Performance Evaluation

- No free lunch theorem
  - There isn’t a classifier that always performs better than the others
  - It is then important to properly evaluate classifier performance
- Apparent Error Rate (resubstitution error)
  - Evaluated on the whole dataset
  - Do not evaluate the generalization power of the classifier
- Error estimation
  - if it is possible the dataset must be split into three sets:
    - Training set (for training the classifier)
    - Validation set (for avoiding the *overtraining*)
    - Test set (for performance evaluation)

Error estimation

- Hold-out
  - Splitting dataset in a training set and a test set (50/50)
  - More than one split (for example, 10)

- K-fold cross validation
  - The dataset is divided into *K* subsets: *K*-1 are used for training and 1 for testing the classifier. The process is repeated for *K* times; then results are averaged.
  - Leave-one-out: when *K*=*n* (*n* is the size of the dataset)

- Bootstrap
  - Subsampling with replacement
  - We average the error rates on *L* sets

Observations

- Hold-out
  - can be used for “large” datasets

- K-fold cross validation & Bootstrap
  - it is a good choice for “small” datasets

- Note:
  - Statistical significance tests should be used for evaluating differences in classification performance

Classifier Complexity

- There are several problems when the number d of the *feature* to be used grows:

  – computational complexity;

  – Hughes phenomenon.

- Computational complexity

  – When the value of d grows, the computational complexity of the classifier increases. For some architectures such an increment is linear with d, for other architectures it is superlinear (e.g.: quadratic).

  – The increase in complexity determines longer computational times and more significant memory requirements.

Hughes phenomenon ("curse of dimensionality")

- When d (the number of available features) grows, the amount of information for the classifier should grow, too. The classifier accuracy should then improve as well.

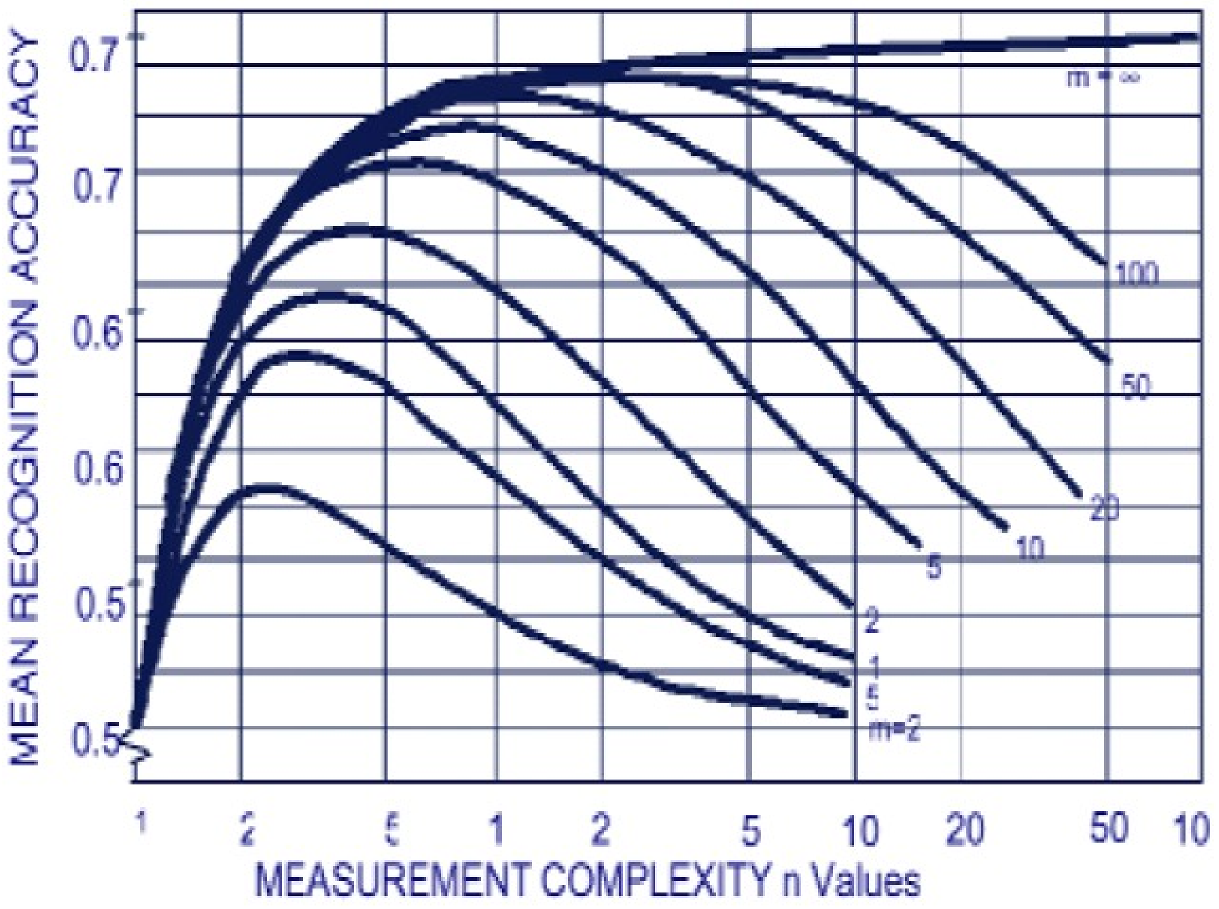
But…

- **Experimental observation**:
  – On the contrary, fixing the number n of the *training* patterns, the probability of a correct decision of the classifier increases when $1 \leq d \leq d^*$ and then decreases for $d > d^*$ (*Hughes phenomenon*).
- Interpretation

– When d grows, the number K_d of the classifier parameters is higher and higher.

– When the ratio K_d/n grows, "too few" training patterns are available for performing a reliable estimation

Hughes phenomenon



Feature reduction

- A possible solution is the reduction of the number *d* of the feature employed in the classification process (*feature reduction*).

- Drawback: the reduction of the feature space implies a possible information loss.

- There are two basic strategies for feature reduction:
  - feature selection: the identification, within the set of the *d* available *features*, of a subset of *m features*. Such a subset is chosen so as to minimize the information loss, by using a predefined optimality criterion;
  - feature extraction: a (often linear) transformation of the original (*d*-dimensional) feature space in another space having a dimension $m<d$, defined so as to minimize the information loss.

Feature selection

- Problem formulation::

– given a set $X = \{x_1, x_2, \ldots, x_d\}$ of *d features*, we have to identify the subset $S \subset X$, made up of *m features* ($m < d$), which maximizes a suitably chosen functional $J(\cdot)$:

$$S^* = \arg\max_{S \subset X} \mathcal{J}(S)$$

- A *feature selection* algorithm is then defined by means of two distinct objects:

  – the functional $J(\cdot)$, which must be defined so that $J(S)$ is a measure of the "goodness" of the *feature* subset S in the classification process;

  – the algorithm for searching the subset S^*. The number of subsets of X is, in fact, exponential. So, an exhaustive search can be affordable only for *very small values* of d.

- Sub-optimal strategies are adopted in practical cases.

The functional $J(\cdot)$

- The functional $J(\cdot)$ must be a measure of the "goodness" of the *feature* subset S when used in the classification process
- The "class separation statistic" (CSS) can be used

$$J = \int_0^1 \left| p(x \mid \omega_1) - p(x \mid \omega_2) \right| dx$$

- In practical cases, simpler forms are used for $J(\cdot)$, e.g.:

$$J = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

Sequential Forward Selection

- In general the search for the optimal subset of the *m* features is carried out by means of a suboptimal algorithm.

- Let us illustrate, as an example, the *sequential forward selection* (SFS) algorithm:

  - initialize $S^* = \varnothing$;
  - evaluate the value of the functional $J(.)$ for all the subset $S^* \cup \{x_i\}$, with $x_i \not\subset S^*$ and choose the *feature* $x^* \not\subset S$ which maximizes $J(S^* \cup \{x_i\})$;
  - update S^*: $S^* = S^* \cup \{x^*\}$;
  - iteratively add one *feature* at time, until the size of S^* reaches the desired *m* value or the value of $J(.)$ reaches a stable value (saturation).

SFS: observations

- SFS identifies the optimal subset that can be obtained by adding one feature at time.
- As first step, it chooses the *single* feature that gives rise to the maximum value of the predefined functional $J(.)$. At the second step, it adds the feature that, together with the previous one, provides the maximum value of the functional, and so on.

– The method is suboptimal, e.g., the best *couple* of features could not contain the best *single* feature.

- Advantages

– The computational burden of SFS is quite limited even if X is made up of hundreds of features.

- Disadvantages

– A feature inserted into the subset S^* within a certain iteration cannot be eliminated during a successive iteration, i.e. SFS does not allow *backtracking*.

Feature extraction: the Karhunen-Loeve transform

- In “small sample size” cases another approach can be the use of a transformation of the feature space that converts a distribution with an arbitrary S in a distribution with a diagonal S, which allows a feature reduction

- The aim of the KL transformation (or PCA, Principal Component Analysis) is the generation of *d uncorrelated features* $y_1, y_2, \ldots, y_d$ starting from *d* generic features $x_1, x_2, \ldots, x_d$. In the Gaussian case, the *d features* will be also independent.

– By composing row-by-row the eigenvectors, the following linear transformation is built:

$$A = \begin{bmatrix} \mathbf{e}_1^t \\ \mathbf{e}_2^t \\ \vdots \\ \mathbf{e}_d^t \end{bmatrix}, \quad \mathbf{y} = A\mathbf{x}$$

– The covariance matrix in the transformed space becomes diagonal:

$$\mathrm{Cov}\{\mathbf{y}\} = \begin{bmatrix} \sigma^*_{11} & 0 & \cdots & 0 \\ 0 & \sigma^*_{22} & \cdots & 0 \\ \cdots & \cdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma^*_{dd} \end{bmatrix}$$

Whitening

- The KL transformation generates, by means of a diagonalization of Σ , d uncorrelated features y_1, y_2, …, y_d, whose variances are the eigenvalues of Σ :
  var$\{y_i\} = \lambda_i$, $i = 1, 2,\ldots, d$.

- The *whitening* generates other d features z_1, z_2, …, z_d, which, besides to be uncorrelated, have equal variances.

– It is necessary to normalize *features* on the basis of their variance.

- We can normalize all the variances, by defining:

$$z_i = \frac{y_i}{\sqrt{\lambda_i}} \Rightarrow \text{var}\{z_i\} = 1, \qquad i = 1, 2, \ldots, d$$

– Starting from a gaussian d-dimensional vector it is possible to obtain d monodimensional gaussians, independent each other and with equal “dispersion”.

- At this point it is possible to select the first $m<d$ eigenvalues

PCA (KL transform): feature extraction

- By sorting the *n* eigenvalues in a descending order (i.e.: $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_d$), the PCA transformation projects the pattern along the axes $\mathbf{e}_1$, $\mathbf{e}_2$, …, $\mathbf{e}_m$ that correspond to the first *m* eigenvalues:

$$y_{ik} = \mathbf{e}_i^t(\mathbf{x}_k - \mu) \;\; \forall i,k \Rightarrow \mathbf{y}_k = T(\mathbf{x}_k - \mu) \text{ with } T = \begin{bmatrix} \mathbf{e}_1^t \\ \mathbf{e}_2^t \\ \vdots \\ \mathbf{e}_m^t \end{bmatrix}$$

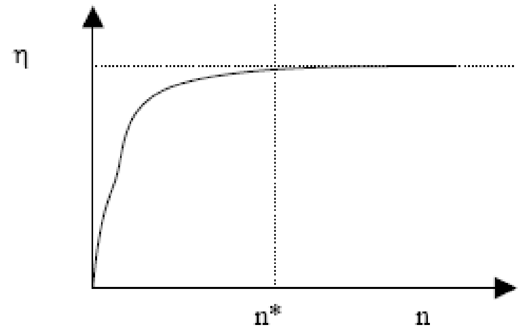
- Operatively, the PCA is carried out by:
- – evaluating the center μ and the covariance of the *dataset*;
- – calculate the eigenvalues of Σ and the corresponding eigenvectors;
- – sorting the eigenvalues in descending order;
- – building the transformation matrix *T* by linking row-by-row the eigenvectors that corresponds to the first *m* eigenvalues

PCA: observations

– The PCA transformation is: $\mathbf{y} = T(\mathbf{x} - \mu)$.

– The amount of information which is loss with the feature reduction can be evaluated by means of the following factor:

$$\eta = \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$



- The components along the axes: $\mathbf{e}_1$, $\mathbf{e}_2$, …, $\mathbf{e}_n$ are the *Principal Components*; so “the PCA holds the first *m* principal components”.
- $\mathbf{e}_1$ is the direction of the maximum dispersion of the patterns. The PCA implicitly assumes that the information is incorporated within data variance

Selection vs. Extraction

- Extraction methods: pros
  - The extraction method projects the original feature space onto a subspace chosen in order to preserve the maximum possible amount of information. It then presents an high flexibility (*selection* can be seen as a particular case of *extraction*).

- Selection methods: pros
  - *Features* given by a selection method are a subset of the original ones and then their physical meaning is preserved. This can be important when information about the interpretation of the various feature must be integrated during the classification process (e.g.: *knowledge-based* methods). On the contrary, the extraction method generates “virtual” *features*, defined by means of linear combinations of the original ones and then without a real physical meaning.