Classification

Author: Magne H. Johnsen

December 18, 2017

1 Introduction to classification

The term classification is used for a variety of (related) concepts. A person deciding between "right" or "wrong" based on learned ethics and moral is doing a form of classification. The two classes are dependent on the different humans interpretation and moral opininons, and the decision is based on a form of reasoning. In sum we can call this **human intelligence (HI)** based classification. If we try to automate this process, the system has to learn how humans think, so called **artificial intelligence (AI)**.

However, in this course we will concentrate on the other end of the classifier alternatives. We will assume that a human is not involved in the classification process. Instead a sensor is measuring some values or signals which can be transformed into appropriate features suited for classification. Further, the classes are logically and scientifically defined, i.e. not suspect to different humans interpretations (like "right" and "wrong"). The feature based classification is performed by a program on some digital hardware. However, first the program has to learn (from example features) the behaviour of the different classes. Typically this behaviour is represented by mathematical class models. We often say that this classifier is based on **statistics and machine-learning**. This type of classifiers is much more used than the AI-based. Further, on the same problem they usually outperform the AI-alternative.

Some practical problems have only two classes. However, we can not regard these cases in the same way as we do detection. Assume we set up a camera in the mountains to take pictures of everything moving. However we are only interested in the rare species arctic fox. In a detection mode we want to know if the animal moving is an arctic fox or not; i.e.. there is only a single 'class' of interest, namely arctic fox. If such a fox is detected the next step is to decide upon the sex, i.e. male or female. We now have two classes, both of equal interest. Thus we have a classification case.

1.1 Introducing basic terminology by an example

We humans are quite good to do classification. Let us assume that we are using our eyes to decide on whether a person is a male or female.

Obviously we use several cues/features to help us decide, i.e. height, body shape, hair length, aso. Still it happens that we do some mistakes, i.e. the features overlap (see figure 2) and thus our decision strategy are not error free. However it is clear that the **error rate** decreases as the number of features increases. Thus if we also could use our ears, i.e. the person's voice; this would obviously help in some cases. Further skin color should be added as an "eye feature" since the mean height of Asians is lower than for Eurasians and most Africans. On the "output side" we could introduce **conditions**, i.e. female given Asian. Alternatively we could introduce combinations thorough **subclasses** , i.e "Asian female".

How is it that we as humans are able to do the above classification? We are obviously not born with this ability. The answer is of course by observing and learning. In the classification terminology this is called **"supervised learning/training"**. We observe a person and is told by a "teacher'/supervisor" that this is an Asian female. However we must also learn to **"generalize"**. Every human is special (except for identical twins...), thus we need to classify a person we never have seen before. It is therefore obvious that we need to learn by observing more than one example from each class. In fact this is often the core of learning by example; you have to see/learn a representative number of each class in order to do a qualified "guess" of class for an unseen person. A golden rule for learning is that ".. the more data the better..".

(a) Typical heights

(b) Less typical heights

Figure 1: *Male versus female heights*

I some cases we do not have a teacher/supervisor. We just suspect that some data can be organized into a finite number of distinct groups. Even the optimal number of groups is unknown and needs to be estimated as a part of the analysis. This problem type is called **"unsupervised training"** or **"clustering"**. Using the same case as above one unsupervised problem can be to organize people into groups based on skin color. Obviously there is no single solution to this, still we should be able to learn from most of the "suggestions" coming out of the clustering procedure.

Turning back to the features, most of them are continuous valued. Let us use height as an example. Even if this is a continuous variable we usually quantize the value. However we must be sure to use an accurate enough resolution. That is probably the reason that most people is used to

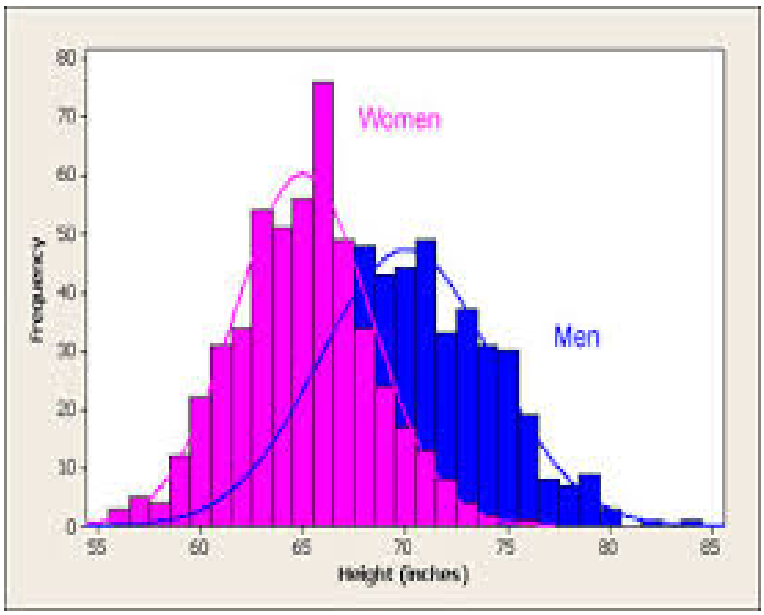

Figure 2: *Histogram of heights*

applying centimeter, however even coarser quantization (e.g. inches) is sometimes used . Further, when seing a person we of course just take a qualified guess with respect to the height. Thus we **estimate** the input feature height.

Finally the input stimulus from our different senses are sent to the brain for decision. As today we (luckily..?) do know little or nothing about how the brain implements the decision. However this is the core of the HI based classification system. Thus a variety of mathematically based method and algorithms are implemented. For many applications we are able to evaluate the quality of them by comparing them to the HI performance. However for most of the applications we have no human baseline as the features are derived from a set of non-human like sensors, like ultrasound, radar, sonar, hyperspectral camera, accelerator, strain gauge aso. Further for all applications there is a typical maximum tolerable error rate.

1.2 A generic classification system.

In many practical problems classifiation is either a mandatory part or even the main goal. Figure 3 shows a complete classification system. An application dependent sensor measures a signal from a physical source. As an example a microphone will convert sound pressure to an electrical signal. The sensor forwards a signal which seldom is directly applicable for classification. The discriminative information is often embedded/hidden in the redundant signal. Thus in a following step features (hereby named x) are extracted, typically in the form of a vector. The features ideally contains enough discriminative information to yield a satisfactory performance with respect to class error rate. Then the main part concists of deciding which class ω_i $i = 1, \ldots, C$ the system "believes" the signal/feature vector belongs to. This decision can of course be correct or wrong. As an integrated part of the design one must estimate the performance of the classifier. How this is done is explained in a separate subchapter.

There exists a variety of different classifiers. They differ both with respect to method/principle and complexity. In this course we will limit ourselves to a few of the most used and simplest.

However, at the end of the lectures we will give a short overview of the state-of-the art. Finally we will stress the following. It is the total system performance which is important. It does not help to use a complex classifier method if the error rate is 30%. In many cases a much better strategy is to search for better sensors and/or feature extaction methods! It is for instance well known that in speech recognition the choice of microphone can result in success or disaster.....

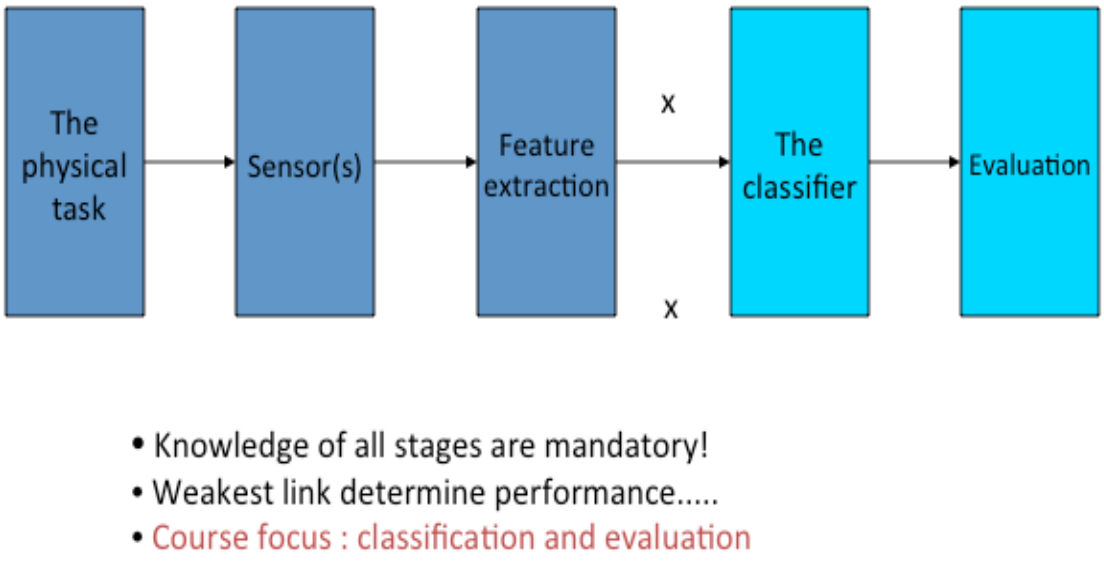

Figure 3: *A complete classification system*

In classification the output is a discrete value correponding to one out of C possible classes. **Regression** is a concept related to classification. The difference lies in that the regression outputs can be any continuous valued vector, often with the same dimension as the input feature vector. Thus we can define regression as a kind of mapping method. Models and training methods for classification are often also used for regression. Regression is a central part in a variety of applications, for instance :

- Prediction of future input values
- A signal in noise is input and the same signal with (much) less noise is wanted as output
- Finding connection with pair of values, i.e. respectively the input and the wanted output

In this course we will however focus on classification.

2 Different classifier types.

We can divide the classifiers into two groups based on static versus dynamic input/outputs. Static means that the the feature input x does not have any temporal information and belongs to a single

class. Dynamic means that an input vector sequence (i.e. embedded temporal information) belongs to a single class or to a sequence of classes. An example of the latter is speech recognition where the sound is transformed into a vector sequence which is classified into a sentence (sequence of words).

In this course we will restrict ourselves to static classifiers. However, dynamic based classifiers will be shortly described as a part of the state-of-the-art presentation.

2.1 Different problem types of classification.

We can define three main categories, as illustrated in figure 2.1 for a two-dimensional feature vector $x = [x_1 x_2]^T$ and two classes ω_1 and ω_2. The simplest category is called a **linearly separated** problem. As the name indicates we can in this case separate the two classes without errors by a simple line. If the input dimension is three this decision border is a plane. For higher dimensions we use the term "hyperplane". Note that there in general is an infinite number of lines/planes/hyperplanes that will give error free decision. However, an intuitive and robust choice would be to use the border with the same minimum distance to both classes. If more than two classes exist a corresponding decision border is needed for each pair of classes. A classifier which uses linear decision border is (logical enough) called a **linear classifier**. Such classifers seldom have the best performance, however they usually are simple to design and apply.

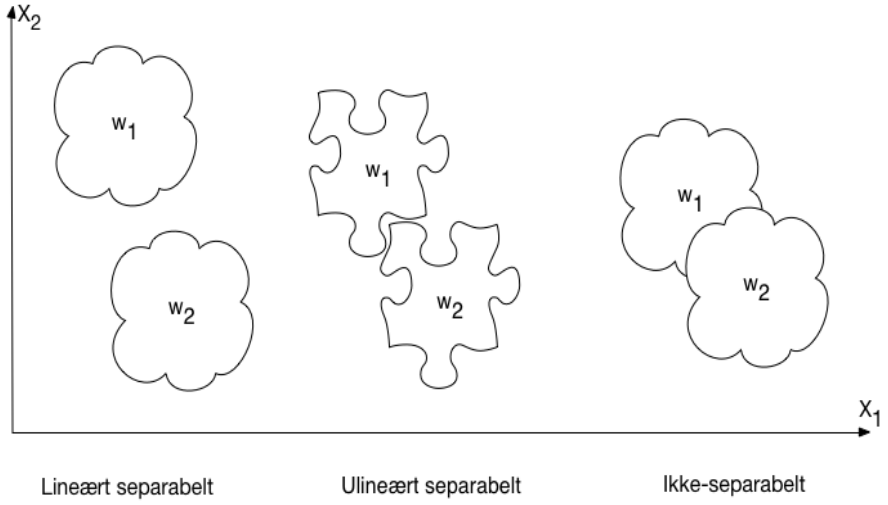

Figure 4: *Three different classification types*

For the **nonlinear separated** problem type it is also possible to find decision borders which give error free performance. However, these borders can not be hyperplans. They have to have a nonlinear form, thus the corresponding classifiers are called **nonlinear classifiers**. In the last problem type the classes overlap in the input room. Thus it is not possible to find decision borders that give error free performance. "Unfortunately" most practical problem are of this third type, i.e. so called **nonseparable** problems. Whether one should go for a linear or nonlinear classifier in this case is a balance between complexity and performance (error rate). If the resulting error rate turns up to be too high for practical use, a good strategy is to search for alternative/better features (and even sensors), hoping that this will result in a more separable problem.

We will later see that the classifier performance also is highly dependent on how we design/train them. Central ques here are the **amount of training data** (class labeled features) and also the

representativeness of the data. .

2.2 The optimal classifier.

The theoretically optimal classifier in the sense of minimum error rate has a statistical form. In order to introduce this classifier we have to refresh our basic knowledge of statistics and probabilities. Let us assume that that our features are represented by a continuous valued vector x and that we have $\omega_i,\ i = 1, C$ classes. Only four basic formulaes are needed to describe not only this classifier but any statistically based classifer :

- A priori probablities $P(\omega_i)\ \ i = 1, \ldots, C$ where of course $\sum_i P(\omega_i) = 1$
- Class conditioned densities $p(x/\omega_i)\ i = 1, \ldots, C$ where $\int p(x/\omega_i)dx = 1\ \ i = 1, \ldots, C$
- A posteriori probabilities $P(\omega_i/x)\ \ i = 1, \ldots, C$ hvor $\sum_i P(\omega_i/x) = 1$ for any x
- Joint distributions $p(x, \omega_i)\ i = 1, \ldots, C$ where :
  - $\sum_i p(x, \omega_i) = p(x)$ for any x
  - $\int p(x, \omega_i)dx = P(\omega_i)\ \ i = 1, \ldots, C$
  - $p(x, \omega_i) = p(x/\omega_i)P(\omega_i) = P(\omega_i/x)p(x)\ \ i = 1, \ldots, C$ (named **Bayes law**)

Bayes law can be rewritten as :

$$P(\omega_i/x) = \frac{p(x/\omega_i)P(\omega_i)}{p(x)} \tag{1}$$

One can show (not part of syllabus here) that the following decision rule leads to the least possible error rate :

$$x \in \omega_j \quad \Leftrightarrow \quad P(\omega_j/x) = \max_i P(\omega_i/x) \tag{2}$$

This decision rule is called the **Bayes Decision Rule (BDR)**.
Applying Bayes law the BDR can be written as :

$$x \in \omega_j \quad \Leftrightarrow \quad p(x/\omega_j)P(\omega_j) = \max_i p(x/\omega_i)P(\omega_i) \tag{3}$$

This is due to that the denominator $p(x)$ in equation (1) is the same for all classes. A BDR-based classifier is logically enough also called a **Maximum A Posteriori (MAP)** klassifiserer.

However, the cruical point is that the BDR/MAP classifier only is a **theoretically** optimal classifier. The involved densities and probabilities are never known exactly. This is especially critical for the class conditoned densities. Usually one has to choose a parametric form (for instance a Gaussian). And then one will need a set of (training) data to estimate the parameters (for a Gaussian the mean and variance). This is the same kind of challenge as was met in the course Digital Signal Processing where one had to choose a parametric process (i.e. AR[P]) to model physical signals and thereafter estimate the model parameters.

However, choosing a model and estimating model parameters transforms a theoretically optimal classifier to a practical but **sub-optimal** classifier. This again opens up for searching for alternative classifier types (than BDR/MAP) which may show a better performance! In this course we will start with the sub-optimal BDR/MAP classifier, also called a "Plug-in-MAP" classifier.

Many classifiers are not statistically based, i.e. they do neither apply densities nor probabilities. The two largest groups are called respectively discriminant classifiers and template (reference) based classifiers. The linear classifier belongs to the discriminant category.

The three categories of classifiers mentioned above will be described in the following subchapters.

2.3 The Plug-in MAP classifier

This classifiser type uses BDR (equation (3)) based a chosen parametric density form for which the parameters must be estimated by a set of training data. The most popular parametric form is the Gaussian density

$$p(x/\omega_i) = N(\mu_i, \Sigma_i) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_i|}} \exp\left[-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)\right] \quad i = 1, \ldots, C \tag{4}$$

The real (but unknown) density usually have a complex and "noisy" form with multiple peaks. This is especially the case when the input feature dimension $D = dim(x)$ is large. Thus the "bell shaped" Gaussian form is seldom a good approximation. However, for a large number of applications a density in the form of a weighted mixture of Gaussians has shown succesful, i.e. resulted in a good approximation to the real densities. This density model is called a **Gaussian Mixture Model (GMM)** and have the form

$$p(x/\omega_i) = \sum_{k=1}^{M_i} c_{ik} N(\mu_{ik}, \Sigma_{ik}) = \sum_{k=1}^{M_i} \frac{c_{ik}}{\sqrt{(2\pi)^D |\Sigma_{ik}|}} \exp\left[-\frac{1}{2}(x-\mu_{ik})^T \Sigma_{ik}^{-1}(x-\mu_{ik})\right] \; i = 1, \ldots, C \tag{5}$$

where the weights fulfil the constraint $\sum_k c_{ik} = 1$. An important part of the model approximation is to find a good choice of the number of mixtures M_i for each class. ω_i. Note that Gaussian based classifiers are nonlinear as they have quadratic terms (in x) in the exponent.

The priors $P(\omega_i), \quad i = 1, \ldots, C$ must be estimated as well. However good estimates are usually not as critical as for the densities. The estimates are usually found either by knowledge of the problem, or by doing a simple frequency count in the training set. For many applications even equal priors $P(\omega_i) = 1/C, \quad i = 1, \ldots, C$ are assumed without any noticable performance degradation.

2.4 The linear classifier

In chapter 3.1 it was said that a linear classifier is adequate for a linear separable problem. However, more or less all practical problems are non-separable, thus a linear classifier will usually give a worse performance than a nonlinear classifier. But the performance difference is sometimes acceptable small, and a linear classifier can be preferred due to simplicity.

In a discriminant classifier each class is described by a (so called dicriminant) function $g_i(x)$ and the decision rule is given by

$$x \in \omega_j \quad \Leftrightarrow \quad g_j(x) = \max_i g_i(x) \tag{6}$$

Note that this decison rule is quite similar to the BDR. In fact one can define the MAP classifier as a special case of a discriminant classifier. However, the discriminant functions $g_i(x)$ can have a much more general form.

The linear discriminant classifier is defined by the function

$$g_i(x) = w_i^T x + w_{io} \quad i = 1, \ldots, C \tag{7}$$

where w_{io} is offset for class ω_i. For $C > 2$ classes this can written in a compact matrix form

$g = Wx + w_o$ where g and w_o eare vectors of dimension C (number of classes) and W is a CxD matrix. $C = 2$ is a special case where we can rewrite the decision as $x \in \omega_1 \quad \Leftrightarrow \quad g_1(x) - g_2(x) > 0$. Thus we can define a single function $g(x)$ and a decision rule $x \in \omega_1 \quad \Leftrightarrow \quad g(x) > 0$ and of course $x \in \omega_2$ else.

2.5 The template based classifier.

The decision rule is simple for this kind of classifier. The input x is matched towards a set of references (templates) which have the same form as x. The rule finds the reference which is closest (or most "similar") to x and assumes that x belongs to the same class as this reference. This method is called "nearest neighbour - NN". Several variants of this classifier category exist dependent on

- Which decision rule that is used (other than NN)
- The kind of distance or similarity measure that is used.
- How the references are chosen and used

Assume $k = 1, \ldots, M_i$ references ref_{ik} for class ω_i, i.e. a total of $M = \sum_i M_i$ references. The most general reference set is then given by $ref_{ik} = (\mu_{ik}, \Sigma_{ik})$ $k = 1, \ldots, M_i$; for each class ω_i $i = 1, \ldots, C$. A corresponding distance measure (called the Mahalanobis distance) is defined by

$$d(x, ref_{ik}) = (x - \mu_{ik})^T \Sigma_{ik}^{-1} (x - \mu_{ik}) \tag{8}$$

A more advanced decision rule is called KNN. Here one finds the $K > 1$ nearest references. The decision is then based on a majority vote within this K. If several classes end up with the same number, the closest class among them is chosen. .

The parameters in the reference set (as given by equation (8)) must be estimated from data in a training/design phase. Especially the covariance matrixes Σ_{ik} requires a lot of data to be well estimate. One therefore often uses simplifications of the Mahalanobis distance, i.e. a) common class matrix $\Sigma_{ik} \rightarrow \Sigma_i$ b) global matrix, i.e. $\Sigma_i \rightarrow \Sigma$ or c) a simple Euclidian distance as follows

$$d(x, ref_{ik}) = (x - \mu_{ik})^T (x - \mu_{ik}) \tag{9}$$

i.e. $\Sigma \rightarrow I$ (the unity matrix).

An advantage of the Euclidian case is that the parameters $ref_{ik} = \mu_{ik}$ can be chosen directly from the the training data set. This is discussed later on and also in connection with the subchapter on clustering.

A similarity measure can be used as an alternative to a distance measure, i.e. they are in principle inverse measures. Then the decision rule is to choose the reference which gives the highest similarity score to the feature x. Thus the Plug-in-MAP classifier can be thought of as a tempate based classifier, i.e. equation (3) gives the discriminant functions while equations (4) and (5) give the similarity measures (assuming Gaussian class densities).

2.6 Analyzing the features.

In most cases the feature extraction method is given by the choice of the sensor and the knowledge of the application. However, for some cases we will have the opportunity to choose among several types of features. Anyhow, it is always useful to be able to inspect the features with respect to class discrimination and the form of the distribution. This information is important regarding which classifier type to choose. To visualize the features one can use the so called histogram. A histogram shows an approximation to the distribution of a scalar feature. All the training set features are quantized into one of a finite number of values (bins). Then the (relative) numbers of features in the training set in each bin are shown. Note that when the bins increases (i.e. the bin width decreases) the histogram converges towards an estimate of the feature distribution. In figure 5 we show histograms for three different classes for a chosen feature. In figure 6 we show the same for another feature. We easily see that the first feature is good for discriminating class 2 from the others. The same applies to the second feature and class 3.

In all practical cases we will us a feature **vector**. It is custom to analyze/visualize one vector element at a time. However, this method will not detect any correlation between the vector elements. Such a correlation can help to discriminate classes even if the histograms do not show this. This is examplified in figure 2.1. We clearly see that using both features X_1 and X_2 give a more discriminatory overview than only looking into one direction at a time. Methods exist that extracts the correlation in ways which helps classification, however this is not included into this course.

The histograms are also useful for choosing good alternatives for classifier type and structure. For feature 2 (figure 6) the histogram of class 3 clearly indicates a Gaussian like shape, i.e. a single Gaussian model. However, the histogram for class 1 has two "peaks", thus a 2-mix GMM would be a better class model. It is custom to input the whole feature vector to the same structure. Thus a reference based or linear classifier is a potential alternative to a Gaussian based classifier in this case.

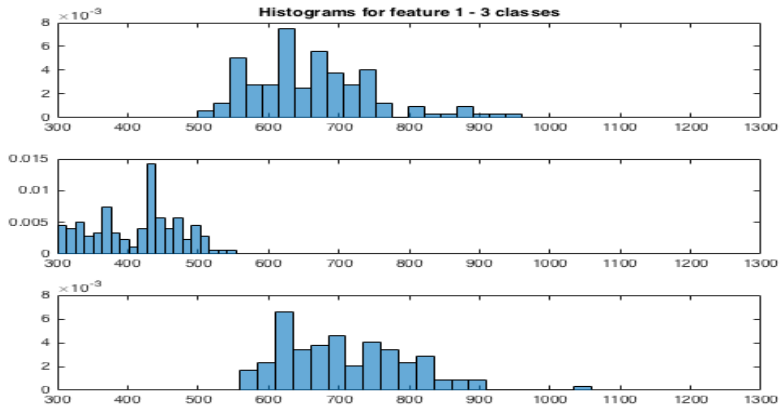

Figure 5: *Class histograms for feature 1*

2.7 State-of-the-art classifiers

A variety of classifiers exist. Some of them have great similarities to the three we have described. For example the classifier called "Support Vector Machine - SVM" started as a simple linear classifier, however the training/design method was based on complex discriminating principles. Today

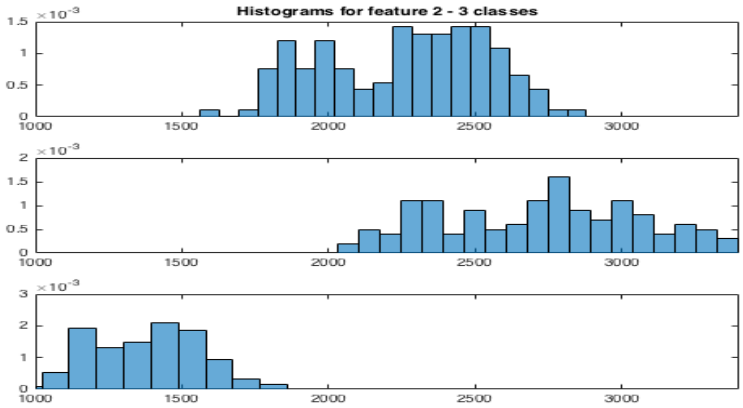

Figure 6: *Class histograms for feature 2*

SVM is only used as a nonlinear classifier due to so called kernel based techniques.

The two largest groups of classifiers used today are respectively the **nonlinear and the context-based classifiers**. In the first group the neural network based classifiers dominates. They have shown impressive performance, especially for problems where one has no good model. Much of the good performance is due to a more complex structure called a deep neural network (DNN), and a correponding new training technique. However, the potentially success is conditioned on that one has access to a large amount of training data. Thus deep (neural network) learning is often mentioned together with an other buzzword, namely **"Big Data"**. The combination of large data sizes and complex DNNs demands large resources during training with respect to memory and CPU-power. Especially graphical cards have shown powerful for this type of training. When neural networks was introduced late 1980s, the training was called **machine learning**. This is another buzzword today and now includes nearly all kinds of data driven automatic learning strategies.

Context-based classifiers are primarily used for sequence classification where much of the class information is embedded in the temporal evoluation within the sequence. The statistical based Hidden Markov Model (HMM) classifier is probably the most known within this category. Speech recognition and gene classification are examples where temporal evolution is cruical, and HMM is therefore state-of-the-art for this kind of problems.

3 Design/training and testing/evaluation

The first step in the design phase is to choose the classifier category and the structure/form. The next step is then to estimate the parameters of the chosen structure. For the Plug-in-MAP category the GMM form in equation (5) is the most popular structure. The corresponding parameters to be estimated are the mixture weights, the means and the covariance matrixes of the Gaussians. Usually one has to choose the mixture numbers, however searching for good numbers is included in some design techniques.

In nearly all training cases one will need a so called **labeled training set** for the parameter estimation. This set has to be large to achieve robust estimates. If the data set is too small the classifier will perform much worse on unseen data then on the training set, i.e the classifier does not **generalize**. Further the data has to be **representative**, i.e. all variants of input/class pairs should be present in a large enough scale. Further, the data must be unbiased, i.e. using identical sensors, recording conditions and so on.

Let us assume we have a training set of size N. We name the input data $X_N = \{x_1, \ldots, x_N\}$ and the corresponding class labels $T_N = \{t_1, \ldots, t_N\}$ (so called **targets**). This class information can either have the simple scalar form $t_i = \omega_j \;\; \leftrightarrow \;\; x_i \in \omega_j \;\; i = 1, \ldots, N$ or a vectorial form $t_i = [0, \ldots, 0, 1, 0, \ldots, 0]^T$, i.e. only element j in the vector is 1 ($t_i(j) = 1 \;\; \leftrightarrow \;\; x_i \in \omega_j \; i = 1, \ldots, N$). In the latter case the dimension of the target vector is the same as the number of classes C.

Assuming a representative data set one could ask if we can know/guess what size N we need for training. In many cases knowledge of the problem can help us to this. However, there is a rule which always is correct : "the more data the better...".

Most training algorithms are iterative. One applies the training set several times and the algorithm should give better performance (correctness) on the training set for each iteration. However, a so called **validation set** should be used in parallel. This set is not used for estimating parameters, but only evaluated with respect to performance. When the performance for this set stops improving the training should also stop. As both the training and validation set are used for design, a third **test set** must be used for evaluation of the true/unseen performance. Typically, the performance is best for the training set and worst for the test set. The difference between the validation set and the test set should not be large as it tells us how well the classifier generalizes. If the difference is (too) large the reason can be that

- the training/validation sets are too small and/or not representative
- wrong category classifier is chosen
- wrong structure is chosen

Assume a test set of size M. The performance p_M of the test set gives only an **estimate** of the true (but unknown) performance p_t for the trained classifier. This true performance p_t can only be found if the test set is unbiased and is infinitely large. To explain this let us assume we have two different test sets of the same size. It is understandable that we will find some difference in the performances for these two sets. The same applies if we extend this to any finite number of finite sized test sets! Thus we can think of the performance as a statistical variable, i.e an estimate of the true performance. Using statistical theory it is common to give the true performance as a 95% confidence interval. This says that with 95% certainty will the true performance p_t fulfil

$$p_M - \alpha_l \leq p_t \leq p_M + \alpha_u \tag{10}$$

The bounds α_l and α_u are dependent on p_M and M. Logically these bounds will shrink towards zero as the test set size M increases. See chapter 3.5.2 for more on this subject.

In the following section we will describe the training process for our three different classifiers.

3.1 Plug-in-Map using Maximum Likelihood (ML) based training

The ML algorithm estimates the parameters for a single class at a time. Given a training subset $X_{N_i} = \{x_{i1}, \ldots, x_{iN_i}\}$ where all inputs are independent and all belong to the same class ω_i. We further use the notation $\Lambda_i = \{\mu_i, \Sigma_i\}$ for the unknown parameters we want to estimate. The likelihood for the subset is given by

$$L[X_{N_i}, \Lambda_i] = \prod_{k=1}^{N_i} p(x_{ik}/\Lambda_i) \tag{11}$$

A density has known parameters and is a function of the variable x. A likelihood, however, has known values of x. Thus if also the parameters are known the likelihood is a number. However, we can have situations where the likelihood is a function of unknown parameters. This is obviously the case during training since the goal is to estimate the unknown parameters Λ_i of the class.

When the distributions $p(x/\Lambda_i) = p(x/\omega_i)$ are assumed to have an exponential form (for example Gaussian - see equations (4) and (5) in subchapter 3.3), the logarithm will have a closed analytical form. Thus one can simplify the equation (11) by instead using loglikelihood (LL), i.e.

$$LL[X_{N_i}, \Lambda_i] = \log[\prod_{k=1}^{N_i} p(x_{ik}/\Lambda_i)] = \sum_{k=1}^{N_i} \log[p(x_{ik}/\Lambda_i)] \tag{12}$$

As mentioned the x-values in equation (12) are known while the parameters are unknown. Thus one wishes to find the parameters which maximize this LL. This is a problem which is equivalent to curve fitting, for instance finding the line which has least possible distance sum to a set of points. Maximation of a function is done by taking the gradient and finding the parameters which give the gradient the value zero.

$$\nabla_{\Lambda_i} LL[X_{N_i}, \Lambda_i] = \sum_{k=1}^{N_i} \nabla_{\Lambda_i} \log[p(x_{ik}/\Lambda_i)] = 0 \tag{13}$$

3.1.1 ML-training in the single Gauss case.

If we decide on using single Gaussians as class densities we can easily derive optimal closed expressions. These expressions are called **estimators** and when we apply a known set of x-values in the estimator we get a concrete value called an **estimate**. Starting with equation (4) we easily get

$$\log[p(x_{ik}/\Lambda_i)] = -\frac{D}{2} log(2\pi) - log|\Sigma_i| - \frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \tag{14}$$

Taking the gradient with respect to μ_i we get

$$\nabla_{\mu_i} \log[p(x_{ik}/\Lambda_i)] = \Sigma_i^{-1}(x - \mu_i) \tag{15}$$

Using equation (15) into equation (13) we easily find that the estimator for the mean is given by equation (16). Doing the same exercise for the covariance matrix we end up by equation (17). Note that the original covariance estimator uses the true but unknown mean, while we for obvious reasons have to replace it with the estimate.

$$\hat{\mu}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} x_{ik} \tag{16}$$

$$\hat{\Sigma}_i = \frac{1}{N_i} \sum_{k=1}^{N_i} (x_{ik} - \hat{\mu}_i)(x_{ik} - \hat{\mu}_i)^T \tag{17}$$

The above two estimators are called respectively a sample mean and a sample covariance estimator.

It can be shown that the mean estimator is an MVU-estimator. This is not the case for the covariance estimator, however it can be shown to be asymptotical efficient.

3.1.2 ML-training using GMM densities

If we choose GMM as the class densities (equation (5)), we end up with the logarithm of a sum in equation (12). This leads to an insintric gradient based equation. The estimates appears on both sides of the equal sign, i.e. mathematically we will have $\Lambda_i = f(\Lambda_i)$ where the function $f()$ is application dependent. However, there exists an efficient iterative algorithm called **Expectation Maximation - EM** which will find a (suboptimal) solution. The term "Iterative" means that we start with a chosen initial set of estimates $\Lambda_i(0)$ which we put into the function to get a new and better estimate (higher LL), i.e.

$$\Lambda_i(m) = f(\Lambda_i(m-1)) \;\Leftrightarrow\; LL(m) > LL(m-1) \quad m = 1, 2, \ldots \tag{18}$$

The algorithms stops after a finite number of iterations, i.e. when the improvement in LL gets less than a chosen value.

Later we will present the basics of clustering. We will then relate the EM-algorithm to clustering and show the similarities between the two methods.

3.2 MSE based training of a linear classifier.

For $C > 2$ classes the output of the linear classifier was given by the discriminant vector $g = Wx + w_o$ with dimension C. This expression can be rewritten as $g = [W\ w_o][x^T\ 1]^T$. If we then redefine/merge the input and matrix to $[W\ w_0] \rightarrow W$ og $|x^T\ 1]^T \rightarrow x$ we end up with $g = Wx$.

The linear classifier is, in contrast to the the Plug-in-MAP classifier, not statistically based. Thus one can not use an optimization criteria like ML during training. Since most practical problems have a non-separable form, a criteria which works well for this case should be chosen. The most

popular variant is the "Minimum Square Error - MSE" . MSE requires target values at the output, i.e. in the vectorial form as given in the introduction to subchapter 3.

$$MSE = \frac{1}{2}\sum_{k=1}^{N}(g_k - t_k)^T(g_k - t_k) \tag{19}$$

The total matrix W is trained as one single entity, i.e. the total training set for all classes is used simultaneously. Thus both the training technique and the classifier are named discriminative. This is in contrast to ML-training of the Plug-in-MAP classifier where each class was trained separately.

The output vector $g_k = Wx_k$ corresponding to a single input x_k is a continuous valued vector. In order to match this output to the target vector with binary values $\{0, 1\}$ a heavyside function should be ideally be used. However, we shall see that we need a smooth function that has a derivative. A squashing function called a sigmoid (see figure 3.2) is an acceptable approximation for the Heavyside function

$$g_{ik} = sigmoid(x_{ik}) = \frac{1}{1 + e^{-z_{ik}}} \quad i = 1, \ldots, C \tag{20}$$

Here we have $z_k = Wx_k$ (which was previously called g_k). Further, the index i points to vector element number i

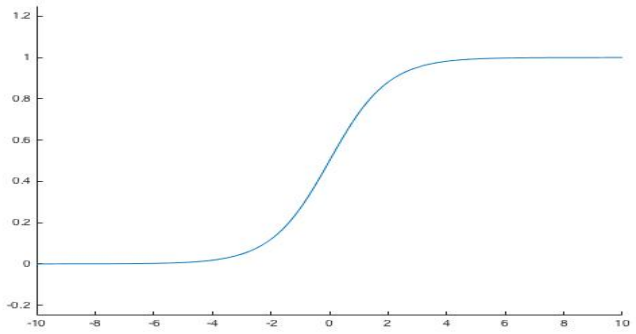

Figure 7: *Sigmoid*

A linear classifier with binary targets thus has the structure given in figure 3.2

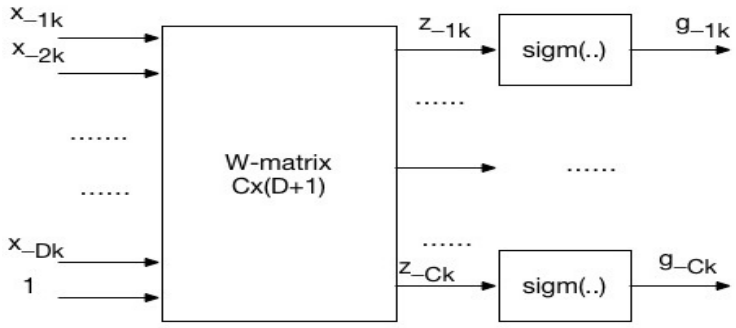

Figure 8: *Sigmoide*

As for the GMM-based Plug-in-MAP classifier there exists no explisit solution to equation (19). One therefore has to use a gradient based technique, i.e. update the W matrix in the opposite direction of the gradient. Using the chain rule for gradients we get

$$\nabla_W MSE = \sum_{k=1}^{N} \nabla_{g_k} MSE \; \nabla_{z_k} g_k \nabla_W z_k \tag{21}$$

It is easily shown that

$$\begin{aligned} \nabla_{g_k} MSE &= g_k - t_k \\ \nabla_{z_k} g &= g_k \circ (1 - g_k) \\ \nabla_W z_k &= x_k^T \end{aligned}$$

where $g_k \circ (1 - g_k)$ means elementwise multiplication

Innserting the above gradients into equation (21) we get

$$\nabla_W MSE = \sum_{k=1}^{N} [(g_k - t_k) \circ g_k \circ (1 - g_k)] x_k^T \tag{22}$$

In order to reduce the MSE we have to move W in the opposite direction of the gradient, i.e.

$$W(m) = W(m-1) - \alpha \nabla_W MSE \tag{23}$$

where m gives the iteration number.

The step factor α is an important constant which must be chosen with care. A too large value will give a corresponding large change in W and results in large fluctuations in the MSE. Too small value will demand unnecessary many iterations in order to even get close to the minimum MSE. There exists however methods which will adapt the step factor during training. However often a "cut and try" technique will lead to satisfactory results.

The equation (22) tells us to use the whole training set of size N in each iteration, i.e. a so called "batch" based iteration. Often one prefers to update the matrix W more frequently. Thus one can choose to update based on subsets of size $L << N$. Thus is called a "semibatch" strategy. The extreme variant is when $L = 1$, i.e we update for each input/output value. This last method is called "online" or "realtime" updating. Which to use is typically application dependent. The step factor should increase as L decrease.

3.3 Training of a template based classifier using clustering

Training of a template based classifier is the same as finding good references. The simplest approach is not to train at all but instead to use the whole training set as templates. The main objection with this is the amount of processing, i.e. one has to calculate N distances for every new input x. Further it is not possible to find reference specific covariance matrixes Σ_{ik} (see equation (8)), i.e. one has to use the simplified distance versions described in subchapter 2.5.

A clearly suboptimal procedure is to use a subset $M << N$ of the references randomly drawn from the total training set. This will reduce the processing amount but will not help with regard to the lack of the reference covariances and thus the possible distance types. The subset is normally chosen such that the relative amount of references per class $M_i \;\; i = 1, C$ reflects the original class amounts N_i (i.e. the prior $P(\omega_i)$) in the total training set.

Drawing references randomly can lead to a reference set that is not representative for one or more classes. Thus a much better strategy is to use a method which guarantees good coverage. This method, called **clustering**, estimates both the necessary number of references, the references themselves and also the reference specific covariance matrixes.

Most clustering algorithms works on one class at a time. There exist some discriminative versions; however in the following we will concentrate of the class-for-class method. Thus we simplify our notation by omiting the class index i, i.e. $N_i \rightarrow N \, , M_i \rightarrow M \, , \mu_{ij} \rightarrow \mu_j, \; \Sigma_{ij} \rightarrow \Sigma_j, \;\; j = 1, M$. Note that a cluster and a reference means the same in the following!

The iterative procedure for clustering is relatively simple. It is based on successive increasing the number of clusters (i.e. number of class references)

1. Start with a single cluster $M = 1$
2. Use equations (16) and (17) to calculate $\Lambda_1 = \{\mu_1, \Sigma_1\}$ and the corresponding accumulated distance $D_1 = \sum_{k=1}^{N} d(x_k, \Lambda_1)$ where the distance is defined in subchapter 2.5.
3. Increase number of clusters $M = M + 1$. Split $\Lambda_1 \;\; \rightarrow \;\; \{\Lambda_1, \Lambda_2\}$ (see below for splitting technique.)
4. Then we iterate to find the best values of the M references.
   For q =1,... (iteration counter for a given cluster size M)
   - If $q = 1$ set $D_{M_0} = \infty$ (in practice a large number)
   - Classify each training set vector in the class to the closest of the M clusters and calculate the accumulated distance D_{M_q}.
   - If D_{M_q} is "clearly smaller" then the previous $D_{M_{q-1}}$ use the classification labels from above and the equations (16) and (17) to calculate/update $\Lambda_i \;\; i = 1, \ldots, M$. Increase

$q = q + 1$ and loop within step 4.
Else $D_M = D_{M_{q-1}}$ and go to next step 5

5. If D_M is "clearly smaller" than D_{M-1} increase number of references/cluster $M = M + 1$ by splitting.
   Else the clustering is finished for the specific class

Note that using the equations (16) and (17) leads to that the corresponding references do not exist as training vectors. However, this is usually not required. If for some reason this is needed, one simply replaces each reference by the closest training vector of the same class.

We did use the term "clearly smaller" twice in the algorithm above. Note that for each iteration q in step 4 the classification usually lead to some change of class labels as the references was modified in the previous iteration. This is equivalent to $D_{M_q} = \beta D_{M_{q-1}}$ where $\beta < 1$. Usually the algorithm converges towards a non-significant change in a few iterations, corresponding to a β close to 1. Usually one terminates when $\beta \approx 0.99$. In the extreme case we can experience no change at all in class labels, thus $D_{M_q} = D_{M_{q-1}}$.

The general form for splitting $M \rightarrow M + 1$ is based on that we first find the cluster Λ_s to split (in step 3 this is given). The most popular choice is to use the cluster which has most vectors classified to it. If reference specific covariance matrixes are used, an alternative is to choose the cluster with largest matrix norm. After choosing cluster Λ_s we define the new cluster, i.e. Λ_{M+1} by slightly modifying the reference $\mu_{M+1} = \mu_s + w$ where w is a uniformly distributed noise vector where the elements obey $\delta < w(j) < \delta \; j = 1, \ldots, dim(x)$ hvor $\delta << 1$. The covariance matrix is just copied $\Sigma_{M+1} = \Sigma_s$.

The above clustering algorithm is well suited for generating references for a template classifier based on a deterministic approach, i.e. without statistical terms like densities aso. However we can "easily" generalize the algorithm to work in a statistical framework, more precisely to estimate the parameters in a GMM-based Plug-in-MAP classifier. We then need to introduce the so called "soft decision" by using the Posteriori probabilities $P(C_i/x_k) = \gamma_{ik}$. We will not evaluate how to find these values, but obviously they tells us how well the input x_k matches a given cluster $C_i \; i = 1, \ldots, M$ on a scale from zero to one. Instead of classifying x_k to belong to a specific cluster (see point 4 in the clustering algorithm above), we calculate the probability of the same. Thus any input will have probability larger than zero for belonging to any cluster. This we can use to weigh the contribution of x_k to all clusters; i.e. the equations (16) and (17) are modified to

$$\hat{\mu}_i = \frac{\sum_{k=1}^{N} \gamma_{ik} x_k}{\sum_{k=1}^{N} \gamma_{ik}} \tag{24}$$

$$\hat{\Sigma}_i = \frac{\sum_{k=1}^{N} \gamma_{ik} (x_k - \mu_i)(x_k - \mu_i)^T}{\sum_{k=1}^{N} \gamma_{ik}} \tag{25}$$

Note that the sums now will be over all $N = \sum_i N_i$ training vectors x_k. Also note that equations (24) and (25) become identical to (16) and (17) if

$$\gamma_{ik} = \begin{cases} 1 & x_k \in C_i \\ 0 & x_k \in C_j \neq C_i \end{cases} \tag{26}$$

The equations (16) and (17) are therefore often called "hard" decision" (or sometimes "winner-takes-all"), i.e. the vector x_k only belongs to the closest cluster

The clustering algorithm described above was based on distances. However, the equations (16), (17), (24) and (25) are based on statistics, i.e. maximization of a likelihood (ML). One can show that an ML-based approach for deriving optimal GMMs for a plug-in-MAP classifier leads to exactly the equations (24) and (25). In this case a reference is replaced by a Gaussian. This clustering variant was developed from a statistical viewpoint and therefore is named the **Expectation-Maximation (EM)** algorithm".

Finally, one should mention the existance of clustering variants which are correlated to the total classification error rate. This type of clustering works simultaneously on the total training set (all classes) and tries to find class references which minimize the error rate on the training set. The most known of these methods are named "The learning vector quantizer - LVQ".

3.4 Examplifying using two data sets.

To illustrate clustering and the different classifier strategies given in this chapter we will use two kinds of data examples.
The first is a case were we generate the data as **Gaussian sources/classes** by using the command "randn" in Matlab. We generate 100 2-dimensional samples of each of four Gaussians symmetrically centered around the origo, i.e. with means respectively $[-1\ -1], [+1\ -1], [-1\ +1], [+1\ +1]$. In the clustering example we use all four sources, while we in the classifier cases use only two of the sources.
In the second example we use **real data** from the "classical" Iris flower case. There are three Iris classes (Setosa, Versicolour and Virginica). All three classes have two different flowers (sepal and petal) which differ in length and width, i.e. the features are 4-dimensional vectors. We have 50 samples of each class. For plotting purpose we only use 2 of the 4 dimensions. The two data sets are plotted in figure 9

3.4.1 Clustering the two example data sets

We start by regarding a data set as unlabeled, i.e. we have no information on which class each sample belongs to. Thus our goal is just to find structures in the data which will lead to a natural division into a number (which also must be found) of clusters. Note that in all clustering cases we us the standard Euclidian distance, i.e. equation (9).

For the Gaussian data set the figure 10 shows the cluster centres when increasing the number of clusters from one to five. In this case it is visually easy to see that four clusters match the source data well. In the general case it is not possible to do such a visual inspection, either due to the input dimension or more overlapping data. However, by monitoring the accumulated/total distance, it is possible to decide upon a reasonable number of clusters. The lower right plot in figure 10 has several horisontal parts. They correspond to when the distance have stopped decreasing, i.e. a stable situation for a given number of clusters. Thus the distance between two consequtive horisontal parts gives the improvement in distance by adding an extra cluster. We see that this improvement stops when we go from four to five clusters.

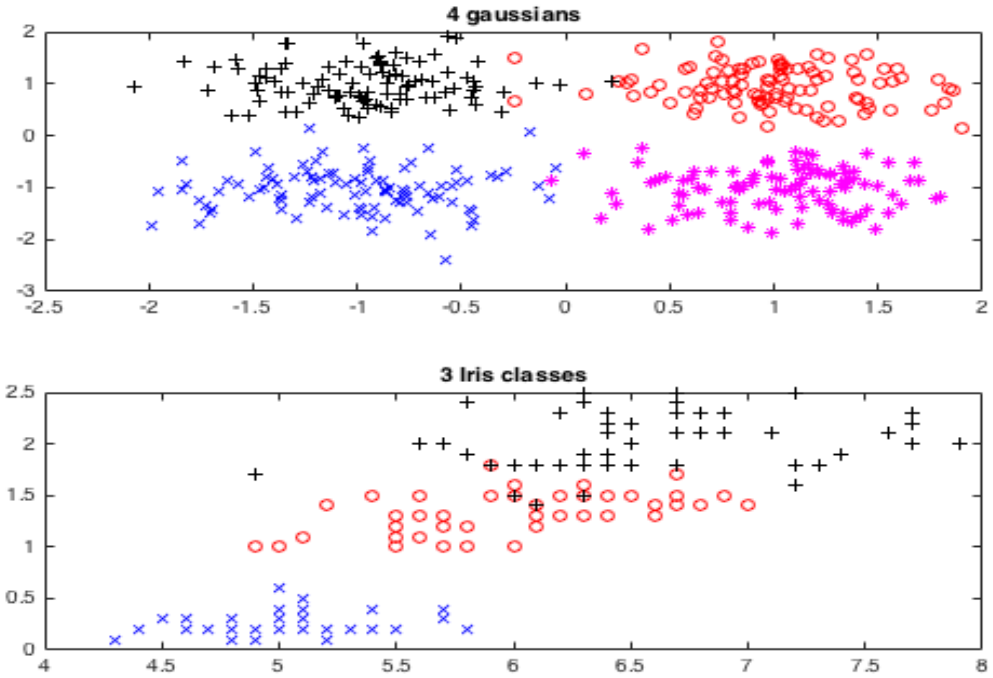

Figure 9: *The two data sets used as examples*

In the Gaussian case (figures 9 and 10) the clusters are well separated. The degree of overlap (standard deviation) will of course influence the clustering. In figure 11 four different standard deviation are used for the Gaussian sources . The solid lines (Voronoi lines) show the resulting **cluster borders** and a cluster is called a Voronoi cell. We see that the cluster borders deviate from horizontal/vertical lines when the overlap increases. We also see that the cluster centres deviate from the "true" source centers as a function of the overlap.

Figure 12 shows the same clustering process, but now for the Iris data. This time we see that three clusters are a reasonable choice. This corresponds to to the number of Iris classes, thus indicating that the clustering follows the class labeling to a great extent. Note that we now can not vary the standard deviation as we work with real data!

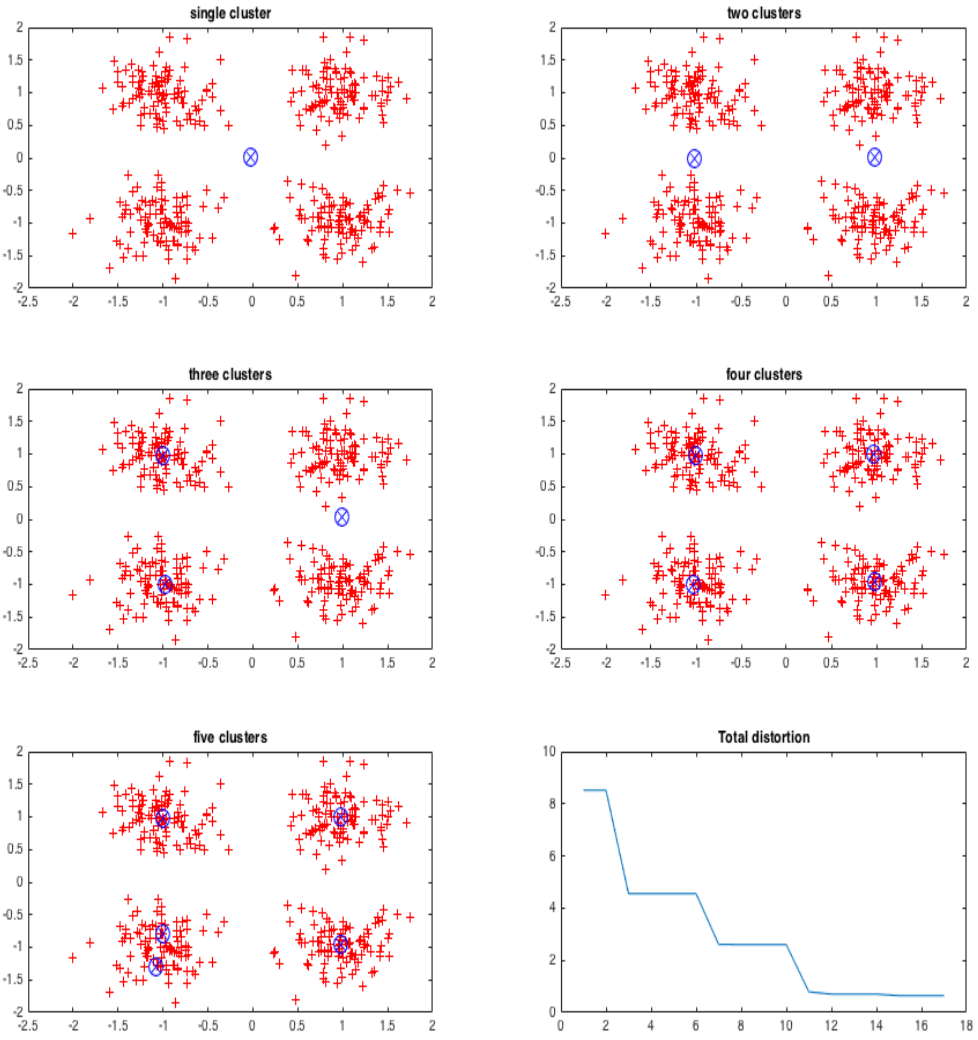

Figure 10: *Clustering the Gaussian data*

3.4.2 Using single Gaussian class models for classifying the data sets

We now use data from two of the four Gaussian sources and try to classify the data by choosing a single Gaussian model for each class (as described in subchapter 3.1.1). Note that this choice gives a match between the data sources and the class models as both are single Gaussians. To simplify we use all data for training. In figure 13 the results are plotted for different standard deviations (variances) for the sources. The ellipses give the contours for equal likelihood while the black curve shows the decision border. Note that when the standard deviation is equal we have a linear separable problem. Thus we ideally could use a linear classifier, however even the Gaussian classifier gives a decision border which approximately is a straight line. However, the border gets more and more curved as the difference in variance increases and the problem becomes nonseparable. Further, when we in addition have that the variance is comparable to the distance between the means, the decision border is an ellipse illustrating that the class with smallest variance is surrounded by the

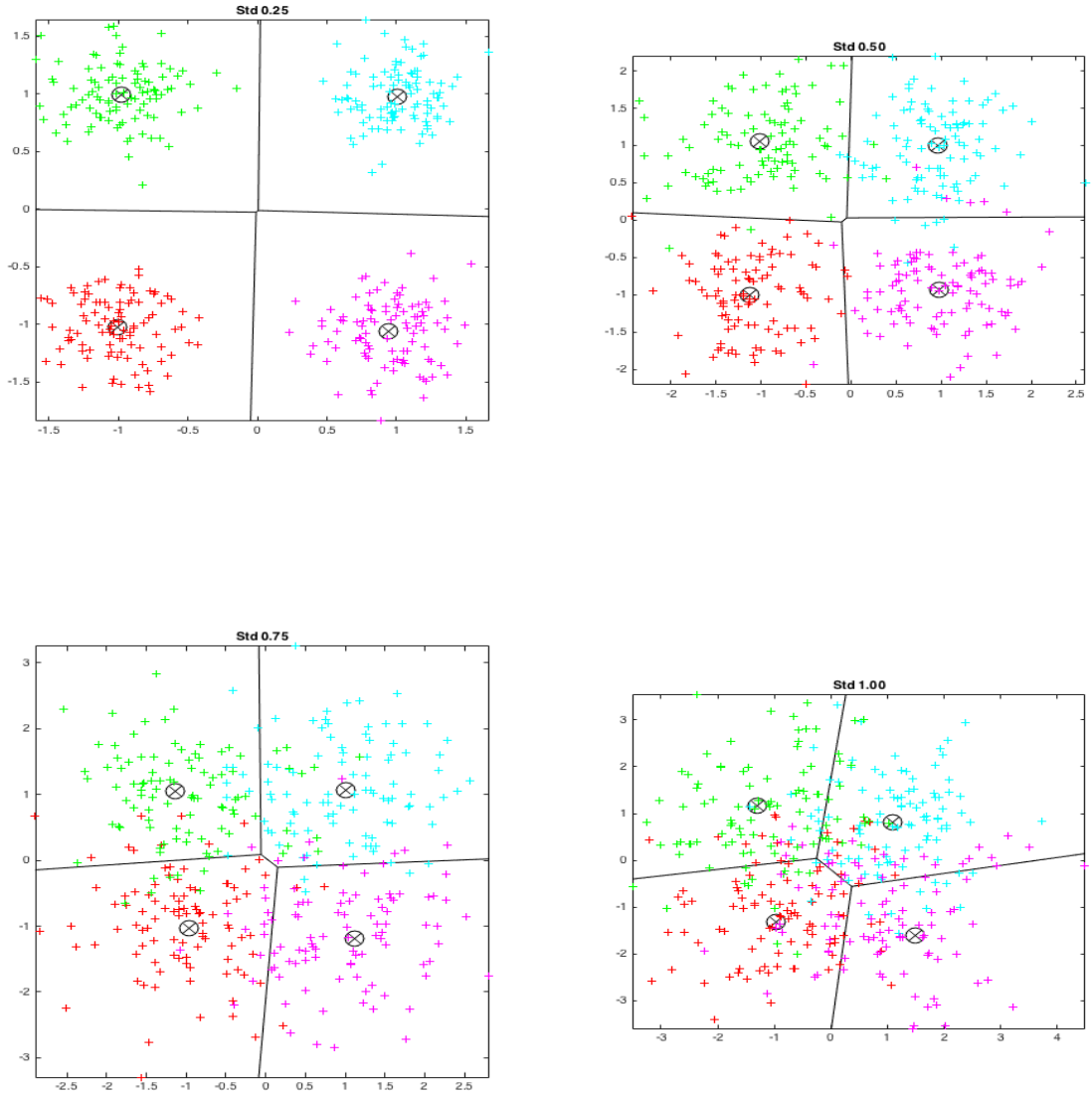

Figure 11: *Clustering as a function of degree of data overlap*

wider class. Else we see, as expected, that the two class models fit the data well. Further, the class with largest variance is responsible for more or less all the errors.

We then use the same single Gaussion models for two "lowest" of the three Iris classes using only the two input dimensions as given by figure 9. Note that we of course have no control of the variance in this case, as the data are real measurements. The model contours and the decision border is given by figure 14. We can easily observe from the data samples that this is a linear separable case. However, figure 9 shows that this is not the case if we include the third class.

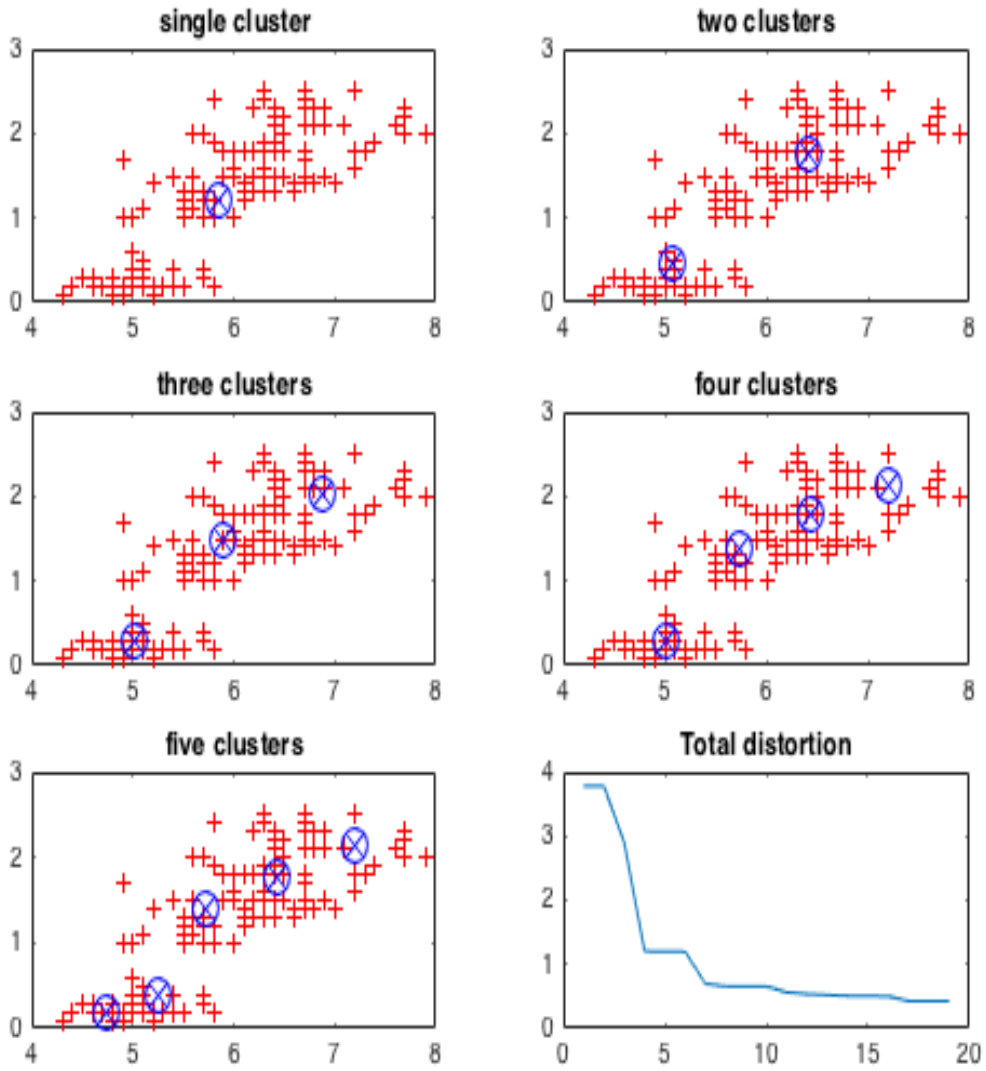

Figure 12: *Clustering the Iris data*

3.4.3 The linear classifier (perceptron)

The linear classifier is described in subchapter 3.4.4. In order to visualize the decison borders we choose the Iris data with only two features as shown in figure 9, thus the border will be a line. Further we choose only two classes. We further divide the data set into a training part of 30 samples and a test set of 20 samples for each of the two classes. The border line is shown in figure 15 both

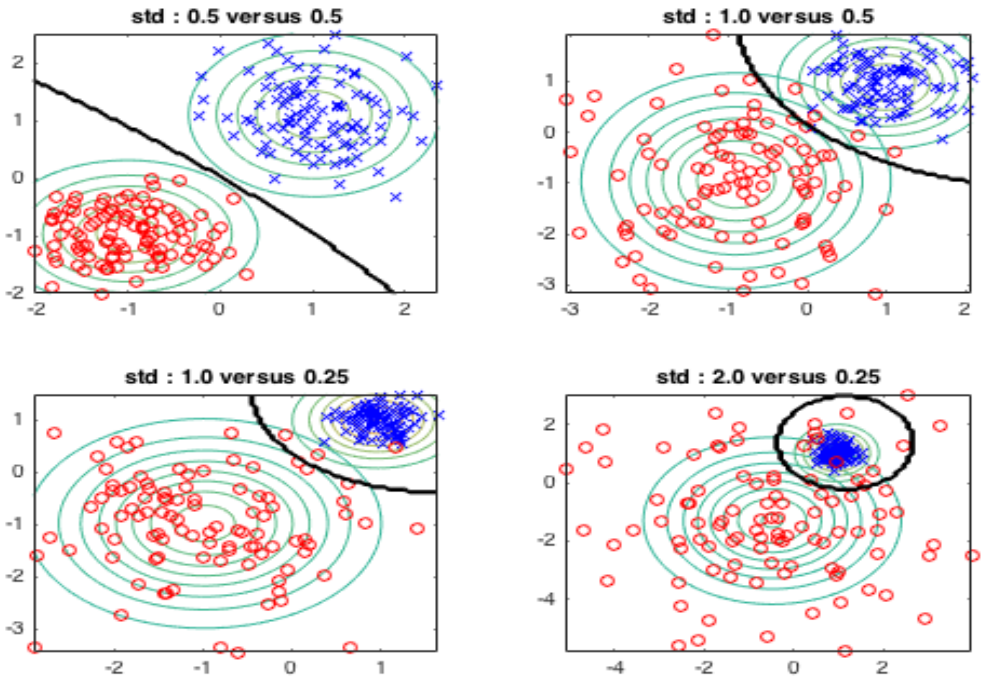

Figure 13: *Single Gaussian sources and class models showing likelihood contours and decision border*

for the training set) and the test set. In figure 16 we show the MSEs and the error rates for the training and test set as a function of the number of iterations (gradient updates).

3.4.4 The reference based classifier

In subchapter 3.3 we discussed the two methods for generating references for a template based classifier. The first method was simple; namely using all training samples as references. This method is discussed and illustrated in subchapter 3.4.1 and figures 10, 11 and 12. The second method uses clustering to find a small set of good references. We will focus on this last method in the following. Note the following differences between standard clustering and reference generation. In standard clustering we do not have any class labels for the training samples. The goal is to split the training set into "natural" clusters. However, in reference based clustering the training samples are labeled, and the goal is to find some representative references for each class. Thus we split the training samples into class subsets and perform standard clustering for **each class** separately. The figures 17 and 18 show the the case where a **single** reference is used for each class. Here, the references in both cases are found by the sample mean, i.e. equation (16). Note that we in this single reference case onlyuse the two first steps in the clustering algorithm. The full algorithm is of course used when we want more than one reference per class.

3.5 Evaluation of classifiers

The Gaussian source example was described initially in subchapter 3.4. This is an artificially generated problem, thus we know exactly the parameters of the sources and can thus calculate **the true/theoretical minimum error rate (MER)**. Such artifically data are very useful for evaluating classifiers as we know the MER as un upper bound wrt performance. However, we will of course

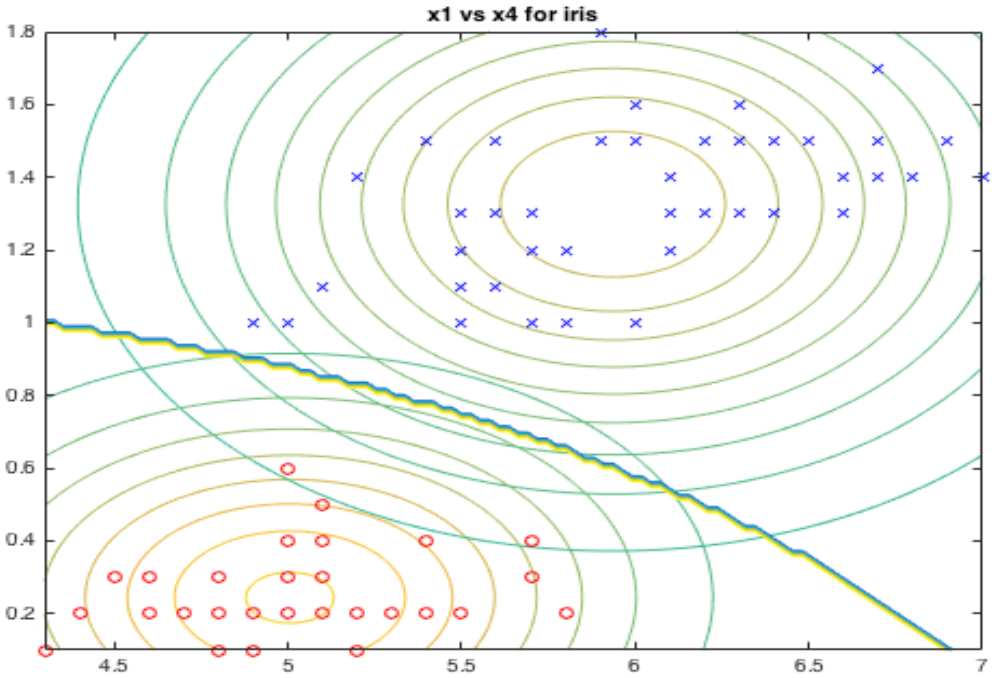

Figure 14: *The Iris case for two classes using single Gaussian class models showing likelihood contours and decision border*

also wish our classifiers to have an error rate close to the MER for practical problems (real data). Thus we must be able to estimate the true error rate for our chosen classifier. This subchapter deals with how to do this.

For practical cases we have an additional problem. We do not know the true class distribution of the sources/features, thus we can not calculate the MER. However, we still need to calculate the error rate for our chosen classifier for any practical problem! In both cases we will need a large set of N_D samples for training/development and N_T samples for testing. Further we need to have an estimator (formula) for the error rate. A logical choice is just to count all the errors E in the data set and divide by the data set size N! We will call this the **estimated error rate** in the following.Thus the estimated error rates for respectively the training and test sets are given by $EER_D = E_D/N_D$ and $EER_T = E_T/N_T$. Note that it is EER_T which is the estimated error rate for the classifier.

We will describe the uncertainty/variation in classifier design and performance by using the example of classifier using single Gaussian class models (as described in subchapter 3.1). First assume that we have **two different training sets** of the same size N_D. It is clear that the sample means and covariances (as given by equations (16) and (17)) will have different values for the two resulting classifiers. Thus the error rate on the (same) test set will therefore differ for the two classifiers. Now let us assume a single training set but **two different test sets** of equal size N_T. In this case we have a single classifier but we will of course get two different estimated error rates as we have two test sets. Thus we see that the bf true (but unknown) error rate depends on the training set (and of course the classifier structure), while the resulting **estimated** error rate depends on the test set.

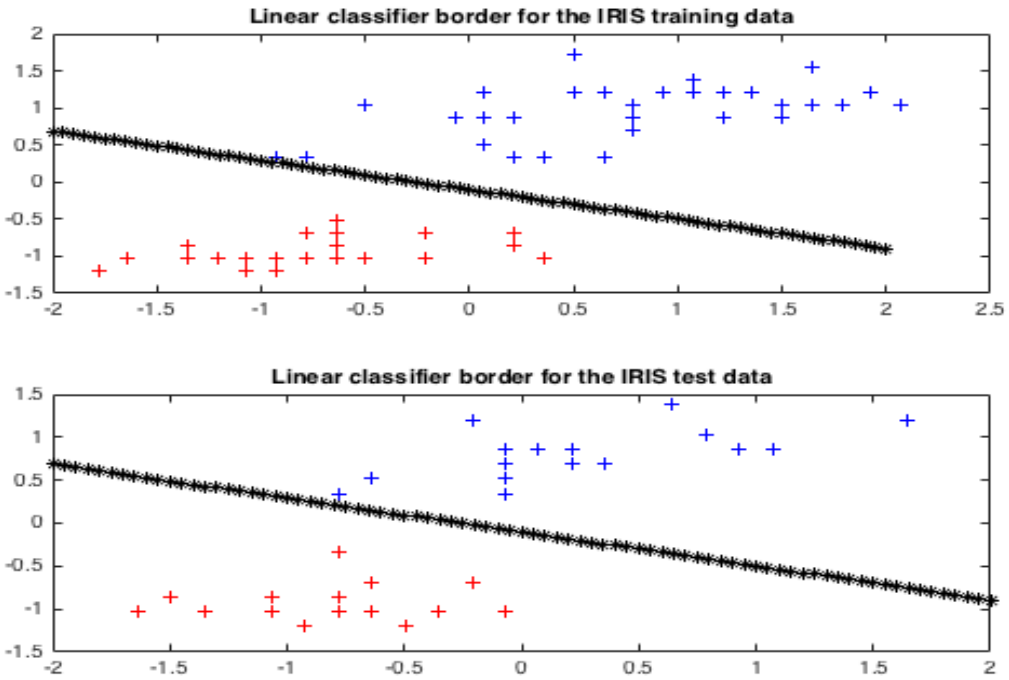

Figure 15: *Linear classifier border for the IRIS case*

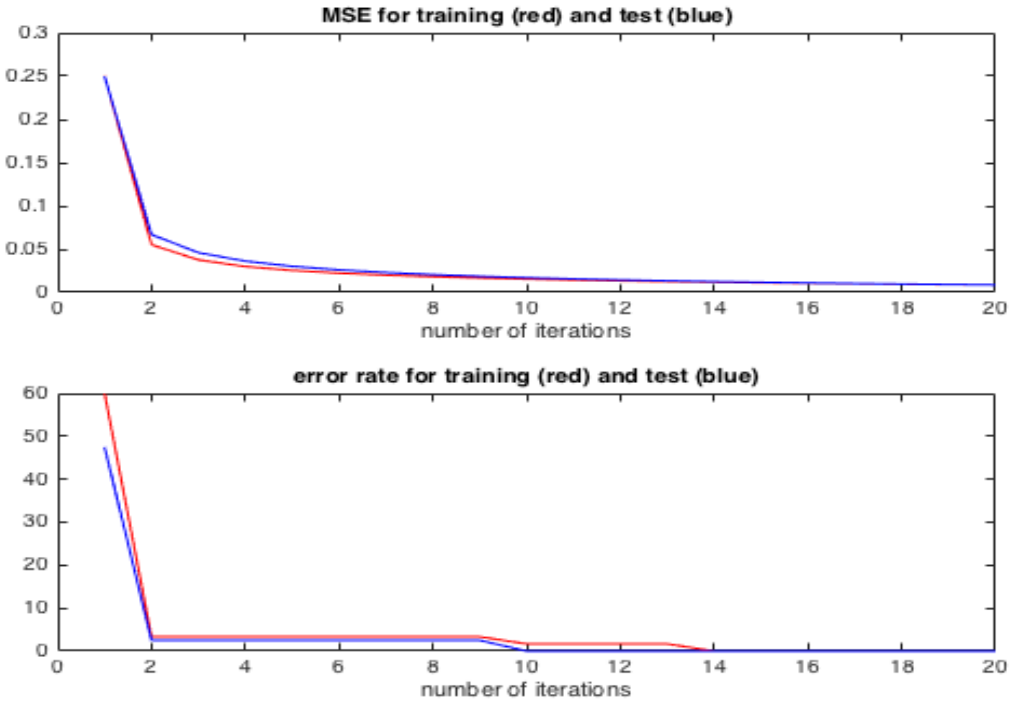

Figure 16: *MSE and error rate curves for the linear classifier*

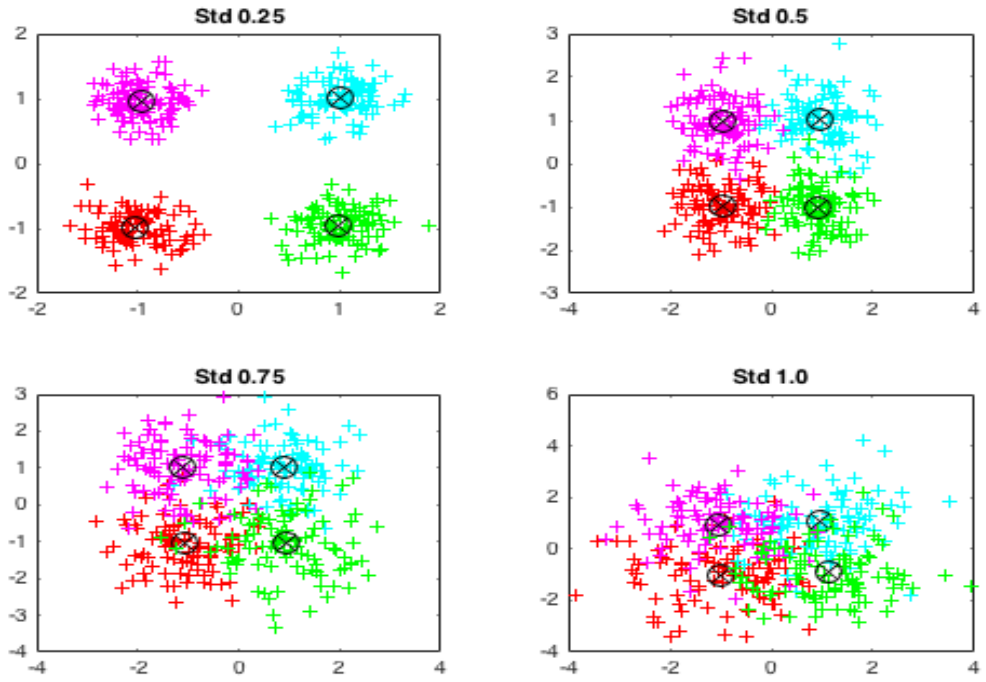

Figure 17: *Single references for the four classes of Gaussian data*

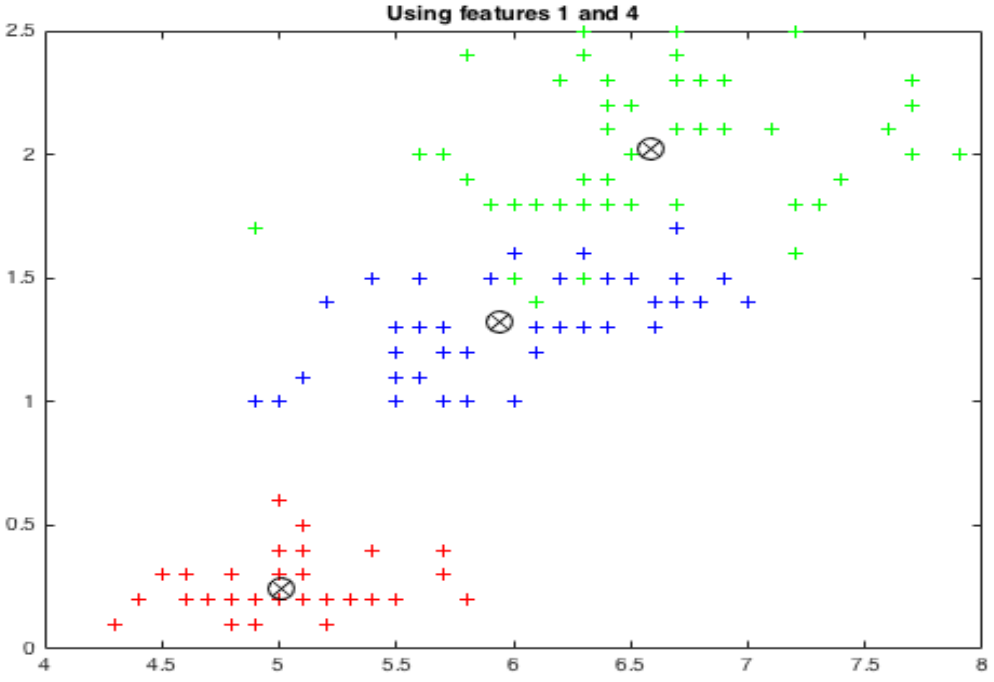

Figure 18: *Single references for the three classes of Iris data*

Then let us analyze the impact of the data set **sizes**. We will use some extreme examples to illustrate this. Let us start by the training set. We assume that we have only a single sample. Thus the estimated mean will be identical to this value, but the estimated variance is impossible to calculate! If we increase the size to two samples we will find estimated values for all parameters. However, choosing two different samples as training set will probably lead to quite different means and variances. From this we can infer that we should wish as many training samples as possible, i.e. ideally should $N_D \to \infty$. In most cases we will experience an error rate reduction as the training set increases in size. However, we can **not guarantee** that the estimated test error rate decrease consistently with the training set size! This is especially a problem when we choose the "wrong type" of classifier, i.e. in our example the sources are far from Gaussian distributed. Then let us analyse the test set size. Again we start by only one sample. Thus we have only two possible estimated error rates, namely 100% or 0%. If we increase to two samples we additionally have the possibility of 50% error rate. We further easily understand that using two other test samples can lead to a different error rate, ie. a difference in minimum 50%. Again we see that this difference decreases as the test set size increases. Of course the ultimate situation is where $N_T \to \infty$! Thus we should use a biggest possible test set.

3.5.1 How well do the classifier generalize?

In a practical world the classifier is first trained and tested (by labeled data) and then is set to work on real, unlabeled data. The goal is of course to design a classifier which will give the same error rate on the real data as the test data. However as the real data (of course) is unlabeled we can not calculate/estimate the corresponding error rate! We therefore choose the following strategy : if the difference in performance between the test set and the training set, $|EER_T - EER_D|$ is "small enough" we can assume the same for the difference between the test set and the real data. If this former difference is small we say that the classifier "generalizes" well. This strategy can also be used to choose between different types of classifiers. A classical mistake wrt generalization is to choose a complex classifier in combination with a small training set. Thus the classifier will learn "too well" the exact training set values (i.e. often error free $EER_D \approx 0$). As a consequence the error rate on the test set will become significantly larger than for the training set. This estimated error rate difference is therefore a good indication of how well the classifier generalizes. An absolute difference on $2-4\%$ indicates a good generalization while more than 10% normally is not acceptable. Note, however that normally we have $EER_D < EER_T$ as the classifier has "learnt" the training data during training/design.

Finally, note that a professional design also requires a third data set called a validation set. The corresponding error rate is used during training as an indication when to stop training. Thus the EER_T is not used during training!

3.5.2 True versus estimated error rate

In the above we use the term "estimated" error rate for a specific test set on a given, trained classifier. We also discussed the difference between this and the true, but unknown, error rate for the classifier. It turns out that it is possible to calculate a "$\pm$ interval" centered on the estimated error rate which with "high" probabilty will include the true error rate. A standard way of reporting the performance is to use a socalled 95% interval, i.e. claiming that with 95% certainty the true error rate will be in this interval. The interval is a function of two parameters, i.e. the value of the estimated error rate and the size of the test set. Figure 19 shows intervals for test set sizes ranging from $N_T = 10$ to $N_T = 1000$ for the (unknown) true error rate p as a function of the estimated error rate $\hat{p}$ (i.e. EER_T). The interval corresponds to the uncertainty/variance of the estimate, and thus shrinks as a function of increasing test set size N_T.

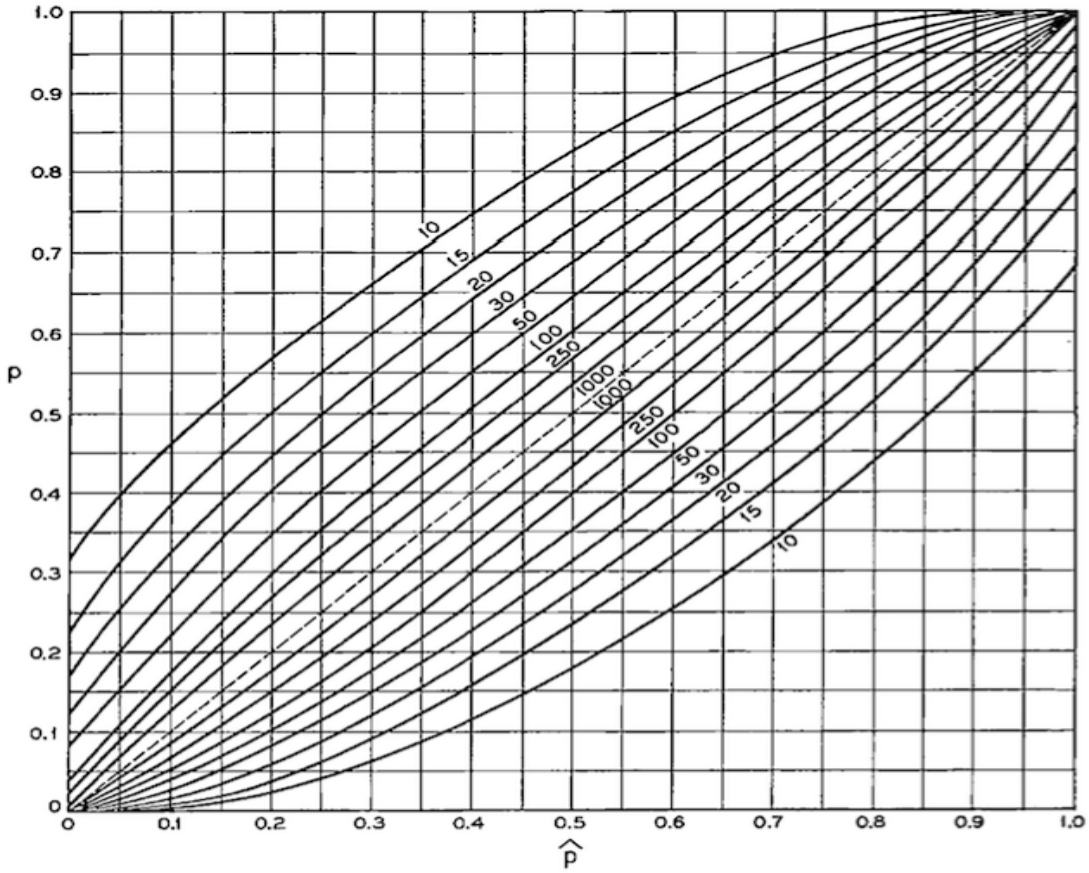

Figure 19: 95% *confidence intervals for the error rate as a function of test set size*

3.5.3 The leave-one-out strategy

In the introduction to this subchapter we discussed the sizes of the training and test sets. We found the simple rule; the bigger the better. However, quite often one has only access to a limited amount of labeled data. Further this data has to be split up into a training and a test part (and ideally also a validation set ...). The Iris case is a typical example of this problem, as it consists of only 50 samples per class. Using most (or nearly all) for training will give the best classifier but the estimated error rate on the test set will have a large uncertainty as seen in figure 19. The "leave-one-out" strategy is a suboptimal technique to circumvent this problem. For the Iris case this method suggests using 49 samples for training and only one sample for test per class. We however has to do this 50 times, i.e. systematically changing the test sample for each time. The drawback is that the classifier will be slightly different for each time. The estimated error rate is then the mean of the error rate for each of the 50 experiments. This method will give a larger confidence interval than if we had 50 test samples to apply to the *same* classifier. However, the estimated error rate is usually closer to the true error rate than if we for instance used 30 samples for training and 20 for test. Note that one has a huge amount of alternatives for this strategy, i.e one could chose to use all possible combinations of $50 - x$ training and x test samples, where $x = 2, 3, 4, \ldots$.

3.5.4 The concept of significance

Above we discussed the impact of **finite** training and test sets. We argued for that this led to uncertainty both in generalization and performance estimation. Here we will introduce another concept, namely **significance** which is closely related to the error rate confidence interval. Now assume we have two different classifiers which we want to rank using the same test set. We find the corresponding values $EER1_T$ and $EER2_T$. Assume $EER1_T < EER2_T$, should this indicate that the classifier1 is the best? The answer to this is yes only if the difference is significant. What if we had another test set which gave us the opposite result? The probability of this happening is of course highest when the differences are small, i.e. socalled insignificant. Thus we can use Figure 19 in another way. If the difference in the estimated error rate for the same test set is larger than the confidence interval this indicates with 95% certainty that classifier1 is better than classifier2.

3.5.5 The confusion matrix

In the start of this subchapter on evaluation we defined the estimated error rate which is the "total" error rate. Many times we would like to monitor both the performance for the individual classes and the class confusibility, i.e. which classes is "closest" in the input room. To do this it is common to use a socalled confusion matrix. In table 1 a confusion matrix is given for the case of four Gaussian classes where class 1 and class 2 is somewhat closer than the other class distances. The test set consists of 50 samples for each of the four classes. Note that the matrix shows both the number of samples which are correctly classified and the errors. The total error rate is $ERR_T = 14/200 = 7\%$ while the error rate for class 2 is $6/50 = 12\%$ Further we see that class 1 and class 2 is the most confusable, which was expected from the placement in the input room. Often the numbers in the matrix is given in percentage (especially when we have large sample numbers). The same results will then have the form as in table 2.

Classified : $\rightarrow$ True/label : $\downarrow$	Class 1	Class 2	Class 3	Class 4
Class 1	47	3	0	0
Class 2	6	44	0	0
Class 3	0	1	48	1
Class 4	1	1	1	47

Table 1: Confusion matrix for four Gaussian classes - 50 samples per class

Classified : $\rightarrow$ True/label : $\downarrow$	Class 1	Class 2	Class 3	Class 4
Class 1	94	6	0	0
Class 2	12	88	0	0
Class 3	0	2	96	2
Class 4	2	2	2	94

Table 2: Confusion matrix in percentage for four Gaussian classes - 50 samples per class