Mini Tutorial

Three Hours on Multiple Classifier Systems

Lecturer

Fabio Roli

University of Cagliari

Dept. of Electrical and Electronics Eng., Italy

email roli@diee.unica.it

Terminology

According to Nello's terminology, today we speak of *pattern matching* with multiple classes

also called *pattern classification* [Duda,Hart,Stork]

Terminology

Multiple classifier systems is only one of the "multiple" names used for the topic of this mini tutorial

- ✓combination of multiple classifiers [Lam95,Woods97,Xu92,Kittler98]
- ✓ensemble learning [Jordan95]
- ✓classifier fusion [Cho95,Gader96,Grabisch92,Keller94,Bloch96]
- ✓mixture of experts [Jacobs91,Jacobs95,Jordan95,Nowlan91]
- ✓consensus aggregation [Benediktsson92,Ng92,Benediktsson97]
- ✓voting pool of classifiers [Battiti94]
- ✓composite classifier systems [Dasarathy78]
- ✓classifier ensembles [Drucker94,Filippi94,Sharkey99]
- ✓modular systems [Sharkey99]
- ✓collective recognition [Rastrigin81,Barabash83]
- ✓stacked generalization [Wolpert92]
- ✓divide-and-conquer classifiers [Chiang94]
- ✓pandemonium system of reflective agents [Smieja96]
- ✓etc.

Tutorial Aims and Outline

- An introductive, three hours, tutorial on multiple classifiers

  - Part 1: Motivations and basic concepts
  - Part 2: Main methods for creating multiple classifiers
  - Part 3: Main methods for fusing multiple classifiers

PART I

Motivations and basic concepts

The traditional approach to Pattern Classification

- Unfortunately, no dominant classifier exists for all the data distributions ("no free lunch" theorem), and the data distribution of the task at hand is usually unknown

•CLASSIFIER EVALUATION AND SELECTION: *evaluation* of a set of different classification algorithms (or different "versions" of the same algorithm) against a *representative* pattern sample, and *selection* of the best one

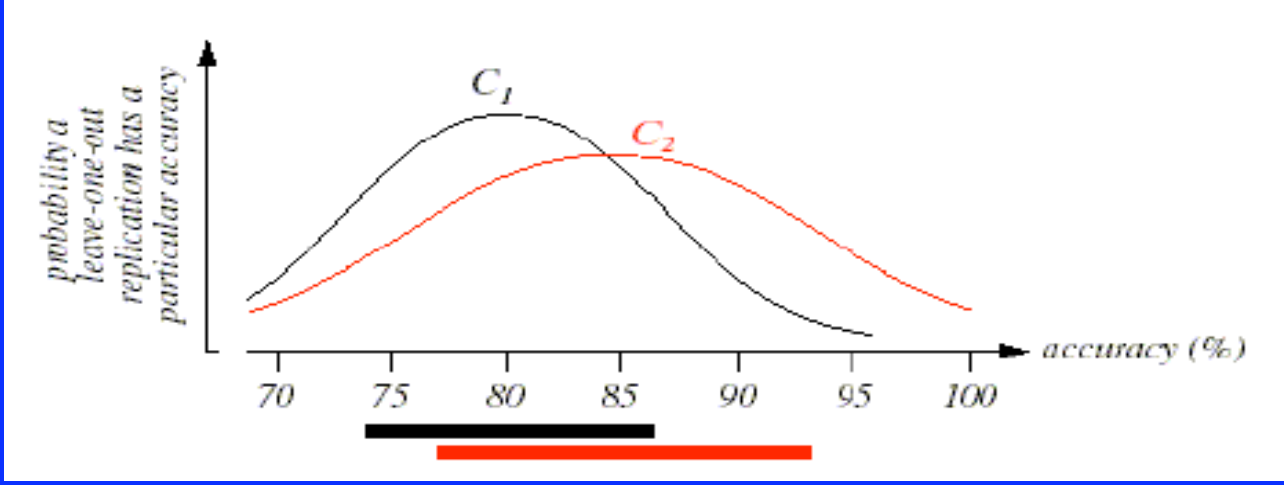
➢I design a set of N classifiers $C_1, C_2, \ldots, C_N$

➢I *evaluate* classifier errors $E_1 < E_2 < E_3 < \ldots < E_N$ (with related confidence intervals) using a validation set

➢I *select* the best classifier C_1 , and consider it the "optimal" one

The traditional approach: Small Sample Size Issue

- The traditional approach works well when a large and representative data set is available *("large" sample size cases),* so that estimated errors allow to select the best classifier



•However, in many small sample-size real cases, validation set provides just *apparent* errors that differ from true errors E_i:

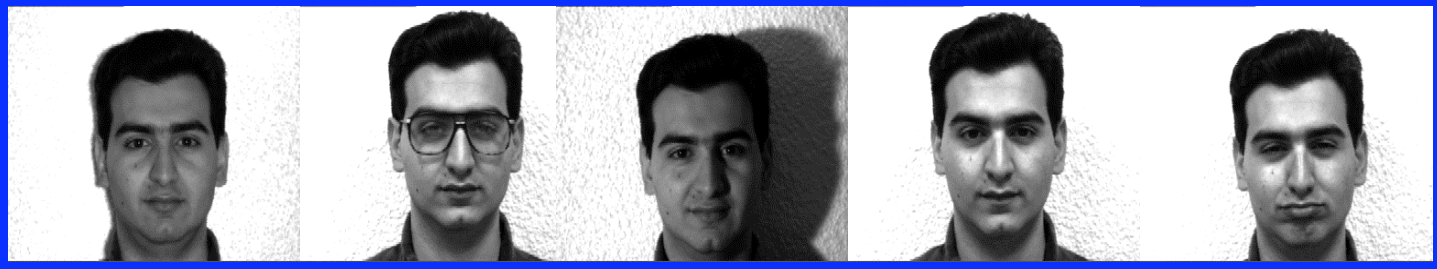
$$\hat{E}_i = E_i \pm \Delta_i$$

This can make impossible the selection of the optimal, if any, classifier, and, in the ***worst case****, I could select the* ***worst*** *classifier*

A practical example

Face recognition using PCA and LDA algorithms

Faces in the **validation set** (*Yale data base*)



High "Variance"

Faces in the **test set**

Apparent error caused from poorly representative validation set can make impossible to select the best one between PCA and LDA

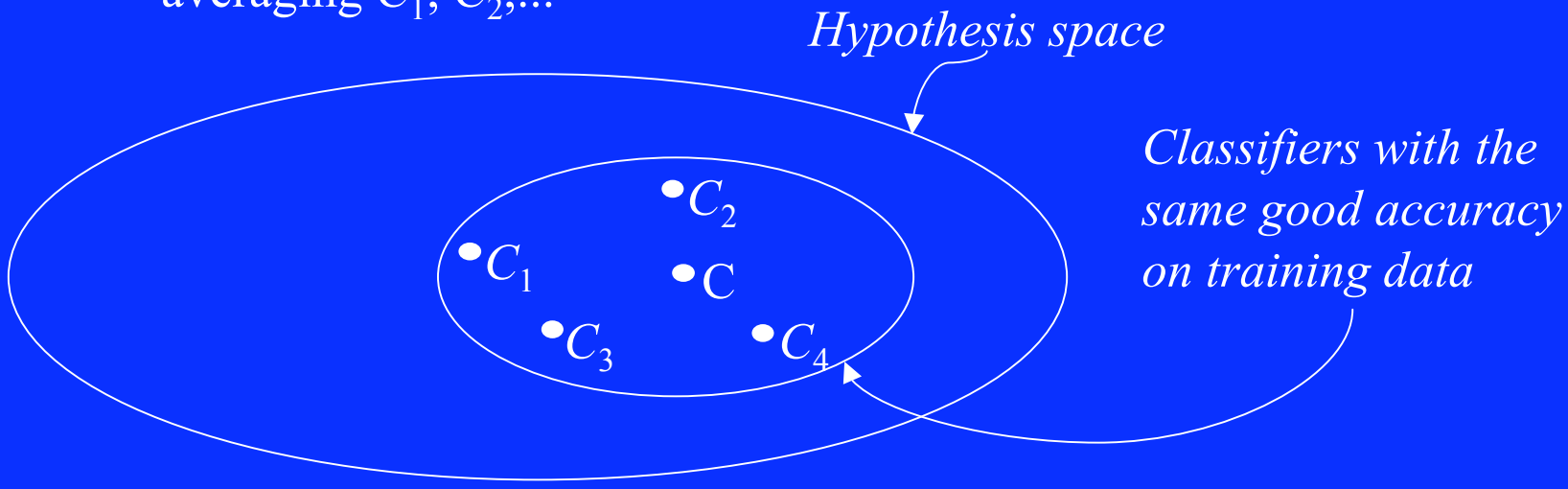
Multiple Classifier Fusion: *Worst Case* Motivation

- In the small sample size case, it is pretty intuitive that I can avoid selection of the worst classifier by, for example, averaging over the individual classifiers

A paradigmatic example (Tom Dietterich, MCS 2000 Workshop)

Few training data with respect to the size of the hypothesis space

- several classifiers (C_1,C_2,...) can provide the same accuracy on validation data
- a good approximation of the optimal classifier C can be found by averaging C_1, C_2,...



A practical example

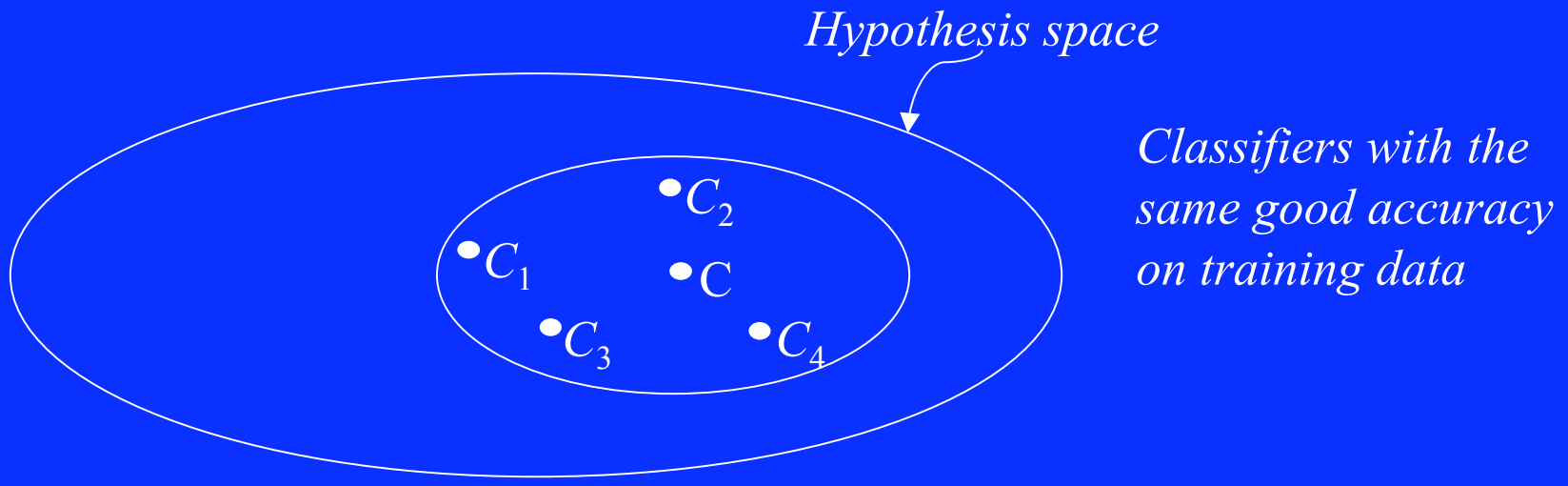
Face recognition using PCA and LDA algorithms (Yale data base)

For different choices of the training set (different "trials"), the best classifier varies. Fusion by averaging avoids to select the worst classifier for some test cases (Marcialis and Roli, *Int. Journal of Image and Graphics,* 2006).

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
PCA	76,7%	87,8%	**92,2%**	84,4%	**88,9%**
LDA	**83,3%**	**90,0%**	85,6%	84,4%	86,7%
Fusion by Average	80,0%	92,2%	88,9%	86,7%	88,9%

Theoretical support for the *worst case* motivation

Tom Dietterich's claim (2000)



In 2005, Fumera and Roli confirmed theoretically the claim of Tom Dietterich.

They proved that averaging of classifiers outputs guarantees a better test set performance than the worst classifier of the ensemble (IEEE-T on PAMI, June 2005).

Multiple Classifier Fusion: *Best Case* Motivation

- Beside avoiding the selection of the worst classifier, under particular hypotheses, fusion of multiple classifiers can improve the performance of the best individual classifiers and, in some special cases, provide the optimal Bayes classifier

•This is possible if individual classifiers make "different" errors.

•*Luckily, we have many experimental evidences about that ! !*

•Theoretical support for some classes of fusers (e.g., linear combiners, majority voting)

•For linear combiners, Tumer and Ghosh (1996) showed that averaging the outputs of individual classifiers with unbiased and uncorrelated errors can improve the performance of the best individual classifier and, for an infinite number of classifiers, provides the optimal Bayes classifier

Experimental evidences: Multimodal Biometrics
(Roli et al., Information Fusion Conf., 2002)

- XM2VTS database
  - face images, video sequences, speech recordings
  - 200 training and 25 test clients, 70 test impostors

•Eight classifiers based on different techniques: two speech classifiers, six face classifiers

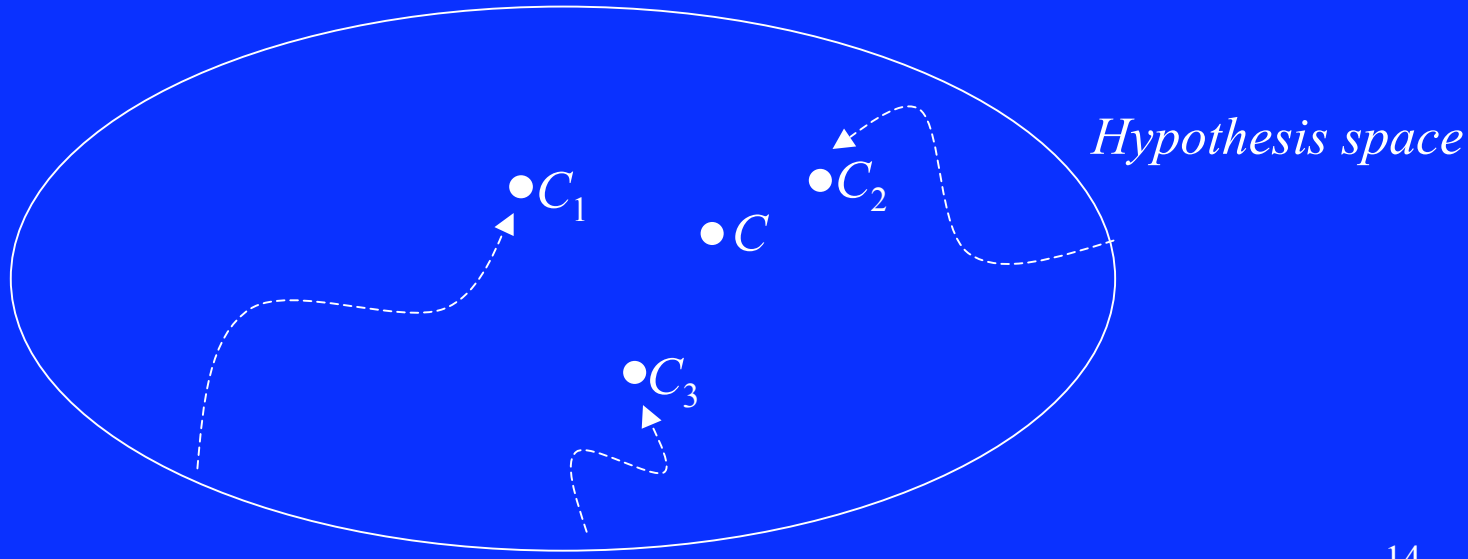
•Simple averaging allows avoiding the selection of the worst classifier for some test cases and, in some experiments, outperformed the best individual classifier

Fusion of multiple classifiers: computational motivation (T.Dietterich, 2000)

Many learning algorithms suffer from the problem of local minima

- Neural Networks, Decision Trees (optimal training is NP-hard!)
- Finding the best classifier *C* can be difficult even with enough training data

➢Fusion of multiple classifiers constructed by running the training algorithm from different starting points can better approximate *C*



Further Motivations for Multiple Classifiers

- In ***sensor fusion***, multiple classifiers are naturally motivated by the application requirements

- The ***"curse" of*** pattern classifier ***designer***

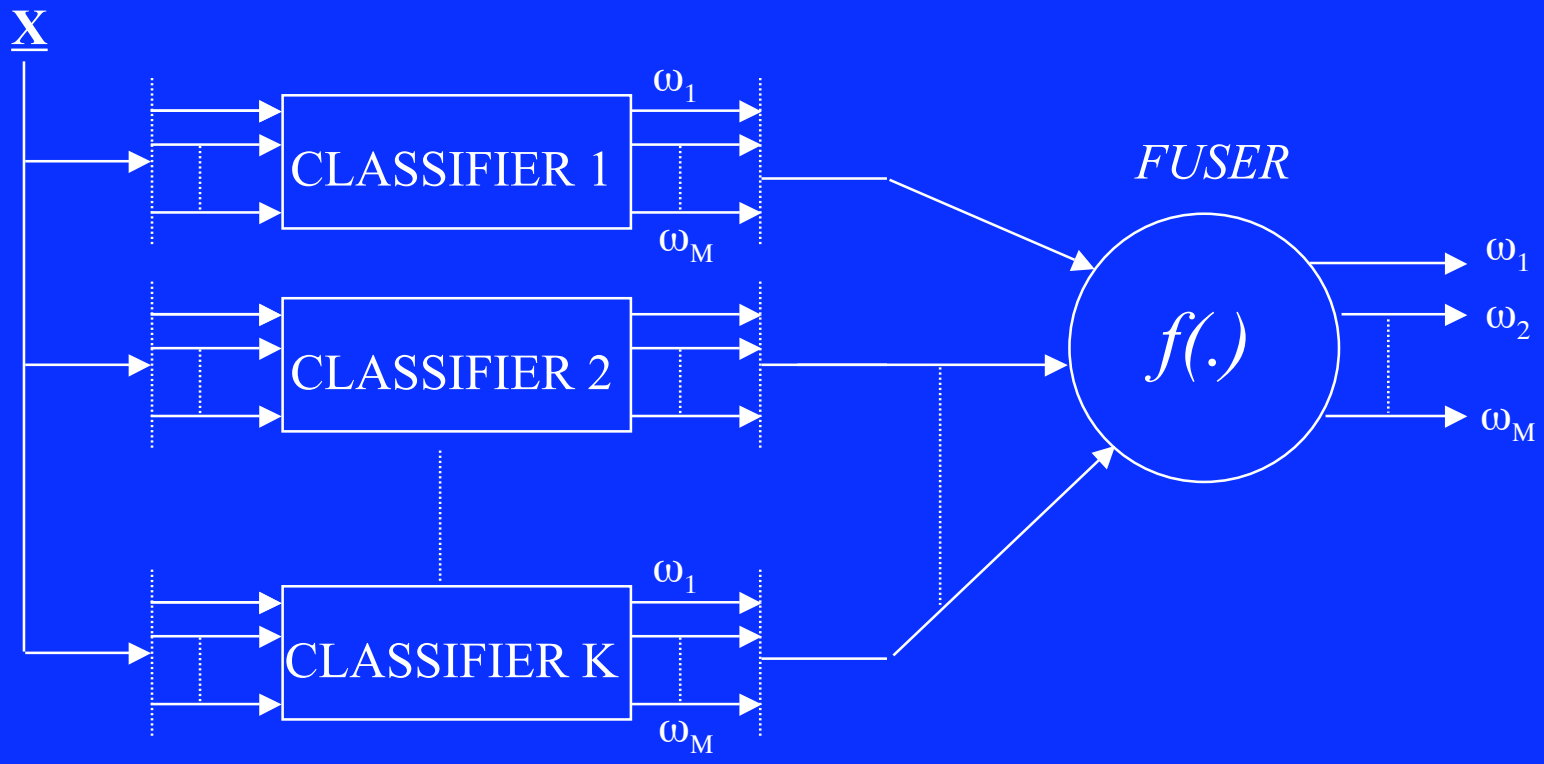
  - The need of avoiding having to make a meaningful choice of some arbitrary initial condition, such as the initial weights for a neural network

  - The intrinsic difficulty of choosing appropriate design parameters

  - "Saturation" of classifier's design *(F. Roli, Pattern Rec. Letters, 2000)*

- **Monolithic** vs. **Modular** classifier systems: different classifiers can have different domains of competence

Basic Architecture of a Multiple Classifier System



Basically, Multiple Classifier System (MCS) consists of an ensemble of different classification algorithms and a "function" f(.) to "fuse" classifiers outputs. The parallel architecture is very natural !

MCS: Basic Concepts

MCS can be characterized by:

- The **Architecture/Topology**
  - The classifier **Ensemble**: type and number of ***base*** classifiers. The ensemble can be subdivided into subsets in the case of non parallel architectures
  - The **Fuser**

MCS Architectures/Topologies

•**Parallel** topology: multiple classifiers operate in parallel. A single combination function merges the outputs of the individual classifiers

•**Serial/Conditional** topology

-Classifiers are applied in succession, with each classifier producing a reduced set of possible classes

-A primary classifier can be used. When it rejects a pattern, a secondary classifier is used, and so on

•**Hybrid** topologies

The Ensemble

•The most common type of MCS, widely used and investigated, includes an ensemble of classifiers, named "base" classifiers, and a function for parallel combination of classifier outputs

•The base classifiers are often algorithms of the same type (e.g., decision trees or neural networks), and statistical classifiers are the most common choice.

•The use of hybrid ensembles containing different types of algorithms has been investigated much less, as well as ensembles of structural, graph-based, classifiers have not attracted much attention, although they could be important for some real applications.

Fuser ("combination" rule)

Two main categories of fuser:

Integration (fusion) functions: for each pattern, all the classifiers contribute to the final decision. Integration assumes **competitive** classifiers

Selection functions: for each pattern, just one classifier, or a subset, is responsible for the final decision. Selection assumes **complementary** classifiers

- Integration and Selection can be "merged" for designing hybrid fuser
- *Multiple* functions for non parallel architecture can be necessary

Focus on Parallel Architecture

- So far research on MCS focused on parallel architectures
- Accordingly, general methodologies and clear foundations are mostly available for parallel architectures
- MCSs based on other architectures (serial, hierarchical, hybrid, etc) were highly specific to the particular application
- In the following, we focus on parallel architectures and briefly discuss the relation between classifier ensemble and combination function. Many of the concepts we discuss also hold for different architectures

Classifiers "Diversity" vs. Fuser Complexity

- *Fusion is obviously useful only if the combined classifiers are not the same classifier…*
  - Intuition: classifiers with high accuracy and high "diversity"

➢ The required degree of error *diversity* depends on the fuser complexity

- Majority vote fuser: the majority should be always correct
- Ideal selector ("oracle"): only one classifier should be correct for each pattern

An example, four diversity levels (A. Sharkey, 1999)

Level 1: no more than one classifier is wrong for each pattern
Level 2: the majority is always correct
Level 3: at least one classifier is correct for each pattern
Level 4: all classifiers are wrong for some patterns

Classifiers Diversity Measures: An Example

- Various measures (classifier outputs correlation, Partridge's diversity measures, Giacinto and Roli compound diversity, etc.) can be used to assess how similar two classifier are.

L. Kuncheva (2000) proposed the use of Q statistics:

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}}$$

Q varies between –1 and 1. Classifiers that tend to classify the same patterns correctly will have values of Q close to 1, and those which commit errors on different patterns will render Q negative

Classifiers' diversity is an elusive concept..

[Kuncheva, 03]

- Measures of diversity in classifier ensembles are a matter of on-going research (L.I. Kuncheva book, 2005)

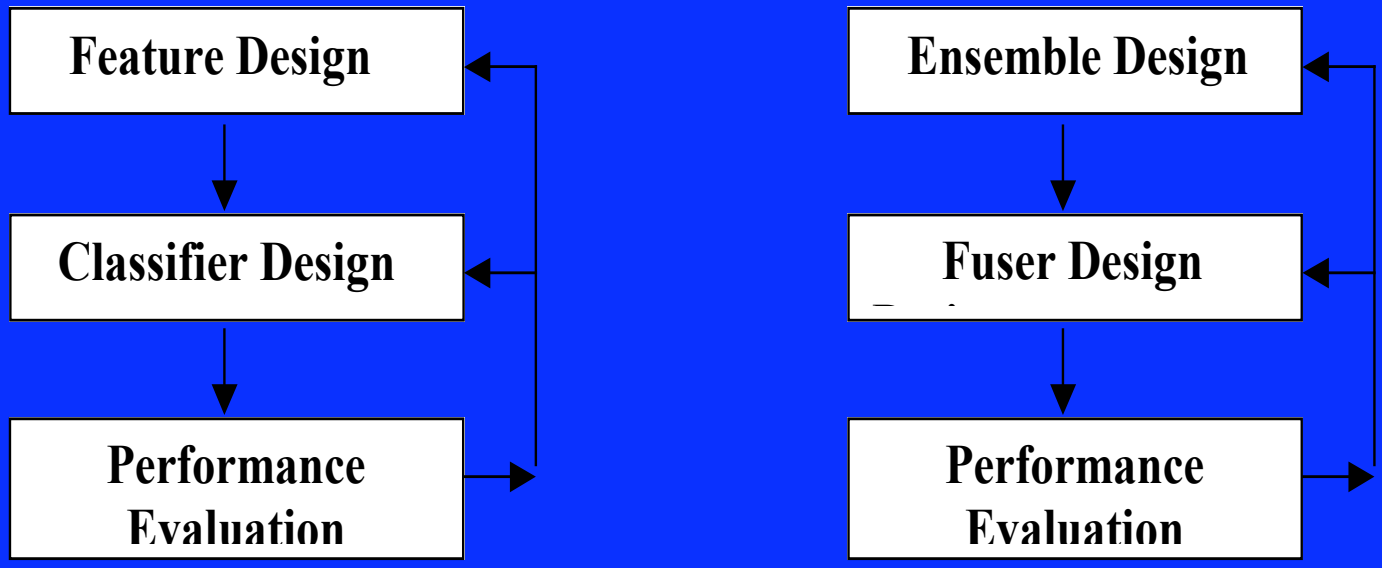
- Key issue: how are the diversity measures related to the accuracy of the ensemble ?

  - Simple fusers can be used for classifiers that exhibit a simple complementary pattern (e.g., majority voting)

  - Complex fusers, for example, a dynamic selector, are necessary for classifiers with a complex dependency model

  - *The required "complexity" of the fuser depends on the degree of classifiers diversity*

Analogy between MCS and Single Classifier Design



Design cycles of single classifier and MCS (Roli and Giacinto, 2002)

Two main methods for MCS design (T.K. Ho, 2000):

- Coverage optimization methods
- Decision optimization methods

MCS Design

- The design of MCS involves two main phases: the design of the classifier ensemble, and the design of the fuser
- The design of the classifier ensemble is aimed to create a set of "complementary/diverse" classifiers
- The design of the combination function/fuser is aimed to create a fusion mechanism that can exploit the complementarity/diversity of classifiers and optimally combine them
- The two above design phases are obviously linked (Roli and Giacinto, Design methods for MCS, Book Chapter, 2002)
- In the following (Parts II and III), we illustrate the main methods for constructing and fusing multiple classifiers

A small homework…

Using any tool in your hands (e.g., Google), do a search and let me know which is the oldest paper dealing with the topic of "multiple classifier systems" that you are able to find, and explain me shortly how you did your search.

Send me via mail (roli@diee.unica.it) the result of your search. Thanks!

Question…

Using independent classifiers is always better than using dependent classifiers for combination purposes?

Send me via mail (roli@diee.unica.it) your justified answer

Mini Tutorial

Three Hours on Multiple Classifier Systems

PART II

PART II

Methods for creating classifier ensembles

Methods for creating MCS

- The effectiveness of MCS depends both on the base classifiers and the combination function

Several approaches have been proposed to create classifiers which should be "good" for combination. Among the others:

- Using problem and designer knowledge
- Injecting randomness
- Varying the classifier type, architecture, or parameters
- Manipulating training data
- Manipulating input features
- Manipulating output features

Using problem and designer knowledge

- When problem or designer knowledge is available, "complementary" classification algorithms can be designed quite naturally

  - In applications with multiple sensors

  - In applications where complementary representations of patterns are possible (e.g., statistical and structural representations)

  - When designer knowledge allows varying the classifier type, architecture, or parameters to create complementary classifiers

*These are **heuristic** approaches, perform as well as the problem/designer knowledge allows to design "complementary" classifiers which can be combined effectively*

Injecting randomness

- Simple design methods are based on injecting randomness in the classification/training algorithm

  –Neural Networks: the back-propagation algorithm is often run several times using different (random) starting points (initial weights)

  –Decision Trees: the test at each internal node can be chosen randomly between the top *n* best tests

  –Random Forests (Leo Breiman, 2001)

These are basically heuristic approaches. We can only hope that they produce complementary classifiers. However, we have many experimental evidences which support this conjecture.

Methods based on training data manipulation

- These methods are based on training *N* classifiers with *N* different training sets

Data splitting

–Training data are randomly subdivided into *N* disjoint subsets

–Each classifier is trained on a different subset (infeasible for small training sets)

Cross-validated committees

–Training data are randomly subdivided into *N* disjoint subsets

–*N* overlapping training sets are constructed by dropping out a different one of the N subsets

➢**Bagging**

➢**Boosting**

Bagging

- Method proposed by L. Breiman (1996) for constructing multiple classifiers by training data manipulation

- Bagging is based on obtaining different training sets of *equal size* as the original one L, by using a statistical technique named *bootstrap*

- The resulting training sets L_i, i=1,…,N, contain usually small changes with respect to L

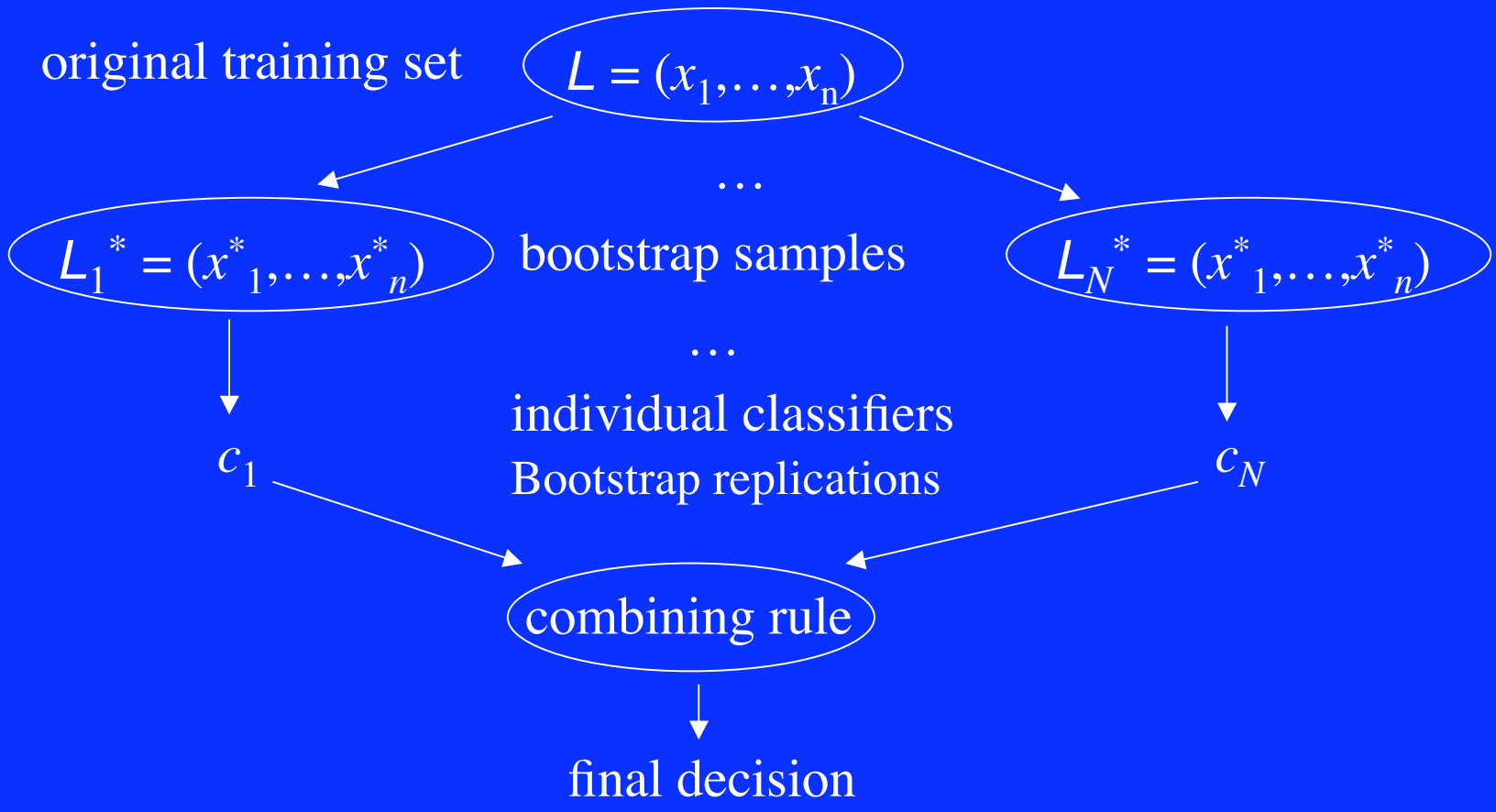
Bootstrap

- The bootstrap technique is based on the concepts of *bootstrap sample* and *bootstrap replication*

- Bootstrap sample
  - $\mathbf{x}^* = (x^*_1,\ldots,x^*_n)$: random sample of size n drawn *with replacement* from the original sample $\mathbf{x} = (x_1,\ldots,x_n)$
  - each sample in $\mathbf{x}$ can appear in $\mathbf{x}^*$ zero times, once, twice, etc.

- Bootstrap replication
  - a classifier trained with a bootstrap sample

Bagging (Bootstrap AGGregatING)



- Rationale behind:
  - –Qualitative: instances of an *unstable* classifier constructed on different bootstrap samples can exhibit significant differences
  - –Quantitative: variance reduction [Serrau et al., IEEE-T PAMI 2008]

Combining rules for Bagging

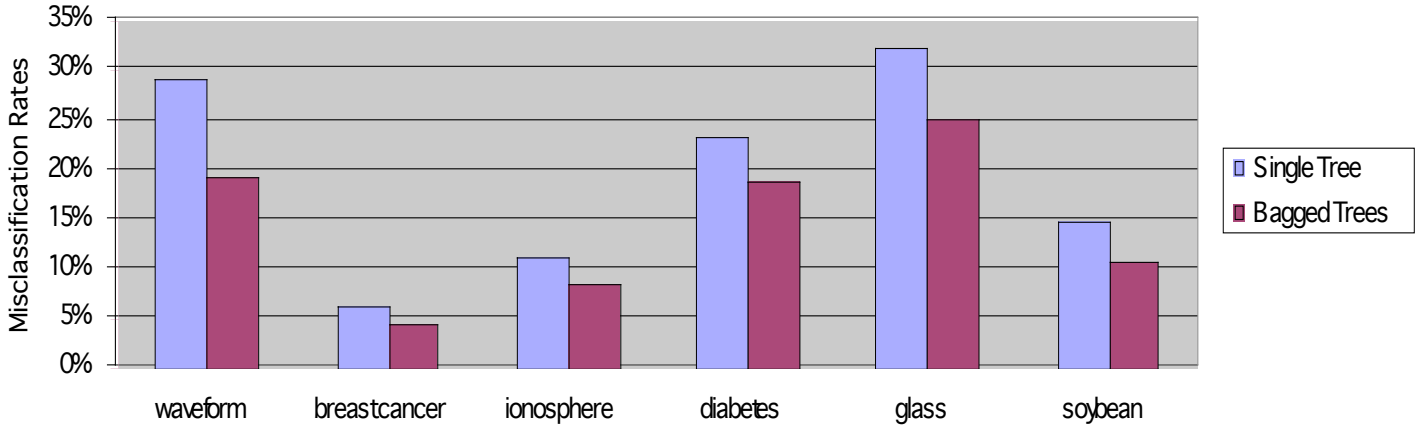
- Bagging is a method for *constructing* multiple classifiers, not a fusion rule

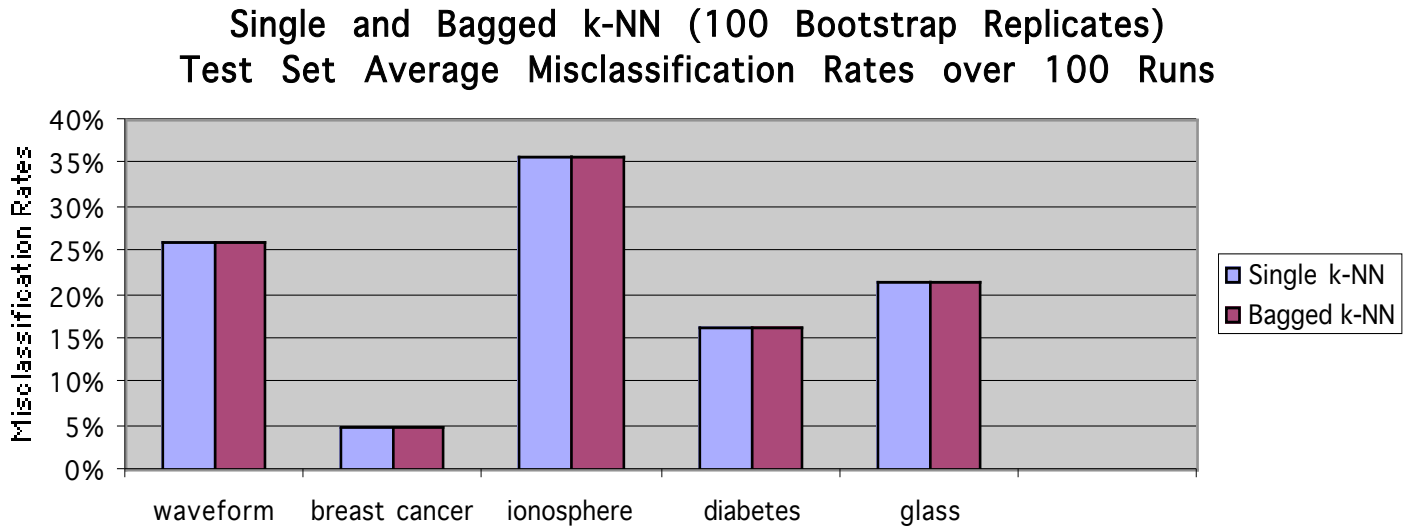
- In principle, any combining technique can be applied

- Usually, simple combining rules are used
  - simple averaging
  - majority vote

- Experimental results show that bagging is effective when used with simple combining rules.
- The optimality of the simple average rule has been also proved theoretically [Serrau et al, 08]

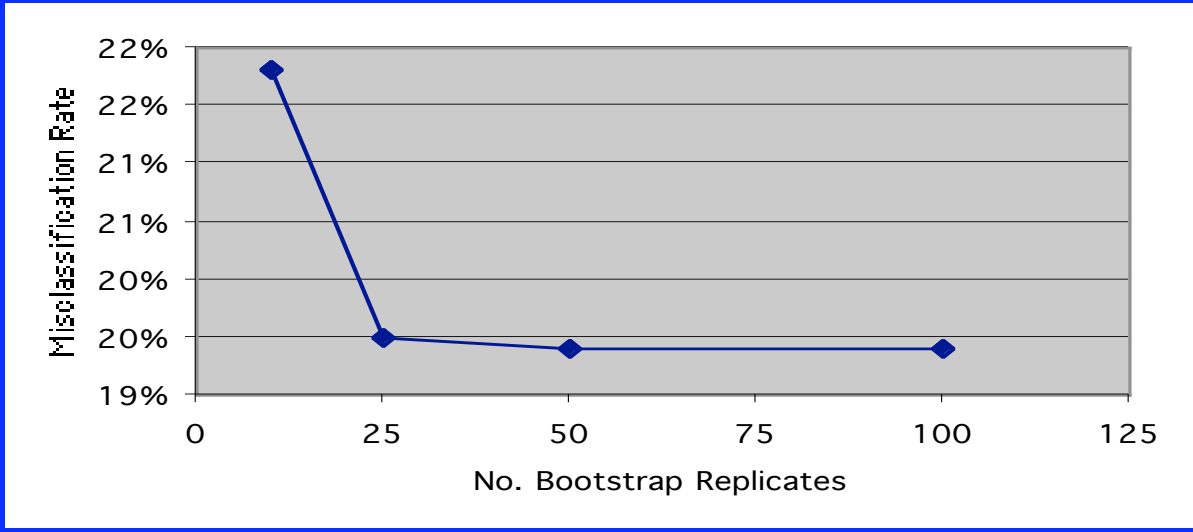
Examples of bagging (Breiman, 1996)





The right number of bagged classifiers

- How many bagged classifiers are enough?
  - Experimental results show that 50 bootstrap samples are often sufficient for classification problems
  - Example for the *soybean* data set (Breiman, 1996):



➢Stopping Bagging, namely, determining the sufficient number of bagged classifiers is a crucial issue for real applications with strict constraints on memory size and CPU time.

The right number of bagged classifiers

Recently, Fumera, Roli, and Serrau *(IEEE-T on PAMI, July 2008)* proved that the average error rate of *m* bagged classifiers can be modelled as:

$$E_\infty + \frac{1}{m}\left[E_1 - E_\infty\right]$$

Asymptotic error

Error using just one bagged classifier

➢This theoretical result says us that combining *m* bagged classifiers one can expect on average to reach a fraction *(m-1)/m* of the maximum error reduction achievable with an infinite number of classifiers (asymptotic error of Bagging)

✓With m=10 the error reduction is already 90%

✓This model fits well with results of Breiman and other researchers

AdaBoost

- AdaBoost algorithm (Freund and Schapire, 1995) is aimed at producing highly accurate ("strong") classifiers by combining "weak" instances of a given base classifier

- AdaBoost *iteratively* constructs an ensemble of *N* complementary classifiers

- Additional weak classifiers are introduced iteratively if necessary, and they are trained on samples that previous classifiers have misclassified

- The resulting classifiers are combined by *weighted* voting

➢ AdaBoost is an ensemble learning method, not a general purpose method for constructing multiple classifiers like Bagging

Basic Scheme of AdaBoost

Given a set $L = (x_1,\ldots,x_n)$ of n training patterns

Initialize $D_1(i) = 1/n$, $i=1,\ldots,n$; $L_1 = L$

– $D_t(i)$ denotes the weight of pattern x_i on round t

For $t=1,\ldots,N$:

– Train the base classifier c_t on L_t

– Compute the error rate ε_t of c_t on the original training set L

– Set $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

– Update $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } x_i \text{ is correctly classified} \\ e^{\alpha_t} & \text{if } x_i \text{ is misclassified} \end{cases}$

Combine the N classifiers by weighted majority voting, using the weights α_t

Methods based on Input Feature Manipulation

- Manual or automatic feature selection/extraction can be used for generating diverse classifiers using different feature sets

•For example, subsets related to different sensors, or subsets of features computed with different algorithms

•Different feature sets can be generated using different feature extraction algorithms applied to the original set

•Manual or automatic selection can work with set of redundant/irrelevant features

•The "hope" is that classifiers using different features are complementary

The Random Subspace Method

The Random Subspace Method (RSM) consists in random selection of a certain number of subspaces from the original feature space, and train a classifier on each subspace *(T.K. Ho, IEEE-T on PAMI, 1998)*.

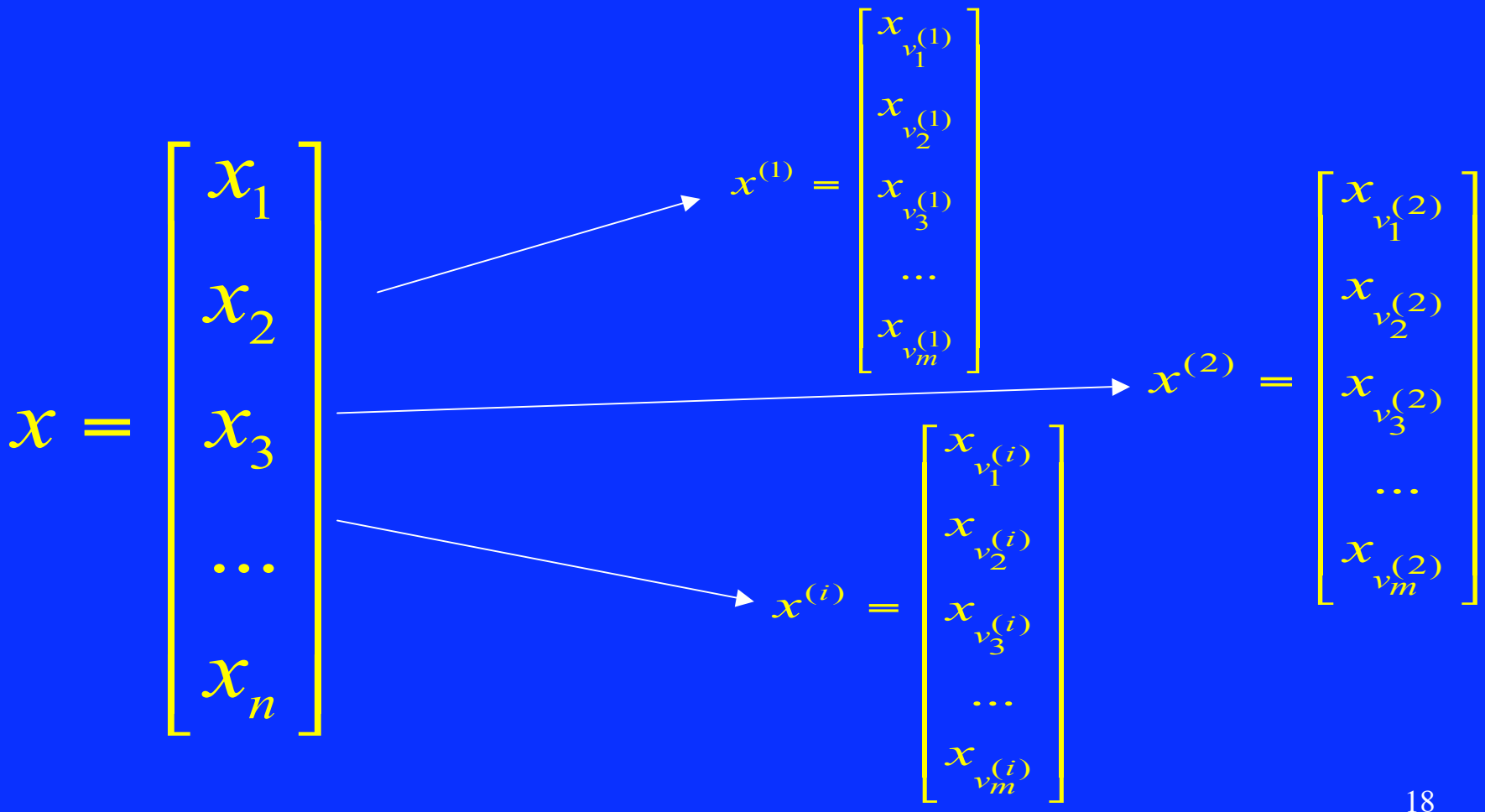
Let $X \subseteq \Re^n$ be a *n*-dimensional feature space.

$$x = [x_1, x_2, x_3, \ldots, x_i, \ldots, x_{n-2}, x_{n-1}, x_n]$$

We can project this vector into a *m*-dimensional subspace, by selecting *m* random components.

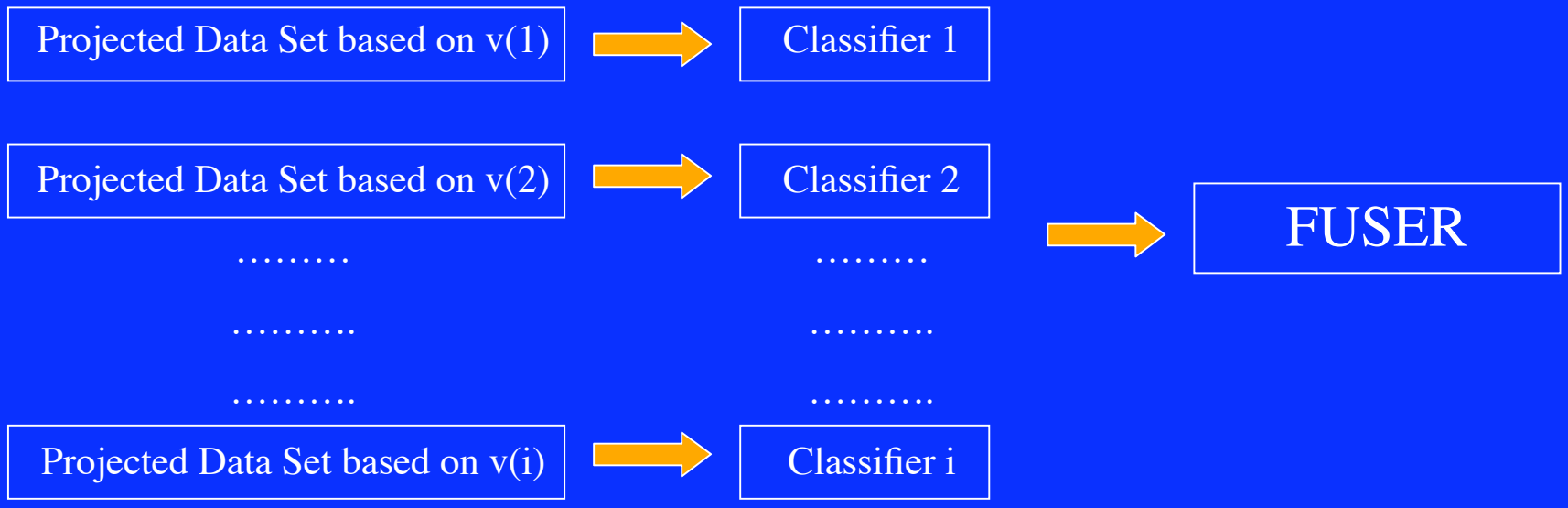
RSM: multiple subspace generation

We can generate multiple "projected" data sets, by varying the vector *v*.



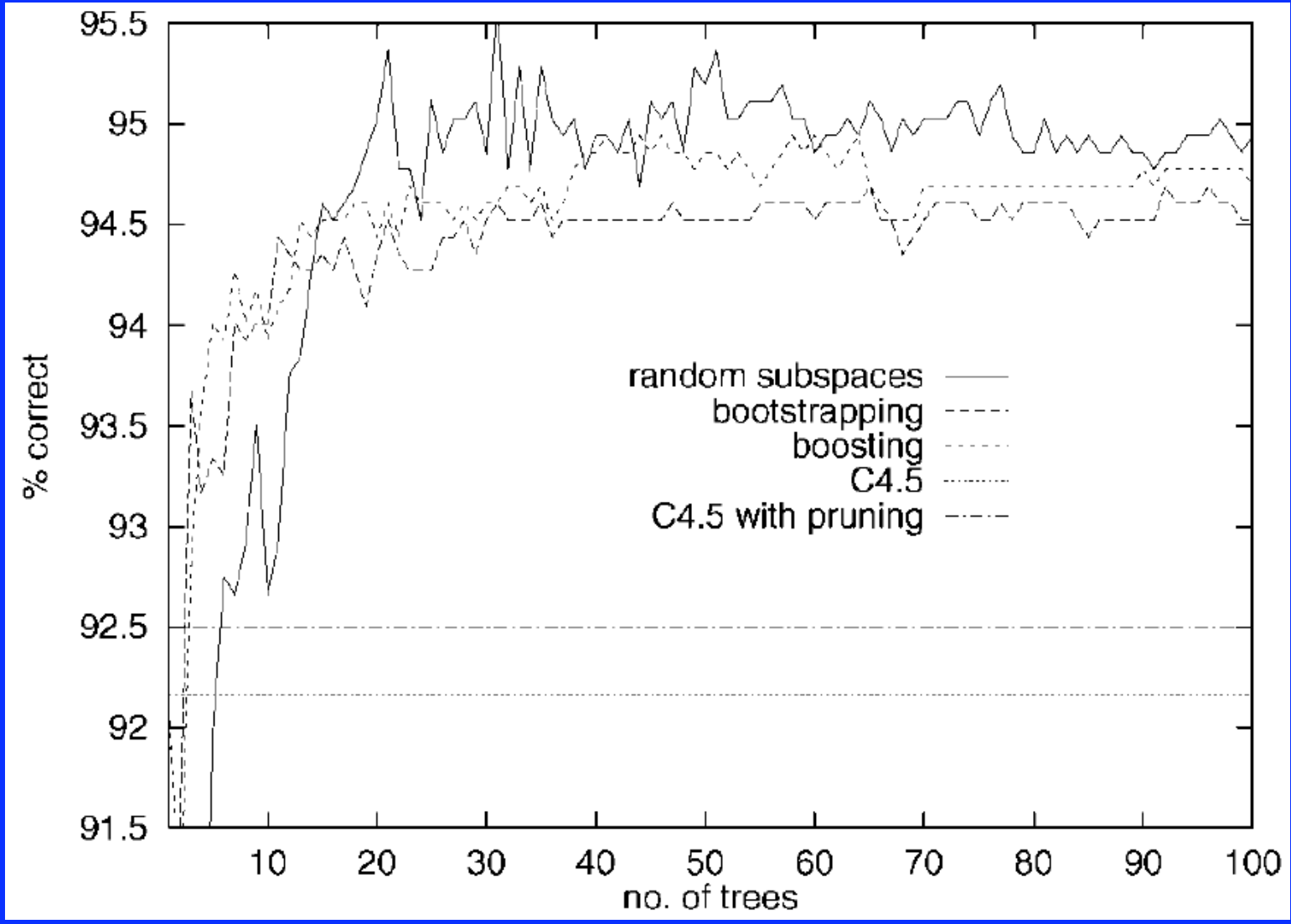
Decision fusion with RSM

The next step is to combine the information extracted by each classifier trained on the feature subspace



Experiments showed that simple combiners (e.g., average of classifiers outputs) work well with RSM generated classifiers.

RSM: Application to Decision Forests



Some Remarks on RSM

RSM works well for large feature sets with redundant features

In some sense, this approach does not suffer from the "curse" of dimensionality.

Key issue: the number of ramdom features to generate

Random Subspace Method exploits concepts of the theory of stochastic discrimination by E. Kleinberg.

See L.I. Kuncheva, F. Roli, G.L. Marcialis and C.A. Shipp, "Complexity of Data Subsets Generated by the Random Subspace Method: an Experimental Investigation", MCS 2002.

The concept of "weak" classifier

- Some methods (Bagging, Boosting, RSM) use "weak" classifiers

- Why should we use "weak" classifiers if we can design strong ones ?

- Because designing a strong classifier by fusion of multiple weak classifiers can be simpler (the "curse" of designer)

- Because weak classifiers, with low "variance", can suffer less small sample size issues

Noise Injection

Injecting noise into the input features can be used to manipulate the training data, so creating different training sets.

For example, we can add a zero mean and small covariance noise vector *n* to each training vector X:

$$X^{new} = X + n$$

It is possible to generate ***m*** *artificial* vectors for each training pattern.

Raviv and Intrator (1996) combined bootstrap sampling of the training data with injecting noise. The **x** value of each training example was perturbed by adding Gaussian noise

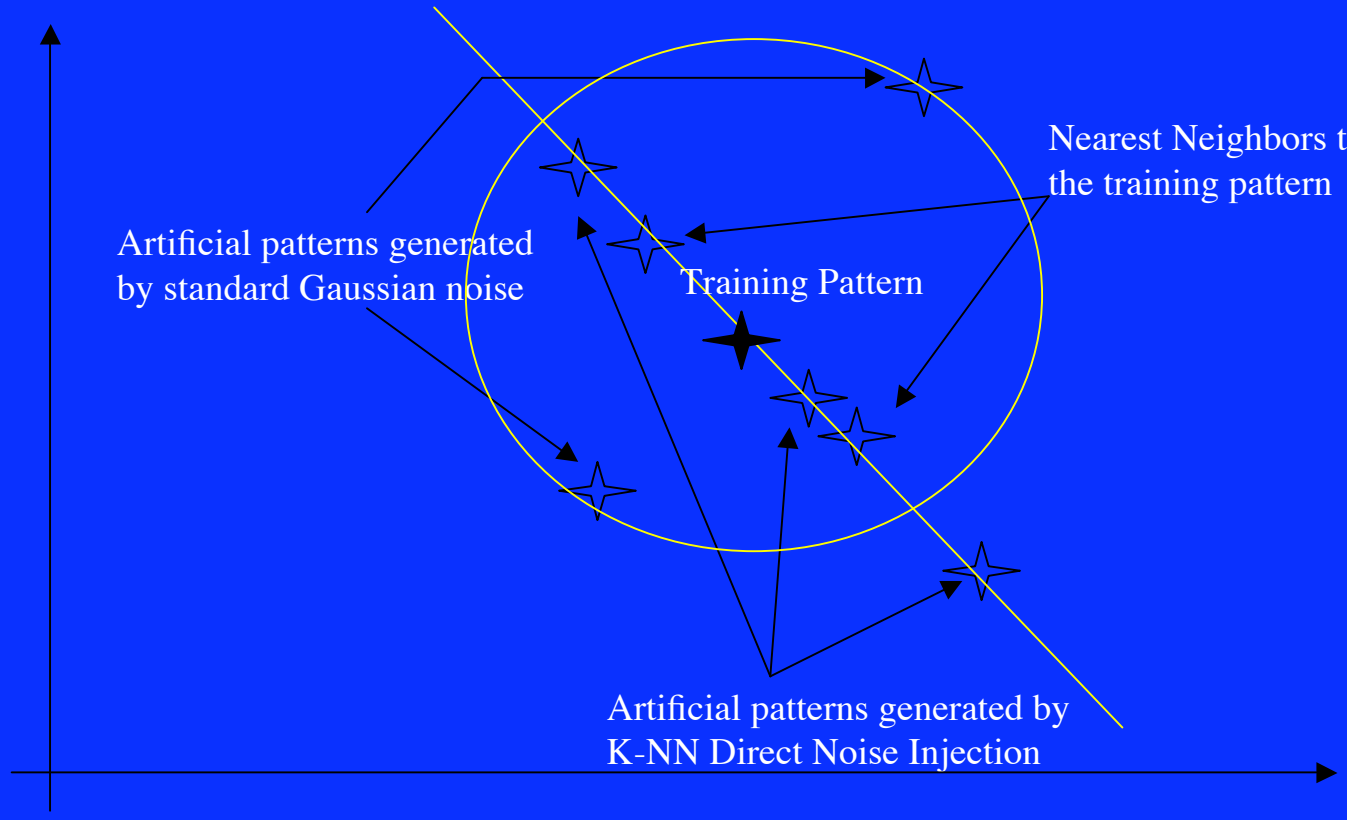
Other possibility: data splitting + adding noise

The K-NN Direct Noise Injection

In order to take in account the *intrinsic dimensionality* of the data, we can add noise along the direction of the K nearest neighbors of each pattern.

M.Skurichina et al., 2000

F.Roli, S.Raudys, G. Marcialis, MCS 2002



Manipulating the Output Features

Another interesting idea is building complementary classifiers by partitioning the set of classes in different ways

Each component classifier is trained to solve a subset of the *N* class problem. For instance, each classifier could solve a two class problem **(e.g., One vs. All strategy)**.

A suitable combination method able to "recover" the original *N* class problem is necessary.

To this end, Dietterich and Bakiri described a technique called Error-Correcting Output Coding (ECOC)

ECOC works well for a large number of classes. But it could be applied to subclasses within a smaller number of classes

ECOC: Basic Idea

Let $X \subseteq \Re^n$ be a n-dimensional input space.

Let $\{c_1,\ldots,c_k\}$ be a set of classes.

Let $\{f_0,\ldots,f_{m-1}\}$ be a set of m functions, with $f_i : X \rightarrow \{0,1\}$

For each class c_j, let $\boldsymbol{b}^{(j)}=\{b_0,\ldots,b_{m-1}\}$ be the associated "codeword", with

$$f_i = b_i \in \{0,1\}$$

We construct a *decoding matrix* whose rows are the classes c_j and columns are the bit b_i of the codeword associated to each class.

ECOC: An example of Decoding Matrix

A 15-bit ECOC for a ten-class problem:

Class	Code Word														
	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

ECOC classification

In the previous example, a separate boolean function f_i is learned (e.g. trough a MLP or a DT) for each bit position of the error-correcting code.

To classify a new example $x \in X$, each of the learned functions $f(x) = \{f_0(x),\ldots,f_{14}(x)\}$ is evaluated to produce a 15-bit string.

This is then mapped to the nearest of the ten codewords, according to a "distance measure" (e.g., the Hamming distance):

$$class = \arg\min_k d(b^{(k)}, f(x))$$

Question…

Why should Bagging work?

Send me via mail (roli@diee.unica.it) your justified answer

Mini Tutorial

Three Hours on Multiple Classifier Systems

PART III

PART III

Methods for combining multiple classifiers

Methods for fusing multiple classifiers

Methods for fusing multiple classifiers can be classified according to the type of information produced by the individual classifiers (Xu et al., IEEE-T on SMC, 1992):

Abstract-level outputs: each classifier outputs a unique class label for each input pattern

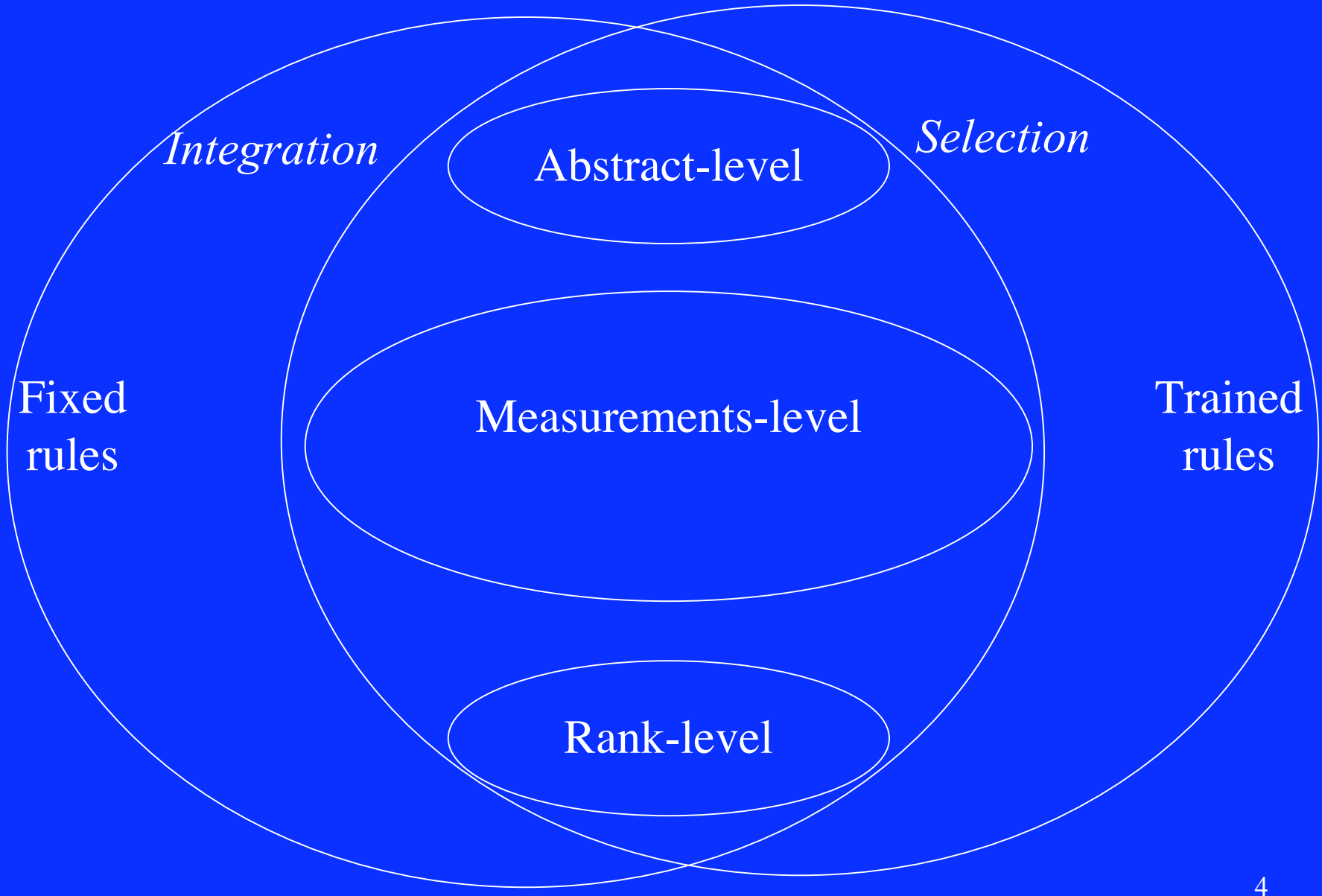
Rank-level outputs: each classifier outputs a list of possible classes, with ranking, for each input pattern

Measurement-level outputs: each classifier outputs class "confidence" levels for each input pattern

For each of the above categories, methods can be further subdivided into:

Integration vs. **Selection rules** and **Fixed** rules vs. **Trained** Rules

Methods for fusing multiple classifiers

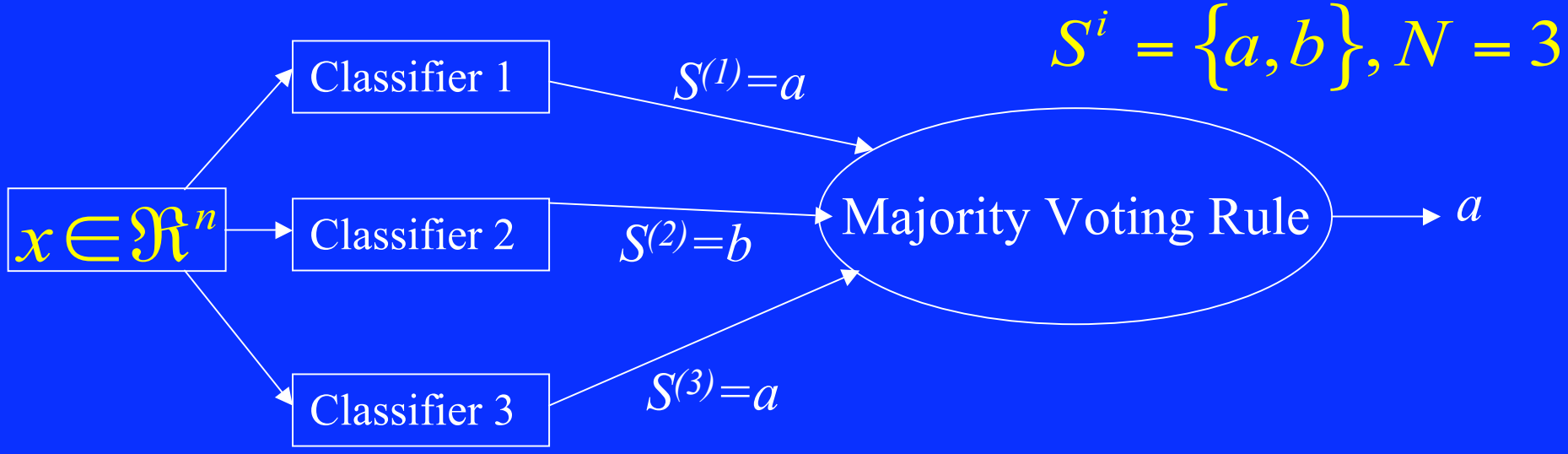


Fixed Rules at the abstract-level

The Majority Voting Rule

Let us consider the N abstract ("crisp") classifiers outputs $S^{(1)}, \ldots, S^{(N)}$ associated to the pattern x.

Majority Rule: *Class label* $\mathbf{c_i}$ *is assigned to the pattern* x *if* $\mathbf{c_i}$ *is the most frequent label in the crisp classifiers outputs.*



Majority Vote Rule

Usually N is odd.

The frequency of the winner class must be at least $\lceil N/2 \rceil$

If the N classifiers make ***independent*** errors and they have the same error probability $e<0.5$, *then* it can be shown that the error E of the majority voting rule is monotonically decreasing in N (Hansen and Salamon, IEEE-T on PAMI, 1990):

$$\lim_{N \to \infty} \sum_{k > N/2}^{N} \binom{N}{k} e^k (1-e)^{N-k} = 0$$

- Clearly, performances of majority vote quickly decreases for **dependent** classifiers

Majority Vote Rule vs. Classifiers Dependency: An Example

- 3 Classifiers
- Two class task

Classifiers C1 and C3

exhibit error correlations

$$P(1 \mid 0,1,0) \underset{<}{\overset{>}{-}} P(0 \mid 0,1,0)$$

Classifier C1	Abstract C2	Outputs C3	Majority Class	True Class
0	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

Rules based on the Bayes Approach

This kind of trained fusion rules are based on the Bayes approach.

Pattern x is assigned to the class c_i if its posterior probability:

$$P(c_i \mid S^{(1)}, S^{(2)}, \ldots, S^{(N)}) > P(c_j \mid S^{(1)}, S^{(2)}, \ldots, S^{(N)}) \; \forall j \neq i$$

$$S^i \in \{c_1, c_2, \ldots, c_k\}$$

Fusion rules based on the Bayes Formula try to estimate, by an ***independent*** validation set, these *posterior probabilities*

- This fusion rule coincides with the multinomial statistical classifier, that is, the optimal statistical decision rule for discrete-valued feature vectors (Raudys and Roli, MCS 2003)

Behaviour Knowledge Space (BKS)

- In the BKS method, every possible combination of abstract-level classifiers outputs is regarded as a cell in a look-up table.
- Each cell contains the number of samples of the validation set characterized by a particular value of class labels.
- Reject option by a threshold is used to limit error due to "ambiguous" cells

Classifiers outputs $S^{(1)}$, $S^{(2)}$, $S^{(3)}$

Class	0,0,0	0,0,1	**0,1,0**	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1
0	100	50	**76**	89	54	78	87	5
1	8	88	**17**	95	20	90	95	100

$$P(c=0 \mid S^{(1)}=0, S^{(2)}=1, S^{(3)}=0) = \frac{76}{76+17} = 0.82 \geq th$$

BKS Small-Sample Size Drawback

• If k is the number of classes and N is the number of the combined classifiers, BKS requires to estimate k^N posterior probabilities.

- K and N are two critical parameter for the BKS rule, because *the number of posterior probabilities to estimate increases very quickly.*
- If this number is too high, we can have serious problems, because the number of training sample is often small.

BKS rule suffers a lot from the small sample size problem

BKS Improvements

•In order to avoid the small sample size problem:

-we can try to reduce the number of the parameters to estimate (Xu et al., 1992 ; Kang, and Lee, 1999).

For example, under the assumption of classifier independence given the class (Xu et al., 1992):

$$P(S^{(1)},\ldots,S^{(N)} \mid c_i) = \prod_j P(S^{(j)} \mid c_i) \propto \prod_j P(c_i \mid S^{(j)}) \cdot P(S^{(j)})$$

$$bel(i) = P(c_i \mid S^{(1)},\ldots,S^{(N)}) = \eta \prod_j P(c_i \mid S^{(j)})$$

-We can use noise injection to increase sample size of training set (Roli, Raudys, Marcialis, MCS 2002).

BKS Improvements

•If the number of classifiers is small, BKS cells can become "large" and contain vectors of different classes, that is, *ambiguous* cells can exist.

Classifiers outputs $S^{(1)}$, $S^{(2)}$, $S^{(3)}$

Class	0,0,0	0,0,1	**0,1,0**	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1
0	100	50	**51**	89	54	78	87	5
1	8	88	**50**	95	20	90	95	100

Raudys and Roli (MCS 2003) proposed a method to address this issue

Remarks on Abstract Level Fusers

- Abstract level methods are the most general fusion rules

- They can be applied to any ensemble of classifiers, even to classifiers of different types

- The majority voting rule is the simplest combining method
  - This allows theoretical analyses (Lam and Suen, 1997)

- When prior performance is not considered, the requirements of time and memory are negligible

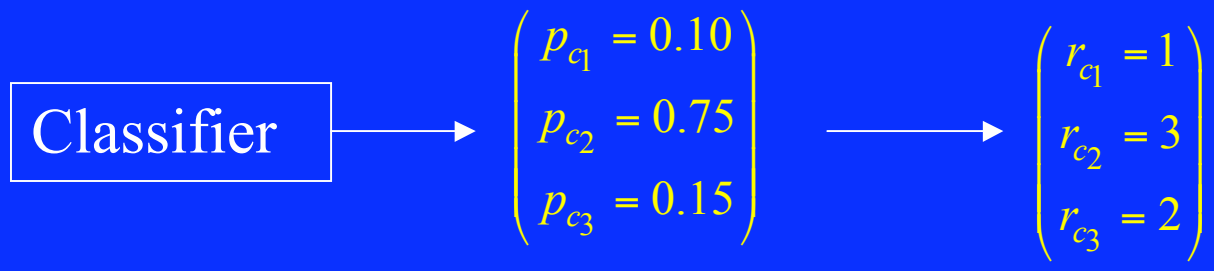
- As we proceed from simple rules to adaptive (weighted voting) and trained (BKS, Bayesian rule) the demands on time and memory quickly increase

- Trained rules impose heavy demands on the quality and size of data set

Rank-level Fusion Methods

Some classifiers provide class "scores", or some sort of class probabilities

This information can be used to "*rank*" each class.



In general, if $\Omega = \{c_1, \ldots, c_k\}$ is the set of classes, these classifiers can provide an "ordered" (ranked) list of class labels.

The Borda Count Method: an example

Let N=3 and k=4. $\Omega = \{ a, b, c, d \}$.

For a given pattern, the ranked outputs of the three classifiers are as follows:

Rank value	**Classifier 1**	**Classifier 2**	**Classifier 3**
4	*c*	*a*	*b*
3	*b*	*b*	*a*
2	*d*	*d*	*c*
1	*a*	*c*	*d*

The Borda Count Method: an example

So, we have:

$$r_a = r_a^{(1)} + r_a^{(2)} + r_a^{(3)} = 1 + 4 + 3 = 8$$

$$r_b = r_b^{(1)} + r_b^{(2)} + r_b^{(3)} = 3 + 3 + 4 = 10$$

$$r_a = r_c^{(1)} + r_c^{(2)} + r_c^{(3)} = 4 + 1 + 2 = 7$$

$$r_a = r_d^{(1)} + r_d^{(2)} + r_d^{(3)} = 2 + 2 + 1 = 5$$

The winner-class is *b* because it has the maximum overall rank.

Remarks on Rank level Methods

- Advantages over abstract level (majority vote):

–ranking is suitable in problems with many classes, where the correct class may appear often near the top of the list, although not at the top

➢Example: word recognition with sizeable lexicon

•Advantages over measurement level:

–rankings can be preferred to soft outputs to avoid lack of consistency when using different classifiers

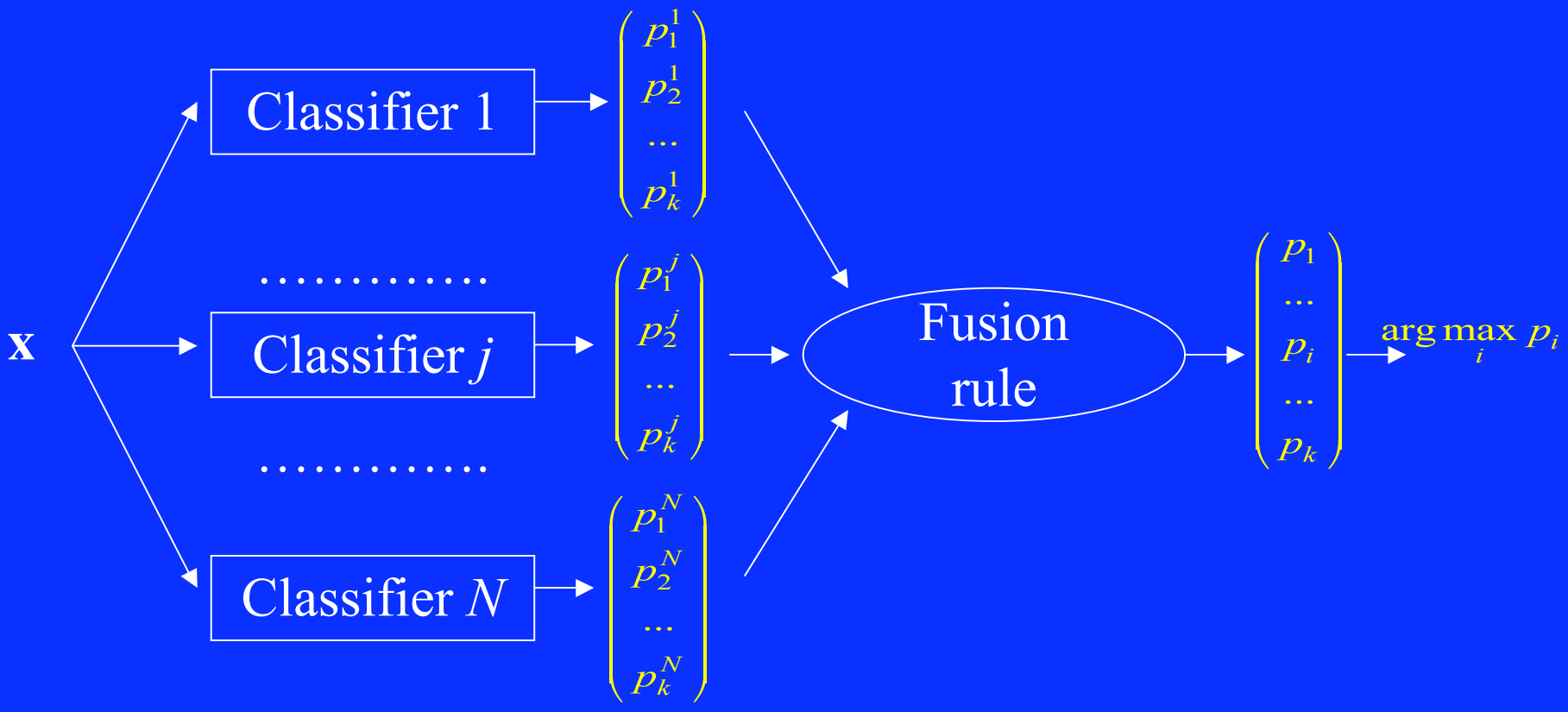
–rankings can be preferred to soft outputs to simplify the combiner design

•Drawbacks:

–Rank-level methods are not supported by clear theoretical underpinnings

–Results depend on the scale of numbers assigned to the choices

Measurement-level Fusion Methods



➢Normalization of classifiers outputs is not a trivial task when combining classifiers with different output ranges and different output types (e.g., distances vs. membership values).

Linear Combiners

$$p_i^{ave}(x) = \sum_{k=1}^{N} w_k p_i^k(x)$$

- Simple and Weighted averaging of classifiers' outputs

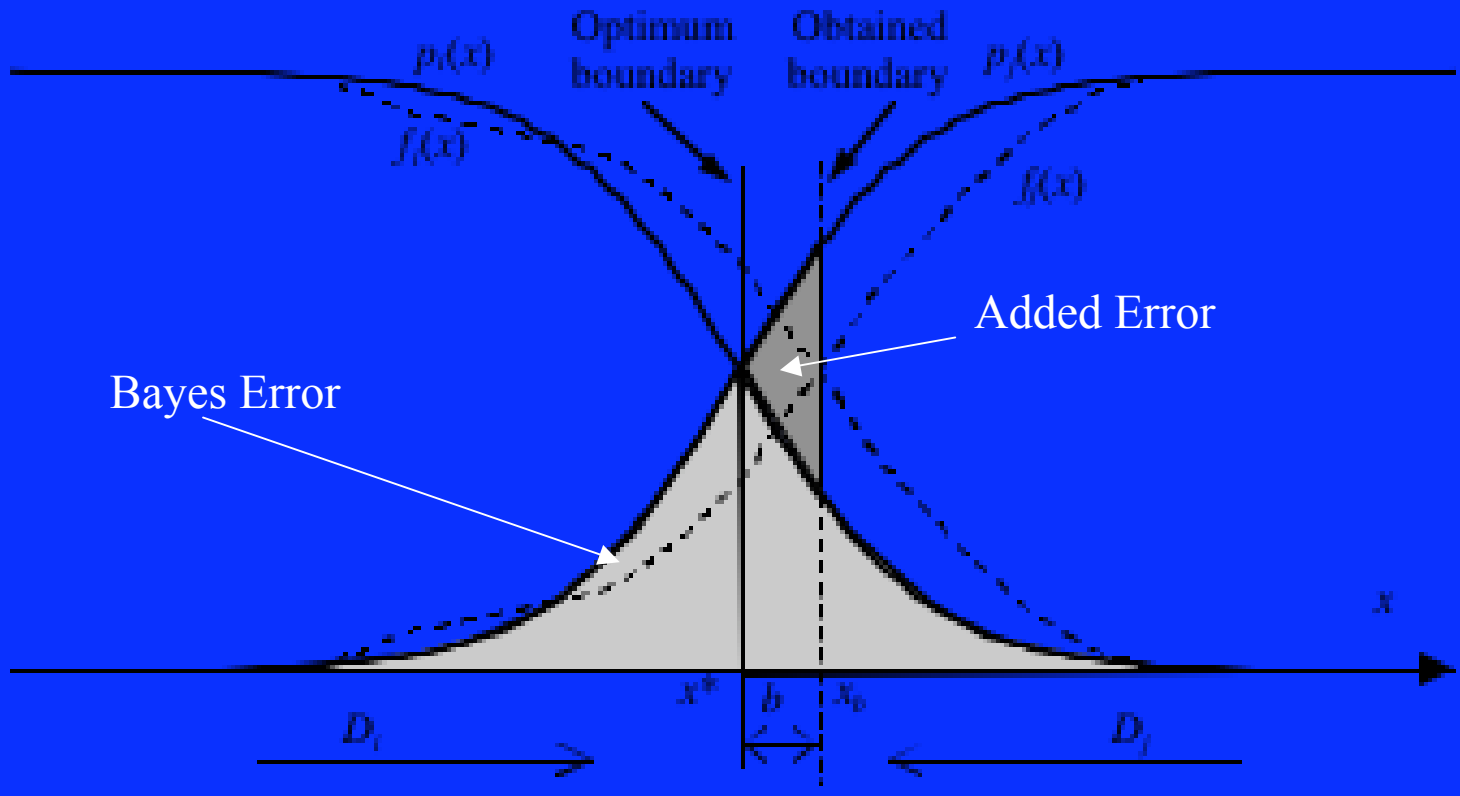
- Simple average is the optimal fuser for classifiers with the same accuracy and the same pair-wise correlations (Fumera and Roli, IEEE-T on PAMI, 2005)

- Weighted average is required for *imbalanced* classifiers, that is, classifiers with different accuracy and/or different pair-wise correlations (Fumera and Roli, 2005)

- Improvement of weighted average over simple average has been investigated theoretically and by experiments (Roli and Fumera)

Bias/Variance Analysis in Linear Combiners

- The "added" error over the Bayes one can be reduced by linear combinations of classifiers outputs (Tumer and Ghosh; Roli and Fumera)



Bias/Variance Analysis in Linear Combiners

Correlated and Unbiased Classifiers

- Added error of the simple average of *N* classifiers (Tumer and Ghosh):

$$E_{add}^{ave} = E_{add}\left(\frac{1+(N-1)\delta}{N}\right)$$

- The reduction of variance component of the added error achieved by simple averaging depends on the correlation factor δ between the estimation errors

➢ Negatively correlated estimation errors allow to achieve a greater improvement than independent errors

Bias/Variance Analysis in Linear Combiners

Correlated and Biased Classifiers

- Added error of the simple average of *N* classifiers (Tumer and Ghosh):

$$E_{add}^{ave} = \frac{1}{s}\sigma^2\left(\frac{1+(N-1)\delta}{N}\right) + \frac{1}{2s}\left(\bar{\beta}_i - \bar{\beta}_j\right)^2$$

➢ *Simple averaging is effective for reducing the variance component, but not for the bias component*

➢ *So, individual classifiers with low biases should be preferred*

Analysis of bias/variance for weighted average can be found in (Fumera and Roli, IEEE-T on PAMI, 2005)

Product and Order Statistics Fusers

$$p_i = P^{-(N-1)}(\omega_i)\prod_{j=1}^{N} p_i^j, \quad i = 1,\ldots,k$$

–Product is obviously sensible to classifiers outputting probability estimates close to zero ("overconfident" classifiers)

Fusers based on order statistics operators are *max*, *min*, and *med*:

$$p_i^{\max}(\mathbf{x}) = p_i^{N:N}(\mathbf{x}), \quad i = 1,\ldots,k$$

$$p_i^{\min}(\mathbf{x}) = p_i^{1:N}(\mathbf{x}), \quad i = 1,\ldots,k$$

$$p_i^{\text{med}}(\mathbf{x}) = \begin{cases} \dfrac{p_i^{\frac{N}{2}:N}(\mathbf{x}) + p_i^{\frac{N+1}{2}:N}(\mathbf{x})}{2} & \text{if } N \text{ is even} \\ p_i^{\frac{N+1}{2}:N}(\mathbf{x}) & \text{if } N \text{ is odd} \end{cases}, \quad i = 1,\ldots,k$$

A Theoretical Framework

- A theoretical framework for the *product*, *min*, *med* and *max* fusers was established by J. Kittler et al. (1998)

•These rules can be formally derived assuming that the *N* individual classifiers use ***distinct*** feature vectors

$$p^j(\omega_i / X_1, X_2, \ldots\ldots, X_N)$$

–the product and min rules are derived under the hypothesis that classifiers are ***conditionally statistically independent***

–sum, max, median rules are derived under the further hypothesis that the ***a posteriori probabilities*** estimated by the classifiers ***do not deviate significantly from the class prior probabilities***

Fusers with Weights

A natural extension of many fixed rules is the introduction of weights to the outputs of classifiers (weighted average, weighted majority vote)

A simple criterion is to introduce weights proportional to the accuracy of each individual classifier

Weights related to classes can also be computed by extracting them from the confusion matrix of each classifier

- Methods for robust weights estimation are a matter of on-going research

"Stacked" Fusion

The k soft outputs of the N individual classifiers, $p_i^j(\mathbf{x})$, $i=1,\ldots,k$, $j=1,\ldots,N$, can be considered as features of a new classification problem (*classifier-output* feature space)

Classifiers can be regarded as feature extractors !

Another classifier can be used as fuser: this is the so-called "stacked" approach (D.H. Wolpert, 1992), or "meta-classification" (Giacinto and Roli, 1997), or "brute-force" approach (L.I. Kuncheva, 2000)

•To train the metaclassifier, the outputs of the N individual classifiers on an **independent** validation set must be used

- Experts' Boasting Issue ! (Raudys, IEEE-T on PAMI, 2003)
- An **independent** validation set is required

Pros and Cons of the "Stacked" approach

Pros:

•the meta-classifier can work in a "enriched" feature space

•No classifiers dependency model is assumed

Cons:

•The dimensionality of the output space increases very fast with the number of classes and classifiers.

•The meta-classifier should be trained with a data set different from the one used for the individual classifiers (Experts' Boasting Issue)

•***The space of classifiers outputs might not be well-behaved***

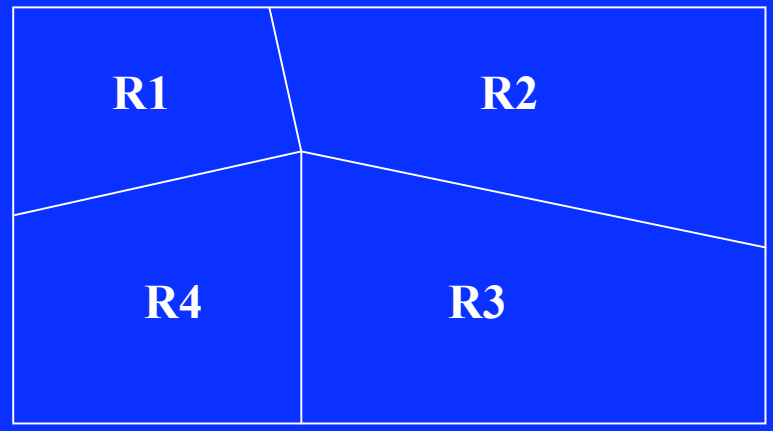
Classifier Selection

Goal: for each input pattern, select the classifier, if any, able to correctly classify it

Problem formulation

Two dimensional feature space

Partitioned in 4 regions



➢Easy to see that selection outperforms individual classifiers if I am able to select the best classifier for each region

➢Two critical issues: i)definition of regions; ii)selection algorithm

Classifier Selection

Two main approaches to design a classifier selection system:

Static vs. **Dynamic** Selection

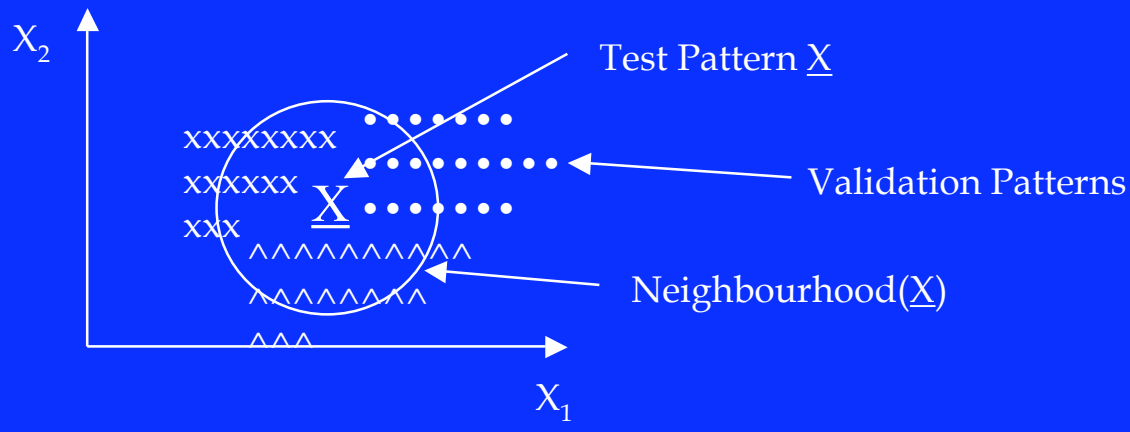
Static Selection

- Regions are defined before classification.
- For each region, a responsible classifier is identified
- Regions can be defined in different ways:
  - Histogram method: Space partition in "bins"
  - Clustering algorithms

Classifier Selection

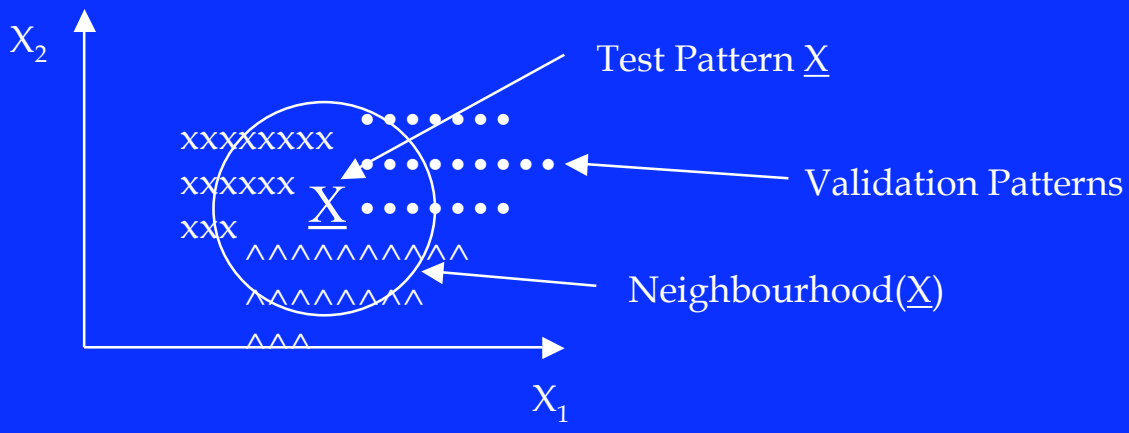
Dynamic Selection

SELECTION CONDITIONS based on estimates of classifiers accuracies in local regions of feature space surrounding an unknown test pattern X (Neighbourhood(X))



See works of Woods et al.; Giacinto and Roli

Selection Condition: An Example



- Woods et al. (1997) simply computed classifier local accuracy as the percentage of correctly classified patterns in the neighbourhood

- Giacinto and Roli (1997, etc) proposed some probabilistic measures:

$$\hat{P}(correct_j|\mathbf{X}^*) = \frac{\sum_{n=1}^{N} \hat{P}^{(j)}(\omega_i \mid \mathbf{X}_n \in \omega_i) \cdot W_n}{\sum_{n=1}^{N} W_n}$$

Some remarks on classifier selection

- Theoretically, "local" fusion rules can outperform "global" ones

- Selection can effectively handle correlated classifiers

- In practice, large and representative data sets are necessary to design a good selection system

- Further work is necessary to develop "robust" methods for estimating classifier local accuracy

➢ Works on Adaptive Mixtures of Local Experts (M. Jordan, et al.), not discussed for the sake of brevity, can be regarded as methods for dynamic classifier selection

Final Remarks on Fixed vs. Trained Fusers

- Fixed rules

  –Simplicity

  –Low memory and time requirements

  –Well-suited for ensembles of classifiers with independent/low correlated errors and similar performances

- Trained rules

  -Flexibility: potentially better performances than fixed rules

  -Trained rules are claimed to be more suitable than fixed ones for classifiers correlated or exhibiting different performances

  –High memory and time requirements

  ➢*Heavy demands on the quality and size of the training set*

MCS DESIGN

Parts 2 and 3 showed that designer of MCS has a toolbox containing a large number of instruments for generating and fusing classifiers.

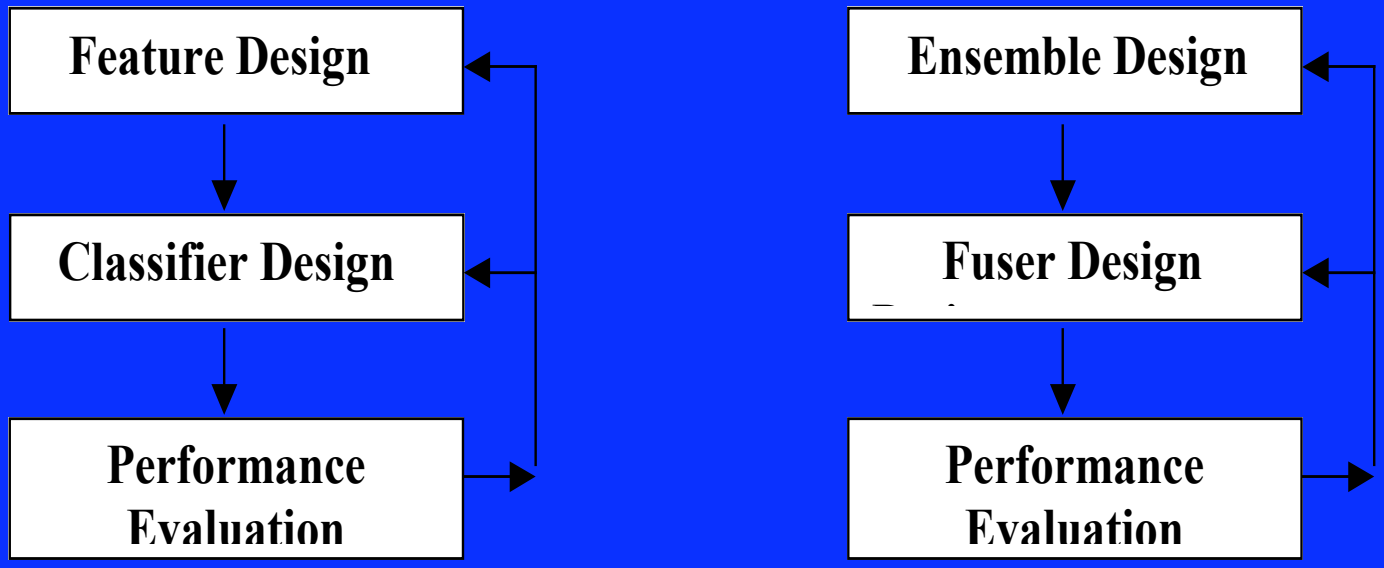
Two main design approaches have been defined so far

Coverage optimisation methods

Decision optimisation methods

➢However, combinations of the two above methods and hybrid, ad hoc, methods are often used

MCS DESIGN



Coverage optimisation methods: a simple fuser is given without any design. The goal is to create a set of complementary classifiers that can be fused optimally

-Bagging, Random Subspace, etc.

Decision optimisation methods: a set of carefully designed and optimised classifiers is given and unchangeable, the goal is to optimise the fuser

MCS DESIGN

- Decision optimisation method to MCS design is often used when previously carefully designed classifiers are available, or valid problem and designer knowledge is available

- Coverage optimisation method makes sense when creating carefully designed, "strong", classifiers is difficult, or time consuming

- Integration of the two basic approaches is often used

However, in general, no design method guarantees to obtain the "optimal" ensemble for a given fuser or a given application (Roli and Giacinto, 2002)

- The best MCS can only be determined by performance evaluation.

Question…

Explain why classifier selection should, in principle, always outperform individual classifiers. Try to provide a theoretical explanation.
Why it can fail in practice?

Send me via mail (roli@diee.unica.it) your justified answer

The very last question…

Identify an application/problem, if any, for which fusion of multiple classifiers does not work (i.e., does not provide any benefit)

Send me via mail (roli@diee.unica.it) your justified answer