

## CHAPTER 1 - INTRODUCTION

**Signals** are varying quantities that carry **information** about a physical phenomenon or process under analysis. They can be *unidimensional* or *multidimensional* functions that represent the evolution with respect to one or more correlated or even uncorrelated variables translatable into some physical reality elements (such as time, frequency, temperature etc.)) The physical world is mapped into digital signals by sensors, which sample reality and quantize the *energy* feedback to produce discrete values. These values are influenced by the build quality of the sensor, which determines the overall quality, the disturbance, the cleanness of the signal, the quantization error, the sampling period (the ideal frequency is defined by the Nyquist-Shannon theorem, which tells that the minimum sampling rate is double the maximum frequency of the signal), etc.

**Machine learning** is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.

A machine can be trained in various methods, such as **supervised learning** (the learner is fed with a set of input/output pairs (training set)), **unsupervised learning** (no output information is provided, but just input data which are modeled by the learner (e.g., data clustering), **semi-supervised learning** (besides a limited amount of input/output pairs, the plenty unlabeled data are exploited during the learning process), **active learning** (it's a branch of supervised learning where the learner interactively queries a supervisor (oracle) to label new samples with the desired outputs) and **reinforcement learning** (learning process of an arbitrary being (agent) in the world surrounding it (environment). The agent seeks to maximize the rewards it receives from the environment, and performs different actions in order to learn how the environment responds and gain more rewards).

The first possible supervised task is represented by **Classification**. **Binary classification**, where the model is trained to tell whether the input is or not an element of a class. When the model is required to classify an object among a set of classes, it's called **multiclass classification**. **Multilabel classification** is when given an input the model returns a vector of multiple outputs, classifying various classes inside the same model.

The ultimate frontier of classification is **Captioning**, where the model is asked to describe the input data with words.

Another set of supervised tasks, where the *output is not discrete, but continuous*, is **Regression**. **Ranking** refers to the process of ordering items or entities based on their relative importance or relevance, often in a search or recommendation system.

A **Machine Learning System** starts with the **acquisition** of the data, which is done mainly by sensors, which provide raw signals sensed from the real world.

The signals now need to be **pre-processed** to remove noise and improve (manipulation, low level segmentation (grouping similar pixels that are close together into super-pixels), semantic segmentation (grouping together areas of the image that belong to the same object)) them.

Next step is **feature extraction** based on features, either pure from the pre-processed data, or additionally elaborated, even brought to other feature spaces. Features are generally reduced with a selection process, to remove useless ones.

Feature extraction is followed by **classification**, which we've already seen.

**Post-processing** helps the decision maker by filtering the results of the classifier, which may produce recoverable errors, due to signal noise or other events. This step is dependent on the specific task and has to be implemented with ad-hoc rules.

Finally, a **decision** is taken based on the perfected information.

The *Design Phase of a Machine Learning System* is composed of **data collection** (create the training set with quality data properly balanced), useful **features selection** (depending on the task of interest, only the most powerful one should be considered in order to discriminate easily and fast), **model choice** (depending on the software/hardware and on the task), **classifier training** (with techniques depending on the specific task) and **evaluation** (check for overfitting and assess accuracy, prediction time, etc.)

Other applications of Machine Learning algorithms are the **Query by image** and **Query by text**, **Text to image synthesis**, **Painting to image Synthesis**, **Tracking** and many others.

## CHAPTER 2 - STATISTICAL ESTIMATION THEORY

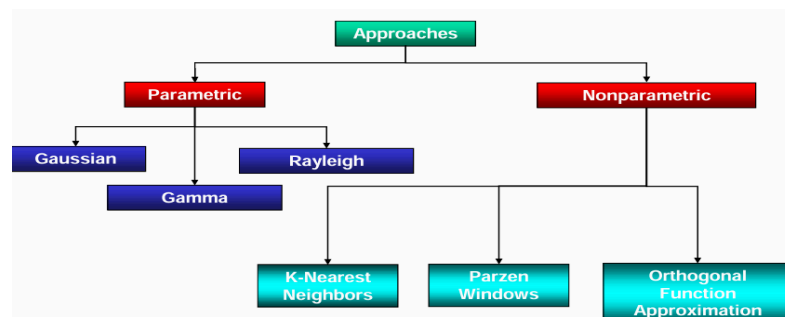
Given a source of information, the signal that the *sensor* senses has to be mapped into some discrete values; to do this the **Pulse Amplitude Modulation** elaborates the input signal into a well defined signal that can be transported into a *channel*. At the end of the channel a *sampler* can just sample the signal and pass it to a *classifier* that re-transforms the signal into the original values. In most cases the channel cannot transmit the signal clearly, thus some noise will interfere with it. *The probability density of one specific output class  $y$  is thus given by the convolution of the probability distribution of the input  $x$  with the probability distribution of the noise:  $P_y = P_x * P_n$ , where the result is thus a Normal distribution centered in the value corresponding to the class  $y$ .* Since the peak of the distribution could be shifted, the training phase of a model is done to correctly set the threshold that separates the output classes (a common approach is to set it in the intersection point between the distributions of the classes). Estimating the statistical distribution of a stochastic signal from a set of available observations is necessary to develop a valid machine learning model, as it's used in pre-processing, feature extraction and also in recognition.

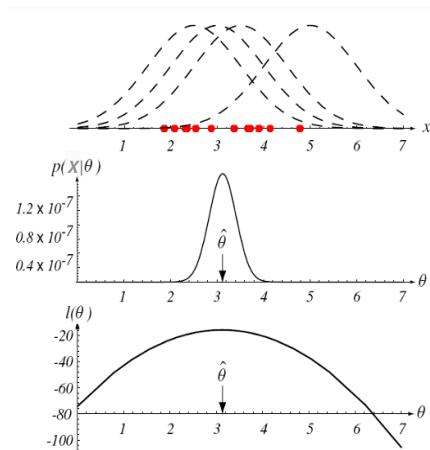
In estimation theory, stochastic signal can be subdivided into three categories:

- **Noisy deterministic signals:** the information source is completely known. The noise interference intervenes during the transmission and/or the acquisition phases (e.g., transmission of signals in telecommunication systems based on PAM)
- **Noisy parametric signals:** the information source is only partially known. The observations allow estimating the random parameters controlling the behavior of the signal (e.g., target speed velocity estimation in sonar systems)
- **Noisy random signals:** the signal is completely unknown. In this case, the estimation should rely completely on the available observations (e.g., buried object detection with ground penetrating radar)

Probability Density Function estimation can be done via either parametric methods or non-parametric methods  
→

The density of a point can be computed as the mass (number of samples) divided by the volume (in the features space) as  $D = \frac{m}{V}$ .





Let's assume that the model of  $p(x)$  is characterized by  $r$  parameters which define a vector  $\theta = (\theta_1, \theta_2, \dots, \theta_r)$ , the dependency of the model on  $\theta$  is underlined by the notation  $p(x|\theta)$ .

Given a set of samples  $X$  (vector of  $N$ ), we can define a related likelihood function as  $p(X|\theta) = \prod_{k=1}^N p(x_k|\theta)$ , which defines the likelihood of  $\theta$  with respect to the considered set of samples  $X$ ; giving a measure of compatibility/agreement between  $\theta$  and  $X$ . It is thus used to estimate, given the set of samples, the most probable density function that can have those examples as an internal set.

The main procedures of estimation are:

- **Maximum Likelihood Estimation:**  $\theta$  is viewed as a vector of quantities whose values are fixed but unknown. The best estimate of their value is defined to be the one that maximizes the probability of obtaining the samples actually observed
- **Bayesian Estimation:** the parameters are viewed as random variables having some known prior distribution. Observation of the samples converts this into a posterior density

The estimation almost always contains some degrees of error, that are due to either approximation error/ **bias** and/or estimation error/ **variance**. Bias is related to the accuracy of the parameters, so the difference between the estimated  $\hat{\theta}$  and the actual value  $\theta$ , while variance is the extent to which estimates vary across different subsets of the same dataset, so the consistency of the predicted parameters.

In formulas:  $bias = E_B = |\hat{\theta} - \theta|$  and  $var = E_V = \{(\hat{\theta} - \bar{\theta})^2\}$  for each  $i=1,2,\dots,r$

- An estimate is **asymptotically unbiased** if  $\lim_{N \rightarrow \infty} E\{\epsilon\} = 0 \Rightarrow \lim_{N \rightarrow \infty} E\{\hat{\theta}\} = \theta$
- An estimate is **asymptotically efficient** if it achieves the lowest possible variance among all unbiased estimators as the sample size goes to infinity. This means the estimator attains the **Cramér-Rao Lower Bound (CRLB)** asymptotically, which represents the minimum variance for any unbiased estimator of a parameter.  

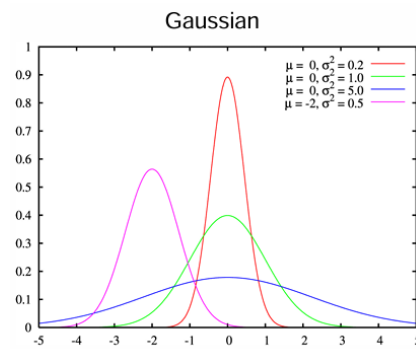
$$\lim_{N \rightarrow \infty} \frac{var\{\epsilon\}}{[I^{-1}(\theta)]_{ii}} = 1, i = 1, 2, \dots, r$$
where  $I$  is the **Fisher Information** (quantifies the amount of information that an observable random variable  $X$  carries about an unknown parameter  $\theta$  of its probability distribution. It measures how sensitive the likelihood function is to changes in the parameter  $\theta$ , meaning how much data can "tell us" about  $\theta$ )
- An estimate is **consistent** if  $\lim_{N \rightarrow \infty} P\{||\epsilon|| < \delta\} = 1 \quad \forall \delta > 0$  so the basic condition for consistency is that *the estimate is asymptotically unbiased and with variance converging to zero when  $N$  goes to infinity*. It means that as the sample size goes to infinity, the estimator converges in probability to the true value of the parameter being estimated.

The **maximum likelihood estimate** of  $\theta$  is  $\hat{\theta} = \arg \max_{\theta} p(X|\theta)$ ; by varying  $\theta$ , the pdfs will change too. *The Maximum Likelihood estimate corresponds to the value of  $\theta$  that best fits the training samples.* For analytical purposes it's usually easier to work with the logarithm of the ML:  $\hat{\theta} = \arg \max_{\theta} \ln(p(X|\theta))$  where  $p(X|\theta) = p(X|\mu) = \frac{1}{\sqrt{2\pi\delta^2}} \exp(-\frac{(X-\mu)^2}{2\delta^2})$  with  $\delta^2 = 1$ .

*The maximum likelihood estimates the mean of a mono-dimensional Gaussian Probability Density Function with known variance based on a single observation.*

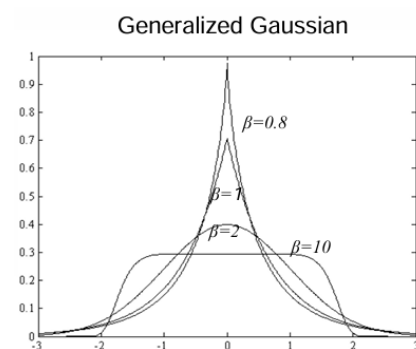
The ML is asymptotically unbiased, efficient and consistent.

The most popular Statistical Models for  $p(x)$  are:



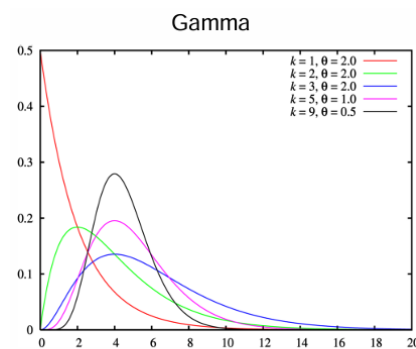
$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Also known as the Normal distribution, it's characterized by its symmetric, bell-shaped curve that's perfectly balanced around its mean. This model is particularly useful for describing phenomena that cluster around a central value, with deviations becoming less likely the further they are from the mean. The Gaussian model is defined by two parameters: the mean ( $\mu$ ), which represents the center of the distribution, and the standard deviation ( $\sigma$ ), which measures the spread of the data. Its widespread use is partly due to the Central Limit Theorem, which states that the sum of a large number of independent random variables tends to follow a normal distribution, regardless of their original distributions.



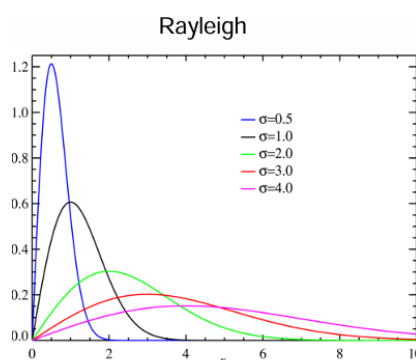
$$f(x, m, b, \beta) = \frac{b\beta}{2\Gamma(1/\beta)} e^{-[b|x-m|]^\beta}$$

The Generalized Gaussian model is an extension of the standard Gaussian model that allows for greater flexibility in modeling data with different tail behaviors. It introduces an additional shape parameter ( $\beta$ ) that controls the rate of exponential decay in the tails. When  $\beta = 2$ , it reduces to the standard Gaussian distribution; when  $\beta < 2$ , it has heavier tails, and when  $\beta > 2$ , it has lighter tails than the normal distribution. This adaptability makes it particularly useful in signal processing and image analysis, where data often deviate from strict normality.



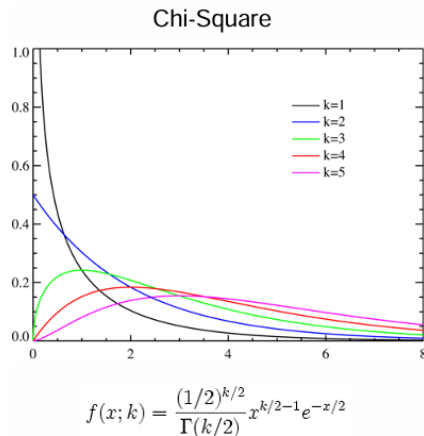
$$f(x; k, \theta) = x^{k-1} \frac{e^{-x/\theta}}{\theta^k \Gamma(k)} \text{ for } x > 0$$

The Gamma model is a highly flexible distribution that can accommodate a wide range of shapes for positive continuous data. It has an additional shape parameter that allows for even greater versatility. This model is particularly useful when dealing with data that exhibit varying degrees of skewness and kurtosis. One of its key strengths is that it includes several other important distributions as special cases, the Chi-Square ( $\theta = 2, k = \frac{\nu}{2}$ ), and exponential ( $k = 1$ ) distributions. This makes it a powerful tool in fields like reliability analysis and survival analysis, where it can adapt to different patterns of failure rates or survival times. The model's flexibility comes at the cost of increased complexity in parameter estimation, often requiring sophisticated numerical methods.

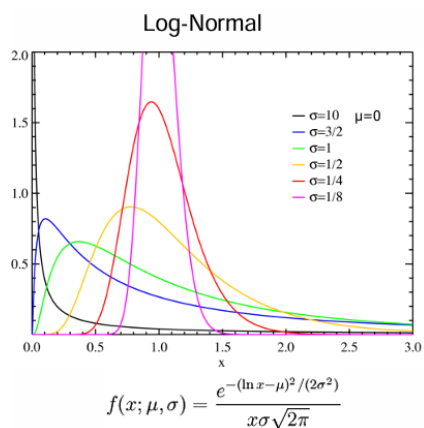


$$f(x|\sigma) = \frac{x \exp\left(-\frac{x^2}{2\sigma^2}\right)}{\sigma^2}$$

The Rayleigh model is named after Lord Rayleigh and has a specific niche in statistical modeling. It's particularly useful for describing the magnitude of a two-dimensional vector when its components are independently and identically distributed. It only has a parameter  $\sigma$  for the mode of the distribution. Interestingly, it's closely related to the Chi distribution with two degrees of freedom. Its probability density function has a distinctive shape, starting at zero, rising to a peak, and then declining with a longer right tail.



The Chi-square model represents the distribution of the sum of the squares of  $k$  independent standard normal random variables, where  $k$  is the degrees of freedom. This distribution plays a crucial role in inferential statistics, particularly in hypothesis testing and the construction of confidence intervals. As the degrees of freedom increase, the Chi-square distribution becomes more symmetric and approaches a normal distribution. It's widely used in goodness-of-fit tests, independence tests, and quality control applications.



The Log-Normal model describes a variable whose logarithm follows a normal distribution. This results in a distribution that's always positive and right-skewed, making it suitable for modeling phenomena that can't take negative values and tend to have a few very large values. It's particularly useful in finance for modeling asset prices, in biology for organism growth, and in environmental science for pollutant concentrations. The model is characterized by two parameters:  $\mu$  (the mean of the log of the variable) and  $\sigma$  (the standard deviation of the log of the variable).

The multivariate Gaussian Probability Density Function is analytically expressed as:

$$p(x|\theta) = p(x|m, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp\left[-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right] \quad \text{where } \Sigma \text{ is the } \underline{\text{covariance matrix}} \text{ and } (x - \mu) \text{ is the } \underline{\text{mean vector}}.$$

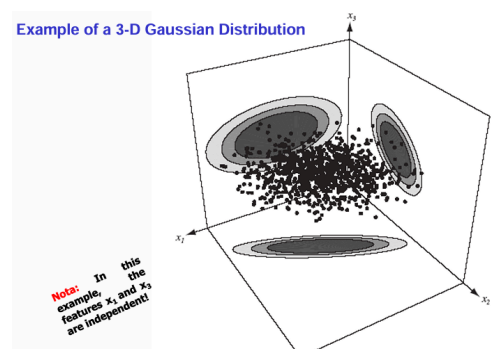
Key properties of the covariance matrix,  $\Sigma$ , include:

- $\Sigma$  is symmetric:  $\Sigma = \Sigma^T$ .
- $\Sigma$  is positive semidefinite.
- For independent features,  $\Sigma$  becomes diagonal, with each entry corresponding to the variance of a feature.

A 3-D Gaussian pdf forms a bell of unit volume. The horizontal slices of the bell, corresponding to **isolevels**, are ellipses whose axes are directed by the eigenvectors  $\Sigma$ . The **eigenvector** associated with the largest **eigenvalue** determines the major axis of the ellipse.

Expanding this for an  $n$ -D Gaussian pdf:

- Eigenvalues  $(\lambda_1, \lambda_2, \dots, \lambda_n)$  with  $(\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n)$  and eigenvectors  $(e_1, e_2, \dots, e_n)$  govern the shape of the pdf.
- **Isolevels** are hyperellipses, with the eigenvectors determining the direction of the axes, thus the orthogonal basis. The first eigenvector defines the principal axis, while the last corresponds to the smallest axis



It can be shown that the Maximum Likelihood estimation, in a multivariate Gaussian pdf, given a set of N iid training samples, of the **mean vector**  $m = E\{x\} \Rightarrow \hat{m} = \frac{1}{N} \sum_{k=1}^N x_k$  is unbiased, efficient and consistent, but the estimation of the **covariance matrix**

$$\Sigma = Cov\{x\} = E\{(x - m)(x - m)^t\} = E\{xx^t\} - mm^t \Rightarrow \hat{\Sigma} = \frac{1}{N} \sum_{k=1}^N (x_k - \hat{m})(x_k - \hat{m})^t = \frac{1}{N} \sum_{k=1}^N x_k x_k^t - \hat{m}\hat{m}^t$$

can be slightly biased, so we can remove the bias with a formula adjustment:

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{k=1}^N (x_k - \hat{m})(x_k - \hat{m})^t. \text{ We can also note that } E\{\hat{m}\} \Rightarrow m, \text{ and } E\{\hat{\Sigma}\} = \frac{N-1}{N} \Sigma.$$

While in Maximum likelihood estimation the output parameter vector  $\theta$  is considered to be fixed, in **Bayesian estimation**  $\theta$  is considered to be a set of random variables which need to be statistically described by associating to them a statistical model, allowing embedding of prior knowledge about the distribution in the estimation process. The shape of the density  $p(x|\theta)$  is assumed to be known, but the values of the parameter vector  $\theta$  are not known exactly. *Initial knowledge* about  $\theta$  is assumed to be contained in a known prior density  $p(\theta)$  and *the rest of the knowledge* about  $\theta$  is contained in a set  $X$  of N samples  $x_1, x_2, \dots, x_N$  drawn independently according to the unknown probability density  $p(x)$ . So it starts with a “**prior**” **distribution** which is then perfected through the evaluation of the samples into a final “**posterior**” **distribution**, which should present a *narrower and sharper peak* around the true value of  $\theta$ .

The final goal is to compute  $p(x|X) = \int p(x, \theta|X) d\theta \rightarrow \text{Bayes theorem} \rightarrow \int p(x|\theta, X) p(\theta|X) d\theta$

where  $p(\theta|X)$  is the posterior density (which is used as weight of the model) and, since the prior knowledge of  $X$  is useless as the distribution of  $x$  is completely known once we know the value of  $\theta$ , we can rewrite it as:  $\int p(x|\theta) p(\theta|X) d\theta$ .

Since the exact value of  $\theta$  is unknown, Bayesian estimation directs to a sort of average  $p(x|\theta)$  over all possible values of  $\theta$ , weighted with the posterior density, which is computed again by

$$\text{using Bayes theorem, so } p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} = \frac{p(X|\theta)p(\theta)}{\int p(X|\theta)p(\theta) d\theta} \text{ with } p(X|\theta) = \prod_{k=1}^N p(x_k|\theta) \text{ being}$$

the likelihood function and  $p(\theta)$  being the prior density, given from prior knowledge.

If the set of training samples is huge or if the prior density is uniform, the Bayesian estimation is equivalent to the maximum likelihood estimation, otherwise Bayesian estimation is usually more precise, as it also give a precise value for the confidence, hence telling if the model is ready to use or if it needs more samples, but computationally more complex and more difficult to interpret.

If no prior knowledge about the shape of the pdf is given and/or the parametric model does not offer a good approximation of the pdf, **nonparametric estimation** can be used.

Given a generic sample  $x^*$  and a sufficiently small predefined region of the feature space  $R$ , and assuming the true pdf  $p(x)$  is a continuous function, then

$$P_R = P\{x \in R\} = \int_R p(x) dx = p(x^*) V_R \text{ with } V_R = \text{volume of } R.$$

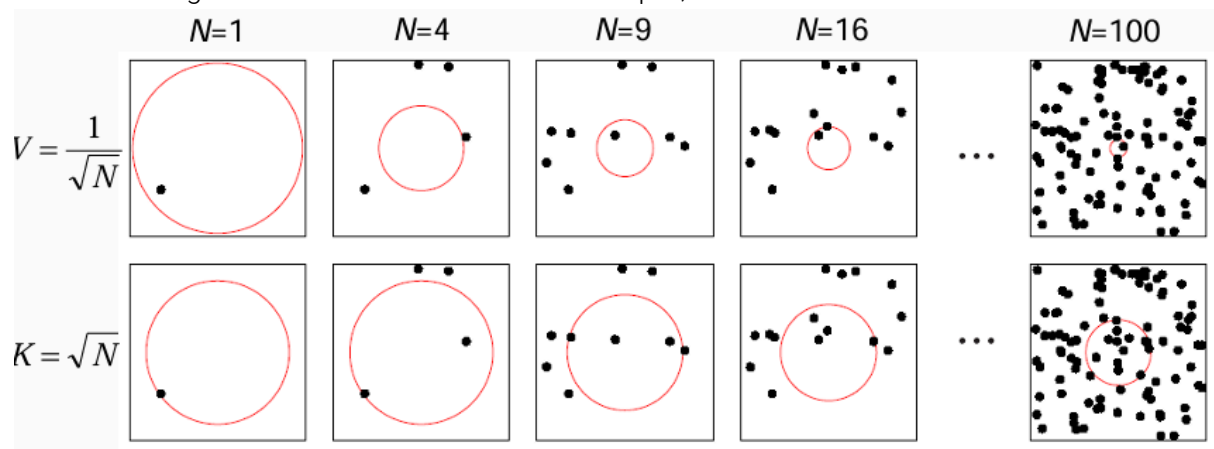
Now, let  $K$  be the number of training samples belonging to the region  $R$ , among a total of  $N$  training samples, then a consistent estimate of  $P_R$  can be achieved through the computation

of the relative frequency  $\widehat{P}_R = \frac{K}{N}$ ,  $\lim_{N \rightarrow \infty} \{|\widehat{P}_R - P_R| < \delta\} = 1 \quad \forall \delta > 0 \rightarrow$  **law of large**

**numbers**, thus  $p(x^*)V_R = \frac{K}{N} \Rightarrow p(x^*) = \frac{K}{NV_R}$  so  $R$  should be large enough to contain a sufficient number of samples, but small enough to limit the variability of  $p(x)$  within it.

Depending on the role taken by each of the two parameters  $K$  and  $V$ , one may lead to two different nonparametric estimation methods:

- **Parzen**:  $R$  (and thus  $V_R$ ) is fixed and  $K$  is calculated (from the training samples) to determine the pdf estimate.
- **K-nearest neighbor**:  $K$  is fixed and the hypervolume  $V_R$  is computed on the basis of the training set to deduce the estimate of the pdf;



Both of these methods do in fact converge, although it is difficult to make meaningful statements about their finite-sample behavior.

In **K-NN** the number  $K$  and the shape of the volume are priorly set. The volume is centered on  $x^*$  and expanded until it contains  $K$  training samples. The pdf is thus  $\widehat{p}(x^*) = \frac{K}{NV_K(x^*)}$ . It can be shown that if  $K$  is chosen as a function of  $N$  where  $p(x)$  is continuous, the following statement holds:  $\lim_{N \rightarrow \infty} K_N = +\infty$ , for example, for  $K(N) = \sqrt{N} \rightarrow \lim_{N \rightarrow \infty} \frac{K_N}{N} = 0$ .

A value of  $K$  too large will over-smooth (**underfitting**) the distribution, losing details, while a value of  $K$  too small will under-smooth (**overfitting**) the distribution, revealing details which are not relevant and may cause confusion.

In **Parzen Windows** Estimation  $R$  is initially assumed to be a  $n$ -dimensional hypercube centered on  $x^*$  with side length of  $h$  and volume  $V = h^n$ . The window function gives an analytical expression for the number  $K$  of samples contained in the hypercube:

$\gamma(x) = \{1 \text{ if } x \in V, 0 \text{ otherwise}\}$ , so  $\gamma\left(\frac{x_k - x^*}{h}\right) = 1$  if  $x_k$  falls into  $V$  and is equal to 0 otherwise.

The total number of samples in the hypercube is thus  $K = \sum_{k=1}^N \gamma\left(\frac{x_k - x^*}{h}\right)$  which leads to the

estimate:  $\widehat{p}(x^*) = \frac{K}{NV} = \frac{1}{N} \sum_{k=1}^N \frac{1}{h^n} \gamma\left(\frac{x_k - x^*}{h}\right) = \frac{1}{N} \sum_{k=1}^N \frac{1}{h^n} \gamma\left(\frac{x^* - x_k}{h}\right)$ , which is a collection of

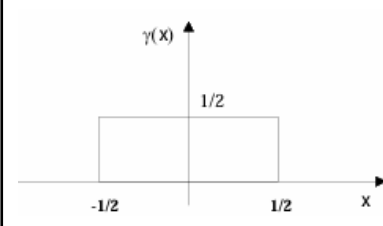
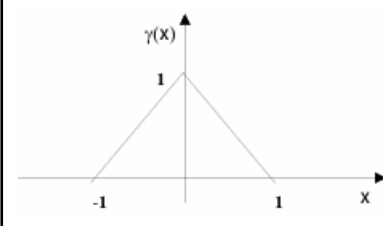
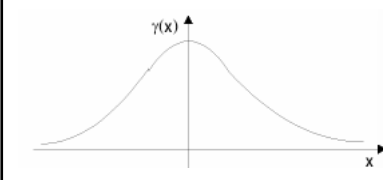
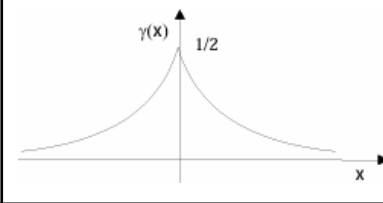
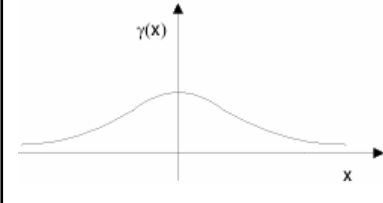
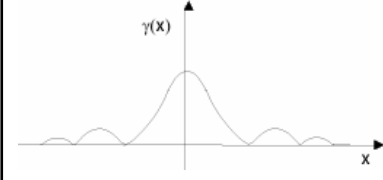
contributions / a count of the number of training samples that fall in the hypercube that resembles the density function (non-negative and that integrates to 1) of the model we are

considering. This approach can be generalized by allowing more general classes of window functions, thus changing the estimate function into an average of functions:

$$\hat{p}(x^*) = \frac{1}{N} \sum_{k=1}^N \frac{1}{v(h)} \gamma\left(\frac{x-x_k}{h}\right), \text{ where the function } \gamma(\cdot) \text{ is the **Parzen Window / kernel**. The}$$

window function, in essence, is used for **interpolation**, where each parameter contributes to the final pdf according to its distance. Other than being non-negative and integrating to 1, a good Parzen Window function should also be continuous, take a maximal value at the origin and tend to 0 as  $x$  approaches infinity.

Some commonly used kernels are:

Kernel Type	Characteristic Formula	Graphical Representation
Rectangular	$\gamma(x) = \Pi(x)$	
Triangular	$\gamma(x) = \Lambda(x)$	
Gaussian	$\gamma(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$	
Exponential	$\gamma(x) = \frac{1}{2} \exp(- x )$	
Cauchy	$\gamma(x) = \frac{1}{\pi} \frac{1}{1+x^2}$	
sinc2	$\gamma(x) = \frac{1}{2\pi} \left(\frac{\sin(x/2)}{x/2}\right)^2$	

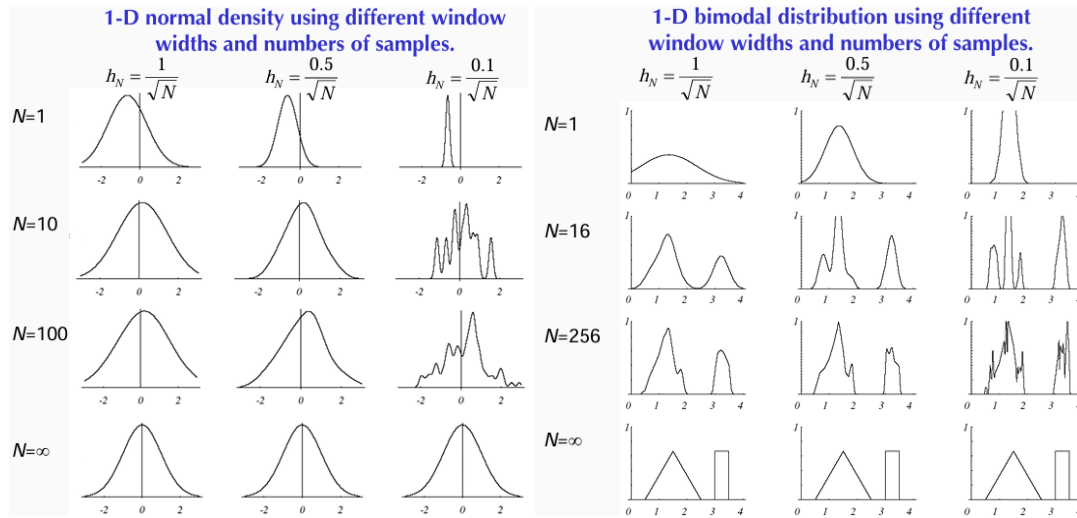


Since the training samples are independently and identically distributed accordingly to the unknown final density function, we have that the error is given by the convolution of  $E\{\hat{p}(x)\} = p(x) * \frac{1}{h^n} \gamma(\frac{x}{h})$ , meaning that the found estimate is a blurred version of the true one. The blurring decreases asymptotically with the number of training samples if the kernel width  $h^n$  is tuned correctly. As the number of samples goes to infinite, de bandwidth goes to 0 and the kernel approaches a delta function:  $\lim_{N \rightarrow \infty} \{\frac{1}{h_N^n} \gamma(\frac{x}{h_N})\} = \delta(x) \Rightarrow \lim_{N \rightarrow \infty} E\{\hat{p}(x)\} = p(x)$ , reducing the approximation error, or bias.

Given the same premises, the estimate variance is bounded by the formula:

$E\{(\hat{p}(x) - \bar{\bar{p}}(x))^2\} \leq \frac{\sup(\gamma(\cdot)) \bar{\bar{p}}(x)}{N h_N^n}$ . This means that the Parzen estimate is consistent if

$\lim_{N \rightarrow \infty} h_N = 0$ ,  $\lim_{N \rightarrow \infty} N h_N^n = +\infty$ , for example with  $h_N = \frac{1}{\sqrt[2n]{N}}$ .



**Gaussian kernel** with spherical symmetry is a commonly used kernel in Parzen Window estimation, as it's very ubiquitous and easy to manage. In this case the *pdf* can be expressed

as  $\hat{p}(x) = \frac{1}{N} \sum_{k=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{\|x-x_k\|^2}{2\sigma^2})$ , where the *standard deviation*  $\sigma$  of the kernel represents

a smoothing parameter whose tuning is required to prevent both *over- and under-fitting*. Since the *variance* is the same along all features, it is important to *normalize* them before undertaking the pdf estimation process. Furthermore, one may think to compute it in an adaptive way, i.e. by associating a  $\sigma_k$  to each training sample  $x_k$  ( $k = 1, 2, \dots, N$ ). In any case  $\sigma_k$  needs to be associated to a distance measure, which is usually the Euclidean distance

between  $x_k$  and the  $L$  nearest training samples  $y_1, y_2, \dots, y_L$ :  $\sigma_k = \frac{1}{L} \sum_{l=1}^L \|x_k - y_l\|$ . This

implementation, called **Specht method**, can be implemented by means of **Probabilistic Neural Networks (PNN)**,

In some cases, the model depends on unobserved latent variables, making it necessary to work out an estimation with incomplete data. Given  $N$  observations in a  $n$ -dimensional space  $X$ , where each observation is drawn from a distribution  $p(x)$  defined as a mixture of Gaussian

modes:  $p(x) = \sum_{i=1}^M P_i p(x|m_i, \Sigma_i)$  where  $P_i, m_i, \Sigma_i$  are, respectively, the prior probability, the mean vector and the covariance matrix of each component. The final goal is to estimate a vector of Gaussian distributions (thus a vector of  $M$  elements, where each element is a triplet with those three parameters) and the final distribution  $p(x)$  of each Gaussian mode that composes it. The Expectation Maximization (EM) algorithm can be used to define the vector of Gaussian distributions  $\theta^* = \arg \max l(\theta|X)$  where  $l(\theta|X) = p(X|\theta) = \prod_{i=1}^N p(x_i|\theta)$  and where the difficulty of the problem depends on the form of  $p(x_i|\theta)$ .

Given  $Y$  the set of missing data complementary to  $X$ , which form  $Z = (X, Y)$ , the joint density function is  $p(Z|\theta) = p(X, Y|\theta) = p(Y|X, \theta) \cdot p(X|\theta)$  and the complete-data likelihood function is  $l(\theta|Z) = p(Y|X, \theta)$ , while the incomplete one is  $l(\theta|X) = p(X|\theta)$ .

The **Expectation Maximization** algorithm then proceeds to find the best expected value of  $\log p(X, Y | \theta)$  given  $X$  and a current estimate of  $\theta$ , which is initially randomly chosen and then iteratively perfected until a local maximum is reached. This algorithm alternates between performing an expectation step, where conditional expectation  $E$  is computed via the formula:

$$Q(\theta, \theta^{(k)}) = E\{\log(p(X, Y|\theta) | X, \theta^{(k)})\} \Rightarrow \int \log(p(X, y|\theta)) p(y | X, \theta^{(k)}) dy ,$$

which computes an expectation of the likelihood by including the latent variables as if they were observed, and a maximization (M) step, which computes the maximum likelihood estimate of the parameters by maximizing the expected likelihood found on the E step via the formula:

$$\theta^{(k+1)} = \arg \max Q(\theta, \theta^{(k)}) .$$

The parameters vector  $\theta$  found on the M step is then used in the next expectation step, and the process is repeated until a local maximum is reached, as it can be shown that  $l(\theta^{(i+1)}) \geq l(\theta^{(k)})$

This algorithm can be used multiple times with different initial  $\theta^{(k)}$  to find all the maximums in the density function and to recover from local minima and saddle points.

EM equations for estimating the parameters of each mode are:

$$P_i^{[k+1]} = \frac{1}{N} \sum_{j=1}^N \frac{P_i^{[k]} p(\mathbf{x}_j | \mathbf{m}_i^{[k]}, \Sigma_i^{[k]})}{p(\mathbf{x}_j | \theta^{[k]})} \quad \mathbf{m}_i^{[k+1]} = \frac{\sum_{j=1}^N \frac{P_i^{[k]} p(\mathbf{x}_j | \mathbf{m}_i^{[k]}, \Sigma_i^{[k]})}{p(\mathbf{x}_j | \theta^{[k]})} \cdot \mathbf{x}_j}{\sum_{j=1}^N \frac{P_i^{[k]} p(\mathbf{x}_j | \mathbf{m}_i^{[k]}, \Sigma_i^{[k]})}{p(\mathbf{x}_j | \theta^{[k]})}}$$

$$\Sigma_i^{[k+1]} = \frac{\sum_{j=1}^N \frac{P_i^{[k]} p(\mathbf{x}_j | \mathbf{m}_i^{[k+1]}, \Sigma_i^{[k]})}{p(\mathbf{x}_j | \theta^{[k]})} \cdot (\mathbf{x}_j - \mathbf{m}_i^{[k+1]}) \cdot (\mathbf{x}_j - \mathbf{m}_i^{[k+1]})^t}{\sum_{j=1}^N \frac{P_i^{[k]} p(\mathbf{x}_j | \mathbf{m}_i^{[k+1]}, \Sigma_i^{[k]})}{p(\mathbf{x}_j | \theta^{[k]})}}$$

### CHAPTER 3 - FEATURE REDUCTION

In practical applications, the number of features is in the order of tens or hundreds, with the classical idea that increasing the number of considered features will increase the quality of the output. This is generally true, but after a certain point the number of features becomes too large to manage and the system loses accuracy. This effect is called "*curse of dimensionality*", or **Hughes effect**, and it's produced by wrong assumptions in model selection or estimation errors generated by the fact that higher dimensional observations need bigger amounts of training samples (**overfitting**). *For example, in a real world situation, just considering ten features would require at best  $10^5$  samples, an at worse  $10^{20}$  samples!*

Since most of the time is not possible to get more samples, a common solution is to simplify the model by assuming that the parameters are independent (thus setting to 0 all the covariances in the  $\Sigma$  matrix, reducing from  $\frac{n^2-n}{2} + n$  features to just  $n$ ) or even to 1 by further assuming that all the variances along the main diagonal are equal (thus reducing the whole covariance matrix to  $\Sigma = \sigma^2$ ). Another option is to reduce directly the number of features (thus keeping assumptions of inter-dependency between the parameters).

Given the volume of a hypersphere of radius  $r$  in  $n$  dimension  $V_s(r) = \frac{2r^n}{n} \frac{\pi^{n/2}}{\Gamma(n/2)}$  with

$\Gamma = \text{Gamma function}$  and the volume of a hypercube in  $[-r, r]^n$  to be  $V_c(r) = (2r)^n$  we can compute the fraction of the volume of a hypersphere inscribed in the hypercube of the same dimension with the formula  $f_n = \frac{V_s(r)}{V_c(r)} = \frac{1}{n2^{n-1}} \frac{\pi^{n/2}}{\Gamma(n/2)} \Rightarrow \lim_{n \rightarrow \infty} = 0$  meaning that as  $n$

increases, the volume of the inner hypersphere will be negligible with respect to the hypercube, so the volume will be concentrated in the corners of the hypercube. This means that the density is subject to a sort of centrifugal effect. In lower-dimensional spaces, data points are typically more evenly spread across the space, making it easier for a classification algorithm to identify clusters, boundaries, and relationships between data points. However, in higher dimensions, the data points occupy a much smaller fraction of the overall space, which can lead to a significant portion of the space being devoid of data points ("empty") and makes typical distance metrics irrelevant.

For this reason, **feature reduction** is very useful to keep the discrimination capability of the model as high as possible. This process can be done via **selection** methods or **transformation/ extraction**.

Features selection aims to select a subset  $F^*$  (set of  $m$  features) of  $F$  (set of  $n$  features) such that  $m < n$  and  $F^* = \arg \max \{J(F^*)\}$  where  $J(\cdot)$  is a function that measures the separability between the classes in the space defined by the considered subset of features.

The separability measure should be compatible with the discrimination criterion adopted by the classifier. Main feature selection strategies are:

- **Filtering:** feature selection is performed before training the model. The idea is to rank or give a score to each feature based on some statistical measure or separability criterion, independent of the learning algorithm. The model is trained once, and then features are filtered (or selected) in a single step. Popular methods include correlation metrics, mutual information, or statistical tests (like ANOVA or chi-square). It is computationally efficient but might not always capture interactions between features and the classifier
- **Wrapping:** it involves an iterative process where the feature selection is directly linked to the performance of the classifier. It selects features based on how well the classifier performs on different subsets of features, typically using accuracy or a related metric. The process loops through different feature subsets, trains the classifier, evaluates its

performance (often through cross-validation), and adjusts the selected features accordingly. This method usually provides better feature sets compared to filtering but is computationally expensive because it requires retraining the model for every subset of features.

- **Embedding:** feature selection happens during the training of the model itself. The model identifies and selects the most relevant features as part of the learning process. Algorithms like decision trees or Lasso regression inherently perform feature selection by assigning weights to features during the training process. Features with non-zero or higher weights are considered important. Embedding is more efficient than wrapping but tends to be specific to the model being used.

Typical separability measures used in feature selection are:

- **Accuracy:** the most simple and trivial, leads to scarce results, but very easy to use
- **Divergence:** it computes the overlap degree between two related densities. It's

implemented considering the likelihood ratio  $L_{ij}(x) = \frac{p(x|\omega_i)}{p(x|\omega_j)}$  with the formula:

$$D_{ij}(F') = E\{L'_{ij}(x)\} + E\{L'_{ji}(x)\} \text{ with } L'_{ij}(x) = \ln[L_{ij}(x)] = \ln[p(x|\omega_i)] - \ln[p(x|\omega_j)],$$

$$\text{thus } D_{ij}(F') = \int_x \{[p(x|\omega_i)] - [p(x|\omega_j)] \ln\left[\frac{p(x|\omega_i)}{p(x|\omega_j)}\right]\} dx.$$

This formula is drastically simpler when the considered distributions are *Gaussians*:

$$D_{ij}(F') = \frac{1}{2} \text{Tr}\{(\Sigma_i - \Sigma_j) \cdot (\Sigma_i^{-1} - \Sigma_j^{-1})\} + \frac{1}{2} \text{Tr}\{(\Sigma_i^{-1} - \Sigma_j^{-1}) \cdot (m_i - m_j) \cdot (m_i - m_j)^t\}$$

where  $\text{Tr}\{\cdot\}$  is the matrix *trace operator*,  $\Sigma$  are the *covariance matrices* and  $m$  are the *mean vectors*.

**Properties** of this measure are:  $\omega_i = \omega_j \Rightarrow D_{ij} = 0$ ,  $\omega_i \neq \omega_j \Rightarrow D_{ij} > 0$ ,  $D_{ij} = D_{ji}$ ,

$$D_{ij}(f_1, \dots, f_k) \leq D_{ij}(f_1, \dots, f_{k+1}), D_{ij}(f_1, \dots, f_k) = \sum_{q=1}^k D_{ij}(f_q) \text{ if features are independent.}$$

In other words the larger the divergence, the better the separability between classes.

A *drawback* of this measure is that as the difference (or distance) between the two classes continues to increase, the divergence measure also keeps increasing indefinitely, without reaching a limit or plateau, leading to *overemphasis* on already well-separated features and potential *distortion* in feature ranking.

For multiclass problems, the divergence measure can be deduced by averaging the binary divergence measures between all possible couples of the  $C$  classes:

$$D_{av}(F') = \sum_{i=1}^C \sum_{j>1}^C P(\omega_i) \cdot P(\omega_j) \cdot D_{ij}(F') \text{ or by simply taking the lowest binary}$$

divergence measure among all the possible binary measures.

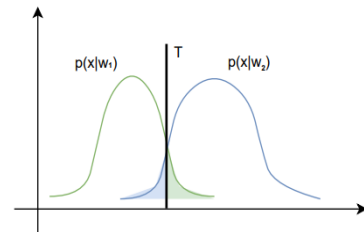
- **Bhattacharyya distance:** given a threshold that separates two classes, this distance provides a *measure that depends analytically on the probability of error of the Bayes classifier*, which is:

$$P(err) = P(err|w_1)p(w_1) + P(err|w_2)p(w_2)$$

$$P(err) = P(x \rightarrow w_2|w_1)p(w_1) + P(x \rightarrow w_1|w_2)p(w_2)$$

$$P(err) = \int_T^{+\infty} p(x|w_1) dx \cdot p(w_1) + \int_{-\infty}^T p(x|w_2) dx \cdot p(w_2)$$

the optimal position of the threshold, where the probability of error is *minimum*, is thus the intersection of the distributions.



Given  $\min[a, b] \leq a^s b^{1-s}$ ,  $0 < s < 1$  and  $a = p(x|w_1)p(w_1)$ ,  $b = p(x|w_2)p(w_2)$  we can define the minimum error as

$$\min P(err) = \int_{-\infty}^{+\infty} \min\{p(x|w_1)p(w_1), p(x|w_2)p(w_2)\} dx \rightarrow$$

$$\min P(err) \leq \int_{-\infty}^{+\infty} [p(x|w_1)p(w_1)]^s \cdot [p(x|w_2)p(w_2)]^{1-s} dx \rightarrow$$

$$\min P(err) \leq p(w_1)^s p(w_2)^{1-s} \int_{-\infty}^{+\infty} p(x|w_1)^s \cdot p(x|w_2)^{1-s} dx \text{ where the integral part can}$$

be written as  $\exp[-\mu_{12}(s)]$ , which is called "**Chernoff Distance**". This representation introduces an upper bound (or **Chernoff Bound**)  $\epsilon_\mu$  of the classification error, which is

computed as:  $\epsilon_\mu = p(w_1)^s p(w_2)^{1-s} \exp[-\mu_{ij}(s)]$ . A smaller Chernoff Bound implies a smaller minimum error, and thus a better separability between the two classes.

When  $s$  is set to  $s = \frac{1}{2}$  the bound is called **Bhattacharyya Bound** and the distance

becomes  $B_{ij} = \mu_{ij}(\frac{1}{2})$ . It follows that if the distance is always greater than 0 except

when the distributions are equal, in that case the distance is 0. **Bhattacharyya distance** decreases with increasing number of features, as a multidimensional space provides better distinction between the classes. If the features are independent then the distance of the final set of features is given by the sum of the distances of each feature. *The same concepts of divergence can be applied for multiclass definition.*

- **Jeffries-Matusita distance:** it's based on the *difference between the square roots of the distributions* and states that if the difference is bigger than 0, then the two classes are separate. It's computed using the formula:

$$JM_{ij} = \int \sqrt{p(x|w_1)} - \sqrt{p(x|w_2)} dx \Leftrightarrow JM_{ij}^2 = \int \left( \sqrt{p(x|w_1)} - \sqrt{p(x|w_2)} \right)^2 dx =$$

$$= \int p(x|w_1) + p(x|w_2) - 2\sqrt{p(x|w_1)}\sqrt{p(x|w_2)} dx =$$

$$= \int p(x|w_1) dx + \int p(x|w_2) dx - 2 \int \sqrt{p(x|w_1)}\sqrt{p(x|w_2)} dx \text{ where the first two terms are equal to 1 and the last integral is the Bhattacharyya Distance, this means:}$$

$$JM_{ij}^2 = 2 - 2\exp(B_{ij}) \Leftrightarrow JM_{ij} = \sqrt{2 \cdot (1 - \exp(-B_{ij}))}$$

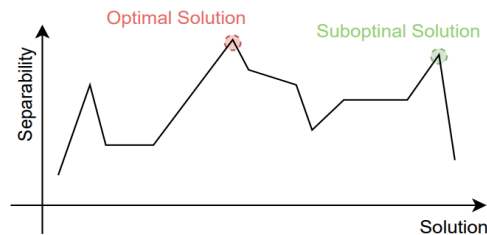
The next step after choosing the separability criterion is to *define a **search strategy*** to identify the best subset of features that optimizes the criterion.

A basic strategy is to employ **exhaustive search** to find the optimal solution at the cost of a computational cost that grows exponentially with the number of features  $m$ :  $(n, m) = \frac{n!}{m!(n-m)!}$

Since finding the optimal solution often requires a prohibitive cost, **suboptimal strategies** represent a fair compromise between efficiency and efficacy.

Most popular Suboptimal Strategies are:

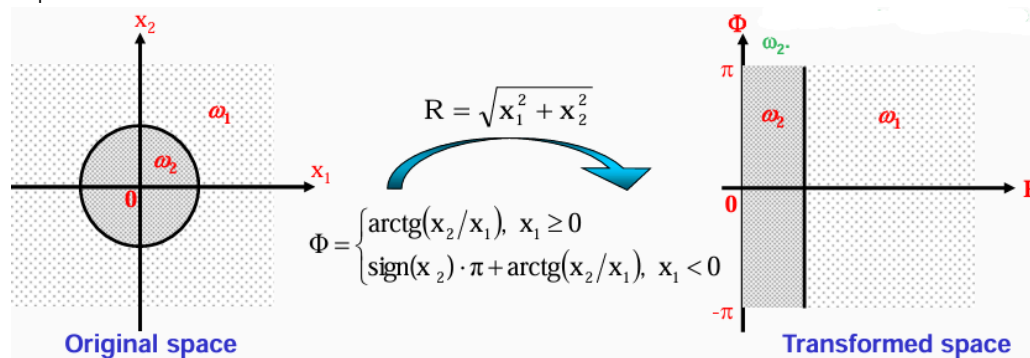
- **Sequential Forward Selection:** this strategy is based on a bottom up approach,



starting from 0 features and iteratively increasing the number until a predefined condition is reached. The *first feature* is the one that best singularly optimizes the adopted separability measure. *Subsequent features* are selected according to their joint optimization score, privileging the ones that best complement the so far selected set. This process stops when the set is complete with a predefined  $m$  number of features, and it is particularly useful when the number of features needs to be decreased drastically.

- **Sequential Backward Selection:** this strategy is based on a top-down approach, starting from all features and iteratively decreasing the number until a predefined condition is reached. At each iteration the *removed feature* is the one that less decreases the adopted separability measure. This process stops when the set is complete with a predefined  $m$  number of features, and is particularly useful when the number of features doesn't have to be reduced drastically. It has to be noted that, for the same number of iterations, SBS is computationally more expensive than SFS.

Another option is to combine features to compress information with a lower loss with respect to feature selection, allowing for better data representation. This process is called **features transformation** and it works by migrating to a transformed space where classes can be better separated.



The transformation can be either **linear** (easier to understand but not always so powerful) or **non-linear** (more difficult to understand but generally more powerful). This process is usually complemented by extraction methods that allow to derive entirely new features from the transformed space. These new features can provide better representation of the data, with a focus on dimensionality reduction. Important aspects that impact the feature extraction process are the eventual use of labels and the training objective, which can be **discriminative training** (*minimizing classification error*), **LDA** (*maximizing class separability*), **projection pursuit** (*retaining interesting directions*), **PCA** (*minimizing reconstruction error - capturing maximum variance*), **ICA** (*making features as independent as possible*).

**Linear methods** project the high-dimensional data onto a lower dimensional space, making linear combinations *simple to compute* and analytically tractable, as they reduce complexity in estimation and classification and enable visual examination of data.

Linear transformations take a *matrix of weights* and *multiply it with a vector* that can then be projected along the *different directions*, which are contained in the transformation matrix  $\Phi$ . Most common approaches are **Principal Components Analysis**, which aims to find the projection that best represents the data in a least square sense, thus *minimizing reconstruction error*, and **Linear Discriminant Analysis**, which aims instead to best separate the data.

**PCA**, or **Karhunen-Loève transform**, is an unsupervised feature extraction method used mainly in *pattern recognition* and *signal/image processing*, and it's based on the formula:

$J_m = \sum_{i=1}^N \left\| \sum_{k=1}^m y_{ik} \phi_k - x_i \right\|^2$  where  $\phi$  are the bases for the *subspace* and  $y$  are the principal components, which are the projections of  $x$  on the subspace.  $J_m$  is minimum when  $\phi$  are the eigenvectors of the scatter matrix  $S = \sum_{i=1}^N (x_i - m)(x_i - m)^t$  with the largest eigenvalues.

When the eigenvectors are sorted in descending order of the corresponding eigenvalues, the greatest variance (thus entropy) of the data lies on the first principal component and so on. In most real world scenarios there are just few large eigenvalues, and this implies that the  $m$ -dimensional subspace contains the signal and the remaining  $n-m$  dimensions generally contain noise.

**Linear Discriminant Analysis** aims to find a *projection* onto a line defined as  $y = w^t x$  where the points corresponding to the two classes are well separated. The **Fisher criterion** is used to *quantify the separability between classes* by considering means and variances along the projection direction. In addition to that the **scatter matrices** are defined, both **between**

**classes**:  $S_B = (m_1 - m_2)(m_1 - m_2)^t$  and **within classes**:  $S_W = S_1 + S_2$ . Now  $J$  can be

expressed as  $J = \frac{W^t S_B W}{W^t S_W W}$  where

$$W^t S_B W = W^t (m_1 - m_2)(m_1 - m_2)^t W = (W^t m_1 - W^t m_2)(W^t m_1 - W^t m_2)^t = (\tilde{m}_1 - \tilde{m}_2)^2$$

and  $W^t S_W W = W^t (S_1 + S_2) W = W^t S_1 W + W^t S_2 W$  and the result is given by the  $w$  that maximizes  $J(w)$ .

It has to be noted that both  $S_W$  and  $S_B$  are symmetric and positive semidefinite, but while the first one is also nonsingular if  $N > n$ , the second's one rank is at most 1.

To generalize to a  $C$  number of classes case,  $C - 1$  discriminant functions are needed, where the projection is onto a  $C - 1$  dimensional subspace. In this case, the scatter matrix within

classes will be given by the simple sum of each matrix:  $S_W = \sum_{i=1}^C S_i$ , while the scatter matrix

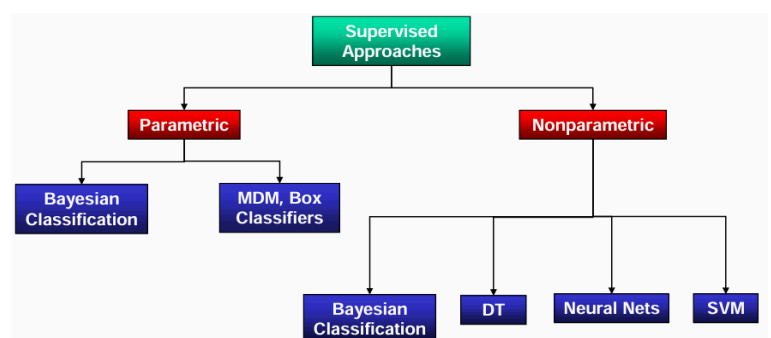
between classes will be given by the formula  $S_B = \sum_{i=1}^C [\#(X_i)](m_i - m)(m_i - m)^t$  where

$m = \frac{1}{N} \sum_{x \in X} x = \text{global mean vector}$ . The matrix  $W$  needs to be scalarized by using the

determinant. The final  $J(W)$  is maximized when the columns of  $W$  are the eigenvectors of  $S_W^{-1} S_B$  with the largest eigenvalues. Since the scatter matrix between classes is the sum of  $C$  matrices of *rank*  $\leq 1$  and given that only  $C - 1$  of these are independent, we can say that the matrix  $S_B$  is of *rank*  $\leq C - 1$ , thus no more than  $C - 1$  eigenvalues can be non-zero.

The **design of a supervised classifier**

depends on the available set of features, the available set of training samples, the typology of the adopted classification model and the cost function to optimize and the decision criterion.



An **MDM** classifier is based on the assumptions that classes are characterized by a *small sample dispersion around a barycenter* or that they have the *same statistical behavior*. The **Minimal Distance to Means** classifier models each class through its barycenter, which defines the class prototype. The classification is then done by *computing the distance of an element from each class prototype and assigning the element to the closest class*, partitioning the feature space with linear frontiers.

A **Box classifier** exploits 2° order statistics by modeling each class with a uniform pdf, centered on the class barycenter and with size proportional to the standard deviation along each dimension of the feature space. So each class is represented with a 3D box that represents the probability of the class. It has to be noted that the sum of the volumes of the boxes has to be 1, so for an element inside a box the probability of belonging to the class is not 1, but it's the height of the box.

**Bayesian classification** quantifies the *tradeoffs* between various classification decisions, on the assumption that the problem is posed in probabilistic terms and that relevant probability values are known. In problems with incomplete parametrization the adopted criterion used is either the **MinMax** or the **Neyman-Pearson**, while in problems with full parametrization the approach is to take the minimum risk, employing either the **MAP** Criterion or the **ML** Criterion.

**Maximum A Posteriori** criterion relies on the knowledge of the classes' probability  $P(\omega_j|x)$  and aims to *minimize the average probability of errors*. Decisions are taken by assigning a pattern  $x$  to the class that maximizes the posterior probability, which is computed using Bayes theorem as:  $x \in \omega \Leftrightarrow P(\omega_j)P(\omega_j|x) \geq P(\omega_j)P(x|\omega_j)$ . The probability of error is thus given by

$$P(err) = \int_{-\infty}^T P(\omega_2|x)p(x)dx + \int_T^{+\infty} P(\omega_1|x)p(x)dx = \int_{-\infty}^T P(err|x)p(x)dx + \int_T^{+\infty} P(err|x)p(x)dx$$

where  $T$  is the threshold, the first term is the slice of the second class before the threshold and the second term is the slice of the first class after the threshold. This means that maximizing the probability of the classes, thus separating them, will also minimize the probability of errors.

Given a decision region  $\mathfrak{R}_i$  generated for a classifier for the class  $\omega_i$  the average probability of

$$\text{error is given by: } P_e = \sum_{i=1}^C P(err|\omega_i)P(\omega_i) = \sum_{i=1}^C P(x \notin \mathfrak{R}_i|\omega_i)P(\omega_i).$$

In the special case where prior probabilities are equal or unknown, we talk about **Maximum Likelihood Criterion**. Unlike the Maximum A Posteriori (MAP) criterion, which leverages the prior probabilities of the classes  $P(j|x)$ , the **ML** criterion focuses solely on the likelihood of the data given a specific class,  $P(x|j)$ , without considering prior probabilities.

These two criteria belong to the **Minimum Risk Theory**, but they do not take into consideration the cost of the error, which is not always the same and often must be taken into account. Since the cost of an error depends on the true class the element belongs to, it's possible to define a **cost matrix**  $\Lambda$ , which can then be integrated into the decision making process. For each pattern  $x$  a conditional risk can be introduced. This risk is the *expected loss*

*of taking the wrong action*:  $R(\alpha_i|x) = \sum_{j=1}^C \lambda(\alpha_i|\omega_j)P(\omega_j|x)$ . In this way it's possible to choose the action  $\alpha$  that minimizes the conditional risk, resulting in an overall risk (or **Bayes risk**) of

$$R^* = \int_x R(\alpha^*|x)p(x)dx. \text{ The decision rule can be thus defined (for a 2 classes problem) as:}$$



$L(x) = \frac{p(x|\omega_1)}{p(x|\omega_2)} \geq \frac{p(\omega_2)(\lambda_{12}-\lambda_{22})}{p(\omega_1)(\lambda_{21}-\lambda_{11})}$  for  $\lambda_{21} - \lambda_{11} > 0$ , so if  $\lambda_{21} = \lambda_{12}$  and  $\lambda_{11} = \lambda_{22} = 0$  the minimum risk is equivalent to the MAP criterion.

**Discriminant functions** assign a score to each class and then choose the one with the highest score. The **score** is determined by **functions**  $g(x)$  that divide the space into C regions, separated by **decision boundaries**.

A classifier that minimizes conditional risk will have a score function  $g_i(x) = -R(a_i|x)$ , while the score function of one that minimizes error will be  $g_i(x) = P(\omega_i|x)$ . In the common case where the C classes are characterized by multivariate Gaussian distributions, the score function will be  $g_i(x) = \ln p(x|\omega_i) + \ln P(\omega_i)$  and can thus be rewritten (applying the log) as:

$$g_i'(x) = -\frac{1}{2}(x - m_i)^t \Sigma_i^{-1}(x - m_i) - \frac{n}{2} \ln 2\pi - \frac{1}{2} |\Sigma_i| + \ln P(\omega_i).$$

In the simplest case, *classes are assumed to be circularly symmetric*, meaning that the associated covariance matrix will be symmetric, and it can also be assumed that *all classes have the same covariance matrix*  $\Sigma_i = \sigma^2$ . This means that the covariance matrix is just a constant and can thus be removed from the formula of the score function, reducing it to:

$g_i'(x) = w^t x + \text{bias}$ , where  $w = \frac{m_i}{\sigma^2}$  and the  $\text{bias} = w_{i0} = -\frac{1}{2\sigma^2} m_i^t m_i + \ln P(\omega_i)$ , making it a simple *linear function*.

The set of points where a certain score function  $g_1(x)$  is equal to another function  $g_2(x)$  is called "**decision boundary**".

In the simplest case where covariance matrices are spherically symmetric the the decision boundary satisfy the equation  $w^t(x - x_0) = 0$ , with  $w = m_i - m_j$

and  $x_0 = \frac{1}{2}(m_i + m_j) - \frac{\sigma^2}{\|m_i - m_j\|^2} \ln \frac{P(\omega_i)}{P(\omega_j)} (m_i - m_j)$ . The decision boundary is thus a

*hyperplane orthogonal to the vector  $w$  and that passes through the point  $x_0$* . This means that it's sufficient to know the sign of the score function to determine the class (class 1 if positive, class 2 if negative, and boundary if 0).

Another simple case is when covariance matrices are radially symmetric, meaning that isolevels of the distributions will be ellipses instead of circles. In this case the discriminant function terms become:  $w = \Sigma^{-1} m_i$  and the  $\text{bias} = w_{i0} = -\frac{1}{2} m_i^t \Sigma^{-1} m_i + \ln P(\omega_i)$  and the

decision boundaries will be defined by the terms:  $w = \Sigma^{-1}(m_i - m_j)$

and  $x_0 = \frac{1}{2}(m_i + m_j) - \frac{\ln[P(\omega_i)/P(\omega_j)](m_i - m_j)}{(m_i - m_j)^t \Sigma^{-1}(m_i - m_j)}$ . This means that the hyperplane still passes

through  $x_0$ , but it isn't necessarily orthogonal to the line between the means.

In the most common and most general case, where covariance matrices are arbitrary, the *discriminant function will be quadratic*:  $g_i'(x) = x^t W_i x + w_i^t x + \text{bias}$  where the terms are:

$W_i = -\frac{1}{2} \Sigma_i^{-1}$ ,  $w_i = \Sigma_i^{-1} m_i$ ,  $\text{bias} = w_{i0} = -\frac{1}{2} m_i^t \Sigma_i^{-1} m_i - \frac{1}{2} \ln |\Sigma_i| + \ln P(\omega_i)$ . This means that the *boundaries are hyperquadrics*.

When addressing problems that require a categorical answer, where the answer is composed of a list of “natural” attributes instead of numerical features, a useful approach is to ask consequent questions, ordered in a tree-like map, that allows to iteratively gain more and more information until the question can be answered with a certain confidence. This kind of classification, implemented through **decision trees**, provides fast classification and easy embedding of prior human knowledge, with the drawback of introducing interpretability.

**Classification And Regression Trees (CART)** provide a general framework for decision trees, and are based on a set of different characteristics:

- **Split Number** (or **branching factor**) is the number of nodes each node can lead to. A common choice is to adopt a branching factor of 2, thus making the tree a binary tree.
- **Test Selection** is the logic behind the spatial separation of the classes, defining the decision boundaries. The goal is to find a splitting that favors simple and compact trees with few nodes.
- **Node Impurity** is a measure that quantifies the goodness of the test selection. It's denoted as  $i(N)$ . It can be computed in many ways, including:
  - Entropy:  $i(N) = - \sum_j P(\omega_j) \log_2(P(\omega_j))$  measures the disorder or unpredictability of a dataset, with higher values indicating more mixed classes. Most recommended when you prioritize maximizing information gain; often used in **classification tasks** to build deeper decision trees.
  - Variance:  $i(N) = P(\omega_1)P(\omega_2)$  reflects the spread of continuous target values, with higher variance indicating more diverse data points in regression tasks. Most recommended in **regression tasks**, as it measures the spread of continuous target values and helps in selecting splits that minimize prediction errors.
  - Gini:  $i(N) = \sum_{i \neq j} P(\omega_i)P(\omega_j) = 1 - \sum_j p^2(\omega_j)$  quantifies the probability of misclassifying a random data point, with higher values indicating more class mixing. Most recommended in **classification tasks** when computational efficiency is a priority; it's faster to calculate than entropy and commonly used in CART (Classification and Regression Trees).
  - Misclassification:  $i(N) = 1 - \max_j P(\omega_j)$  represents the fraction of incorrect predictions made by labeling all data with the most common class. Most recommended when simplicity and interpretability are important, often used as a baseline in **classification tasks**, though it may not perform as well as Gini or entropy for deeper trees.

Once the impurity measure is chosen, the nodes on the various levels of the tree can be chosen accordingly to which node apport a higher drop in impurity. For large trees it's common to implement locally greedy decisions that are computationally feasible, but do not guarantee global optimization.

- **Split Stop** is the criterion that decides when to stop splitting the space. If the space is too splitted the model will probably overfit, while if it is not split enough it will probably underfit. To decide the best moment, there are many techniques that can be employed:
  - **Cross-validation** is a technique where the dataset is split into subsets to evaluate model performance on unseen data, ensuring robustness, and is most recommended to prevent **overfitting** by assessing how well the tree generalizes across different data splits; it stops splitting when further divisions don't improve validation performance.

- **Impurity reduction threshold** is the minimum reduction in impurity needed to justify a split, and is used to avoid **insignificant splits** that over complicate the tree; it's recommended when you need to control tree depth and complexity for better interpretability.
- **Complexity-accuracy tradeoff criterion** balances building a simpler tree with achieving higher predictive accuracy, often controlled by pruning; it's recommended when you want to avoid **overfitting** and maintain a good balance between **model performance** and **interpretability**, ensuring added splits truly improve accuracy.
- **Hypothesis testing** applies statistical tests, such as chi-squared or F-tests, to determine if a potential split significantly improves the model; it's recommended when you need **rigorous statistical justification** for splits, especially in smaller datasets to avoid overfitting.
- **Pruning** is a technique used in decision trees to reduce their size by removing sections of the tree that provide little to no predictive power, with the goal of improving generalization and preventing overfitting. There are two main types: pre-pruning, which *stops the tree from growing further if certain conditions* (like a maximum depth or impurity threshold) are met, and post-pruning, which *removes branches from a fully grown tree based on performance on a validation set*.  
Pruning is most effective when a decision tree becomes overly complex and fits the training data too closely, leading to poor performance on unseen data. By simplifying the model, pruning strikes a balance between accuracy and complexity, making it a critical step in maintaining both model performance and interpretability.
- **Leaf Node Label Assignment** is just the labeling process of the nodes, it has to be done in a way that allows both humans and machines to clearly distinguish the various leaves.

If the feature space is subdivided by lines that are not strictly parallel to the axes, decisions at node level can be made with linear classifiers or implementing other conditions.

Since decision trees are prone to overfitting, a possible solution is to ensemble multiple trees to create a random forest of slightly different trees. The final decision is thus given by the majority vote.

**Approximation error**, also known as **bias**, is the error that arises from the inability of a model to perfectly capture the underlying true relationship between input features and the target variable. It is related to the complexity of the model and its capacity to represent the true function. This occurs when the model is too simple to capture the complexity of the data (e.g., using a linear model to fit data that has a nonlinear relationship). Models with high approximation error typically underfit the data, leading to poor performance both on the training set and on unseen data.

**Estimation error**, also referred to as **variance**, is the error that arises from the model's sensitivity to the specific dataset used for training. It reflects how much the model's predictions would vary if it was trained on different subsets of the data. This error is introduced when the model is too complex and adapts too closely to the training data, including noise and random fluctuations. High estimation error often leads to **overfitting**, where the model performs well on the training data but poorly on unseen test data.

**Generalization error** is the overall error that occurs when a model is applied to unseen data (test set). It quantifies how well a model trained on a particular dataset performs on new, unseen data. The generalization error is a combination of both approximation error and estimation error, plus the irreducible error.

The expected risk of error for a trained machine is given by the formula:

$$R(a) = \int_{X \times Y} L(y, f(x, a)) p(x, y) dx dy \text{ where } L(.) = \frac{1}{2} |y - f(x, a)| \text{ is the loss function.}$$

By averaging the measured average loss on the training set we obtain the Empirical Risk:

$$R(a) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i, a)).$$

Since the bigger N is, the most accurate this measure is, and in

most cases N is significantly smaller than infinite, the accuracy evaluation rely on some approximation methods:

	Leave One Out	Hold-Out	K-Fold	Monte Carlo	Kxl-Fold
<i>method</i>	In LOO, for a dataset with n data points, the model is trained on n-1 data points and tested on the remaining one. This process is repeated n times, each time leaving out a different data point.	The dataset is randomly split into two subsets: a training set and a test set (e.g., 70% for training, 30% for testing). The model is trained on the training set and tested on the test set.	The dataset is split into K equal parts (folds). The model is trained on K-1 folds and tested on the remaining fold. This process is repeated K times, each time using a different fold as the test set. The final performance is the average of all K test results.	This method involves randomly splitting the dataset multiple times into training and testing sets (like the hold-out method), but it repeats this process many times (e.g., 100 iterations) to get a more robust performance estimate.	This is a variation of K-fold cross-validation where the dataset is split into K folds, and the cross-validation is repeated L times with different random splits each time. The results of all rounds are averaged.
<i>pros</i>	Uses almost all data for training, resulting in low bias.	Simple and quick to implement.	Each data point gets to be in both training and test sets.	Provides a more stable estimate of performance by averaging over random splits.	Provides a more robust estimate compared to regular K-fold cross-validation.
<i>cons</i>	Very computationally expensive for large datasets since it requires n models to be trained.	The model might not generalize well to new data if the split is not representative.	Requires training K models, which can be computationally expensive.	Computationally expensive and can suffer from random sampling bias in each split.	Computationally more intensive than standard K-fold.
<i>use case</i>	Suitable for small datasets where each data point is valuable.	Often used for initial, fast evaluations.	Widely used when a more accurate estimate is needed.	Used when more robustness is needed.	Useful when looking for more stability in the performance estimate.
<i>type</i>	Exhaustive	Non-Exhaustive	Non-Exhaustive	Non-Exhaustive	Nested

For binary classifiers multiple measures are used to express the quality of the model. These measures rely on the so called **confusion matrix**:

Confusion Matrix and Common Derivations		
	Decision ( $D_0$ )	Decision ( $D_1$ )
True ( $H_0$ )	TN	FP
True ( $H_1$ )	FN	TP

$$Accuracy = \frac{TN+TP}{TN+TP+FN+FP}$$

$$Specificity = \frac{TN}{TN+FP}$$

$$Precision = \frac{TP}{TP+FP}$$

$$F_\beta = \frac{(1+\beta^2)(Precision \cdot Sensitivity)}{\beta^2 Precision + Sensitivity}$$

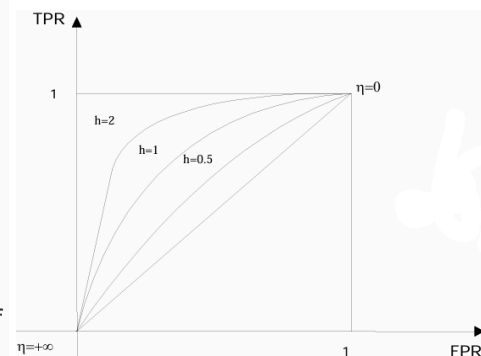
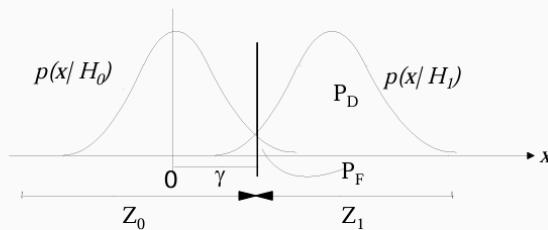
$$Sensitivity = \frac{TP}{TP+FN}$$

$$F_1 = \frac{2 \cdot Precision \cdot Sensitivity}{Precision + Sensitivity}$$

↑  
Also called 'Recall'

The **Receiver Operating Characteristic** is another tool to assess binary classifiers' performance. It's a curve that captures the behavior of the TPR as a function of the FPR by varying the decision threshold. In the 1-D Gaussian case example we obtain:

$$n = 1, \quad p(x|H_0) = N(0, \sigma^2), \quad p(x|H_1) = N(m, \sigma^2)$$



In this case, the ROC curves can be **parameterized** in terms of  $h=m/\sigma=0.5, 1, 2, \dots$

The confusion matrix in the case of a C multiclass classifier becomes

	$\hat{\omega}_1$	$\hat{\omega}_2$	...	$\hat{\omega}_C$
$\omega_1$	$N_{11}$	$N_{12}$	...	$N_{1C}$
$\omega_2$	$N_{21}$	$N_{22}$	...	$N_{2C}$
...	...	...	...	...
$\omega_C$	$N_{C1}$	$N_{C2}$	...	$N_{CC}$

$$Overall Accuracy = \frac{\sum_{i=1}^C N_{ii}}{\sum_{i=1}^C \sum_{j=1}^C N_{ij}}$$

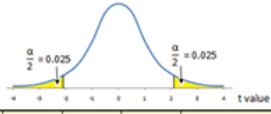
$$Accuracy(\omega_i) = \frac{N_{ii}}{\sum_{j=1}^C N_{ij}}$$

$$Average Accuracy = \frac{\sum_{i=1}^C Accuracy(\omega_i)}{C}$$

To define which model is better, statistical measures can be adopted. The most common is the so-called **Student test** (T-Ttest) that relies on a first measure of the accuracy of the two models using k-fold cross validation. It starts with a null hypothesis of the mean difference being zero. It defines a T-value as the measured difference between the two groups' means

divided by the variability of the groups. The T-Value is then compared with a critical value  $T_c$ . If the measured value is higher than the critical value it means that the difference in accuracy is statistically significant. We thus define a table to easily see the critical values associated to each level of accuracy:

**Student's t Distribution Table**



	90%	95%	97.5%	99%	99.5%	99.95%	1-Tail Confidence Level
	80%	90%	95%	98%	99%	99.9%	2-Tail Confidence Level
	0.100	0.050	0.025	0.010	0.005	0.0005	1-Tail Alpha
df	0.20	0.10	0.05	0.02	0.01	0.001	2-Tail Alpha
1	3.0777	6.3138	12.7062	31.8205	63.6567	636.6192	
2	1.8856	2.9200	4.3027	6.9646	9.9248	31.5991	
3	1.6377	2.3534	3.1824	4.5407	5.8409	12.9240	
4	1.5332	2.1318	2.7764	3.7469	4.6041	8.6103	
5	1.4759	2.0150	2.5706	3.3649	4.0321	6.8688	
6	1.4398	1.9432	2.4469	3.1427	3.7074	5.9588	
7	1.4149	1.8946	2.3646	2.9880	3.4995	5.4079	
8	1.3968	1.8595	2.3060	2.8965	3.3554	5.0413	
9	1.3830	1.8331	2.2622	2.8214	3.2498	4.7809	
10	1.3722	1.8125	2.2281	2.7638	3.1693	4.5869	
11	1.3634	1.7959	2.2010	2.7181	3.1058	4.4370	
12	1.3562	1.7823	2.1788	2.6810	3.0545	4.3178	
13	1.3502	1.7709	2.1604	2.6503	3.0123	4.2208	
14	1.3450	1.7613	2.1448	2.6245	2.9768	4.1405	
15	1.3406	1.7531	2.1314	2.6025	2.9467	4.0728	
16	1.3368	1.7459	2.1199	2.5835	2.9208	4.0150	
17	1.3334	1.7396	2.1098	2.5669	2.8982	3.9651	
18	1.3304	1.7341	2.1009	2.5524	2.8784	3.9216	
19	1.3277	1.7291	2.0930	2.5395	2.8609	3.8834	
20	1.3253	1.7247	2.0860	2.5280	2.8453	3.8495	
21	1.3232	1.7207	2.0796	2.5176	2.8314	3.8193	
22	1.3212	1.7171	2.0739	2.5083	2.8188	3.7921	
23	1.3195	1.7139	2.0687	2.4999	2.8073	3.7676	
24	1.3178	1.7109	2.0639	2.4922	2.7969	3.7454	
25	1.3163	1.7081	2.0595	2.4851	2.7874	3.7251	
26	1.3150	1.7056	2.0555	2.4786	2.7787	3.7066	
27	1.3137	1.7033	2.0518	2.4727	2.7707	3.6896	
28	1.3125	1.7011	2.0484	2.4671	2.7633	3.6739	
29	1.3114	1.6991	2.0452	2.4620	2.7564	3.6594	
30	1.3104	1.6973	2.0423	2.4573	2.7500	3.6460	

**Loss function:** Loss functions are used to quantify the error between predictions and actual values. In regression, common loss functions include:

- **Mean Squared Error (MSE):** Measures the average squared difference between predictions and actual values. It penalizes larger errors more than smaller ones.
- **Least Squares Error (LSE)** is commonly used to measure the accuracy of predictions in regression tasks. Mean Squared Error (MSE) is the average of these squared errors.
- **Mean Absolute Error (MAE):** Measures the average absolute difference, treating all errors equally, which makes it more robust to outliers than MSE.
- **Huber Loss:** Combines MSE and MAE, behaving like MAE for larger errors and MSE for smaller errors, providing a balance between sensitivity to outliers and small errors.

**Maximum likelihood (ML) estimation** is a method for estimating the parameters of a statistical model. The goal is to find the parameter values that maximize the likelihood function, which measures the probability of observing the given data given those parameters. For example, if we assume our data follow a Gaussian (normal) distribution, maximum likelihood estimation would involve finding the mean ( $\mu$ ) and variance ( $\sigma^2$ ) that makes the observed data most probable under this Gaussian model.

Given the assumption of a Gaussian distribution, the mean  $\mu$  that maximizes the likelihood of observing the data is simply the average of the observed data points. This aligns with our intuition about the Gaussian distribution: the most probable center of the data is the point around which the data cluster, which is captured by the sample mean.

Demonstration: Given the PDF of a Gaussian distribution  $f(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$  with  $x$

being a vector of dimension  $n$ , the likelihood is given by  $L(\mu, \sigma^2) = \prod_{i=1}^n f(x_i; \mu, \sigma^2)$ . We use the

log-likelihood to simplify the computations, replacing multiplications with additions. The Maximum Likelihood is thus given by the derivative for  $\mu$  of the log-likelihood:

$$\log L(\mu) = \sum_{i=1}^n \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}\right) \rightarrow \text{expanding the log} \rightarrow = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2 \rightarrow$$

$$\rightarrow \text{deriving} \rightarrow \frac{d}{d\mu} \log L(\mu) = -\frac{1}{2\sigma^2} \cdot 2 \cdot \sum_{i=1}^n (x_i - \mu) \cdot (-1) \text{ note that the first term was taken out as}$$

$$\text{it's derivative is 0 since it does not contain } \mu. \rightarrow \text{simplifying} \rightarrow = \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu) \rightarrow \text{setting to 0}$$

$$\text{to maximize} \rightarrow \mu = \frac{1}{n} \sum_{i=1}^n x_i \text{ which is the sample mean.}$$