We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

14

Output Coding Methods: Review and Experimental Comparison

Nicolás García-Pedrajas and Aida de Haro García
University of Cordoba, Spain

1. Introduction

Classification is one of the ubiquitous problems in Artificial Intelligence. It is present in almost any application where Machine Learning is used. That is the reason why it is one of the Machine Learning issues that has received more research attention from the first works in the field. The intuitive statement of the problem is simple, depending on our application we define a number of different classes that are meaningful to us. The classes can be different diseases in some patients, the letters in an optical character recognition application, or different functional parts in a genetic sequence. Usually, we are also provided with a set of patterns whose class membership is known, and we want to use the knowledge carried on these patterns to classify new patterns whose class is unknown.

The theory of classification is easier to develop for two class problems, where the patterns belong to one of only two classes. Thus, the major part of the theory on classification is devoted to two class problems. Furthermore, many of the available classification algorithms are either specifically designed for two class problems or work better in two class problems.

However, most of the real world classification tasks are multiclass problems. When facing a multiclass problem there are two main alternatives: developing a multiclass version of the classification algorithm we are using, or developing a method to transform the multiclass problem into many two class problems. The second choice is a must when no multiclass version of the classification algorithm can be devised. But, even when such a version is available, the transformation of the multiclass problem into several two class problems may be advantageous for the performance of our classifier. This chapter presents a review of the methods for converting a multiclass problem into several two class problems and shows a series of experiments to test the usefulness of this approach and the different available methods.

This chapter is organized as follows: Section 2 states the definition of the problem; Section 3 presents a detailed description of the methods; Section 4 reviews the comparison of the different methods performed so far; Section 5 shows an experimental comparison; and Section 6 shows the conclusions of this chapter and some open research fields.

2. Converting a multiclass problem to several two class problems

A classification problem of K classes and n training observations consists of a set of patterns whose class membership is known. Let $T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ be a set of n training

Source: Pattern Recognition Techniques, Technology and Applications, Book edited by: Peng-Yeng Yin,
ISBN 978-953-7619-24-4, pp. 626, November 2008, I-Tech, Vienna, Austria

samples where each pattern $\boldsymbol{x}_i$ belongs to a domain X. Each label is an integer from the set $Y = \{1, ..., K\}$. A multiclass classifier is a function $f: X \rightarrow Y$ that maps a pattern $\boldsymbol{x}$ to an element of Y.
The task is to find a definition for the unknown function, $f(\boldsymbol{x})$, given the set of training patterns. Although many real world problems are multiclass problems, $K > 2$, many of the most popular classifiers work best when facing two class problems, $K = 2$. Indeed many algorithms are specially designed for binary problems, such as Support Vector Machines (SVM) (Boser et al., 1992). A class binarization (Fürnkranz, 2002) is a mapping of a multi-class problem onto several two-class problems in a way that allows the derivation of a prediction for the multi-class problem from the predictions of the two-class classifiers. The two-class classifier is usually referred to as the *binary classifier* or *base learner*.
In this way, we usually have two steps in any class binarization scheme. First, we must define the way the multiclass problem is decomposed into several two class problems and train the corresponding binary classifier. Second, we must describe the way the binary classifiers are used to obtain the class of a given query pattern. In this section we show briefly the main current approaches of converting a multiclass problem into several two class problems. In the next section a more detailed description is presented, showing their pros and cons. Finally, in the experimental section several practical issues are addressed.
Among the proposed methods for approaching multi-class problems as many, possibly simpler, two-class problems, we can make a rough classification into three groups: one-vs-all, one-vs-one, and error correcting output codes based methods:

- One-vs-one (*ovo*): This method, proposed in Knerr et al. (1990), constructs $K(K-1)/2$ classifiers. Classifier ij, named f_{ij}, is trained using all the patterns from class i as positive patterns, all the patterns from class j as negative patterns, and disregarding the rest. There are different methods of combining the obtained classifiers, the most common is a simple voting scheme. When classifying a new pattern each one of the base classifiers casts a vote for one of the two classes used in its training. The pattern is classified into the most voted class.
- One-vs-all (*ova*): This method has been proposed independently by several authors (Clark & Boswell, 1991; Anand et al., 1992). *ova* method constructs K binary classifiers. Classifier i-th, f_i, is trained using all the patterns of class i as positive patterns and the patterns of the other classes as negative patterns. An example is classified in the class whose corresponding classifier has the highest output. This method has the advantage of simplicity, although it has been argued by many researchers that its performance is inferior to the other methods.
- Error correcting output codes (*ecoc*): Dietterich & Bakiri (1995) suggested the use of error correcting codes for multiclass classification. This method uses a matrix M of $\{-1, 1\}$ values of size $K \times L$, where L is the number of binary classifiers. The j-th column of the matrix induces a partition of the classes into two *metaclasses*. Pattern $\boldsymbol{x}$ belonging to class i is a positive pattern for j-th classifier if and only if $M_{ij} = 1$. If we designate f_j as the sign of the j-th classifier, the decision implemented by this method, $f(\boldsymbol{x})$, using the Hamming distance between each row of the matrix M and the output of the L classifiers is given by:

$$f(x) = argmin_{r \in 1,2,...,K} \sum_{i=1}^{L} \left(\frac{1 - sign\left(M_{ri} f_i(x)\right)}{2} \right) \quad (1)$$

These three methods comprehend all the alternatives we have to transform a multiclass problem into many binary problems. In this chapter we will discuss these three methods in depth, showing the most relevant theoretical and experimental results.
Although there are differences, class binarization methods can be considered as another form of ensembling classifiers, as different learners are combined to solve a given problem. An advantage that is shared by all class binarization methods is the possibility of parallel implementation. The multiclass problem is broken into several *independent* two-class problems that can be solved in parallel. In problems with large amounts of data and many classes, this may be a very interesting advantage over monolithic multiclass methods. This is a very interesting feature, as the most common alternative for dealing with complex multiclass problems, ensembles of classifiers constructed by boosting method, is inherently a sequential algorithm (Bauer & Kohavi, 1999).

3. Class binarization methods

This section describes more profoundly the three methods mentioned above with a special interest on theoretical considerations. Experimental facts are dealt with in the next section.

3.1 One-vs-one

The definition of one-vs-one (*ovo*) method is the following: *ovo* method constructs, for a problem of K classes, $K(K-1)/2$ binary classifiers[1], f_{ij}, $i = 1, ..., K-1$, $j = i+1, ..., K$. The classifier f_{ij} is trained using patterns from class i as positive patterns and patterns from class j as negative patterns. The rest of patterns are ignored. This method is also known as *round-robin classification*, *all-pairs* and *all-against-all*.
Once we have the trained classifiers, we must develop a method for predicting the class of a test pattern $\boldsymbol{x}$. The most straightforward and simple way is using a voting scheme, we evaluate every classifier, $f_{ij}(\boldsymbol{x})$, which casts a vote for either class i or class j. The most voted class is assigned to the test pattern. Ties are solved randomly or assigning the pattern to the most frequent class among the tied ones. However, this method has a problem. For every pattern there are several classifiers that are forced to cast an erroneous vote. If we have a test pattern from class k, all the classifiers that are not trained using class k must also cast a vote, which cannot be accurate as k is not among the two alternatives of the classifier. For instance, if we have $K = 10$ classes, we will have 45 binary classifiers. For a pattern of class 1, there are 9 classifiers that can cast a correct vote, but 36 that cannot. In practice, if the classes are independent, we should expect that these classifiers would not largely agree on the same wrong class. However, in some problems whose classes are hierarchical or have similarities between them, this problem can be a source for incorrect classification. In fact, it has been shown that it is the main source of failure of *ovo* in real world applications (García-Pedrajas & Ortiz-Boyer, 2006).
This problem is usually termed as the problem of the *incompetent classifiers* (Kim & Park, 2003). As it has been pointed out by several researchers, it is an inherent problem of the method, and it is not likely that a solution can be found. Anyway, it does not prevent the usefulness of *ovo* method.

1 This definition assumes that the base learner used is class-symmetric, that is, distinguishing class i from class j is the same task as distinguishing class j from class i, as this is the most common situation.

Regarding the causes of the good performance of *ovo*, Fürnkranz (2002) hypothesized that *ovo* is just another ensemble method. The basis of this assumption is that *ovo* tends to perform well in problems where ensemble methods, such as bagging or boosting, also perform well. Additionally, other works have shown that the combination of *ovo* and ADABOOST boosting method do not produce improvements in the testing error (Schapire, 1997; Allwein et al, 2000), supporting the idea that they perform a similar work.

One of the disadvantages of *ovo* appears in classification time. For predicting the class of a test pattern we need to evaluate *K(K-1)/2* classifiers, which can be a time consuming task if we have many classes. In order to avoid this problem, Platt et al. (2000) proposed a variant of *ovo* method based on using a directed acyclic graph for evaluating the class of a testing pattern. The method is identical to *ovo* at training time and differs from it at testing time. The method is usually referred to as the Decision Directed Acyclic Graph (*DDAG*). The method constructs a rooted binary acyclic graph using the classifiers. The nodes are arranged in a triangle with the root node at the top, two nodes in the second layer, four in the third layer, and so on. In order to evaluate a *DDAG* on input pattern **x**, starting at the root node the binary function is evaluated, and the next node visited depends upon the results of this evaluation. The final answer is the class assigned by the leaf node visited at the final step. The root node can be assigned randomly. The testing error reported using *ovo* and *DDAG* are very similar, the latter having the advantage of a faster classification time.

Hastie & Tibshirani (1998) gave a statistical perspective of this method, estimating class probabilities for each pair of classes and then coupling the estimates together to get a decision rule.

3.2 One-vs-all

One-vs-all (*ova)* method is the most intuitive of the three discussed options. Thus, it has been proposed independently by many researchers. As we have explained above, the method constructs *K* classifiers for *K* classes. Classifier f_i is trained to distinguish between class *i* and all other classes. In classification time all the classifiers are evaluated and the query pattern is assigned to the class whose corresponding classifier has the highest output.

This method has the advantage of training a smaller number of classifiers than the other two methods. However, it has been theoretically shown (Fürnkranz, 2002) that the training of these classifiers is more complex than the training of *ovo* classifiers. However, this theoretical analysis does not consider the time associated with the repeated execution of an actual program, and also assumes that the execution time is linear with the number of patterns. In fact, in the experiments reported here the execution time of *ova* is usually shorter than the time spent by *ovo* and *ecoc.*

The main advantage of *ova* approach is its simplicity. If a class binarization must be performed, it is perhaps the first method one thinks of. In fact, some multiclass methods, such as the one used in multiclass multilayer Perceptron, are based on the idea of separating each class from all the rest of classes.

Among its drawbacks several authors argue (Fürnkranz, 2002) that separating a class from all the rest is a harder task than separating classes in pairs. However, in practice the situation depends on another issue. The task of separating classes in pairs may be simple, but also, there are fewer available patterns to learn the classifiers. In many cases the classifiers that learned to distinguish between two classes have large generalization errors due to the small number of patterns used in their training process. These large errors undermine the performance of *ovo* in favor of *ova* in several problems.

3.3 Error-correcting output codes

This method was proposed by Dietterich & Bakiri (1995). They use a "coding matrix" $M \in \{-1,+1\}^{KxL}$ which has a row for each class and a number of columns, *L*, defined by the user. Each row codifies a class, and each column represents a binary problem, where the patterns of the classes whose corresponding row has a +1 are considered as positive samples, and the patterns whose corresponding row has a -1 as negative samples. So, after training we have a set of *L* binary classifiers, $\{f_1, f_2, ..., f_L\}$. In order to predict the class of an unknown test sample **x**, we obtain the output of each classifier and classify the pattern in the class whose coding row is *closest* to the output of the binary classifiers $(f_1(\mathbf{x}), f_2(\mathbf{x}), ..., f_L(\mathbf{x}))$. There are many different ways of obtaining the closest row. The simplest one is using Hamming distance, breaking the ties with a certain criterion. However, this method loses information, as the actual output of each classifier can be considered a measure of the probability of the bit to be 1. In this way, L^1 norm can be used instead of Hamming distance. The L^1 distance between a codeword M_i and the output of the classifiers $F = \{f_1, f_2, ..., f_L\}$ is defined by:

$$L^1(M_i, F) = \sum_{j=0}^{L} \left| M_{ij} - f_j \right| \tag{2}$$

The L^1 norm is preferred over Hamming distance for its better performance and as it has also been proven that *ecoc* method is able to produce reliable probability estimates. Windeatt & Ghaderi (2003) tested several decoding strategies, showing that none of them was able to improve the performance of L^1 norm significantly. Several other decoding methods have been proposed (Passerini et al., 2004) but only with a marginal advantage over L^1 norm.

This approach was pioneered by Sejnowski & Rosenberg (1987) who defined manual codewords for the NETtalk system. In that work, the codewords were chosen taking into account different features of each class. The contribution of Dietterich & Bakiri was considering the principles of error-correcting codes design for constructing the codewords.

The idea is considering the classification problem similar to the problem of transmitting a string of bits over a parallel channel. As a bit can be transmitted incorrectly due to a failure of the channel, we can consider that a classifier that does not predict accurately the class of a sample is like a bit transmitted over an unreliable channel. In this case the channel consists of the input features, the training patterns and the learning process. In the same way as an error-correcting code can recover from the failure of some of the transmitted bits, *ecoc* codes might be able to recover from the failure of some of the classifiers.

However, this argumentation has a very important issue, error-correcting codes rely on the independent transmission of the bits. If the errors are correlated, the error-correcting capabilities are seriously damaged. In a pattern recognition task, it is debatable whether the different binary classifiers are independent. If we consider that the input features, the learning process and the training patterns are the same, although the learning task is different, the independence among the classifiers is not an expected result.

Using the formulation of *ecoc* codes, Allwein et al. (2000) presented a unifying approach, using coding matrices of three values, *{-1, 0, 1}*, *0* meaning "don't care". Using this approach, *ova* method can be represented with a matrix of 1's in the main diagonal and -1 in the remaining places, and *ovo* with a matrix of *K(K-1)/2* columns, each one with a +1, a -1 and the remaining places in the column set to 0. Allwein et al. also presented training and

generalization error bounds for output codes when loss based decoding is used. However, the generalization bounds are not tight, and they should be seemed more as a way of considering the qualitative effect of each of the factors that have an impact on the generalization error. In general, these theoretical studies have recognized shortcomings and the bounds on the error are too loose for practical purposes. In the same way, the studies on the effect of *ecoc* on bias/variance have the problem of estimating these components of the error in classification problems (James, 2003).

As an additional advantage, Dietterich & Bakiri (1995) showed, using rejection curves, that *ecoc* are good estimators of the confidence of the multiclass classifier. The performance of *ecoc* codes has been explained in terms of reducing bias/variance and by interpreting them as large margin classifiers (Masulli & Valentini, 2003). However, a generally accepted explanation is still lacking as many theoretical issues are open.

In fact, several issues concerning *ecoc* method remain debatable. One of the most important is the relationship between the error correcting capabilities and the generalization error. These two aspects are also closely related to the independence of the dichotomizers. Masulli & Valentini (2003) performed a study using 3 real-world problems without finding any clear trend.

3.3.1 Error-correcting output codes design

Once we have stated that the use of codewords designed by their error-correcting capabilities may be a way of improving the performance of the multiclass classifier, we must face the design of such codes.

The design of error-correcting codes is aimed at obtaining codes whose separation, in terms of Hamming distance, is maximized. If we have a code whose minimum separation between codewords is d, then the code can correct at least $\lfloor (d-1)/2 \rfloor$ bits. Thus, the first objective is maximizing minimum row separation. However, there is another objective in designing *ecoc* codes, we must enforce a low correlation between the binary classifiers induced by each column. In order to accomplish this, we maximize the distance between each column and all other columns. As we are dealing with class symmetric classifiers, we must also maximize the distance between each column and the complement of all other columns. The underlying idea is that if the columns are similar (or complementary) the binary classifiers learned from those columns will be similar and tend to make correlated mistakes.

These two objectives make the task of designing the matrix of codewords for *ecoc* method more difficult than the designing of error-correcting codes. For a problem with K classes, we have $2^{k-1} - 1$ possible choices for the columns. For small values of K, we can construct exhaustive codes, evaluating all the possible matrices for a given number of columns. However, for larger values of K the designing of the coding matrix is an open problem.

The designing of a coding matrix is then an optimization problem that can only be solved using an iterative optimization algorithm. Dietterich & Bakiri (1995) proposed several methods, including randomized hill-climbing and BCH codes. BCH algorithm is used for designing error correcting codes. However, its application to *ecoc* design is problematic, among other factors because it does not take into account column separation, as it is not needed for error-correcting codes. Other authors have used general purpose optimization algorithms such as evolutionary computation (García-Pedrajas & Fyfe, 2008).

More recently, methods for obtaining the coding matrix taking into account the problem to be solved have been proposed. Pujol et al. (2006) proposed *Discriminant ECOC*, a heuristic

method based on a hierarchical partition of the class space that maximizes a certain discriminative criterion. García-Pedrajas & Fyfe (2008) coupled the design of the codes with the learning of the classifiers, designing the coding matrix using an evolutionary algorithm.

4. Comparison of the different methods

The usual question when we face a multiclass problem and decide to use a class binarization method is which is the best method for my problem. Unfortunately, this is an open question which generates much controversy among the researchers.

One of the advantages of *ovo* is that the binary problems generated are simpler, as only a subset of the whole set of patterns is used. Furthermore, it is common in real world problems that the classes are pairwise separable (Knerr et al., 1992), a situation that is not so common for *ova* and *ecoc* methods.

In principle, it may be argued that replacing a K classes problem by $K(K-1)/2$ problems should significantly increase the computational cost of the task. However, Fürnkranz (2002) presented theoretical arguments showing that *ovo* has less computational complexity than *ova*. The basis underlying the argumentation is that, although *ovo* needs to train more classifiers, each classifier is simpler as it only focuses on a certain pair of classes disregarding the remaining patterns. In that work an experimental comparison is also performed using as base learner `Ripper` algorithm (Cohen, 1995). The experiments showed that *ovo* is about 2 times faster than *ova* using `Ripper` as base learner. However, the situation depends on the base learner used. In many cases there is an overhead associated with the application of the base learner which is independent of the complexity of the learning task. Furthermore, if the base learner needs some kind of parameters estimation, using cross-validation or any other method for parameters setting, the situation may be worse. In fact, in the experiments reported in Section 5, using powerful base learners, the complexity of *ovo* was usually greater than the complexity of *ova*.

There are many works devoted to the comparison of the different methods. Hsu & Lin (2002) compared *ovo*, *ova* and two native multiclass methods using a SVM. They concluded that *ova* was worse than the other methods, which showed a similar performance. In fact, most of the previous works agree on the inferior performance of *ova*. However, the consensus about the inferior performance of *ova* has been challenged recently (Rifkin & Klautau, 2004). In an extensive discussion of previous work, they concluded that the differences reported were mostly the product of either using too simple base learners or poorly tuned classifiers. As it is well known, the combination of weak learners can take advantage of the independence of the errors they make, while combining powerful learners is less profitable due to their more correlated errors. In that paper, the authors concluded that *ova* method is very difficult to be outperformed if a powerful enough base learner is chosen and the parameters are set using a sound method.

5. Experimental comparison

As we have shown in the previous section, there is no general agreement on which one of the presented methods shows the best performance. Thus, in this experimental section we will test several of the issues that are relevant for the researcher, as a help for choosing the most appropriate method for a given problem.

For the comparison of the different models, we selected 41 datasets from the UCI Machine Learning Repository which are shown in Table 1. The estimation of the error is made using 10-fold cross-validation. The datasets were selected considering problems of at least 6 classes for *ecoc* codes (27 datasets), and problems with at least 3 classes for the other methods. We will use as main base learner a C4.5 decision tree (Quinlan, 1993), because it is a powerful widely used classification algorithm and has a native multiclass method that can be compared with class binarization algorithms. In some experiments we will also show results with other base learners for the sake of completeness. It is interesting to note that this set of problems is considerably larger than the used in the comparison studies cited along the paper.

When the differences between two algorithms must be statistically assessed we use a Wilcoxon test for several reasons. Wilcoxon test assumes limited commensurability. It is safer than parametric tests since it does not assume normal distributions or homogeneity of variance. Thus, it can be applied to error ratios. Furthermore, empirical results show that it is also stronger than other tests (Demšar, 2006).

Dataset	*Cases*	*Inputs*	*Classes*	*Binary classifiers*		
				Dense ecoc	*Sparse ecoc*	*One-vs-one*
Abalone	4177	10	29	2.68e+8	3.43e+13	406
Anneal	898	59	5	15	90	10
Arrhythmia	452	279	13	4095	7.88e+5	78
Audiology	226	93	24	8.38e+6	1.41e+11	276
Autos	205	72	6	31	301	15
Balance	625	4	3	3	6	3
Car	1728	16	4	7	25	6
Dermatology	366	34	6	31	301	15
Ecoli	336	7	8	127	3025	28
Gene	3175	120	3	3	6	3
Glass	214	9	6	31	301	15
Horse	364	58	3	3	6	3
Hypo	3772	29	4	7	25	6
Iris	150	4	3	3	6	3
Isolet	7797	617	26	3.35e+7	1.27e+12	325
Krkopt	28056	6	6	1.31e+5	1.93e+8	153
Led24	200	24	10	511	28501	45
Letter	20000	16	26	3.35e+7	1.27e+12	325
Lrs	531	101	10	511	28501	45
Lymph	148	38	4	7	25	6
Mfeat-fou	2000	76	10	511	28501	45
Mfeat-kar	2000	64	10	511	28501	45
Mfeat-mor	2000	6	10	511	28501	45

Dataset	*Cases*	*Inputs*	*Classes*	*Binary classifiers*		
				Dense ecoc	*Sparse ecoc*	*One-vs-one*
Mfeat-zer	2000	47	10	511	28501	45
New-thyroid	215	5	3	3	6	3
Nursery	12960	23	5	15	90	10
Optdigits	5620	64	10	511	28501	45
Page-blocks	5473	10	5	15	90	10
Pendigits	10992	16	10	511	28501	45
Primary	339	23	22	2.09e+6	1.56e+10	231
Satimage	6435	36	6	31	301	15
Segment	2310	19	7	63	966	21
Soybean	683	82	19	2.62e+5	5.80e+8	171
Texture	5500	40	11	1023	86526	55
Vehicle	846	18	4	7	25	6
Vowel	990	10	11	1023	86526	55
Waveform	5000	40	3	3	6	3
Wine	178	13	3	3	6	3
Yeast	1484	8	10	511	28501	45
Zip	9298	256	10	511	28501	45
Zoo	101	16	7	63	966	21

Table 1. Summary of datasets used in the experiments.

The first set of experiments is devoted to studying the behavior of *ecoc* codes. First, we test the influence of the size of codewords on the performance of *ecoc* method. We also test whether the use of codes designed by their error correcting capabilities are better than codes randomly designed. For the first experiment we use codes of 30, 50, 100 and 200 bits.

In many previous studies it has been shown that, in general, the advantage of using codes designed for their error correcting capabilities over random codes is only marginal. We construct random codes just generating the coding matrix randomly with the only post-processing of removing repeated columns or rows. In order to construct error-correcting codes, we must take into account two different objectives, as mentioned above, column and row separation. Error-correcting design algorithm are only concerned with row separation so their use must be coupled with another method for ensuring column separation. Furthermore, many of these algorithms are too complex and difficult to scale for long codes. So, instead of these methods, we have used an evolutionary computation method, a genetic algorithm to construct our coding matrix.

Evolutionary computation (EC) (Ortiz-Boyer at al., 2005) is a set of global optimization techniques that have been widely used over the last years for almost every problem within the field of Artificial Intelligence. In evolutionary computation a population (set) of individuals (solutions to the problem faced) are codified following a code similar to the genetic code of plants and animals. This population of solutions is evolved (modified) over a certain number of generations (iterations) until the defined stop criterion is fulfilled. Each

individual is assigned a real value that measures its ability to solve the problem, which is called its *fitness*.

In each iteration, new solutions are obtained combining two or more individuals (crossover operator) or randomly modifying one individual (mutation operator). After applying these two operators a subset of individuals is selected to survive to the next generation, either by sampling the current individuals with a probability proportional to their fitness, or by selecting the best ones (elitism). The repeated processes of crossover, mutation and selection are able to obtain increasingly better solutions for many problems of Artificial Intelligence. For the evolution of the population, we have used the CHC algorithm. The algorithm optimizes row and columns separation. We will refer to these codes as CHC codes throughout the paper for brevity's sake.

This method is able to achieve very good matrices in terms of our two objectives, and also showed better results than other optimization algorithms we tried. Figure 1 shows the results for the four sizes of code length and both types of codes, random and CHC. For problems with few classes, the experiments are done up to the maximum length available. For instance, glass dataset has 6 classes, which means that for dense codes we have 31 different columns, so for this problem only codes of 30 bits are available and it is not included in this comparison.

The figure shows two interesting results. Firstly, we can see that the increment in the size of the codewords has the effect of improving the accuracy of the classifier. However, the effect is less marked as the codeword is longer. In fact, there is almost no differences between a codeword of 100 bits and a codeword of 200 bits. Secondly, regarding the effect of error correcting capabilities, there is a general advantage of CHC codes, but the differences are not very marked. In general, we can consider that a code of 100 bits is enough, as the improvement of the error using 200 bits is hardly significant, and the added complexity important.

Allwein et al. (2000) proposed sparse *ecoc* codes, where 0's are allowed in the columns, meaning "don't care". It is interesting to show whether the same pattern observed for dense codes, is also present in sparse codes. In order to test the behavior of sparse codes, we have performed the same experiment as for dense codes, that is, random and CHC codes of 30, 50, 100 and 200 bits and C.45 as base learner. Figure 2 shows the testing error results. For sparse codes we have more columns available (see Table 1), so all the datasets with 6 classes or more are included in the experiments.

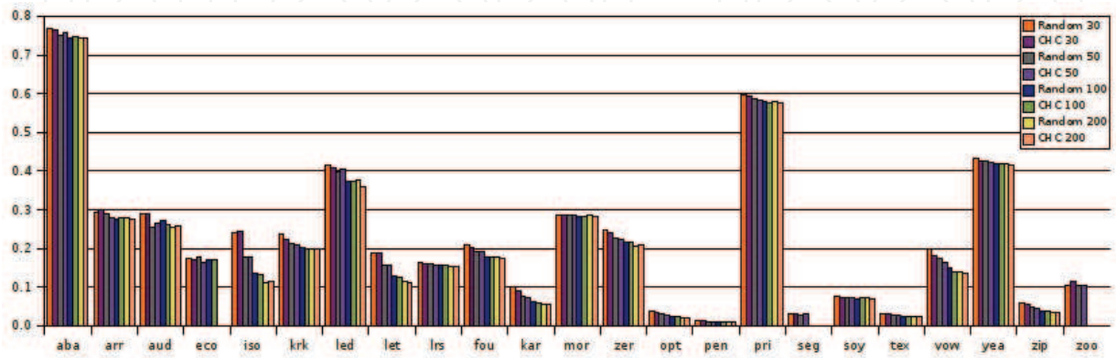


Fig. 1. Error values for *ecoc* dense codes using codewords of 30, 50, 100 and 200 bits and a C4.5 tree as base learner.

As a general rule, the results are similar, with the difference that the improvement of large codes, 100 and 200 bits, over small codes, 30 and 50 bits, is more marked than for dense codes. The figure also shows that the performance of both kind of codes, dense and sparse, is very similar. It is interesting to note that Allwein et al. (2000) suggested codes of $\lfloor 10\log_2(K) \rfloor$ bits for dense codes and of $\lfloor 15\log_2(K) \rfloor$ bits for sparse codes, being K the number of classes. However, in our experiments it is shown that these values are too small, as longer codes are able to improve the results of codewords of that length.

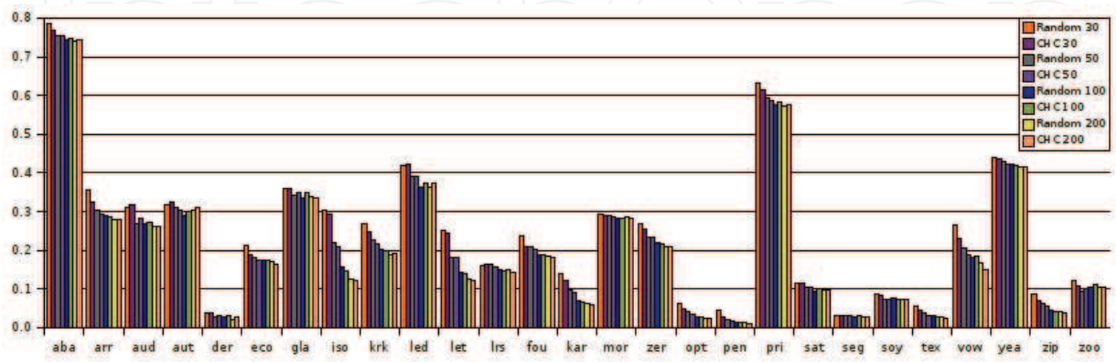


Fig. 2. Error values for *ecoc* sparse codes using codewords of 30, 50, 100 and 200 bits and a C4.5 tree as base learner.

We measure the independence of the classifiers using Yule's Q statistic. Classifiers that recognize the same patterns will have positive values of Q, and classifiers that tend to make mistakes in different patterns will have negative values of Q. For independent classifiers the expectation of Q is 0. For a set of L classifiers we use an average value Q_{av}:

$$Q_{av} = \frac{2}{L(L-1)} \sum_{i=1}^{L-1} \sum_{k=i+1}^{L} q_{i,k}, \tag{3}$$

where $q_{i,j}$ is the value of Q statistic between i and j classifiers which is given by:

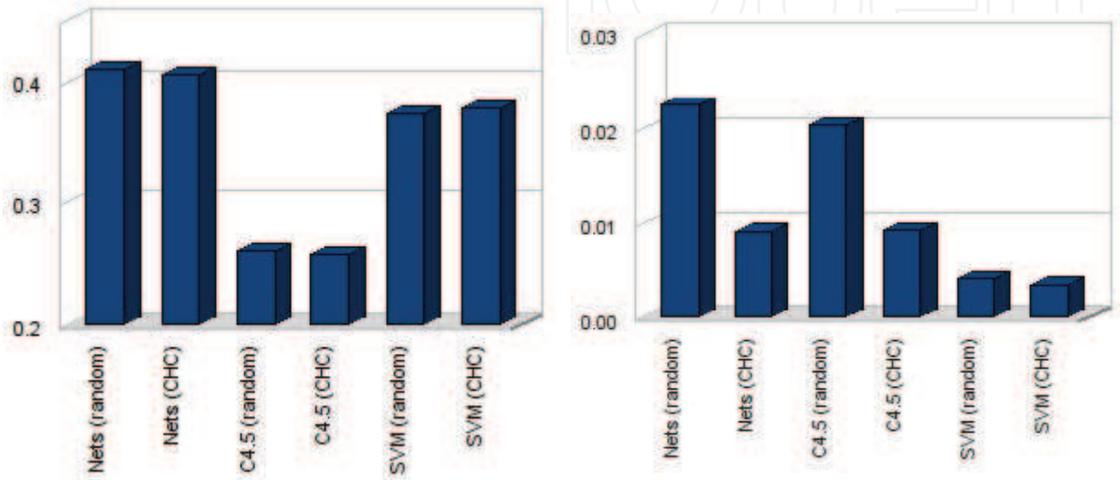
$$q_{i,j} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}}, \tag{4}$$

where N^{11} means both classifiers agree and are correct, N^{00} means both classifiers agree and are wrong, N^{01} means classifier i is wrong and classifier j is right, and N^{10} means classifiers i is right and classifier j is wrong. In this experiment, we test whether constructing codewords with higher Hamming distances improves independence of the classifiers.

After these previous experiments, we consider that a CHC code of 100 bits can be considered representative of *ecoc* codes, as the improvement obtained with longer codes is not significant.

It is generally assumed that codes designed by their error correcting capabilities should improve the independence of the errors between the classifiers. In this way, their failure to improve the performance of random codes is attributed to the fact that more difficult dichotomies are induced. However, whether the obtained classifiers are more independent is not an established fact. In this experiment we study if this assumption of independent errors is justified.

For this experiment, we have used three base learners, C4.5 decision trees, neural networks and support vector machines. Figure 3 shows the average values of Q statistic for all the 27 datasets for dense and sparse codes using random and CHC codes in both cases. For dense codes, we found a very interesting result. Both types of codes achieve very similar results in terms of independence of errors, and CHC codes are not able to improve the independence of errors of random codes, which is probably one of the reasons why CHC codes are no better than random codes. This is in contrast with the general belief, showing that some of the assumed behavior of *ecoc* codes must be further experimentally tested.



(a) Dense codes (b) Sparse codes

Fig. 3. Average Q value for dense and sparse codes using three different base learners

The case for sparse codes is different. For these types of codes, CHC codes are significantly more independent for neural networks and C4.5. For SVM, CHC codes are also more independent although the differences are not statistically significant. The reason may be found in the differences between both types of codes. For dense codes, all the binary classifiers are trained using all the data, so although the dichotomies are different, it is more difficult to obtain independent classifiers as all classifiers are trained using the same data. On the other hand, sparse codes disregard the patterns of the classes which have a 0 in the corresponding column representing the dichotomy. CHC algorithm enforces column separation, which means that the columns have less overlapping. Thus, the binary classifiers induced by CHC matrices are trained using datasets that have less overlapping and can be less dependent.

So far we have studied *ecoc* method. The following experiment is devoted to the study of the other two methods: *ovo* and *ova*. The differences in performance between *ovo* and *ova* is a matter of discussion. We have stated that most works agree on a general advantage of *ovo*, but a careful study performed by Rifkin & Klautau (2004) has shown that most of the reported differences are not significant. In the works studied in that paper, the base learner was a support vector machine (SVM). As we are using a C4.5 algorithm, it is interesting to show whether the same conclusions can be extracted from our experiments. Figure 4 shows a comparison of results for the 41 tested datasets of the two methods. The figure shows for each dataset a point which reflects in the x-axis the testing error of *ovo* method, and in the y-axis the testing error of *ova* method. A point above the main diagonal means that *ovo* is

performing better than *ova*, and vice versa. The figures shows a clear advantage of *ovo* method, which performs better than *ova* in 31 of the 41 datasets. The differences are also marked for many problems, as it is shown in the figure by the large separation of the points from the main diagonal. As C4.5 has no relevant parameters, the hypothesis of Rifkin & Klautau of a poor parameter setting is not applicable.

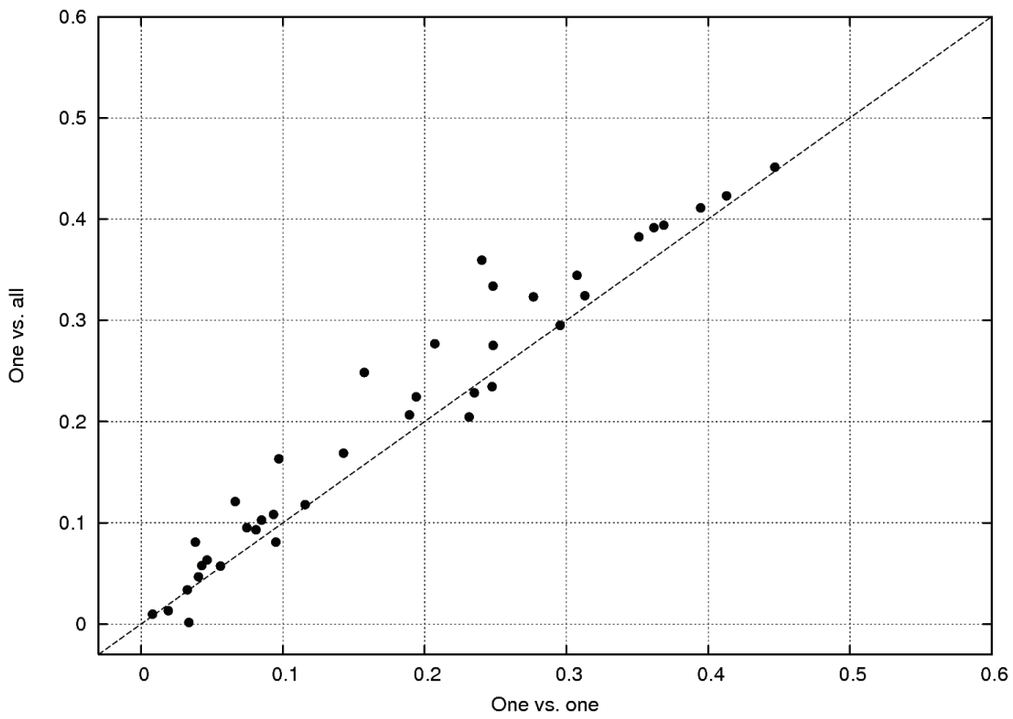


Fig. 4. Comparison of *ovo* and *ova* methods in terms of testing error.

In the previous experiments, we have studied the behavior of the different class binarization methods. However, there is still an important question that remains unanswered. There are many classification algorithms that can be directly applied to multiclass problems, so the obvious question is whether the use of *ova*, *ovo* or *ecoc* methods can be useful when a "native" multiclass approach is available. For instance, for *C4.5 ecoc* codes are more complex than the native multiclass method, so we must get an improvement from *ecoc* codes to overcome this added complexity. In fact, this situation is common with most classification methods, as a general rule class binarization is a more complex approach than the available native multiclass methods.

We have performed a comparison of *ecoc* codes using a CHC code of 100 *bits, ovo* and *ova* methods and the native multiclass method provided with C4.5 algorithm. The results are shown in Figure 5, for the 41 datasets.

The results in Figure 5 show that *ecoc* and *ovo* methods are able to improve native C4.5 multiclass method most of the times. In fact, *ecoc* method is better than the native method in all the 27 datasets. *ovo* is better than the native method in 31 out of 41 datasets. On the other hand, *ova* is not able to regularly improve the results of the native multiclass method. These results show that *ecoc* and *ovo* methods are useful, even if we have a native multiclass method for the classification algorithm we are using.

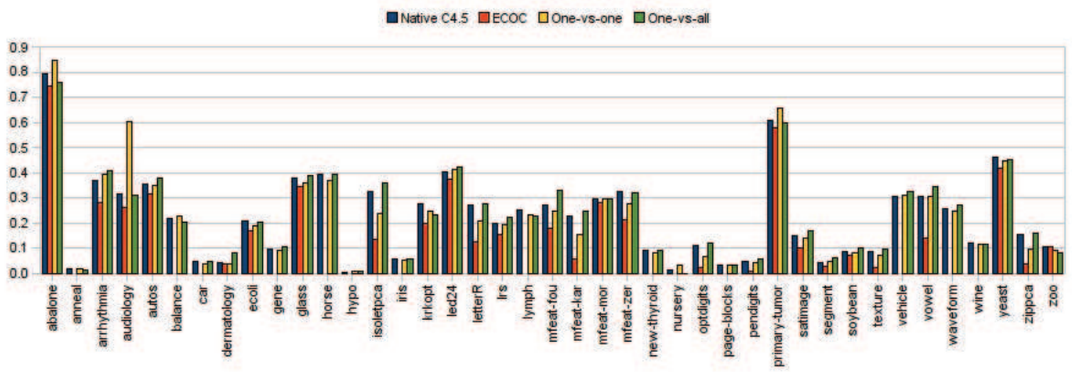


Fig. 5. Error values for *ovo*, *ova* and *ecoc* dense codes obtained with a CHC algorithm using codewords of 100 bits (or the longest available) and a C4.5 tree as base learner, and the native C4.5 multiclass algorithm.

Several authors have hypothesized that the lack of improvement when using codes designed by their error correcting capabilities over random ones may be due to the fact that some of the induced dichotomies could be more difficult to learn. In this way, the improvement due to a larger Hamming distance may be undermined by more difficult problems. In the same way, it has been said that *ovo* binary problems are easier to solve than *ova* binary problems. These two statements must be corroborated by the experiments.
Figure 6 shows the average generalization binary testing error of all the base learners for each dataset for random and CHC codes. As in previous figures a point is drawn for each

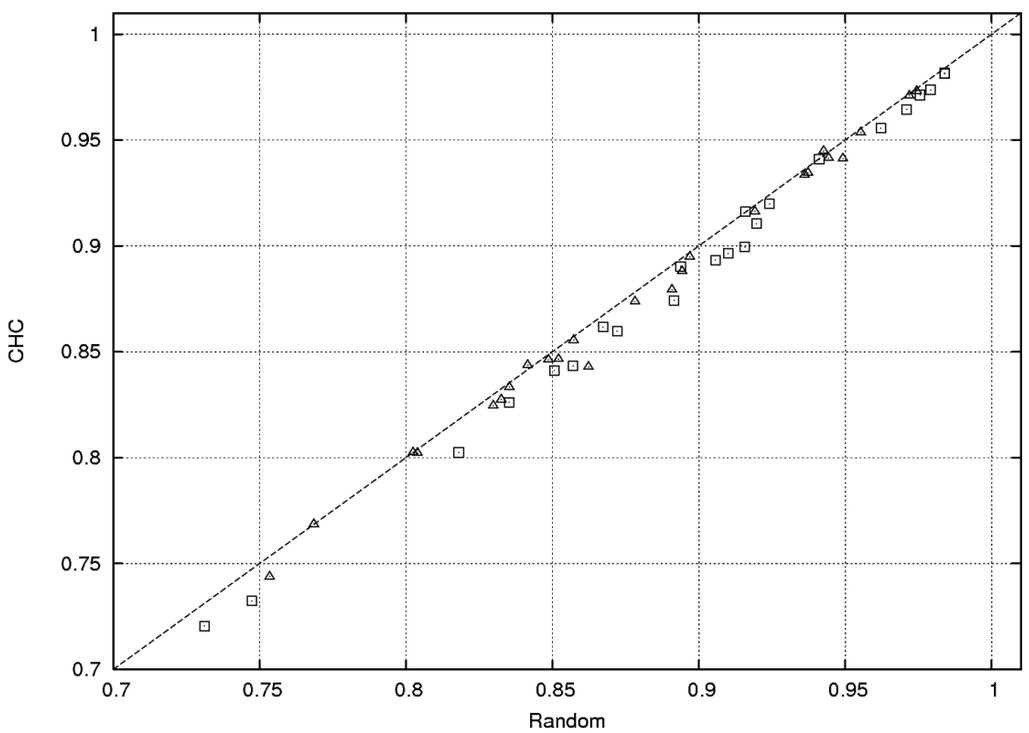


Fig. 6. Average generalization binary testing error of all the base learners for each dataset for random and CHC codes, using a C4.5 decision tree. Errors for dense codes (triangles) and sparse codes (squares).

dataset, with error for random codes in *x*-axis and error for CHC codes in *y*-axis. The figure shows the error for both dense and sparse codes. The results strongly support the hypothesis that the binary problems induced by codes designed by their error correcting capabilities are more difficult. Almost all the points are below the main diagonal, showing a general advantage of random codes. As the previous experiments failed to show a clear improvement of CHC codes over random ones, it is clear that the fact that the binary performance of the former is worse may be one of the reasons.

In order to assure the differences shown in the figure we performed a Wilcoxon test. The test showed that the differences are significant for both, dense and sparse codes, as a significance level of 99%.

In the same way we have compared the binary performance of *ovo* and *ova* methods. First, we must take into account that this comparison must be cautiously taken, as we are comparing the error of problems that are different. The results are shown in Figure 7, for a C4.5 decision tree, a support vector machine and a neural network as base learners.

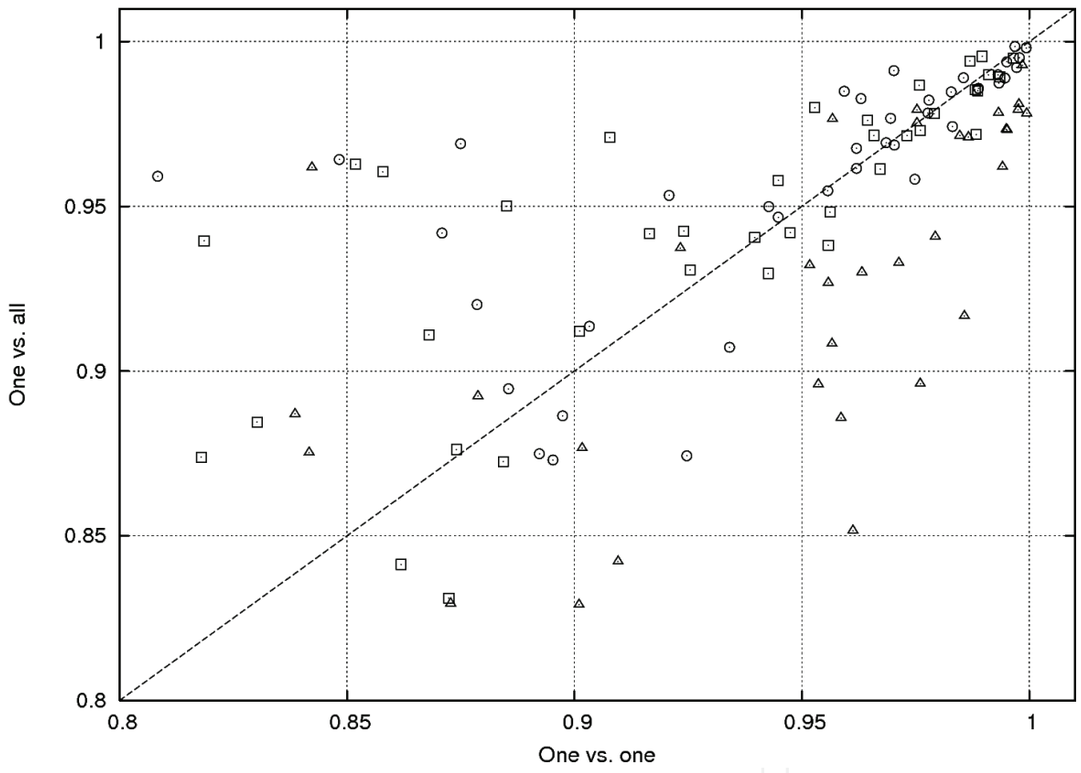


Fig. 7. Average generalization binary testing error of all the base learners for each dataset for *ovo* and *ova* methods, using a C4.5 decision tree (triangles), a support vector machine (squares) and a neural network (circles).

In this case, the results depend on the base learner used. For C4.5 and support vector machines, there are no differences, as it is shown in the figure and corroborated by Wilcoxon test. However, for neural networks the figure shows a clear smaller error of *ovo* method. The difference is statistically significant for Wilcoxon test at a significance level of 99%.

We must take into account that, although separating two classes may be easier than separating a class for all the remaining classes, the number of available patterns for the

former problem is also lower than the number of available patterns for the latter. In this way, this last problem is more susceptible to over-fitting. As a matter of fact, binary classifiers training accuracy is always better for one-vs-one method. However, this problem does not appear when using a neural network, where one-vs-one is able to beat one-vs-all in terms of binary classifier testing error. As in previous experiments, C4.5 seems to suffer most from small training sets.

It is noticeable that for some problems, namely abalone, arrhythmia, audiology, and primary-tumor, the minimum testing accuracy of the binary classifiers for one-vs-one method is very low. A closer look at the results shows that this problem appears in datasets with many classes. For some pairs, the number of patterns belonging to any of the two classes is very low, yielding to poorly trained binary classifiers. These classifiers might also have a harmful effect on the overall accuracy of the classifier. This problem does not arise in one-vs-all methods, as all binary classifiers are trained with all the data.

7. Conclusions

In this chapter, we have shown the available methods to convert a *k* class problem into several two class problems. These methods are the only alternative when we use classification algorithms, such as support vector machines, which are specially designed for two class problems. But, even if we are dealing with a method that can directly solve multiclass problems, we have shown that a class binarization can be able to improve the performance of the native multiclass method of the classifier.

Many research lines are still open, both in the theoretical and practical fields. After some recent works on the topic (García-Pedrajas & Fyfe, 2008) (Escalera et al., 2008) it has been shown that the design of the *ecoc* codes and the training of the classifiers should be coupled to obtain a better performance. Regarding the comparison among the different approaches, there are still many open questions, one of the most interesting is the relationship between the relative performance of each method and the base learner used, as contradictory results have been presented depending on the binary classifier.

8. References

Allwein, E. L., Schapire, R. E. & Singer, Y (2000). Reducing multiclass to binary: A unifying approach for margin classifiers, *Journal of Machine Learning Research*, vol. 1, pp. 113-141.

Anand, R., Mehrotra, K. G., Mohan, C. K. & Ranka, S. (1992). Efficient classification for multiclass problems using modular neural networks, *IEEE Trans. Neural Networks*, vol. 6, pp. 117-124.

Bauer, E. & Kohavi, R. (1999). An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants, *Machine Learning*, vol. 36, pp. 105-139.

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers, *Proceedings of the 5th Annual ACM Workshop on COLT*, pp. 144-152, D. Haussler, Ed.

Clark, P. & Boswell, R. (1991). Rule induction with CN2: Some recent improvements, *Proceedings of the 5th European Working Session on Learning (EWSL-91)*, pp. 151-163, Porto, Portugal, Spinger-Verlag.

Cohen, W. W. (1995). Fast effective rule induction, In: *Proceedings of the 12th International Conference on Machine Learning (ML-95)*, Prieditis A. & Russell, S. Eds., pp. 115-123, Lake Tahoe, CA, USA, 1995, Morgan Kaufmann.

Dietterich, T. G. & Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes, *Journal of Artificial Intelligence Research*, vol. 2, pp. 263-286.

Demšar, J. (2006). Statistical Comparisons of Classifiers over Multiple Data Sets, *Journal of Machine Learning Research*, vol. 7, pp. 1-30.

Escalera, S., Tax, D. M. J., Pujol, O., Radeva, P. & Duin, R. P. W. (2008). Subclass Problem-Dependent Design for Error-Correcting Output Codes, *IEEE Trans. Pattern Analysis and Machine Intyelligence*, vol. 30, no. 6, pp. 1041-1054.

Fürnkranz, J. (2002). Round robin classification, *Journal of Machine Learning Research*, vol. 2, pp. 721-747.

García-Pedrajas, N. & Fyfe, C. (2008). Evolving output codes for multiclass problems, *IEEE Trans. Evolutionary Computation*, vol. 12, no. 1, pp. 93-106.

García-Pedrajas, N. & Ortiz-Boyer, D. (2006). Improving multiclass pattern recognition by the combination of two strategies, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 1001-1006.

Hastie, T. & Tibshirani, R. (1998). Classification by pairwise coupling, *The Annals of Statistics*, vol. 26, no. 2. pp. 451-471.

Hsu, Ch.-W. & Lin, Ch.-J. (2002). A Comparison of methods for support vector machines, *IEEE Trans. Neural Networks*, vol. 13, no. 2, pp. 415-425.

James, G. M. (2003). Variance and bias for general loss functions, *Machine Learning*, vol. 51, no. 2, 115-135.

Kim, H. & Park, H. (2003). Protein secondary structure prediction based on an improved support vector machines approach, *Protein Engineering*, vol. 16, no. 8, pp. 553-560.

Knerr, S., Personnaz, L. & Dreyfus, G. (1990). Single-layer learning revisited: A stepwise procedure for building and training a neural network, In: *Neurocomputing: Algorithms, Architectures and Applications*, Fogelman, J. Ed., Springer-Verlag, New York.

Knerr, S., Personnaz, L. & Dreyfus, G. (1992). Handwritten digit recognition by neural networks with single-layer training, *IEEE Trans. Neural Networks*, vol. 3, no. 6, pp. 962-968.

Masulli, F. & Valentini, G. (2003). Effectiveness of error correcting output coding methods in ensemble and monolithic learning machines, *Pattern Analysis and Applications*, vol. 6, pp. 285-300.

Ortiz-Boyer, D., Hervás-Martínez, C. & García-Pedrajas, N. (2005). CIXL2: A crossover operator for evolutionary algorithms based on population features, *Journal of Artificial Intelligence Research*, vol. 24, pp. 33-80.

Passerini, A., Pontil, M. & Frasconi, P. (2004). New results on error correcting output codes of kernel machines, *IEEE Trans. Neural Networks*, vol. 15, no. 1, pp. 45-54.

Platt, J. C., Cristianini, N. & Shawe-Taylor, J. (2000). Large margin DAGs for multiclass classification, In: *Advances in Neural Information Processing Systems 12 (NIPS-99)*, Solla, S. A., Leen, T. K. & Müller, K.-R. Eds., pp. 547-553, MIT Press.

Pujol, O., Radeva, P. & Vitriá, J. (2006). Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 1007- 1012.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, CA, USA.

Rifkin, R. & Klautau, A. (2004). In defense of one-vs-all classification, *Journal of Machine Learning Research*, vol. 5, pp. 101-141.

Schapire, R. E. (1997). Using output codes to boost multiclass learning problems, In: *Proceedings of the 14th International Conference on Machine Learning (ICML-97)*, Fisher, D. H. Ed., pp. 313-321, Nashville, TN, USA, 1997, Morgan Kaufmann.

Sejnowski, T. J. & Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text, *Journal of Complex Systems*, vol. 1, no. 1, pp. 145-168.

Windeatt, T. & Ghaderi, R. (2003). Coding and decoding strategies for multi-class problems, *Information Fusion*, vol. 4, pp. 11-21.

Pattern Recognition Techniques, Technology and Applications

Edited by Peng-Yeng Yin

ISBN 978-953-7619-24-4
Hard cover, 626 pages
Publisher InTech
Published online 01, November, 2008
Published in print edition November, 2008

A wealth of advanced pattern recognition algorithms are emerging from the interdiscipline between technologies of effective visual features and the human-brain cognition process. Effective visual features are made possible through the rapid developments in appropriate sensor equipments, novel filter designs, and viable information processing architectures. While the understanding of human-brain cognition process broadens the way in which the computer can perform pattern recognition tasks. The present book is intended to collect representative researches around the globe focusing on low-level vision, filter design, features and image descriptors, data mining and analysis, and biologically inspired algorithms. The 27 chapters coved in this book disclose recent advances and new ideas in promoting the techniques, technology and applications of pattern recognition.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen