Pattern Recognition Summary

Summary

im Fachgebiet Informatik

vorgelegt von: Christopher Syben

Studienbereich: Informatik

This is a Summery of Patter Recognition. Some Parts are copied from the existing Summary. There are a lot of gramma mistakes but the Theory should be correct in the most cases. If you are not sure use the Videos ':)'

Inhaltsverzeichnis

1 Pattern Recognition Introduction

1.1 Good too remember

1.1.1 Definitions

$\vec{x} \in \mathbb{R}^d$:	d-dimensional feature vector the single components of $\vec{x}$ are continuous random variables
y :	Class Number (for example: $y \in \{0, 1\}$ or $y \in \{-1, +1\}$ In the Classification case y is a discrete random variable In the Regression case y is a continuous random variable
$p(y)$:	prior probability of pattern class y | Priors depend not on a current Observation
$p(\vec{x})$:	evidence. The probability that we observe a certain feature vector $\vec{x}$ independent on his class assignment
$p(\vec{x}, y)$:	joint probability density function (pdf). More mathematical is $p(\vec{x} \cap y)$ the joint probability of having a feature Vector $\vec{x}$ and the feature vector belongs to class y
$p(\vec{x}\|y)$:	class conditional density the probability that we observe $\vec{x}$ given a certain class y if you observe feature of a selected class y what is the pdf of the features belonging to this class
$p(y\|\vec{x})$:	posterior probability. the probability of class y given a certain feature vector $\vec{x}$

1.1.2 Bayes Rule

$$p(\vec{x}, y) = p(y) \cdot p(\vec{x}|y) = p(\vec{x}) \cdot p(y|\vec{x}) \tag{1.1}$$

an importan formulation is:

$$p(y|\vec{x}) = \frac{p(y) \cdot p(\vec{x}|y)}{p(\vec{x})} \tag{1.2}$$

and also very important is:

$$p(y|\vec{x}) = \frac{p(y) \cdot p(\vec{x}|y)}{\sum\limits_{y'} p(y') \cdot p(\vec{x}|y')} \tag{1.3}$$

this is done with the concept of marginalisation:

$$p(\vec{x}) = \sum_{y} p(y) \cdot p(\vec{x}|y) \tag{1.4}$$

1.1.3 Density Function

Gaussian Probability Density Function:

$$\mathcal{N}(\vec{x}; \vec{\mu}, \Sigma) = \frac{1}{\sqrt{det(2\pi\Sigma_y)}} \cdot e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})} \tag{1.5}$$

- you can compute the probability of an Interval $[a, b]$ with the Integral $\int\limits_a^b$
- the probability of a singe value is zero: $p(a) = 0$
- the Integral $\int\limits_{-\infty}^{\infty}$ over the density Function is $= 1$

1.1.4 ML-Estimations for Covariance matrices

in general:

$$\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T \tag{1.6}$$

for a specific class:

$$\hat{\Sigma}_1 = \frac{1}{\#\{y_i = 1\}} \sum_{\substack{i=1\\ y_i=1}}^{N} (\vec{x}_i - \vec{\mu}_1)(\vec{x}_i - \vec{\mu}_1)^T \tag{1.7}$$

the joint covariance matrix:

$$\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\vec{x}_i - \vec{\mu}_{y_i})(\vec{x}_i - \vec{\mu}_{y_i})^T \tag{1.8}$$

1.1.5 Other interesting stuff

- the Hessian matrix is symmetric !

2 Bayesian Classifier

The Bayesian classifier decide for the class y that maximizes the posterior probability
The discriminative model is:

$$y^* = \delta(\vec{x}) = \arg\max_y p(y|\vec{x}) \tag{2.1}$$

using the Bayesian Rule you come up with the generative Model:

$$y^* = \arg\max_y \frac{p(y) \cdot p(\vec{x}|y)}{p(\vec{x})} \tag{2.2}$$

the Probability of $p(\vec{x})$ is independent of y.
To find the position of the maximum its enough to maximize the nominator !

$$y^* = \arg\max_y p(y) \cdot p(\vec{x}|y) \tag{2.3}$$

Why is $p(\vec{x}|y)$ easier to compute / model than $p(y|\vec{x})$?
because y is a discrete variable. For each class you have to setup a density function. In a 2-Class problem that are only 2 pdf's.
$\vec{x}$ is a continuous Variable, in this case you have to setup an unlimited Number of density functions !

2.1 Optimality of the Bayesian Classifier

The Bayesian Classifier is optimal with respect to the 0/1-Loss function.
This means if you assign a feature Vector $\vec{x}$ to the correct Class y then you pay 0.
If you assign the feature Vector $\vec{x}$ to the wrong Class y you pay 1.

3 Logistic Regression

3.1 logistic Function | Sigmoid Function

$$g(x) = \frac{1}{1 + e^{-x}}, \qquad x \in \mathbb{R} \tag{3.1}$$

a nice Property of the logistic Function is:

$$g'(x) = g(x) \cdot (1 - g(x)) \tag{3.2}$$

3.2 logistic regression

- discriminative model

logistic regression means you can compute the posterior probability directly:

$$p(y = 0|\vec{x}) = \frac{1}{1 + e^{-F(x)}} \qquad p(y = 1|\vec{x}) = \frac{1}{1 + e^{F(x)}} \tag{3.3}$$

$F(\vec{x}) = 0$ is the decision boundary. A point on the decision boundary full fill following equation:

$$\log \frac{p(y = 0|\vec{x})}{p(y = 1|\vec{x})} = 0 \tag{3.4}$$

If the covariance matrices are not identical, the optimal decision boundary will be a quadratic one:

$$F(\vec{x}) = \vec{x}^T \mathbf{A} \vec{x} + \vec{\alpha}^T \vec{x} + \alpha_0 \tag{3.5}$$

3.2.1 Example with normally distributed classes

assume:

$$p(\vec{x}|y) = \mathcal{N}(\vec{x}; \vec{\mu}, \Sigma) \tag{3.6}$$

and $y \in \{0, 1\}$ then we will get as the decision Boundary the following:

$$\mathbf{A} = \frac{1}{2} \cdot (\Sigma_1^{-1} - \Sigma_0^{-1}) \tag{3.7}$$

$$\vec{\alpha} = \vec{\mu}_0^T \Sigma_0^{-1} - \vec{\mu}_1^T \Sigma_1^{-1} \tag{3.8}$$

$$\alpha_0 = log\frac{p(y=0)}{p(y=1)} + \frac{1}{2}(log\frac{det(2\pi\Sigma_1)}{det(2\pi\Sigma_0)} + \vec{\mu}_1^T \Sigma_1^{-1}\vec{\mu}_1 - \vec{\mu}_0^T \Sigma_0^{-1}\vec{\mu}_0) \tag{3.9}$$

α_0 is a constant offset of the decision boundary.
If a Class get a higher prior probability the decision boundary will move away from that class.
For example, if $p(y_0) = 0.8$ then the decision boundary will shift closer to the class y_1

if both classes share the same covariances $\Sigma = \Sigma_0 = \Sigma_1$, the optimal desicion boundary will be a linear one:

$$\mathbf{A} = 0 \tag{3.10}$$

$$\vec{\alpha} = (\vec{\mu}_0 - \vec{\mu}_1)^T \Sigma^{-1} \tag{3.11}$$

$$\alpha_0 = log\frac{p(y=0)}{p(y=1)} + \frac{1}{2}((\vec{\mu}_1 + \vec{\mu}_0)^T \Sigma^{-1}(\vec{\mu}_1 - \vec{\mu}_0)) \tag{3.12}$$

3.2.2 Dimension Lifting to achieve linear DB

Assume a quadratic decision boundary:

$$F(\vec{x}) = \vec{x}^T \mathbf{A}\vec{x} + \vec{\alpha}^T\vec{x} + \alpha_0 \tag{3.13}$$

with

$$\vec{x} = (x_1, x_2)^T \in \mathbb{R}^2, \qquad \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \qquad \vec{\alpha} = (\alpha_1, \alpha_2), \alpha_0 \tag{3.14}$$

so $F(\vec{x})$ looks like:

$$F(\vec{x}) = a_{11}x_1^2 + (a_{12} + a_{21})x_1x_2 + a_{22}x_2^2 + \alpha_1 x1 + \alpha_2 x_2 + \alpha_0 \tag{3.15}$$

rewrite $F(\vec{x}) = \vec{\theta}^T\vec{x}'$ with:

$$\vec{\theta} = (a_{11}, a_{12} + a_{21}, a_{22}, \alpha_1, \alpha_2, \alpha_0)^T, \vec{x}' = (x_1^2, x_1x_2, x_2^2, x_1, x_2, 1)^T \tag{3.16}$$

3.2.3 Parameterization of the sigmoid function

every sigmoid function can be reformulate as a inner product of a parameter vector $\vec{\theta}$ (which can be created with the technique above) an the feature vector $\vec{x}$:

$$g(\theta^{\vec{T}}\vec{x}) = \frac{1}{1 + e^{\pm\vec{\theta}^T\vec{x}}} \tag{3.17}$$

This is used for the training step with ML-Estimation!

3.2.4 Perceptron and logistic Regression

Compute the decision boundary by a linear combination of the component of the feature Vector $\vec{x}$.
The prefactors α can be computed with logistic regression.

$$F(\vec{x}) = \sum_{i=0}^{d} \alpha_i \vec{x}_i = \vec{\alpha}^T \vec{x} \tag{3.18}$$

so the sigmoid function can be reformulate with the inner product $\alpha\vec{x}$ and α can be computed with the ML-Estimation.

4 Gaussian Classifier

if you use a generative modelling of the posterior probability and the class conditional pdf is a gaussian, then this is a Gaussian classifier. if all $p(\vec{x}|y)$ are gaussians then the optimal decision Boundary is quadratic! have all the Gaussians the same covariance matrices then the optimal decision Boundary is linear!
If the covariance matriceses are the Identity matrix then the classifier turns into a nearest-neighbour.

5 Naive Bayes

Naive Bayes assume that **all components** of the feature vector are **mutually independent** !

with this assumption you can factorize the class conditional pdf as follows:

$$p(\vec{x}|y) = \prod_{i=1}^{d} p(\vec{x}_i|y) \tag{5.1}$$

this means, each dimension is modeled by a pdf and all these pdfs are independent of all the other dimensions.

this gives the following decision rule:

$$y^* = \arg\max_y p(y|\vec{x}) = \arg\max_y p(y)p(\vec{x}|y) = \arg\max_y p(y)\prod_{i=1}^{d} p(\vec{x}_i|y) \tag{5.2}$$

the covariance matrix is a diagonal matrix. The decision boundary is a linear one if the classes share the same covariance matrix and is a quadratic one if they share not the same.
Naive Bayes get better the higher the dimension of the feature vectors is.

5.1 Curse of Dimensionality

Assume a 100-dimensional feature vector $\vec{x} \in \mathbb{R}^{100}$ belonging to class y which is normally distributed and all components are mutually dependent.
For the ccpdf (used in a normal Gaussian classifier) $\mathcal{N}(\vec{x}; \vec{\mu}, \Sigma)$ means that:

$$\vec{\mu}_y \in \mathbb{R}^{100}, \qquad \text{and} \qquad \Sigma = \Sigma^T \in \mathbb{R}^{100\times 100} \tag{5.3}$$

so the total number of parameters to be estimated for each class is:

$$100 + 100 \cdot (100 + 1)/2 = 5150 \tag{5.4}$$

In this case you have to estimate 5150 parameters with the ML-Estimation. The optimization problem have 5150 unknowns.

now use the **Naive Bayes** with the assumption that all components of $\vec{x}$ are **mutually independent**!
For the ccpdf $\mathcal{N}(\vec{x};\vec{\mu},\Sigma)$ means that:

$$p(\vec{x}|y) = \prod_{i=1}^{100} p(\vec{x}_i|y) = \prod_{i=1}^{100} \mathcal{N}(\vec{x}_i;\mu_i,\sigma_i^2) \tag{5.5}$$

For each component $i = \{1, 2, ..., 100\}$ we have to estimate the mean $\mu_i \in \mathbb{R}$ and the variance $\sigma_i^2 \in \mathbb{R}$ for the Gaussian.
so the total number of parameters to be estimated for each class is:

$$100 + 100 = 200 \tag{5.6}$$

In this case you have to estimate 200 parameters with the ML-Estimation which is feasible.

5.2 Decision Boundary

Folie 5-11 logit transform und generalized model abklären.

5.3 Statistical Dependencies of limited Order

This means dependencies between the components of feature vector between the two extremes of Bayes(all dependent) and Naive Bayes(all indipendent).
this would be of Order one:

$$p(\vec{x}|y) = p(\vec{x}_i|y) \cdot p(\vec{x}_2|y, x_1) \cdot p(x_3|y, x_2) \cdot .... = p(x_1|y) \prod_{i=2}^{d} p(x_i|y, x_{i-1}) \tag{5.7}$$

of Order two:

$$p(\vec{x}|y) = p(\vec{x}_i|y) \cdot p(\vec{x}_2|y, x_1) \cdot p(x_3|y, x_1, x_2) \cdot (x_4|y, x_2, x_3) \tag{5.8}$$

the order means the number of dependencies to the components before.

6 Transformations

The following transformations will introduce because they full fill one of the following three points:

- transform the features in a space where they are normally distributed with a identity matrix as the covariance matrix. (then it is the nearest neighbour classifier)

$$\Phi : \vec{x} \mapsto \vec{x}' \ , \qquad \underbrace{\vec{x}' \sim pdf}_{\vec{x}' \text{have a certain pdf}} \tag{6.1}$$

- transform the features in a higher dimensional space, where the decision boundary is linear

$$\Phi : \begin{matrix} \vec{x} \mapsto \vec{x}' \\ \mathbb{R}^d \rightarrow \mathbb{R}^D \end{matrix} \ , \text{ such that the decision boundary is linear} \tag{6.2}$$

- transform the features in a subspace where the classification problem can be solved sufficiently.

$$\Phi : \begin{matrix} \vec{x} \mapsto \vec{x}' \\ \mathbb{R}^d \rightarrow \mathbb{R}^D \end{matrix} \ , \ D << d \tag{6.3}$$

6.1 Transform to achieve a Gaussian ccpdf with identity Matrix

To find a feature transform such that the transformed features share the same covariance matrices which are the identity matrix we have to do the following reformulation: we start with the known Gaussian

$$\mathcal{N}(\vec{x}; \vec{\mu}, \Sigma) = \frac{1}{\sqrt{\det 2\pi\Sigma}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})} \tag{6.4}$$

now use the SVD to decompose Σ:
normally the SVD would be $\Sigma = UDV^T$ but you take benefit that Σ is semi definite symmetrical Matrix so it is:

$$\Sigma = UDU^T = (UD^{\frac{1}{2}}) \cdot I \cdot (UD^{\frac{1}{2}})^T \ , \qquad \text{where } I \text{ is the identity} \tag{6.5}$$

the covariance Matrix have to be full rank (which means it is invertible, and the inverse is used in the formula of the Gaussian)
we can get the inverse with the SVD:

$$\Sigma^{-1} = UD^{+}U^T = (UD^{-\frac{1}{2}}) \cdot I \cdot (UD^{-\frac{1}{2}})^T \ , \qquad \text{where } D^{+} \text{ means: invert the diagonal elements} \tag{6.6}$$

now we can use the gained formulation for Σ^{-1} in our Gaussian:

$$\mathcal{N}(\vec{x};\vec{\mu},\Sigma) = \frac{1}{\sqrt{\det 2\pi\Sigma}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T (UD^{-\frac{1}{2}}) \cdot I \cdot (UD^{-\frac{1}{2}})^T (\vec{x}-\vec{\mu})} \tag{6.7}$$

the goal is to get a normal distribution with an identity matrix as the covariance matrix. so you need to mix $(UD^{-\frac{1}{2}})$ in $(\vec{x}-\vec{\mu})$:

$$\mathcal{N}(\vec{x};\vec{\mu},\Sigma) = \frac{1}{\sqrt{\det 2\pi\Sigma}} \cdot e^{-\frac{1}{2}\left(\underbrace{(D^{-\frac{1}{2}}U^T)\vec{x}}_{\text{transformed } \vec{x}} - \underbrace{(D^{-\frac{1}{2}}U^T)\vec{\mu}}_{\text{transformed } \vec{\mu}}\right)^T \cdot I \cdot \left(\underbrace{(D^{-\frac{1}{2}}U^T)\vec{x}}_{\text{transformed } \vec{x}} - \underbrace{(D^{-\frac{1}{2}}U^T)\vec{\mu}}_{\text{transformed } \vec{\mu}}\right)} \tag{6.8}$$

summary of the mathematical crazy stuff above:
transform the features with the following transformation:

$$\vec{x}' = \Phi(\vec{x}) = D_y^{-\frac{1}{2}} U_y^T \vec{x} \tag{6.9}$$

then our Gaussian looks like:

$$p(\vec{x}'|y) = \mathcal{N}(\vec{x}';\vec{\mu}'_y,\Sigma'_y) = \mathcal{N}(\vec{x}'; D_y^{-\frac{1}{2}} U_y^T \vec{\mu}, I) \tag{6.10}$$

The problem is that the matrices are Class dependent! For the transformation we need to know the class a feature belongs to!
This problem solve the Linear Discriminant Analysis.

6.2 Linear Discriminant Analysis

Linear Discriminant Analysis deal with the problem that the feature transform is class dependent.

You can say that the LDA is a PCA on the mean vectors. We have a training set $S = \{(\vec{x}_i, y_i); i = 1....n)\}$

1. ML-Estimation of the joint covariance matrix:
$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\vec{x}_i - \vec{\mu}_{y_i})(\vec{x}_i - \vec{\mu}_{y_i})^T \tag{6.11}$$
the interpretation is that you compute the mean of the covariance matrices belonging to a class and use the mean covariance matrix for the classes.
2. Compute the SVD of the joint covariance matrix:
$$\hat{\Sigma} = UDU^T \tag{6.12}$$
3. Assign the transform:
$$\Phi = D^{-\frac{1}{2}}U^T \tag{6.13}$$
4. Compute mean vectors for all y:
$$\vec{\mu}'_y = \Phi(\vec{\mu}_y) = D^{-\frac{1}{2}}U^T\vec{\mu}_y \tag{6.14}$$

Output: feature transform Φ, transformed mean vectors $\vec{\mu}'_y$

the transformation in Step 3 generates spherical Data(important type of Data normalisation).
with the output of the linear discriminant Analysis we can set up our decision rule:

$$y^* = \arg\max_y p(y|\Phi(\vec{x})) = \arg\max_y \Big(\log p(y) - \frac{1}{2}(\Phi(\vec{x}) - \Phi(\vec{\mu}_y)^T \cdot I \cdot (\Phi(\vec{x}) - \Phi(\vec{\mu}_y)\Big) \tag{6.15}$$

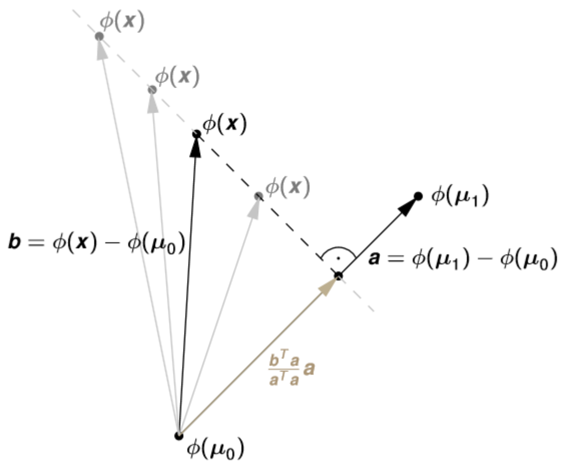
$\log p(y)$ is a constant, so you need to minimize the second term:

$$y^* = \arg\min_y \Big(\frac{1}{2}\left\|\Phi(\vec{x}) - \Phi(\vec{\mu}_y)\right\|_2^2 - \log p(y)\Big) \tag{6.16}$$

the feature transform Φ does not change the dimension of features!
the decision rule is the nearest neighbour (as we want to achieve with our transformation) and the prior probabilities are the offset the boundary.

6.2.1 Interesting knowledge about LDA and spherical Data

In 6.2.1 you can see that all points on the line can projected on the vector $\vec{\mu}_0 - \vec{\mu}_1$ and a valid classification is still possible. In a two class problem you can reduce the


$\phi(\boldsymbol{x})$
$\phi(\boldsymbol{x})$
$\phi(\boldsymbol{x})$
$\phi(\boldsymbol{x})$
$\phi(\boldsymbol{\mu}_1)$
$\boldsymbol{b} = \phi(\boldsymbol{x}) - \phi(\boldsymbol{\mu}_0)$
$\boldsymbol{a} = \phi(\boldsymbol{\mu}_1) - \phi(\boldsymbol{\mu}_0)$
$\frac{\boldsymbol{b}^T\boldsymbol{a}}{\boldsymbol{a}^T\boldsymbol{a}}\boldsymbol{a}$
$\phi(\boldsymbol{\mu}_0)$

dimensionality to one and do the nearest neighbour classification in the 1-D subspace and get the same results.
This can be formulated more general for K-classes with spherical Data:

- Class centroids span $(K-1)$-dimensional subspace
- Relative difference are not affected by coordinates in the $(d-K+1)$-dimensional subspace that is orthogonal to the $(K-1)$-dimensional subspace spanned by class centroids.

6.2.2 Rank-Reduced Linear Discriminant Analysis

The main target of Rank-Reduced LDA is to find a transformation Φ that project feature onto a sub-manifold such that the spread/the variance of the projected features is maximum.
The dimension of the sub-manifold is $L < K-1$. This works cause some dimensions may not providing a lot of separation between the classes, but just noise.
Search a linear mapping Φ such that the spread of the projected features is maximized.
For that compute the mean vector $\bar{\vec{\mu}}$ from the mean vectors of the classes.

1. Compute covariance matrix of LDA transformed mean vectors

$$\hat{\Sigma}_{inter} = \frac{1}{K}\sum_{y=1}^{K}(\vec{\mu}'_y - \bar{\vec{\mu}}')(\vec{\mu}'_y - \bar{\vec{\mu}}')^T \ , \text{ where } \bar{\vec{\mu}}' = \frac{1}{K}\cdot\sum_{y=1}^{K}\vec{\mu}'_y \tag{6.17}$$

   cause the means are used in this equation the inter class distance is maximized.

2. Compute the L eigenvectors of the covariance matrix belongig to the largest eigenvalue.

3. The eigenvectors are the rows of the mapping Φ from the $(K-1)$ to the L-dimensional feature space

output: matrix Φ

6.2.3 Fisher Transform

From the postulates of PR you know that the distance between to classes should be maximized (that is done above) but also that the distance between the features belonging to the same class should be minimized.
The Fisher Transform is a transform that transform the features with respect to these two points of the Postulate.

$$\vec{a}^* = \arg\max_{\vec{a}} \frac{\vec{a}^T \Sigma_{inter} \vec{a}}{\vec{a}^T \Sigma_{intra} \vec{a}} \tag{6.18}$$

6.3 Principal Component Analysis PCA

The main difference between PCA and LDA is that the PCA don't need the class information.
PCA transformed features are approximately normally distributed.(cause of the central limit theorem).
Components of PCA transformed features are mutually independent. This means its a good idea to use naive Bayes here which assume that the features are independent.
Compute the scatter matrix (covariance matrix) Σ:

$$\Sigma = \frac{1}{N} \sum_{i}^{N} \vec{x}_i \vec{x}_i^T \in \mathbb{R}^{d \times d} \tag{6.19}$$

Compute the eigenvector $\vec{v}$ and the eigenvalues λ:

$$\Sigma \vec{v} = \lambda \vec{v} \tag{6.20}$$

then sort the eigenvectors with decreasing eigenvalues.
When you project the feature vectors on the greatest eigenvector, then you have a dimensionality reduction and you can (not in all cases e.g adidas problem) separate the classes with a simply threshold.

6.4 Shape Modelling

Is used to get statistical Models.
Using principal component analysis, you can model shapes and their principal components. This gives you a parametrized model, where you can change some parameters and get new models which are realistic!

1. Sample an object at n surface points $p_k \in \mathbb{R}^3$ and encode these points as a feature Vector.So you get for each sampled object one feature Vector

$$\vec{x} = \begin{pmatrix} \vec{p}_1 \\ \vec{p}_2 \\ . \\ . \\ \vec{p}_n \end{pmatrix} = \begin{pmatrix} \vec{p}_{1,1} \\ \vec{p}_{1,2} \\ \vec{p}_{1,3} \\ \vec{p}_{2,1} \\ . \\ . \\ \vec{p}_{n,3} \end{pmatrix} \in \mathbb{R}^3 \tag{6.21}$$

2. Compute the covariance matrix from m shapes using

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} \vec{x}_i \vec{x}_i^T \tag{6.22}$$

3. then compute the principal components to get the spectral decomposition of Σ

$$\Sigma = \sum_{i=1}^{m} \lambda_i \vec{e}_i \vec{e}_i^T \tag{6.23}$$

4. where λ_i denote eigenvalues and $\vec{e}_i$ are the corresponding eigenvectors.
   You can change the λ_i to get new models: Shape vectors x^* within the eigenvector space can be computed using linear combination of l eigenvectors:

$$x^* = \bar{\vec{x}} + \sum_{i=1}^{l} a_i \vec{e}_i \tag{6.24}$$

7 Linear Regression

In the two class situation $y \in \{-1, +1\}$ this decision rule can be used:

$$y^* = sgn(\vec{\alpha}^T \vec{x} + \alpha_0) \tag{7.1}$$

to get $\vec{\alpha}$ and α_0 with the least square estimation, you need to use matrix notation for a given set of learning data:

$$\mathbf{X} = \begin{pmatrix} \vec{x}_1^T & 1 \\ \vec{x}_2^T & 1 \\ . & . \\ . & . \\ \vec{x}_m^T & 1 \end{pmatrix} \in \mathbb{R}^{m \times (d+1)}, \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ . \\ . \\ y_m \end{pmatrix} \text{, define: } \vec{\theta} = \begin{pmatrix} \vec{\alpha} \\ \alpha_0 \end{pmatrix} \tag{7.2}$$

You can solve the Problem with SVD

$$\mathbf{X}\vec{\theta} = \mathbf{y} \;\Leftrightarrow\; \vec{\theta} = \mathbf{X}^+\mathbf{y}, \qquad \text{with } \mathbf{X}^+ = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T \tag{7.3}$$

7.1 Ridge Regression

we extend the objective function $||\mathbf{X}\vec{\theta} - \mathbf{y}||_2^2 \to min$ by an additional term constraining the Euclidean length of $\vec{\theta}$
The Way to achieve this can be expressed in two ways:

1 penalize the log-likeliood by $-\lambda \vec{\theta}^T \vec{\theta}$

2 enforece $\vec{\theta}$ to be normally distributed according to $\mathcal{N}(0, \text{diag}(\frac{1}{\lambda}))$

rewrite the objective function with added contraint gives you:

$$\hat{\vec{\theta}} = \arg\min_{\vec{\theta}} (\mathbf{X}\vec{\theta} - \mathbf{y})^T (\mathbf{X}\vec{\theta} - \mathbf{y}) + \lambda \cdot \vec{\theta}^T \vec{\theta} \tag{7.4}$$

which gives you this estimator:

$$\hat{\vec{\theta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y} \tag{7.5}$$

The more weight you put on the length of $\vec{\theta}$ the more the Term in the brackets looks like a diagonal Matrix.

7.1.1 Lasso

Its the objective function from Ridge Regression except that the Lasso uses the L_1-norm for the constraint $\vec{\theta}$ instead of the Euclidean

$$\hat{\vec{\theta}} = \arg\min_{\vec{\theta}} ||\mathbf{X}\vec{\theta} - \mathbf{y}||_2^2 + \lambda \cdot ||\vec{\theta}||_1 \tag{7.6}$$

Cause of the L_1-norm you get in the most cases a sparse solution.

8 Norms

A norm is a measure for the length of a vector.

- the inner product of vectors $\vec{x}, \vec{v} \in \mathbb{R}^d$ is defined by:

$$\langle \vec{x}, \vec{v} \rangle = \vec{x}^T \vec{v} = \sum_{i=1}^{d} x_i v_i \tag{8.1}$$

- the inner product of matrices $\mathbf{X}\mathbf{Y}, \in \mathbb{R}^{m \times n}$ is defined by:

$$\langle X, Y \rangle = \mathrm{tr}(\mathbf{X}^T \mathbf{Y}) = \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j} y_{i,j} \tag{8.2}$$

- The Forbenius norm in term of an inner matrix product corresponds to the L_2-norm for vectors:

$$||\mathbf{X}||_F = \sqrt{\langle \mathbf{X}, \mathbf{X} \rangle} = \sqrt{\mathrm{tr}(\mathbf{X}^T \mathbf{X})} \tag{8.3}$$

L_0-norm: denotes the number of non-zero entries. The L_0-norm is not a norm because it is not homogeneous

L_p-norm: (p $>$ 0)

$$||\vec{x}||_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{\frac{1}{p}} \tag{8.4}$$

L_1-norm: sum of absolute values

$$||\vec{x}||_1 = \sum_{i=1}^{d} |\vec{x}_i| \tag{8.5}$$

L_2-norm: sum of squared values

$$||\vec{x}||_2 = \sqrt{\sum_{i=1}^{d} \vec{x}_i^2} \tag{8.6}$$

L_∞-norm: maximum norm

$$||\vec{x}||_\infty = \max_i\{|x_i|; i = 1, 2, ..., d\} \tag{8.7}$$

$L_\mathbf{P}$-norm: **P** is a symmetric positive definite matrix
the quadratic $L_\mathbf{P}$-norm is defined by:

$$||\vec{x}||_\mathbf{P} = \sqrt{\vec{x}^T\mathbf{P}\vec{x}} = \sqrt{(\mathbf{P}^{\frac{1}{2}}\vec{x})^T\mathbf{P}^{\frac{1}{2}}\vec{x}} = ||\mathbf{P}^{\frac{1}{2}}\vec{x}||_2 \tag{8.8}$$

Norms can be also used to measure the distance between to vectors $\vec{x}$ and $\vec{v}$:

$$\text{dist}(\vec{x}, \vec{v}) = ||\vec{x} - \vec{v}|| \tag{8.9}$$

The Malahanobis-Distance is a $L_\mathbf{P}$-norm where **P** is the inverse of the covariance Matrix Σ:

$$||\vec{x} - \vec{v}||_{\Sigma^{-1}} = \sqrt{(\vec{x} - \vec{v})^T\Sigma^{-1}(\vec{x} - \vec{v})} \tag{8.10}$$

The interpretation with respect to PR is that points in the direction of the first principal axis of the covariance matrix are less penalized than points in the direction of the last principal axis.

8.0.2 Unit Balls

The set of vectors of length less or equal to one according to the norm $||.||$ is called the unit ball.

L_0-norm coordinate Axis

L_1-norm diamond

L_2-norm circle

L_∞-norm square

L_P-norm is usually an ellipsis

8.1 Norm Dependent Linear Regression

The different norms can be used in the linear Regression. its all about to minimize the following:

$$\vec{x}^* = \arg\min_{\vec{x}} ||\mathbf{A}\vec{x} - \vec{b}||_? \tag{8.11}$$

so minimizing with respect to a difference between vectors, here you can use different norms and of course get different results.(For example use the L_1-norm results in the Lasso).

the estimation error ϵ is defined by:

$$\epsilon = ||\vec{x}^* - \hat{\vec{x}}||_? \tag{8.12}$$

where $\vec{x}^*$ is the correct value. We also know that the residual $\mathbf{A}\vec{x} - \vec{b}$ will be zero, if $\vec{b}$ is in the range of $\mathbf{A}$

8.1.1 Chebyshev Linear Regression

The Chebyshev linear regression minimizes the residual using the L_∞-norm:

$$||\mathbf{A}\vec{x} - \vec{b}||_\infty \to \min = \max\{|r_1|, |r_2|, ..., |r_m|\} \tag{8.13}$$

This problem can be rewritten such that the problem is convex.
minimize the residual subject to:

$$-r \cdot \mathbf{1} \succeq \mathbf{A}\vec{x} - \vec{b} \preceq r \cdot \mathbf{1}, \qquad \mathbf{1} \in \{1\}^m,\ r \in \mathbb{R} \tag{8.14}$$

$\preceq$ means component wise smaller or equal.

8.1.2 Sum of Absolute Residuals

Minimizing the residual using the L_1-norm:

$$||\mathbf{A}\vec{x} - \vec{b}||_1 \to \min \tag{8.15}$$

this optimization problem can be also rewritten such that the problem is a convex one.
minimizing $\vec{1}^T\vec{r}$ subject to:

$$-\vec{r} \succeq \mathbf{A}\vec{x} - \vec{b} \preceq \vec{r} \tag{8.16}$$

8.1.3 Compressed Sensing

Assume we have fewer measurements than required to estimate the parameter vector $\vec{x}$ in a regularized linear regression (i.e. ridge regression) problem. We need a solution for the underdetermined case. We can solve this by assuming the vector $\vec{x}$ is sparse, i.e. by optimizing w.r.t. the L_1-norm. We minimize $||\vec{x}||_1$ subject to

$$\mathbf{A}\vec{x} = \vec{b} \tag{8.17}$$

Note that this is basically the Lasso method.

8.2 Penalty Functions

$$\text{minimize} \sum_{i=1}^{m} \phi(r_i), \quad \text{subject to} \;\; \vec{r} = (r_1, r_2, ..., r_m)^T = \mathbf{A}\vec{x} - \vec{b} \tag{8.18}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the penalty function for the components of the residual vector. The penalty function computes "costs"for the residual and these costs have to be minimized. Norms are a special case of penalty functions. You can use norms to calculate a value for the residual and minimize that with respect to the choosen norm:

$$L_1\text{-norm} \qquad \phi_{L_1}(r) = ||r||_1 \tag{8.19}$$

$$L_2\text{-norm} \qquad \phi_{L_2}(r) = ||r||_2 \tag{8.20}$$

8.2.1 The Log Barrier Function

$$\phi_{barrier}(r) = \begin{cases} -a^2 \log(1 - (\frac{r}{a})^2) & \text{if } \; |r| < a \\ \infty & \text{otherwise} \end{cases} \tag{8.21}$$

the log barrier function only accepts solutions in a tube. This can be used to exlude solutions that would not be feasible.

8.2.2 Dead Zone Linear Penalty Function

$$\phi_{barrier}(r) = \begin{cases} 0 & \text{if } \; |r| < a \\ |r| - a & \text{otherwise} \end{cases} \tag{8.22}$$

defines a dead zones where the penalty is 0 and otherwise behaves like the L_1-norm.

8.2.3 The Large Error Penalty Function

$$\phi_{barrier}(r) = \begin{cases} r^2 & \text{if } \; |r| < a \\ a^2 & \text{otherwise} \end{cases} \tag{8.23}$$

penalize below a like the L_2-norm and outside with a constant.

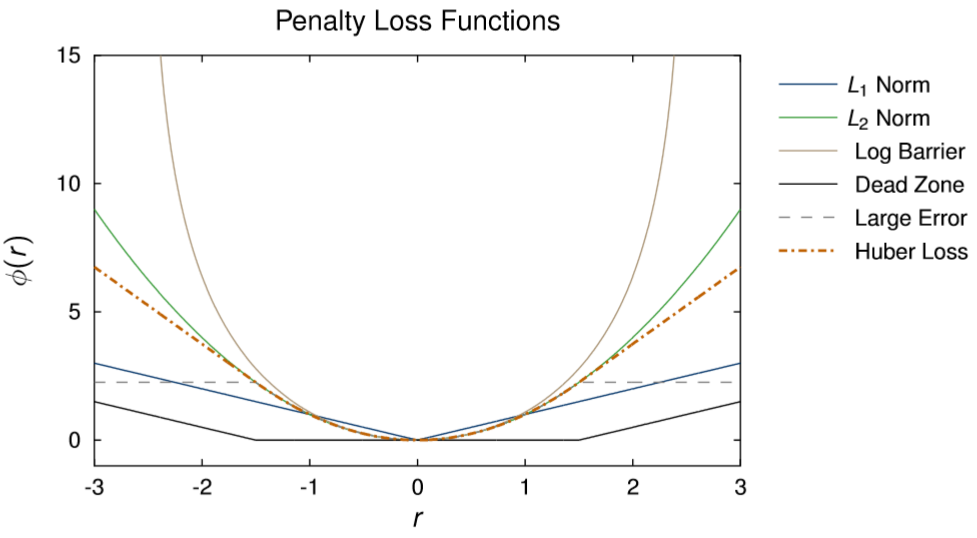
8.2.4 The Huber Function

$$\phi_{barrier}(r) = \begin{cases} r^2 & \text{if } |r| < a \\ a \cdot (2|r| - a) & \text{otherwise} \end{cases} \tag{8.24}$$

The Huber function is an approxmiation of the absolute value, but is smooth at the origin. It penalize below a like the L_2-norm and over a like L_1. The transition between these two functions is smooth.

8.2.5 Overview Penalty Functions

In 8.2.5 all penalty functions are drawn.



9 Perceptron

9.1 Rosenblatt's Perceptron

The motivation is to compute a linear decision boundary. Assume we have linearly separable classes.
The distance of the misclassified features is used to model the decision boundary. This distance should be minimized.

$$y^* = \operatorname{sgn}(\vec{\alpha}^T \vec{x} + \alpha_0) \tag{9.1}$$

Assume the class numbers are $y = \pm 1$. $\vec{\alpha}^T \vec{x} + \alpha_0$ computes the distance to the line / hyperplane which is defined by $\vec{\alpha}$. On the basis of the sign of the distance you can do the classification.
This leads to the optimization problem: The feature Vectors get classified with initialized parameter $\alpha_0, \vec{\alpha}$.
The optimization will be done with respect to all misclassified Features!

$$\text{minimize} \quad D(\alpha_0, \vec{\alpha}) = -\sum_{x_i \in \mathcal{M}} \underbrace{y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0)}_{\text{always negative}} \tag{9.2}$$

The misclassified features are in$\mathcal{M}$.
The consequence is that the cardinality of $\mathcal{M}$ changes each time you change the parameter $\alpha_0, \vec{\alpha}$.
To minimize this objective function we need the gradient with respect to $\alpha_0, \vec{\alpha}$:

$$\frac{\delta}{\delta \alpha_0} D(\alpha_0 \vec{\alpha}) = -\sum_{x_i \in \mathcal{M}} y_i \tag{9.3}$$

$$\frac{\delta}{\delta \vec{\alpha}} D(\alpha_0 \vec{\alpha}) = -\sum_{x_i \in \mathcal{M}} y_i \cdot \vec{x}_i \tag{9.4}$$

This can be used to create a iterative scheme to update the parameter $\alpha_0, \vec{\alpha}$.
The following scheme will be done for every misclassified feature.

$$\begin{pmatrix} \alpha_0^{(k+1)} \\ \vec{\alpha}^{(k+1)} \end{pmatrix} = \begin{pmatrix} \alpha_0^{(k)} \\ \vec{\alpha}^{(k)} \end{pmatrix} + \lambda \begin{pmatrix} y_i \\ y_i \cdot \vec{x}_i \end{pmatrix} \tag{9.5}$$

λ is the learning rate! After updating the parameter $\alpha_0, \vec{\alpha}$ you classify all features again with the updated parameters. This will be done till all features are correct classified (if the classes are not linear separable this scheme wont end).
The interpretation is:
The parameter $\vec{\alpha}$ of the decision boundary is a linear combination of feature vectors. (Hint: cause of this its interesting to initialize with 0, so the initial value falls of the equation).
This means you can interpret your decision Boundary $F(x)$ as follow:

$$F(\vec{x}) = \underbrace{\left(\sum_{i\in\epsilon} y_i \cdot \vec{x}_i\right)^T}_{\vec{\alpha}} \vec{x} + \underbrace{\sum_{i\in\epsilon} y_i}_{\alpha_0} = \sum_{i\in\epsilon} y_i \cdot \langle \vec{x}_i, \vec{x} \rangle + \sum_{i\in\epsilon} y_i \tag{9.6}$$

where ϵ is the set of indices that required an update. Theses sum are the summary of the training process.
You can model your decision boundary just by the instances themselves. In contrast to a NN where you have to test against all the instances in the dataset.

- the final linear decision boundary depends on the initialization
- the number of iterations can be rather large

9.1.1 Convergence of the learning algorithm

Assume that for all $(i = 1, 2, ..., m)$

$$y_i(\vec{x}_i^T \vec{\alpha}^* + \alpha_0^*) \geq \rho \tag{9.7}$$

where $\rho > 0$ and $||\vec{\alpha}^*||_2 = 1$. Let $M = \max_i ||\vec{x}_i||_2$
then the learning algorithm is converges to a linear decision boundary (if its possible) after k iterations, where k is bounded by

$$k \leq \frac{(\alpha_0^{*^2} + 1)(1 + M^2)}{\rho^2} \tag{9.8}$$

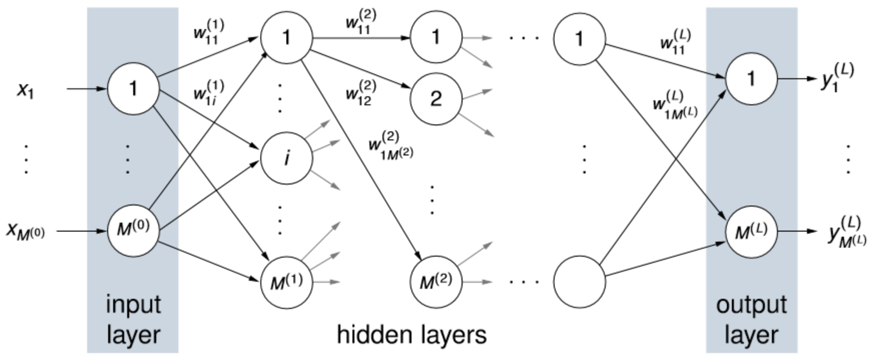
The convergence of the algorithm thus depends on the maximum distance from the coordinate origin, the offset of the decision boundary and the width of the area around the decision boundary without any feature vectors.
Note:

- that the objective function changes in each iteration step $\rightarrow$ non-linear optimization

- the optimization problem is discrete
- that the number of iterations does **not** depend on the dimension of the feature vectors!

9.2 Multi-Layer Perceptrons

A Multi-Layer Perceptron consists of several Perceptrons. The topology is shown in 9.2



One Perceptron in the Multi-Layer Perceptron can be modelled by:

$$net_j^{(l)} = \sum_{i=1}^{M^{(l-1)}} y^{(l-1)} \underbrace{w_{ij}^{(l)}}_{\vec{\alpha}} - \underbrace{w_{0j}^{(l)}}_{\alpha_0} \tag{9.9}$$

$$y_j^{(l)} = f(net_j^{(l)}) \tag{9.10}$$

$f(...)$ is the so called activation function, Rosenblatts Perceptron uses here the sign-function. But its not limited to that, you can also use for example the sigmoid function.
To train the mulit-layer perceptron, use the gradient descent method to ajust the weights:

$$\Delta w\{ij\}^{(l)} = -\eta \frac{\delta \epsilon}{\delta w_{ij}^{(l)}} \tag{9.11}$$

A typical error function is the mean squared error:

$$\epsilon_{\text{MSE}}(w) = \frac{1}{2} \sum_{k=1}^{M^{(L)}} (t_k - y_k^{(L)})^2 \tag{9.12}$$

where t_k is the expected result from the labeled sample and $y_k^{(L)}$ is the classification from the output layer. Using the update rule and the mean square error, we can setup a backpropagation algorithm that will work layer by layer from the back to estimate the input weights of any given layer l.

10 Optimization

10.1 general stuff

A short interpretation of Convexity:
The geometric interpretation is of the convexity of a function is that if you draw a line between two points on the line, the line should be above the function and don't cross the function.

10.2 unconstrained optimization

Compute the minimum of f and assume that the function $f : \mathbb{R}^d \to \mathbb{R}$ is convex and twice differentiable.
The unconstrained optimization is just the solution of the minimization problem

$$\vec{x}^* = \arg\min_{\vec{x}} f(\vec{x}) \tag{10.1}$$

where $\vec{x}^*$ denotes the optimal point. For this family of functions, a necessary and sufficient condition for the minimum are the zero-crossings of the function's gradient:

$$\Delta f(\vec{x}^*) = 0 \tag{10.2}$$

In the most cases you don't find a close form solution. So use iterative scheme e.g. Newton-Raphson.

10.3 Descent Methods

The principal Idea is a iterative scheme of computing the gradient and go into the negative direction of the gradient with a stepsize t.

$$\vec{x}^{(k+1)} = g(\vec{x}^{(k)}) = \vec{x}^{(k)} + t^{(k)} \Delta \vec{x}^{(k)} \tag{10.3}$$

where

$$\Delta\vec{x}^{(k)} \in \mathbb{R}^d : \quad \text{is the search direction in the k-th iteration} \tag{10.4}$$

$$t^{(k)} \in \mathbb{R} : \quad \text{denotes the step length in the k-th iteration} \tag{10.5}$$

and where we expect:

$$f(\vec{x}^{(k+1)}) < f(\vec{x}^{(k)}) \qquad \text{except} \qquad \vec{x}^{(k+1)} = \vec{x}^{(k)} = \vec{x}^{*}) \tag{10.6}$$

A good value for the Stepsize t is important. If t is oversized you jump back and forth, if t will be reduced after each iteration step there is a high probability to get stuck before you reach the optimal point. To find a stepsize its called line search. You can see the it as a function $F(\vec{x}^{(k)}) + t^{(k)}\Delta\vec{x}^{(k)})$ where $\vec{x}^{(k)})$ and $\Delta\vec{x}^{(k)}$ is fixed, so $F(...)$ is a function in the variable t. This function can be optimized in t.
A very good algorithm to estimate t is the **Armijo-Goldstein Algorithm**

To get the Direction there is the very intuitive way to use the negative gradient $\Delta\vec{x}^{(k)} = -\nabla f(\vec{x}^{(k)})$, which is the steepest descent direction.
A more general Method is ne normalized steepest Descent Method where you include Norms:

$$\Delta\vec{x} = \arg\min_{u}\{\nabla f(\vec{x})^T\vec{u}; ||\vec{u}||_p = 1\} \tag{10.7}$$

Which means the compute the inner product of the gradient and a normal vector $\vec{u}$ (project the gradient on $\vec{u}$).
And we use the normal vector where the projection of the gradient is maximum. If you use the L_2-norm then its the normal steepest descent method.
The steepest descent direction depends on the chosen norm.
Using the L_1-norm leads to a coordinate descent method, which means you choose always a coordinate axis as the direction.
Using the L_∞-norm the direction will be always one of the 45° diagonals according to the coordinate system.
Using the $L_{\mathbf{P}}$-norm means you project the gradient on the greatest principal component.
That's something you did already in the LDA. Use the transform to get spherical data and there you can use the normal gradient descent method.
You can rewrite:

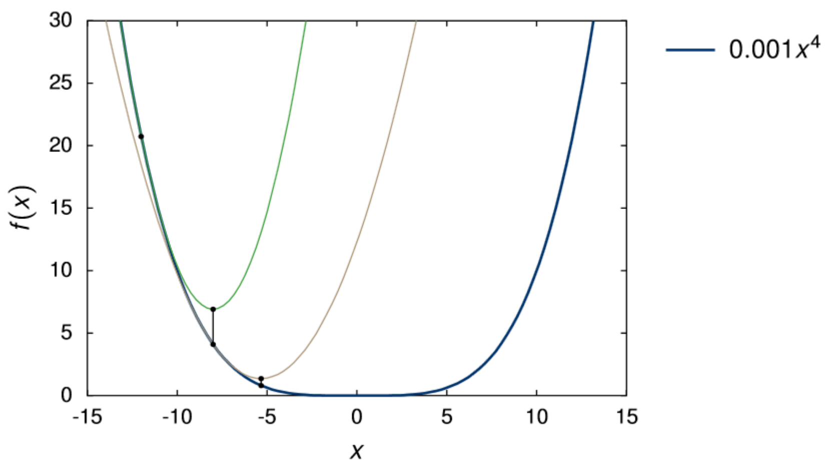
$$\Delta\vec{x} = \arg\min_{u}\{\nabla f(\vec{x})^T\vec{u}; ||\vec{u}||_{\mathbf{P}} = 1\} \tag{10.8}$$

and at the end you get the following to compute the direction in the $L_{\mathbf{P}}$-norm:

$$\Delta\vec{x} = -\mathbf{P}^{-1}\nabla f(\vec{x}) \tag{10.9}$$

10.4 Newton's Method

The Newton's Method finds the nearest Extrema.
It's also a iterative scheme(shown in 10.4). At the Point $x^{(k)}$ you have to approximate the function with a second order Taylor approximation and compute the minimum of this approximation. This minimum is the new $x^{(k+1)}$. The second order Taylor



approximation is:

$$f(\vec{x} + \Delta\vec{x}) \approx f(\vec{x}) + \nabla f(\vec{x})^T \Delta\vec{x} + \frac{1}{2}\Delta\vec{x}^T(\nabla^2 f(\vec{x}))\Delta\vec{x} \tag{10.10}$$

Compute the gradient of the Taylor approximation and set it to zero, then you get:

$$\Delta\vec{x} = -\underbrace{(\nabla^2 f(\vec{x}))^{-1}}_{\text{inverse of the Hessian}} \nabla f(\vec{x}) \tag{10.11}$$

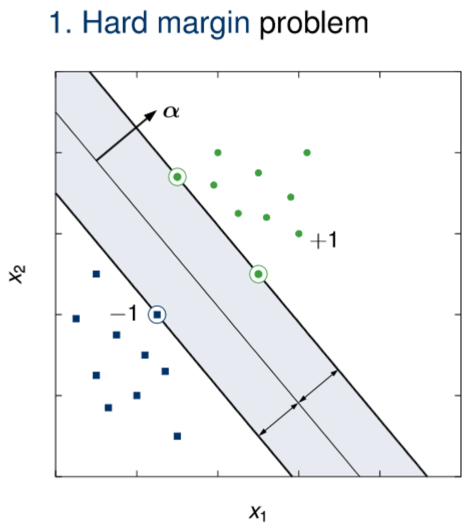
Note: This is the same you see at the gradient descent method using the $L_{\mathbf{P}}$-norm !
The Newton method is an $\vec{x}$-dependent steepest descent methid regarding the $L_{\mathbf{P}}$-norm, where $\mathbf{P} = \nabla^2 f(\vec{x})$ is the Hessian matrix.

11 Support Vector Machines

SVM's are very good in terms of their generalization properties.
Assume two linearly separable classes. Computation of a unique linear decision boundary that allows the separation of training data and that generalizes well.
The main Idea is to look for an decision boundary that separates to classes but maximize the distance the nearest points of each class. The solution is unique and depends only on the features that a close to the decision boundary.

11.1 Hard Margin Problem

The hard margin SVM 11.1 needs linearly separable classes.



Let's assume there is an affine function that defines the decision boundary:

$$f(\vec{x}) = \vec{\alpha}^T \vec{x} + \alpha_0 \tag{11.1}$$

Note: where $\vec{\alpha}$ is the not normalized normal vector of the hyperplane.
There are three points to think about before you end up with a optimization:
Let's introduce the following constraints:

$$\vec{\alpha}^T\vec{x}_{y=+1} + \alpha_0 \geq 1$$
$$\vec{\alpha}^T\vec{x}_{y=-1} + \alpha_0 \leq -1$$

this means, if you take a sample from the +1 class then the decision rule computes a value equal or greater then 1. And for a sample from the -1 class the decision rule returns a value smaller or equal to -1.
this constraint can be rewritten such that you have only one equation ($y \in \{+1, -1\}$):

$$y_i \cdot (\vec{\alpha}^T\vec{x}_i + \alpha_0) - 1 \geq 0 \tag{11.2}$$

To compute the width of the margin, take a samples from both classes which lay directly on the margin $y_i(\vec{\alpha}^T\vec{x}_i + \alpha_0) - 1 = 0$. Subtract them from each other if you now project the resulting vector on a normalized vector which is orthogonal to the hyperplane, then you get the width of the margin:

$$\text{width} = \frac{\vec{\alpha}}{||\vec{\alpha}||_2} \cdot (\vec{x}_{y=+1} - \vec{x}_{y=-1}) \tag{11.3}$$

If you rewrite the constraint above for these two Vectors $x_{y=+1}; x_{y=-1}$

$$\vec{\alpha}\vec{x}_{y=+1} = 1 - \alpha_0 \qquad -\vec{\alpha}\vec{x}_{y=-1} = 1 + \alpha_0$$

you can use this to rewrite the equation of the width of the margin:

$$\text{width} = \frac{\vec{\alpha} \cdot \vec{x}_{y=+1}}{||\vec{\alpha}||_2} - \frac{\vec{\alpha} \cdot \vec{x}_{y=-1}}{||\vec{\alpha}||_2}$$
$$\text{width} = \frac{1 - \alpha_0}{||\vec{\alpha}||_2} - \frac{-(1 + \alpha_0)}{||\vec{\alpha}||_2}$$

the goal is to maximize the width, simplify the equation above leads to:

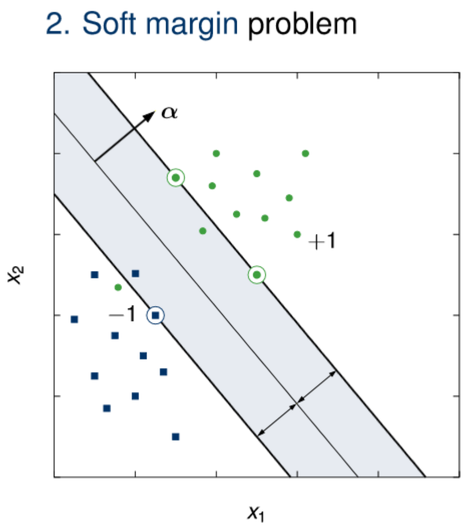
$$\max \text{width} = \max \frac{2}{||\vec{\alpha}||_2} \tag{11.4}$$
$$\text{subject to} \quad y_i \cdot (\vec{\alpha}^T\vec{x}_i + \alpha_0) - 1 \geq 0$$

instead of maximizing the fraction you can minimize the denominator. To get a better optimization problem take the square of $||\vec{\alpha}||_2$:

$$\begin{aligned} &\text{minimize } \frac{1}{2}||\vec{\alpha}||_2^2 \\ &\text{subject to} \quad y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1 \geq 0 \end{aligned}$$

11.2 Soft Margin Problem

The Soft Margin SVM11.2 allows features to be on the wrong side of the decision boundary. So the classes are not linearly separable anymore. In this case the



optimization change to:

$$\begin{aligned} &\text{minimize } \frac{1}{2}||\vec{\alpha}||_2^2 + \mu \sum_i \xi_i \\ &\text{subject to} \quad -(y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1 + \xi_i) \leq 0 \\ &\qquad -\xi_i \leq 0 \end{aligned}$$

ξ_i denotes misclassified features, so you minimizes like in the Hard Margin case and add as a constraint that the sum over the misclassified features $\mu \sum_i \xi_i$ should be also minimum.

11.3 Applying Constrained Optimization to SVM

Take the optimization problem from the Soft Margin SVM and set up the Lagrangian Function L:

$$L(\underbrace{\vec{\alpha}, \alpha_0, \vec{\xi}}_{\substack{\texttt{this was}\\ \vec{x}\ \texttt{in the}\\ \texttt{explanation}\\ \texttt{Part}}}, \vec{\lambda}, \mu) = \underbrace{\frac{1}{2}||\vec{\alpha}||_2^2 + \mu \sum_i \xi_i}_{\substack{\texttt{primal problem}\\ f_0(\vec{x})}} - \underbrace{\sum_i \mu_i \xi_i}_{\substack{\texttt{1. inequality}\\ \texttt{constraint}}} \underbrace{- \sum_i \lambda_i (y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1 + \xi_i)}_{\substack{\texttt{2. inequality}\\ \texttt{constraint}}}$$

Now use the KKT-conditions.
First get the derivative of $\nabla L \overset{!}{=} 0$ with respect to $\vec{\alpha}, \alpha_0, \vec{\xi}$.

$$\frac{\delta L}{\delta \vec{\alpha}} = \vec{\alpha} - \sum_i \lambda_i y_i \vec{x}_i \overset{!}{=} 0 \Longleftrightarrow \vec{\alpha} = \sum_i \lambda_i y_i \vec{x}_i \tag{11.5}$$

the implication of this is that $\vec{\alpha}$, which defines the decision boundary, is a linear combination of feature Vectors!

$$\frac{\delta L}{\delta \alpha_0} = -\sum_i \lambda_i y_i \overset{!}{=} 0 \tag{11.6}$$

The following is the derivative with respect to a particular ξ_i:

$$\frac{\delta L}{\delta \xi_i} = \vec{\mu} - \mu_i - \lambda_i \overset{!}{=} 0 \tag{11.7}$$

For the basic understanding its enough to go further with the Hard Margin SVM, so the following equations are without the terms with the slack-variables ξ.
For a better overview here again the dual problem for the hard margin:

$$L_D = \frac{1}{2}\vec{\alpha}^T \vec{\alpha} - \sum_i \lambda_i (y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1) \quad = L(\vec{\alpha}, \alpha_0, \vec{\lambda})$$

this can be rewritten and then you can use your knowledge achieved by the derivatives above how the function has to be that the 4th KKT-condition is fulfilled:

$$L(\vec{\alpha}, \alpha_0, \vec{\lambda}) = \frac{1}{2}\vec{\alpha}^T \vec{\alpha} - \Bigg(\underbrace{\sum_i \lambda_i y_i \cdot \vec{x}_i}_{\substack{\vec{\alpha}\\ \texttt{see 11.5}}}\Bigg)^T \vec{\alpha} - \underbrace{\sum_i \lambda_i y_i}_{\substack{=0\\ \texttt{see 11.6}}} \alpha_0 + \sum_i \lambda_i$$

this can be rewritten such that you get a function which depends neither on $\vec{\alpha}$ nor on α_0:

$$L(\vec{\lambda}) = -\frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j \cdot \vec{x}_i^T \vec{x}_j + \sum_i \lambda_i \tag{11.8}$$

this yields a simpler optimization Problem which contains our original constraints:

$$\text{maximize} \quad -\frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j \cdot \vec{x}_i^T \vec{x}_j + \sum_i \lambda_i \tag{11.9}$$

$$\text{subject to:} \quad \begin{array}{c} \vec{\lambda} \geq 0 \\ \sum_i \lambda_i y_i = 0 \end{array} \tag{11.10}$$

For strong convex functions the duality gap is zero (what you want to achieve), if the KKT-conditions are satisfied. Especially the complementary slackness condition must be fulfilled:

$$\forall i : \qquad \lambda_i \underbrace{(y_i \cdot (\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1)}_{f_i(\vec{\alpha}, \alpha_0)} = 0 \tag{11.11}$$

there are two possibilities that the complementary slackness is fulfilled.
Either the Lagrangian-multiplier λ_i or the feature Vector $\vec{x}_i$ is Zero.
If $\lambda_i > 0$ then the following is left:

$$y_i(\vec{\alpha}^T \vec{x}_i + \alpha_0) - 1 = 0 \tag{11.12}$$

$$y_i(\vec{\alpha}^T \vec{x}_i + \alpha_0) = 1 \tag{11.13}$$

this are exactly the constraint 11.2 we introduce at the beginning with the purpose that the feature vectors which lay exactly on the margin have the distance of one. This means that the Lagrangian-multiplier λ_i for all samples which are not on the margin has to be Zero. Remember 11.5:

$$\vec{\alpha} = \sum_i \lambda_i y_i \vec{x}_i \tag{11.14}$$

The linear combination of the normal Vector $\vec{\alpha}$ from the decision boundary can only depend on Vectors with a $\lambda_i > 0$.
The Conclusion is that only the feature vectors exactly on the margin have a Lagrangian-multiplier $\lambda_i > 0$ and only that Vectors generates $\vec{\alpha}$! Using the knowledge from 11.5 you can rewrite the decision boundary as follows:

$$f(\vec{x}) = \underbrace{\left(\sum_{i=1}^{m} \lambda_i y_i \vec{x}_i\right)^T}_{\vec{\alpha}} \vec{x} + \alpha_0 \tag{11.15}$$

11.4 Feature Transform in the case of SVM

Both hard and soft margin SVM's can only generate a linear decision boundary. Which have serious limitations:

- Non-linearly separable data cannot be classified
- Noisy data can cause problems
- Formulation deals with vectorial data only

To get rid of these problems you can use Feature Transform. Feature transform can be easily applied cause of the rewritten decision function using the knowledge from the dual Problem.
Map data into richer feature space using non-linear feature transform, then use a linear classifier.
Select a feature transform $\Phi : \mathbb{R} \to \mathbb{R}^D$ such that the resulting features $\Phi(\vec{x}_i); i = 1, 2, ..., m$ are linearly separable. A short Example for explanations: As already shown in the Logistic Regression part, assume the decision boundary is given by the quadratic function:

$$f(\vec{x}) = \alpha_0 + \alpha_1 x_1^2 + \alpha_2 x_2^2 + \alpha_3 x_1 x_2 + \alpha_4 x_1 + \alpha_5 x_2 = \vec{\alpha}^T \Phi(\vec{x}) \tag{11.16}$$

By the following mapping you get features that have a linear decision boundary;

$$\Phi(\vec{x}) = \begin{pmatrix} 1 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_1 \\ x_2 \end{pmatrix} \tag{11.17}$$

Because you kicked out $\vec{\alpha}$ from the decision boundary, you don't need to estimate or compute the components of $\vec{\alpha}$ for the transformation above.
You can rewrite the decision boundary with an inner product of the Φ:

$$f(\vec{x}) = \sum_{i=1}^{m} \lambda_i y_i \langle \Phi(\vec{x}_i), \Phi(\vec{x}) \rangle + \alpha_0 \tag{11.18}$$

The optimization problem changes as follow:

$$\text{maximize} \quad -\frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j y_i y_j \cdot \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle + \sum_i \lambda_i$$

$$\text{subject to:} \quad \begin{aligned} \vec{\lambda} &\geq 0 \\ \sum_i \lambda_i y_i &= 0 \end{aligned}$$

Now we have a linear decision boundary in a higher dimensional space.

Abstract this approach further using so called kernel-functions:

$$k(\vec{x}, \vec{x}') = \langle \Phi(\vec{x}), \Phi(\vec{x}') \rangle \tag{11.19}$$

Typical kernel-functions are:

Linear:

$$k(\vec{x}, \vec{x}') = \langle \vec{x}, \vec{x}' \rangle$$

Polynomial: the example above was a Polynomial Kernel with $d = 2$

$$k(\vec{x}, \vec{x}') = (\langle \vec{x}, \vec{x}' \rangle + 1)^d$$

Using kernel functions we can avoid modelling the transformation $\Phi(\vec{x})$ but still get the same result as if we had applied Φ.

12 Kernels

A kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a symmetric function that maps a pair of features to a real number.
For a kernel function the following property holds:

$$k(\vec{x}, \vec{x}') = \langle \Phi(\vec{x}), \Phi(\vec{x}') \rangle \tag{12.1}$$

for any feature mapping Φ.
If you have linear decision boundary and in their definition is an inner product of feature Vectors, you can take the kernel function to get a non-linear decision Boundary. We can use this Kernel trick as seen for the SVM but you can use this also for the Perceptron:

$$\text{Perceptron: } F(\vec{x}) = \sum_{i \in \epsilon} y_i \cdot \langle \vec{x}_i, \vec{x} \rangle + \sum_{i \in \epsilon} y_i$$

Another good thing about this is, you can pre-Compute a so called kernel matrix for a given set of feature vectors $\vec{x}_1, \vec{x}_2, ...\vec{x}_m$:

$$\mathbf{K} = [K_{i,j}], i, j = 1, 2, ..., m; \qquad \text{where} K_{i,j} = \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle \tag{12.2}$$

in this matrix are all inner products pre-computed and at the appropriate position. The entries of the Matrix are a measurement for the similarity of the transformed feature pairs.
The Kernel Matrix is positive semi definite!

We can compute for any kernel function$k(\vec{x}, \vec{x}')$ a feature mapping Φ such that the kernel function can be written as an inner product (Mercer's Theorem). This means we don't have to actually know Φ to use this technique.

12.1 The Kernel Trick

If any algorithm that is formulated in terms of a positive semidefinite kernel k, we can derive an alternative algorithm by replacing the kernel function k by another positive semidefinit kernel k'.

The message is that you don't have to know the feature transformation Φ. You just have to know what the inner product is of the transformed features. And a combination of the computation of the inner product and the transformation **is** the Kernel. The Kernel function does both!

12.2 Kernel PCA

Assume you have feature Vectors $\vec{x}_1, \vec{x}_2, ..., \vec{x}_m \in \mathbb{R}^d$ with zero mean.
Compute the scatter matrix and after that the eigenvalues λ and the eigenvectors $\vec{e}$.
Short facts about linear algebra:

- The eigenvectors $\vec{e}_i$ span the same space as the feature vectors
- Each eigenvector $\vec{e}_i$ can be written as a linear combination of feature vectors:

$$\vec{e}_i = \sum_k \alpha_{i,k} \vec{x}_k \tag{12.3}$$

with this knowledge the eigenvector/-value problem for the PCA computation can be rewritten:

$$\Sigma \vec{e}_i = \lambda_i \vec{e}_i$$
$$\underbrace{\left(\frac{1}{m} \sum_{j=1}^{m} \vec{x}_j x_j^T \right)}_{\Sigma} \cdot \underbrace{\sum_k \alpha_{i,k} \vec{x}_k}_{\vec{e}_i} = \lambda_i \underbrace{\sum_k \alpha_{i,k} \vec{x}_k}_{\vec{e}_i}$$
$$\sum_{j,k} \alpha_{j,k} \vec{x}_j \vec{x}_j^T \vec{x}_k = m \cdot \lambda_i \sum_k \alpha_{i,k} \vec{x}_k$$

to come up with a inner product of the feature vectors we multiply the equation with $\vec{x}_l$. The following equation have to be fulfilled for all projections on $\vec{x}_l$ for all indices l:

$$\sum_{j,k} \alpha_{j,k} \vec{x}_l^T \vec{x}_j \vec{x}_j^T \vec{x}_k = m \cdot \lambda_i \sum_k \alpha_{i,k} \vec{x}_l^T \vec{x}_k \tag{12.4}$$

and now all feature vectors show up in terms of inner products and we can use the kernel trick and express the inner prodcuts with the kernal function k!

$$\sum_{j,k} \alpha_{j,k} k(\vec{x}_l, \vec{x}_j) \cdot k(\vec{x}_j \vec{x}_k) = m \cdot \lambda_i \sum_k \alpha_{i,k} k(\vec{x}_l, \vec{x}_k) \tag{12.5}$$

Instead of the kernel-function you can Write also the Kernel-Matrix. The equation above holds for all l so you can sum up on both side.
This gives you the Key-Equation fpr Kernel-PCA:

$$\mathbf{K}^2 \cdot \vec{\alpha}_i = m \cdot \lambda_i \cdot \mathbf{K} \tag{12.6}$$

rearrange the equation:

$$\mathbf{K} \underbrace{(\mathbf{K}\vec{\alpha}_i - m\lambda_i\vec{\alpha}_i)}_{=0} = 0$$

The Matrix $\mathbf{K}$ cant be zero, so the term in the brackets has to be Zero.
So $\mathbf{K}\vec{\alpha}$ is a eigenvector of $\mathbf{K}$ And this term is again a eigenvalue/-vector problem:

$$\mathbf{K}\vec{\alpha}_i = m\lambda_i\vec{\alpha}_i \tag{12.7}$$

Kernel PCA (and thus also classical PCA) can be computed by solving an eigenvalue problem for a $m \times m$-matrix where m is the number of training samples. If you want to compute the direction with respect to your new coordinate system that's given by your eigenvectors. Then you have to project your feature vectors on these eigenvectors.
Projection c of transformed feature vector $\Phi(\vec{x})$ on principal component $\vec{e}_i$:

$$c = \Phi(\vec{x})^T \vec{e}_i = \sum_k \alpha_{i,k} \Phi(\vec{x})^T \Phi(\vec{x}_k) = \sum_k \alpha_{i,k} k(\vec{x}, \vec{x}_k) \tag{12.8}$$

So you can compute the coordinates in the new system without knowing Φ!
You assumed that the transformed features have zero mean. Unfortunately you can't say that the transformation Φ (which you don't know) have zero mean after the transformation.
So look for a transformation $\tilde{\Phi}$ such that the transformed features $\tilde{\Phi}(\vec{x})$ have Zero mean:

$$\tilde{\Phi}(\vec{x}_i) = \Phi(\vec{x}_i) - \frac{1}{2} \sum_{k=1}^{m} \Phi(\vec{x}_k) \tag{12.9}$$

You get the components of the centred Kernel-Matrix $\tilde{\mathbf{K}}$ based on the old entries without knowing Φ:

$$\tilde{\mathbf{K}}_{i,j} = \mathbf{K}_{i,j} - \frac{1}{m} \sum_{k=1}^{m} \mathbf{K}_{i,k} - \frac{1}{m} \sum_{k=1}^{m} \mathbf{K}_{k,j} + \frac{1}{m^2} \sum_{k,l=1}^{m} \mathbf{K}_{k,l} \tag{12.10}$$

13 Laplacian SVM

This is an extension to Kernel SVMs and is state of the art in research.
The Laplacian SMV works on partially labeled training sets:
Training data $\mathcal{S} = \mathcal{L} \cup \mathcal{U}$

labeled data: $\mathcal{L} = \{(\vec{x}_i, y_i), \quad i = 1, ..., l\}$

unlabeled data: $\mathcal{U} = \{\vec{x}_i, \quad i = \underbrace{l+1, ..., m\}}_{u}$

The Graph Laplacian L associated with $\mathcal{S}$:

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{13.1}$$

where $\mathbf{W}$ is the adjacency matrix, this matrix tells you where the edges are. $\mathbf{D}$ is the diagonal Matrix, with the degree of each node $d_i i = \sum_{j=1}^{m} w_{ij}$, so $\mathbf{D}$ tells you how many edges leaving node i.
with $\mathbf{K}$ as the Kernel Matrix. The goal is the decision boundary:

$$f(\vec{x}) = [f(\vec{x}_i), i = 1, ..., m]^T$$

You can do this by minimizing the Loss-Function with certain constraints:

$$f^* = \underset{f \in \mathcal{H}_{||}}{\arg\min} \sum_{i=1}^{l} V(\vec{x}_i, y_i, f) + \gamma_A \|f\|_A^2 + \gamma_l \|f\|_I^2 \tag{13.2}$$

with the Loss function $V(\vec{x}_i, y_i, f)$:

- Squared loss function $y_i - f(\vec{x}_i))^2$ for Regularized Least Squares (RLS)
- Hinge loss function $\max[0, 1 - y_i f(\vec{x}_i)]$ for SVM

the regularization terms are:

- Ambient norm $\|\cdot\|_A$. It enforces a smoothness condition on the possible solutions
- Intrinsic norm $\|\cdot\|_I$.
  - norm of the function f in the low dimensional manifold

– enforces a smoothness along the sampled $\mathcal{M}$

The definition of the Intrinsic norm is:

$$\|f\|_I^2 = \sum_{i=1}^{m}\sum_{j=1}^{m} w_{ij}(f(\vec{x}_i) - f(\vec{x}_j))^2 = f^T\mathbf{L}f$$

which ensures points close to each other produce a similar result with f applied. The solution is:

$$f^*(\vec{x}) = \sum_{i=1}^{m} \alpha_i^* k(\vec{x}_i, \vec{x})$$

the primal optimization problem is:

$$\min_{\vec{\alpha}\in\mathbb{R}^d,\xi\in\mathbb{R}^l} \sum_{i=1}^{l} \xi_i + \gamma_{\mathbf{A}} \underbrace{\vec{\alpha}^T\mathbf{K}\vec{\alpha}}_{\text{original primal problem}} + \underbrace{\gamma_{\mathbf{L}}\vec{\alpha}^T\mathbf{KLK}\vec{\alpha}}_{\text{Intrinsic Norm}}$$

$$\text{subject to}\; y_i\left(\sum_{j=1}^{m} \vec{\alpha}_i k(\vec{x}_i, \vec{x}_j) + \alpha_0\right) \geq 1 - \xi_i, \qquad i = 1, ..., l$$

$$\xi_i \geq 0, \qquad i = 1, ..., l$$

Setting up the Lagrangian multiplier and computing the gradients w.r.t. $\alpha_0, \vec{\alpha}$ and ξ_i allows us to simplify the Lagrangian further:

$$\mathcal{L}(\vec{\alpha}, \vec{\lambda}) = \frac{1}{2}\vec{\alpha}^T\left(2\gamma_A\mathbf{K} + 2\gamma_I\mathbf{KLK}\right)\vec{\alpha} - \vec{\alpha}^T\mathbf{K}\mathbf{J}_{\mathcal{L}}^T\mathbf{Y}\vec{\lambda} + \sum_{i=1}^{l}\lambda_i \tag{13.3}$$

where $\mathbf{J}_{\mathcal{J}} = (\text{I}0) \in \mathbb{R}^{l\times m}$, i.e. a combination of a $(l \times l)$ identity matrix and a $(l \times u)$ zero matrix and $\mathbf{Y} \in \mathbb{R}^{l\times l}$ is a diagonal matrix composed by the l class labels y_i. Computing the partial derivative with respect to $\vec{\alpha}$ and substitute the expression in the equation above leads to this simplified equation:

$$\text{maximize} \qquad \sum_{i=1}^{l}\lambda_i - \frac{1}{2}\vec{\lambda}^T\mathbf{Q}\vec{\lambda} \tag{13.4}$$

$$\text{subject to} \qquad \sum_{i=1}^{l}\lambda_i y_i = 0, \quad 0 \leq \lambda_i \leq 1, \quad i = 1, ..., l \tag{13.5}$$

$$\tag{13.6}$$

$$\text{where} \qquad \mathbf{Q} = \mathbf{Y}\mathbf{J}_{\mathcal{L}}\mathbf{K}(2\gamma_{\mathbf{A}}\mathbf{I} + 2\gamma_{\mathbf{I}}\mathbf{KL})^{-1}\mathbf{J}_{\mathcal{L}}^T\mathbf{Y} \tag{13.7}$$

14 Expectation Maximization Algorithm

14.1 Parameter Estimation Methods

The expectation maximization method are developed to deal with

- high dimensional parameter spaces
- latent, hidden, incomplete data

Maximum likelihood estimation and maximum a-posteriori estimation are known parameter estimation techniques from statistics:

- Maximum likelihood estimation
  - All observations are assumed to be mutually statistically independent
  - the observations are kept fixed
  - the log-likelihood function is optimized regarding the parameters

$$\hat{\vec{\theta}} = \arg\max_{\vec{\theta}} p(\vec{x}_1, \vec{x}_2, ..., \vec{x}_m; \vec{\theta}) = \arg\max_{\vec{\theta}} \prod_{i=1}^{m} p(\vec{x}_i; \vec{\theta}) = \arg\max_{\vec{\theta}} \sum_{i=1}^{m} \log p(\vec{x}_i; \vec{\theta})$$

- Maximum a-posteriori estimation
  - the probability density function of the parameters $p(\theta)$ to be estimated is known

$$\hat{\vec{\theta}} = \arg\max_{\vec{\theta}} p(\vec{\theta}|\vec{x}) = \arg\max_{\vec{\theta}} \frac{p(\vec{\theta})(\vec{x}|\vec{\theta})}{p(\vec{x})} = \arg\max_{\vec{\theta}} \log p(\vec{\theta}) + \log p(\vec{x}|\vec{\theta})$$

14.2 Gaussian Mixture Models

so far, you have consider parameter estimation for statistical models with:

- one class-dependent distribution component
- uni- or multivariate feature vectors
- the type was mostly Gaussian (normally distributed features)

Now extend this model by representing the observations with a set of K multivariate Gaussian distributions, so called Gaussian Mixture Model (GMM)

$$p(\vec{x};\vec{\theta}) = \sum_{k=1}^{K} p_k \cdot p(\vec{x};\vec{\theta}_k) \qquad \text{subject to} \qquad \sum_{k=1}^{K} p_k = 1 \tag{14.1}$$

This is also a PDF because the Integral: $\int p(\vec{x};\theta)d\vec{x} = 1$.
Given m feature vectors in an d-dimensional space, find a set of K multivariate Gaussian distribution that best represent the observations. GMMs are an example of classification by **unsupervised learning**:

- it is not known which feature vector are generated by which of the K Gaussian
- the desire output is, for each feature vector, an estimate of the probability that it is generated by distribution k

$$p(k|\vec{x}_i) =: p_{ik}$$

is the probability that the feature vector $\vec{x}_i$ was generated by the distribution k. The variable which need to be estimated:

μ_k the K means

Σ_k the K covariance matrices of size $d \times d$

p_k fraction of all features in component k

$p(k|\vec{x}_i) \equiv p_{ik}$ the K probabilities for each of the m feature vectors $\vec{x}_i$

additional estimates:

$p(\vec{x})$ probability distribution of finding a feature at location $\vec{x}$

$\mathcal{L}$ overall log-likelihood function of the estimaed parameter set

The key to the estimation problem is the overall log-likelihood objective function:

$$\mathcal{L} = \sum_{i=1}^{m} \log p(\vec{x}_i)$$

split $p(\vec{x}_i)$ into its contributions from the K Gaussian.
This is the probability that a certain feature vector is generated:

$$p(\vec{x}_i) = \sum_{i=1}^{K} \overbrace{p_k \underbrace{\mathcal{N}(\vec{x}_i;\mu_k,\Sigma_k)}_{p(\vec{x}_i|k)}}^{p(\vec{x}_i,k)}$$

the individual probabilities for the K contributions are:

$$p_{ik} = p(k|\vec{x}_i) = \frac{p_k \cdot \mathcal{N}(\vec{x}_i; \mu_k, \Sigma)}{p(\vec{x}_i)} \tag{14.2}$$

If you know the μ_k, Σ_k and the mixture wheight p_k then you can estimate p_{ik}. Similar to the ML estimate for on Gaussian, we maximize the log-likelihood by deriving with respect to the unknown and end up with:

$$\hat{\vec{\mu}}_k = \frac{\sum\limits_i p_{ik}\vec{x}_i}{\sum\limits_i p_{ik}}$$

$$\hat{\Sigma} = \frac{\sum\limits_i p_{ik}(\vec{x}_i - \hat{\vec{\mu}}_k)(\vec{x}_i - \hat{\vec{\mu}}_k)^T}{\sum\limits_i p_{ik}}$$

$$\hat{p}_k = \frac{1}{m}\sum_{i=1}^{m} p_{ik}$$

This are very similar to the formulas where you have only one Gaussian, but in this case they Gaussians are weighted by p_{ik}. The probability that a feature vector $\vec{x}_i$ was generated by the distribution k is p_{ik} and instead saying $\vec{x}_i$ is from k you say x_i is with a certain probability from k and all probabilities summed up to One. This is called soft decision.
You need the parameter $\vec{\mu}_k, \Sigma_k$ and p_k to compute p_{ik}, this is the expectation step. After you computed p_{ik} with initialized parameters you can compute new values for these parameter with p_{ik}.
Instead of a closed form Solution this leads to an iterative scheme to compute the Gaussian mixture models, which holds right at the ML solution for both the expectation and the maximization step:

1. Initialize $\mu_k^{(0)} = 0, \Sigma_k^{(0)} = 0, p_k^{(0)} = 0$
2. Set $j := 0$, repeat
   a) Expectation step: compute new values for p_{ik}, L
   b) Maximization step: update values for $\mu_k^{(j)}, \Sigma_k^{(j)}, p_k^{(j)}$
   c) Set $j := j + 1$

   until L is no longer changing
3. Output: estimates $\hat{\mu}_k, \hat{\Sigma}_k, \hat{p}_k$

14.3 Hidden Information

Expectation Maximization is used when trying to model hidden information. We formulate the missing information principle as follows:

$$\text{observable information} = \text{complete information} - \text{hidden information}$$

The GMMs are such a case where you don't know which component generates which samples.
Write this in a more mathematical way:

- observable random variable X
- hidden random variable Y parameter set: $\vec{\theta}$

the joint probability density of the events x and y is:

$$p(x,y;\vec{\theta}) = p(x;\vec{\theta})p(y|x;\vec{\theta}) \Leftrightarrow p(x;\vec{\theta}) = \frac{p(x,y;\vec{\theta})}{p(y|x;\vec{\theta})}$$
$$\underbrace{-\log p(x;\vec{\theta})}_{\text{observable-}} = \underbrace{-\log p(x,y;\vec{\theta})}_{\substack{\text{complete-}\\ \text{-information}}} - \underbrace{(-\log p(y|x;\vec{\theta}))}_{\text{hidden-}}$$

The key to find a solution for this is marginalization!

$$\int p(\vec{x},y;\vec{\theta})dy = p(x;\vec{\theta}) \tag{14.3}$$

You can design an iterative scheme to estimate the parameters.
Consider the key equation (i+1)-st iteration:

$$\log p(x;\hat{\vec{\theta}}^{(i+1)}) = \log p(x,y;\hat{\vec{\theta}}^{(i+1)}) - \log p(y;\hat{\vec{\theta}}^{(i+1)}) \tag{14.4}$$

multiply both sides with $p(y|x;\hat{\vec{\theta}}^{(i)})$ and integrate over the hidden event y. On the left side this changes nothing, this is a Integral over a PDF so its 1 and the other term is independent from y so its a factor in front of the integral.
The left Term on the right-hand side:
This is the Kulback-Leiber statistic (also called Q-function), with respect to θ' given θ this is the conditional expectation:

$$Q(\theta,\theta') = E[\log p(x,y;\theta')|x,\theta] = \int p(y|x;\vec{\theta}) \log p(x,y;\vec{\theta}')\mathrm{d}y \tag{14.5}$$
$$\tag{14.6}$$

The right Term on the right hand side is called Entropy $H(\vec{\theta}, \vec{\theta}')$. For the Entropy the following inequation holds:

$$H(\vec{\theta}; \vec{\theta}') \geq H(\vec{\theta}; \vec{\theta}) \tag{14.7}$$

cause of the, you don't care about the Entropy in me maximization of this rewritten key equation from the EM-Algorithm:

$$\log p(x; \hat{\vec{\theta}}^{(i+1)}) = Q(\vec{\theta}^{(i)}; \hat{\vec{\theta}}^{(i+1)}) + H(\vec{\theta}^{(i)}; \hat{\vec{\theta}}^{(i+1)}) \tag{14.8}$$

14.4 Expectation Maximization Algorithm

Instead of maximizing the log-likelihood function on the left side of the key-equation, you maximize the Kullback-Leibler statistics iteratively while ignoring the entropy term:

1. Initialize $\hat{\theta}^{(0)}$
2. Set $i := -1$. Repeat
   a) Set $i := i + 1$
   b) Expectation step:
   $$Q\left(\hat{\theta}^{(i)}; \theta\right) = \int p\left(y|x; \hat{\theta}^{(i)}\right) \log p(x, y; \theta) \mathrm{d}y$$
   c) Maximization step:
   $$\hat{\theta}^{(i+1)} = \arg\max_{\theta} Q\left(\hat{\theta}^{(i)}; \theta\right)$$

   until $\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)}$.
3. Output: estimate $\hat{\theta} := \hat{\theta}^{(i)}$

The maximum of the KL-statistics is usually computed using zero crossings of the gradient.The iteration scheme is numerically robust and has constant memory requirements.
The expectation maximization algorithm however also has a few drawbacks; it converges very slowly. It also only is a local optimization method, i.e. the initialization heavily influences the results.

14.5 Constrained Optimization

Many optimization problems that are usually solved by the EM algorithm are of the following form:

$$\begin{aligned} \text{optimize} \quad & f_0(p_1, p_2, \dots, p_K) = \sum_{k=1}^{K} a_k \log p_k \\ \text{subject to} \quad & \sum_{k=1}^{K} p_k = 1 \\ & p_k \leq 0 \end{aligned}$$

We apply the Lagrange multiplier method

$$\mathcal{L}(p_1, p_2, \dots, p_K, \mu) = \sum_{k=1}^{K} a_k \log p_k + \mu \left(\sum_{k=1}^{K} p_k - 1 \right)$$

and compute the derivative

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial p_k} = \frac{a_k}{p_k} + \mu \overset{!}{=} 0 \\ a_k = -\mu p_k \end{aligned}$$

The p_ks however, are unknown. To get μ, we sum both sides over all k:

$$\mu = -\sum_{k=1}^{K} a_k$$

We can plug this in for μ and get an estimator for p_k:

$$\hat{p}_k = \frac{a_k}{\sum_{l=1}^{K} a_l}$$

15 Independent Component Analysis

An Example to come into the Topc:
Cocktail-Party Problem: Imagine two microphones in a room at different location which record the time signals $x_1(t), x_2(t)$. Each recorded signal is a weighted sum of two speakers $s_1(t), s_2(t)$:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t)$$
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t)$$

where the parameters a_{ij} depends on th distance of the microphones to the speakers. Without knowing a_{ij} it's not easy to solve these linear equations.
The core idea to solve this problem is to use information about the statistical properties of the signals $s_i(t)$ to estimate a_{ij}. It is sufficient to assume that the $s_i(t)$ are **statistically independent** at each time point t.

15.1 Latent Variables and Factor Analysis

Rewrite the time series into n linear mixture observations $x_1, ..., x_n$. Each mixture x_i as well as each component s_j are random variables:

$$x_i = \sum_{j=1}^{m} a_{ij}s_j, \qquad i = 1, ..., n \tag{15.1}$$

in matrix notation:

$$\vec{x} = \mathbf{A}\vec{s} \tag{15.2}$$

where $\mathbf{A}$ is a constant mixing matrix, s_j are latent random variables (independent components) and both $\mathbf{A}$ and s_j have to be estimated based on observation x_i.
The first step is decorrelation ! Assuming we have de-meaned data $\bar{\vec{x}} = 0$. For decorrelation you use the approach from the LDA. If the covariance matrix is the Identity Matrix then the data are decorrelated! To get the Covariance Matrix you

compute $\frac{1}{n}\sum_i \vec{x}_i\vec{x}_i^T$ which gives you a symmetric Matrix which means you have a easier SVD:

$$\Sigma = \mathbf{U}\mathbf{D}\mathbf{U}^T$$

do the same trick to get a Identity Matrix:

$$\Sigma^{-1} = (\mathbf{U}\mathbf{D}^{-\frac{1}{2}})\cdot\mathbf{I}\cdot(\mathbf{U}\mathbf{D}^{-\frac{1}{2}})^T$$

which gives you the mapping:

$$\tilde{\vec{x}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{U}^T\vec{x}$$

The mapped random variables $\tilde{x}_i$ are uncorrelated and the covariance matrix is the identity.
This is called Whitening Transform.
You could interpret the mapped random variable $\tilde{\vec{x}}$ as an estimate of the latent variable model

$$\vec{s} = \tilde{\vec{x}} \tag{15.3}$$

but this gives you a poor result.
The Problem is, the whitening transform is not unique, because it can be rotated, or in other words you can apply any orthogonal matrix and it will not change the statistical properties.
As the second step is Independence!
Now use the independency assumption. For two independent random variables $y_{1,2}$ the following computation of the Expectation holds:

$$E\{h_1(y_1)h_2(y_2)\} = E\{h_1(y_1)\}E\{h_2(y_2)\}$$

this assumed independency allows you to identify the elements of $\mathbf{A}$ uniquely. Note that any Gaussian independent components can be determined only up to a rotation, but we want a unique solution, therefore, you assume that the s_i are independent and **non**-Gaussian.
The whitening transform is usually done before ICA as a pre-processing step and its transform the mixing Matrix $\mathbf{A}$ to $\tilde{\mathbf{A}}$:

$$\tilde{\vec{x}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{U}^T\mathbf{A}\vec{s} = \tilde{\mathbf{A}}\vec{s}$$

the new mixing Matrix is orthogonal and has $\frac{n(n-1)}{2}$degrees of freedom, while $\mathbf{A}$ had n^2.
Writing the ICA model in terms of the columns of $\mathbf{A}$:

$$\vec{x} = \sum_{i=1}^{n}\vec{a}_i s_i \tag{15.4}$$

Any scalar multiplier for s_i can be eliminated by scaling $\vec{a}_i$ appropriately. The matrix $\mathbf{A}$ can be adapted to restrict the s_i to have unit variance. This does not fix the ambiguity of the sign: multiplying by ± 1 does not affect the model - however this is insignificant in most applications.
Another ambiguity is the order of the summation, which we can formalize using a permutation matrix P:

$$\vec{x} = \underbrace{\mathbf{AP}}_{\mathbf{A}^*}\underbrace{\mathbf{P}^{-1}\vec{s}}_{\vec{s}^*} \tag{15.5}$$

$\mathbf{A}^*$ is just a new mixing matrix to be solved.

15.2 Basic Principle of ICA

If you knew A, you could compute its inverse A^{-1} to obtain the independent components:

$$\vec{s} = A^{-1}\vec{x}$$

That would lead you to s_i being a linear combination of x_i with a weight vector $\vec{w}$, which is a row of $\mathbf{A}$:

$$s_i \overset{?}{=} y = \vec{w}^T\vec{x}$$

so y equals one of the independent components if $\vec{w}$ is one of row of $\mathbf{A}^{-1}$.
You can rewrite this:

$$y = \vec{w}^T\vec{x} = \vec{w}^T\underbrace{\mathbf{A}\vec{s}}_{\vec{x}} = \vec{z}^T\vec{s}; \qquad \text{where } \vec{z} = \mathbf{A}^T\vec{w}$$

The result of the central limit theorem is, that the tum of number of independent random variables tends toward a normal distribution.
$\vec{z}\vec{s}$ is more Gaussian than any of the s_i.
The implication of this is that $\vec{z}^T\vec{s}$ is least Gaussian exactly when it is one of the s_i

The Key principle of ICA is:

Maximizing the non-Gaussianity of $\vec{w}^T\vec{x}$ results in the independent components!

This is also a reason why you assumed non-Gaussianity of the components. Note that if just one of the components is Gaussian, independent analysis will still work. The Gaussian is the most random distribution, therefore it is the least informative pdf!

The randomness of a pdf can be measured using entropy. The entropy $H(X)$ of a continuous random Variable

$$X$$

with a density of $p(x)$ is defined as:

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

The Gaussian is the distribution with the highest entropy given a certain mean and a certain covariance.

15.2.1 ICA Estimation Algorithm

With all this knowledge you can build an algorithm:

1. Apply centering transform
2. Apply whitening transform
3. Set $i := 1$ and repeat
   - Take a random vector $\vec{w}_i$
   - Maximize non-Gaussianity of $\vec{w}_i^T$ subject to $\|[\|]\vec{w}_i = 1$ and $\vec{w}_j^T \vec{w}_i$ (where $j < i$)
   - Increase i

   until $i > n$ (where n is the number of independent components)
4. Use $W = \left(\vec{w}_1^T, \vec{w}_2^T, \ldots, \vec{w}_n^T\right)$ to compute $\vec{s}$
5. Output: independent components $\vec{s}$

You need a measurement for non-Gaussianity to do this.

15.3 Measures of Non-Gaussianity

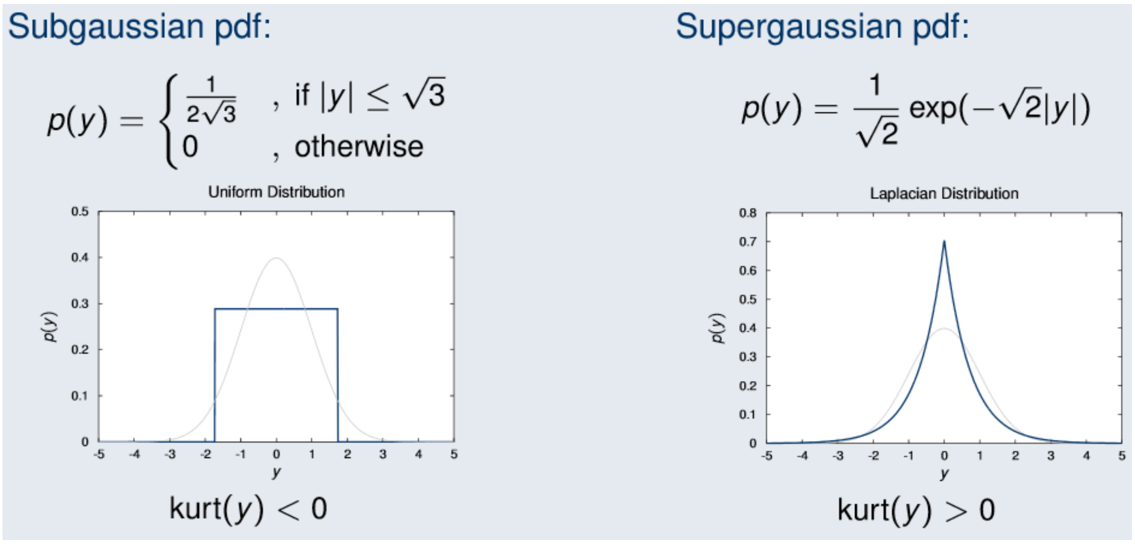
Consider three measures of non-Gaussianity, the Kurtosis, the Negentropy and the Mutual Information.
Note that we assume zero mean and the Identity Matrix as the covariance matrix (spherical data).

15.3.1 Kurtosis

The Kurtosis of a Gaussian is Zero.
If the random variable y is normally distributed then:

$$\mathrm{kurt}(y) = 0$$

To find a function that is as non-Gaussian as possible we can thus maximize $\|\mathrm{kurt}(y)\|$. The Kurtosis can be positive or negative (15.3.1) Some drawbacks are that the Kur-



tosis can be very sensitive to outliers, its not optimal for supergaussian variables. Its not a robust measure for non-Gaussianity

15.3.2 Negentropy

15.3.2.1 Negentropy

The Negentropy $J(y)$ of a random variable is defined as

$$J(y) = H(y_{\text{Gauss}}) - H(y)$$

where y_{Gauss} is a Gaussian random variable of the same covariance as y. If $J(y) = 0$ then y is Gaussian-distributed. In theory, negentropy is an optimal statistica estimator of non-Gaussianity. But Computing the negentropy from a measures set of samples requires the estimation of the pdf. The (non-parametric) estimation of a pdf from samples is however non-trivial and computationally expensive. Instead, there are approximations for negentropy.

15.3.3 Mutual Information

Negentropy measures the difference to a Gaussian random variable. Instead, we can measure statistical dependency between two random variables directly using *mutual information*.
Instead of maximizing the negentropy, we can minimize the Mutual Information to compute the direction of the highest non-Gaussianity.

Negentropy and Mutual Information are equivalent under certain conditions!

16 Model Assessment

16.1 No Free Lunch

Are there any reason to favor one algorithm over another? The no free lunch theorem states that the sum over all cost functions given two algorithms is equal, i.e. an algorithm might perform well regarding a specific cost function, but no algorithm will perform well according to all possible cost functions.
The consequences of the Theorem are:

- if no prior assumptions about the problem are made, there is **NO** overall superior or inferior classification method!
- if an algorithm achieves superior results on some problems, it must pay with inferiority on other problems
- We have to focus on the aspects that matter most for the classification problem at hand, e.g. prior information, the data distribution, the amount of training data and/or cost functions.

So there is no general best classifier !

16.2 Off-Training Set Error

Specifies the error on samples that are not contained within the training set. For large training data sets, the off-training set is necessarily small. We can use the off-training set error to compare general classification performance of algorithms.
Consider a two class problem with training data set $S = \{(\vec{x}_i, y_i); i = 1....m)\}$ with two classes $y \in \{-1, +1\}$.
The class labels y_i are generated by an underlying target function $y_i = F(\vec{x}_i)$. This unknown function contains a random component in most cases of interest: the same input $\vec{x}$ could lead to different class labels y.
classification model $h(\vec{x}))$ described by the set of parameters $h\ in \mathcal{H}$.
For stochastic classification methods is $p(h|S)$ a probability mass function that th algorithm trained in S leads to the trained model h.
For deterministic learning algorithms $p(h|S)$ is 1 for exactly one trained model h.

expected off-training set classification error for learning algorithm A_k and a given true target function $F(\vec{x})$:

$$E_k\{e|F,S\} = \sum_{h\in\mathcal{H}} \sum_{\vec{x}\notin S} p(\vec{x}) \left[1 - \underbrace{\delta(F(\vec{x}), h(\vec{x}))}_{=1 \text{ if } F(\vec{x})=h(\vec{x})} \right] p_k(h|S) \tag{16.1}$$

16.3 Bias and Variance

You can analyse the bias-variance relation to assess the quality of a learning algorithm in terms of the alignment to the given problem.

Bias The bias measures the accuracy of the match. In other words Bias is a systematic error:

high bias means poor match

Variance The variance measures the precision of specificity for the match. In other words variance is how things jump back and forth:

high variance implies a weak match

The bias-variance relation is very demonstrative in regression: Let $g(\vec{x})$ be the regression function. The mean-square deviation from the true function $F(\vec{x})$ is:

$$E\left\{(g(\vec{x}) - F(\vec{x}))^2\right\} = \underbrace{E\left\{g(\vec{x}) - F(\vec{x})\right\}^2}_{(\text{bias})^2} + \underbrace{E\left\{(g(\vec{x}) - E\left\{g(\vec{x})\right\})^2\right\}}_{\text{variance}}$$

This formula shows you the bias-variance trade-off. Methods with high flexibility to adapt to the training data generally have low bias, but yield a high variance. Methods with few parameters and less degrees of freedom tend to have a high bias, as they may not fit the data well, but have a lower variance. We can virtually never get both zero bias and zero variance. You can reduce both values using as much prior information as possible.

We can apply the same to a two-class classification problem. We cannot compare $g(\vec{x})$ and $F(\vec{x})$ based on the mean-square error as in regression, because the classification results are discrete variables. For simplicity, let you assume identical priors. We can get a similar result where the bias and variance are multiplied together (vs summation in regression).

16.4 Estimating and Comparing Classifiers

How do we determine bias and variance for some learning algorithm applied to a new problem with unknown distributions?

Suppose we want to estimate a parameter θ that depends on a random sample set $X = (x_1, \ldots, x_n)$. Assume we have an estimator e.g. $\phi_n(X) = \frac{1}{n}\sum\limits_{i=1}^{n} x_i)$ for θ but do not know its distribution.
Resampling methods then try to estimate the bias and variance of $\phi_n(X)$ using subsamples from X.

16.4.1 Jackknife

Let $PS_i(X)$ be the i-th pseudovalue of $\phi_n(X)$:

$$\begin{aligned} PS_i(X) &= n\phi_n(X) - (n-1)\phi_{n-1}(X_{(i)}) \\ &= \phi_n(X) - \underbrace{(n-1)(\phi_{n-1}(X_{(i)}) - \phi_n(X))}_{\text{bias}_{\text{jack}}} \end{aligned}$$

where $X_{(i)} = (x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ is the set without the i-th element. $\text{PS}_i(X)$ can be interpreted as a bias-corrected version of $\phi_n(X)$, because the bias trend is assumed to be in the estimators from $\phi_{n-1}\left(X_{(i)}\right)$ to $\phi_n(X)$. We can treat the pseudovalues as independent random variables and estimate their mean μ_{PS} and variance σ^2_{PS} using maximum likelihood estimators.

16.4.2 Bootstrap

A bootstrap data set is created by randomly selecting n points from the sample set with replacement. In bootstrap estimation this selection process is independently repeated B times. The B bootstrap data sets are treated as independent sets.

The bootstrap estimate of a statistic θ and its variance are the mean of the B estimates $\hat{\theta}^B$ and its variance:

$$\begin{aligned} \mu_{\text{BS}} &= \frac{1}{B}\sum_{i=1}^{B} \hat{\theta}_i^B \\ \sigma^2_{\text{BS}} &= \frac{1}{B-1}\sum_{i=1}^{B} \left(\hat{\theta}_i^B - \mu_{\text{BS}}\right)^2 \end{aligned}$$

The bias is the difference between the bootstrap estimate and the estimator $\phi_n(X)$:

$$\text{bias}_{\text{BS}} = \mu_{\text{BS}} - \phi_n(X)$$

Bootstrapping does not change the priors, because we choose with replacements. The larger B the more the bootstrap estimate will tend towards the true statistic θ. In contrast to the jackknife, the bootstrap does not require exactly n repetitions.

16.4.3 Estimating and Comparing Classifiers

In cross-validation, the training samples are split into two disjoint parts:

- the first set is the training set used for the traditional training
- the second set is the test set used to estimate the classification error
- in a second step, both sets are swapped
- by that, the classification error can be estimated on the complete data set
- yet, training and test set are always disjoint

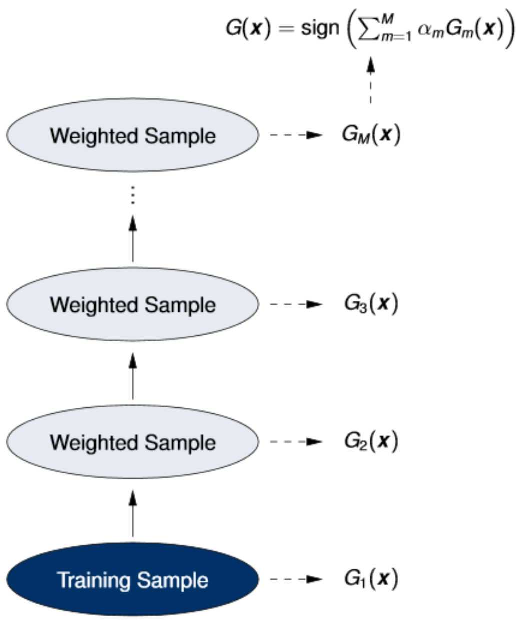
An m-fold cross-validation splits the data into m disjoint sets of size $\frac{n}{m}$, where one set is used as test set and the other $m-1$ sets are used for training. Cross-validation degrades to the jackknife for $m = n$.

17 AdaBoost

The core Idea is to combine many weak classifiers.

A weak classifier is one whose error rate is only slightly better than random guessing.

After the first classification, the missclassified features get a higher priority ("Weighted Sampleïn 17): The Algorithm-schema is:



1. Initialize weights $w_i = \frac{1}{N}$
2. Set $m := 1$, repeat
   - Fit classifier $G_m(\vec{x})$ to training data using $\vec{w}$
   - Compute classification error

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G(_m(x_i)))}{\sum_{i=1}^{N} w_i}$$

- Compute classifier weights

$$\alpha_m = \log\left(\frac{1-\mathrm{err}_m}{\mathrm{err}_m}\right)$$

- Compute new sample weights

$$w_i \leftarrow w_i \exp\left(\alpha_m I(y_i \neq G_m(x_i))\right)$$

- Increase m

until $m = M$

3. Output: $G(x) = \operatorname{sgn}\left(\sum_{i=1}^{M} \alpha_m G_m(\vec{x})\right)$

18 Mathematical Methods

18.1 Maximum Likelihood Estimation

Used to Estimate Parameters. If we assume that the class conditional density function $p(\vec{x}|y)$ is a Gaussian $\mathcal{N}(\vec{x};\vec{\mu},\Sigma)$. And we don't know $\vec{\mu}$ and Σ, we can estimate them with the Maximum Likelihood.

Given a observed random variable X with a number of samples $\vec{x}_1,\vec{x}_2,...,\vec{x}_n \in \mathbb{R}^d$, find parameters θ such that the log likelihood function is maximized:

$$\hat{\theta} = \arg\max_{\theta} p(\vec{x};\theta) = \arg\max_{\theta} \log p(\vec{x};\theta) \tag{18.1}$$

The likelihood function is $\mathcal{L}(\theta) = \log p(\vec{x};\theta)$ For the estimation of the parameters from the Gaussian the parameter $\theta = (\vec{\mu},\Sigma)$. You can setup the likelihood function as follow(note that the assumption that $\vec{x}_k$ are mutually independent is used):

$$\mathcal{L}(\vec{\mu},\Sigma) = \sum_{k=1}^{n} \log p(\vec{x}_k;\vec{\mu},\Sigma) \tag{18.2}$$

now you can insert the formula for the Gaussian $\mathcal{N}(\vec{x};\vec{\mu},\Sigma)$:

$$\mathcal{L}(\vec{\mu},\Sigma) = \sum_{k=1}^{n} -\frac{1}{2}\log(\det(2\pi\Sigma)) - \frac{1}{2}(\vec{x}_k-\vec{\mu})^T\Sigma^{-1}(\vec{x}_k-\vec{\mu}) \tag{18.3}$$

To maximize the function $\mathcal{L}(\vec{\mu},\Sigma)$ its often necessary to use a numerical approach. In this case there are existing close form solutions for the parameter $\vec{\mu}$ and Σ. Just compute the gradient with respect to $\vec{\mu}$ / Σ and set it to zero.
Then you end up with:

$$\vec{\mu} = \frac{1}{n}\sum_{k=1}^{n}\vec{x}_k \qquad \Sigma = \frac{1}{n}\sum_{k=1}^{n}(\vec{x}_k-\vec{\mu})(\vec{x}_k-\vec{\mu})^T \tag{18.4}$$

18.2 Constrained Optimization

Consider the primal optimization problem:

$$\mathrm{f}_0(\vec{x}) \tag{18.5}$$

$f_0(\vec{x})$ is not necessarily convex!
with constraints:

$$f_i(\vec{x}) \leq 0, i = 1, 2, ..., m$$
$$h_i(\vec{x}) = 0, i = 1, 2, ..., p$$

To optimize a function with given constraints you have to use Lagrange-multiplier:

$$L(\vec{x}, \vec{\lambda}, \vec{\nu}) = f_0(\vec{x}) + \underbrace{\sum_{i=1}^{m} \lambda_i f_i(\vec{x})}_{\text{sum over all inequality constraints}} + \underbrace{\sum_{i=1}^{p} \nu_i h_i(\vec{x})}_{\text{sum over all equality constraints}} \tag{18.6}$$

You add your constraint and weight them by multipliers λ, ν (dual variables). The optimization of $L(\vec{x}, \vec{\lambda}, \vec{\nu})$ is the optimization of $f_0(\vec{x})$ with respect to the constraints. For ν there is no restriction but for λ there is the constraint : $\lambda \geq 0$
The Lagrange dual function is defined as the infimum of L:

$$g(\vec{\lambda}, \vec{\nu}) = \inf_{\vec{x}} \left(f_0(\vec{x}) + \sum_{i=1}^{m} \lambda_i f_i(\vec{x}) + \sum_{i=1}^{p} \nu_i h_i(\vec{x}) \right) \tag{18.7}$$

This dual function is a pointwise affine function (which means after taking the $\inf_{\vec{x}}$ then $\vec{x}$ is fix) in the dual variables.
This resulting dual function is always concave!

If $f(\vec{x}*) = p^*$ is our optimal Value of the original constrained optimization problem then the maximum of the dual function g is equal or below p^*:

$$g(\vec{\lambda}, \vec{\nu}) \leq p^*; \qquad \vec{\lambda} \geq 0 \tag{18.8}$$

Optimizing the dual function g gives you the best lower bound for the primal optimization problem:

$$\begin{array}{ll} \text{maximize} & g(\vec{\lambda}, \vec{\nu}) \\ \text{subject to} & \vec{\lambda} \geq 0 \end{array}$$

Let d^* the optimal value of the dual function:

- the difference $p^* - d^*$ is called **optimal duality gap**
- if $p^* = d^*$ then its called **strong duality**. $p^* = f_0(\vec{x}^*) = g(\vec{\lambda}^*, \vec{\nu}^*) = d^*$
- if $p^* > d^*$ then its called **weak duality**

The duality gap is zero when Slater's Condition is fulfilled:
If the primal problem $f_0(\vec{x})$ is convex and all inequalities are strikt smaller than 0 then the duality gap is 0.

If $\vec{x}^*$ the optimal point for the primal problem and $\vec{\lambda}^*, \vec{\nu}^*$ are the optimal points for the dual Problem with a zero duality gap. The gradient of L at these optimal point has to be zero:

$$\nabla L(\vec{x}^*, \vec{\lambda}^*, \vec{\nu}^*) = \nabla f_0(\vec{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(\vec{x}^*) + \sum_{i=1}^{p} \nu_i^* \nabla h_i(\vec{x}^*) \overset{!}{=} 0 \tag{18.9}$$

The **K**arush-**K**uhn-**T**ucker Conditions are:

1. Primal constrains:

$$f_i(\vec{x}) \leq 0, \quad i = 1, 2, ..., m$$
$$h_i(\vec{x}) = 0, \quad i = 1, 2, ..., p$$

2. Dual constraints: $\vec{\lambda} \geq 0$
3. Complementary slackness: $\lambda_i \cdot f_i(\vec{x}) = 0$. The most Important for SVMs
4. Gradient of Lagrangian is zero (as shown in 18.9)

The 4 condition is necessary and the most important condition is the Complementary slackness.

For **any optimization problem** with differentiable objective and constraint functions for which strong duality holds: then you know any pair of primal$\vec{x}^*$ and dual optimal points $\vec{\lambda}^*, \vec{\nu}^*$ **must satisfy** the KKT-Conditions.
In the other direction this means: any pair of primal$\vec{x}^*$ and dual optimal points $\vec{\lambda}^*, \vec{\nu}^*$ which satisfy the KKT-Conditions then these points are candidates for the points we are looking for.

For any **convex optimization problem** with differentiable objective and and constraint functions, any points that satisfy the KKT conditions are primal and dual optimal, and **have** zero duality gap.

19 Examples

19.1 Training of Bayesian Classifier with Gaussian ccpdf

We have a training set $S = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_n, y_n)\}$ with $y \in \{0, 1\}$.
we assume that:

- the class conditional $p(\vec{x}|y)$ are Gaussian
- the covariance matrices are the same $\Sigma = \Sigma_0 = \Sigma_1$

Problem: Train the classifier based on S.

$p(y|\vec{x}) \cong p(y) \cdot p(\vec{x}|y)$ (move from the discriminative to the generative model)

1. Step:
Estimate the priors:
$\hat{p}(y = 0)$ and $\hat{p}(y = 1)$
with

$$\hat{p}(p = 0) = \frac{1}{N} \sum_{i=1}^{N} \chi(y_i = 0), \qquad \text{where } \chi(x) = \begin{cases} 0, & \text{if } x \text{ is false,} \\ 1, & \text{if } x \text{ is true.} \end{cases} \tag{19.1}$$

$$\hat{p}(y = 1) = 1 - p(y = 0) \tag{19.2}$$

2. Step:
Estimate the $\vec{\mu}$ for the class conditional Gaussians $\mathcal{N} = (\vec{x}; \vec{\mu}, \Sigma)$.

For the estimation of the mean Vectors $\vec{\mu}$ use the ML-estimate.

$$\vec{\mu}_1 = \underset{\vec{\mu}_0}{\arg\max} \log \prod_{\substack{i = 0 \\ y_i = 0}}^{N} p(\vec{x}_i | y_i = 0) \tag{19.3}$$

the product is limited to these sample out of the dataset s which have the class label $y = 0$.
the following notation describe the number of samples belonging to the same class

(here $y = 0$).
$\#\{(\vec{x}_i, y_i); y_i = 0\}$
Then you end uo with this formula:

$$\vec{\mu}_0 = \frac{1}{\#\{(\vec{x}_i, y_i); y_i = 0\}} \sum_{\substack{i=0 \\ y_i=0}}^{N} \vec{x}_i \tag{19.4}$$

$\vec{\mu}$ is the mean vector of class $y = 0$ and can be interpreted as the centroid of the class.

3. Step:
Estimate the covariance matrix Σ
for the following formulas x_k denotes all feature vectors which belong to class $y = 0$, and k is the number of that samples.

$$\Sigma_0 = \frac{1}{K} \sum_k \vec{x}_k \cdot \vec{x}_k^T \tag{19.5}$$

19.2 Training Logistic Regression Using Maximum Likelihood Estimation

We have a training set $S = \{(\vec{x}_i, y_i); i = 1....n)\}$ with two classes $y \in \{0, +1\}$.
Assume $F(\vec{x})$ to be parametric function and estimate the parameters using ML.
Known:

$$p(y_i|\vec{x}_i) = \frac{1}{1 + e^{\pm F(\vec{x})}} \tag{19.6}$$

We assume $F(\vec{x})$ is a linear decision boundary.
Then we have this parametric form:

$$F(\vec{x}) = a_0 + a_1\vec{x}_1 + a_2\vec{x}_2 \tag{19.7}$$

Know we need to estimate the three parameters a_0, a_1, a_2 :

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \vec{a} = \arg\max_{\vec{a}} \log \prod_{i=1}^{N} p(y_i|\vec{x}_i) \tag{19.8}$$

and now plug in for the posterior the parametric form of $F(\vec{x})$ and this will be maximized.
(Hint: the log is cause of numerical problem on a computer multiplying a lot of numbers between 0 and 1.)

For the optimisation we reformulate the $F(\vec{x})$ in the sigmoid function as a inner product of $\vec{\theta}$ and $\vec{x}$ (like in 3.2.3).

$$p(y_1|\vec{x}) = g(\vec{\theta}^T\vec{x}) = \frac{1}{1+e^{-\vec{\theta}^T\vec{x}}}, \qquad \text{and} \qquad p(y_0|\vec{x}) = 1 - g(\vec{\theta}^T\vec{x}) = \frac{1}{1+e^{\vec{\theta}^T\vec{x}}} \tag{19.9}$$

with this formulation we can write our posteriori probability using our class number $\{0, +1\}$ as follows:

$$p(y|\vec{x}) = g(\vec{\theta}^T\vec{x})^y(1 - g(\vec{\theta}^T\vec{x}))^{1-y} \tag{19.10}$$

To optimize our sigmoid function we set up the log-likelihood function $\mathcal{L}(\vec{\theta})$:

$$\mathcal{L}(\vec{\theta}) = \log \prod_{i=1}^{n} p(y_i|\vec{x}_i) \tag{19.11}$$

now use the formula and replace the posteriori with 19.10 and simplify it:

$$\mathcal{L}(\vec{\theta}) = \sum_{i=1}^{n} y_i\vec{\theta}^T\vec{x}_i + \log(1 - g(\vec{\theta}^T\vec{x}_i)) \tag{19.12}$$

$\mathcal{L}(\vec{\theta})$ needs to be maximized. There exist no close form solution.
To find the maximum we use numerical methods, e.g. Newton-Raphson algorithm.
Here is the iterative scheme of the Newton-Raphson method:

$$\vec{\theta}^{k+1} = \vec{\theta}^{k} - \underbrace{\left(\frac{\delta^2}{\delta\vec{\theta}\delta\vec{\theta}^T}\mathcal{L}(\vec{\theta}^k)\right)^{-1}}_{\text{Inverse of the Hessian Matrix}} \frac{\delta}{\delta\vec{\theta}}\mathcal{L}(\vec{\theta}^k) \tag{19.13}$$

19.3 Naive Bayes and Guassians

Assume $p(\pm 1) = \frac{1}{2}$, and the covariance matrices are both the identity matrix. $\Sigma_{-1} = \Sigma_{+1} = 1$
classification:

$$\begin{aligned} y^* = \arg\max_y p(y|\vec{x}) = \arg\max_y p(\vec{x}|y) \\ = \arg\max_y \log(\frac{1}{\sqrt{\det 2\pi 1}}) - \frac{1}{2}(\vec{x} - \vec{\mu}_y)^T 1(\vec{x} - \vec{\mu}_y) \\ = \arg\min_y (\vec{x} - \vec{\mu}_y)^T(\vec{x} - \vec{\mu}_y) \end{aligned} \tag{19.14}$$

The interpretation is:
A Naive Bayes that use the identical matrix for the covariance matrices is a **nearest neighbor Classifier**.

the reference vectors for the nearest neighbor are the mean vectors of the classes.
If you have feature vectors with independent components and the classes are normally distributed with a variance of 1 then the nearest neighbour classifier is the best classifier with respect to the 0/1-Loss function.

19.4 Note to PCA

You have 50 Images with $2^{10} \times 2^{10}$ Pixels with 32 bit each Pixel. The feature vector have:

$$\vec{x} \in \mathbb{R}^{2^{10} \cdot 2^{10}} \tag{19.15}$$

if we try to use PCA we need the eigenvectors and eigenvalues.To get that we need the covariance matrix:

$$\Sigma = \frac{1}{50} \sum_i \vec{x}\vec{x}^T \qquad \in \mathbb{R}^{2^{20} \times 2^{20}} \tag{19.16}$$

This way leads to a Matrix which is way too big to store in a Computer.

19.5 Roseblatts Perceptron

We have a training set $S = \{(\vec{x}_i, y_i); i = 1....m)\}$ with two classes $y \in \{-1, +1\}$.
initialization for the iterative scheme:
$k = 0$; $\alpha_0^{(0)} = 0$ and $\vec{\alpha}^{(0)} = 0$

19.6 Classical vs Kernel PCA

Consider $m = 50$ Images with 1024^2 pixels. The pixels define 1024^2-dimensional feature vectors $\vec{x}_1, \vec{x}_2, ..., \vec{x}_{50} \in \mathbb{R}^{2^{20}}$ The Scattermatrix for the classic PCA would be:

$$\Sigma = \frac{1}{50} \sum_{i=1}^{50} \vec{x}_i \cdot \vec{x}_i^T \tag{19.17}$$

if you say each entry is an int value with 4 Byte:

$$4\text{Byte} \cdot 2^{20} \cdot 2^{20} = 2^{42}\text{Bytes} = 4\text{Tera-Byte} \tag{19.18}$$

the size of the scatter matrix for this small images is too large. So Classical PCA don't work here !
Lets have a look on the Kernel PCA.

The Kernel-Matrix have the dimension of $\mathbf{K} \in \mathbb{R}^{50 \times 50}$. And this you can handle on a Computer.

19.7 String Kernels

In speech recognition we do not have feature vectors but sequences of feature vectors. In order to use kernel methods we need a kernel for time series. Feature vectors are considered in $\mathbb{R}^d = \chi$. Sequences of feature vectors are elements of χ^*. How can we define a kernel over the sequence space χ^*? If we can define such a kernel, we can use PCA and SVM on those sequences.

We compare these sequences using dynamic time warping. Given the feature sequences ($p, q \in \{1, 2, \ldots\}$):

$$\langle \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_p \rangle \in \chi^*$$
$$\langle \vec{y}_1, \vec{y}_2, \ldots, \vec{y}_q \rangle \in \chi^*$$

We want to map the features denoted by x on to the features denoted by y given the constraint that $\vec{x}_1 \mapsto \vec{y}_1$ and $\vec{x}_p \mapsto \vec{y}_q$, i.e. the top left and the bottom right element in matrix form correspond to each other. This can be solved using dynamic programming.

The distance between the two signals is given by DTW:

$$D(\langle \vec{x}_1, \ldots, \vec{x}_p \rangle, \langle \vec{y}_1, \ldots, \vec{y}_q \rangle) = \frac{1}{p} \sum_{k=1}^{p} ||\vec{x}_{v(k)} - \vec{y}_{w(k)}||_2$$

where v, w define the mapping of indices. The DTW kernel can be defined as

$$k(\vec{x}, \vec{y}) = e^{-D(\langle \vec{x}_1, \ldots, \vec{x}_p \rangle, \langle \vec{y}_1, \ldots, \vec{y}_q \rangle)}$$

19.8 Maximum Likelihood Estimation Example

assume a Gaussian distributed random vector $\vec{x}_1, \vec{x}_2, ..., \vec{x}_m$:

$$p(\vec{x}; \underbrace{\vec{\mu}, \Sigma}_{\theta}) = \frac{1}{\sqrt{\det(2\pi\Sigma)}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})} = \mathcal{N}(\vec{x}; \vec{\mu}, \Sigma) \tag{19.19}$$

$$\begin{aligned}\hat{\vec{\mu}}, \hat{\Sigma} \quad &= \arg\max_{\vec{\mu},\Sigma} \prod_{i=1}^{m} p(\vec{x}_i; \vec{\mu}, \Sigma) \\ &= \arg\max_{\vec{\mu},\Sigma} \sum_{i=1}^{m} \log p(\vec{x}_i; \vec{\mu}, \Sigma) \\ &= \arg\max_{\vec{\mu},\Sigma} \mathcal{L}(\vec{x}_1, \vec{x}_2, ..., \vec{x}_m; \vec{\mu}, \Sigma)\end{aligned}$$

set up the log-likelihood function:

$$\mathcal{L} = \sum_{i=1}^{m} \Big(-\frac{1}{2}\log(\det(2\pi\Sigma)) - \frac{1}{2}(\vec{x}_i - \vec{\mu})^T \Sigma^{-1} (\vec{x}_i - \vec{\mu}) \Big)$$

a necessary condition for the maximization of the log-likelihood function is:

$$\frac{\delta\mathcal{L}}{\delta\vec{\mu}} \overset{!}{=} 0$$

$$\frac{\delta\mathcal{L}}{\delta\vec{\mu}} = \Sigma^{-1}\sum_{i=1}^{m}(\vec{x}_i - \vec{\mu}) \overset{!}{=} 0$$

The covariance Matrix is not Zero, so the sum has to be Zero:

$$\sum_{i=1}^{m}(\vec{x}_i - \vec{\mu}) = 0 \Longleftrightarrow \sum_{i=1}^{m}\vec{x}_i = \sum_{i=1}^{m}\vec{\mu} \Longleftrightarrow \sum_{i=1}^{x}\vec{x}_i = m\vec{\mu}$$

$$\hat{\vec{\mu}} = \frac{1}{m}\sum_{i=1}^{m}\vec{x}_i$$

for the covariance it's the same way:

$$\frac{\delta\mathcal{L}}{\delta\Sigma} \overset{!}{=} 0 \tag{19.20}$$

$$\hat{\Sigma} = \frac{1}{m}\sum_{i=1}^{m}(\vec{x}_i - \vec{\mu})(\vec{x}_i - \vec{\mu})^T \tag{19.21}$$

19.9 EM Algorithm Example

Estimate the priors p_k of classes $k = 1, 2, ..., k$ from the ovservation x where the probability density function of observations os given by the marginal over all classes:

$$p(x, \beta) = \sum_{k=1}^{K} p_k p(x|k; \beta) \tag{19.22}$$

Application of the EM scheme:

- observable random measruement: x
- hidden random measurement: k

- parameter set: $\theta = p_k; k = 1, ..., K$

In this case you only want to estimate the mixture weight p_k the parameters for the components are fixed.
Setup the Q-function:

$$\begin{aligned}
Q(\hat{\vec{\theta}}^{(i)}; \hat{\vec{\theta}}^{(i+1)}) &= \int p(y|x; \hat{\vec{\theta}}^{(i)}) \log p(x, y; \hat{\vec{\theta}}^{(i+1)}) \mathrm{d}y \\
&= \sum_{k=1}^{K} a_k \log \big(\underbrace{\hat{p_k}^{(i+1)} p(x|k; \beta)}_{p(k,x;\beta)} \big) \\
&= \sum_{k=1}^{K} a_k \log \hat{p}_k^{(i+1)} + \sum_{k=1}^{K} a_k \log p(x|k; \beta)
\end{aligned}$$

where

$$a_k = \frac{\hat{p}_k^{(i)} p(x|k; \beta)}{\sum_j \hat{p}_j^{(i)} p(x|j; \beta)} \qquad = \frac{p(k, x; \beta)}{p(x; \beta)} = p(k|x; \beta)$$

now maximize the Q-function with respect to $\hat{p}_k$: using the solution $\hat{p}_k = \frac{a_k}{\sum_{l=1}^{K} a_l}$ you found by deriving the Lagrangian:

$$\hat{p}_k^{(i+1)} = \frac{a_k}{\sum_l a_l} = \frac{\frac{\hat{p}_k^{(i)} p(x|k; \beta)}{\sum_j \hat{p}_j^{(i)} p(x|j; \beta)}}{\underbrace{\sum_l \frac{\hat{p}_l^{(i)} p(x|l; \beta)}{\sum_j \hat{p}_j^{(i)} p(x|j; \beta)}}_{=1}} = \frac{\hat{p}_k^{(i)} p(x|k; \beta)}{\sum_j \hat{p}_j^{(i)} p(x|j; \beta)}$$

Its very important to get a estimate for initialization, so you should use all prior information that you have. If no prior is available, assume a uniform distribution.

19.10 Computation of Marginals

Assume five hidden variables $l_1, l_2, \ldots, l_5$.

$$p(\vec{x}_1, \ldots, \vec{x}_m; \vec{\theta}) = \underbrace{\underbrace{\sum_{l_1 \ldots l_m}^{K}}_{\mathcal{O}(K^m)} \underbrace{\prod_{i=1}^{m} p_{l_i} p\left(\vec{x}_i; \vec{\theta}_{l_i}\right)}_{\mathcal{O}(m)}}_{\mathcal{O}(K^m \cdot m)}$$

We can rearrange to reduce the complexity:

$$\begin{aligned} p(\vec{x}_1, \ldots, \vec{x}_m; \vec{\theta}) &= \left(\sum_{l_1 \ldots l_{m-1}}^{K} \prod_{i=1}^{m-1} p_{l_i} p\left(\vec{x}_i; \vec{\theta}_{l_i}\right) \right) \sum_{l_m=1}^{K} p_{l_m} p\left(\vec{x}_m; \vec{\theta}_{l_m}\right) \\ &= \prod_{i=1}^{m} \left(\sum_{l_i=1}^{K} p_{l_i} p\left(\vec{x}_i; \vec{\theta}_{l_i}\right) \right) \in \mathcal{O}(m \cdot k) \end{aligned}$$

Let's also consider discrete mixtures: assume $q_i(x)$ defining a discrete distribution. We observe a random variable that uses $q_i(x)$ with a probability of p_i. We know neither $q_i(x)$ nor p_i and want to estimate both. Our observable is given by

$$p(\vec{x}; \vec{\theta}) = \sum_{k=1}^{K} p_k \underbrace{p(\vec{x}; \vec{\theta}_k)}_{q_k(\vec{x})}$$

constrained by

$$\sum_{k=1}^{k} p_k = 1; \qquad \sum_{\vec{x}} q_k(\vec{x}) = 1$$

We can setup the Q-function to use expectation maximization:

$$\begin{aligned} Q\left(\vec{\theta}, \vec{\theta}'\right) &= \sum_{k=1}^{K} p\left(k | \vec{x}; \vec{\theta}_k\right) \log p\left(k, \vec{x}; \vec{\theta}'_k\right) \\ &= \sum_{k=1}^{K} \frac{p\left(k, \vec{x}; \vec{\theta}_k\right)}{p\left(\vec{x}; \vec{\theta}_k\right)} \log p\left(k, \vec{x}; \vec{\theta}'_k\right) \\ &= \sum_{k=1}^{K} \frac{p\left(k, \vec{x}; \vec{\theta}_k\right)}{\sum_{l=1}^{K} p\left(l, \vec{x}; \vec{\theta}_l\right)} \log p\left(k, \vec{x}; \vec{\theta}'_k\right) \\ &= \sum_{k=1}^{K} \frac{p_k q_k(\vec{x})}{\sum_{l=1}^{K} p_l q_l(\vec{x})} \log p'_k q'_k(\vec{x}) \\ &=: \sum_{k=1}^{K} \frac{p_k q_k(\vec{x})}{a_k(\vec{x})} \left(\log p'_k + \log q'_k(\vec{x})\right) \end{aligned}$$

where $\vec{\theta}$ is fixed and $\vec{\theta}'$ is the variable we want to optimize for. Using the Q-function, we can set up an objective function and thus an optimization problem:

$$\mathcal{L} = Q\left(\vec{\theta}, \vec{\theta}'\right) + \sum_{k=1}^{K} \lambda_k \left(\sum_{\vec{x}} q_k(\vec{x}) - 1 \right) + \nu \left(\sum_{k=1}^{k} p_k - 1 \right)$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial q'_k(\vec{x})} &= a_k(\vec{x})\frac{1}{q'_k(\vec{x})} + \lambda_k \overset{!}{=} 0 \\
\frac{a_k(\vec{x})}{q'_k(\vec{x})} &= -\lambda_k \\
a_k(\vec{x}) &= -\lambda_k q'_k(\vec{x}) \\
\sum_{\vec{x}} a_k(\vec{x}) &= -\lambda_k \sum_{\vec{x}} q'_k(\vec{x})
\end{aligned}$$

we can plus this into the equation we had before

$$\begin{aligned}
\frac{a_k(\vec{x})}{q'_k(\vec{x})} &= \sum_{\vec{x}} a_k(\vec{x}) \\
q'_k(\vec{x}) &= \frac{a_k(\vec{x})}{\sum_{\vec{x}} a_k(\vec{x})}
\end{aligned}$$

We can also compute p'_k:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial p'_k} &= a_k(\vec{x})\frac{1}{p'_k)} + \nu \overset{!}{=} 0 \\
p'_k &= \frac{a_k(\vec{x})}{\sum_{k=1}^{K} a_k(\vec{x})} \\
\sum_{\vec{x}} p'_k &= \sum_{\vec{x}} \frac{a_k(\vec{x})}{\sum_{k=1}^{K} a_k(\vec{x})} \\
p'_k &= \frac{1}{N} \sum_{\vec{x}} \frac{a_k(\vec{x})}{\sum_{k=1}^{K} a_k(\vec{x})}
\end{aligned}$$

And we thus have estimated $q_i(\vec{x})$ and p_i as wanted.

20 Big Picture

The whole Lecture is about to estimate the posterior probability $p(y|\vec{x})$!

20.1 Classification

Classification: reading a feature Vector $\vec{x}$ and mapping the feature Vector to a Class index y

$$\delta(\vec{x}) = y \tag{20.1}$$

20.2 Regression

Regression: mapping a feature Vector $\vec{x}$ to a real valued number

$$y = \delta(\vec{x}), \qquad y \in \mathbb{R} \tag{20.2}$$

20.3 supervised vs. unsupervised Learning

supervised learning mean you get feature Vectors $\vec{x}$ and the assigned Class number y.

unsupervised learning mean you just observe features without any knowledge to which class number they belong to.

20.4 discriminative vs. generative model

discriminative model means you compute the posterior probability $p(y|\vec{x})$ directly

generative model means you compute the prior $p(y)$ and the class conditional $p(\vec{x}|y)$ and multiply them together to get the posterior probability

20.5 Bayesian Classifier

The Bayesian classifier decide for the class y that maximizes the posterior probability

$$y^* = \delta(\vec{x}) = \arg\max_y p(y|\vec{x}) = \arg\max_y \frac{p(y) \cdot p(\vec{x}|y)}{p(\vec{x})} \tag{20.3}$$

the Probability of $p(\vec{x})$ is independent of y.
To find the position of the maximum its enough to maximize the nominator !

$$y^* = \arg\max_y p(y) \cdot p(\vec{x}|y) \tag{20.4}$$

Why is $p(\vec{x}|y)$ easier to compute / model than $p(y|\vec{x})$?
because y is a discrete variable. For each class you have to setup a density function. In a 2-Class problem that are only 2 pdf's.
$\vec{x}$ is a continuous Variable, in this case you have to setup an unlimited Number of density functions !

The Bayesian Classifier is optimal with respect to the 1/0-loss function!

20.6 optimal classifier

a optimal classifier minimize the average loss!

20.7 Gaussian Classifier

if you use a generative modelling of the posterior probability and the class conditional pdf is a Gaussian, then this is a Gaussian classifier. if all $p(\vec{x}|y)$ are Gaussians then the optimal decision Boundary is quadratic! have all the Gaussians the same covariance matrices then the optimal decision Boundary is linear!

20.8 logistic regression

- discriminative model

logistic regression means you can compute the posterior probability directly:

$$p(y=0|\vec{x}) = \frac{1}{1+e^{-F(x)}} \qquad p(y=1|\vec{x}) = \frac{1}{1+e^{F(x)}} \tag{20.5}$$

$F(\vec{x}) = 0$ is the decision boundary. A point on the decision boundary full fill following equation:

$$\log \frac{p(y=0|\vec{x})}{p(y=1|\vec{x})} = 0 \tag{20.6}$$

20.9 logistic Function | Sigmoid Function

$$g(x) = \frac{1}{1+e^{-x}}, \qquad x \in \mathbb{R} \tag{20.7}$$

20.10 The Big Picture

Pattern Recognition is the problem of classification: $y^* = \delta(\vec{x})$.

- Bayesian classifier
  $y^* = \arg\max_y p(y|\vec{x}) = \arg\max_y \frac{p(y)p(\vec{x}|y)}{p(\vec{x})} = \arg\max_y p(y)p(\vec{x}|y)$

- Optimal classifier
  Minimizes the average loss. The Bayesian classifier is optimal w.r.t/ the 0-1-loss function

- Logistic Regression
  $p(y = 0|\vec{x}) = \frac{1}{1+e^{yF(\vec{x})}}$, where $F(\vec{x}) = 0$ is the decision boundary. $\log \frac{p(y=1|\vec{x})}{p(y=0|\vec{x})} = 0$. A decision boundary is quadratic, if $F(\vec{x}) = \vec{x}^T A\vec{x} + \vec{\alpha}^T \vec{x} + \alpha_0 = 0$.

- Gaussians
  $p(\vec{x}|y) = \mathcal{N}(\vec{x}; \mu, \Sigma)$. If $\Sigma_1 = \Sigma_{-1} = \Sigma$, then we get a linear decision boundary. If $\Sigma = \mathrm{I}$, we get the nearest neighbour classifier.

- NaÃ¯ve Bayes
  Assumes components of $\vec{x}$ are statistically independent. Reduces dimensionality of the result space! The covariance matrix is a diagonal matrix.

- Perceptrons
  Linear decision boundary, minimize the number of misclassified samples.

- Norms, Penalty Functions
  Different norms and their unit balls, penalty functions as abstractions of norms. Difference of classification and regression.

- Linear Regression
  And its derivation with constraints, e.g. Ridge Regression, Lasso

- Optimization Problems
  Unconstrained (gradient descent, steepest descent) and constrained optimization (Lagrange multiplier)

- Support Vector Machines
  Minimize $\frac{1}{2} \|\vec{\alpha}\|^2$ subject to $y_i(\vec{\alpha}^T \vec{x}_i + \alpha_0) \geq 1 \ \forall i$. Complimentary slackness and KKT conditions can be used to show only support vectors influence the decision boundary. $\delta(\vec{x}) = \vec{\alpha}^T \vec{x} + \alpha_0$. Where is $\vec{\alpha}$?

- Kernels
  Allow us to compute non-linear decision boundaries using transformed feature vectors without knowing the transformation $\phi(\vec{x})$.

- Kernel PCA
  Optimal if we have few, but high-dimensional samples; the covariance matrix cannot be computed, but the Kernel PCA can.
- Gaussian Mixture Models
  Model an arbitrary pdf as a combination of Gaussians. Estimate the Gaussian parameters using EM.
- Expectation Maximization
  Deals with estimation of hidden variables.
- Independent Component Analaysis
  Estimate $\vec{s}, A$ from $\vec{x} = A\vec{s}$ by maximizing the non-Gaussianity of a projection of $\vec{x}$.
- Model Assassement
  Bootstrap, Jackknife, Cross-Validation
- Boosting
  AdaBoost, Viola & Jones