}$$

and thus we draw the result that

$$\nabla_A \det(A) = A^{\complement}$$

where $A^{\complement}$ is the complementary matrix (see 9.7) to A.

(c) estimating $\hat{\Sigma}_l^{(i+1)}$
combining (a) and (b), we get

$$\nabla_A \det(A) = \left(A^{-1}\right)^T \det(A)$$

since

$$A^{\complement} = \left(A^{-1}\right)^T \det(A) \cdot \mathrm{Id}^T$$

Now the partial derivation to Σ_l of our optimization term looks like this

$$\nabla_{\Sigma_l}\left(Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right)\right) = \sum_j^N F_j \nabla_{\Sigma_l}\left[\log \hat{p}_l^{(i+1)} - \frac{1}{2}\log\left[(2\pi)^d\right]\right] -$$

$$-\frac{1}{2}\log\det(\Sigma_l) - \frac{1}{2}\left(\vec{c}_j - \vec{\mu}_l\right)^T \Sigma_l^{-1}\left(\vec{c}_j - \vec{\mu}_l\right)$$

using the LinAlg tricks

$$\sum_j^N F_j\left[-\frac{1}{2}\frac{1}{\det(\Sigma_l)}\left(\Sigma_l^{-1}\right)^T \cdot \det(\Sigma_l)\right] - \frac{1}{2}\left[\left(\vec{c}_j - \vec{\mu}_l\right)^T\left(-\Sigma_l^{-1}\right)\left(\nabla_{\Sigma_l}\Sigma_l\right)\Sigma_l^{-1}\left(\vec{c}_j - \vec{\mu}_l\right)\right]$$

$(\nabla_{\Sigma_l}\Sigma_l)$ is just the identity matrix

$$\sum_j^N F_j\left[-\frac{1}{2}\left(\Sigma_l^{-1}\right)^T + \frac{1}{2}\Sigma_l^{-1}\right]\left(\vec{c}_j - \vec{\mu}_l\right)\mathrm{Id}\left(\vec{c}_j - \vec{\mu}_l\right)^T\Sigma_l^{-1} \doteq 0$$

$-\frac{1}{2}\left(\Sigma_l^{-1}\right)^T + \frac{1}{2}\Sigma_l^{-1}$ equals Σ_l^{-1}

$$\sum_j^N F_j \Sigma_l^{-1}\left(\vec{c}_j - \vec{\mu}_l\right)\left(\vec{c}_j - \vec{\mu}_l\right)^T\Sigma_l^{-1} \doteq 0$$

solving for Σ_l we get our closed formula

$$\hat{\Sigma}_l^{(i+1)} = \frac{\sum_j^N F_j\left(\vec{c}_j - \vec{\mu}_l\right)\left(\vec{c}_j - \vec{\mu}_l\right)^T}{\sum_m^N F_m}$$

Properties

- convex combination of gaussians: $p(\vec{c} \mid \Omega_k) = \sum_l^M p_l \mathcal{N}(\vec{c}_i; \vec{\mu}_l, \Sigma_l)$
  convex combination means, the probabilities sum up to 1.
- use EM to decompose the search space of the parameters, to estimate the pdf parameters more efficiently
- see also Parzen Estimation (4.4.2)

Pro/Contra

+ allows good approximations of complicated distributions

- approximates the world normal

- big memory requirements (the whole training set must be stored and evaluated in every step)

4.3 Parametric Classifiers

The core idea of parametric classifiers is to replace the density by a parametric function, e.g. a polynom. Then of course we also have to replace the probability p by some metric distance measure.

Example

$d_\lambda(\vec{c}) = a_1 \cdot c_1 + a_2 \cdot c_2 + a_3 \cdot c_1 \cdot c_2 + a_4 \cdot c_1^2 + a_5$

The general procedure of parametric classifiers is to approximate the optimal decission rule δ^* by a metric $d_\lambda(\vec{c})$:

$$d_\lambda(\vec{c}) = a_\lambda^T \varphi(\vec{c})$$

We decide for the class that maximizes d:

$$\text{if } d_\kappa(\vec{c}) = \max_\lambda d_\lambda(\vec{c}) \text{ then assign } \vec{c} \in \Omega_\kappa$$

a can be computed by optimization methods, e.g. by minimizing the **mean squared error**:

$$\varepsilon = E\left\{\sum_{\lambda=1}^{K} \left(\delta_\lambda^*(\vec{c}) - d_\lambda(\vec{c})\right)^2\right\}$$

Pro/Contra

+ probability density functions do not have to be computed, estimated or statistically approximated
+ in theory the approach of using a parametric familily of functions is more general than using parametric densities
+ A^* can be computed efficiently by nummerical methods
+ experiments show little to no difference between parametric and statistical classifiers

4.3.1 Polynomial Classifier

The simplest form of parametric functions are polynomials. Designing a polynomial classifier you choose the degree of the polynomial and then compute the parameters.

We have given a metric $d(\vec{c}) = \begin{pmatrix} d_1(\vec{c}) \\ d_2(\vec{c}) \\ \vdots \\ d_K(\vec{c}) \end{pmatrix}$,
a decission rule to approximate $\delta(\vec{c}) = \begin{pmatrix} \delta_1(\vec{c}) \\ \delta_2(\vec{c}) \\ \vdots \\ \delta_K(\vec{c}) \end{pmatrix}$
and our parameters $A = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{pmatrix}$.

We choose the metric: $d(\vec{c}) = A^T \varphi(\vec{c})$, but of course any other metric could be choosen instead. $\varphi(\vec{c})$ is the seperating function we are looking for.

That gives us the goal of minimizing the expectation of the least square error: $\varepsilon(A) = E\left\{\left(\delta(\vec{c}) - A^T\varphi(\vec{c})\right)^2\right\}$, and we want

$$A^* = \operatorname{argmin}_A \varepsilon(A)$$

A^* is called the **optimal parameter matrix**.

Applying linear algebra, we figure that we have to build the pseudo inverse:

$$A^* = \left[E\left\{\varphi(\vec{c})\,\varphi(\vec{c})^T\right\}\right]^{-1} \cdot E\left\{\varphi(\vec{c})\,\delta(\vec{c})^T\right\}$$

Remeber that the expectation is defined as:

$$E\{g(x)\} = \int g(x)\,p(x)\,dx \simeq \frac{1}{N}\sum_{i=1}^{N} g(x_i)$$

So we get

$$E\left\{\varphi\left(\vec{c}\right)\varphi\left(\vec{c}\right)^T\right\} = \frac{1}{N}\sum_{i=1}^{N}\varphi\left(c_i\right)\varphi\left(c_i\right)^T$$

$$E\left\{\varphi\left(\vec{c}\right)\delta\left(\vec{c}\right)^T\right\} = \frac{1}{N}\sum_{i=1}^{N}\varphi\left(c_i\right)\delta\left(c_i\right)^T$$

To proove this we would build the first partial derivatives and set them equal to zero.

So we have computed our parameter matrix A^*, now we can build the seperating function $\varphi\left(\vec{c}\right)$

$$\varphi\left(\vec{c}\right)^T \cdot A^*$$

Since each class Ω_κ is represented by its own distance metric $d_\kappa\left(\vec{c}\right)$, we have to repeat this procedure for every class. Having $d_\kappa\left(\vec{c}\right) = A^T\varphi\left(\vec{c}\right)$ for every class we classify $\vec{c}$ to the class it has the minimal distance to.

Example

given $\varphi\left(\vec{c}\right) = \begin{pmatrix} 1 \\ c_1 \\ c_2 \end{pmatrix}$ and computed $A^* = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 \end{pmatrix}$,
then we get the decision rule by matrix multiplication:

$$(1 c_1 c_2) \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 \end{pmatrix}$$

The vector multiplied with the first column gives $\varphi_1\left(\vec{c}\right) = \frac{1}{2} + \frac{1}{2}\cdot c_1$
and the vector multiplied with the second column gives: $\varphi_2\left(\vec{c}\right) = \frac{1}{2} - \frac{1}{2}\cdot c_1$.

In Summa

The Polynomial Classifer minimizes the expection of the least square error $E\left\{\left(\delta\left(\vec{c}\right) - d\left(\vec{c}\right)\right)^2\right\}$ by approximated the ideal decision boundary $\delta\left(\vec{c}\right)$ by distance metric function $d\left(\vec{c}\right)$. For that it uses polynom functions $\varphi\left(\vec{c}\right)$, $d\left(\vec{c}\right) = A^T\varphi\left(\vec{c}\right)$.

Properties

- There are various other methods to find A^*, e.g. by iteration, by stochastical approximation, or by a recursive procedure.
- Having a metric distance function, we can easily include a rejection class Ω_0 by adding a threshold θ. If the minimal distance of a feature $\vec{c}$ to a class Ω_κ is greater than the threshold: $\min_\kappa d_\kappa\left(\vec{c}\right) > \theta$, then we reject this feature $\vec{c} \rightarrow \Omega_0$.

- If there are no restrictions for the distance metric function, like number of parameters or a parameter range, then we exactly get the Bayes classifier

Abbildung 37: Proof that polynomial classifier can be equal to the Baysean classifier
we choose the "optimal" metric d^*

$$d^* = \begin{pmatrix} p(\Omega_1 \mid \vec{c}) \\ p(\Omega_2 \mid \vec{c}) \\ \vdots \\ p(\Omega_K \mid \vec{c}) \end{pmatrix}$$

the vector composed of the a-posteriori probabilities.
If the decission rule δ was choosen according to the optimal one δ^*, we get:

$$(d^* = E\{\delta^* \mid \vec{c}\})$$

Pro/Contra

- the degree of the polynomial is limited by the number of coefficients

- because polynoms of degree p with n variables have $\begin{pmatrix} n+p \\ n \end{pmatrix}$ coefficients, the degree is practically limited to $p = 2$ or $p = 3$.

4.3.2 Least Square Estimation

Alternatively to the method above, we can also use Least Square Estimation to approximate the optimal decission rule δ^*:

$$\sum_{\lambda=1}^{K} \sum_{i=1}^{N} \|\delta_\lambda(\vec{c}_i) - d_\lambda(\vec{c}_i)\|^2 \to \min$$

we can achieve this by holding λ fixed:

$$\forall \vec{c} : \delta_\lambda(\vec{c}) = d_\lambda(\vec{c})$$

Example

given $\delta_\lambda(\vec{c}) = \begin{cases} 1 & \text{if } \vec{c} \in \Omega_\lambda \\ 0 & \text{if } \vec{c} \notin \Omega_\lambda \end{cases}$ and a parametric function of degree 5.

$$d_\lambda(\vec{c}) = \begin{pmatrix} c_1 & c_2 & c_1c_2 & c_1^2c_2^2 & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix}$$

that gives

$$\vec{c} \cdot \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

where

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = A^T \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

and finally using SVD we compute the pseudo inverse

$$\left(A^T \cdot A\right)^{-1} A^T$$

and are able to solve the above equation.

Properties

- To reduce the dimension Parameter Tying can be applied. E.g. try to set $a_1 = a_2$, or something like that, that seems to make sense in your application. For more on Parameter Tying, see 8.14.

4.3.3 Linear Regression

In general Linear Regression means the estimation of a statistic variable y (in our case $\delta_\kappa(\vec{c})$) that is dependent on a vector $\vec{a}$. The relation between $\vec{a}$ and

y is assumed to be linear and approximated as such, giving the technique its name. Regression, in general, is the problem of estimating a conditional expected value. While this relation is linear, the function approximating the relation is not restricted to linearity (i.e. lines) and can also be quadratic or of higher dimension (i.e. arbitrary curves).

In our case we want to approximate a probability density function by a parametric function, i.e. a polynomial.

$$\delta_\kappa(\vec{c}) = \sum_{i=1}^{d} a_i c_i + a_0$$

where d is the degree of the polynomial.

Computation of a_i

To get the parameters of our polynomial a_i we apply Linear Algebra:

we can write the above formula like this $(a_0, a_1, \ldots, a_d) \cdot \begin{pmatrix} 1 \\ c_1 \\ c_2 \\ \vdots \\ c_d \end{pmatrix}$

and get a system of linear equations: $\begin{pmatrix} 1 & c_1^1 & \ldots & c_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & c_n^d & \ldots & c_n^d \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_d \end{pmatrix} = \begin{pmatrix} \delta(\vec{c}^1) \\ \vdots \\ \delta(\vec{c}^n) \end{pmatrix}$

which we can solve with: $\begin{pmatrix} a_0 \\ \vdots \\ a_d \end{pmatrix} = M^t \begin{pmatrix} \delta(\vec{c}^1) \\ \vdots \\ \delta(\vec{c}^n) \end{pmatrix}$ where M^t is the pseudo-inverse of the above matrix.

4.3.4 Logistic Regression

In Logistic Regression we want to approximate the Bayesean a-posteriori quotient by functions linear in $\vec{c}$ (e.g. a polynomial) ensuring at the same time that they sum up to one and remain in $[0; 1]$:

$$\log \frac{p(\Omega_\kappa \mid \vec{c})}{p(\Omega_\lambda \mid \vec{c})} \to \vec{\alpha}^T \vec{c} + \alpha_0$$

For the stochastic criterion we use Lagrange constraints verifying the postulation.

Theory

assume gaussian distribution.

$$\log \frac{p\left(\Omega_\kappa \mid \vec{c}\right)}{p\left(\Omega_\lambda \mid \vec{c}\right)} = \log \frac{p\left(\Omega_\kappa\right)}{p\left(\Omega_\lambda\right)} + \log \frac{\frac{1}{\sqrt{2\pi\Sigma_\kappa}} \exp^{-\frac{1}{2}(\vec{c}-\mu_\kappa)^T \Sigma_\kappa^{-1}(\vec{c}-\mu_\kappa)}}{\frac{1}{\sqrt{2\pi\Sigma_\lambda}} \exp^{-\frac{1}{2}(\vec{c}-\mu_\lambda)^T \Sigma_\lambda^{-1}(\vec{c}-\mu_\lambda)}}$$

$$\cdots = \log \frac{p\left(\Omega_\kappa\right)}{p\left(\Omega_\lambda\right)} + \left(\mu_\kappa - \mu_\lambda\right)^T \Sigma^{-1} \vec{c} + \frac{1}{2} \left(\mu_\kappa - \mu_\lambda\right)^T \Sigma^{-1} \left(\mu_\kappa - \mu_\lambda\right)$$

We see that we have achieved linearity in the components of $\vec{c}$. Thus we can express the above as a linear function:

$$\alpha_0 + \vec{\alpha}^T \vec{c}$$

For arbitrary probability density functions we can estimate α_0 and $\vec{\alpha}^T$ by Maximum Likelihood (see 8.1).

$$\prod_i p\left(\Omega_{\delta(\vec{c})} \mid \vec{c}\right)$$

where $\delta\left(\vec{c}\right)$ is a class number given by a trainer. We set the ML parameter vector to $\theta = \left\{ \begin{array}{c} \alpha_0 \\ \vdots \\ \alpha_{K-1} \end{array} \right\}$ and get

$$\hat{\Theta} = \text{argmax}_\Theta \, L\left(\Theta\right) = \text{argmax}_\Theta \, p\left(\Omega_{\delta(\vec{c})} \mid \vec{c}\right)$$

Derive $L\left(\Theta\right)$ and solve for critical points. Since this is going to be non-linear, methods like Newton Iteration (see 8.5) can be applied.

Decision Boundary

Consider maximal a-posteriors with the Bayes classifier. In accordance here we have the objective: $\log 1 = 0$

$$\delta_{\kappa,\lambda} = \left\{ \vec{c} \mid \alpha_0 + \vec{\alpha}^T \vec{c} = 0 \right\}$$

Properties

- results in the same linear function as LDA (see 3.1.15) does, but is more general than LDA, since it makes less assumptions

4.4 Non-Parametric Classifiers

In this chapter we look into classifier, which do neither use probability density functions nor parametric approximations of the same.

Pro/Contra

+ Does not require any parameters and parameter estimation

- While it was sufficient to store a parameter vector in the previous two sections, with the non-parametric approach we have to save the whole sample set

4.4.1 Direct Estimation

Direct Estimation makes use of volumes. The goal is to estimate

$$\hat{p}\left(c \mid \Omega_\kappa\right) = \frac{P_\kappa}{V}$$

Tabelle 2: Direct Estimation Variable Lookup Table

variable	interpretation
P_κ	the probability that a feature from class Ω_κ lies within the volume V
$\hat{P}_\kappa$	$\frac{m_\kappa}{N_\kappa}$, $P_\kappa$estimated from a sample set
V	the Volume containing $\vec{c}$
m	the number of patterns in V
N	the sample set
probability	estimation
$p\left(\Omega_\kappa\right)$	$\hat{p}\left(\Omega_\kappa\right) = \frac{N_\kappa}{N}$
$p\left(\vec{c}\right)$	$\hat{p}\left(\vec{c}\right) = \frac{m}{N \cdot V}$
$p\left(\vec{c} \mid \Omega_\kappa\right)$	$\hat{p}\left(\vec{c} \mid \Omega_\kappa\right) = \frac{\hat{P}_\kappa}{V} = \frac{m_\kappa}{N_\kappa \cdot V}$
$p\left(\Omega_\kappa \mid \vec{c}\right)$	$\hat{p}\left(\Omega_\kappa \mid \vec{c}\right) = \frac{\hat{p}(\Omega_\kappa) \cdot \hat{p}(\vec{c} \mid \Omega_\kappa)}{\hat{p}(\vec{c})} = \frac{m_\kappa}{m}$

Example

The simplest way to apply this theory is Histogram Estimation. In this case the Volume is constant, since the area of interest is equally partitioned into intervals.

E.g. throw a dice 1000 times, draw a histogram and estimate the probabilities from it.

Properties

- good estimations postulate: $N_\kappa \to \infty$ and $V \to 0$

Pro/Contra

+ it works without saving the sampleset, since the result of the estimaton is just an estimate of the parameter vector of the previous two approaches

- Curse Of Dimensionality (see 8.18)
  especially unfit for analogue values in $\mathbb{R}^n$, imagine a histogram of such values: There would be an endless number of gaps and only single entries on petty specks. That again means, referring to the Curse Of Dimensionality, that apllying distance measures, all features will be clos to each other.

+ A possible solution is to choose V variable (great activity $\rightarrow$ small V, low activity $\rightarrow$ broad V) and set m_κ to a constant value. That means we count per area and can avoid most gaps.

- Yet a variable V also means, that now we have to store the complete sample set though.

4.4.2 Parzen Windowing

The initial idea of Parzen Windowing was to compare the statistics of incoming patterns with the sample set:

$$p\left(\vec{c} \mid \Omega_\kappa\right) = \frac{1}{N_\kappa} \sum_{j=1}^{N_\kappa} \delta\left(\vec{c} - \vec{c}_{j\kappa}\right)$$

when the incoming pattern is to befound in the sample set, the classifier δ will return 1, else 0:

$$\delta\left(x, y\right) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$$

This is a good approach, but the probability that an exact copy of the incoming pattern is stored in the sample set is relatively low. Here comes Parzen's second idea into play: We use gaussian bells to blur our peaks (the sample patterns in the sample set $\vec{c}_j$) to make them much broader:

$$p\left(\vec{c} \mid \Omega_\kappa\right) = \frac{1}{N_\kappa} \sum_{j=1}^{N_\kappa} f\left(\frac{\vec{c} - \vec{c}_{j\kappa}}{h_n}\right)$$

This now approach converges to the density in the quadratic mean.

Tabelle 3: Parzen Windowing Variable Lookup Table

variable	interpretation
N_κ	the sample set
$\vec{c}_{j\kappa}$	j^{th} vector of the sample vector, labeled to belong to class κ
f	windowing function (usually gaussian bells)
h_N	determines the width of the window function

Properties

- because of the stochastic criterion, window functions f have to fullfill two requirements
  1. $f(x) \geq 0$
  2. $\int_{-\infty}^{+\infty} f(x)\, dx = 1$
- combining this with Parameter Tying (see 8.14), we assume all covariance matrices are equal: $\forall \kappa : \Sigma_\kappa = \Sigma$

Pro/Contra

- the whole sample set has to be stored and for each sample vector parameters for an additional gaussian bell

\+ a possible solution is to combine several gaussian bells to one mixture (see 4.2.3)

- that again makes the system less dynamical for adding new vectors to the sample set

4.4.3 Nearest Neighbour (NN)

The Nearest Neighbour classifier is probably the most intuitive and widest known classification method. It stores a couple of reference vectors $\vec{c}_\kappa$ for each class, compares the distance of a new pattern $\vec{c}$ to these references and decides for the class corresponding to the reference vector having the smallest distance to the new pattern.

Properties

- requires a metric distance measure, e.g. the euclidean distance: $d(\vec{c}, \vec{c}_\kappa) = \sqrt{(\vec{c} - \vec{c}_\kappa)^T (\vec{c} - \vec{c}_\kappa)}$
- approximates the a-posteriori probabilities: $p(\Omega_\kappa \mid \vec{c}) = \frac{m_\kappa}{m}$
- the classification error ERR can be bounded to: $\mathrm{ERR}_{\mathrm{Bayes}} \leq \mathrm{ERR}_{\mathrm{NN}} \leq 2 \cdot \mathrm{ERR}_{\mathrm{Bayes}}$
- the nearest neighbour can efficiently be found using branch & bound search (see 3.2.14)

Pro/Contra

+ easy to implement, fast and intuitive

+ at least half as good as the Baysean classifier

- makes no use of any statistical background knowledge

- it is very difficult to find a good metric distance measure in high dimensional spaces

4.4.4 K-Nearest Neighbour (K-NN)

K-Nearest Neighbour is a modification to the standard NN classifier. Here we compare patterns to several reference vectors at once deciding for the class the average of those vectors belongs to. Thresholds decide how many reference features will be used.

K-Threshold Use the K nearest reference vectors

D-Distance Use no reference vectors that are farther away than D

Example

Given metric: euclidean distance, dimension: 1, $K = 3$, $D = 5$, class 1 $\Omega_1 = \{0, 1, 5\}$, class 2 $\Omega_2 = \{8, 10, 11\}$, pattern to classify $c = 6$.

The three (K) nearest samples to c are 5, 8 and 10.

The NN classifier would classify 6, to class Ω_1, since 5 is the nearest sample vector and 5 belongs to class Ω_1.

K-NN however classifies 6 to class Ω_2, since the average of those three ($\frac{2}{3}$) belong to class Ω_2.

Properties

- having two thresholds it is easy to introduce a rejection class:
  - If the minimal occuring distance is above a threshold, the pattern is rejected
  - If K is below a threshold, the pattern is rejected
- in praxis K is choosen between $K = 3$ and $K = 7$

4.5 Classifications Levels

These classifiers partition the classification process in several sublevels. In general we distinguish two kinds of such partitions: seperation in a hierarchy and in sequences.

Hierachy

A hierarchical classification order means that on the root level an aspect is classified that ideally seperates the number of possible classes in half, or allows the application of a specialized classifier with a much better error rate. Classifiers using hierachical classification are decision trees (see below, 4.5.1).

Example

A good example is speech recogition. At the root level the decission could be felt, what language the participant is speaking. At a second one could be determined whether the participiant is female or not. Then at a third level could be decided whether the speaker is of age or not. Eventually in the last level a classifier specialized for the resulting group could be applied. E.g. a classifier specalized to old russian women.

Sequences

Sequential classification is similar. At first a minor classifier classifies the pattern roughly with a small number of features. Then depending on this classification, additional features are computed and the next classifier is choosen. That means features are a temporal sequence of single vectors: $f_1 f_2 \dots f_n$. That also means features now have a variable not necessarily equally long length. This gives our classification problem a new aspect: finding the best mapping between two sequence:

Optimization Problem:

$$\kappa = \text{argmin}_{\kappa,\zeta} \sum_{i=1}^{N} \left\| f_i - f_{\kappa,\zeta(i)} \right\|^2$$

where κ symbolizes the class affiliation and ζ a discrete mapping (e.g. mapping c_1 to $m_1 \dots m_k$).

Ergo we know have two major goals for classifying patterns:

1. mapping of the elements
2. deciding for a class using a distance measure

In respect to application dependent restrictions that help to minimize the search space (e.g. monotony, or λ-contraints)

Example

A good example for this kind of classification is the diagnoses of diseases, because the features are expesinve to compute. At a first step an assistent doctor could decide what features are needed from the patient to classify her disease (e.g. EKG, Radiology,...). Then afterwards a more specialized examination, according to these new features follows. Evenntually a doctor will classify the disease relying on many preclassifications from the previous steps.

Requirements

- a set of prototypes / reference samples
- a proper distance measure

Prototypes

- the average of all samples within one class (statistical classifiers)
- all samples (NN)

Suitable Distance Measures

- average contribution difference: $d_\lambda = \sum_{j=0}^{M-1} (f_{\lambda j} - f_j)$
- average quadratic distance: $d_\lambda = \sum_{j=0}^{M-1} (f_{\lambda j} - f_j)^2$
- **Levenstein Distance**: the smallest distance between two chains of symbol is the smallest amount of replacements, insertions and deletions of symbols to transform the first chain into the other:

$$d_\lambda = \sum_{j=0}^{N} d(\vec{\nu}_{\lambda j}, \vec{\nu}_j)$$

with

$$d(\vec{\nu}_{\lambda j}, \vec{\nu}_j) = \begin{cases} 1 & \vec{\nu}_{\lambda j} \neq \vec{\nu}_j \\ 0 & \vec{\nu}_{\lambda j} = \vec{\nu}_j \end{cases}$$

Pro/Contra

\+ The number of features and the feature space is drastically reduced

\+ Strongly specialized classificators can be applied

\- Overhead for the organisation

\- Many parameters for all the different classifiers

\- Comparisson of samples (e.g. for distance) becomes harder, since they are now also different in length

4.5.1 Decision Trees

4.5.2 Linear Normalization

Linear Normalization is an intuitive and easy method to bring two sequence vectors to the same length, to allow comparisions between them. The idea is to simply strech respectively shrink the new pattern vector until it has the same length as the sample to compare with.

Properties

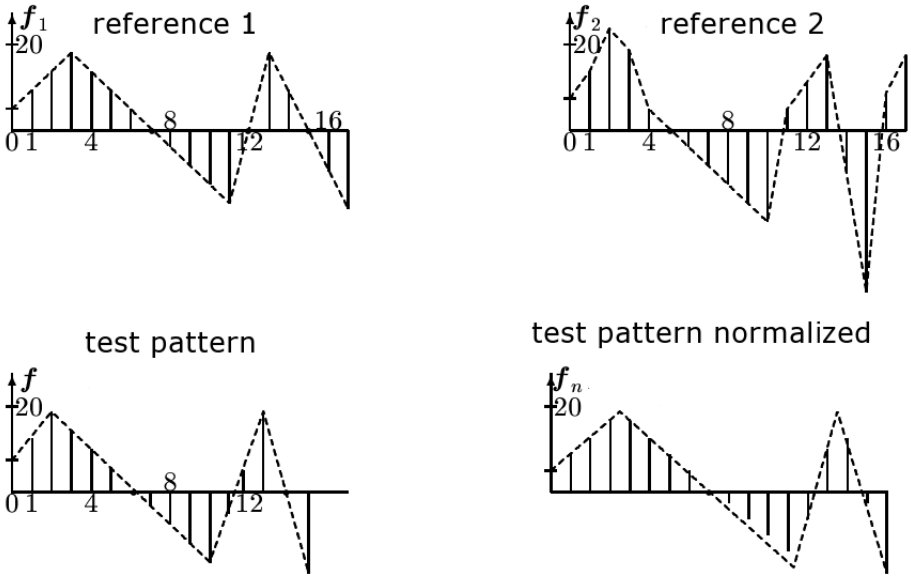
- reference sample vector are usually strechted respectively shrinked to a normalized length to make the classification process more simple and faster

Pro/Contra

\+ intuitive, fast

\- apparently this doesn't work for many applications.
For example imagine speech recognition; the word "sword", spoken "ssswoooord". Linear Normalization would normalize the sequence to something like: "sswwoorrdd" which is entirely nonsense!

Abbildung 38: Linear Normalization



Intuitively the test pattern looks much more like reference pattern 1. Yet linear normalization draws that it is equally equal to both references.

4.5.3 Dynamic Time Wraping (DTW)

The idea of Dynamic Time Wrap is an answer to the problems of Linear Normalization. We still want to expand and shrink the sequence, but no longer linearly. We seek a wraping ω that minimizes the distance:

$$D_\lambda^* = d_\lambda(\omega^*) = \min_\omega d_\lambda$$

The idea is not new, it is just an application of Bellman's Optimality Principle (see 8.19) to our problem: In an optimal sequence of decisions is every subsequence itself an optimal sequence of decisions.

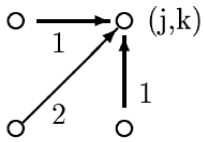
Requirements

- monotonic cost function (no negative costs)
- speperable cost function ($c(a) + c(b) = c(a + b)$)
- independent normalization for each reference pattern

Optimization

The space of every possible mapping between two sequences is immense, therefore we apply different techniques and add some restrictions to it:

1. Predecessor Function

   The idea is only to allow a certain amount of predecessors (see illustration), which may add different costs depending on the path (e.g. delete/insert 1, replace 2).

2. Level Restrictions

   restrict the mapping per level

Properties

- complexity: $O\left(n^2\right)$ (two for loops)
- typical applications: row matching of car stereo pictures, pull a baby with tongs out of the mother womb

4.5.4 Context

Context per se is no classifier, but many classifier can benefit from using context knowledge. An example from natural language would be the probability that 'i' is followed by 't'.

[1] N-Gram The idea of N-Grams is to restrict the number of neighbouring features a feature depends on:

$$p\left(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_{i-n-1}\right)$$

where indices $i-1 \ldots i-n-1$ refer to predecessors of w_i.

Example

Handwriting detection:
Try to detect the follwoing word by yourself

Without context knowledge such words are impossible to classify.[2]

The most promionent application of this form of context knowledge are Hidden Markov Models (see 4.8):

$$p\left(\vec{c}_i \mid \vec{c}_{i-1}\right)$$

where N usually is choosen as 1 or 2, since a higher degree would go beyond the scope of computer computation.

[2] Simplifications We need a couple of additional simplifications to be able to work with context:

1. statistical independency

   features have to be statistically independent of each other:

   $$p\left(A \cap B\right) = p\left(A\right) \cdot p\left(B\right)$$

   respectively

   $$p\left(\vec{c} \mid \Omega\right) = \prod_{j=1}^{N} p\left(\vec{c}_j \mid \Omega_j = \Omega_\kappa\right)$$

   that gives us $K \cdot N$ densities instead of K^N ones.
   Note: This assumption is critical, because although we simplify the statistics, we de facto cancel the possibility of context relations, by saying features are independent of each other.

2. dependency on only the direct predecessors

   $$p\left(\vec{c}_i \mid \vec{c}_{i-1} \ldots \vec{c}_{i-n-1}\right) \simeq p\left(\vec{c}_i \mid \vec{c}_{i-1}\right)$$

3. a-priori knwoledge

   e.g. for speech a lexicon or a dictionary. Such a dictionary can be used for:

   - additional probabilities: $p\left('t' \mid' i'\right)$, the probabilty that 'i' is followed by 't'

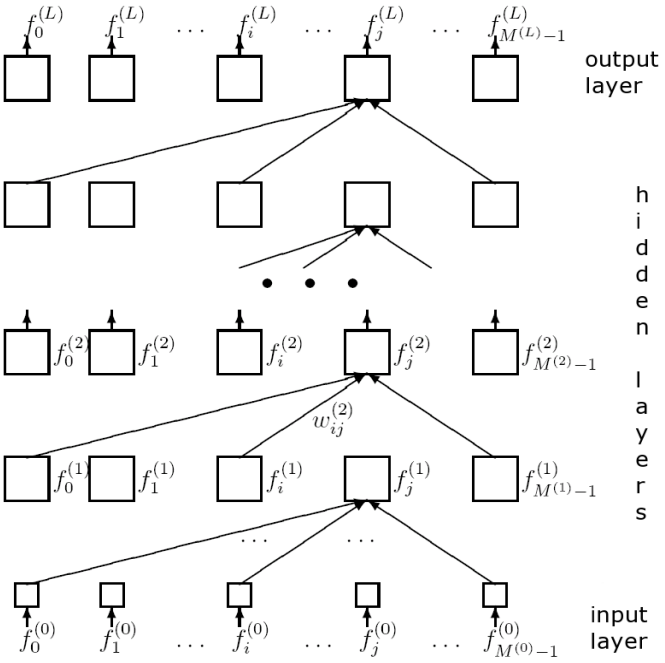
[2] The word is Aluminiumminimum

- a list of all valid words: the classifier has decided that the most probable sequence is the word 'blod', yet 'blod' is not in the dictionary, so the classifier decides for the second best guess 'plot' (or Nr. 3: 'plod').

Having all that we can compute $p(\vec{c}_j \mid \Omega_\kappa)$ applying the Viterbi Algorithm (see 8.21).

4.6 Neural Networks

Abbildung 39: Neural Network (Multilayer Perceptron)



A multilayer perceptron (see below) as example for a neural network.

Neural Networks or **Artificial Neural Networks** intend to reconstruct the human **Natural Neural Network**. Neural networks consist out of a set of nodes (**perceptrons**, **neurons**) with connections between them (**nerves**). The neurons are modelling natural human neurons, little nodes that fire an impulse under certain circumstances (modelled by thresholds). Those neurons are often orderd in layers, where one layer serves as input layer and one as output layer. The layers in between are called the **hidden layers**. The perceptrons have an (usually very simple) internal function or process-function that influences the data sent through them. In restricting the number of perceptrons, the number

of layers and the allowed connections between perceptrons, Neural Networks are modelled.

Components

Perceptron/Neuron nodes with an internal processing function and a threshold β

Threshold β a real valued vector, each component corrsponds to the threshold of a neuron

Process Function a rule influencing the trigger behaviour of a neuron

Nerve edges, connections between the neurons

Structure the number of neurons for input, output and the hidden layer (very difficult to determine)

Type restrictions: e.g. backcoupling, number of layers, number of edges per layer

Training

The most fundamental aspect of Neural Networks is the training. Usually this is an iterative process that determines the weights and thresholds of the neurons β. More advanced learning algorithms may also influence the function's structure. Yet the best learning algorithm can do nothing if the network structure and type were choosen poorly. So the most difficult part, that can almost solemly be learned by practice, is to decide for a structure and restrictions on it.

The objective function is as usual to minimize the least square error ε

$$\varepsilon = \sum_i \left\| \zeta\left(\vec{c}_i\right) - \mathrm{ANN}\left(\vec{c}_i\right) \right\|^2 \to \min$$

Extrema for this function can be found with standared methods like gradient descent (see 8.6) or coordinate descent (see 8.7). In coordinate descent alternately the threshold β and the weights w_{ij} remain fixed.

The result influcences the weights additive, multiplied with a step variable λ:

$$w_{ij}^{(l)} = \hat{w}_{ij}^{(k)} + \lambda \nabla f_{ij}$$

Properties

- the process-function of the neurons is usually the same for all nodes within a neural network
- there are many types without fixed layers and much backcoupling
- structure optimization can be connected with evolutionary algorithms

Pro/Contra

- there is no method to find the optimal structure + restrictions
-
- trainability
- generalization: towards different applications and often new unmodelled feature vectors
- efficiency compared to other classifiers
- fixed number of inputs → input of variable length not allowed → unsuited for speech recognition
+ parallelism
+ self-adaptability
+ focused on learning

Types

There are various different types for neural networks, designed for various applications. We will discuss some of the more general ones.

4.6.1 Multilayer Perceptron

Multilayer Perceptrons are very powerful. They are ordered in hierachical layers. Connections are only allowed between one layer and the following one. They allow non-linearities, like the Sigmoid Θ (see 9.13), as perceptron processing function and thus are able to model arbitrary class borders. Because of that, training algorithms do not operate linearly. Quadratic learning algorithms do exist. Also because of their power, the weights w_{ij} and the threshold β can not be optimizied simultaneously. This would lead into the Curse Of Dimensionality (see 8.18). They are rather to be optimizied seperately by e.g. using coordinate descent (see 8.7). Once the Multilyer Perceptron Neural Network was trained, it works in three steps:

Algorithm 17 Multilayer Perceptron

1. for all nodes: compute the weighted (w_{ij}) sum of the input f_i

$$y_j^{(l+1)} = \sum_{i=0}^{M_{l-1}} w_{ij}^{(l+1)} f_i^{(l)} - w_j^{(l+1)}$$

   for $0 \leq j \leq M_{l+1} - 1$

2. choose a non-linear function, e.g. the Sigmoid Θ

$$\Theta(y) = \frac{1}{1 - e^{-y}}$$

3. compute the output, apply the process-function with the non-linearity

$$f_j^{(l+1)} = \Theta\left(y_j^{(l+1)}\right)$$

Training

1. We choose supervised learning and label our sample vectors by the desired output δ of the process-function.
2. Determine the actual output δ_i.
3. Compute the error between them ε.

$$\varepsilon = \sum_{i=1}^{M^{(l)}} \left(\delta_i - f_i^{(l)}\right)^2$$

4. Change the weights using gradient descent, until the error ε has reached a minimum:

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \gamma \frac{\partial \varepsilon}{\partial w_{ij}^{(l)}} = w_{ij}^{(l)} + \Delta w_{ij}^{(l)}$$

   where γ determines the step width amd direction of the descent.

Properties

- 2 layers for all boolean functions
- 3 layers for approximating non-linear functions ($\Rightarrow$ arbitrary class borders!)
- supervised learning
- high degree of parallelism

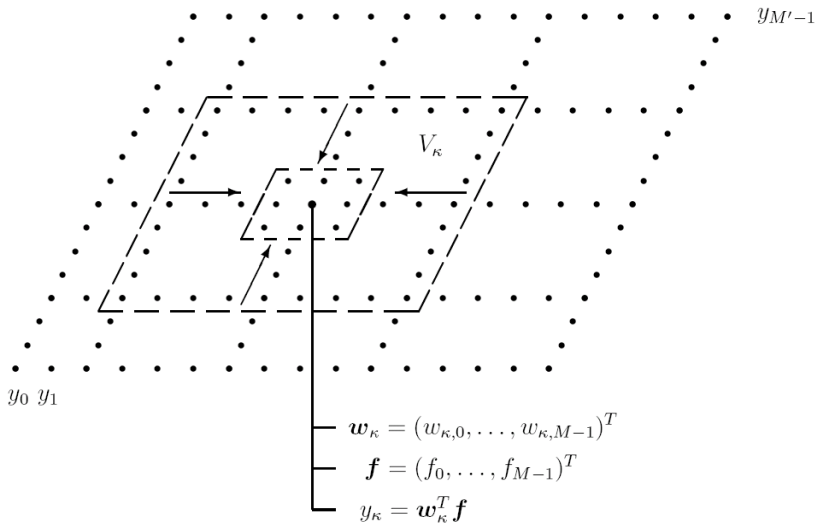
Pro/Contra

\+ arbitrary class borders

\+ high degree of parallelism

4.6.2 Feature Map

Abbildung 40: Neural Network with a Feature Map



$y_0, \ldots, y_{M'-1}$ output neurons

f observation, signal, features

w_j weight vectors

V_κ training area, that is compactified during training

A feature map pursues the same idea as the Sammon Transformation (3.1.18), namely to project features unto a 2D space to illustrate the classification process visually. So the features themselves are the input of the Neural Network and the output are the 2D pointvectors on a 2D map. The classification underlies the following objectives:

- features belonging to the same class should be projected to the same output
- features belonging to similar classes should be projected to neighbouring outputs

That means Feature Maps model similarity of observation grouping them in neighbouring regions according to their degree of similarity. A core aspect resulting from this, is that we have no static classes, but build them during the learning process by grouping features according to their similarity.

Algorithm 18 Feature Map

Find the weight vector w_κ closest (= most similar) to the input vector f using a Minimum Distance Classifier:

$$w_\kappa = \text{argmin}_{w_j} |f - w_j|^2$$

activiate the output neuron corresponding to this weight vector y_κ.

Training

Training works in two steps:

1. initialize the weights by random values
2. ↻ until a good error rate: $\varepsilon = \sum_i (f_i - w_{ij})^2$ has been reached
   - specify the neighbourhood of an output (i.e. when are outputs neighbours)
   - determine the output neuron y_κ, that would be activated:
     find the weight vector closest (= most similar) to the input vector in this neighbourhood using a Minimum Distance Classifier:

     $$w_\kappa = \text{argmin}_{w_j} |f - w_j|^2$$

     the output neuron corresponding to this weight vector is y_κ.
   - adjust the weights according to the new input vector using gradient descent (see 8.6):

     $$w_{ij} \leftarrow w_{ij} - \gamma \frac{\partial \varepsilon}{\partial w_{ij}} + \gamma (f_i - w_{ij})$$

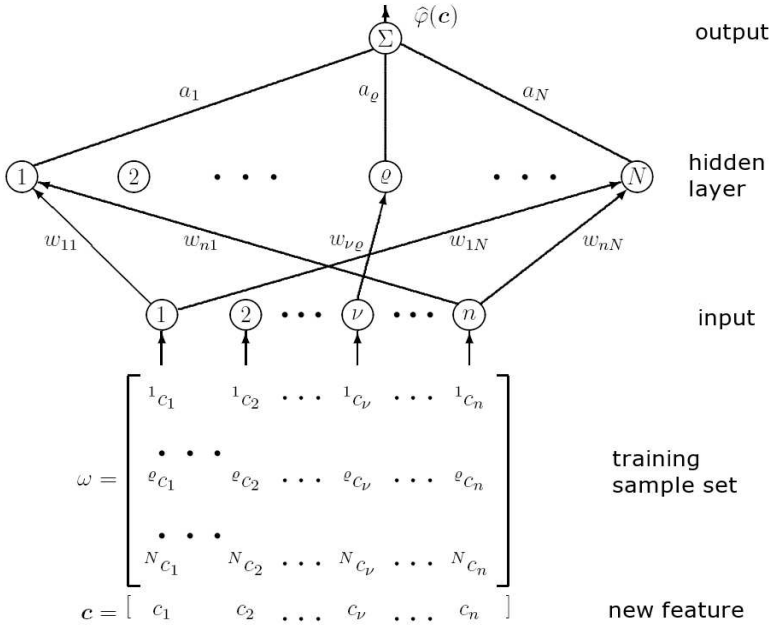
   - compactify the neighbourhood along proceeding training time

Properties

- unsupervised learning

4.6.3 Radial Basis Functions

Abbildung 41: Neuro Network with Radial Basis Functions



The idea of Radial Basis Functions is similar to that of Parzen Windowing (see 4.4.2). Neurons represent radial basis function (e.g. a gaussian bell). Incoming feature vectors are then reckoned up with the radial basis functions (the process functions) then the difference is taken and a Norm is applied. This means another difference to the Multilayer Perceptron is that while the Sigmoid has a global effect, these radial basis functions do have localy limited effects.

Properties

- only 1 hidden layer
- the simple structure allows simple direct computation of the weights
- a regulation term (e.g. a polygon) can be added to the process function of the perceptrons modelling in prior knowledge:

$$f(\beta, w) + \lambda(1 - w) + V_\beta f(\beta, w)$$

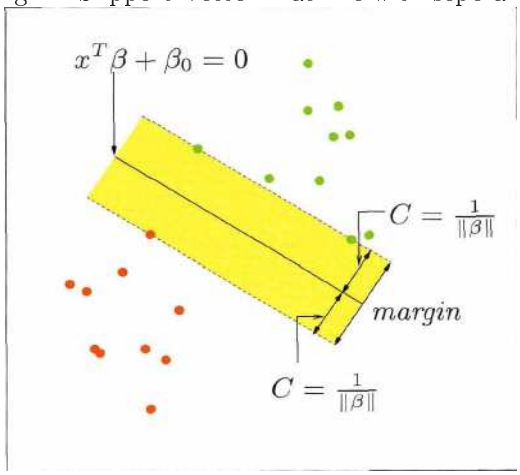
4.7 Support Vector Machines (SVM)

4.7.1 Basic SVM

Idea

Assume two linear seperable classes. Maximize the margin between the two classes and choose this margin as decision boundary. The margin will be fixed to the so called **support vectors**. Vectors of both classes which lie on the border of the margin and define the margin by that. This is very similar to classifiers called **Separating Hyperplanes**(see for Hyperplane: 9.14), which search for a hyperplane seperating the classes. Those seperating hyperplane classifiers are restricted to linear seperable classes, SMV however can be extended to overlapping classes, as we will see later.

Abbildung 42: Support Vector Machine with seperable classe



As you can see the margin is built by three support vectors. Two from the upper class and one from the lower. $x^T\beta + \beta_0$ is the optimal seperating hyperplane, that maximizes the margin. There are many more seperating hyperplanes, but they are less optimal and show a worse generalization ability (when new features are classified).

Theory

[A] The margin

1. Two feature vectors $\vec{c}_1, \vec{c}_2$ of the margin L have to fulfill the decision boundary equation:

$$\vec{\alpha}^T\vec{c}_1 + \alpha_0 = \vec{\alpha}^T\vec{c}_2 + \alpha_0$$
$$\Leftrightarrow \vec{\alpha}^T(\vec{c}_1 - \vec{c}_2) = 0$$

2. The margin is then defined as

$$L = \left\{ \vec{c} \mid \vec{\alpha}^T \vec{c} + \alpha_0 = 0 \right\}$$

3. Distance of a feature vector to the decision boundary: We have a parametric representation $f(\vec{x})$ of the margin and thus only need to plug in $\vec{c}$ and consider the normalisation: $\frac{1}{\|\alpha\|} f(\vec{c})$ to get the distance.

[B] Rosenblatt's Perceptron Rosenblatt's Perceptron learning algorithm tries to find a seperating hyperplane by minimizing the distance of misclassified points to the decission boundary.

Assume a two class problem.

1. decision rule:

$$\delta(\vec{c}) = \begin{cases} -1 & f(\vec{c}) < 0 \\ +1 & f(\vec{c}) \geq 0 \end{cases} = y$$

2. training vectors labelled by y_i:

$$y_i \cdot \left(\vec{\alpha}^T \vec{c}_i + \alpha_0 \right) = \begin{cases} > 0 & \text{correct classification} \\ = 0 & \text{rejection?} \\ < 0 & \text{wrong classification} \end{cases}$$

   where $\left(\vec{\alpha}^T \vec{c}_i + \alpha_0 \right)$ is the classifier which returns a scaled ($\vec{\alpha}^T$ is not normalized) distance to the separating plane and $y_i \in \{-1, +1\}$ a training label.

3. objective function

$$D = -\sum_{i \in M} y_i \cdot \left(\vec{\alpha}^T \vec{c}_i + \alpha_0 \right)$$

   where M is the set of misclassified vectors. The $-$ is because y_i will always be negative for misclassified vectors.

4. optimization problem

$$\left(\vec{\alpha}^T, \alpha_0 \right) = \text{argmin}_{\cdot \vec{\alpha}^T, \alpha_0} D$$

5. optimization procedure

$$\frac{\partial D}{\partial \vec{\alpha}} = D' \vec{\alpha} = -\sum_{i \in M} y_i \vec{c}_i \doteq 0$$

$$\frac{\partial D}{\partial \alpha_0} = D' \alpha_0 = -\sum_{i \in M} y_i \doteq 0$$

use stoachstic gradient descent (see also 8.6):

Algorithm 19 Stochastic Gradient Descent

(a) Randomly select a misclassified feature vector $\vec{c}_i$ (stochastical element)

(b) Update! $\begin{pmatrix} \vec{\alpha} \\ \alpha_0 \end{pmatrix} \leftarrow \begin{pmatrix} \vec{\alpha} \\ \alpha_0 \end{pmatrix} + \lambda \begin{pmatrix} y_i \cdot \vec{c}_i \\ y_i \end{pmatrix}$

6. **Pro/Contra**

\+ converges if the training set is linear seperable

\- result depends on the initialization of $\vec{\alpha}$ and α_0

Can be outmaneuvered by adding Lagrange constraints to the seperating hyperplane: **Optimal Separating Hyperplanes** seperates two classes and maximizes the distance to the closest point of either class.

\- if the training set is not linear seperable, the algortihm does not converge and cycles are hard to find

\- even if conversion is guaranteed, the number of "finite" steps can be pretty large

[C] Combination Of Margin And Perceptron Goal: Determine a linear function $f(\vec{c}) = \vec{\alpha}^T \vec{c} + \alpha_0$ with maximal margin M.

$$\max_{\vec{\alpha}, \alpha_0, \|\vec{\alpha}\|=1} M$$

subject to

$$y_i \left(\vec{\alpha}^T \vec{c} + \alpha_0\right) \geq M$$

draws

$$L(\vec{\alpha}, \alpha_0) = \frac{1}{2} \|\vec{\alpha}\|^2 + \sum_i^N \lambda_i \left(y_i \left(\vec{\alpha}^T \vec{c}_i + \alpha_0\right) - 1\right)$$

deriving for $\vec{\alpha}$ gives

$$\frac{\partial L}{\partial \vec{\alpha}} = \vec{\alpha} + \sum_i^N \lambda_i y_i \vec{c}_i$$

$$\vec{\alpha} = -\sum_i^N \lambda_i y_i \vec{c}_i$$

deriving for α_0 gives

$$\frac{\partial L}{\partial \alpha_0} = \sum_i^N \lambda_i y_i$$

$$y_i = -\sum_{i}^{N} \lambda_i$$

That tells us $\vec{\alpha}$ is a linear combination of feature vectors (support vectors) that build the margin.
Pluging in $\vec{\alpha}$ and y_i in $L(\vec{\alpha}, \alpha_0)$ draws

$$L(\vec{\alpha}, \alpha_0) = \frac{1}{2} \sum_i \sum_k \lambda_i \lambda_k y_i y_k \vec{c}_i^T \vec{c}_k - \sum_i \lambda_i$$

Remark: On this point extended Lagrange Multiplier constraints, called Wolfe Dual, are added for optimization.

Properties

- pretty good
- concentration on the interesting, difficult regions in the feature space
- requires no probability information of the feature vectors (like Bayes)
- SVM can be extended to multiclass problems, essentially by solving multiple two class problems
- $O\left(m^3 + mN + mpN\right)$ where m is the number of support vectors, N the number of features and p the number of predictors

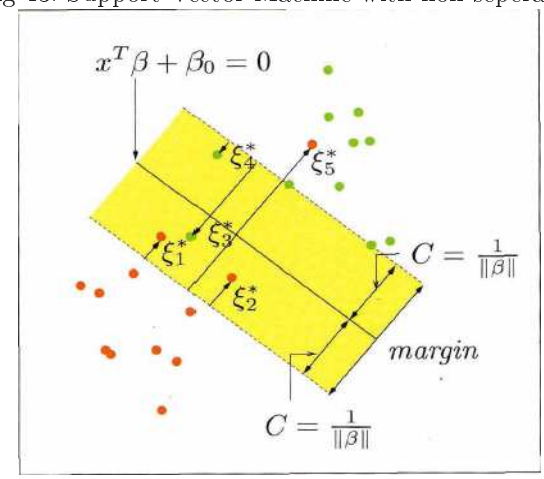
4.7.2 SVM for non-seperable classes

We first assumed that our two classes are linear seperable. If they're not, we have too adjust the SVM method and introduce **slack variables** ξ_i, which model vectors on the wrong side of the margin:

$$y_i \left(\vec{\alpha}^T \vec{c}_i + \alpha_0\right) \geq M\left(1 - \xi_i\right)$$

where $\xi_i \geq 0$ and $\sum_i \xi_i <$ const.

Abbildung 43: Support Vector Machine with non-seperable classes



By ξ_i we can model the proportional amount by which the predicition $f(\vec{c}) = \vec{\alpha}^T\vec{c} + \alpha_0$ is on the wrong side of the margin. All features on the wrong side of the margin are added to the set of support vectors and have an additional weight ξ_i. The original support vectors remain and gain a weight of $\xi_i = 0$.

The change in classification is:

1. features $\vec{c}_i$ on the correct side of the margin are weighted with $\xi_i = 0$
2. if $\xi_i \leq 1$ then $\vec{c}_i$ is element of the correct stripe
3. if $\xi_i > 1$ then $\vec{c}_i$ is misclassified

adapted optimization

by minimizing $\frac{1}{2}\|\vec{\alpha}\|^2$ we get rid of M. We furthermore have to consider the two conditions for ξ_i, $\xi_i \geq 0$ and $\sum_i \xi_i <$ const. The adapted Lagrange Multiplier term is:

$$L(\vec{\alpha}, \alpha_0) = \frac{1}{2}\|\vec{\alpha}\|^2 + \sum_i \lambda_i \left(y_i\left(\vec{\alpha}^T\vec{c} + \alpha_0\right) - 1\right) + \xi_i + \lambda\sum_j \xi_j + \sum_j \gamma_i\xi_i$$

deriving this term and considering the Wolfe Dual and the Karush-Kuhn-Tucker (see 8.11) conditions, we get our closed formula for the parameters.

Suport Vectors

Vectors on the wrong side of the margin are added to the support vectors. In addition we still have one ore more support vectors on the correct side of the margin, that lie very close to the margin's border.

Properties

- The maximization of the Wolfe-Dual under the Karush-Kuhn-Tucker conditions can be improved by adding a tuning variable γ. Points on the edge of the margin ($\xi_i = 0$) will be characterized by $0 < \alpha_i < \gamma$. The remaining points ($\xi_i > 0$) have $\alpha_i = \gamma$. This tuning variable γ can be obtained by Cross Validation (see 6.2).

4.7.3 Quadratic SVM

Estimating the SVM paramters $(\vec{\alpha}, \alpha_0)$ for a quadratic decission boundary with for example ML results in quadratic terms:

$$\sum_i \left\| \delta\left(\vec{c_i}\right) - f\left(\vec{c_i}\right) \right\|^2 \to \min$$

To avoid this we can apply a trick from Computergraphics and lift the dimension of the feature space by transforming the features. By this we are able to keep the function linear.

Example

$$\vec{c'} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \\ c_1 c_2 \\ \vdots \\ c_5^2 \\ \vdots \end{pmatrix} \Rightarrow f\left(\vec{c'}\right) = \vec{\alpha}^T \vec{c'} + \alpha_0$$

By that we get a linear decision boundary in a higher feature space.

Adapted Optimization

$$L\left(\vec{\alpha}, \alpha_0\right) = \sum_i \lambda_i - \frac{1}{2} \sum_i \sum_k \lambda_i \lambda_k y_i y_k \left(\vec{c_i}^T \vec{c_k}\right)$$

where $\left(\vec{c_i}^T \vec{c_k}\right)$ is replaced by a kernel function[3] $\left\langle h\left(\vec{c_i}\right), h\left(\vec{c_k}\right) \right\rangle$ which performs the transformation in the new feature space.

[3] for example: $k\left(\vec{c_i}, \vec{c_k}\right) = \left(1 + \vec{c_k}^T \vec{c_i}\right)^d$ for d-th degree polynomials

4.7.4 SVM with different basis functions

Untill now we have restricted ourselved to linear and quadratic basis function to define the margin of the SVM. Because in training only scalar products $\vec{c}^T\vec{c}$ have to be computed, in principle we could also take any non linear basis functions, but praxis has shown that it is much better to lift the space and look for linear functions in a feature space of higher degree, as we did in the previous section. You still have to pay attention not to overdo this, else you might run into overfitting.

For that we choose a kernel function $K(\vec{c}, \vec{c}')$ for the homomorphism into the space we'd like to build the margin in, and keep the linear function. Some possible kernels are:

d^{th} **degree polynomials:** $K(\vec{c}, \vec{c}') = (1 + \langle \vec{c}, \vec{c}' \rangle)^d$

radial basis: $K(\vec{c}, \vec{c}') = e^{\left(\frac{-\|\vec{c}-\vec{c}'\|^2}{z}\right)}$ where z is a constant

neural network $K(\vec{c}, \vec{c}') = \tanh(\kappa_1 \langle \vec{c}, \vec{c}' \rangle + \kappa_2)$

That changes our original SVM decission boundary

$$L(\vec{\alpha}, \alpha_0) = \frac{1}{2} \sum_i \sum_k \lambda_i \lambda_k y_i y_k K(\vec{c}, \vec{c_i}) - \sum_i \lambda_i$$

and gives us the linear function:

$$f(\vec{c}) = \sum_{i=1}^{N} \lambda_i y_i K(\vec{c}, \vec{c_i}) + \alpha_0$$

Pro/Contra

- Lifting can feature space easily leads to the Curse Of Dimensionality (see 8.18)
- Lifting the feature space too much makes it also hard for adaptive Kernel methods to produce an ideal kernel for a certain space

4.7.5 SVM for Regression

Regression means to deal with continous values instead of discrete class numbers. In SVM Regression we want to concentrate on a problematic region and pay less attention to the rest.

Theory

given linear model $f(\vec{x}) = \vec{\alpha}^T\vec{x} + \alpha_0$ and sample points $\{(\vec{y_i}, \vec{x_i}); i = 1 \dots N\}$ then $y_i = \alpha_i x_i + \alpha_0$ and we get the least square estimation: $\sum_i \|y_i - \alpha_i x_i + \alpha_0\|^2 \rightarrow$

min and thus $\vec{y} = A \begin{pmatrix} \alpha_0 \\ \alpha_i \end{pmatrix}$. Solving this with the Pseudoinverse, our parameters are:

$$\begin{pmatrix} \alpha_0 \\ \alpha_i \end{pmatrix} = \left(A^T A\right)^{-1} A^T \vec{y}$$

Now to focus on the problematic region, we change the residual function (Fehlerfunktion), i.e. the sum of squared differences.

$$V_\epsilon(t) = \begin{cases} 0 & |t| < \epsilon \\ |t| - \epsilon & \text{otherwise} \end{cases}$$

This error measure is called **ϵ-Intensive Error Measure**. This means we are ignoring errors of size less than ϵ. This roughly corresponds to the SVM methods, where points far away from the margin are ignored in the optimization. We can then deal with the two different regions with different adapted care: For the region falling through the ϵ-Intensive Error Measure we apply a simple function (i.e. linear) and for the difficult problematic region a quadratic one.

Example

We want to combine the abs function $|r|$ with a quadratic function r^2. Then our residual would simply be:

$$V_H(r) = \begin{cases} \frac{r^2}{2} & |r| \leq c \\ c|r| - \frac{c^2}{2} & |r| > c \end{cases}$$

Adapted Optimization

The general form of our residual function is:

$$V_\epsilon(t) = \begin{cases} 0 & |t| < \epsilon \\ |t| - \epsilon & \text{otherwise} \end{cases}$$

our optimization term is then:

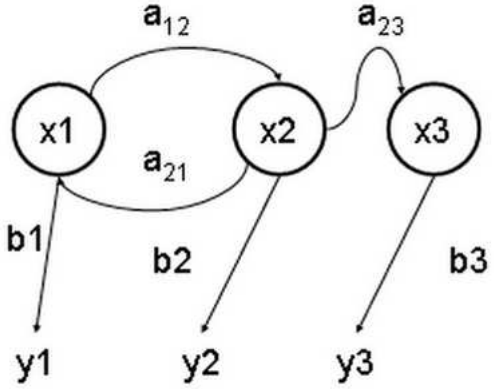
$$L(\vec{\alpha}, \alpha_0) = \sum_i V_\epsilon(y_i - f(x_i)) + \lambda \left(\|\overrightarrow{\alpha}\|^2 - 1 \right)$$

where the last λ term is for regularization and can be estimated by e.g. Cross Validation (see 6.2).

4.8 Hidden Markov Models (HMM)

4.8.1 Theory

Abbildung 44: HMM



A Hidden Markov Model is a method to classify sequences of random variables. A **sequence of random variables** is defined as

$$p(\vec{c}) = \prod_{i=1}^{n} p(\vec{c_i} \mid \vec{c_1}, \ldots, \vec{c}_{i-1})$$

where $p(\vec{c_i} \mid \vec{c_1}, \ldots, \vec{c}_{i-1})$ is the probability that $\vec{c}_i$ occurs, given $\vec{c_1}, \ldots, \vec{c}_{i-1}$ are the previous features occurances, also called the **history**. Taking this history unreflected into account, results in astronomically large search spaces (see Pro/Contra 4.8.1). Therefore we discuss some simplifications:

- many histories will turn out similar, therefore a many-to-one mapping of histories won't effectively worsen our model Φ:

$$p(\vec{c}) = \prod_{i=1}^{n} p(\vec{c_i} \mid \Phi(\vec{c_1}, \ldots, \vec{c}_{i-1}))$$

- a large vocabulary is not required, since the majority of the words contained will never occur in the language
- another idea is the usage of a grammar having a certain state Φ_i, then we could replace the history by this state, implicitely containing past happenings:

$$p(\vec{c}) = \prod_{i=1}^{n} p(\vec{c_i} \mid \Phi_{i-1})$$

- usage of n-gramms: reducing the history to the last $n-1$ observed features. Most commonly used are **Trigrams**:

$$p(\vec{c}) = \prod_{i=1}^{n} p(\vec{c_i} \mid \vec{c}_{i-2}\vec{c}_{i-1})$$

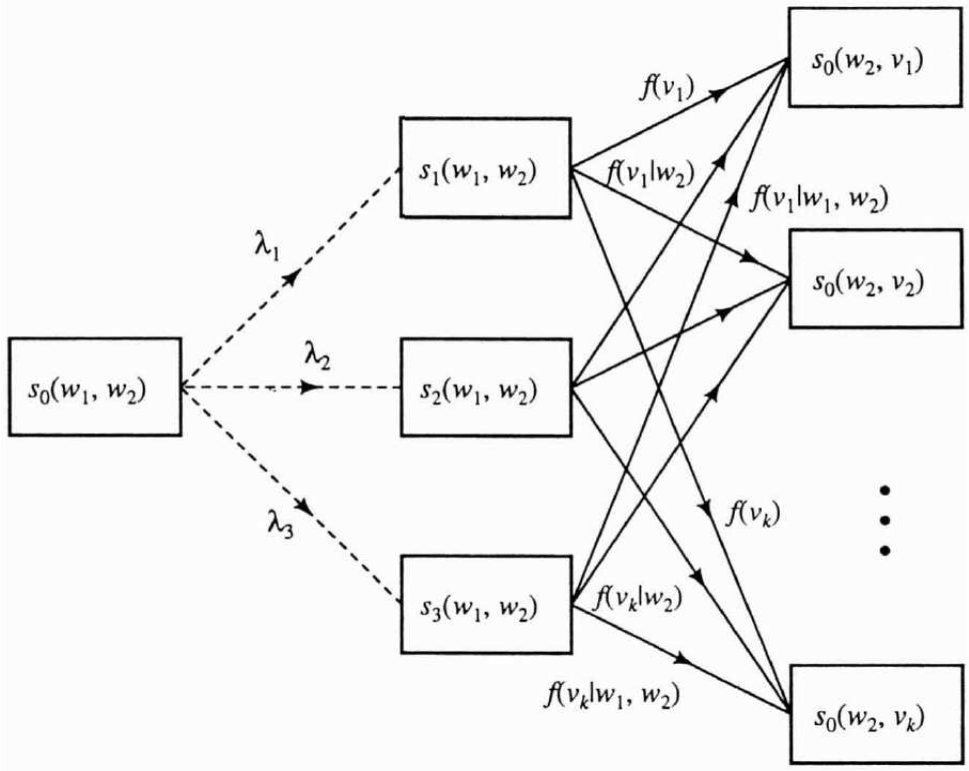
  That means sequences are considered to be equal if the end in the same two words.

- n-grams alone are very crude, because they model any possible combination of features. For example with recognizing words, many combinations result in a probability of 0. Therefore we add a smoothing to the n-grams, e.g. by interpolation:

$$p(\vec{c_i} \mid \vec{c}_{i-2}\vec{c}_{i-1}) = \lambda_3 f(\vec{c_i} \mid \vec{c}_{i-2}\vec{c}_{i-1}) + \lambda_2(\vec{c_i} \mid \vec{c}_{i-1}) + \lambda_3(\vec{c_i})$$

  where the non negative weights λ_i sum up to 1.
  This last idea is what HMMs do:

Abbildung 45: Linear Smoothing Of HMMs



In HMMs those interpolating smooth variables become initial probabilities ($\pi_i = \lambda_i$) and state transition probabilities ($a_{ij} = f(\nu_j \mid \vec{c})$). Here comes the aspect of similar histories into account: All transitions from state s_i to state s_j are considered to have the same probability, therefore a_{ij} is independent of $\vec{c}$. The states itself are called output probabilities ($b_i(\vec{c}) = s_i(\vec{c})$).

The HMM theory assumes that there is a hidden process behind such a sequence of random variables, that can be seen as an automata with states and state transitions. This hidden process is to be estimated by observing visible output symbols (Estimating the state sequence). In this automata or graph one distinguishes between three fundamental probabilities:

1. initial state probabilities: π_i

   initial probability for a certain state. Often determined by relative frequencies.

2. state transition probabilities: $a_{i,j}$

   probability to go from state s_i to state s_j.

3. output probabilities: $b_i(x)$ (alternative notation: $p(x, B_i)$, where B_i is a density parameter belonging to state s_i)

   probability to produce output symbol x, being in state i, according to the probability density function associated with state s_i.

Note that $\sum_{j=1}^{M} a_{i,j} = 1$ and $\sum_{i=1}^{M} \pi_i = 1$ where M is the number of states (see 10.3).
The probability for a certain state sequence is then:

$$p(\vec{c}_1, \ldots, \vec{c}_n) = \prod_{i}^{N} \sum_{l}^{M} \pi_l p\left(\vec{c}_i, \vec{B}_l\right)$$

the computational complexity is $O\left(N \cdot M^2\right)$.

N number of features

M number of states

A complete HMM stands then for a e.g. a word. The probability for a HMM is $p(\lambda \mid \vec{c}_1 \ldots \vec{c}_n)$, where

$p(\lambda \mid \vec{c}_1 \ldots \vec{c}_n)$ is $p(\lambda \mid \vec{c}_1 \ldots \vec{c}_n) = \frac{p(\lambda) p(\vec{c}_1 \ldots \vec{c}_n \mid \lambda)}{p(\vec{c}_1 \ldots \vec{c}_n)}$

$p(\lambda)$ is the prior probability for the word

$p(\vec{c}_1 \ldots \vec{c}_n \mid \lambda)$ is $p(\vec{c}_1 \ldots \vec{c}_n \mid \lambda) = \sum_{l_1 \ldots l_N}^{M} \pi_{l_1} p(\vec{c}_1, B_1) \cdot a_{l_1 l_2} \cdot p(\vec{c}_2, B_2) \cdot a_{l_2 l_3} \cdots$

Definition

$$\lambda = \left(\vec{\pi}, A, \begin{pmatrix} p(\vec{c}_1, B_1) \\ \vdots \\ p(\vec{c}_n, B_n) \end{pmatrix}\right)$$

where n is the number of features and A the matrix of the transition probabilities for any two states $a_{i,j}$. $p(\vec{c}_i, B_i)$ is the probability to observe $\vec{c}_i$, having parameters B_i (state and pdf).

Types

1. discrete

   features are discrete random variables.
   training: relative frequencies.

2. continous

   features are continous random variables
   training: state parameters

3. ergodic

   the graph associated with the HMM is complete (maximum number of transitions for each node)

4. left-right

   only forward edges are allowed: $a_{i,j} = 0 \Leftrightarrow i > j$

Example

given: $c \in \{0,1\}$, $s_1 : p(0) = \frac{1}{3}, p(1) = \frac{2}{3}$ and $s_2 : p(0) = \frac{1}{4}, p(1) = \frac{3}{4}$
wanted: $p(00)$
sum over all possible state sequences:
result: $p(00) = p(00; s_1, s_1) + p(00; s_1, s_2) + p(00; s_2, s_1) + p(00; s_2, s_2) =$
$= \pi_1 \cdot b_1(0) \cdot a_{11} b_1(0) + \pi_1 \cdot b_1(0) \cdot a_{12} b_2(0) + \pi_2 \cdot b_2(0) \cdot a_{21} b_1(0) + \pi_2 \cdot b_2(0) \cdot a_{22} b_2(0) =$
$= \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{4} + \frac{2}{3} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{3} + \frac{2}{3} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{4} = 0.08256172$

Problems

1. Training of the HMM (estimating $\pi_i, a_{i,j}.b_i(x)$)

   ⇒1.4.2. EM-Algorithm

2. computation of the marginal over all possible state sequences

   ⇒1.4.3. Forward-Backward-Algorithm

3. computation of the optimal state sequence

   ⇒1.4.4. Viterbi-Algorithm

Properties

- complexity $O(N \cdot M^2)$ with, $O(M^N)$ without rearangement of the sums and products (see 4.8.3)
- works with and only with sequences of random variables
- the initial probabilities π_i do not have to be computed seperately. Simply assume an initial state s_o having $\pi_o = 1$ and you get the state transition probabilites $a_{i,j}$ and π_i in one effort ($\pi_i = a_{0i}$).
- referring to the discussion of modelling histories: As you can see in the figure on 4.8.1, HMMs strongly resemble mixture densities (see 4.2.3). The weight of the distributions are π_i and a_{ij}, which indeed both sum up to 1. And the mixtures themselves are modelled in the states $b_i(x) = p(x, B_i)$ where B_i denotes the parameter vector of the distribution

- the initial probabilities π_i and the state transition probabilities a_{ij} can also be computed by Deleted Interpolation (see 4.9.2). This is especially usefull, when the available sample set to estimate both the relative frequencies as π_i and a_{ij} is sparse
- referring to the discussion of modelling histories: a very known way to realize the idea with the grammar is **Part Of Speech Tagging**, where grammatical functions like *verb, noun, adjective*, etc. are assigned to sequences of features. This effecively leads to a second layer HMM, where the second HMM answers the question: Given an observed sequence of features $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_n$, which was the most likely tag sequence $g_1, g_2, \ldots g_n$ underlying $\vec{c}_1, \vec{c}_2, \ldots, \vec{c}_n$? As first layer HMM this is resolved by the Viterbi-Algorithm and the parameters can be estimated by the Baum-Welch-Algorithm (resp. EM).

Pro/Contra

\+ statistical knowledge can be used (Bayes: $\text{argmax}_\lambda\, p\,(\lambda \mid \vec{c}_1 \ldots \vec{c}_n)$, Bayes means optimality)

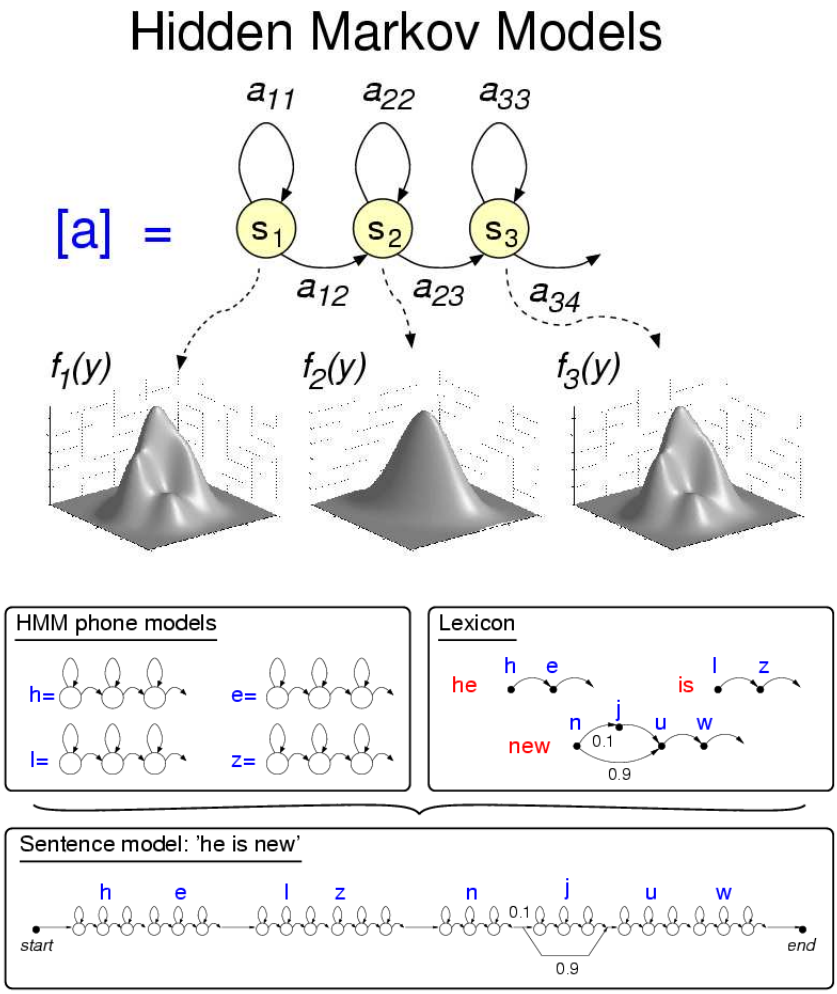
\+ different statistical models can be considered for the output probabilities (mixture densities)

\- a more realistic modelling of the history is impossible, because the combinations are just too many, resulting in an astronomically large search space. E.g. with a vocabulary of 5000 words and a history of degree 3, we'd have 125 billion combinations.

Application

Above all in speech recognition to classify phonems out of speech windows, words out of phonems and sentences out of words.

Abbildung 46: HMM in Speech Recognition



4.8.2 Training, Parameter Estimation (EM-Algorithm)

parameters to be trained: $\pi_i, a_{i,j}, b_i(x)$
Apply EM!

hidden: state sequence $l_1 \dots l_N$

observable: features (output) $\vec{c}_1 \dots \vec{c}_n$

$$Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) = \sum_{l_1 \dots l_N}^{M} \frac{p\left((\vec{c}_1 \dots \vec{c}_n) \mid l_1 \dots l_N, \hat{\lambda}^{(i)}\right)}{\sum_{k_1 \dots k_N}^{M} p\left((\vec{c}_1 \dots \vec{c}_n) \mid k_1 \dots k_N, \hat{\lambda}^{(i)}\right)} \log p\left((\vec{c}_1 \dots \vec{c}_n) \mid l_1 \dots l_N, \hat{\lambda}^{(i+1)}\right)$$

replacing the probabilites

$$Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) = \sum_{l_1 \dots l_N}^{M} \frac{\pi_{l_1}\left(\prod_{i=1}^{M} a_{l_i l_{i+1}}\right)\prod_{i=1}^{M} b_{l_i}^{(i)}\left(\vec{c}_i\right)}{\sum_{k_1 \dots k_N}^{M} \pi_{l_1}\left(\prod_{i=1}^{M} a_{k_i k_{i+1}}\right)\prod_{i=1}^{M} b_{k_i}^{(i)}\left(\vec{c}_i\right)} \log \pi_{l_1}\left(\prod_{i=1}^{M} a_{l_i l_{i+1}}\right)\prod_{i=1}^{M} b_{l_i}^{(i+1)}\left(\vec{c}_i\right)$$

1. $\pi_1 \dots \pi_M$

   Use Lagrange Multiplier Method with the additional condition: $\sum_i^M \pi_i = 1$ (see 10.3)

   see EM-Algorithm(8.3) for details.

2. $a_{i.j}$

   calculate the gradient regarding $a_{i,j}$ in respect to the stochastic criterion $\sum_j^M a_{ij} = 1$:

$$\nabla_{a_{i,j}} Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) =$$

$$\sum_{\substack{l_1 \dots l_N \\ \exists m: \; l_m = i \\ l_{m+1} = j}}^{M} \frac{\pi_{l_1}\left(\prod_{i=1}^{M} a_{l_i l_{i+1}}\right)\prod_{i=1}^{M} b_{l_i}^{(i)}\left(\vec{c}_i\right)}{\sum_{k_1 \dots k_N}^{M} \pi_{l_1}\left(\prod_{i=1}^{M} a_{k_i k_{i+1}}\right)\prod_{i=1}^{M} b_{k_i}^{(i)}\left(\vec{c}_i\right)} \cdot \frac{1}{\hat{a_{i,j}}^{(i+1)}} + \frac{\partial\lambda\left(1 - \sum_i \hat{a_{ij}}^{(i+1)}\right)}{\partial \hat{a_{ij}}^{(i+1)}}$$

3. $b_i(x)$

   The output probability is dependent on the HMM type, initial and state transition probability are independent of it.

4.8.3 Computation of the marginals (Forward-Backward-Algorithm)

We look for an efficient way to compute the marginal

$$p\left(\vec{c}_1 \dots \vec{c}_n \mid \lambda\right) = \sum_{l_1 \dots l_N}^{M} \pi_{l_1}\left(\prod_{i=1}^{M} a_{l_i l_{i+1}}\right)\prod_{i=1}^{M} b_{l_i}^{(i)}\left(\vec{c}_i\right)$$

without considering every possible state sequence alone ($O\left(M^N\right)$).

Key Idea $\sum_i \sum_j q_i r_j = \sum_i q_i \sum_j r_j$

$$\sum_{l_1 \dots l_N}^{M} \pi_{l_1} b_{l_1}\left(\vec{c}_1\right) \cdot a_{l_1 l_2} \cdot \dots \cdot \sum_{l_N}^{M} a_{l_{N-1} l_N} b_{l_N}\left(\vec{c}_N\right)$$

a_{l_N} is independent of $a_{l_1} \dots a_{l_{N-1}}$ so we take it out ($b_{l_N}\left(\vec{c}_N\right) \rightarrow a_{l_N, l_{N-1}}$)

$$\sum_{l_1 \dots l_N}^{M} \pi_{l_1} b_{l_1}\left(\vec{c}_1\right) \cdot a_{l_1 l_2} \cdot \dots \cdot \sum_{l_{N-1}}^{M} a_{l_{N-2} l_{N-1}} b_{l_{N-1}}\left(\vec{c}_{N-1}\right) a_{l_{N-2}, l_{N-1}}$$

$a_{l_{N-1}}$ is independent of $a_{l_1} \dots a_{l_{N-2}}$so get it out $(b_{l_{N-1}}(\vec{c}_{N-1}) \rightarrow a_{l_{N-1}l_{N-2}}) \dots$ and so on.
What we get out of this reordering is a reduction of the complexity to $O\left(M^2 \cdot N\right)$.

$$M\left(\sum_{l_1 \dots l_N}^{M}\right) \cdot M\left(\sum_{l_N}^{M}\right) \cdot N\left(l_1 \dots l_N\right)$$

Algorithm 20 Forward-Backward-Algorithm

---

backward step $p(\vec{c}_1 \dots \vec{c}_n \mid \lambda) = \sum_{l_1 \dots l_N}^{M} \pi_{l_1} b_{l_1}(\vec{c}_1) \cdot a_{l_1 l_2} \cdots \cdot \sum_{l_N}^{M} a_{l_{N-1} l_N} b_{l_N}(\vec{c}_N)$

forward step $p(\vec{c}_1 \dots \vec{c}_n \mid \lambda) = \sum_{l_1}^{M} \pi_1 b_{l_1}(\vec{c}_1) \cdot \sum_{l_2 \dots l_N}^{M} a_{l_1 l_2} b_{l_2}(\vec{c}_2) \dots$

4.8.4 Computation of the optimal state sequence (Viterbi-Algorithm)

We estimate the optimal state sequence by maximizing

$$\max_{l_1 \dots l_N} p(\vec{c}_1 \dots \vec{c}_N, s_{l_1} \dots s_{l_N} \mid \lambda)$$

simply replace the marginals of the Forward-Backward-Algorithm with this maxmimum operator.

$$\max_{l_1 \dots l_N} \pi_{l_1} b_{l_1}(\vec{c}_1)\, a_{l_1 l_2} \dots \max_{l_N} a_{l_{N-1} l_N} \cdot b_{l_N}(\vec{c}_N)$$

see also 8.21.

4.9 Acoustic Models For Speech Recognition

4.9.1 Theory

We observe a sequence of features (speech windows $w_1 \dots w_m$), yet for classifying them we are still missing a mapping $w_1 \dots w_m \rightarrow \vec{c}_1 \dots \vec{c}_n$. In Baysean notation that means:

$$\hat{w_1} \dots \hat{w_m} = \text{argmax}_{w_1 \dots w_m}\, p(w_1 \dots w_m \mid \vec{c}_1 \dots \vec{c}_n)$$

$$= \text{argmax}_{w_1 \dots w_m} \frac{p(w_1 \dots w_m)\, p(\vec{c}_1 \dots \vec{c}_n \mid w_1 \dots w_m)}{p(\vec{c}_1 \dots \vec{c}_n)}$$

Note Due to the Curse Of Dimensionality (see 8.18) $p(w_1 \dots w_m)$, can only be estimates for small m.

A solution to this problem, by that we can find the correct sequence of features, is to use n-grames and apply Deleted Interpolation:

1. n-grames
   $p(w_m \mid w_1 \dots w_{m-1}) = \xi_1 \frac{1}{m} + \sum_{i=1}^{m} \xi_i p(w_i) + \sum_{i=2}^{m} \xi_i p(w_i \mid w_{i-1}) + \sum_{i=3}^{m} \xi_i p(w_i \mid w_{i-1} w_{i-2}) + \dots$
2. Deleted Interpolation

4.9.2 Deleted Interpolation

Motivation Estimate probabilities from a sparse set of obervation (e.g. a Histogram with gaps).

A solution to this is either to apply Parzen Windowing (see 4.4.2) or to use Deleted Interpolation. For Deleted Interpolation we use adjusted n-grames:

$$p(w_n \mid w_1 \dots w_{n-1}) = \xi_1 \frac{1}{M} + \xi_2 p(w_1) + \sum_{i=2}^{n} \xi_i p(w_i \mid w_{n-i+1} \dots w_{n-1})$$

where $\sum_i \xi_i = 1$ and M denotes the number of all possible 'words'. Using these parameters ξ_i we can 'fill the gaps' and get reasonable results (see also 10.3).

Parameter Estimation

[A] Least Square Estimation

$$\hat{\xi_1} \dots \hat{\xi_n} = \text{argmin}_{\xi_1 \dots \xi_n} \left\| p(w_n \mid w_1 \dots w_{n-1}) - \xi_1 \frac{1}{M} + \xi_2 p(w_1) + \sum_{i=2}^{n} \xi_i p(w_i \mid w_{n-i+1} \dots w_{n-1}) \right\|^2$$

This does not work, since $w_1 \dots w_{n-1}$ are unknown, and Least Sqaue Estimation requires known parameters.

Since we see that this n-gramms ressemble very much mixture densities, we come to the idea to use the EM-Algotrithm.

[B] Expectation Maximization We observe that $p(w_n \mid w_1 \dots w_{n-1})$ is approximated by a mixture of densities, where ξ_i denote the mixture coefficients (weights). Therefore we can apply the EM-Algorithm.

For details see 8.3.

Example

$$p(w_4 \mid w_1 w_2 w_3) = \sum_{i=1}^{4} \xi_i p(w_i) + \sum_{i=1}^{4} \eta_i p(w_i \mid w_{i-1})$$

ξ_i and η_i are then estimated by the EM-formulas for mixtures (see 4.2.3).

4.9.3 Vocabulary

Every speech recognizer needs a vocabulary of finite size. If the vocabulary is too large, our search space increases dramatically, so every word in it should be carefully selected. On the other hand will unkown words quickly lead to erros in the classification. Therefore we have two somewhat contradictory postulates on vocabularies:

1. The vocabulary should be as small as possible
2. The recognizer should encounter as few as possible unkown words

One basic idea to model both claims, is to limit the vocabulary to the field of application. That means to model scientific terms, that are typical for the field of application, to model words fitting the education of the people that will use it, and so on.

Another idea is to use dynamic vocabularies consisting of the L last words used. These words are extracted from saved texts of previous applications.

A third idea is the use of several dictionaries. We would have one dictionary containing a couple of core words, found in any thinkable field of application and several dictionaries for certain domains. Now the recognizer first estimates the domain incoming words belong to and than choose the corresponding dictionary as extension to the core words.

Example

We want to use a vocabulary of size $L = 15.000$. To produce the 15.000 most last used words, we however need a textsize from approximately 640.000 words. The good thing, with this vocabulary about 99% coverage is reached.

5 Stochastic Modeling Of Objects

Theory

Model classification divides into six parts:

- object modelling (e.g. by a pdf)
- model learning (e.g. parameter estimation of this pdf)
- statistical inference (evaluation of the pdf)
- matching (usually between image and model features)
- pose estimation (maximize the probability)
- classficiation (Bayes decission rule)

Although there are many purely geometry based methods, we decide for hybrid approach using both statistics and geometry. This solves one major problem of geometry based approaches: They are very vulnerable to noise and variances in illumination etc. With a probabilistic approach we can model these phenomena and take them into account. Furthermore we are provided with many well studied methods for parameter estimation and can make use of the Baysean decision rule, prooven to be optimal with respect to misclassification rates.

In general model based approaches differ in the following aspects:

- model representation
- the measure / method for comparison
- judgement /estimation of object classes and pose parameters

In this approach we choose probability density functions as model representation and make use of the ML- and EM-Algorithm for parameter estimation (translation, rotation, projection).

Problems

There is usually more than one object on a picture, the object looks different from different angles, it is just a projection from the real 3D-object and it is not always on the same place in the picture. Last but not least pictures include a high level of noise and differences in illumination, that easily trick edge-detectors.

A lot of problems. We start trying to include projection, translation and rotation of the object into the probability density function. Let $\vec{c}$ be a feature of the object and $\vec{c}'$ the same feature after translation and rotation.

Rotation And Translation

We start with simply including parameters for rotation (rotation matrix R) and for translation (translation vector $\vec{t}$) into the probability densisty function:

$$p\left(\vec{c}';\mathcal{B},R,\vec{t}\right)$$

where $\mathcal{B}$ denotes the parameter vector for the density (e.g. $\vec{\mu}$ and Σ for $\mathcal{B}=\mathcal{N}$).

How do we get the extended density? Consider the density transform: $Y = g(X)$ with $\exists g^{-1}$. The pdf of Y given the pdf of X is:

$$p_y(Y) = \left|\mathcal{J}_{g^{-1}}(Y)\right| p_X\left(g^{-1}(Y)\right)$$

We further consider the special case of affine mappings[4], that means $\vec{c}' = R\cdot\vec{c}+\vec{t}$.

$$p\left(R\cdot\vec{c}+\vec{t}\right) = \frac{1}{\det(R)} p_x\left(R^{-1}\left(\vec{c}-\vec{t}\right)\right) = \frac{1}{\det(R)} p_x\left(R^T\left(\vec{c}-\vec{t}\right)\right)$$

Specializing even more, we now assume $\mathcal{B}=\mathcal{N}$, that gives us

$$\mathcal{N}\left(R\cdot\vec{c}+\vec{t};\vec{\mu};\Sigma\right)$$

since $\mathcal{J}=1$ (because of the assumed affine mappings), we still got a normal distribution after the density transform:

$$\Sigma' = R\Sigma R^T$$

$$\vec{\mu}' = R\vec{\mu}+\vec{t}$$

What remains is estimating the parameters R and $\vec{t}$.

Parameter Estimation

The simplest way to get estimates for R and $\vec{t}$ is to use Maximum Likelihood Estimation:

$$\hat{R},\hat{\vec{t}} = \text{argmax}_{R,\vec{t}}\, p\left(\vec{c}';\mathcal{B},R,\vec{t}\right)$$

for details see 8.1. Note that standard optimization techniques like Newton iteration cannot be applied, because we need to find optimal maxima. That means we have to fall back to global optimization techniques. Usually pose parameters are estimated by geometrical relations. But we excluded such relations by assuming statistical independency and thus have to use ML. See below (5) for a reduction of the search space and such global techniques.

[4] affine mappings are translation, isotropic scaling, reflection and shearing

Projection

We still have not factored in the projection of the object, so let's do it now. For convenience let us assume an orthographic projection[5].

$$\overrightarrow{c}' = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \curvearrowright \begin{pmatrix} c_1' \\ c_2' \end{pmatrix}$$

we use the old trick: "To get rid of a random variable, just marginalize it away":

$$p\left(\begin{pmatrix} c_1 \\ c_2 \end{pmatrix}; \mathcal{B}, R, \vec{t}\right) = \int p\left(\vec{c}'; \mathcal{B}, R, \vec{t}\right) d_{c_3}$$

Combining projection with rotation and translation, we get the density transform:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \curvearrowright R \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} + \overrightarrow{t} \curvearrowright_{\text{projection}} R' \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$$

with $R = \begin{pmatrix} & R' & \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$. As far as good, but are we still in a normal distribution? Sure we are. If $\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}$ was normally distributed, $R' \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ is also normally distributed with $\Sigma_\rho = R'\Sigma R'^T$ and $\mu_\rho = R \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$.

Now that we have incorporated rotation, translation and projection into the pdf. Let's take a look at our features. We got

$$p\left(\vec{c}_i; \mathcal{B}_{k,j(i)}\right)$$

where $\mathcal{B}$ is the parameter vector of the pdf (see above), k refers to the object and j is the index of the 3D feature associated to the 2D feature $\vec{c}_i$. Now let us assume that all features are mutually independent of each other (what they ain't, yet it makes things a lot easier, i.e. linear), then we get

$$p\left(\vec{c}_1 \ldots \vec{c}_n\right) = \prod_{i=1}^{n} p\left(\vec{c}_i; \mathcal{B}_{k,j(i)}\right)$$

respectively $p\left(\vec{c}_1 \ldots \vec{c}_n; R, \vec{t}, \rho\right)$ incorporating rotation, translation and projection. Since the 3D feature vector corresponding to index j, is not observable, we

[5] an affine, parallel projection of a 3D object unto a plane (without perspective transformations).

better marignalize it away[6]:

$$p\left(\vec{c}_1 \ldots \vec{c}_n\right) = \sum_j \prod_j p\left(j\left(i\right)\right) p\left(\vec{c}_i; \mathcal{B}_{k,j(i)}\right)$$

We already indicated, that parameter vector $\mathcal{B}_{k,j(i)}$ can be estimated using ML. For $p\left(j\left(i\right)\right)$ however, we need to apply the EM-Algorithm, since $j\left(i\right)$ is not observable (hidden). See 8.3.

Neighbourhood Relationships

We extend our model by respecting neighbourhood relationships, i.e. that two points that build an edge of the object, build a line in the projection. We can get such relationships by an edge detector (see 2.1.9). Extending our pdf with this knowledge, we get

$$p\left(\vec{c}_1 \ldots \vec{c}_n; \chi\left(\vec{c}_1, \vec{c}_2\right), \chi\left(\vec{c}_2, \vec{c}_3\right) \ldots \chi\left(\vec{c}_{n-1}, \vec{c}_n\right)\right) =$$

where

$$\chi\left(\vec{c}_k, \vec{c}_l\right) = \begin{cases} 0 & \text{if } \vec{c}_k \text{ and } \vec{c}_l \text{ are not connected by a line} \\ 1 & \text{if a connection between them exists} \end{cases}$$

Combining this extension with our knowledge about projection, we get

$$= \sum_j \left(\prod_j p\left(j\left(i\right)\right) \right) \cdot \left(\prod_i \prod_k p\left(\chi\left(\vec{c}_{j(i)}, \vec{c}_{j(k)}\right)\right) j\left(i\right) j\left(k\right) \right)$$

This relationship χ can of course be extended to arbitrary neighbourhood relationsships.

Object Localization

Yet line detection alone won't suffice, since there will be many lines detected, that are created by noise. The main problem with that is, that the low probability of background noise, makes the above product collapse. The solution of this problem is to simply include noise into the statistical modelling and training. We call them **background features**. What we get out of that is a two stage assignment :

In the first stage features are classified either as background or object features. In the second stage background features are modelled by an uniform distribution with density parameter vector $\mathcal{B}_B$ and object features by a statistical model (e.g. gaussian) with parameter vector $\mathcal{B}_k$ corresponding to the object k. Object features are then matched to features of the corresponding model.

[6] The alternative would be to solve a discrete search problem of finding the best match. Marginalization simply considers all possible assignments.

The result is a mixture density between uniform distribution and the chosen statistical model.

$$\prod_{i=1}^{N} p\left(\text{assign } \vec{c}_i \text{ to background}\right) \cdot p\left(\vec{c}_i, \mathcal{B}_B\right) + p\left(\text{assign } \vec{c}_i \text{ to object}\right) \cdot p\left(\vec{c}_i; \mathcal{B}_{k,j(i)}; R, \vec{t}, \rho\right)$$

We can model the assignment to background and object by simple prior probabilities: the prior probability for the background is $p(B)$, and the prior for the object is than simply $p(O) = (1 - p(B))$:

$$\prod_{i=1}^{N} p(B) \cdot p\left(\vec{c}_i, \mathcal{B}_B\right) + (1 - p(B)) \cdot p\left(\vec{c}_i; \mathcal{B}_{k,j(i)}; R, \vec{t}, \rho\right)$$

Alas the modelling of background noise / features makes real time classification really slow.

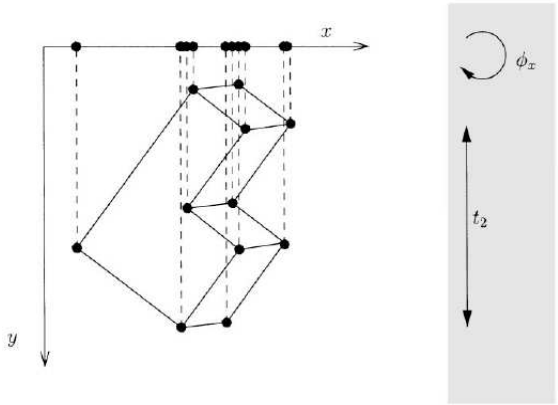
Dimension Of The Search Space

The dimension of the search space is 6: 3 (due to translation) + 3 (due torotation). Yet we can apply several tricks to deal with it:

1. If we assume orthographic projection, t_3 (moving forward and backward) does not influence the projection and we get dimension 5.
2. We reduce the dimension further by orthographically projecting the object unto the x-axis (gaining invariance to moving up and down, t_2). By that we also make rotations around the x-axis (ϕ_x) effecless. This this trick gives us a dimension of 3. Applying this restrictions to our pdf, we simply marginalize y away:

$$p\left(\text{1D feature}\right) = \int p\left(\{\vec{c}_1 \dots \vec{c}_n\}; \mathcal{B}_d, R, \vec{t}\right) dy$$

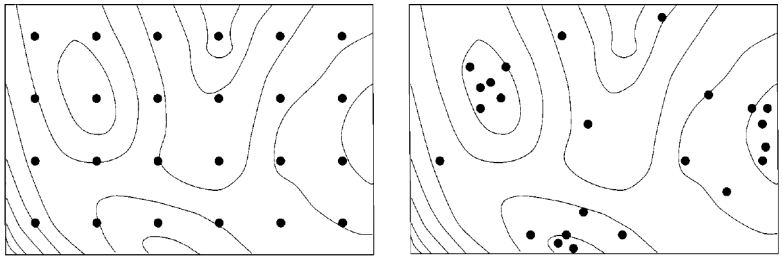
Abbildung 47: Orthographic projection unto the x-axis



3. We can repeat 2. for the y-axis, classify with both methods and combine the results.

4. **Adaptive Random Search**:

Abbildung 48: Adaptive Random Search



The left image corresponds to a grid search with contours, the right illustrates the same search with adapative random search. Grid points are choosen in equidistant positions, while the adapative random search illustraions shows how points are choosen according to the evaluation of previous randomly picked points.

Since a grid search of the search space is uneffictive for such large dimensions, we can apply adaptive random search. By that we partition the search space in statistical areas and pick points accordingly:

Algorithm 21 Adaptive Random Search

(a) set the probability of all points according to uniform distribution (they are all equally probable).

(b) randomly pick a number of points and evaluate them.

(c) store the k best evaluated points into a list (if the list already contains better evaluated points, dismiss the point)

(d) place a gaussian bell over each point in the list (peak = evaluation value), to effect the probability of all neighbouring points respectively

(e) use the pdf generated in 4. to generate new sample points

(f) ↻3.

(g) a termination criterion is for example the difference in evaluation between the first and the last point in the list

By this algorithm we ensure that points are picked according to how close they are to the correct one.

These steps provide us with the follwing procedure to estimate e.g. pose parameters R and $\vec{t}$:

1. compute the set of maxima M of the model density corresponding to 1D features projected to the x-axis (adaptive random search)
2. starting with the elements of M compute the maxima for projection to the y-axis (adaptive random search)
3. use the resulting set and start with local optimization

Properties

- The cross-ratio (see 9.2) e.g. is a feature that is invariant (11) to projection.
- We can not apply HMM for object modelling, since we have a set of features and no sequence of features. The difference is, that sets are not ordered.
- Use an illustration of a 3D Cube to derive the methods of statistical object modelling (projection, rotation, translation, features (corners), relations (indicator variables), noise modelling, reduction of the search space).
- $j\left(i\right)$ should have constraints, that two different model features can not be assigned to the same image features. This could happen, due to noise and segmentation, but is not desirerable. The other way round is more diffciult: For the same reasons not every model feature will find a corresponding image feature.
- $p\left(\vec{c}_1 \ldots \vec{c}_n\right) = \prod_{i=1}^{n} p\left(\vec{c}_i; \mathcal{B}_{k,j(i)}\right)$ can be evaluated in $O\left(n \cdot K\right)$, where n is the number of features and K the number of objects
- Choose the most discriminating views on your objects for training. To determine the best views, you can apply the entropy as a measure.

Pro/Contra

\+ statistical independency and marginalization proove to be a valid weapon against the Curse Of Dimensionality

\- background noise can lead to problems (collapsing product)

\- real time efficiency is pretty bad, especially when modelling background features

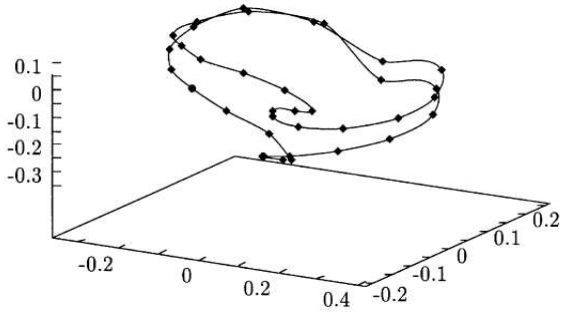
\- another problem is that due to noise and segmentation errors not every model feature has a corresponding image feature

\- the decomposition of the search space performed in the EM-Algorithm prevents effective use of line features, because they are effectively decomposed to point features again

Alternatives

- **Geometrical Modeling Of Objects**: the old school method. Lacks quality espeically with objects showing a complex geometry
- **Appearance Based Modeling Of Objects**: focuses on the appearance of objects on a signal, rather than real life geometrical properties. Thus this method makes heavily use of all the methods introduced in pattern recognition (preprocessing, automatic feature extraction, probaility based classification, etc.). Objects are then e.g. modelled as feature **manifolds** (a kind of interpolation between features).

Abbildung 49: Apearance Based Object Modeling: Manifold



Appearance based object modeling in fact hates geometrical items, as praxis has shown. The worst accuracy in a test set was gained by using a edge picture showing lines etc. The best results are gained with simple graylevel images. That means never combine this technique with any form of segmentation.

6 Model Assessment And Model Selection

By now we have a broad variety of classifiers and models with very different approaches, yet what is still missing is a way to determine, which classifier works better for an application.

Quality Criterions

- the chosen model: LDA, PCA, SVM, HMM, ANN
  - how good is the choosen parameterization for the model
  - how good are the estimates for those parameters
- generalization of the chosen model (does it still work on a test set entirely different and disjoint to the training set?)
- learning/training strategy: supervised, unsupervised, ML, MAP, EM
- measured data: speed, number of mistakes (which classifier works best)

Furthermore we distinguish two phases discussing the performance of models:

model selection estimate the performance of different models in order to choose the most suitable one.

model assessment having chosen a model, estimate the prediction/generalization error

Error

Talking about the error we distinguish the **Training Error** and the **True Error**.

The first is calculated using the very same training set, we used to fit the classifer:

$$\mathrm{ERR}_{\mathrm{training}} = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, \hat{f}\left(\vec{c}_i\right)\right)$$

where $\vec{c}_i$ are features from the training set and y_i the corresponding correct outcomes. As you can imagine this error is usually not a good bet, since it is highly dependent on the training set and even more on the level of fitting. E.g. an overfitted model will have a training error close to zero, while the true error will diverge to infinty.

The true error is the error we get theoretically using the space of all possible features C and the space of all correct outcomes Y:

$$\mathrm{ERR}_{\mathrm{true}} = E\left[L\left(Y, \hat{f}\left(C\right)\right)\right]$$

Remember the golden rule of pattern recognition: Never use the same features for training and testing. The solution we will find later is straight forward: We

will seperarte the training set into two parts and use one for training resp. fitting the model and the other for calculating the error. This errors is much less optimistic than the training error introduced above. The later error is also called **Extra Sample Error**.

6.1 Bias-Variance Trade-Off

Bias indicates the deviation from correct values (see also 11)[7]

$$\text{Bias}\,(\theta) = \left| \hat{\theta} - \frac{1}{m} \sum_{i}^{m} \hat{\theta}_i \right|$$

where $\hat{\theta}$ is the estimate obtained by using the complete sample set, while $\hat{\theta}_i$ correspond to a subset (e.g. by the use of Bootstrap (6.3) or Cross Validation (6.2).

Variance indicates how far the values differ from each other.

$$\text{Variance}\,(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{\theta}_i - \frac{1}{m} \sum_{k=1}^{m} \hat{\theta}_k \right)^2$$

Example

We have a feature set $\{3, 5, 2, 1, 7\}$ with a mean of $\mu = 3.6$. Using Bootstrap (see 6.3) we make 3 experiments to obtain 3 μ_i values.

experiment	mean
$7, 5, 2, 3, 1$	3.2
$5, 1, 1, 3, 7$	3.4
$2, 2, 7, 1, 3$	3.0

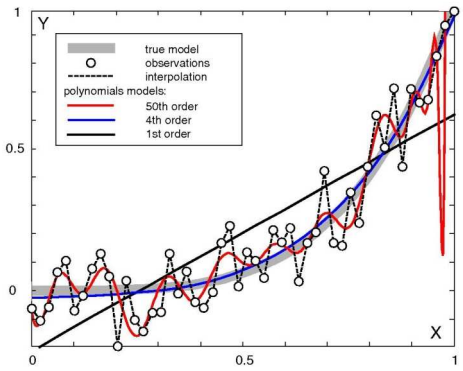
That gives us the mean of the means $\bar{\mu} = 3.2$. With that we can compute Bias and Variance:

$$\text{Bias}\,(\mu) = |3.6 - 3.2| = 0.4$$

$$\text{Variance}\,(\mu) = \frac{1}{3} \sum_{i=1}^{3} (\mu_i - \bar{\mu})^2 = 0.02\bar{3}$$

[7] computation of Bias and Variance require methods like Bootstrap and Cross Validation, that are introduced later on in this chapter.

Abbildung 50: Bias Variance Trade Off



As you clearly can see those two indicators act contrary. An overly complex model (red) perfectly fit to a set of observations (overfitted) will show a low bias and a high variance, while a very simple model (e.g. a line through the middle of the feature set) (black) will show a high bias and a low variance. Weighting both extremes we might find the better (blue) model in between.

So we have to find the golden path through the middle. We therefore define a trade-off between bias and variance:

$$E\left(\left(\theta^* - \hat{\theta}\right)\left(\theta^* - \hat{\theta}\right)^T\right) = E\left(\left(\theta^* - \bar{\theta}\right)\left(\theta^* - \bar{\theta}\right)^T\right) + \left(\theta^* - \hat{\theta}\right)\left(\theta^* - \hat{\theta}\right)^T \left(\bar{\theta} - \theta^*\right)\left(\bar{\theta} - \theta^*\right)^T$$

Tabelle 4: Bias-Variance Trade-Off Vraiable Look Up Table

symbol	meaning
θ	parameter vector
θ^*	correct value
$\bar{\theta}$	mean value
$\hat{\theta}$	estimated value
$\left(\bar{\theta} - \theta^*\right)\left(\bar{\theta} - \theta^*\right)^T$	bias
$E\left(\left(\theta^* - \hat{\theta}\right)\left(\theta^* - \hat{\theta}\right)^T\right)$	variance

Loss Function(ERR) a measurement for the number of mistakes a classifier produces

1. ERR for regression

$$\text{ERR} = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(\vec{c}_i\right) - \vec{\theta}\right)^2$$

2. ERR for classification

$$\mathrm{ERR} = \frac{1}{N} \sum_{i=1}^{N} L\left(\Omega_{k,i}, \delta\left(\vec{c}_i\right)\right)$$

where L is a standard loss function, e.g. the 0/1-loss function.

6.2 Cross Validation

Cross Validation is a simple and widely used method to estimate the prediction error. The core idea of Cross Validation is a better use of the training data. Thinking of situations where a big training set is hard to obtain, it can be essential to use the existing one as good as possible.

K-Fold Cross Validation means the partition of the whole set in a training and test set. For example 4-Fold Cross Validation partitions the complete set in 4 parts, of which it uses 3 for training and 1 for testing. The whole procedure will go over 4 turns, where the test part changes to every one of those partitions.

Tabelle 5: 4-Fold Cross Validation

TRAIN	TRAIN	TRAIN	TEST
TRAIN	TRAIN	TEST	TRAIN
TRAIN	TEST	TRAIN	TRAIN
TEST	TRAIN	TRAIN	TRAIN

A good model should show similar results for any given partition.

Algorithm 22 K-Fold Cross Validation

1. partition a model into k parts
2. train model with the set $S = \{1, \ldots, (k-1)\}$
3. test the model with part $p = k$
4. ↻ for $p = (k-1) \ldots 1$ and $S = \{1, \ldots, k\} \setminus \{p\}$

Leave-One-Out Cross Validation In Leave-One-Out Cross Validation the cardinality of the test set is set to 1.

Algorithm 23 Leave-One-Out Cross Validation

given a set of features $c_1 \dots c_n$, a decision rule $\delta(\vec{c}_i) = \Omega_{\kappa(i)}$

1. randomly select a feature $c_k \in \{c_1 \dots c_n\}$
2. train classifier with training set $\{c_1 \dots c_n\} \setminus \{c_k\}$
3. classify $\vec{c}_k$ and compute the loss $L_l(\vec{c}_k, \delta(\vec{c}_k))$
4. $l = l + 1$
5. ↻ repeat m times
6. compute average loss function $L = \frac{1}{m} \sum_{l=1}^{m} L_l$

Consider furthermore the situation of having different training sets. We will get different estimates of our parameter vector θ out of every one of them. We conclude that our parameter θ is a random variable that underlies a certain pdf $p(\theta)$. Looking closer on this distribution (mean and covariance), we can judge the performance of the model.

Example

We got the complete feature set: $S = \{3, 4, 2, 1, 5, 2, 5\}$, the index i denotes the feature that has been left out, respectively the corresponding subset. μ denotes the mean vector of the features as usually and $\bar{\mu}$ the mean of the means (since we get mean values for every subset of S. σ denotes the variance of the features, while σ_μ denotes the variance of the mean vectors.

1. We start with estimating the mean vectors μ_i for every subset.

$$\mu_i = \frac{1}{6} \sum_{\substack{j=1 \\ j \neq i}}^{7} c_j$$

$$\mu_1 = \frac{19}{6}, \mu_2 = \frac{18}{6}, \mu_3 = \frac{20}{6}, \mu_4 = \frac{21}{6}, \mu_5 = \frac{17}{6}, \mu_6 = \frac{20}{6}, \mu_7 = \frac{17}{6}$$

2. The variance of the features is

$$\sigma_i^2 = \frac{1}{6} \sum_{\substack{j=1 \\ j \neq i}}^{7} (c_j - \mu_i)^2$$

3. The mean of the means

$$\bar{\mu} = \frac{1}{7}\sum_{i=1}^{7}\mu_i = \frac{22}{7}$$

4. The variance of the means

$$\sigma_{\mu}^2 = \frac{1}{7}\sum_{i=1}^{7}(\mu_i - \bar{\mu})^2$$

If we compute and compare those values, using two representations of a line a polar one ($r = x \cdot \cos(\phi)\, y \cdot \sin(\phi)$) and then a parametric one ($y = m \cdot x + t$), we will notice huge variances in the second one, while the first model is more stable. We can conclude, that the first model is more suitbale for classification.

Properties

- Cross Validation is in principle ubiased, but can show a high variance, since the training sets are so similar to each other
- Using 5− or 10−Cross Validation, the error will have a low variance, but if the training set at a given size shows a big slope, Cross Validation will overestimate the predicition error and we get a bias.
- The number of partitions K, is always hard to determine. Practical values for K are 5 or 10.

Pro/Contra

- Cross Validation changes the prior probabilities of the classes (see next Chapter: 6.3)

6.3 Bootstrap (Efron 1979)

Cross Validation has one major drawback: by choosing samples without replacement ('Ziehen ohne zurücklegen'), we modify the prior probabilities for each class.

Example

We have given the training set of labelled features: $\{(c_1, \Omega_1), (c_2, \Omega_1)\,(c_3, \Omega_1)\,(c_4, \Omega_2)\,(c_5, \Omega_2)\}$
The priors of the two classes are: $p(\Omega_1) = \frac{3}{5}$ and $p(\Omega_2) = \frac{2}{5}$
Now we draw without replacement a random feature, we draw: (c_1, Ω_1).
The priors of the classes are now: $p(\Omega_1) = \frac{1}{2}$ and $p(\Omega_2) = \frac{1}{2}$.
They changed!

Bootstrap solves this drawback by simply putting drawn features back into the set ('Ziehen mit zurücklegen'). That of course means, that we can draw the same feature several times (which need not be a bad thing).

Algorithm 24 Bootstrap

1. randomly select M samples with replacement out of a set of N samples and use them for training.
2. The samples that are not used for training are used for testing.
3. ↻ repeat m times.

Example

We have given samples: c_1, c_2, c_3, c_4, c_5 and the associated class numbers $\Omega_{k(1)}, \Omega_{k(2)}, \Omega_{k(3)}, \Omega_{k(4)}, \Omega_{k(5)}$ where $k \to \{0, 1\}$. That means we only got two classes. We test with $m = 4$.

experiment	training set	test set	loss function
No.1	c_1, c_1, c_4, c_4	c_2, c_3, c_5	L_1
No.2	c_5, c_5, c_5, c_5	c_1, c_2, c_3, c_4	L_2
No.3	c_1, c_2, c_3, c_4	c_5	L_3

and the average loss function $L = \frac{1}{3} \sum_{k=1}^{3} L_k$

Pro/Contra

+ Bootstrap increases the variance, observed in each subset. This is good, since it models the real world.
+ Prior probabilities remain constant.
+ Boostrap leads to more reliable estimates for mean and covariance.
+ Boostrap is well studied.

6.4 The Effective Number Of Parameters

Another idea is too evalute the number of parameters a model uses having the goal to choose the best amount of parameters. We apply a generalisation for the number of parameters:

Let $\vec{y}$ be the vector of all N outcomes and $\hat{y}$ the predicted outcomes:

$$\vec{y} = S\hat{y}$$

where S is a $N \times N$ matrix depending on the input features $\vec{c}_i$ but not on the outcomes y_i. Using linear fitting methods (regression) and smoothing methods with quadratic shrinkage (rigid regression) give the effective number of parameters as:

$$d(S) = \text{trace}(S)$$

It turns out that trace (S) is exactly to replace d, the number of parameters of the model corresponding to S.

Note A much more general description of model complexity including the number of parameters is the Vapnik-Chernovenkis Dimension, that is not discussed in this script

6.5 Baysean Information Criterion (BIC)

The idea of BIC is to compare the a-posteriori probabilities of two models.
Suppose we have M models $\mathcal{M}$ (e.g. HMMs with different number of states, different HMM type, left-right, ...). Let furthermore θ_m denote the parameter vector corresponding to model $\mathcal{M}_m$. We assume the prior probability of the parameters is known: $p(\theta_m \mid \mathcal{M}_m)$.
The a-posteriori probabilities can be computed:

$$\frac{p\left(\mathcal{M}_m \mid \left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right)\right)}{p\left(\mathcal{M}_l \mid \left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right)\right)} = \frac{p\left(\mathcal{M}_m\right)}{p\left(\mathcal{M}_l\right)} \cdot \frac{p\left(\left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right) \mid \mathcal{M}_m\right)}{p\left(\left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right) \mid \mathcal{M}_l\right)}$$

where $\frac{p\left(\left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right) \mid \mathcal{M}_m\right)}{p\left(\left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right) \mid \mathcal{M}_l\right)}$ is called the **Baysean Factor**.
Approximating the class dependent density by Laplace Approximation, we get

$$\log p\left(\left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right) \mid \mathcal{M}_m\right) \simeq \log p\left(\left(\overrightarrow{c_1}, \Omega_{k(1)}\right), \ldots, \left(\overrightarrow{c_n}, \Omega_{k(n)}\right) \mid \hat{\theta}_m \mathcal{M}_m\right) - \frac{d_m}{2} \log N$$

where d_m denotes the number of free parameters of the model, N the number of features and $\hat{\theta}_m$ are the parameters of model $\mathcal{M}_m$ obtained by Maximum Likelihood. If the odds of this ratio are greater than 1, we choose model $\mathcal{M}_m$, otherwise we choose model $\mathcal{M}_l$.
Choosing the model with the lowest BIC is equivalent to choosing the model with the largest a-posterior probability. Even more, having computed the BIC for a set of M models, the estimates of the a-posteriors of any choosen model $\mathcal{M}_m \in M$ can be accessed very easily by:

$$\frac{e^{-\frac{1}{2}\,\mathrm{BIC}_m}}{\sum_{l=1}^{M} e^{-\frac{1}{2}\,\mathrm{BIC}_l}}$$

Rippley (1996) says, the more free parameters a model has, the poorer the model is rated. This is closely connected to the phenomenon of overfitting(see 11).

Properties

- BIC gives a heavy penalty on the complexity of models. This means it tends to choose simple ones. However having the number of samples diverge to infinity $N \to \infty$, BIC will indeed chose the best model out of a set of models.

6.6 MDL (Minimum Descritption Length)

The idea of MDL is to find a model, that is as small as possible and contains as much information as possible. The theory can be compared to **Prefix Coding** from Coding Theory:

Messages	He-Man	Optimus Prime	Lobo	Thor
Code 1	0	10	110	111
Code 2	10	110	111	0

No code may be prefix of another code. The word with the highest apearance (highest probability) gets the shortest code reducing the average length of a message.

Information Length(Shanon)

$$E\left(\text{length}\right) \geq -\sum_i p\left(c_i\right) \log_2 p\left(c_i\right)$$

In average we need $\log p\left(e\right)$ bits of information.

For pattern recognition this means: To transmit a random variable c having the probability denisty function $p\left(c\right)$, we require about $-\log_2 p\left(c_i\right)$ bits of information.

Now we want to apply this principle to Model Selection. We assume a model $\mathcal{M}$ with parameter vector θ and a training set $S = \left\{\left(\vec{c}_1; \Omega_{\kappa(1)}\right) \ldots \left(\vec{c}_n; \Omega_{\kappa(n)}\right)\right\}$. We assume that both sender and receiver know about the features, so we just transmit the class number. We can then give a defintion for the Message Lenght:

$$MDL = -\log_2 p\left(\Omega_\kappa \mid \theta, \mathcal{M}, \vec{c}_1 \ldots \vec{c}_n\right) - \log_e p\left(\theta \mid \mathcal{M}\right)$$

Using MDL for model selection we simply pick the model that mimizes the MDL.

Example

Let $\mathcal{M}$ be a gaussian. How can we compute $p\left(\theta \mid \mathcal{M}\right)$?

Use Bootstrap to estimate $p\left(\vec{\mu} \mid \mathcal{M}\right)$ and $p\left(\Sigma \mid \mathcal{M}\right)$ and combine it to $p\left(\theta \mid \mathcal{M}\right)$.

Properties

- We in fact ignored that transmitting continous random variables would require messages of infinite length, however introducing restrictions to the length just adds a constant factor to the formular that can be ignored for minimization procedures or comparissions between models

6.7 Ada Boosting

The idea of Ada Boosting is to combine several simple and weak classifier to obtain an overall better one. "Weak" means that the classifier's error rate is only slightly better than a random guessing.
A few ideas to combine classifiers:

- grouping features (subsets are classified by different classifiers)
- a weighted combination of classifier results
- weighted training vectors (problematic samples get a higher weight, similar to SVM 4.7)

The idea here is that the classified class number results from a majority vote of weighted (α) predicitions of all classifiers involved. Hereby weights give preference to more accurate classifiers. The are computed by the Ada-Boost Algorithm. A second aspect of this methods is the introduction of observation weights (w). At first all observations are weighted equally $w_i = \frac{1}{N}$. With the iterations proceeding misclassified observations gain higher weights, exercising a bigger influence on the weighting of the classifiers.

Algorithm 25 Ada Boosting

Given: a labeled training set $\left\{\left(\vec{c}_1, \Omega_{\kappa(1)}\right), \ldots, \left(\vec{c}_N, \Omega_{\kappa(N)}\right)\right\}$ where $\kappa \to \{0, 1\}$.
Initialization: initialize observation weights uniformly: $\forall_{i=1\ldots N} : w_i = \frac{1}{N}$.

1. fit a classifier $\delta_n(\cdot)$ to training data using the observation in respect to their weightings $w_i, i = 1 \ldots N$ (e.g. use a high weighted sample multiple times)
2. compute the classification error
$$\mathrm{ERR} = \frac{\sum_{i=1}^{N} w_i \chi\left(\delta_n\left(\vec{c}_i\right) \neq \kappa\left(i\right)\right)}{\sum_{i=1}^{N} w_i}$$
3. weight the classifier according to its classification error
$$\alpha_n = \log\left(\frac{(1 - \mathrm{ERR})}{\mathrm{ERR}}\right)$$
4. update the observation weights according to whether they have been misclassified (higher weight) or not (lower weight).
$$\forall_{i=1\ldots N} : w_i \leftarrow w_i \exp^{\alpha_n \chi\left(\delta_n\left(\vec{c}_i\right) \neq \kappa(i)\right)}$$
5. ↻ repeat for all classifiers $n = 2 \ldots M$, M being the number of classifiers involved.
6. The resulting classifier is a committee of all trained classifiers, where a majority vote weighted by the classifier weights α is cast
$$\delta\left(\vec{c}\right) = \mathrm{sgn} \sum_{n=1}^{M} \alpha_n \delta_n\left(\vec{c}\right)$$

7 State Estimation (Kalman-Filter)

The Kalman-Filter is a technique for estimating states of dynamic systems (see ??).
The internal state of the system is hidden, because we don't know the dynamic exterior influence completely. That is why we try to estimate the internal state from observations and steadily synchronize it with experience gained from new observations. That means we have to develop a way to correlate input (cause) with output (effect).

State

Because we are dealing with dynamic systems the output can not solely be determined by the input, but also depends on past observations (history of the system). Therefore we need a third quantity called state, a summary over all past happenings and present observations. The observations at a time instant number k, x_k and the input at the time interval $[k, k+1[$ u_k must be enough to compute the output at time k:

$$\vec{y}_k = h\left(\vec{x}_k, k\right) + \xi_k$$

Furthermore the internal state is changed, which we can describe by a equation:

$$\vec{x}_{k+1} = f\left(\vec{x}_k, u_k, k\right) + \eta_k$$

η_k and ξ_k denote noise and will be described later.
A discrete system is completely described by this two equations plus an initial state x_0.

Uncertainty

The system described above assumes that the external process is perfectly described in the state x and that the functions f and h are capable of modelling those relations exactly. This is very unrealistic for real world processes, therefore we introduce a fourth term called **uncertainty**. This states that the above functions f and h are in fact only approximations of much more complex systems, the approximation error is modelled by additive uncertainty noise terms η and ξ. For convenience we assume that they are normally distributed having zero mean, this simplifies the involved mathematics: $\eta_k \simeq \mathcal{N}\left(\vec{0}; Q\right)$, $\xi_k \simeq \mathcal{N}\left(\vec{0}; R\right)$.

Linearity

For this part we need some dimensions, so we say the input vector $\vec{u} \in \mathbb{R}^p$, the state vector $\vec{x} \in \mathbb{R}^n$ and the output vector $\vec{y} \in \mathbb{R}^m$.
If we find linear functions for f and h, the equations above become much more simple:

$$\vec{x}_{k+1} = F_k \vec{x}_k + G_k u_k + \eta_k$$

$$\vec{y}_k = H_k \vec{x}_k + \xi_k$$

we call these matrices:

state propagation matrix $F \in \mathbb{R}^{n \times n}$

input matrix $G \in \mathbb{R}^{n \times p}$

output matrix $H \in \mathbb{R}^{m \times n}$

The covariance matrix of the **system noise** η is called $Q \in \mathbb{R}^{n \times n}$ and the covariance matrix of the **output noise** ξ is called $R \in \mathbb{R}^{m \times m}$.

Example (motion tracking)

input images

output (observation) position, velocity

internal state trajectory (e.g a parabola $\curvearrowright$)

initial state guess for the initial speed and direction

error the estimation errors for the state η and the output ξ

update functions from physics modelling motion

Let us recapitulate: The Kalman-Filter allows for the estimation of $\vec{x}_{k|k}$ at time k and the corresponding covariance matrix $P_{k|k}$ given the measured observations $\vec{y_0}, \vec{y_1}, \ldots \vec{y_k}$. While $\vec{x}_{k+1|k}$ means the estimation of $\vec{x}_{k+1}$ using $\vec{y_0}, \vec{y_1}, \ldots \vec{y_k}$.

7.1 Variable Overview

Tabelle 6: Kalman-Filter Variable Lookup Table

variable	description
$\vec{x}$	internal state, a summary of all past happenings and the present observations
x_k	variable x at timestep k
$x_{k\|j}$	k: timestep the variable was estimated for j: number/index of measurements used for estimation $(\vec{y_0}, \vec{y_1}, \dots \vec{y_j})$
$\vec{y}$	measured observation
P	covariance matrix of the state, corresponding to the uncertainty (incorporates η and ξ)
η	error concerning state estimation
ξ	error concerning measuring observation
$\vec{n}$	general noise term
W	A transformation (matrix) for scaling noise
F, G	functions/matrices used for state update
H	function/matrix used for correlation with the observations
L	estimator for $\vec{x}$ $(\vec{x} = L(\vec{y}))$
R	covraiance matrix of the observation noise ξ
Q	covariance matrix of the state noise η
K	Kalman-Gain-Matrix

7.2 Least Square Estimation

Intuitively it might be clever to use Least Square Estimation to estimate the new state. The following algorithm would look like this:

Algorithm 26 Kalman-Filter (Least Square)

1. Initialization

   initialize $\vec{x}_0$ the initial state estimate and P_0 the corresponding covariance matrix.

2. Update

   Our current state is $\vec{x}_{k|k-1}$. Now we receive observation $\vec{y}_k$ and want to udpate the state to $\vec{x}_{k|k}$. Using only our current state our current best estimate for the output is:

   $$\vec{y}_{k|k-1} = H_k \cdot \vec{x}_{k|k-1}$$

   receving $\vec{y}_k$ we check the residue between our estimate and the observation: $\left(\vec{y}_k - \vec{y}_{k|k-1}\right)$. If this residue is nonzero, we need to correct the estimate for the state $\vec{x}_{k|k}$. We do this by using least square estimation:

   $$\left\|\vec{y}_k - \vec{y}_{k|k-1}\right\| \rightarrow \min$$

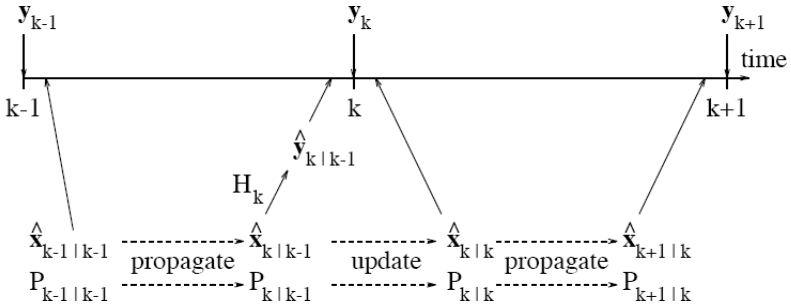
   our degree of freedom is $\vec{x}_{k|k}$

   $$\vec{x}_{k|k} = \text{argmin}_{\vec{x}_{k|k}} \left\|\vec{y}_k - H_k \cdot \vec{x}_{k|k}\right\|$$

   Note: The lest square estimation of $\vec{x}_{k|k}$ is not recommended:

   - observation $\vec{y}_k$ is taken for granted, noise is not considered.
   - uncertainty of the state $P_{k|k-1}$ is not considered.
   - no update of $P_{k|k-1}$.
   - a quantity for comparing $\vec{y}_k, \vec{y}_{k|k-1}$ is missing, this quantity is the uncertainty

3. Propagation



   When $\vec{y}_k$ is measured, we update both $\vec{x}_{k|k}$ and $P_{k|k}$. With Propagation we first update $\vec{x}_{k-1|k-1}$ and $P_{k-1|k-1}$ to $\vec{x}_{k|k-1}$ and $P_{k|k-1}$, then we receive the update and update both to $\vec{x}_{k|k}$ and $P_{k|k}$. After that we continue again with propagating $\vec{x}_{k|k}$ to $\vec{x}_{k+1|k}$ and $P_{k|k}$ to $P_{k+1|k}$.

   Propagation is transfering the covariance matrices Q and R of the uncertainty noise terms η and ξ into the covaraince matrix $P_{k|k-1}$. Therefore P contains all the probabilistic aspects and the degree in which η and ξ corrupt the quality of the state estimate $\vec{x}_{k|k}$

7.3 Best Linear Unbiased Estimator (BLUE)

We are still lacking a way to estimate states taking uncertainty, in the form of the covariance matrix P, into account. An estimation method taking into account additional probabilistic information is the BLUE.

The BLUE is an essential part of the Kalman-Filter, it alows the estimation of $\vec{x}$ in $\vec{y} = h(\vec{x})$, given $\vec{y}$ and h. We introduce a mapping L and call it estimator of $\vec{x}$:

$$\vec{x} = L(\vec{y})$$

7.3.1 linear

If h is invertible and if the noise is zero, L is simple the inverse of h.
If L is linear (e.g. a matrix), the estimator is called **linear estimator**.
In both cases, we either compute the Inverse H^{-1} (h is invertible) or the Pseudo-Invsere H^t (L is linear, but h is not) of H using SVD:

$$H^t\vec{y} = V\Sigma^{-1}U^T\vec{y} = \left(H^TH\right)^{-1}H^T\vec{y}$$

7.3.2 best

If we want to use a term like **best**, we have to define some kind of measure, which we can compare. In our case we take the euclidean norm of the residue:

$$\vec{y} = H\vec{x} + \vec{n}$$

that gives us the noise term n as a measure of comparing residues:

$$\vec{n} = \vec{y} - H\vec{x}$$

We could now simply minimize the noize by using least squares $\|\vec{n}\|^2 = \vec{n}^T\vec{n}$, but this won't work since we have different noise terms in different equations. Therefore we try to find a transformation W which produces equally scaled noise, then we minimize $(W \cdot \vec{n})^T (W \cdot \vec{n})$ instead.

Scaling Noise

We have different noise terms in different equations and therefore can not apply a simple least square estimation. To overcome this deficit we introduce the idea to scale the noise terms in a way, that noise in all equations becomes the same (in terms of variance). With noise scaling we also assure that the different noise covariances do not get lost, but result in a weight for the estimation (Weighted Least Squares, see below). If a covariance matrix is small, we believe in the authenticity of a measurement and consequently it should be weighted more heavily.

let $\vec{n}$ denote the noise vector. We look for a transform W

$$W \cdot \vec{n} = \vec{n}'$$

that minimizes

$$(W \cdot \vec{n})^T (W \cdot \vec{n}) = \vec{n}^T W^T W \vec{n} \to \min$$

this is called **Weighted Least Square Estimation**.

$$\vec{y} = H \cdot \vec{x} + \vec{n}$$

Apply Bootstrap to estimate $\vec{n}$ and then W.

We can compute W using Newton-Iteration (see 8.5):

let Σ be the covariance matrix of $\vec{n}$. Now we look for the covariance matrix of $W\vec{n}$:

$$\Sigma_{W\vec{n}} = W\Sigma W^T = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

After we got W we start scaling:

$$W\vec{y} = WH\vec{x} + W\vec{n}$$

$$\Leftrightarrow W\vec{n} = W\vec{y} - WH\vec{x}$$

$$\min_{\vec{x}} \|W\vec{n}\| = \min_{\vec{x}} \|W\vec{y} - WH\vec{x}\|$$

Now we use the Pseudo-Inverse

$$\hat{\vec{x}} = \left((WH)^T WH\right)^{-1} (WH)^T \cdot W\vec{y}$$

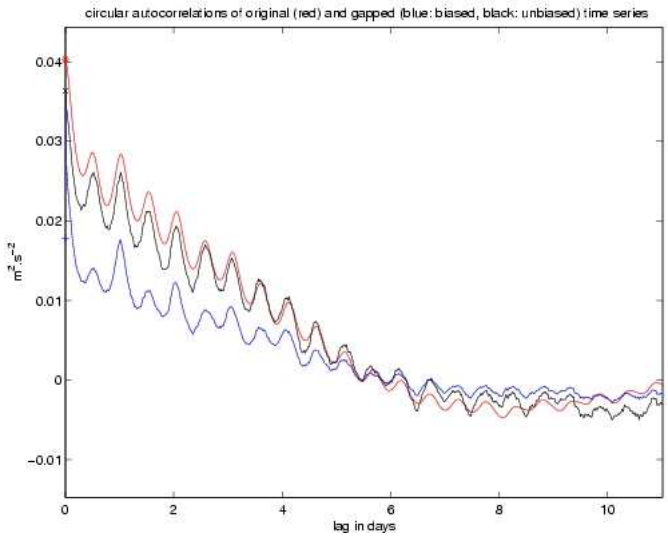
7.3.3 unbiased

An estimator is called **unbiased** if

$$E\left(\vec{x}^* - \hat{\vec{x}}\right) = \vec{0}$$

Remeber that the Bias was defined as BIAS=$E\left(\vec{x}^* - \hat{\vec{x}}\right)$, the expectation of the difference between the estimated value $\hat{\vec{x}}$ and the real value $\vec{x}^*$. Unbiased in general means that if we repeat the same estimation experiment several times, we never consistently overestimate or underestimate $\vec{x}$.

Abbildung 51: Unbiased Estimator



As you can see the biased estimator (blue) under and overestimates the origianl red values significantly, while the unbiased estimator's (black) over/underestomations are much less significantly and more constant.

7.3.4 The BLUE

Now we have everything to find the best linear unbiased estimator L to a certain problem.

$$\hat{\vec{x}} = L\vec{y}$$

with

$$\vec{y} = H\vec{x} + \vec{n}$$

First we need a condition, that keeps L unbiased:

Lemma If $E(\vec{n}) = \vec{0}$ (zero mean noise), then L is an unbiased estimator, if and only if $H \cdot L = I$ (identity matrix).

Abbildung 52: Proof For The Zero Mean Lemma

$E\left(\vec{x} - \hat{\vec{x}}\right) = 0$
$= E(\vec{x} - L\vec{y})$
$= E(\vec{x} - LH\vec{x} - L\vec{n})$
$= E((I - LH)\vec{x} - L\vec{n})$
$= E((I - LH)\vec{x}) - L \cdot E(\vec{n})$
$\Rightarrow (I - LH)E(\vec{x}) = \vec{0} \Leftrightarrow L \cdot H = I\square$

Theorem If $E(\vec{n}) = \vec{0}$ (zero mean) and $E\left(\vec{n}\vec{n}^T\right) = R$ (covariance matrix of the noise), then

$$L = \left(H^T R^{-1} H\right)^{-1} H^T R^{-1} = P H^T R^{-1}$$

and

$$P = E\left[\left(\vec{x} - \hat{\vec{x}}\right)\left(\vec{x} - \hat{\vec{x}}\right)^T\right] = \left(H^T R^{-1} H\right)^{-1}$$

By that we can update/compute both the estimator for $\vec{x}$ and the covariance matrix P for $\vec{x}$.

Abbildung 53: Proof For The Covariance Theorem

$$E\left(\left(\vec{x} - \hat{\vec{x}}\right)\left(\vec{x} - \hat{\vec{x}}\right)^T\right)$$
$$= E\left(\left(\vec{x} - L\vec{y}\right)\left(\vec{x} - L\vec{y}\right)^T\right)$$
$$= E\left(\left(\vec{x} - L\left(H\vec{x} + \vec{n}\right)\right)\left(\vec{x} - L\left(H\vec{x} + \vec{n}\right)\right)^T\right)$$
$$= E\left(\left((1 - LH)\cdot\vec{x} - L\vec{n}\right)\left((1 - LH)\cdot\vec{x} - L\vec{n}\right)^T\right) \mid \text{first precondition: } L \text{ is unbiased, so } 1 - LH = 0$$
$$= E\left(\left(L\vec{n}\right)\left(L\vec{n}\right)^T\right)$$
$$= E\left(L\vec{n}\vec{n}^T L^T\right)$$
$$= L \cdot E\left(\vec{n}\vec{n}^T\right) \cdot L^T \mid \text{second precondition: } E\left(\vec{n}\vec{n}^T\right) = R$$
$$= LRL^T$$
$$= \left(H^T R^{-1} H\right)^{-1} H^T R^{-1} R \left(R^{-1}\right)^T H \left(\left(H^T R^{-1} H\right)^{-1}\right)^T \mid R^{-1}R = 1$$
$$= \left(H^T R^{-1} H\right)^{-1}\left(H^T R^{-1} H\right) \cdot \left(\left(H^T R^{-1} H\right)^{-1}\right)^T \mid$$
$\left(H^T R^{-1} H\right)^{-1}\left(H^T R^{-1} H\right) = 1$, $\left(H^T R^{-1} H\right)$ is symmetric so $X^T = X$
$$= \left(H^T R^{-1} H\right)^{-1} \square$$

7.4 The Linear Kalman Filter

The Kalman filter has two distinct phases: Prediction and Update. The prediction phase uses the estimate from the previous timestep to produce an estimate of the current state (Propagation). In the update phase, measurement information from the current timestep is used to refine this prediction to arrive at a new, (hopefully) more accurate estimate.

Remember the two fundamental Kalman-Equations:

(I) $\vec{x}_{k+1} = F_k \cdot \vec{x}_k + G_k \vec{u}_k + \vec{\eta}_k$

(II) $\vec{y}_k = H_k \vec{x}_k + \vec{\xi}_k$

with $\vec{\eta}_k \simeq \mathcal{N}\left(\vec{0}, Q_n\right)$ and $\vec{\xi}_k \simeq \mathcal{N}\left(\vec{0}, R_k\right)$.
With the results from the previous section we know that

$$\hat{\vec{x}} = PH^T R^{-1} \vec{y}$$

with P being the covariance matrix of the estimation error

$$P = \left(H^T R^{-1} H\right)^{-1}$$

is the best linear unbiased estimate of the state $\vec{x}$.

Prediction Phase

We want to predict the state $\vec{x}_{k|k-1}$ and the corresponding covariance matrix $P_{k|k-1}$

We use the BLUE L, to predict new estimates for $\vec{x}_k$ and P_k for time k given the observations $\vec{y}_0 \ldots \vec{y}_{k-1}$. These estimates differ from the real quantities by an error term $\vec{e}_k$:

(I)

$$\hat{\vec{x}}_{k|k-1} = \vec{x}_k + \vec{e}_k$$
$$P_{k|k-1} = E\left(\vec{e}_k \vec{e}_k^T\right)$$

where $\vec{e}_k$ is an error vector.

Update Phase

Now comes the observation $\vec{y}_k$ at timestep k, our current state is $\hat{\vec{x}}_{k|k-1}$ with covariance matrix $P_{k|k-1}$:

(II)

$$\vec{y}_k = H_k \vec{x_k} + \vec{\xi_k}$$

with error covariance

$$E\left(\vec{\xi}_k \vec{\xi}_k^T\right) = R_k$$

We can combine (I) and (II) writing them in matrix form:

$$\begin{pmatrix} \vec{x}_{k|k-1} \\ \vec{y} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \tilde{H}_k \end{pmatrix} \begin{pmatrix} \vec{x}_k \\ \vec{x}_k \end{pmatrix} + \begin{pmatrix} \vec{e}_k \\ \vec{\xi}_k \end{pmatrix}$$

where the last error term has the covariance matrix

$$R = \begin{bmatrix} P_{k|k-1} & 0 \\ 0 & R_k \end{bmatrix}$$

From BLUE we know, that:

$$\hat{\vec{x}} = PH^T R^{-1} \vec{y}$$

so for timestep $k \mid k$ we get

$$\vec{x}_{k|k} = P_{k|k} H_k^T R_k^{-1} \vec{y}_{k|k-1}$$

for the state and

$$P_{k|k} = \left(H_k^T R_k^{-1} H_k\right)^{-1}$$

for the noise covaraince, where

$$R_k = E\left(\begin{pmatrix} \vec{e}_k \\ \vec{\xi}_k \end{pmatrix}\begin{pmatrix} \vec{e}_k \\ \vec{\xi}_k \end{pmatrix}^T\right)$$

These computations are not very efficient, because H and R contain many zeros, therefore we rewrite them in the real update and propagation algorithm:

Update Of The Covariance Matrix $P_{k|k}$

$$\begin{aligned} P_{k|k}^{-1} &= H_k^T R_k^{-1} H_k \text{ (BLUE)} \\ &= \begin{bmatrix} I & H_k^T \end{bmatrix}\begin{bmatrix} P_{k|k-1}^{-1} & 0 \\ 0 & R_k^{-1} \end{bmatrix}\begin{bmatrix} I \\ H_k \end{bmatrix} \\ &= \left(P_{k|k-1}^{-1} H_k R_k^{-1}\right)\begin{pmatrix} I \\ \tilde{H}_k \end{pmatrix} \end{aligned}$$

$$P_{k|k}^{-1} = P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k$$

Update Of The State $\vec{x}_{k|k}$

$$\begin{aligned} \vec{x}_{k|k} &= P_{k|k} H_K^T R_K^{-1} \vec{y} \text{ (BLUE)} \\ &= P_{k|k}\begin{bmatrix} P_{k|k-1}^{-1} & H_k^T R_k^{-1} \end{bmatrix}\begin{bmatrix} \vec{x}_{k|k-1} \\ \vec{y}_k \end{bmatrix} \\ &= P_{k|k}\left(P_{k|k-1}^{-1}\vec{x}_{k|k-1} + H_k^T R_k^{-1}\vec{y}_k\right) \\ &= \hat{\vec{x}}_{k|k-1} - P_{k|k} H_k^T R_k^{-1} H_k \vec{x}_{k|k-1} + H_k^T R_k^{-1}\vec{y_k} \end{aligned}$$

$$\vec{x}_{k|k} = \hat{\vec{x}}_{k|k-1} - P_{k|k} H_k^T R_k^{-1}\left(\vec{y}_k - H_k \hat{\vec{x}}_{k|k-1}\right)$$

The part $\left(\vec{y}_k - H_k \hat{\vec{x}}_{k|k-1}\right)$ is the **residue** between the actual measurement $\vec{y}_k$ and its best estimate based on $\hat{\vec{x}}_{k|k-1}$.
The part $P_{k|k} H_k^T R_k^{-1}$ of the last term is called the **Kalman Gain Matrix** K_k. It specifies the amount by which the residue must be multiplied to botain the correction term that transforms the old estimate $\hat{\vec{x}}_{k|k-1}$ into the updated estimate $\vec{x}_{k|k}$.

Propagation

The state can be propagated by

$$\hat{\vec{x}}_{k|k-1} = F_k \vec{x}_{k-1|x-1} + G_k u_k$$

fulfilling the unbiasedness constraint. The covariance matrix is propagated thanks to the linearity of L by

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

Kalman Filter Equations

Now we can give the correct algorithm for the Kalman-Filter:

Algorithm 27 Kalman-Filter

1. Predict Phase
   $\hat{\vec{x}}_{k|k-1} = F_k \vec{x}_{k-1|k-1} + G_k u_k + \vec{\eta}_k$
   $P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$
2. Update Phase
   $K_k = P_{k|k} H_k^T R_k^{-1}$
   $\vec{x}_{k|k} = \hat{\vec{x}}_{k|k-1} + K_k \left(\vec{y}_k - H_k \hat{\vec{x}}_{k|k-1} \right)$
   $P_{k|k}^{-1} = P_{k|k-1}^{-1} + H_k^T R_k^{-1} H_k$

Invariants

- $E\left[\vec{x}_k - \hat{\vec{x}}_{k|k}\right] = E\left[\vec{x}_k - \hat{\vec{x}}_{k|k-1}\right] = 0$
- $E\left[\tilde{\vec{y}}_k\right] = 0$
- $P_{k|k} = \mathrm{cov}\left(\vec{x}_k - \hat{\vec{x}}_{k|k}\right)$
- $P_{k|k-1} = \mathrm{cov}\left(\vec{x}_k - \hat{\vec{x}}_{k|k-1}\right)$

7.5 Discussion

Properties

- recursive procedure (only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state)

- computes the internal state of a dynamic system and the uncertainty matrix based on past happenings and current measurements.
- does not solve dynamic systems all at once, but steadily refined an initialsolution over time
- the more data is measured, the better the solution become

Pro/Contra

- different equations can have different noise terms with different varaince. BLUE does not respect this.

Fields Of Application

- tracking (estimating the trajectory of images (e.g. throwing of a ball))
- keeping a space ship (or a plane) on track
- PLL filter in radios, TVs, computers and most video/audio communication devices

8 Basic Methods

8.1 Maximum Likelihood (ML)

We have given a sample of n values $C = \{c_1, c_2, \ldots, c_n\}$ which we assume underlie the probability density function f_θ, where θ are the parameters of the distribution (e.g. mean μ and covariance Σ, if the pdf is gaussian distribution $\mathcal{N}$). With ML we will estimate the values of the parameter vector θ.

$$L(\theta) = f_\theta(C \mid \theta)$$

is called the **Likelihood Function**. By that we compute the probability density associated to our sample data. In case that θ is not observable, ML uses the value of θ that maximizes $L(\theta)$. The resulting estimated $\hat{\theta}$ is called the **Maximum Likelihood Estimator** of θ.

$$\hat{\theta}_{ML} = \text{argmax}_\theta \, p(C \mid \theta)$$

where

$$p(C \mid \theta) = \prod_{i=1}^{n} p(c_i \mid \theta)$$

asuming that the features in $\vec{c}$ are statistically independent of each other.

In Summa

The Maximum Likelihood Estimator is the parameter vector $\hat{\theta}_{ML}$ maximizing the probability of a sample set $\{c_1, c_2, \ldots, c_n\}$.

Properties

- The Maximum Likelihood Estimator $\hat{\theta}$ does not always exist and ist not unique
- If no labeled samples are available, ML can still be used adding the unkown class number as an additional parameter to θ.E.g. for gaussian distribution θ would look like this: $\theta = \{\kappa(\vec{\mu}_\kappa, \Sigma_\kappa \mid \kappa = 1 \ldots K)\}$.
- equvivalently we can use a monotonic probability function p, e.g. the log likelihood

$$p(C \mid \theta) = \sum_{i=1}^{n} \log(p(c_i \mid \theta))$$

Pro/Contra

- the whole sample set has to be stored and evaluated each turn (except for solid statistical densities like the normal distribution)

8.2 Maximum A-Posteriori Estimation (MAP)

The idea of **Maximum A-Posteriori Estimation** (or **Bayse Estimation**) is that the parameter vector θ we want to estimate, also unerlies a distribution. It maximizes the a-posteriori probability of θ.

Example

If we choose a gaussian model, one of the parameters will be the mean value. Now imagine school exams. Every exam has a mean value of the marks. Yet after 25 exams even this mean value underlies the distribution of the 24 previous mean values. A mean value of the mean values can be given.

The Maximum A-Posteriori Estimation is defined:

$$\hat{\theta}_{MAP} = \text{argmax}_\theta \, p\left(\theta \mid \vec{c}\right) = \text{argmax}_\theta \, p\left(\theta\right) \cdot p\left(\vec{c} \mid \theta\right)$$

The difference to ML-Estimation is that prior knwoledge is respected and used. Since prior knowledge becomes vain when having a uniform distribution, this is exactly the scenario when MAP equals ML.

Properties

- corresponds to a ML estimation also using prior knowledge
- is euqal to ML, if we have a uniform distribution (prior knowledge becomes useless)
- If no labeled samples are available, MAP can still be used adding the unkown class number as an additional parameter to θ. E.g. for gaussian distribution θ would look like this: $\theta = \{\kappa\left(\vec{\mu}_\kappa, \Sigma_\kappa \mid \kappa = 1 \ldots K\right)\}$.

Pro/Contra

- the whole sample set has to be stored and evaluated each turn (expect for solid statistical densities like the normal distribution)

8.3 Expectation Maximization (EM)

We have objects built given a certain amount of parametres. Some of them can be measured (observable information) others can't (hidden information). The objects are entities of an unknown random variable, which underlies a certain density. The goal of the EM Algorithm is to determine the parameters of this density. Furthermore the EM Algorithm is based on the Hidden Information Principle:

observable information = complete information - hidden information

X observable information

Y hidden information

$\mathcal{B}$ parameter vector of the density

Usually $p(X, Y; \mathcal{B})$ is assumed to be a mixture density (i.e. a weighted sum of standard densities). If we assume gaussian distribution, the parameters to estimate would be mean μ and covariance Σ. If we assume mixtures, we furthermore estimate the weights of the single densities as parameters. Usually you would apply ML in such a case, the porblem is that our parameters depend on hidden information, which cannot be modelled by ML. EM solves this by simultaneously estimating observable and hidden parameters.

Theory

$$p(X, Y; \mathcal{B}) = p(Y, X; \mathcal{B}) \cdot p(X; \mathcal{B}) = p(X \mid Y, \mathcal{B}) \cdot p(Y; \mathcal{B})$$

apply logarithm for easier calculations

$$\log p(X, Y; \mathcal{B}) = \log p(X \mid Y, \mathcal{B}) + \log p(Y; \mathcal{B})$$

solving for $\log p(X \mid Y, \mathcal{B})$ we get the key equation of the EM:

$$\log p(X \mid Y, \mathcal{B}) = \log (X, Y; \mathcal{B}) - \log p(Y; \mathcal{B})$$
$$\Rightarrow \log p(X; \mathcal{B}) = \log p(X, Y; \mathcal{B}) - \log p(Y \mid X; \mathcal{B})$$

this corresponds to the log likelihood of the observed random variables.

Now we consider the $(i+1)$ iteration and seperate the equation into two terms:

$$\log \left(X; \hat{\mathcal{B}}^{(i+1)}\right) = Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) - H\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right)$$

Q $Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) = \int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) \log p\left(X, Y; \hat{\mathcal{B}}^{(i+1)}\right) dY$ (Kullbach-Leibler Statistics)

H $H\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) = \int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) \log p\left(Y \mid X; \hat{\mathcal{B}}^{(i+1)}\right) dY$ (Entropy)

Discussion of the H-Term The H-Term is not important for the maximization, maximizing the Q-Term is sufficient:

$$H\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) - H\left(\hat{\mathcal{B}}^{(i)} \mid \hat{\mathcal{B}}^{(i)}\right) =$$

$$\int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) \log\left(Y \mid X; \hat{\mathcal{B}}^{(i+1)}\right) dY - \int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) \log p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) dY =$$

$$= \int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) \log \frac{p\left(Y \mid X; \hat{\mathcal{B}}^{(i+1)}\right)}{p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right)} dY$$

replace $\log x$ by $\leq x - 1$ ($x - 1 \geq \log x$):

$$\leq \int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) \cdot \left(\frac{p\left(Y \mid X; \hat{\mathcal{B}}^{(i+1)}\right)}{p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right)} - 1\right) dy =$$

$$= \int p\left(Y \mid X; \hat{\mathcal{B}}^{(i+1)}\right) - p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) dy =$$

$$1 - 1 = 0$$

Discussion of the Q-Term An iterative maximization of the Q-Term corresponds to a maximum likelihood estimation. The Q-Term corresponds to the gradient of the estimation. It can easily be decomposed:

$$Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right) = \int \frac{p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right)}{\int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) dy} \cdot \log p\left(X, Y; \hat{\mathcal{B}}^{(i+1)}\right) dY$$

since $\int p\left(Y \mid X; \hat{\mathcal{B}}^{(i)}\right) dy$ marginalizes up to 1.

- the parameter vector $\hat{\mathcal{B}}^{(i)}$is the current best estimate so far
- $\hat{\mathcal{B}}^{(i+1)}$ is a candidate for an improved estimate
- Q calculates the likelihood of the data in respect to $\hat{\mathcal{B}}^{(i+1)}$
- the Q-term includes hidden information Y (the index l, of the candidates)
- the EM-Algorithm finds out the best $\hat{\mathcal{B}}^{(i+1)}$ that maximizes Q

Algorithm

Algorithm 28 Expectation Maximization

---

1. Initialization: choose a good initialization for $\hat{\mathcal{B}}^{(0)}$. Use task specific knowledge.

2. Estimation Step:
   Estimate Y from the observable parameters X and $\hat{\mathcal{B}}^{(i)}$ by computing

$$Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right)$$

3. Maximization Step:
   Determine the best estimate for $\hat{\mathcal{B}}^{(i+1)}$ using ML. Since we have estimatedthe hidden parameters Y, we can now make use of ML.

$$\hat{\mathcal{B}}^{(i+1)} = \text{argmax}_{\hat{\mathcal{B}}^{(i+1)}} Q\left(\hat{\mathcal{B}}^{(i+1)} \mid \hat{\mathcal{B}}^{(i)}\right)$$

4. Return $\hat{\mathcal{B}}^{(i+1)}$ if $\hat{\mathcal{B}}^{(i+1)} - \hat{\mathcal{B}}^{(i)} < \varepsilon$.

Properties

- iterative procedure
- highly dependent on initialization
- only local optima can be found
- always converges
- leads to a decomposition of the search space
- snail method (tremendously slow)
- easy to implement (at least for pattern recognition tasks), because of the closed iteration scheme for parameter update
- numerical robustness
- approximates ML
- read in the book *The Elements Of Statistical Learning*(13) for an interesting way to realize EM with LDA, FDA, PDA and MDA (3.1.15 ff)

8.4 Histogram Estimation

If we assume that all features are statistically independent of each other:

$$p\left(\vec{c} \mid \Omega_\kappa\right) = \prod_{\nu=1}^{n} p\left(c_\nu \mid \Omega_\kappa\right)$$

Then it is sufficient to estimate n one dimensional densities of the features. This can still be done by ML and MAP, yet a much easier way is to use histogramms and count relative frequencies. In this case the density can be more efficiently stored in histogramms.

Pro/Contra

+ only the histogramms must be stored

- does only work for statistical independent features

8.5 Newton Iteration

The Newton Iteration is iterative method to find local maxima/minima not unlike gradient descent. Thus we move along the gradient to the point of the highest ascent f^* (a local maximum) and steadily replace the function by its tangent. Then we compute the zero of this tangent rather than the zero of the function (it is typically a better approximation). That also means this method is a local method and requires a good initial guess x_0. The iterative scheme is very simple:

$$x_{n+1} = x_n - \gamma \frac{f\left(x_n\right)}{f'\left(x_n\right)}$$

γ denotes the step width of the method. In many cases it is simply set to zero $\gamma = 1$.

Pro/Contra

+ faster than gradient descent

8.6 Gradient Descent

Gradient Descent is an iterative optimization algorithm that approaches a local minimum of a function by taking steps proportional to the negative of the gradient of the function at the current point. Which extreme point will be found depends on our initial guess x_0.

$$x_{n+1} = x_n - \gamma_n \nabla f\left(x_n\right)$$

where γ denotes the step width of the method. It can be determined using Cauchy methods.

- it often results in a zig-zag approachal of the maximum and is therefore rather slow

8.7 Coordinate Descent

Coordinate Descent is a fairly simple intuitive way of local optimization. The principle is to hold all expect one dimension/coordinate fixed, and only change the one left, till an optimum is reached. Then continue with another free coordinate.

Example

$f(x_1, x_2, x_3, x_4)$

1. x_2, x_3, x_4 are fixed. Change x_1 until f has reached an optimum in x_1^*
2. x_1^*, x_3, x_4 are fixed. Change x_2 until f has reached an optimum in x_2^*
3. x_1^*, x_2^*, x_4 are fixed. Change x_3 until f has reached an optimum in x_3^*
4. x_1^*, x_2^*, x_3^* are fixed. Change x_4 until f has reached an optimum in x_4^*

Our local optimum is located in $\left(x_{1,}^*, x_2^*, x_3^*, x_4^*\right)$.

8.8 Least-Squares

Least squares is a mathematical optimization technique which, when given a series of measurements y_i, attempts to find a function which closely approximates the data.

$$S = \sum_{i=1}^{n} \left(y_i - f\left(x_i\right)\right)^2$$

f can be any parametric function, e.g. a polygon. The goal is to to choose these parameters, such that S is minimized ('least squares'). For example we can use derivation, gradient descent, ML or similart techniques to determine the parameters.

The measurement error is assumed to:

- be stochastical independent
- be gaussian distributed
- to have a variance σ, that remains constant

8.9 Hough-Transformation

Hough-Transformation is a method to detect lines, circles and other geometrical figures.

Precondition Edge-detection on a binarized picture has been performed.

Idea Build a Dualspace in which every point corresponds to a geometrical object.

Pro/Contra

- brute-force algorithm
- computational intensive

Example: Line

We could use the gradient and the y-value of a line as parameters, but they have prooven to be a very bad parametrization (see 6.2). So we choose the parametric representation with angle and x-value for a line d:

$$d = x \cdot \cos(\alpha) + y \cdot \sin(\alpha)$$

and build the Dualspace upon (x, α), then every point in the dualspace corresponds to a line.

8.10 Lagrange Multiplier Method

8.10.1 Motivation

In standard maximization (minimization) methods, an optimal situation, where we can optimize without caring for any side conditions, is asumed. Since optimization in Pattern Recognition applications is usually always bound to side conditions, like optimizing the inter class distance, while keeping the intra class distance constant, we take a deeper look into the **Lagrange Multiplier Method**, a method allowing for the optimizing of a function under several side constraints.

8.10.2 General Procedure

Let $f(x_1, x_2, \ldots, x_n)$ be a multidimensional function. This will be our base function to optimize. Beneath this function, we are given several constraints to respect during the optimization:

$$g_1(x_1, x_2, \ldots, x_n) = 0$$

$$g_2(x_1, x_2, \ldots, x_n) = 0$$

$$\vdots$$

$$g_m(x_1, x_2, \ldots, x_n) = 0$$

For each constraint we define a **Lagrange Multiplier** λ_j with $j \in \mathbb{Z}_{1,m}$.

Now we can write down an extended objective function f^L containing the base function and all constraints:

$$f^L(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n) + \lambda_1 \cdot g_1(x_1, x_2, \ldots, x_n) + \ldots + \lambda_m \cdot g_m(x_1, x_2, \ldots, x_n)$$

respectively:

$$f^L(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_n) + \sum_{j=1}^{m} \lambda_j \cdot g_j(x_1, x_2, \ldots, x_n)$$

8.10.3 Criterions For Maxima/Minima

First we have to check the **necessary criterion** $\mathcal{J}(f^L) = (0)$, whether the JACOBIAN-Matrix of or extended objective function equals the zero matrix.

Checking this criterion, we have to partially derive f^L to all n variables and m multipliers, resulting in $n + m$ partial derivations. In the next step we use a system of linear equations or respectively Gauss Elimination to determine points satisfying: $\mathcal{J}(f^L) = (0)$. Points in which all partial derivatives draw 0. These points are referred to as **stationary points**.

Finding a **satisfying criterion** for Minima/Maxima is more complicated. As in derivation theory, we have to take a closer look at the environment of our candidate:

Local Minimum A stationary point $(a_1, \ldots, a_n) \in \mathbb{K}^n$ is local minimum of our objective function, if there exists $\varepsilon > 0$ with $U_\varepsilon(a_1, \ldots, a_n) \subseteq D_f$ with U_ε referring to the environment given by ε and

$$f(a_1, \ldots, a_n) \leq f(x_1, \ldots, x_n) \text{ for each } (x_1, \ldots, x_n) \in \mathbb{K}^n \cap U_\varepsilon(a_1, \ldots, a_n)$$

Local Maximum A stationary point $(a_1, \ldots, a_n) \in \mathbb{K}^n$ is local maximum of our objective function, if there exits $\varepsilon > 0$ with $U_\varepsilon(a_1, \ldots, a_n) \subseteq D_f$ and

$$f(a_1, \ldots, a_n) \geq f(x_1, \ldots, x_n) \text{ for each } (x_1, \ldots, x_n) \in \mathbb{K}^n \cap U_\varepsilon(a_1, \ldots, a_n)$$

Example

Let $f : \mathbb{R}^2 \to \mathbb{R}$

$$f(x, y) = -x^2 - \frac{1}{2}y^2 + 4$$

Furthermore let $g(x, y) = 2 - 2x - y = 0$ be a side constraint for f.

Our extended objective function f^L is then

$$f^L(x, y, \lambda) = -x^2 - \frac{1}{2}y^2 + 4 + \lambda(2 - 2x - y)$$

Partially deriving into the directions of all variables draws:
$f_x^{L'}(x, y, \lambda) = -2x - 2\lambda$
$f_y^{L'}(x, y, \lambda) = -y - \lambda$
$f_\lambda^{L'}(x, y, \lambda) = 2 - 2x - y$
Solving for $f^{L'} = 0$ gives us the stationary point $S := (x, y) = \left(\frac{2}{3}, \frac{2}{3}\right)$ and the Lagrange Multiplier $\lambda = -\frac{3}{2}$.

8.11 Karush-Kuhn-Tucker Conditions

given $f(x)$, the function to optimize. $g_i(x)$non equality constraints$(i = 1 \dots l)$.
$h_j(x)$ equality constraints $(j = 1 \dots l)$.

local minimum (necessary)

f, g_i and h_j are continuously differentiable at a regular point x^*[8] .
If x^* is a local Minimum, there exist multipliers $\mu_i > 0$ and ν_j such that

$$\nabla f(x^*) + \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{l} \nu_i \nabla h_i(x^*) = 0$$

with $\mu_i g_i(x^*) = 0$ for all $i = 1 \dots m$.

local maximum (sufficient)

f, g_i and h_j are convex functions. And let there be a feasible point x^*.
x^* is a global Minimum, there exist multipliers $\mu_i > 0$ and ν_j such that

$$\nabla f(x^*) + \sum_{i=1}^{m} \mu_i \nabla g_i(x^*) + \sum_{j=1}^{l} \nu_i \nabla h_i(x^*) = 0$$

with $\mu_i g_i(x^*) = 0$ for all $i = 1 \dots m$.

8.12 Singular Value Decomposition (SVD)

Singular Value Decomposition is an efficient way to factorize matrices and compute the eigenvalues and singular values of a matrix.

[8] a point is regular if the gradients of the active inequality constraints and the gradients of the equality constraints are linearly independent at that point

Singular Values

The matrix we want to decompose is $A \in \mathbb{R}^{m \times n}$.

We factorize A to

$$A = U \Sigma V^T$$

where $U \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$ and $\Sigma^{n \times n}$.

Furthermore

$$U \cdot U^T = I$$

having orthonormal column vectors, and

$$V \cdot V^T = V^T \cdot V = I$$

having orthonormal cloumn and row vectors.

Last but not least Σ is a diagonal matrix:

$$\Sigma = \begin{pmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_n \end{pmatrix}$$

where $\sigma_1 \geq \cdots \geq \sigma_n$ are called the **singular values**.

Condition

The Condition describes how dependent the solution is on disturbances (variances) of the input (numerical robustness). This is expressed by the **condition number**

$$\kappa = \frac{\sigma_1}{\sigma_n}$$

The lower κ, the numerically robust is the solution. A condition number close to 1 ($\kappa \simeq 1$) indicates that a problem is good conditioned. κ can be improved by setting singular values close to zero to zero ($\sigma_i \simeq 0 \rightarrow \sigma_i^* = 0$).

Orthogonalization

SVD is an excellent choice for keeping orthogonal matrices orthogonal. Because of the nummerical instability with computer matrices, orthogonal matriced loose their orthogonal property after operations on them. Using SVD this derivation manifests in singular values that are very close to, but not equal to 1. The solution is to set these $\sigma_i = 1$ and voila, the matrix is orthogonal again.

Orthonormalization

If the task is not to keep a orthogonal/orthonormal matrix orthogonal/orthonormal, but to initially find one, SVD can also be of help: Just take the column vectors of U and you've got yourself an orthonormal basis of the space A is creating.

Pseudoinverse

Another possible application is to compute the Pseudoinverse of a matrix:

1. First choose for every singular value $\sigma_i^* = \frac{1}{\sigma_i}$ instead of σ_i, except for singular values equal to 0
2. That gives us Σ^* with σ^* on the diagonal
3. Now the Pseudoinverse is simply $A^t = V\Sigma^* U^T$

8.13 Computation Of Implicit Eigenvalues

A problem in many applications is the calculation of eigenvalues of the kernel matrix Q (see 10.1) or respectively the covariance matrix Σ (see 10.1). A genious trick is to compute **Implicit Eigenvalues** factorizing Q resp. Σ:

$$Q = F \cdot F^T$$

and we choose $F \in \mathbb{R}^{N \times p}$ with $p \lll N$.

The eigenvalues can be gotten by:

$$Q\vec{\varphi}_c = \lambda_c \vec{\varphi}_c$$

so:

$$FF^T\vec{\varphi}_c = \lambda_c\vec{\varphi}_c$$
$$F^TF\left(F^T \cdot \vec{\varphi}_c\right) = \lambda_c \cdot \left(F^T \cdot \vec{\varphi}_c\right)$$

but F^TF is mereley a $p \times p$ matrix and thus much much smaller than Q.

Factorisation Of Q

To achieve this we have to factorize Q according to: $Q \to F \cdot F^T$. We will use the kernel matrix, gained by solving the objective S_1(see 3.1.12) to examplify this decomposition:

$$Q = \frac{1}{N^2} \cdot \sum_{i=1}^{N}\sum_{j=1}^{N}\left(\vec{f}_i - \vec{f}_j\right)\left(\vec{f}_i - \vec{f}_j\right)^T$$

$$Q = \frac{1}{N^2} \cdot F \cdot F^T$$
$$\Rightarrow F = \begin{pmatrix} \left(\vec{f}_{i_1} - \vec{f}_{j_1}\right)^T \\ \left(\vec{f}_{i_2} - \vec{f}_{j_2}\right)^T \\ \vdots \end{pmatrix}$$

Example

$$\sum_{i}^{N} a_i b_i = \vec{a}^T \vec{b}$$

8.14 Parameter Tying

Parameter Tying in general means to combine two parameters into one. E.g. having parameters a and b we could combine them into a new parameter using their sum $c = a+b$, or any other mathematical operator. In Pattern Recognition we often have too many parameters, i.e. features, and want to reduce their number, also reducing the compputational complexity of the classification process. In this respect Parameter Tying can be of use. Naturally it works best, if the two parameters combined, indeed share some real life connections.

Equal Covariance Matrices

Many pattern recognition methods assume or require equal covariance matrices. E.g. LDA (see 3.1.15) or the Mahalanobis Distance (see 3.2.8), yet in general they never are. But we can use a special form of parametery tying assuming them to be equal. As in the $c = a+b$ example above, this can of course be done in various ways. Here are given two that theoretically make sense:

$$\Sigma = \frac{N_\kappa}{N}\Sigma_\kappa + \frac{N_\lambda}{N}\Sigma_\lambda$$

and

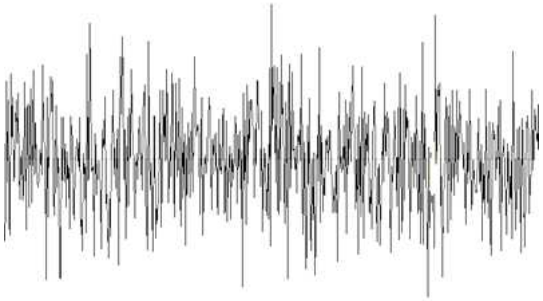
$$\Sigma = \frac{1}{N}\sum_{j=1}^{N} \vec{c}_j \vec{c}_j^T - \vec{\mu}\vec{\mu}^T$$

and the mean

$$\vec{\mu} = \frac{1}{N}\sum_{j=1}^{N} \vec{c}_j$$

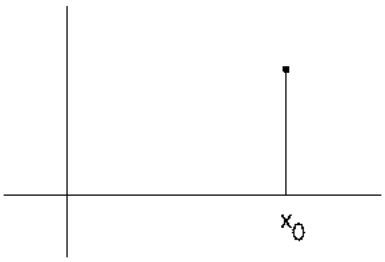
where N denotes the number of features and N_κ the number of features corresponding to class Ω_κ.

8.15 Disturb Signal



goal find interfering signal g

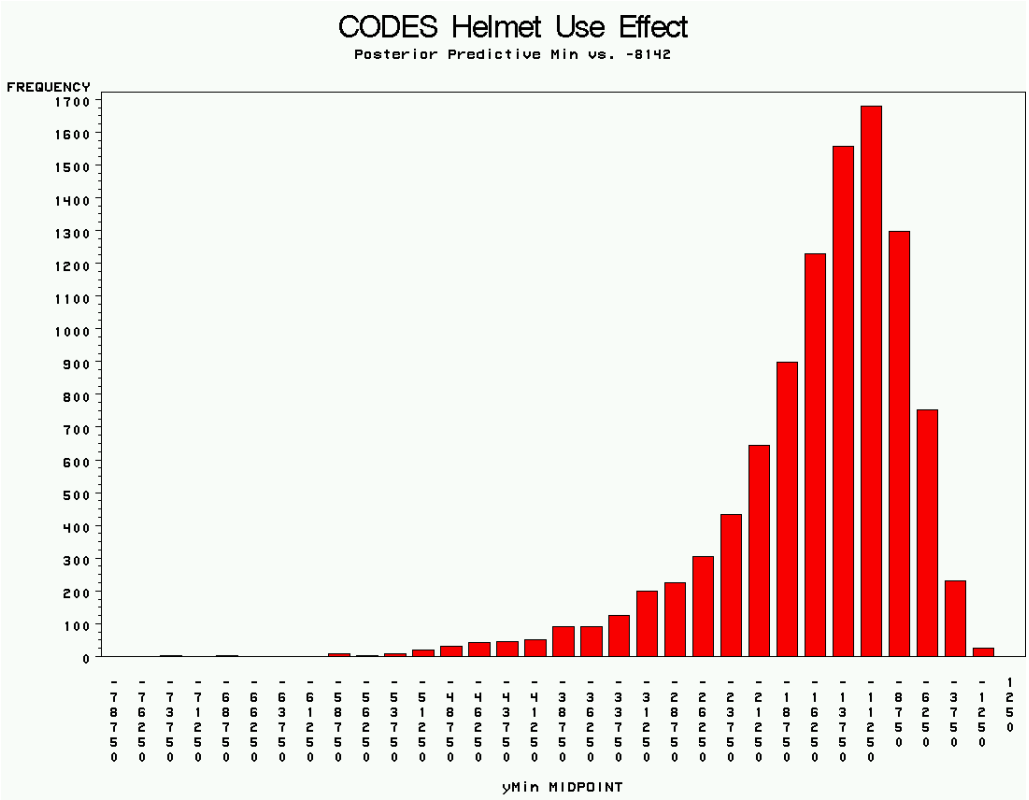
1. Send a dirac impuls (see 9.3) as signal f and measure the impuls response h



2. Do a transform into the frequency space (Fourier Space): $G = \frac{H}{F}$ (remember: $f \oplus g = h \Leftrightarrow F \cdot G = H$, see 9.1)
3. Transform G back, and you got the disturb signal isloated.

8.16 Histogram

Abbildung 54: Histogram



A histogram is a graphical way to display relative frequencies of measurements. In pattern recognition a bar usually sums up the features belonging to one class. In image processing histrogramms usually display the grey or color value distribution of a picture.

8.17 Knowledge Representation

Independent of which type of signal we want to classify or which type of features and classifier we choose, we have to choose a representation for our knowledge. Possible representations are:

- probability density functions, parametric families (for statistics)
- a polynom or any other parametric function
- logic (we know what we can derive/proof)

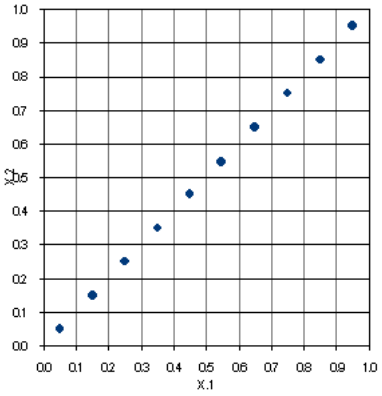
- grammars (information theory, we know what we can derive)
- neural networks (Hopfield, Perceptrons, Neurons)

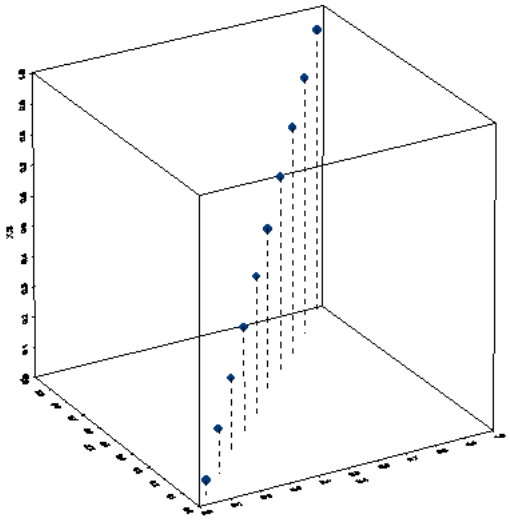
8.18 Curse Of Dimensionality

Bellman's Curse Of Dimensionality means the problem of volume increase in high dimensional spaces. Because of the high range of values single features are like beacons in a storm.

1. Distances converge to zero. The main problem arising from the high dimension is the unintuitive phenomena that every feature is close to every other feature. That is because most of the dimension-values of a feature are 0. Thus the norm of the features are all very similar. Training data would have to be atsronomically huge to cover the whole search space.

Abbildung 55: Curse Of Dimensionality





In the first illustration we see a two dimensional feature space with 10 features. They seem to give an satisfying representation of some class seperation. Now if we take a third feature component, we lift the features space to 3D. Taking the same 10 features (adding the new component) we get the second illustration. Now while we have increased the search space drastically, we have gained no more precision, that we had with the two dimensional featrures, yet the plane seperating the classes in the second illustration is highly underrepresented.

2. Huge search space. Working on and with such a huge search space is really different. Consider for examples search algorithm or local optimization algorithms. Additionally as mentioned above, this search space can never really be covered sufficiently to provide secure data.

Consequences are either to reduce the dimension of the feature space by deleting feature components by e.g. selecting only the n best evaluated features (see

3.2) or by combining them with by Parameter Tying (e.g. PCA 3.1.14 and LDA 3.1.15). Another possibility is to decompose the search space and look at subspaces of it with way lower dimensions and combine the results after the Optimality Principle (see 8.19).

8.19 Principle Of Optimality

Bellman's Principle Of Optimality postulates that in a differentiable optimization procedure, the optimal overall-solution has to be built out of optimal part-solutions. The algortihm of dynamic programming are based on this principle. Applying this principle to solve problems more efficiently, results in the follwoing basic algorithm sceleton:

Algorithm 29 Dynamic Programming (Optimality Principle)

1. Break the problem into smaller subproblems.
2. Solve these problems optimally using this three-step process recursively.
3. Use these optimal solutions to construct an optimal solution for the original problem.

Requirements

1. monotonic cost/distance function

   If s_0 is the start node, s_i the current node and $d(s_0, s_i)$ the distance or costs for the path between them, then $d(s_0, s_{i+1}) \geq d(s_0, s_i)$ must always be true.

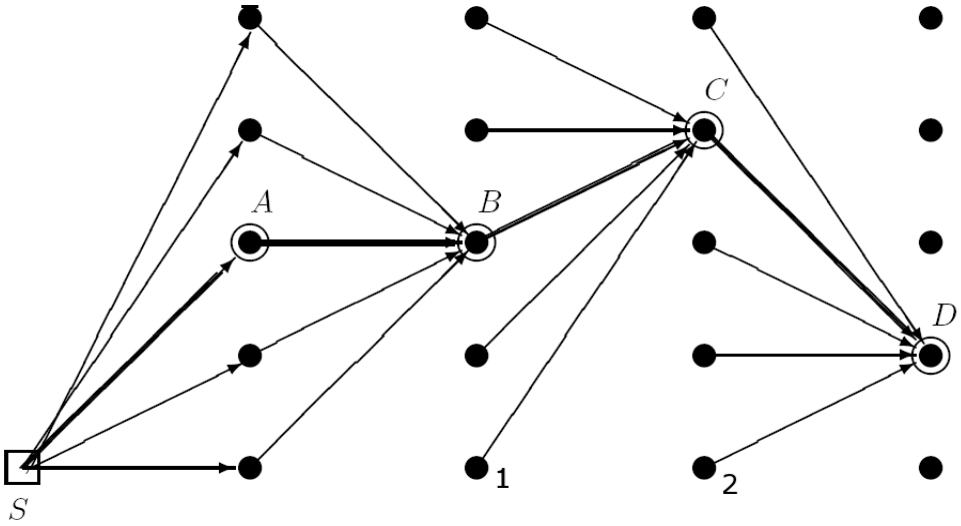
2. seperable problem

   The distance/costs must be seperable (this corresponds to seperating the optimal path into optimal subpaths): $d(s_0, s_j) = d(s_0, s_i) + d(s_i, s_j)$ with $j \geq i$.

8.20 Dynamic Programming

Dynamic Programming is based on the Principle Of Optimality (see above). The idea is to seperate a problem into small and easy to solve subproblems, solve these recursively and construct the solution for the big problem out of them. If the principle of optimality holds, this construction is optimal, if the solutions for the subproblems were optimal. Practically this often means the for solving a problem, we can use the results of the solutions of predecessing subproblems usually stored in a database. This process of memorizing results is called **memoization**. The more the subproblems overlap, the more often they can be used, the more efficient is DP.

Abbildung 56: Dynamic Programming



1. There are n paths to every node. For each node, store the best one of them leading to it (in a lookup table).

2. There are n paths to every node. For each node, store the best one of them leading to it (in a lookup table).

The optimal path from S to D leads over the nodes A, B, C. Having computed the optimal path from S to B, all other non-optimal paths leading to B can be discarded, because according to the principle of optimality, they can not be part of the global optimal path.

Types

There are two common ways to realize this principle:

- **Top-down approach**: The problem is broken into subproblems, and these subproblems are solved and the solutions remembered, in case they need to be solved again. This is recursion and memoization combined together.
- **Bottom-up approach**: All subproblems that might be needed are solved in advance and then used to build up solutions to larger problems. This approach is slightly better in stack space and number of function calls, but it is sometimes not intuitive to figure out all the subproblems needed for solving a given problem.

Requirements

The requirement is the principle of optimality. This is given if we have got a monotonic cost function and if our space is seperable by this function (see 8.19).

Algorithm

Algorithm 30 Dynamic Programming

1. Break the problem into smaller subproblems.
2. Solve these problems optimally using this three-step process recursively.
3. Use these optimal solutions to construct an optimal solution for the original problem.

Example

One of the most popular examples for DP is the computation of the Fibonacci numbers. The Fibonacci numbers are defined:

$$\text{fib}(0) = 0$$
$$\text{fib}(1) = 1$$
$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

If we calculate for example $\text{fib}(5)$, we have to compute

$$\text{fib}(4)+\text{fib}(3) = \text{fib}(3)+\text{fib}(2)+\text{fib}(2)+\text{fib}(1) = \text{fib}(2)+\text{fib}(1)+\text{fib}(1)+\text{fib}(0)+\text{fib}(1)+\text{fib}(0)+\text{fib}(1)$$
$$= \text{fib}(1) + \text{fib}(0) + \text{fib}(1) + \text{fib}(1) + \text{fib}(0) + \text{fib}(1) + \text{fib}(0) + \text{fib}(1)$$

Yet if we use DP to compute it, we still have the solutions of the subproblems: $\text{fib}(3)$ and $\text{fib}(4)$ and can simply add them. Try to implement both variants in any language you like and watch the differences is time, they are enourmous. Here's some Pseudo-Code:

```
fib(n)
  int previousFib = 0, currentFib = 1;
  repeat n - 1 times
    int newFib  = previousFib + currentFib
    previousFib = currentFib
    currentFib  = newFib
  return currentFib
```

8.21 Viterbi Algorithm

The Viterbi Algorithm is a Dynamic Programming algorithm for finding sequences of states.

It makes several assumptions:

Assumptions

- the principle of optimality holds (monotonic cost function, seperable space)
- observed and hidden events are sequences (e.g. over time)
- one observed event must correspond to exactly one hidden event
- an event at time t must only depend on the observable event at that time and the hidden events at times $t-1$ to $t-n$.

According to the principle of optimality several paths can lead to a state t, but only one of them is optimal (if there are more, decide for one at random). This way the algorithm only has to keep track of one path per state. Having a cost function, we keep one number per state: the cummulative cost of the path to this state. Proceeding to a new state $t+1$ there are transition probabilities given, upon which also in respect of the previous path, the best path to the state $t+1$ can be computed. According to DP, proceeding to a new state, optimal paths to predecessing states are saved in some kind of look-up table, because they might be required again for paths to succeeding states.

Algorithm 31 Viterbi Algorithm (from Wikipedia)

```
def forward_viterbi(y, X, sp, tp, ep):
  T = {}
  for state in X:
    ##          prob.      V. path  V. prob.
    T[state] = (sp[state], [state], sp[state])
  for output in y:
    U = {}
    for next_state in X:
      total = 0
      argmax = None
      valmax = 0
      for state in X:
        (prob, v_path, v_prob) = T[state]
         p = ep[state][output] * tp[state][next_state]
         prob *= p
         v_prob *= p
         total += prob
         if v_prob > valmax:
           argmax = v_path + [next_state]
           valmax = v_prob
      U[next_state] = (total, argmax, valmax)
    T = U

  ## apply sum/max to the final states:
  total = 0
  argmax = None
  valmax = 0
  for state in X:
     (prob, v_path, v_prob) = T[state]
     total += prob
     if v_prob > valmax:
       argmax = v_path
       valmax = v_prob
  return (total, argmax, valmax)
```

The function forward_viterbi takes the following arguments:

- y is the sequence of observations
- X is the set of hidden states
- sp is the start probability
- tp are the transition probabilities
- ep are the emission probabilities

The algorithm works on the mappings T and U. Each is a mapping from a state to a triple (prob, v_path, v_prob), where
prob is the total probability of all paths from the start to the current state,
v_path is the Viterbi path up to the current state, and
v_prob is the probability of the Viterbi path up to the current state.
The mapping T holds this information for a given point t in time, and the main loop constructs U, which holds similar information for time t+1. Because of the Markov property[9], information about any point in time prior to t is not needed.

8.22 Hidden Information Principle

observable information = complete information − hidden information

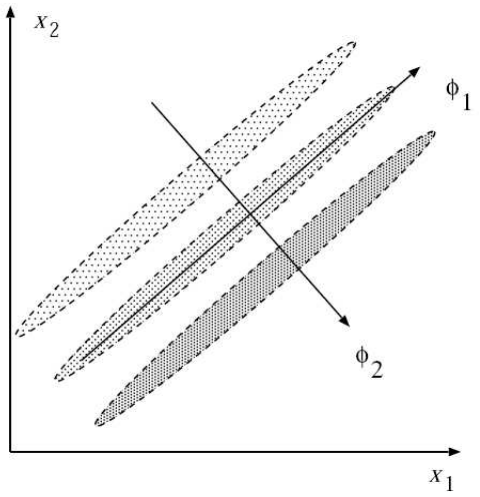
8.23 White Eage On White Background

The white eagle on white background phenomena refers to the fact that the pixels of a picture after applying an operation like filtering often result in color values in a very short range. Imagine a picture of an eagle. After filtering only pixels between $[0; 1]$ remain, yet the picture uses an RGB scale, where 0 is white and 255 is black. That means although the pixels belonging to the eagle will have values close to 1 and the background values close to 0, you will not be able to see anything on the picture.

A common solution is a normalization of the result. For our example a multiplication with 128 would do the trick.

8.24 Adidas Problem

Abbildung 57: The Adidas Problem



Applying PCA features from all three classes will be projected unto the new axis ϕ_1 and thus become indistinguishable. This is because ϕ_1 corresponds to the eigenvector having the biggest eigenvalue.

The **Adidas Problem** occurs with analytical feature optimization methods like PCA (see 3.1.14). The middle stripe corresponds to the eigenvector with the biggest (absolute values) eigenvalue. That means intending an optimal differentiation of all features, the vectors of other tow stripes will be projected unto the middle stripe and distiguishing them is not possible any longer. Because of the three stripes this problem was given the name **Adidas Problem**.

A solution to the Adidas Problem is not to restrict the optimization solely on the spread, like for example the realted LDA does (see 3.1.15).

8.25 Maxims Of Pattern Recognition

1. Maximize the inter class distance.

2. Minimize the intra class distance.

1. means the distance between all features belonging to class Ω_λ to all features belonging to any other class Ω_κ. More illustrative and explanatory is this explanation: Features of different classes should show distinct differences and share as few as possible similarities.
2. means that features within one class should lie as close to each other as possible. More illustrative and explanatory is this explanation: Features of the same class should be as similar to each other as possible and share as many as possible similarities.

9 Mathematics Formulary

9.1 Convolution

The convolution theorem says that:

$$f \oplus g = h \Leftrightarrow FT(f) \cdot FT(g) = FT(h)$$

where FT is the Fourier Transform of a signal. That means a convolution in the spatial domain corresponds to a simple multiplication in the frequency domain. $f \oplus g$ is defined as

$$\int f(u)\, g(x-u)\, du = h(x)$$

Algorithm 32 convolution

1. reflect function g
2. move g across f
3. multiply the peaks of both

Properties Of Convolution Functions

kausal the system response does not occur before the input.

stable $|g_{jk}| < \infty$ There is a threshold.

seperable We can partition a 2D array into 2 1D arrays, if we can seperate them to single factors. The means instead of one 2D Fourier Transfor, we can apply two 1D Fourier Transforms, which is much faster.

9.2 Cross-Ratio

The cross-ratio of a set of four distinct points on the complex plane is given by:

$$(z_1, z_2, z_3, z_4) = \frac{(z_1 - z_3)(z_2 - z_4)}{(z_1 - z_4)(z_2 - z_3)}$$

Historically it was noticed that if four lines in the plane pass through a point P, and a fifth line L not through P crosses them in four points, then the cross-ratio of the directed lengths on L formed by the four points taken in order was independent of L. That is, it is an invariant of the system of four lines.

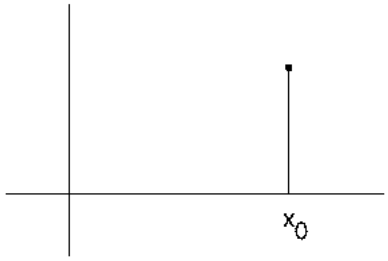
Properties

- Cross-ratios are invariants of projective geometry in the sense that they are preserved by projective transformations.

- The cross-ratio of four complex numbers is real if and only if the four numbers are either collinear or concyclic.

9.3 Dirac-Impuls

Abbildung 58: Dirac Impuls



$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

9.4 Fourier Series

Fourier series can approximate any periodic trigonometric function.

$$f(x) = \frac{a_0}{2} + \sum_{k \geq 1} a_k \cos(k \cdot x) + b_k \sin(k \cdot x)$$

a_k and b_k are called the **Fourier Coefficients**.

An important aspect is that periodic functions that are even can be approximated using only the cosine term $a_k \cos(k \cdot x)$ and odd functions by using only the sine term $b_k \sin(k \cdot x)$. This is because cos is even and sin symmetrical to the origin. The coefficients can be computed by closed formulas:

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\, dx$$

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(k \cdot x)\, dx$$

$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(k \cdot x)\, dx$$

9.5 Fourier Transform

For practical purposes the trigonometric parts of the Fourier Series are rewritten for the **Fourier Transform** with complex numbers in respect to the Euler Formula:

$$e^{i\varphi} = \cos\varphi - i \cdot \sin\varphi$$

Replacing cos and sin by this identity, we can write the Fourier Series as

$$f(x) = \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} c_k e^{(ikx)}$$

as for the non-complex case we can find a closed formula for the coefficients c_k:

$$c_k = \int_{-\pi}^{\pi} f(x)\, e^{(-ikx)} dx$$

If we see c_k as a function $c(k)$(usually written $F(\xi)$), we call this function the Fourier Transform:

$$F(\xi) = \int_{-\pi}^{\pi} f(x)\, e^{(-i\xi x)} dx = FT\{f\}$$

Properties

- requirement: the Fourier Transfor exists if and only if $\int_{-\infty}^{\infty} |f(x)|\, dx < \infty$. This makes sense, because the Fourier Series can only model limited functions (see 1.1).
- convolution theorem: $h(t) = f(t) \oplus g(t) = \int_{-\infty}^{+\infty} f(x)\, g(t-x)\, dx$ corresponds to a simple multiplication in the frequency domain: $H(\xi) = F(\xi)\, G(\xi)$
- scaling: $f(\alpha \cdot x) = \frac{1}{|\alpha|} F\left(\frac{\xi}{\alpha}\right)$
- translation: $f(x - x_0) = e^{(-i\xi x_0)} F(\xi)$
- symmetry: $-\frac{1}{2\pi} \cdot F(x) = f(-\xi)$
- differentiation: $d^n \cdot \frac{f(x)}{d \cdot t^n} = (i\xi)^n F(\xi)$

9.5.1 1D Fourier Transform

$$FT\{f(x)\} = \int_{-\infty}^{\infty} f(x) \cdot e^{-i\xi x} dx = F(\xi)$$

$$FT^{-1}\{F(\xi)\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\xi) \cdot e^{i\xi x} d\xi = f(x)$$

9.5.2 2D Fourier Transform

The Fourier Transform can be easily extended to arbitrary dimensions. Using the Fast Fourier Principle below (see 9.5.4) this computation is usually decomposed into two 1D-Fourier Transforms, since they can be computed much faster.

$$FT\left\{f\left(x,y\right)\right\} = \int\int_{-\infty}^{\infty} f\left(x,y\right) \cdot e^{-i(\xi x+\eta y)} dxdy = F\left(\xi,\eta\right)$$

$$FT^{-1}\left\{F\left(\xi,\eta\right)\right\} = \frac{1}{4\pi^2}\int\int_{-\infty}^{\infty} F\left(\xi,\eta\right) \cdot e^{i(\xi x+\eta y)} d\xi d\eta = f\left(x,y\right)$$

Abbildung 59: Inverse Fourier Transform Proof

$$FT^{-1} \oplus FT\left\{f\left(x\right)\right\} = f\left(x\right)$$

where

$$FT\left\{f\left(x\right)\right\} = F\left(\xi\right)$$

9.5.3 Discrete Fourier Transform

Since all signals we can deal with are discrete, we need to use a discretized form of the continous Fourier Transform. We define our signal as a sequence of samples: $f_0, f_1, \ldots, f_{M-1}$.

1D

$$DFT\left\{f\left(x\right)\right\} = \sum_{j=0}^{M-1} f\left(j\right) \cdot \left(e^{-i\frac{2\pi\xi}{M}}\right)^j = F\left(\xi\right)$$

$$DFT^{-1}\left\{F\left(\xi\right)\right\} = \frac{1}{M}\sum_{\xi=0}^{M-1} F\left(\xi\right)\frac{e^{2\pi i\xi x}}{M} = f\left(x\right)$$

2D

$$DFT\left\{f\left(x,y\right)\right\} = \sum_{j=0}^{M_x-1}\sum_{k=0}^{M_y-1} f\left(j,k\right) \cdot e^{-i2\pi\left(\frac{\xi j}{M_x}+\frac{\eta k}{M_y}\right)} = F\left(\xi,\eta\right)$$

where M is the number of sample values. This transform can be computed by using standard matrix arithmetrics.

The inverse of the DFT is simple, since it is a linear function. Just build the matrix corresponding to the transform and invert it (Pseudoinverse). Both the foward and the inverse are in $O\left(M^2\right)$.

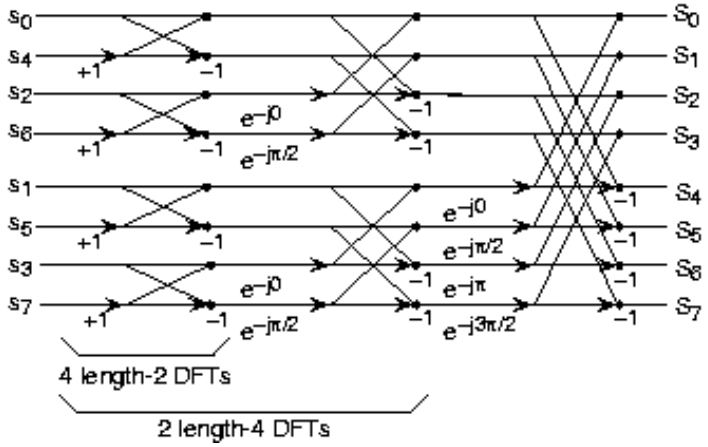
9.5.4 Fast Fourier Transform

We make use of two fundamental principles to reduce the complexity of the Discrete Fourier Transform:

1. homomorphisms (convolution theorem, logartihm)
2. the Divide & Conquer principle: decompose the problem into smaller less complex problems, solve them and reconstruct the solution for the big problem from them (n^{th} root of unity)

The Divide & Conquer principle manifests in the decomposition into even and odd terms and their parallel computation. To achieve this decomposition the Fourier Transform is rewritten using the n^{th}root of unity. By this ongoing decompositions the whole transform resolves into $M = 2 \cdot n$ simple addition and subtractions of terms (see illustration below). These n terms resulting on the different levels (see illustration below) are then multiplied with each other to construct the complete solution given by a final apllication of Fourier Transforms of order n.

Abbildung 60: Fast Fourier Transform



With this decomposition Fast Fourier Transform is located in $O(M \cdot \log M)$.

Example

Having $M = 2^{10}$ samples results in 2^{20} complex multiplications, applying the Discrete Fourier Transform. Applying FFT we reduce this number to $\frac{2^{10}}{2 \cdot 10}$ multiplications.

9.5.5 Short Term Fourier Transform

$$SFT\{f(t)\} = \int_{-\infty}^{\infty} f(t) \cdot \omega \oplus (t - \tau)\, e^{-2\pi i \omega t} dt = SFT(\tau, \omega)$$

where ω is a window function and $(t-\tau)$ a translation on the time axis.

9.5.6 Wavelet Transform

$$WT\left\{f\left(t\right)\right\} = \frac{1}{\sqrt{\alpha}} \int_{-\infty}^{\infty} f\left(t\right) \cdot \psi \oplus \left(\frac{t-\tau}{\alpha}\right) dt = WT\left(\tau, \alpha\right)$$

where ψ is the mother wavelet, α a scaling for the wavelet and $(t-\tau)$ a translation on the time axis.

Both ψ and ω act a a filter convoluted with f.

9.6 The sinc Function

The sinc function is defined as

$$\operatorname{sinc}\left(x\right) = \frac{\sin x}{x}$$

The normalized form is

$$\operatorname{sinc}\left(x\right) = \frac{\sin \pi x}{\pi x}$$

Properties (normalized version)

- $\operatorname{sinc}\left(0\right) = 1$
- $\operatorname{sinc}\left(k\right) = 0$ for all $k \in \mathbb{Z}, k \neq 0$
- $x_k\left(t\right) = \operatorname{sinc}\left(t-k\right)$ build an othonormal basis for bandlimited functions in $L^2\left(\mathbb{R}\right)$
- $\int_{-\infty}^{\infty} \operatorname{sinc}\left(t\right) \exp^{-2\pi i f t}\, dt = \operatorname{rect}\left(f\right)$
- $\int_{-\infty}^{\infty} \operatorname{sinc}\left(x\right) = 1$
- $\int_{-\infty}^{\infty} \left|\operatorname{sinc}\left(x\right)\right| = \infty$

9.7 Special Matrices

adjoint matrix

$A_{i,j}^{\sharp} \in \mathbb{R}^{(n-1)\times(n-1)}$ is the adjoint matrix of $A \in \mathbb{R}^{n\times n}$. $A_{i,j}^{\sharp}$ corresponds to A where the i^{th} row and the j^{th} column have been left out.

complementary matrix

$A^{\complement} \in \mathbb{R}^{n\times n}$ is the complementary matrix of $A \in \mathbb{R}^{n\times n}$. The entries of the complementary matrix $a_{i,j}^{\complement}$ correspond to the determinant of the A corresponding adjoint matrix $A_{i,j}^{\sharp}$, where the i^{th} row and the j^{th} column have been left out:

$$a_{i,j}^{\complement} = \det\left(A_{i,j}^{\sharp}\right)$$

pseudo inverse

The Inverse of a matrix A does only exist, if A is regular. If it isn't and we need to get an Inverse of it anyway, we can compute the Pseudo-Inverse A^t:

$$A^t = \left(A^T \cdot A\right)^{-1} A^T$$

having the property

$$A \cdot A^t \cdot A = A$$

respectively

$$A^t \cdot A \cdot A^t = A^t$$

9.8 Matrix Properties 1

- $\left(A^{-1}\right)^T = \left(A^T\right)^{-1}$ for symmetric matrices
- $(AB)^T = A^T B^T$
- the previous two combined: $\left(\Sigma^{-1}\right)^T = \left(\Sigma^T\right)^{-1} = \Sigma^{-1}$ since Σ is always symmetric.

9.9 Matrix Properties 2

- $\vec{x}^T A \cdot \vec{x} = \text{trace}\left(A \cdot \vec{x} \cdot \vec{x}^T\right)$
- $\text{trace}\,(A) = \text{trace}\left(A^T\right)$

where the **trace** of a matrix is simply the sum over all its diagonal entries.

9.10 Banach's Fixed Point Theorem

$$|F(x_1) - F(x_2)| \leq L(x_1 - x_2)$$

where $L \in]0;1[$ is called Lipschitz constant. If such a constant exists any arbitrary iterative series starting with x_0 will eventually converge to a fixed point x^*.

Example

given: $h(x) = \arctan(x+1)$ and accordingly the first derivation $h'(x) = \frac{1}{1+(x+1)^2}$. Looking at the limes we find that $L = \frac{1}{2}$ will be a suitable limit (since $\frac{1}{1+(x+1)^2} \leq \frac{1}{2}$).

Now we calculate

$$|x_n - \bar{x}| \leq \frac{L^n}{1-L} |x_1 - x_0|$$

choosing $x_0 = 1$ for a simple start value.

$$|x_n - \bar{x}| \leq \frac{0.5^n}{1-0.5} |1.1071 - 1|$$

we accept an error estimation of 0.01 and estimate

$$< 0.5^n \cdot 0.3$$

applying ln

$$n > \frac{\ln\left(\frac{0.01}{0.3}\right)}{\ln\left(\frac{1}{2}\right)} = 4.91$$

We have got our result. We reach a fixed point after $n = 5$ steps.

9.11 Gram-Schmidt Orthogonalization

With the Gram-Schmidt Orthogonalization we are able to find, given a linear independent set of vectors, a orthogonal system spanning the same subspace.

Algorithm 33 Gram-Schmidt Orthogonalization Process

---

given a set of linear independet vectors $v_1 \dots v_n$.

$u_1 \dots u_n$ will be vectors of an orthogonal system.

- $u_1 = v_1$
- $u_2 = v_2 - \frac{\langle v_2, u_1 \rangle}{\langle u_1, u_1 \rangle} \cdot u_1$
- $u_3 = v_3 - \frac{\langle v_3, u_1 \rangle}{\langle u_1, u_1 \rangle} \cdot u_1 - \frac{\langle v_3, u_2 \rangle}{\langle u_2, u_2 \rangle} \cdot u_2$
- $\dots$
- $u_n = v_n - \sum_{i=1}^{n-1} \frac{\langle v_n, u_i \rangle}{\langle u_i, u_i \rangle} \cdot u_i$

Example

We are in the $\mathbb{R}^3$ and use the standard scalar product $\langle \cdot, \cdot \rangle$.

We have given the linear idependent vectors $v_1 = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}$ and $v_2 = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$.

$$u_1 = v_1 = \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix}$$

$$u_2 = v_2 - \frac{\langle v_2, u_1 \rangle}{\langle u_1, u_1 \rangle} \cdot u_1 = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} - \frac{12}{14} \cdot \begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} = \frac{2}{7} \begin{pmatrix} -2 \\ 4 \\ 1 \end{pmatrix}$$

Theory

Having determined $u_1 \dots u_{k-1}$, we try to subtract a suitbale linear combination of v_k, such that $u_k = v_k - \sum_{i=1}^{k-1} \lambda_i u_i$ is orthogonal to each vector $u_1 \dots u_{k-1}$. I.e.

$$\forall_j \langle u_k, u_j \rangle = \langle v_k, u_j \rangle - \sum_{i=1}^{k-1} \lambda_i \langle u_i, u_j \rangle = 0$$

This is done by choosing λ as $\lambda = \frac{\langle v_k, u_i \rangle}{\langle u_i, u_j \rangle}$, inserting that draws:

$$\langle v_k, u_j \rangle - \langle v_k, u_j \rangle = 0$$

9.12 Gram-Schmidt Orthonormalization

Using Gram-Schmidt's Orthonormalization we can find a set of orthonormal vectors that create the same subsapce as a set of linear independent vectors.

Algorithm 34 Gram-Schmidt Orthonormalization Process

---

given a set of linear independet vectors $v_1 \dots v_n$.

$u_1 \dots u_n$ will be vectors of an orthogonal system.

- $u_1 = \frac{v1}{\|v1\|}$
- $u_2' = v_2 - \langle v_2, u_1 \rangle \cdot u_1$ (orthogonize)
  $u_2 = \frac{u_2'}{\|u_2'\|}$ (normalize)
- $u_3 = v_3 - \langle v_3, u_1 \rangle \cdot u_1 - \langle v_3, u_2 \rangle \cdot u_2$ (orthogonize)
  $u_3 = \frac{u_3'}{\|u_3'\|}$ (normalize)
- $\dots$
- $u_n' = v_n - \sum_{i=1}^{n-1} \langle v_n, u_i \rangle \cdot u_i$
  $u_n = \frac{u_n'}{\|u_n'\|}$

Example

given $v_1 = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$ and $v_2 = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$.

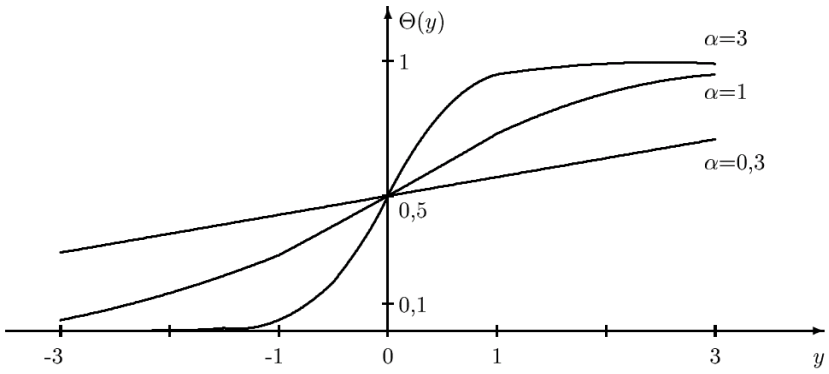
$$u_1 = \frac{v_1}{\|v_1\|} = \frac{1}{\sqrt{10}} \cdot \begin{pmatrix} 3 \\ 1 \end{pmatrix}$$

$$u_2' = v_2 - \langle v_2, u_1 \rangle \cdot u_1 = \begin{pmatrix} 2 \\ 2 \end{pmatrix} - \frac{1}{\sqrt{10}} \cdot \left\langle \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 2 \end{pmatrix} \right\rangle \cdot \frac{1}{\sqrt{10}} \cdot \begin{pmatrix} 3 \\ 1 \end{pmatrix} = \frac{1}{5} \begin{pmatrix} -2 \\ 6 \end{pmatrix}$$

$$u_2 = \frac{u_2'}{\|u_2'\|} = \frac{1}{\sqrt{\frac{40}{25}}} \cdot \frac{1}{5} \begin{pmatrix} -2 \\ 6 \end{pmatrix} = \frac{1}{\sqrt{10}} \cdot \begin{pmatrix} -1 \\ 3 \end{pmatrix}$$

9.13 Sigmoid Θ

Abbildung 61: Sigmoid Function



The Sigmoid function for three different values of α

with y as an input and α as weights, the Sigmoid is defined as

$$\Theta(y) = \frac{1}{1 + \exp^{-\alpha y}}$$

9.14 Hyperplane

A hyperplane in general is a subspace of space, having one dimension less than this space. A hyperplane p can be defined as:

$$h(\vec{x}) = \vec{x}^T \vec{\alpha} + \alpha_0$$

All vectors $\vec{x}$ for which $h(\vec{x}) = 0$ lie on this hyperplane. In fact we have the following properties:

normal vector $n = \frac{-\vec{\alpha}}{\sqrt{\vec{\alpha}^T \vec{\alpha}}} = \frac{-\vec{\alpha}}{|\vec{\alpha}|}$

distance from origin $s_0 = \frac{-\alpha_0}{|\vec{\alpha}|}$

distance from point $\vec{x}$ $s_{\vec{x}} = \frac{-\left(\vec{\alpha}^T \vec{x} + \alpha_0\right)}{|\vec{\alpha}|}$

10 Statistics

10.1 Statistical Terms

expectation $E\{g(x)\}$

$$E\{g(x)\} = \int g(x)\, p(x)\, dx \simeq \frac{1}{N} \sum_{i=1}^{N} g(x_i)$$

a-priori $p(\Omega_i)$

Prior probability that class Ω_i occurs. Usually derived from backgorund knowledge or simple counting.

If nothing is known assume $p(\Omega_i) = \frac{1}{N}$, where N is the number of classes. If there exists a training set, assume $p(\Omega_i) = \frac{\text{number of } (\vec{c}_j, \Omega_i)}{\text{number of } \vec{c}_j}$: The number of training features associated to class Ω_i divided by the number of training features.

a-priori error: $|x_n - \bar{x}| \leq \frac{L^n}{1-L} |x_1 - x_0|$ where L is the Lipschitz constant from Banach's fixed point theorem (see 9.10).

a-posteriori $p(\Omega_\lambda \mid \vec{c})$

Posterior probability for class Ω_λ with the feature $\vec{c}$. A-Posteriori means the probability after n-steps.

This probability is hard to get or calculate directly. Therefore we apply Bayes' Rule:

$$p(\Omega_\kappa \mid \vec{c}) = \frac{p(\Omega_\kappa) \cdot p(\vec{c} \mid \Omega_\lambda)}{p(\vec{c})} = \frac{p(\Omega_\kappa) \cdot p(\vec{c} \mid \Omega_\lambda)}{\sum_\lambda^\kappa p(\Omega_\lambda) \cdot p(\vec{c} \mid \Omega_\lambda)}$$

a-posteriori error: $|x_n - \bar{x}| \leq \frac{L}{1-L} |x_n - x_{n-1}|$ where L is the Lipschitz constant from Banach's fixed point theorem (see 9.10).

probability density function (pdf) $p(\vec{c} \mid \Omega_\lambda)$

The probability denisty function describes the distribution of the features in a class. If we assume they are normally distributed, we can take the gaussian distribution:

$$p(\vec{c} \mid \Omega_\lambda) = \mathcal{N}(\vec{c}, \vec{\mu}_\lambda, \Sigma_\lambda)$$

mean $\vec{\mu}_\kappa$

the mean value of the features of class Ω_κ.

$$\vec{\mu}_\kappa = \frac{1}{N} \sum_{i=1}^{N} \vec{c}_i$$

where N denotes the number of features in class N.

The continous form is

$$\mu = \int_{-\infty}^{\infty} x \cdot p(x)\, dx$$

covariance matrix Σ_κ

the covariance matrix describes the variance of the features within class Ω_κ. It has the following properties:

- quadratic
- symmetric
- positive definit

$$\Sigma_\kappa = \frac{1}{N} \sum_{i=1}^{N} \left(\overrightarrow{c_i} - \overrightarrow{\mu_i}\right) \left(\overrightarrow{c_i} - \overrightarrow{\mu_i}\right)^T$$

In case of one dimensional features, the covariance is also a one dimensional parameter σ_κ.

$$\sigma_\kappa = \frac{\sum_{i=1}^{N} (c_i - \mu_i)^2}{N}$$

variance σ_κ

the variance is the one dimensional form of the covariance matrix. It describes the degree of deviation from the mean value.

$$\sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 \cdot p(x)\, dx$$

Talking in terms of variance, a multi dimensional covariance matrix would look like this:

$$\Sigma = \begin{pmatrix} \sigma_1 & \cdots & \ddots \\ \vdots & \sigma_2 & \vdots \\ \ddots & \cdots & \sigma_3 \end{pmatrix}$$

if all σ_i are independent of each other (for $i = 1, 2, 3$), then the covariance looks like this

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix}$$

kernel matrix

The kernel matrix Q is gained by reducing objectives S_i corresponding to the maxims of pattern recognition (see 8.25) to an eigenvector, eigenvalue problem of a matrix Q. For S_1 this so called **kernel matrix** looks like this:

$$Q = \frac{1}{N^2} \cdot \sum_{i=1}^{N} \sum_{j=1}^{N} \left(\vec{f}_i - \vec{f}_j \right)^T \left(\vec{f}_i - \vec{f}_j \right)$$

where N is the dimension of the features/signals f. The part after the sums is called **Measurement Matrix**.

measurement matrix

$$\mathrm{ME}_{ij} = \left(\vec{f}_i \cdot \vec{f}_j \right)^T \left(\vec{f}_i \cdot \vec{f}_j \right)$$

10.2 Distributions

normal distribution $\mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa\right)$

Normal distribution is the first distribution to assume, if there is no information about the actual distribution of features.

Multidimensional

$$\mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa\right) = \frac{1}{\sqrt{\det\left(2\pi\Sigma\right)}} \exp^{-\frac{1}{2}(\vec{c}-\vec{\mu}_\kappa)^T \Sigma_\kappa^{-1} (\vec{c}-\vec{\mu}_\kappa)}$$

Onedimensional

$$\mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \sigma\right) = \frac{1}{\sqrt{2 \cdot \pi\sigma}} e^{-\frac{1}{2}\left(\frac{\vec{c}-\vec{\mu}}{\sigma}\right)^2}$$

10.3 Properties

statistical independence

The general defintion of statistical independence is:

$$P\left(A \cap B\right) = P\left(A\right) \cdot P\left(B\right)$$

translated into common pattern recognition that means

$$p\left(\vec{c} \mid \Omega_\kappa\right) = \prod_{\nu=1}^{n} p\left(c_\nu \mid \Omega_\kappa\right)$$

In some cases partial statistical independence is useful. In this case some parameters are statistical independent, as above, and others ain't.

statistical properties

- $p(A, B) = p(A \cap B) = p(A) \cdot p(B \mid A)$
- $p(w_1 \dots w_n) = p(w_1) \cdot p(w_2 \dots w_n \mid w_1) = p(w_1) \cdot p(w_2)(w_3 \dots w_n \mid w_1 w_2) = \dots = \prod_{i=1}^{n-1} p(w_i)\, p(w_1 \dots w_{n-1})$

Bayes' Theorem

$$p(A \mid B) = \frac{p(B \mid A) \cdot p(A)}{p(B)}$$

convex combination

a sum of probabilities, that sum up to 1:

$$p(\vec{c} \mid \Omega_k) = \sum_{l}^{M} p_l \mathcal{N}(\vec{c}_i; \vec{\mu}_l, \Sigma_l)$$

or with any other pdf.

stochastic criterion

probailities have to sum up to 1.

10.4 Proof Of A Distribution

Often you will get away just using gaussian distribution, sometimes however is is simply not appropriate. If you have some background knwoledge about your application, you can do the following, to 'proove' that the distribution you assumed holds:

- good classification rates
- statistical testing
- by construction (e.g. see 5)

10.5 Estimating A Distribution

If there is nothing known of the distribution we have to estimate it. Some options are:

- assume gaussian distribution $p(\vec{c} \mid \Omega_\kappa) = \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$ and try if get reasonable results with it
- assume statistical independency (leads to diagonal covariance matrices)
- assume some other parametric family $p(\vec{c} \mid \theta)$ and estimate θ using ML or MAP (see 8.1, 8.2)

- try to approximate the distribution by a convex combination of gaussians (see 4.2.3)
- try a non-parametric estimation of the density by e.g. relative freuqencies

10.6 Moments

Moment is a mathematical term evolved from physics. Moments of functions exist in several levels, each with unique properties. The definition of a moment of a real values function $f(x, y)$ is

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y)\, dxdy$$

The one dimensional form is accordingly

$$m_p = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p f(x)\, dx$$

The first moment about zero is the expectation

$$m_1 = E(x)$$

and the second

$$m_2 = E\left(x^2\right) = \mathrm{Var}(x) + (E(x))^2$$

Absolut Moment

absolut moments are focusing on a point r:

$$M_k(r) = E\left(|x - r|^k\right)$$

this point is simply 0 for the standard moments ("about zero") above.

Central Moments

central moments are absolute moments, where r is the expectation $E(x)$. central moments of degree k are defined as:

$$\mu_k = E\left((x - m_1)^k\right)$$

first central moment

$$\mu_1 = 0$$

second central moment the second central moment corresponds to the variance:

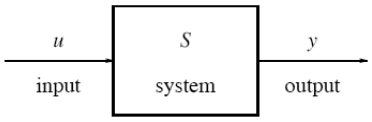
$$\mu_2 = E\left((x - m_1)^2\right)$$

third central moment the third central moment corresponds to the skewness (german: "Schiefe") $\gamma \cdot \sigma^3$

foruth central moment the fourth central moment corresponds to the kurtosis (german: "Wölbung")

11 List Of Terms

dynamic system a system S where the same input u at two different time spots t_1, t_2 leads to different outputs $y_1 \neq y_2$ (contrary to static system 11). So to speak a system that evolves over time.



entropy $\frac{\sum \log p(x)}{s}$ where s is a scaling. The entropy measures how much information there is in an event. In statistics this corresponds to the degree of uncertainty. If all random events are equally probable, the entropy is maximal.

$$H(x) = \sum_{i=1}^{n} p(i) \log_2 \left(\frac{1}{p(i)} \right) = -\sum_{i=1}^{n} p(i) \log_2 p(i)$$

where x is a random event and i sums over all possible outcomes of it.

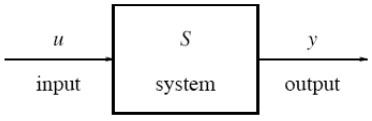
information $\log p(x)$ where $p(x)$ is the probability for x to occur

invariant feature features that do not change, if the object they describe is transformed (e.g. rotation or translation). They are then invariant to these specific transformations.

mutual information see transinformation: 11.

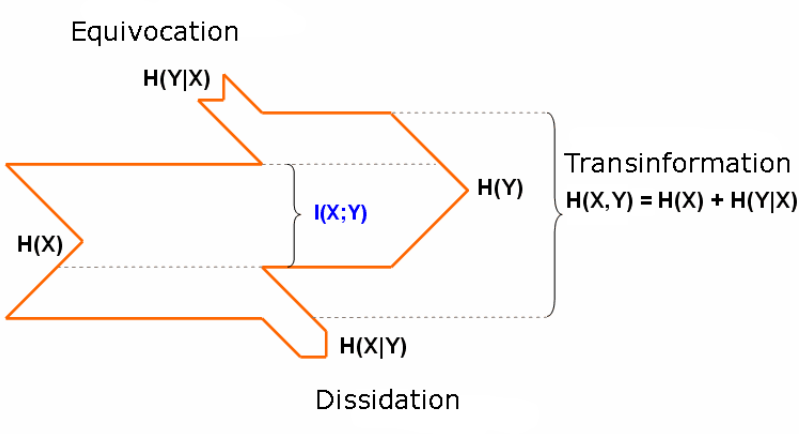
overfitting a term used for models that are too strongly fitted to a training set. That means one additional point in the training set leads to a totally different model. Example: Imagine a polygon of degree 10 and 10 training features. The resulting decission boundary would clearly be overfitted if the polygon would cross every one of those training features (high frequency).

static system a system S where the same input u at two different times t_1, t_2 leads to the same output $y_1 = y_2$ (contrary to dynamic system 11).



transinformation the transinformation, or mutual information, of two random variables is a quantity that measures the mutual dependence of the two variables:

Abbildung 62: Equivocation



As you can see the term reffered to as Equivocation enters the signal, while the Dissidation gets lost. The Transinfromation denoted by $I(X;Y)$ is the amount of original information that is contained in the final signal.

It is closely related to the concept of entropy (see 11).

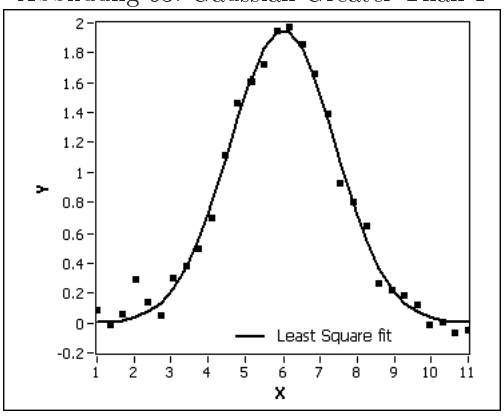
variant feature features that change, if the object they describe is transformed (e.g. rotation or translation). They are then variant to these specific transformations.

12 Miscellaneous / FAQ

Is there a gaussian distribution resp. arbitrary probability density function with a probability value greater than 1?

Yes of course. Only the area has to sum up to 1. Imagine a really tight gaussian bell that goes far beyond 1 on the y-scale, but still returns 1 integrating over the area.

Abbildung 63: Gaussian Greater Than 1



This figure illustrates the **gaussian fit** method with least square approximation:

$$f = \gamma \cdot e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

where γ is the amplitude, μ the mean and σ the variance. The idea of gaussian fit is, to find γ, μ, σ which fit the observation X best.

Is there a logical connection between distances and densities?

Yes. A gaussian distribution where all covrainces equal idenitiy matrices, the pdf is gaussian and the prior probabilities can be neglected corresponds to an euclidean distance. If the mean vectors of the gaussian distribution furthermore correspond to the sample vectors of the NN classifier, the gaussian density equals the NN classifier.

We assume gaussian distribution:

$$p\left(\Omega_\kappa \mid c\right) = p\left(\Omega_\kappa\right) \mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa\right)$$

Additionaly we assume an uniform distribution of the classes, that kicks the prior probabilities and we get:

$$p\left(\Omega_\kappa \mid c\right) = \mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa\right)$$

Now we maximize the log likelihood, like the Baysean classifier:

$$\text{argmax}_\kappa \mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa\right) = \log \text{argmax}_\kappa \mathcal{N}\left(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa\right)$$

maximizing this term corresponds to a minimizing of the exponent:

$$\text{argmax}_\kappa \frac{1}{\sqrt{2\pi}\Sigma_\kappa} \cdot \exp^{-\frac{1}{2}\left((\vec{c}-\vec{\mu_\kappa})^T \Sigma_\kappa^{-1} (\vec{c}-\vec{\mu_\kappa})\right)} = \text{argmin}_\kappa \left(\vec{c} - \vec{\mu_\kappa}\right)^T \Sigma_\kappa^{-1} \left(\vec{c} - \vec{\mu}_\kappa\right)$$

Now if:

- the pdf is gaussian $p(c \mid \Omega_\kappa) = \mathcal{N}(\vec{c}, \vec{\mu}_\kappa, \Sigma_\kappa)$
- the classes are uniformly distributed $\forall \kappa : p(\Omega_\kappa) = p(\Omega_\lambda)$ (in this case they contribute nothing and can be neglected)
- the covariances equal the identity matrix $\Sigma_\kappa = 1$

the optimization above is reduced to the minimization of $\text{argmin}_\kappa (\vec{c} - \vec{\mu}_\kappa)^T (\vec{c} - \vec{\mu}_\kappa)$ which is in fact the euclidean distance.

When is the Baysean Classifer equal to the NN classifier?

see 12.

When is LDA equal with the Baysean Classifier?

- observations are multivariate gaussians in each class
- all covariance matrices are equal

What is the disadvantage of the euclidean distance?

the euclidean distance stretches or contracts values:

- small distances (< 1) become even smaller
- big distances (> 1) become even bigger

Many maximization resp. minimization procedures are local, how to find the global maximum/minimum anyway?

An intutive easy method is to sample the whole signal on equidistant points as a first step. And resample areas that show a great change in value with a smaller sample step. The local optimizer's initial value is then choosen from the vicinity of the area showing the greatest change in value.

What is the difference between Bayes Estimation and Estimators like ML, MAP or EM

In Bayes a *discrete class number* is estimated, while in ML, MAP and EM *continous parameter values* are the goal.

When are ML-Estimation and MAP-Estimation equal?

The difference between both is, that MAP-Estimation uses prior knowledge. If we have a uniform distribution, each value is equally probable and prior knowledge becomes useless. This is exactly the scenario when MAP becomes equal to ML.

When is the Polynomial Classifer equal to Bayes?

If there are no restrictions for the used distance metric function, like number of parameters or a parameter range, then the Polyinomial Classifier equals the Baysean one (for a proof see 4.3.1).

What samples do I choose for training & testing?

The best way is either to pick them at random or, if background knowledge allows for such a distinction, pick the most difficult ones. Watch out for the possibility of overfitting in the last case (see 11).

Apart from that there is one golden rule about picking samples:

Never take the same samples for training and testing. Always keep those two sets distinct.

Problems With Goats / The Quiz Show

Dear candidate you have given 3 doors. Behind one of them a good mark in the pattern recognition exam is waiting for you. Behind the other two are free meals at the local mensa.

The prior probability is clear. The prior probability that the mark hides behind door 1 is $\frac{1}{3}$, the same counts for door 2 and 3:

$$\forall i : p\left(\text{door}_i\right) = \frac{1}{3}$$

You tell the Quizmaster you want door 1. The Quizmaster opens door 2, and a slimy something appears. You guess this is one of the mensa meals. He gives you another chance, do you want to keep door 1 or rather take door 3

Applying statistics, which door gives you the highest probability to a good mark in the exam?

We use the following notation:
M_1 denotes the event: Good mark behind door 1
Q_1 denotes the event: Quizmaster opens door 1

Our goal is to find our the probability $p\left(M_3 \mid Q_2\right)$ and/or $p\left(M_1 \mid Q_2\right)$

$$p\left(M_3 \mid Q_2\right) = \frac{p\left(Q_2 \mid M_3\right) \cdot p\left(M_3\right)}{p\left(Q_2\right)}$$

$$= \frac{1 \cdot \frac{1}{3}}{p\left(Q_2 \mid M_1\right) \cdot p\left(M_1\right) + p\left(Q_2 \mid M_2\right) \cdot p\left(M_2\right) + p\left(Q_2 \mid M_3\right) \cdot p\left(M_3\right)}$$

$$= \frac{\frac{1}{3}}{\frac{1}{2} \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} + 1 \cdot \frac{1}{3}}$$

$$= \frac{2}{3}$$

So your chances are much higher taking door 3. ;-)

German Terms That Are Yet To Translate

- Vektorwertig
- Prüfgröße
- Verwechslungswahrscheinlichkeit (confusion probability?)
- Trennfunktion

13 Bibliography

The Elements Of Statistical Learning (Trevor Hastie, Robert Tibshirani, Jerome Friedman)

14 Appendix

List of Algorithms

Abbildungsverzeichnis

Tabellenverzeichnis

Index