Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 34 (2008) 673–682
Expert Systems with Applications
www.elsevier.com/locate/eswa

Hierarchical classifier with overlapping class groups

Igor T. Podolak *

Jagiellonian University, Institute of Computer Science, Faculty of Mathematics and Computer Science, Nawojki 11, Krakow, Poland

Abstract

In this paper a novel complex classifier architecture is proposed. The architecture has a hierarchical tree-like structure with simple artificial neural networks (ANNs) at each node. The actual structure for a given problem is not preset but is built throughout training.

The training algorithm's ability to build the tree-like structure is based on the assumption that when a weak classifier (*i.e.*, one that classifies only slightly better than a random classifier) is trained and examples from any two output classes are frequently mismatched, then they must carry similar information and constitute a sub-problem. After each ANN has been trained its incorrect classifications are analyzed and new sub-problems are formed. Consequently, new ANNs are built for each of these sub-problems and form another layer of the hierarchical classifier.

An important feature of the hierarchical classifier proposed in this work is that the problem partition forms overlapping sub-problems. Thus, the classification follows not just a single path from the root, but may fork enhancing the power of the classification. It is shown how to combine the results of these individual classifiers.

2000 MSC: 68T05; 68T30; 68T35

Keyword: Classifier neural networks

1. Introduction

One of the main problems in machine learning is classification, in which a model is able to assign correct class labels to unseen examples build upon a training set. Different classifiers are possible: decision trees, which are accurate and have some explanatory power but do not have high generalization rate, *i.e.*, are unable to predict well unseen examples; artificial neural networks (ANN) which provide high accuracy but predictions lack any explanation. It is however hard to find an optimal ANN architecture for a given problem, at least it is a time-consuming process whose successful outcome depends more on experience and luck than on clear rules. Clearly, there is a need for a classifier that will be easy and quick to build, accurate, with explicable predictions, fit for parallel implementation.

In this paper a hierarchical classifier architecture is proposed: the classifier is built of several ANNs organized in a tree-like structure. Each ANN at a single node acts as a *weak* classifier. The performance of a weak classifier is only slightly better than random guessing, *e.g.*, over 1/2 in case of 2 output classes (Friedman, Hastie, & Tibshirani, 2001; Haykin, 1999). Since the classifiers are weak, many examples are classified incorrectly. On the other hand, such classification carries an important information: if an example from true class $\mathscr{A}$ is frequently classified by a single ANN as class $\mathscr{B}$, then examples from classes $\mathscr{A}$ and $\mathscr{B}$ are probably similar and hard to distinguish. It is thus possible to partition the problem into several sub-problems composed of patterns from classes that are harder to discern. The hierarchical classifier can thus be built by analyzing classification performance of ANNs and is fully automatic.

* Tel.: +48 12 633 1324; fax: +48 12 634 1865.
E-mail address: uipodola@theta.uoks.uj.edu.pl
URL: www.ii.uj.edu.pl/~podolak.

0957-4174/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2006.10.007

This paper is organized as follows: first basic definitions are introduced, then the above assumption is substantiated, the algorithm is described in details, experimental results are presented at the end.

The proposed architecture in some aspects resembles a Hierarchical Mixture of Experts (HME) model (Friedman et al., 2001; Haykin, 1999), but shows substantial differences. In the HME a tree structure is built of gating networks in all except leaf nodes, and experts at the terminal nodes. Gating networks perform a soft K-way split, and the terminal nodes provide posterior distribution vectors $Pr(Y = y|X = x, \theta)$ that $Y = y$ provided input features vector $X = x$ and a vector of that network's parameters θ. The overall posterior distribution is computed (for a two-level tree) as

$$Pr(Y = y|X = x, \Phi) = \sum_{i=1}^{K} g_i(x, \gamma_i) \sum_{j=1}^{K} g_{j|i}(x, \gamma_{ij}) \times Pr(Y = y|X = x, \theta_{ij}) \quad (1)$$

where $g_i(x, \gamma_i)$ are the outputs of first-level gating networks, and $g_{j|i}(x, \gamma_{ij})$ are the second-level gating networks' outputs.

The proposed model and HME differ in the following aspects:

- there are no methods for fitting a (good) tree topology for the HME (Friedman et al., 2001);
- in the proposed model the tree structure is built basing on training abilities of the classifiers used at individual nodes which determine the final architecture (*i.e.*, the number of layers and the number of nodes in each), while in the HME the architecture is fixed;
- in the proposed model all nodes share the same machine learning model (although it is not a must);
- in the proposed model the tree branches are extended only as long as they enhance the classifiers accuracy, therefore, different branches can be of a different depth.

Both architectures share similar approach when combining individual classifications into the final one.

2. Problem definition and model

Definition 1. Given a set of training data (x_k, c_k), where each input $x_k \in \mathbb{R}^p$ and the output is nominal with values from a finite set $\mathscr{C} = \{\mathscr{C}^i\}_{i=1}^{K}$, a classifier is a rule

$$Cl : X \to \mathscr{C} \quad (2)$$

which assigns each example x into a class $Cl(x)$.

If the classifier is implemented with a machine learning algorithm it gives some estimate $\widehat{Cl}(x)$ in $\mathscr{C}$. A loss function L which maps errors, *i.e.*, differences between true values and those found by Cl onto real numbers, is minimized during machine learning.

To achieve high accuracy, the ANN usually needs a higher number of hidden neurons, ending up in high model complexity and often poor generalization rate. One possible solution is to build several ANN classifiers and average the outputs (Schapire, 1990; Tresp, 2001). Each classifier training may start from a different point, use a subtly different training set, etc. The combined output has better accuracy due to the limitation of the variance in combined classifier bias–variance decomposition (Friedman et al., 2001; Haykin, 1999). In most averaging/additive models known, like Bayesian, Bootstrap, EM, AdaBoost, etc, each of the classifiers solves the same problem, that is the set of output classes is identical (Friedman, Hastie, & Tibshirani, 2000; Haykin, 1999; Tibshirani & Efron, 1993).

A different approach pursued by *e.g.*, decision trees, is via problem decomposition. The basic problem, depending on some feature values, is partitioned into distinct parts, then the machine learning algorithm applies the same reasoning to the sub-problems corresponding to each of the parts found (Norvig & Russell, 1995; Quinlan, 1993).

In the presented solution the training algorithm searches for sub-problems of the current one which appear to be "hard" to solve. Then the whole problem is split into sub-problems and the training is repeated. It is important to note that the "hard" sub-problems are found through observation of the current results of the classification. This is done through clustering parent classifier classification results. So it is in fact an additive model, like boosting (Friedman et al., 2001).

In the presented model, sub-classifiers are built that perform classifications of the original vectors of attributes into subsets of the set of all possible classes $\mathscr{C}$. The subsets may *overlap*, *e.g.*, an input vector x may be categorized by 2 different sub-classifiers into subsets of classes $\mathscr{C}^1$ and $\mathscr{C}^2$, such that $\mathscr{C}^1 \cap \mathscr{C}^2 \neq \emptyset$. Each sub-classifier returns a probability distribution vector of posterior probabilities of x being from any of the classes in each of the groups and posterior probabilities $Pr(Y \in \mathscr{C}^1|x)$ and $Pr(Y \in \mathscr{C}^2|x)$ (that is for x being from one of the classes in $\mathscr{C}^i$). Thus, a modified Bayesian rule may be applied to compute the final class of the original problem.

The presented model has a tree structure with a classifier Cl^i at each node

$$Cl^i : X^i \to \mathscr{C}^i \quad (3)$$

where $\mathscr{C}^i = \{\mathscr{C}_{i1}, \ldots, \mathscr{C}_{ik}\}$ is a subset of classes of examples from $X^i \subset X$, and X is the original set of examples defining the whole problem. To have a quick algorithm, a simple ANN is used to realize Cl^i, which results in Cl^i being *weak* with an error rate required only to be slightly better than random.

If a classifier Cl^i frequently confuses some classes, then it is presumed that the examples from these classes are harder for Cl^i to distinguish. Therefore, they are combined into groups using algorithms described later. If m groups (clusters) of classes were found

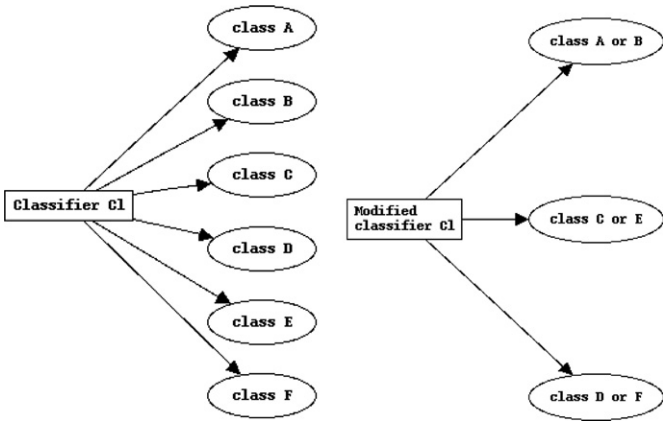

Fig. 1. An example of original Cl and a modified classifier Cl_{mod}.

$$Q^{ij} = \{\mathscr{C}_{il} \in \mathscr{C}^i | l = 1, \ldots, n_{Q^{ij}}\} \quad j = 1, \ldots, m \tag{4}$$

$$Q^i = \{Q^{ij} | j = 1, \ldots, m\} \tag{5}$$

where $\mathscr{C}^i \subset \mathscr{C}$ and $n_{Q^{ij}}$ is the number of classes in Q^{ij} which form Q^i, then the original classifier Cl^i is replaced with classifier Cl^i_{mod}

$$Cl^i_{\mathrm{mod}} : X^i \to Q^i \tag{6}$$

An example is shown in Fig. 1. At a given non-leaf level of the classifier tree, each of the individual classifiers Cl^i_{mod} actually categorizes examples into groups of original problem classes Q^{ij} which may overlap (it will be shown later how a group is chosen). A set of training examples X^i is split into subsets X^{ij} that include all examples with true classes in Q^{ij}. Thanks to that the training time, often prohibitive in case of large data sets, high number of classes, and large networks, is reduced. Classifiers in lower layers categorize into small class subsets and are trained using small training sets, therefore, the training of each is faster. At the same time, each classifier is trained *only* to achieve a *weak* level of accuracy. Thanks to that, the root classifier, which is the only one trained using the whole training set, is trained much quicker than it would be in case of a single net classifier.

On the other hand, there are many small classifiers in the whole tree, but the overall time requirements are favorable. The end classifications obtained at the leaf nodes are then *passed back* to the root node with class distributions for a given instance recalculated. This is where the algorithm differs from my previous proposition (Podolak, Biel, & Bobrowski, 2005). The accuracy of the whole classifier grows with each level of the tree.

3. The generation of sub-problems

Central to the proposed solution is the hypothesis, that if a given classifier Cl repeatedly categorizes examples from class $\mathscr{A}$ similarly to examples from some other class $\mathscr{B}$, then examples form classes $\mathscr{A}$ and $\mathscr{B}$ are *similar*, that is the classifier Cl finds them similar. In the context of the classification task, this amounts to the *inability* of Cl to properly distinguish examples from these classes. Therefore, it is proposed to form a *sub-problem* out of them.

Definition 2. Let D be a set of N training examples

$$D = \{(x^{(i)}, c^{(i)}) | i = 1, \ldots, N\} \tag{7}$$

where $x^{(i)}$ is a vector of input attributes, and $c^{(i)} \in \mathscr{C}$ is one of K classes. Let Cl be a classifier trained using D. After training, Cl returns a probability distribution vector $Pr(\mathscr{C}|x)$ of example x belonging to each of possible classes.

Let $D^{(m)}$ be the set of $N_{D^{(m)}}$ examples with true class $\mathscr{C}_m \in \mathscr{C}$. Then

$$M^{(m)} = \frac{1}{N_{D^{(m)}}} \sum_{(x,c) \in D^{(m)}} Pr(\mathscr{C}|x) \tag{8}$$

is the *prototype* vector of class distribution vectors $Pr(\mathscr{C}|x)$ found by Cl for training examples with true class $\mathscr{C}_m$.

Definition 3. We say that two classes m and n are *Cl-similar* if the distance between two prototype vectors $d(M^{(m)}, M^{(n)})$, where $d(\cdot,\cdot)$ is some similarity measure, is small.

The similarity measure $d(\cdot,\cdot)$ used in Eq. (3) defines the similarity of class approximations $\widehat{Cl}(x)$, therefore, examples are clustered not in the input features space (which is usual) but in the class space *as seen* by Cl. In other words, examples are clustered according *not* to their features or true classes but according to classes they are frequently classified as by Cl. Thanks to that, the problem partition algorithm divides the space of all examples into hard subproblems, solved later by individual classifiers.

The probability distribution vector $Pr(\mathscr{C}|x)$ gives the probabilities of all classes given example input features vector x. If $p_k(x) = Pr(Y = \mathscr{C}_k|x)$, then the actual response $Cl(x)$ is found with

$$Cl(x) = \mathscr{C}_k \quad \text{where } k = \arg\max_j p_j(x) \tag{9}$$

The $M^{(k)}, k = 1, \ldots, K$ correspond to the rows of the *confusion matrix* M of Cl with classifications found with rule (9), as shown in Fig. 2 approximated over the training set. Therefore, the sub-problems can be generated by clustering the rows of the confusion matrix M. Each cluster is a group of classes that are Cl-similar. Simple clustering creates non-overlapping clusters. Overlapping clusters are created with a clustering approach described in Section 4.

4. The hierarchical partition algorithm

The root classifier for the vowel problem, as shown in Fig. 2, classifies many examples incorrectly. It can easily be seen that examples from classes a and b are frequently mistaken with each other, then a cluster of examples with true classes a and b is constructed, and a classifier trained only to discern these two classes, which helps the overall classification. A simple clustering algorithm SAHN produced 3 clusters which *did not* overlap

a	b	c	d	e	f	g	h	i	j	k	<- classified as
77	7	0	0	0	0	0	0	0	6	0	a (true
0	74	15	0	0	0	1	0	0	0	0	b class)
0	3	85	0	0	1	1	0	0	0	0	c
0	0	2	83	0	1	0	0	0	0	4	d
0	0	0	0	78	9	2	0	0	0	1	e
0	0	1	9	22	45	2	0	0	0	11	f
0	0	5	0	18	0	65	2	0	0	0	g
0	0	0	0	0	0	0	73	0	17	0	h
0	0	0	0	0	1	0	2	19	63	5	i
0	3	0	0	0	0	1	1	4	81	0	j
0	0	0	1	0	7	4	0	10	0	68	k

Fig. 2. Confusion matrix CM for the root ANN classifier Cl with rule (9) used for the vowel problem (Blake et al., 1998). The ANN had only four hidden neurons. Beside each class label the number of examples in the training stream is given.

$$(h,i,j,k), \quad (d,e,f,g), \quad (a,b,c) \tag{10}$$

Each is now treated as a sub-problem and a new sub-classifier is built. Clusters found represent sub-problems that were *hard* for the root classifier.

The classifiers are organized in a tree. It is important to note that even if examples were classified incorrectly by the root classifier, then these classifications could be revised at a lower level. For example, if an example with true class a gets classified at the root node as b, then the classification might be corrected since a new classifier would be built to discern examples from the cluster (a,b,c).

However, some problems arise if crisp subsets are generated. In the example in Fig. 2, some examples with true class f are frequently classified as k. Since f and k belong to *different* clusters there is no chance for classification revision! Therefore, a clustering algorithm that produces clusters that overlap to a large extent is proposed. Thanks to that, a misclassification at the root level can be corrected easily.

4.1. Confusion matrix based problem partition

Since the individual classifiers are weak, their classifications are often incorrect. This is because of their low complexity, *i.e.*, high model bias corresponding to classifier's inability to correctly approximate the problem. However the incorrect classifications are *not* random: some regularities within the training set are exhibited. It may be seen that classifier Cl outputs for groups of classes are similar; the classes are said to be Cl-similar. It is called the *clustering effect*. The groups are found through inspection of the confusion matrix.

Definition 4. For a set of N training examples D

$$D = \{(x^{(i)}, c^{(i)}) | i = 1, \ldots, N\} \tag{11}$$

where $x^{(i)}$ is a vector of input attributes, and true class $c^{(i)} \in \mathscr{C}$ is one of K classes, the element $CM[i][j] = a$ of the *confusion matrix* CM is the number of examples from true class $\mathscr{C}_i$ classified with Cl as $\mathscr{C}_j$.

This definition assumes that a rule like that in Eq. (9), or similar, is used to select the final class given the class distribution vector computed with Cl. If classification has been done perfectly, then the confusion matrix is diagonal.

A different approach would be to use the class distribution vectors directly, without the use of a selection rule. This is a better approach because of the weak nature of classifier Cl: the probabilities of different classes for a given feature vector are frequently almost equal but the selection rule chooses only one. Therefore, the *distribution confusion matrix* is introduced.

Definition 5. For a set of N training examples D

$$D = \{(x^{(i)}, c^{(i)}) | i = 1, \ldots, N\} \tag{12}$$

the *distribution confusion matrix* DCM can be defined as a list of rows

$$DCM[i] = \frac{1}{N_{D^{(i)}}} \sum_{(x,c) \in D^{(i)}} Pr(\mathscr{C}|x) \tag{13}$$

where $D^{(i)}$ is the set of examples from class $\mathscr{C}_i$ and $Pr(\mathscr{C}|x)$ is the class probability distribution vector returned by Cl given example x.

An example of distribution confusion matrix DCM corresponding to matrix CM in Fig. 2 is given in Fig. 3. Since the selection rule is not used now, the DCM elements may have non-zero values even if some class $\mathscr{C}_i$ is never selected for any example from true class $\mathscr{C}_j$, *e.g.*, compare values for examples from true class a being classified as i in Figs. 2 and 3. Even for a perfect classifier Cl, whose CM matrix is strictly diagonal, the distributed matrix DCM does not need to be diagonal. This is because non-linearity was introduced by the selection rule *before* CM was computed. Each value in a normalized distribution confusion matrix can be treated as a posterior probability of class prediction given input example.

It can easily be shown that clusterization of DCM matrix gives better results than that of CM. This is because of the use of a selection rule (*e.g.*, Eq. (9)), some information about class similarities found by classifier Cl is lost. This is often the case if Cl is very weak and the selection is based on very weak grounds. If DCM is used, all classifier predictions are taken into consideration.

4.2. Generalized accuracy

If an example is incorrectly classified at the root level (or any paternal level), and assigned to a cluster that does *not* include the true label, then the classifiers at lower levels have no chance of doing a correct classification, *e.g.*, examples from class f assigned to cluster (h,i,j,k).

On the other hand, if clusters were overlapping, *i.e.*, some classes belonged to more than one cluster, then there

a	b	c	d	e	f	g	h	i	j	k	<- class. as
.88	.05	.0	.0	.0	.0	.0	.01	.02	.04	.01	a (true
.04	.77	.15	.0	.0	.0	.02	.0	.0	.01	.0	b class)
.0	.19	.79	.0	.0	.01	.01	.0	.0	.0	.0	c
.0	.0	.02	.79	.01	.16	.0	.0	.0	.0	.03	d
.0	.0	.0	.01	.5	.28	.19	.0	.0	.0	.02	e
.0	.0	.01	.1	.28	.44	.05	.0	.0	.0	.12	f
.0	.02	.04	.0	.18	.18	.52	.03	.0	.0	.04	g
.0	.0	.0	.0	.0	.0	.04	.78	.08	.09	.02	h
.02	.0	.0	.0	.0	.01	.03	.1	.35	.35	.14	i
.05	.04	.0	.0	.0	.0	.04	.11	.28	.41	.07	j
.0	.0	.0	.03	.03	.12	.04	.0	.09	.04	.64	k

Fig. 3. Distribution confusion matrix *DCM* for the root ANN classifier for the vowel problem with normalized rows.

would still be the possibility of a correct final classification. The notion of *generalized accuracy* is helpful:

Definition 6. The generalized accuracy is the rate of examples which are assigned by classifier *Cl* into a cluster including examples' true classes.

In the vowel example (see confusion matrix in Fig. 2) examples from true class *f* were classified correctly 45 times out of 90 (50% accuracy). The first clustering gave non-overlapping clusters (h,i,j,k), (d,e,f,g), (a,b,c), and algorithm described below extended clusters into overlapping groups

$$(h,i,j,k,f), \quad (d,e,f,g,k), \quad (a,b,c,g,j) \tag{14}$$

Since, during training class *f* was classified also as *c*, *d*, *e*, *g* and *k*, the generalized accuracy for class *f* will now be 99.89% with only one *f* example classified into a cluster with no *f*. Class *f* was not added to the third cluster, meaning that if any example with true class *f* would be classified as *a*, *b*, or *c*, then the correct classification would not be recovered. The actual algorithm follows *several* paths, *i.e.*, if an example gets classified by *Cl* as *f*, then both classifiers built for the first and second clusters are used. Therefore, when the generalized accuracy is high, the chances for erroneous classification remain high.

Clustering has two main objectives:

- produce several small clusters, so that simple ANN's would be able to solve them;
- make the clusters overlap to achieve a high generalized accuracy value.

Since these objectives might be contradictory, a smart clustering approach is needed.

The above root classifier is weak. The algorithm is very stable with different parameters since it achieves good results with very weak classifiers, *i.e.*, those with accuracy rates only a bit better than random. Therefore, less expert knowledge is needed to build new architectures.

4.3. Class clustering

The first step in clustering is simple clustering using the Sequential Agglomerative Hierarchical Nesting (SAHN) (Sokal & Sneath, 1973) algorithm which, by a bottom-up procedure, finds non-overlapping clusters. Then the algorithm checks the *column* normalized confusion distribution matrix *CNDCM* (the *CNDCM* for the vowel problem is given in Fig. 4).

An element $CNDCM[p][q]$ gives the posterior probability $Pr(X=p|Y=q,x)$ of a given example x being from true class p given the classifier *Cl* predicts class q. For example, in Fig. 4, it can be seen that there is a high chance that an example is from true class *f* if it is classified as *k*, since the value is 0.11 (in the vowel training set there are 11 possible classes with equal number of examples, thus the approximated prior probability of a class is $1/11 < 0.11$).

In other words, for a classifier Cl^i and non-overlapping clusters

$$Q^{ij} = \{\mathscr{C}_{il} \in \mathscr{C}^i | l = 1,\ldots,n_{Q^{ij}}\} \quad j = 1,\ldots,m \tag{15}$$

(see Eq. (5)), it can be seen that if an example from true class p and a cluster Q^{ik} such that $p \notin Q^{ij}$ any of the values

$$CNDCM[p][q], \quad p \notin Q^{ik},\ q \in Q^{ik} \tag{16}$$

is high, then class p is *Cl*-similar to classes in Q^{ik}. If such is the case, the clustering algorithm adds p to Q^{ik} constructing overlapping clusters.

In the vowel set example, the starting non-overlapping clusters $(h,i,\ j,k)$, (d,e,f,g), (a,b,c) are extended, accordingly, with class *f* (high $CNDCM[f][k]$ value), class *k* (high $CNDCM[k][f]$ value), and classes *g* and *j* (high $CNDCM[g][c]$ and $CNDCM[j][a]$ values) resulting with now overlapping clusters (h,i,j,k,f), (d,e,f,g,k), (a,b,c,g,j).

Since the clusters now overlap heavily, this approach gives a high generalized accuracy, and thus also high final accuracy. In the current solution, the number of classes in each cluster is not to be higher than 1/4 of classes recognized by the parent classifier. This simple solution gives

a	b	c	d	e	f	g	h	i	j	k	<- class. as
.88	.04	.0	.0	.0	.0	.0	.01	.03	.04	.01	a (true
.04	.72	.15	.0	.0	.0	.02	.0	.0	.01	.0	b class)
.0	.18	.78	.0	.0	.01	.01	.0	.0	.0	.0	c
.0	.0	.02	.85	.01	.13	.0	.0	.0	.0	.02	d
.0	.0	.0	.01	.5	.24	.2	.0	.0	.0	.01	e
.0	.0	.01	.11	.28	.37	.05	.0	.0	.0	.11	f
.0	.02	.04	.0	.18	.15	.56	.03	.0	.0	.03	g
.0	.0	.0	.0	.0	.0	.04	.76	.09	.09	.02	h
.02	.0	.0	.0	.0	.01	.03	.1	.43	.37	.13	i
.05	.04	.0	.0	.0	.0	.04	.11	.34	.44	.07	j
.0	.0	.0	.03	.03	.1	.04	.0	.11	.04	.6	k

Fig. 4. Column normalized confusion matrix for the root ANN classifier for the vowel problem. The values correspond to the probability of the true class (row index) given prediction (column).

satisfactory results, but a better supervised clustering algorithm needs to be found.

Other machine learning algorithms (like decision trees), which also divide the example space *but* into crisp subproblems, suffer from poor generalization ability. If the hierarchical classifier is built using artificial neural networks (ANNs), then the use of overlapping clusters can give both high accuracy and generalization, together with a hierarchical structure.

The whole classifier tree is shown in Fig. 5. The construction of the Hierarchical Classifier HC ends when classifiers at leaf nodes gain high accuracy, or the number of classes recognized falls to minimum (2 in the example). It is possible that sub-trees for some of the sub-problems are of different depths. In other words, the HC algorithm fits the architecture to the problem.

4.4. Combination of individual networks' results

After the HC is built, new feature vectors need to be classified. For a given feature vector x, the root classifier gives the class distribution vector $Pr(\mathscr{C}|x)$. Now the clusters and corresponding sub-classifiers to be chosen. This can be done in one of the following ways:

SINGLE PATH only one sub-classifier corresponding to the highest value in $Pr(\mathscr{C}|x)$ is chosen; this results in a single path from the root but classifications might still be corrected.

SEVERAL PATHS when only some of the paths in the tree are used: for a given example the node classifier Cl^i selects a set of clusters

$$Cl^i : X^i \rightarrow 2^{\mathscr{Q}^i} \tag{17}$$

the clusters are selected on the basis of the notion of *measure of belief mb*() borrowed from expert system methodology (Giarratano & Riley, 1994): only classes that have prediction value *higher* than their prior probability (approximated over the training set), *i.e.*, with positive *mb*(), are taken into account and then algorithm uses one of two modes:

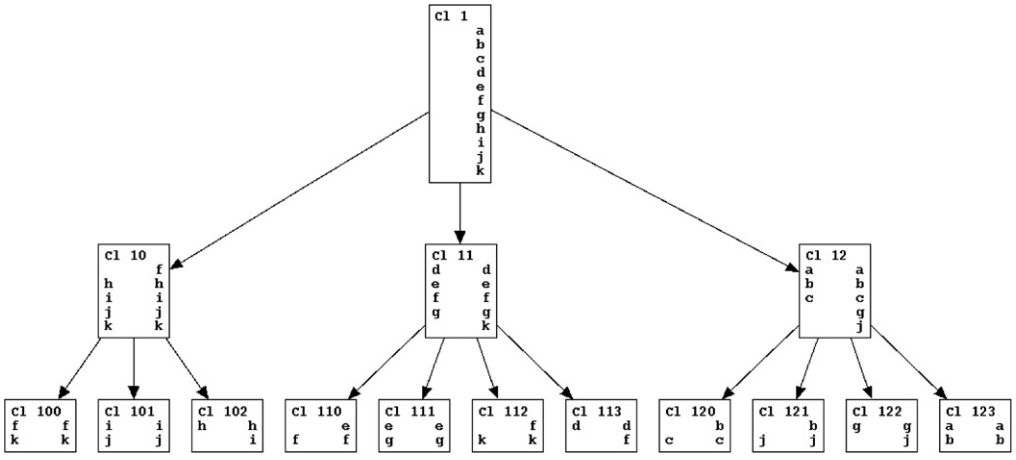

Fig. 5. Whole classifier tree for the vowel problem. Letters on the right correspond to classes recognized by an individual classifier (from overlapping clusters), while these on the left to non-overlapping clusters.

ACCEPTED all sub-classifiers that include classes with measure of belief $mb() > 0$ in their non-overlapping clusters are used, or
RECOGNIZED all sub-classifiers that include classes with measure of belief $mb() > 0$ in their overlapping clusters are used.
WHOLE TREE where all paths in the tree are followed: in that case some classes ranked low in the first classification can induce higher error.

Different paths chosen during classification of the same example with the same classifier trained can be seen in Fig. 6. The RECOGNIZED mode utilizes more sub-classifiers and gives better results because of a more thorough check through the solution space. If examples are simple, then only few classes have their probabilities higher than their prior probability, hence less sub-classifiers are chosen, and the whole classification path is simpler.

After a node classifier Cl^i is trained, m clusters

$$Q^{ij} = \{\mathscr{C}_{il} \in \mathscr{C}^i | l = 1, \ldots, n_{Q^{ij}}\}, \quad j = 1, \ldots, m \tag{18}$$

are found, where $n_{Q^{ij}}$ is the number of classes in cluster Q^{ij}. For each cluster Q^{ij} the following value needs to be computed

$$PQ(Q^{ij}|x) = \sum_{\mathscr{C}_{il} \in Q^{ij}} Pr(\mathscr{C}_{il}|x) \tag{19}$$

which is the sum of individual posterior probabilities of all classes in Q^{ij} computed for the given x. Since the clusters overlap, the overall sum $\sum_{j=1}^{m} PQ(Q^{ij}|x)$ is greater than 1, therefore, the vector

$$Pr(Q^i|x) = [PQ(Q^{i1}|x), \ldots, PQ(Q^{im}|x)] \tag{20}$$

where m is the number of sub-clusters, needs to be normalized

$$Pr(Q^{ij}|x) = Pr(Q^{ij}|x) / \sum_{j=1}^{m} PQ(Q^{ij}|x) \tag{21}$$

so that vector $Pr(Q^i|x)$ might be treated as a cluster distribution vector.

During actual classification of a feature vector x, it is recursively assigned to a cluster which represents a sub-problem, and a sub-classifier for that problem is run. When a terminal node classifier is reached, the probabilities for classes at that level are found and *back-propagated* to the root. Thus, the final class probability distribution at the root of each sub-tree is computed recursively with the following formula

$$Pr(\mathscr{C}|x) = \sum_{i=1}^{m} Pr(Q^i|x) Pr_{Q^i}(\mathscr{C}^i|x) \tag{22}$$

where $Pr(Q^i|x)$ are the computed cluster posterior probabilities, and $Pr_{Q^i}(\mathscr{C}^i|x)$ are the individual class probabilities for the sub-problem composed of classes within Q^i.

After a non-terminal node classifier Cl^i is run, two distribution vectors are computed

- class distribution vector $Pr(\mathscr{C}^i|x)$;
- cluster distribution vector $Pr(Q^i|x)$.

Now *only* classifiers Cl^{ij} corresponding to $Q^{ij} \in Q^i$ clusters are run, which include classes $\mathscr{C}_k \in \mathscr{C}$ that have probability $Pr(\mathscr{C}_k|x)$ higher than the prior probability of that class $Pr(\mathscr{C}_k)$, *i.e.*

$$\text{use} \quad Cl^{ij} \text{ only if} \quad Pr(\mathscr{C}_k|x) > Pr(\mathscr{C}_k) \wedge \mathscr{C}_k \in Q_{ij} \tag{23}$$

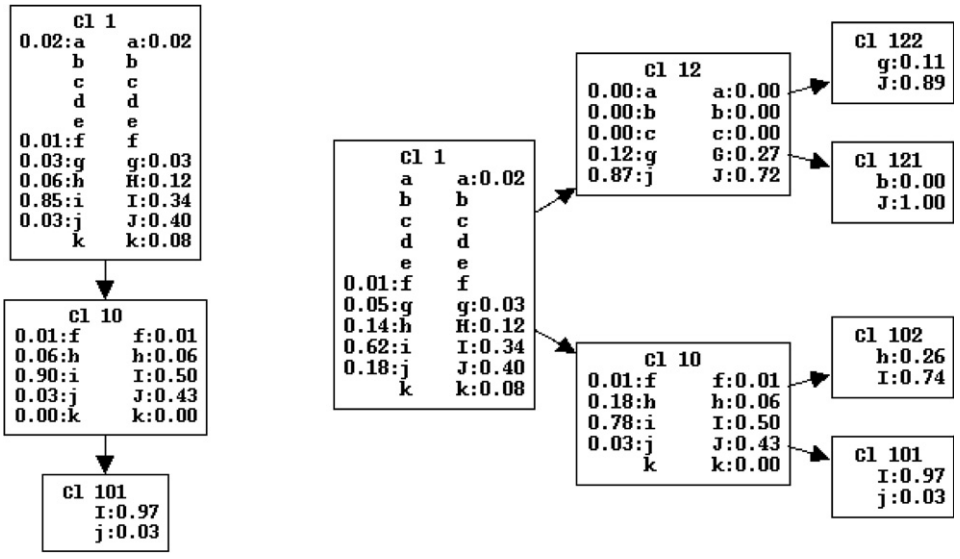

Fig. 6. Vowel problem classification paths for a single example in modes ACCEPTED and RECOGNIZED. Both correct the root classifier prediction (correct class is i while the root classifier predicts j). At each node the values on the right are the class distribution vector $Pr(\mathscr{C}|x)$ values. Class labels in capitals denote classes that had their measure of belief $mb() > 0$. The values on the left of each node are the HC classification values for its sub-tree calculated by the algorithm using Eq. (22). The final classification vector is given on the left of the root classifier.

This bears resemblance to the *measure of belief* used in expert systems (Giarratano & Riley, 1994). Only those subclassifiers are used which include classes that the paternal classifier assigned higher than average probability. The prior probability $Pr(\mathscr{C}_k)$ is approximated over the training set.

4.4.1. Neural network training

Artificial neural networks were used as classifiers at all nodes, although any type of classifier can be used with the architecture proposed. Each ANN had a single layer with a small number of neurons.

Each ANN is trained to minimize the augmented cross-entropy error function (24), as in the FERNN classifier of Setiono (Leow & Setiono, 2000).

$$\theta(w,v) = F(w,v) - \sum_{i=1}^{K}\sum_{k=1}^{|D|}[t_{ik}\log S_{ik} + (1-t_{ik}) \times \log(1-S_{ik})] \tag{24}$$

where each pattern x_k is labeled with one of the K possible classes. t_{ik} is the target value for pattern k ($k = 1,2,\ldots,|D|$) at output unit i. S_{ik} is the ANN's output unit value.

The ANN has J hidden units. $F(w,v)$ is a penalty function with positive parameters $\epsilon_1, \epsilon_2, \beta$ which are added to encourage weight decay.

$$F(w,v) = \epsilon_1 \sum_{j=1}^{J}\left(\sum_{i=1}^{K}\frac{\beta v_{ij}^2}{1+\beta v_{ij}^2} + \sum_{k=1}^{I}\frac{\beta w_{jk}^2}{1+\beta w_{jk}^2}\right) + \epsilon_2 \times \sum_{j=1}^{J}\left(\sum_{i=1}^{K} v_{ij}^2 + \sum_{k=1}^{I} w_{jk}^2\right) \tag{25}$$

where v_{ij} is the weight of the connection from hidden unit j to output unit i and w_{jk} is the weight of the connection from input unit k to hidden unit j. The penalty function causes irrelevant connections to have very small weights. Connections are removed beginning from the smallest one as long as network accuracy is acceptable (Leow & Setiono, 2000). The original algorithm was modified through the use of Resilient Backpropagation algorithm.

This connection reducing approach was used so that the individual ANN classifiers were kept weak but with a high generalization rate. If a network was to be complex, then it would be possible for it to "remember" some examples, therefore, reducing both generalization and generalized accuracy, which is crucial for classifiers overall accuracy. The FERNN type of learning was also used to extract rules for the whole clusters.

Thanks to that, the number of connections is reduced so that a rule-extraction proposed in Leow and Setiono (2000) can be started. During the experiments the ANN's usually had about 25% connections removed. The hidden layer size was set at all single node ANNs to

$$J = \lfloor\sqrt{in + (out)^2}\rfloor \tag{26}$$

where *in* is the number of input feature, and *out* the number of output classes. The number of output classes changes at each level.

5. Experiments

A number of tests was performed using benchmark files from the UCI Repository (Blake, Newman, Hettich, & Merz, 1998; Sokolic & Zwitter, 1973). The results were compared to those obtained by Setiono (Setiono, 2001) on networks with single hidden layer and the number of hidden neurons optimized to obtain the best accuracy. The results were also compared with those obtained by the first version of our classifier where only one path in the classifier tree was followed (Podolak et al., 2005). The comparison is given in Table 1.

The algorithm gives good accuracy rates. It needs to be underlined that the whole tree structure of the classifier is found completely automatically, without the need of trying out a number of architectures. In fact, all the single node ANNs had single hidden layer with a number of neurons set with the same formula (see Eq. (26)).

This approach is especially suited for applications where there is a high number of output classes (such problems were chosen for testing). In such cases the resulting classifiers, especially neural networks, tend to have a very high number of parameters and require expert knowledge for building problem specific knowledge into the classifier design. Usually such knowledge is not available, and the procedure of testing several architectures can be very long.

Table 1
Results of experiments with datasets from the UCI Repository (Blake et al., 1998; Sokolic & Zwitter, 1973) compared with those from Setiono (2001) (arrhythmia and isolet were not included there)

	Features	Classes	Accuracy		Setiono (Setiono, 2001)	
			Train	Test	Train	Test
Audiology	95	24	93.7 ± 1.17	77.0 ± 8.88	95.5 ± 1.14	79.5 ± 1.61
Arrhythmia	279	16	94.8 ± 0.77	67.6 ± 5.71		
Primary tumor	37	21	80.6 ± 1.49	42.8 ± 8.69	62.1 ± 1.41	46.0 ± 1.37
Vowel	28	11	97.9 ± 0.72	92.4 ± 3.06	94.3 ± 1.23	88.9 ± 1.46
Zoo	17	7	99.1 ± 0.69	95.1 ± 5.18	100.0 ± 0.00	94.3 ± 1.46
Soybean	135	19	98.7 ± 0.37	93.1 ± 2.19	96.4 ± 0.96	93.3 ± 0.67
Isolet	617	26	98.6 ± 0.21	95.41 ± 0.64		

All results from 10-fold cross-validation. Some missing feature values (only in audiology, primary-tumor, and soybean) were replaced with mean values.

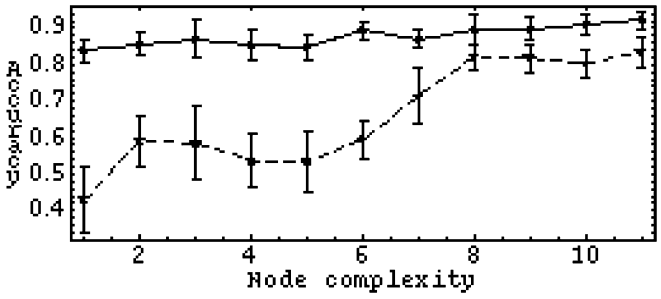

Fig. 7. Accuracy rates of the Hierarchical Classifier for the vowel problem for different complexities of individual classifiers. The bottom dashed line shows the accuracy of the root node classifier, the top thick line the accuracy of the full tree. The complexity of individual nodes grows from three hidden neurons on the left, to $\sqrt{in + out^2}$.

The proposed architecture provides a methodology of building strong classifiers as a combination of several weak ones.

Another important fact is that eventual accuracy rates obtained do not depend heavily on the accuracy of individual classifiers at the HC nodes. Therefore, these classifiers do not need to be complex (see Fig. 7). The final accuracy for very simple, therefore weak, individual classifiers is only slightly inferior to that for larger ANNs at individual nodes. It can also be seen, that accuracy grows with each level of the HC tree.

The whole application was written in Java to fit into the machine learning environment Weka (Frank & Witten, 2000).

6. Conclusions

The proposed methodology of building hierarchical classifier is based on the inspection into the accuracy of single node classifiers and extending the tree, so that errors are reduced. The approach tries to hierarchically split the original problem into simpler sub-problems. Sub-problems are defined as groups of classes that a node classifier had most problems to discern among.

The proposed notion of generalized accuracy helps to find clusters that minimize the probability of irrevocable errors at nodes near the root. This is made possible because the sub-problems overlap and an example classified at a given moment can belong to more than one cluster. Such example is then classified by two or more sub-classifiers. Eventual classification of the whole classifier depends on the confidences of all classifiers in their individual results.

A single example can, therefore, be classified in parallel by two or more branches. Actually, the classifier training algorithm was implemented to run in parallel, giving speed increase proportional to the number of processors used.

It is proposed that only branches that include classes with posterior probability higher than prior probability (approximated over the training set) are used. This corresponds to the measure of belief notion used in expert systems. In that way only sub-classifiers that have some knowledge are used.

7. Future work

The architecture proposed gives high accuracy and is especially suited for problems with a high number of output classes. One such application may be the prediction of electric power consumption. Although the proposed architecture is designed for classification, this problem may be solved using an approach similar to M5' algorithm (Frank & Witten, 2000). The predicted variable may divided into intervals transforming the approximation problem into a classification one. Then, simple regression approximators can be inserted into terminal nodes.

More things need now to be done, among them

- select the most appropriate clustering tool for the approach. A simple SAHN algorithm is used now, but an algorithm in which the generalized accuracy is maximized accomplishing better split into sub-problems is needed
- show that the HC is a universal classifier
- the FERNN algorithm was used since another objective of our work was to extract rules that would be human readable. It is possible to extract rules out of single node ANNs of the form

  IF(input features expression)THEN cluster i (27)

  Such rules would be much simpler than the rules that give specific classes as results. Rules extracted from single node classifiers might be fed into an expert system to extract similar classification results. The results already found are promising.
  In case of single ANNs, the extracted rules tend to be complicated, and the solution does not seem to be scalable – the more complicated the problem, the more complicated the extracted rules. The HC architecture gives a chance to produce a better readable hierarchical system of rules. A combined system with rules at internal nodes, and ANNs at leaf nodes is possible.
- The electricity prediction problem is one that is constantly modified. There are new readings from the electrical system typically every 15 min (such readings from the Polish Power Grid (PSE) for over 4 years are available). Once such a reading comes in, it should be included in the prediction system. The supervised training of feed-forward neural networks is not an incremental one, but a small number of observations can be added. It remains to be seen whether such incremental training, with new examples added, and the oldest removed is possible, and how that will change the classifier proposed here. As long as the single node classifiers are modified only minimally and *do not change the separation into clusters*, the whole architecture will not change much. On the other hand, if new examples change the separation into sub-problems, the whole branch (it does not need to be the whole tree) will have to be re-built.

References

Blake, C. L., Newman, D. J., Hettich, S., & Merz, C. J. (1998). *UCI repository of machine learning databases*.

Frank, E., & Witten, I. H. (2000). Data mining: practical machine learning tools with Java implementations. San Francisco: Morgan Kaufmann.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). *Annals of Statistics, 28*, 307–337.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning. New York: Springer Verlag.

Giarratano, J., & Riley, G. (1994). Expert systems, principles and programming. Boston: PWS.

Haykin, S. (1999). Neural networks, a comprehensive foundation. Upper Saddle River, NJ: Prentice Hall.

Leow, W. K., & Setiono, R. (2000). Fernn: and algorithm for fast extraction of rules from neural networks. *Applied Intelligence, 12*, 15–25.

Leow, W. K., & Setiono, R. (2000). FERNN: An algorithm for fast extraction of rules from neural networks. *Applied Intelligence, 12*(1–2), 15–25.

Norvig, P., & Russell, S. (1995). *Artificial intelligence, a modern approach. Prentice Hall series in Artificial Intelligence*. Prentice Hall, Inc.

Podolak, I., Biel, S., & Bobrowski, M. (2005). Hierarchical classifier. *Springer Lecture Notes on Computer Science, 3911*.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo: Morgan Kaufmann.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*, 197–227.

Setiono, R. (2001). Feedforward neural network construction using cross validation. *Neural Computation, 13*, 2865–2877.

Sokal, R. R., & Sneath, P. H. A. (1973). *Principles of numerical taxonomy*. San Francisco: Freeman.

Sokolic, M., Zwitter, M., Primary tumor data set, 1987.

Tibshirani, R., & Efron, B. (1993). *An introduction to the bootstrap*. London: Chapman and Hall.

Tresp, V. (2001). Committee machines. In *Handbook for neural network signal processing*. CRC Press.