AI*IA 2003

Fusion of Multiple Pattern Classifiers

PART II

Methods for Constructing MCS

- The effectiveness of MCS relies on combining *diverse*/*complementary* classifiers

Several approaches have been proposed to construct ensembles made up of complementary classifiers. 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

  - 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*

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

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, 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

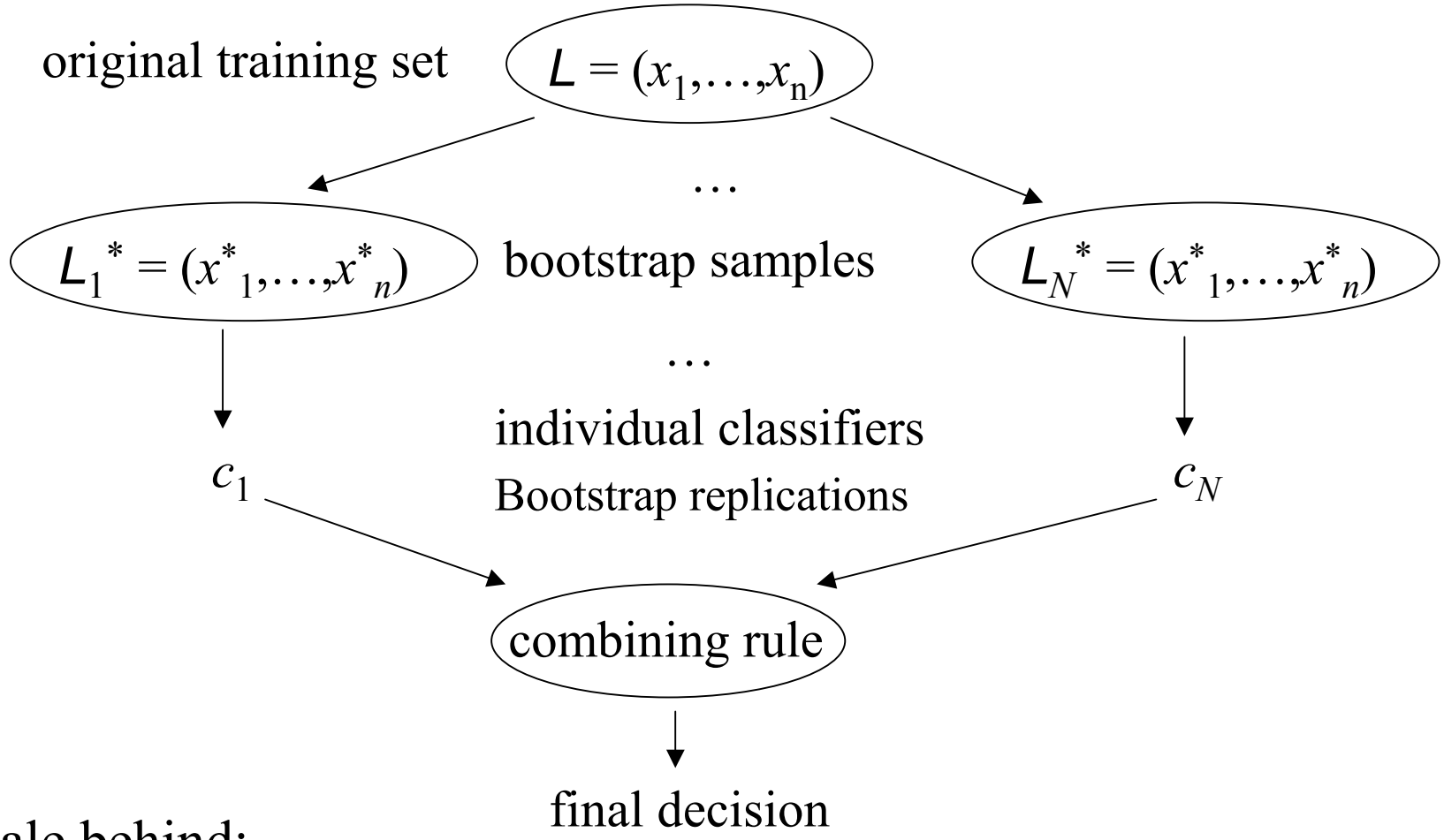
–$\mathbf{x}^* = (x^*_1,\ldots,x^*_n)$: random sample of size n drawn *with replacement* from the original sample $\mathbf{x} = (x_1,\ldots,x_n)$

–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:

  instances of an *unstable* classifier constructed on different bootstrap samples can exhibit significant differences

Combining rules for Bagging

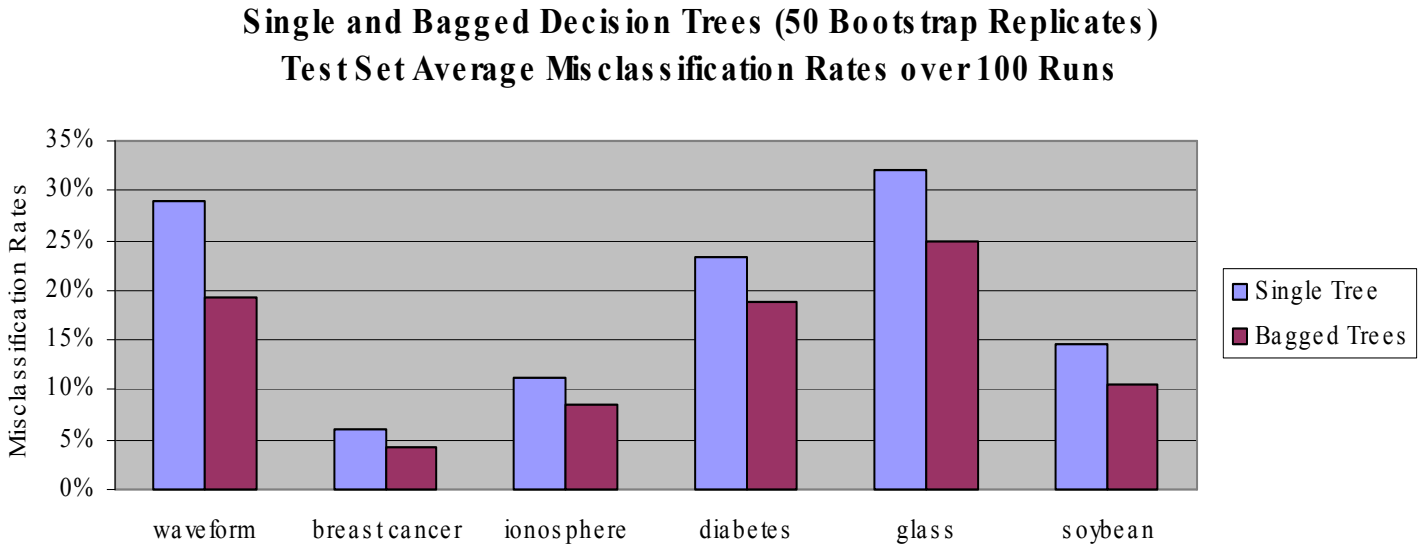
- Bagging is a method for *constructing* multiple classifiers, not a fusion rule

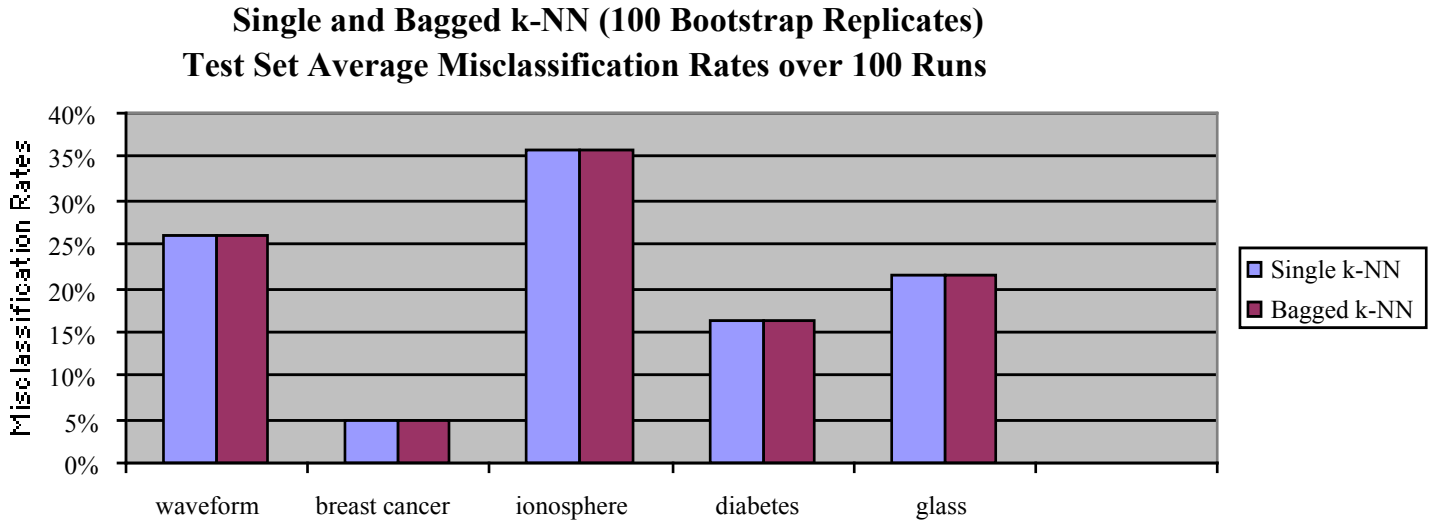
- In principle, any combining technique can be applied

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

- Experimental results show that bagging is effective when used with simple combining rules. However, the use of complex rules should be investigated further.

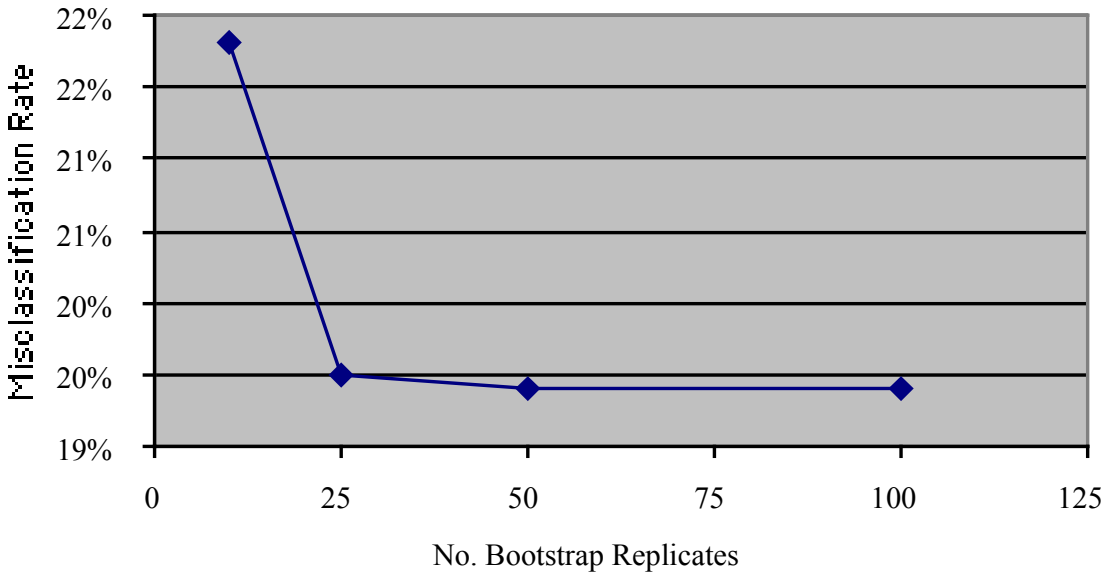
Examples of bagging (Breiman, 1996)





Number of bootstrap samples

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



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, 1998).

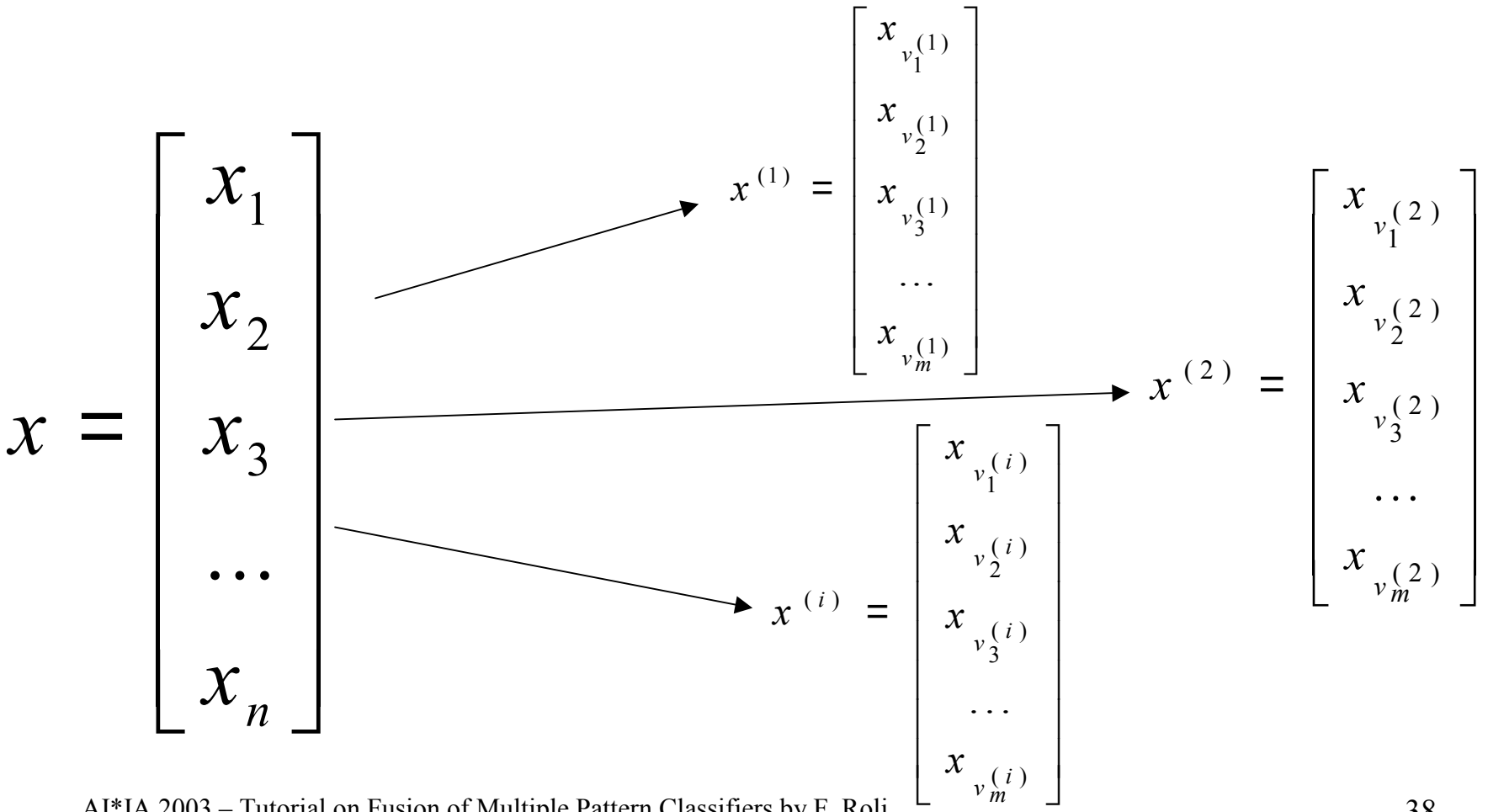
Let $X \subseteq \Re^n$ be a *n*-dimensional feature space.

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

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

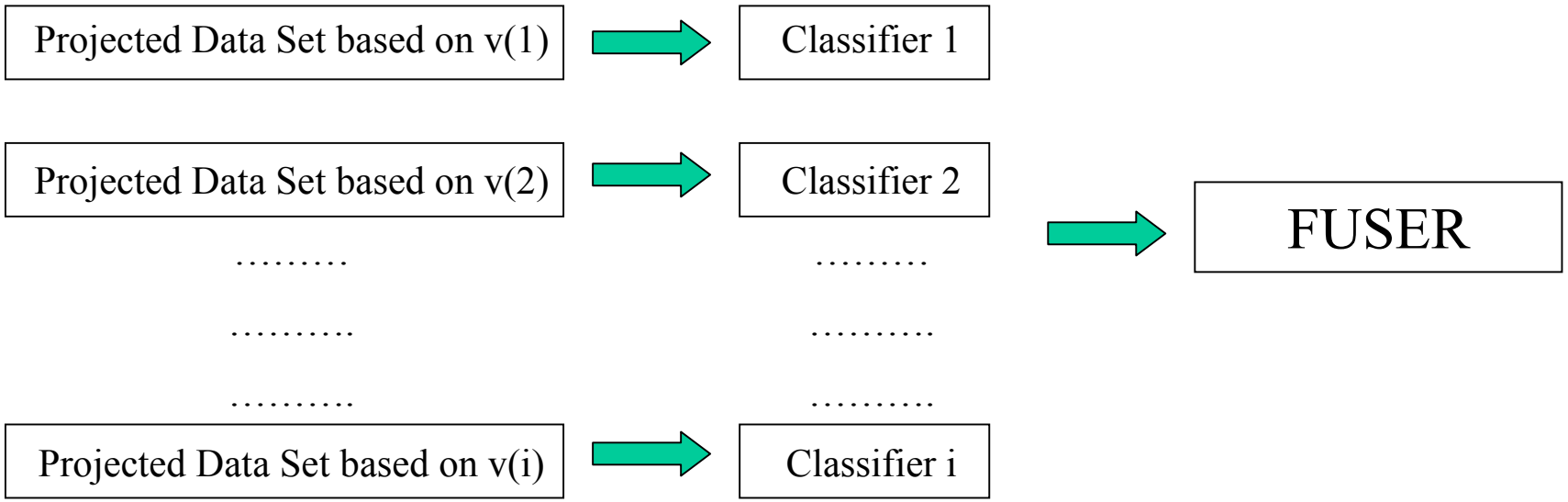
RSM: multiple subspace generation

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



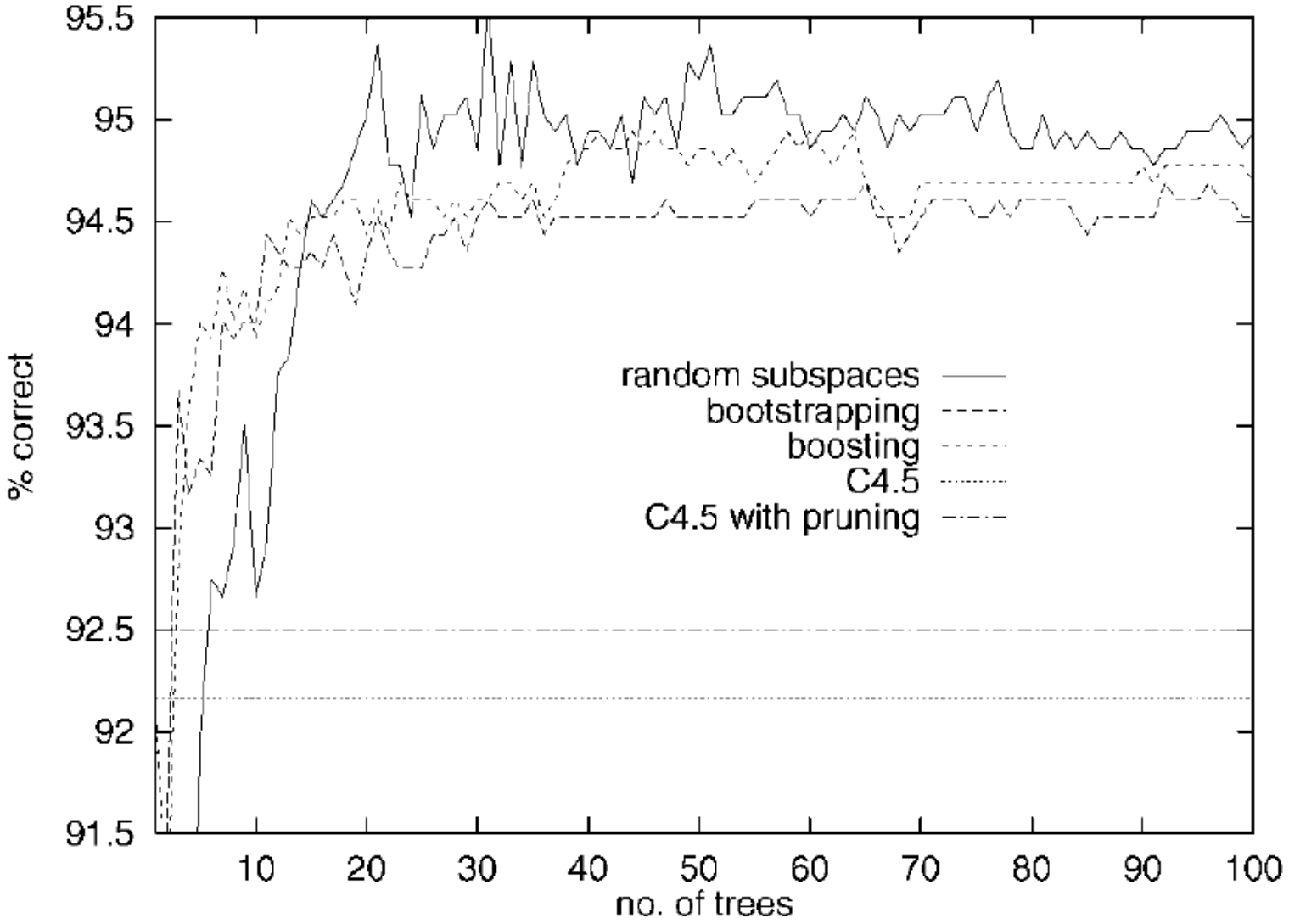
Decision fusion with RSM

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“, 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 (“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

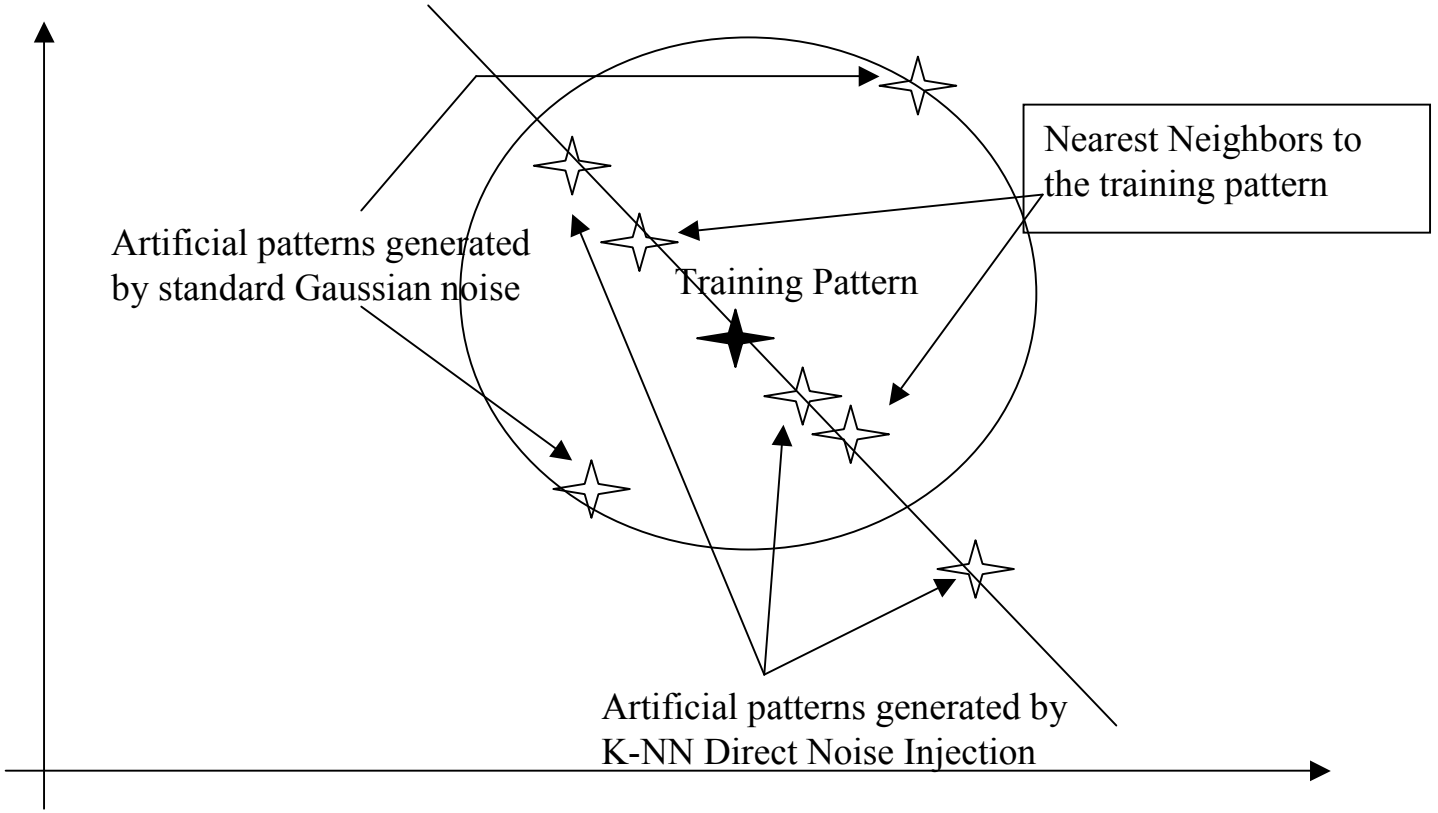
Other possibility: data splitting + adding noise

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

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

For each class c_j, let $\mathbf{b}^{(j)}=\{b_0,...,b_{m-1}\}$ 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):

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