Pattern Recognition Q1

Pattern Recognition (Technische Universiteit Delft)

Summary Pattern Recognition IN4085

Taeke de Haan

October 2017

Q1

WARNING: MAY CONTAIN ALTERNATIVE FACTS!

Contents

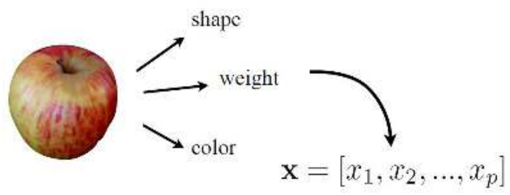
Book section: Ch.1,2.1-2.3

1 Introduction

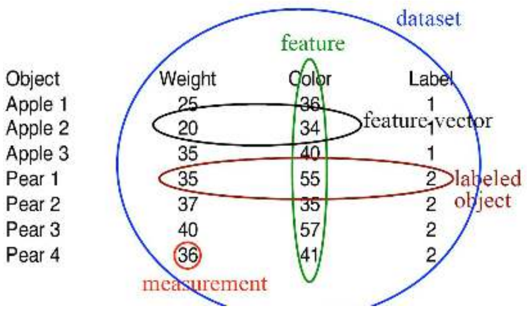
Pattern recognition: a branch of machine learning that focuses on the recognition of patterns and regularities in data

1.1 Datasets and Features

Training set: All examples are labeled, this set is used to train/develop our system.
Test set: These examples cannot be used to train our system, the examples do not have to be labeled. When labels are available, we can objectively evaluate our system.
Features: To do tasks automatically we have to encode the objects. This is often done using features.



When we measure the features of many objects we obtain a dataset.



Object	Weight	Color	Label
Apple 1	25	36	1
Apple 2	20	34	1
Apple 3	35	40	1
Pear 1	35	55	2
Pear 2	37	35	2
Pear 3	40	57	2
Pear 4	36	41	2

Features give a specific view of the objects, the user is responsible for it. Good features allow for pattern recognition, bad features allow for nothing.
Other approaches of defining objects are:

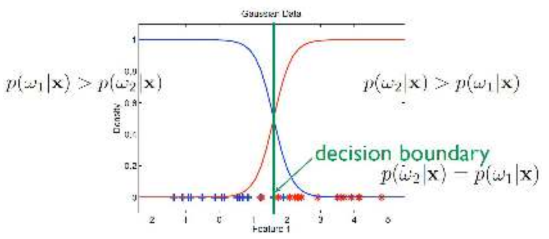
- Dissimilarity approach
- Structural pattern recognition (graphs)

Feature approach is very well developed, other approaches still need more research.

1.2 Assigning objects

Measurements are done to determine the features of an object. These measurements can be interpreted as a vector in a vector space $x = (x_1, x_2, ..., x_p)$. This originates in principle from a probability density over the whole feature space $p(x, y)$.

We want to find $p(\text{class}_n|\text{feature}_i)$ e.g. $p(\omega|x)$ these are known as the class conditional properties.



Decision boundary: Where $p(\omega_1|x) = p(\omega_2|x)$. Once the boundary is determined we can assign objects by using: $p(\omega_1|x) > p(\omega_2|x)$ then assign to ω_1.

1.3 Bayes' rule

In many cases the posterior is hard to estimate but form of the class distributions can be assumed than we can use Bayes' rule:

$$p(\omega|x) = \frac{p(x|\omega)p(\omega)}{p(x)}$$

- $p(\omega|x)$ Class posterior
- $p(x|\omega)$ class conditional distribution
- $p(\omega)$: class prior
- $p(x)$ unconditional data distribution

This results in the Bayes classification rule:

$$p(x|\omega_1)P(\omega_1) \gtrless p(x|\omega_2)P(\omega_2)$$

We find the posterior by following these steps:

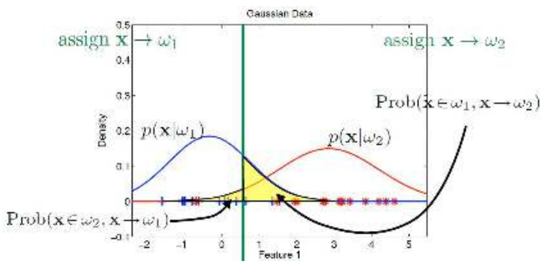
1. Estimitate the class conditional probabilities
   a) Assume a model
   b) Estimate the model parameters such that the example objects fit well
2. Multiply with the class priors
3. Divide by the data distribution.
4. Assign objects to the class with the highest posterior probability

Note: other approaches than Bayes' rule are possible. We discuss it later

1.4 Cost

To place the decision boundary at an good position we need to define what good is. This is done with a cost function.

Minimizing Error



For a two class problem two errors:

$$\epsilon_1 = \int_{\Omega_2} p(x|\omega_1)dx$$

$$\epsilon_2 = \int_{\Omega_1} p(x|\omega_2)dx$$

The total probability of commiting a decision error P_e is:

$$P_e = P(\omega_1)\epsilon_1 + P(\omega_2)\epsilon_2$$

Bayes error ϵ*: the Bayesian classifier is optimal with respect to minimizing the classification error probability error. complicated.

Minimizing Risk

In some cases misclassification of class A to class B is much more dangerous than misclassification of class B to class A. Therefore we introduce a misclassification cost that measures the cost of assigning an object that came from class ω_j to class ω_i: λ_{ji}. Now we want to minimize the following risk:

$$r = \lambda_{12}P(\omega_1)\epsilon_1 + \lambda_{21}P(\omega_2)\epsilon_2$$

For an M-class problem the risk or loss associated with ω_k is defined as:

$$r_k = \sum_{i=1}^{M} \lambda_{ki}\epsilon_k$$

Our goal is to select partitioning region Ω_j so that the following average risk is minimized:

$$\begin{aligned} r &= \sum_{k=1}^{M} r_k P(\omega_k) \\ &= \sum_{i=1}^{M} \int_{\Omega_i} \left(\sum_{k=1}^{M} \lambda_{ki} p(x|\omega_k) P(\omega_k) \right) dx \end{aligned}$$

Risk is minimized if the regions $Omega_i$ are chosen as small as possible, so make x part of Ω_i if:

$$\sum_{k=1}^{M} \lambda_{ki} p(x|\omega_k) P(\omega_k) \leq \sum_{k=1}^{M} \lambda_{kj} p(x|\omega_k|x) P(\omega_k)$$
$$\forall j \neq i$$

Solving results is:

$$x \in \omega_1(\omega_2) \text{ if } \frac{p(x|\omega_1)}{p(x|\omega_2)} > (<) \frac{P(\omega_2)}{P(\omega_1)} \frac{\lambda_{21} - \lambda_{22}}{\lambda_{12} - \lambda_{11}}$$

Neyman-Pearson Criteria

The Neyman-Pearson criteria sets the error for one of the classes equal to a chosen value. Often in radar systems the probability of a false alarm is set to a predetermined threshold.

Book section: 2.4-2.5.5

2 Bayes' Classifiers

The main challenge of pattern recognition using a Bayes' clasifier is estimating $p(x|\omega)$. We have two options:

1. **Parametric** : describe $p(x|\omega)$ with a finite number of parameters
2. **Nonparametric:** describe $p(x|\omega)$ with an infinite number of parameters

This week we discussed parametic Bayes' classifiers

2.1 Models and assumptions

To describe $p(x|\omega)$ we need a model which is based on certain assumptions.

Guassian distribution

Also known as a normal distribution:

$$p(x) = \frac{1}{(2\pi)^{l/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

Mixture modeling

Model complex pdf as weighted sum of simple pdf's.

$$p(x) = \sum_{i=1}^{N} P_i p(x|\sigma_i)$$

2.2 Decision boundaries

In this section we will use the Gaussioan model. because of its exponential form we prefere to work with a logorithmic function:

$$\begin{aligned} g_i(x) &= \ln(p(x|\omega_i)P(\omega_i)) \\ &= \ln(p(x|\omega_i)) + \ln(P\omega_i) \\ &= -\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) + \ln P(\omega_i)... \\ &... - (l/2)\ln 2\pi - (l/2)\ln|\Sigma_i| \end{aligned}$$

Equal diagonal covariance matrices

If we assume equal diagonal covariance matrices the decision boundary has a linear form:

$$g_i(x) = w_i^T x + w_{i0}$$

where:

- $w_i = \Sigma^{-1}\mu_i = \frac{1}{\sigma^2}\mu_i^T$
- $w_{i0} = \ln P(\omega_i) - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i$

The respective decision surfaces are hyper planes:

$$g_{ij}(x) = g_i(x) - g_j(x) = w^T(x - x_0) = 0$$

where:

- $w = \mu_i - \mu_j$
- $x_0 = \frac{1}{2}(\mu_i + \mu_j) - \sigma^2 \ln\left(\frac{P(\omega_i)}{P(\omega_j)}\right) \frac{\mu_i - \mu_j}{||\mu_i - \mu_j||^2}$

Equal non-diagonal covariance matrices

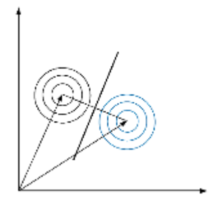
If the covariance matrix are equal but non-diognal the decision hyperplane is no longer orthogonal to the vector $\mu_i - \mu_j$ but to it's linear transformation $\Sigma^{-1}(\mu_i - \mu_j)$:

$$x_0 = \frac{1}{2}(\mu_i + \mu_j) - \ln\left(\frac{P(\omega_i)}{P(\omega_j)}\right) \frac{\mu_i - \mu_j}{||\mu_i - \mu_j||^2_{\Sigma^{-1}}}$$

If the covariance matrixes are diagnol but different than the result will be a quadratic function.
In practice it is quite common to assume a Gaussian distribution. The resulting linear or quadratic Bayesian classifier are known as **linear discriminant analysis** (LDA) (or Fisher's linear discriminant) and **quadratic discriminant analysis** (QDA). They exhibit good performance in a large set of diverse applications and are considered to be among the most popular classifiers. A major disadvantage is is the large amount of parameters which have to be estimated

Nearest Mean Classifier

Minimum distance classifiers using Euclidean distance is also called nearest mean classifier [NMC]



2.3 Estimating parameters

The Gaussian distribution contains many parameter we need to estimate. To show the dependence on these parameters we write $p(x|\omega_i;\theta)$

Maximum likelihood (ML) estimation

Maximum likelihood: maximizes probability of observing data given particular parameters. Often log-likelyhood is considered:

$$L(\theta) = \ln \prod_{k=1}^{N} p(x_k : \theta)$$

This is optimal for:

$$\begin{aligned}\frac{\partial L(\theta)}{\partial \theta} &= \sum_{k=1}^{N} \frac{\partial \ln p(x_k;\theta)}{\partial \theta} \\ &= \sum_{k=1}^{N} \frac{1}{p(x_k;\theta)} \frac{\partial p(x_k;\theta)}{\partial \theta} = 0\end{aligned} \tag{1}$$

Maximum a posteriori (MAP) estimation

Maximum a posteriori: maximizes a posteriori probability parameters given data. Explicit use of a priori knowledge trough a prior. $\arg\max_\theta p(y\theta)p(\theta)$, results in:

$$\frac{\partial}{\partial \theta}\left(p(\theta)p(X|\theta)\right) = 0$$

Bayesian Inference

Given set X of N training vectors and the a priori information about the pdf $p(\theta)$ the goal is to compute the conditional pdf $p(x|X)$

$$p(x|X) = \int p(x|\theta)p\theta|X)d\theta$$

with:

$$P(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta)d\theta}$$

$$P(X|\theta) = \prod_{k=1}^{N} p(x_k|\theta)$$

So it tries to predict expected future data x with the current data set we have X. However these intergrals are often impossible to solve analytically. If $p(\theta|X)$ is known then if a large enough number of samples θ_i, $i = 1, ..., L$ are available than we can approximate the expectation value with:

$$p(x|X) \approx \frac{1}{L}\sum_{i=1}^{L} p(x|\theta_i)$$

Now the problem shifts to generating a set of samples θ_i. This is done using Markov Chain Monte Caarlo techniques (MCMC) which I do noit understand at all.

Expectation maximization

How to go about optimizing $p(x) = \sum_{i=1}^{N} P_i p(x|\theta_i)$?

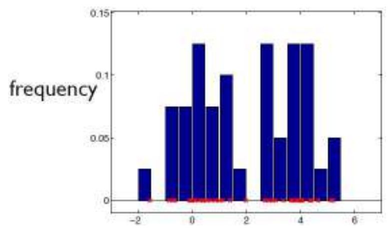
Expectation maximization: Given θ_i, membership can be determined and, given membership, θ_i, can be estimated

Book section: 2.5.6-2.7

3 Non-parametric Bayes' Classifiers

Non-parametic classiefiers are basicly variants of the histogram approximation of an unknown pdf.

3.1 Histogram method



Count the amount of objects in each region k_N and devide it by the total amount of objects N:

$$\hat{p}(x) = \hat{p}(\hat{x}) \approx \frac{1}{h}\frac{k_N}{N}$$

$\hat{p}(x)$ converges to the true value px as $N \to \infty$ provided:

- $h_N \to 0$
- $k_N \to \infty$
- $\frac{k_N}{N} \to 0$

For a reasanble approximation h cannot be too large. However if we make h too small while we don't have enough N the solution will become unstable.

3.2 Parzen density estimation

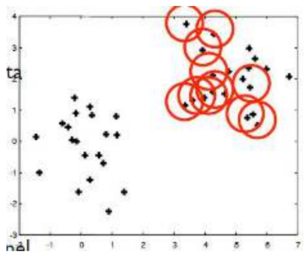
Fix a cell volume around all data points in the training data, and add a contribution to each cell. These functions are known an kernels, potential functions, or Parzen windows. Cell shape can be a function like this:

$$K(x_{ij}, h) = \begin{cases} 0 & \text{if } |x_{ij}| > h \\ 1/V & \text{if } |x_{ij}| \leq h \end{cases} \tag{2}$$

with V the volume of the kernel. For test objects x sum all cells:

$$\hat{p}(x|h) = \frac{1}{n}\sum_{i=1}^{n} K(||x - x_i||, h) \tag{3}$$

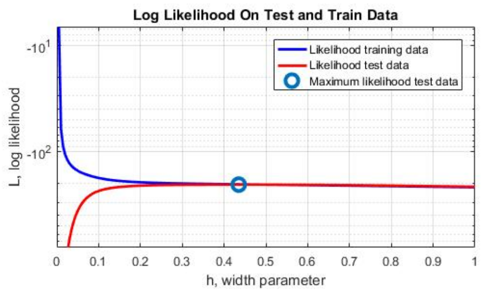
We assign a chance to point x which corresponds to how many objects are within a range h of x.



Optimizing of h

The choice of h is important, there are many methods to imptimize:

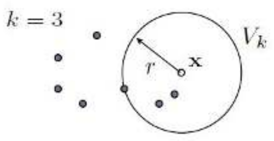
- Use the average k-nearest neighbor distance (k=10 is suggested...)
- Use a hearistic
- Optimize the likelihood using leave-one-out



Note that optimizing h on training data results in h = 0, a typical case of over fitting

3.3 Nearest neighbor classification

Same as parzen, however dont fix the volume of the cell: grow the cell until it covers k objects. find the k-th nearest naighbor.



$$\hat{p}(x|\omega_m) = \frac{k_m}{N_m V_k(x)}$$

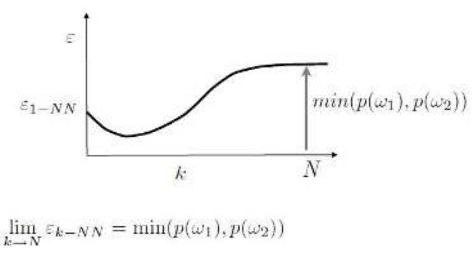
Class prior:

$$\hat{p}(\omega_m) = \frac{N_m}{n}$$

The likelihood ratio test becomes:

$$\frac{V_2}{V_1} > (<) \frac{N_1}{N_2}\frac{P(\omega_2)}{P(\omega_1)}\frac{\lambda_{21} - \lambda_{22}}{\lambda 12 - \lambda 11}$$

The choice of k



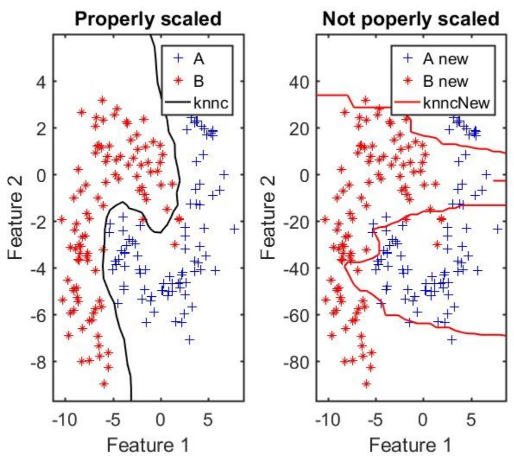
3.4 Advantages and disadvantages

Advantages:

- Simple and flexible classifiers.
- Often a very good classification performance.
- It is simple to adapt the complexity of the classifier.

Disadvantages:

- Relatively large training sets are needed.
- The complete training set has to be stored.
- Distances to all training objects have to be computed.
- The features have to be scaled sensibly.
- The value for k or h has to be optimized.



3.5 The Curse of Dimensional

More features provide more information, Unfortunately there are disadvantages:

- The number of examples required increases exponentially with the number of features.
- Adding more features can increase noise.
- Insufficient data to estimate anything.
- Most of the data is in the tails.

To make high dimensional learning still feasible:

- Reduce features
- Simplify models
- Careful optimization

Variable h

Due to the "data in the tail" effect it can happen that some regions in the feature space are sparsely represented in the data set. sometimes a variable h is used in a situation like that.

Naive Bayes Classifier

To decrease the required training data we can assume that all features are independent so we get:

$$p(x|\omega) = p(x_1, x_2, x_3, x_4, ..., x_l|\omega) = \prod_{i=1}^{l} p(x_i|\omega)$$

Which results in the classifier:

$$\omega_m = \text{argmax}_\omega \left(p(\omega) \prod_{i=1}^{l} p(x_i|\omega) \right)$$

Which enables us to estimate $p(x_i|\omega)$ per feature, no curse of dimensionaltiy.However assuming indepenence can also reult in arge errors.

Bayesion Networks

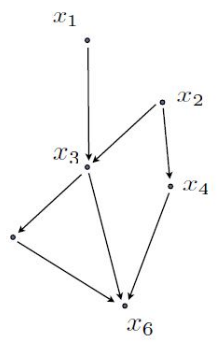
With the Naive Bayes clasifier we went form one extreme: all features depend on each other, to an other extreme; full in dependence. Bayesion networks try to find a balance in between these two. This is done using the probability chain rule:

$$p(x_1, x_2, x_3) = p(x_3|x_2, x_1)p(x_2|x_1)p(x_1)$$

Vissualy shown below

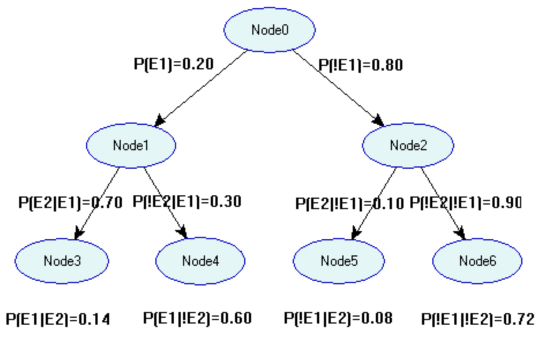
$$\begin{tikzcd} x_1 \arrow[dr] \arrow[dd] & \\ & x_2 \arrow[dl] \\ x_3 & \end{tikzcd}$$

In general, some links maybe missing



Less dependencies makes the pdf easier to estimate. The complete specification of a Bayesian network requirewd:

1. The marginal probability of root nodes
2. The conditional probabilities of the non-root nodes.



The bayesian netowrks allows us to compute the probability of any node in the graph in an efficient way. Which is useful in the field of artificial inteligence, closely related to pattern recognition.

Book section: 3.1-3.5, but NOT pg. 101

4 Linear Classifiers

In the previous chapter we discussed classification using density estimation: $p(x|\omega)$. We will now focus on linear classifiers, regardless of the underlying distributions describing the training data.
We will focus on a two-class case. Than the respective decision hypersurface in the l-dimnesional feature space is a hyperplane:

$$g(x) = w^t x + w_0 = 0$$

With:
- $w = [w_1, w_2, ...w_l]^T$: weight vector
- ω_0: threshold

Classify:

$$\begin{aligned} \omega_1 \quad &\text{if} \quad w^T x + w_0 \leq 0 \\ \omega_2 \quad &\text{if} \quad w^T x + w_0 < 0 \end{aligned}$$

Advantages of such a linear model is its simplicity and computational attractiveness.

Now we have a general model/function for the classifier we need to invent a loss function and optimize the parameters w and w_0 for this loss function.

4.1 Perceptron

Our major concern is computing ω_i, $i = 0, ..., l$. For the perceptron we will have to assume our data is linearly separable.

$$\begin{aligned} w'^T x' > 0 \quad &\forall x \in \omega_1 \\ w'^T x' < 0 \quad &\forall x \in \omega_2 \end{aligned}$$

with:
- $x' = [x^T, 1]^T$
- $w' = [w^T, w_0]^T$

The perceptron cost is defined as:

$$J(\omega) = \sum_{x \in Y} (\delta_x \omega^T x)$$

Where:
- Y: subset of training vectors which are misclassified by the hyperplane.
- $\delta_x = \begin{cases} -1 & \text{if } x \in \omega 1 \\ +1 & \text{if } x \in \omega 2 \end{cases}$

Optimize by setting $\frac{\partial J(\omega)}{\partial \omega}$ to zero is hard/impossible. Thus an iterative gradient descent process will be used. This results in the **perceptron algorithm**:

$$\omega(t+1) = \omega(t) - \rho(t) \sum_{x \in Y} \delta_x x$$

With:
- ρ: learning rate.

With this algorithm we can find ω in an iterative process, which stalls one all x's all correctly identified.

Convergence

The perception algorithm converges if:

$$\lim_{t \to \infty} \sum_{k=0}^{t} \rho_k = \infty$$

$$\lim_{t \to \infty} \sum_{k=0}^{t} \rho_k^2 < \infty$$

The algorithm converges to a solution in a finite number of steps if ρ_t vanishes as $t \to \infty$, but it should not go too fast. The proper choice of ρ_t is vital for the convergence speed of the algorithm.

Advantages and disadvantages

- Just a 'simple' linear classifier
+ Is trained incrementally, or in batches (applicable to very large datasets)
+ When the data is separable, it will find the solution.
- When the data is not separable it will update for ever, and ever, and ever...

4.2 Least Squares Method

In contrast to the perceptron this classifier also works on not linear seperable data, this will of course lead to a certain classification error. The weight vector is computed in a way to minimize the mean square error (MSE):

$$J(\omega) = E[|y - x^T \omega|^2]$$
$$\hat{\omega} = argmin_\omega J(\omega)$$

with y equals the desired output. This results in:

$$\frac{\partial J(\omega)}{\partial \omega} = 2E[x(y - x^T \omega)] = 0$$
$$\hat{\omega} = R_x^{-1} E[xy] = E[xx^T] E[xy]$$

R_x is the correlation, or autocorrelation matrix.

$$R_x = E[\mathrm{xx}^T] = \begin{bmatrix} E[x_1x_1] & \dots & E[x_1x_d] \\ E[x_2x_1] & \dots & E[x_2x_d] \\ \vdots & \vdots & \vdots \\ E[x_dx_1] & \dots & E[x_dx_d] \end{bmatrix}$$

$E[xy]$ is the cross-correlation between the desired output and the input feature vector.

$$E[\mathrm{x}y] = E\left[\begin{bmatrix} x_1y \\ x_2y \\ \vdots \\ x_dy \end{bmatrix}\right]$$

To find these correlations we need the true distribution $p(x,y)$ which we don't have. Therefore, insted of using the MSE, we use the sum of error squares or the least squares (LS):

$$J(\omega) = \sum_{i=1}^{N}(y_i - x_i^T\omega)^2 \equiv \sum_{i=1}^{N} e_i^2$$

$$\hat{\omega} = (X^TX)^{-1}X^Ty$$

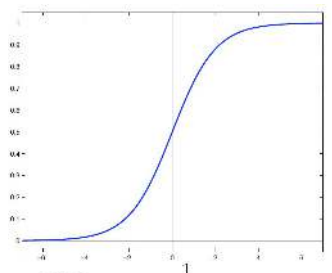
4.3 Logistic classifier

The logarithm of the liklihood ratios is modeled via linear functions:

$$\ln\left(\frac{p(\omega_1|x)}{p(\omega_2|x)}\right) = \omega_0 + \omega^T x$$

The classifier becomes:

$$p(\omega_2|x) = \frac{1}{1 + exp(\omega_0 + \omega^T x)}$$



$$f(x) = \frac{1}{1+\exp(-x)}$$

Which has a sigmoidal shape. To optimize the parameters on a training set log maximum likelihood is used:

$$\ln(L) = \sum_{i=1}^{n_1} \ln p(\omega_1|x_i^{(1)}) + \sum_{i=1}^{n_2} \ln p(\omega_2|x_i^{(2)}) + K$$

now fill in the equations for $p(\omega_n|x)$:

$$\ln(L') = \sum_{r=1}^{n_1}(w_0 + w^T x_r^{(1)}) - \sum_{r=1}^{n_1+n_2} \ln(1+\exp(w_0+w^T x_i))$$

Take the derivative of $\ln(L)$ with respect to β_0, β:

$$\begin{aligned} \frac{\partial \ln(L)}{\partial w_0} &= n_1 - \sum_{i=1}^{n_1+n_2} p(\omega_1|x_i) \\ \frac{\partial \ln(L)}{\partial w_j} &= \sum_{i=1}^{n_1}(x_i^{(1)})_j - \sum_{i=1}^{n_1+n_2} p(\omega_1|x_i)(x_i)_j \end{aligned} \tag{4}$$

Take initial values $w_0 = 0$, $w = 0$, and keep iterating till convergence:

$$w_{\text{new}} = w_{\text{old}} + \eta \frac{\partial \ln(L)}{\partial w}$$

4.4 Bias-Variance Dilema

We attempt to to approximate the function E[x — y] using a function $g(x;\mathcal{D})$ depending on data det $\mathcal{D}$. The effecitveness of an estimator can be evalued by computing it's mean square deviation form the optima value:

$$\begin{aligned} &E_{\mathcal{D}}\ [(g(x;\mathcal{D}) - E[y|x])^2 \\ &\quad = (E_{\mathcal{D}}[g(x;\mathcal{D})] - E[y|x])^2 \\ &\quad + E_{\mathcal{D}}\left[(g(x;\mathcal{D}) - E_{\mathcal{D}}[g(x;\mathcal{D})])^2\right] \end{aligned}$$

The first term is the contribution of the bias, the second term of the variance. For a finite dataset there is a trade off to be made between these two terms.

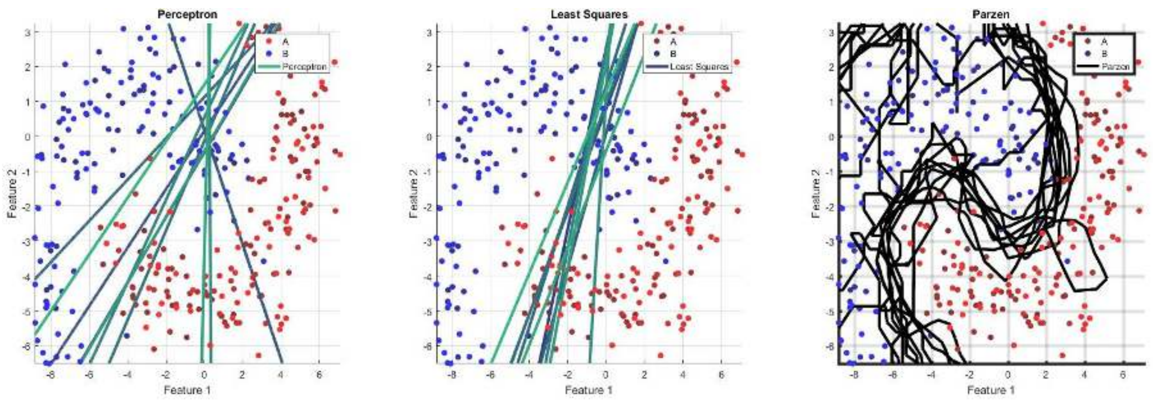
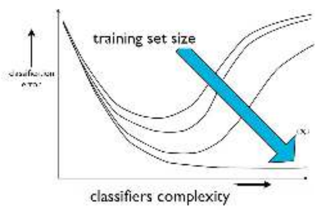


Figure 1: It can be seen that the least square classieifer has a mhuch smaller variance than the Perceptron and Parzen classifier

Book section: 3.6-3.7

5 Support Vector Classifiers

Generalization performance of the clasifier: the capability of the classifier designed using a training set, to operate satisfactory with data outside the training set.



5.1 Regularization

When insufficient data is available (nr of objects is smaller than dimensionality) the inverse covariance matrices are not defined!

$$\Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow \Sigma^{-1} = \begin{bmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{0} \end{bmatrix}$$

Therefore we add a bit of artificial noise to the Gaussian:

$$\hat{\Sigma} = \begin{bmatrix} 5+\sigma & 0 \\ 0 & 0+\sigma \end{bmatrix} \Rightarrow \hat{\Sigma^{-1}} = \begin{bmatrix} \frac{1}{5+\sigma} & 0 \\ 0 & \frac{1}{0+\sigma} \end{bmatrix}$$

This method is called **regularization**

$$\hat{\sigma}_i = \sigma_i = \lambda I$$

If we purley optimize on the training set we may cause overfitting:

$$\min_{\theta} \epsilon_A(\theta)$$

Where:
- ϵ_A: training error

Therefore we can optimize using a regulizer:

$$\min_{\theta} \epsilon_A(\theta) + \lambda\Omega(\theta)$$

with:
- λ: regularization parameter
- $\Omega(\theta)$: regularizer

By changing the regularization parameter, the complexity is changed, how do we measure complexity?

5.2 Vapnik-Chervonenkis dimension

VC-dimension:The largest number h of vectors that can be separated in all the 2^h possible ways.

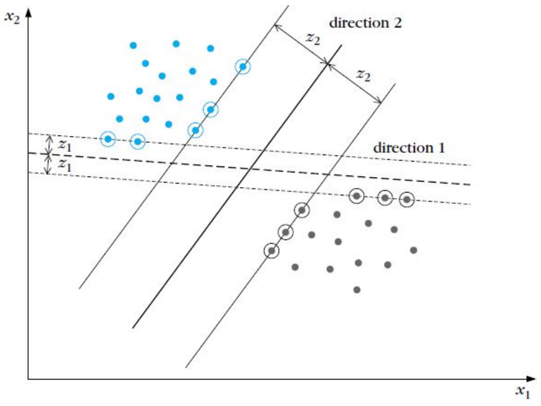
For a linear classifier:

$$h = p + 1$$

Only for a very few classifiers the VC-dimension is known. However when you know h of a clasifier you van bound its true error.

5.3 Support Vecotor Machines

We can reduce the VC-dimension of a linear classifier with extra constraints.



We want to maximize distance z.

$$z = \frac{|g(x)|}{||\omega||}$$

We can now scale w ,w_0 so that the value of $g(x)$, at the nearest points in ω_1, $\omega)2$ is equal to 1 for $\omega 1$ and, thus, equal to 1 for ω_2. To maximize z, we minimize w:

$$J(w, w_0) \equiv \frac{1}{2}||w||^2$$
$$y_i(w^T x_i + w_0) \geq 1, \quad i = 1, 2, ..., N$$

This is a constained optimization problem. It can be rewritten in the Lagrangian:

$$\mathcal{L}(w, \lambda) = \frac{1}{2}||w||^2 - \sum_i \lambda_i(y_i(w^T x_i + w_0) - 1)$$

After solving results in:

$$w = \sum_{i=1}^{N} \lambda_i y_i x_i$$
$$\sum_{i=1}^{N} \lambda_i y_i = 0$$

Where:
- λ_i: Lagrange multipliers

if $\lambda_i \neq 0$, the support vector w lies on either of the two hyperplanes.
The Lagrange multipliers can be found by solving the Lagrangian duality:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j x_i^T x_j\right) \tag{5}$$

$$\text{subject to:} \sum_{i=1}^{N}\lambda_i y_i = 0 \qquad \lambda \geq 0 \tag{6}$$

The classifier is determined by objects, not features: the classifier tends to perform very well in high dimensional feature spaces.

Nonseprerable classes

We will now use the following constraint:

$$y_i[w^T x + w_0] \geq 1 - \xi_i$$

We know have the following categories:
- $\xi_i = 0$ Vectors outside the band and correctly classified
- $0 < \xi_i \leq 1$Vecotrs inside the band and correctly classified
- $\xi_i > 0$ Vectors that are classified

The optimalization goal becomes:

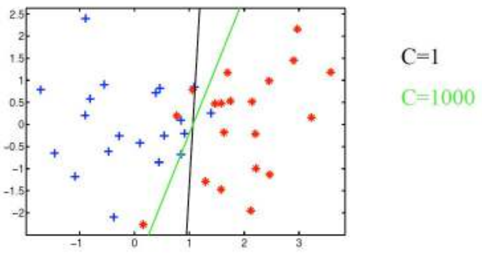
$$\begin{aligned} &\min ||w||^2 + C\sum_{i=1}^{N}\xi_i \\ &y_i[w^T x_i + w_0] \geq 1 - \xi_i, \quad i = 1, 2, ..., N \\ &\xi_i \geq 0 \quad \forall i \end{aligned} \tag{7}$$

This results in the same equation with different constaraints:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j x_i^T x_j\right) \tag{8}$$

$$\text{subject to:} \sum_{i=1}^{N}\lambda_i y_i = 0 \qquad 0 \leq \lambda_i \leq C, \quad i = 1, 2, ..., N \tag{9}$$

Tradeoff parameter C weighs the contribution between the training error and the structural error. Its values if often optimized using cross-validation.



Data transformation

The decision boundary is only linear, and altough we can apply the SVM on a non-lionearly-seperable data set this will always result in errors. By mapping the data we can make it linearly seperable. If we intoduce:

$$K(x, y) = \phi(x)^T\phi(y)$$

With:
- $\phi(\cdot)$: a mapping function

Than we can apply this to the SVM optimalization problem:

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i,j}\lambda_i\lambda_j y_i y_j K(x_i, x_j)\right) \tag{10}$$

$$\text{subject to:} \sum_{i=1}^{N}\lambda_i y_i = 0 \qquad \lambda \geq 0 \tag{11}$$

We only defire $K(\cdot,\cdot)$ and forget about $\phi(\cdot)$.
The Kernel-trikc:
- The idea to replace all inner products by a single function, the kernel function K, is called the kernel trick
- It implicitly maps the data to a (most often) high dimensional feature space
- The practical computational complexity does not change (except for computing the kernel function)
- $K = (< x_i, x_j > +c)$ is often used to create a pylynominal

Advantages and Disadvantages SVM

+ Effective in high dimensional spaces
+Number of dimensions can be higher than number of samples ($l > N$)
+Memory efficient
+Versitile due to different kernels
-Poor peformance when number of features > number of samples
-SVMs do not provide probability estimates
-Scaling dependent

5.4 Linear Classifier Showdown

Simple Data

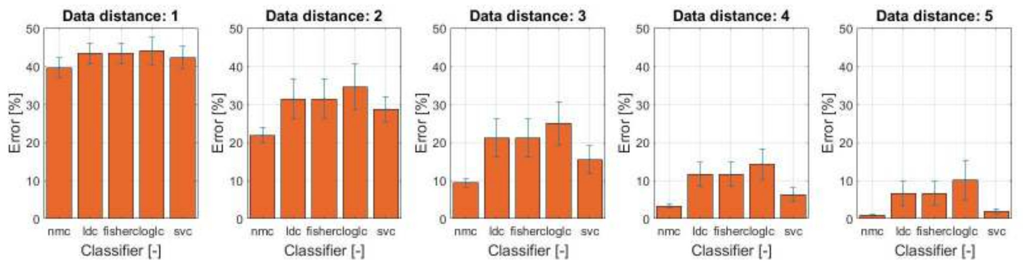


Figure 2: Influence of data distance on error, a training set of 50 has been used (25 for both classes)

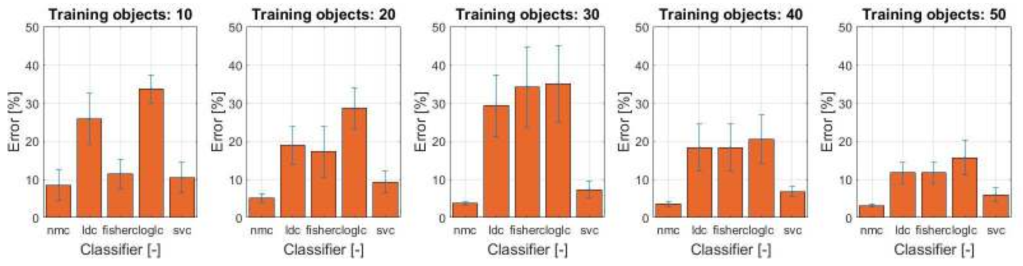


Figure 3: Influence of training data size on error, a data distance of 4 has been used

Difficult data

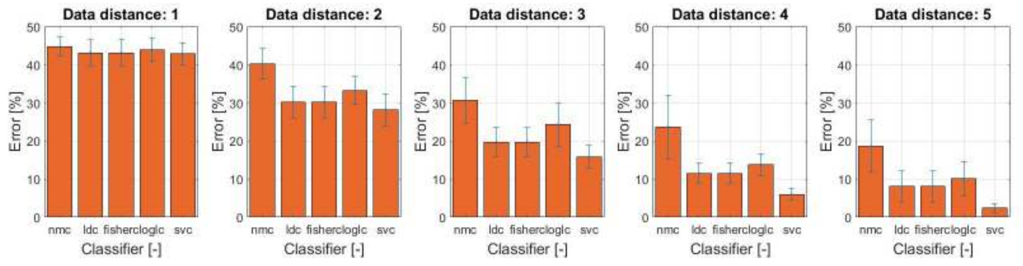


Figure 4: Influence of data distance on error, a training set of 50 has been used (25 for both classes)

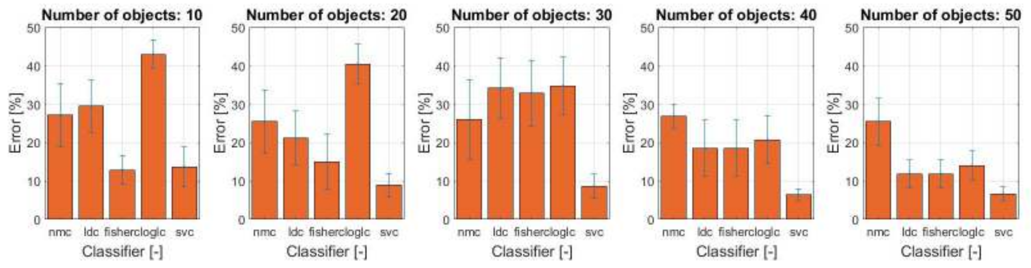


Figure 5: Influence of training data size on error, a data distance of 4 has been used

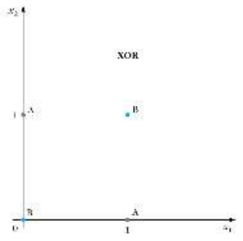
Book section: 4.1-4.4, 4.6, 4.9 (first two pages), 4.10, 4.12-4.14, 4.18, 4.19 (up to but not including 4.19.1)

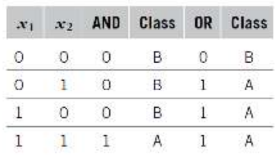
Note: Not all material of this chapter is included in this summery!

6 Nonlinearity

6.1 Multilayer persecptron

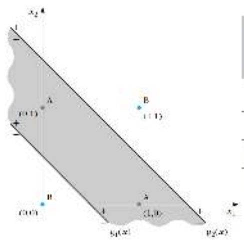
XOR problems cannot be solved with a single linear classifier.

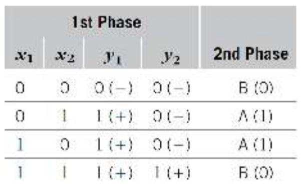




x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A

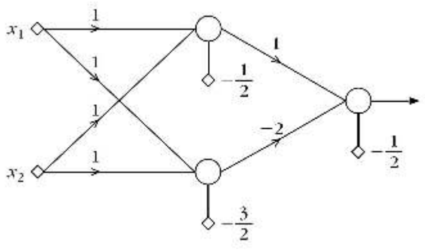
However if we use two classifiers we can are able to separate the classes.





1st Phase				2nd Phase
x_1	x_2	y_1	y_2	
0	0	0(−)	0(−)	B(0)
0	1	1(+)	0(−)	A(1)
1	0	1(+)	0(−)	A(1)
1	1	1(+)	1(+)	B(0)

This can be done with a multilayer perceptron.



If an step activation function is used (alternatives are possible like a sigmoid) then the multilayer perceptron maps l-dimensional input vectors using p neutrons in the hidden layer onto the vertices of the hypercube of unit side length in the p-dimensional space. In this case each neuron in the hidden layer creates a hyperplane.

Two or three layers

The first layer of neurons divides the input l-dimensional space into polyhedra, which are formed by hyperplane intersections. All vectors located within one of these polyhedral regions are mapped onto a specific vertex of the unit hypercube.The output neuron provides the multilayer perceptron with the potential to classify vectors into classes consisting of unions of the polyhedral regions.
Two layer perception: can separate classes each consisting of unions of polyhedral regions but not any union of such regions.
Three layer perceptron: can separate classes resulting from any union of polyhedral regions. The neurons of the first layer form the hyperplanes, those of the second layer form the regions, and finally the neurons of the output layer form the classes.

Training

Gradient decent: we canot take a derivative of a step function (use sigmoid), or we might end up a a local minimum. (initialize a lot of times) **back propagation** is a form of gradient decent.