KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 0318

AGGREGATING CLASSIFIERS WITH MATHEMATICAL PROGRAMMING

by
J. ADEM
W. GOCHET

D/2003/2376/18

Aggregating Classifiers with Mathematical Programming

Jan ADEM
Department of Applied Economics, Katholieke Universiteit Leuven,
Naamsestraat 69, 3000 Leuven, Belgium,
e-mail: jan.adem@econ.kuleuven.ac.be

Willy GOCHET
Department of Applied Economics, Katholieke Universiteit Leuven,
Naamsestraat 69, 3000 Leuven, Belgium,
e-mail: willy.gochet@econ.kuleuven.ac.be

Abstract

Traditionally, bagging takes a majority vote among a number of classifiers. An alternative is to aggregate the classifiers with a mathematical programming model. This approach guarantees that the aggregated classifier will not be worse than the best component classifier on a given criterion function on the design dataset. The approach is illustrated on three real world datasets and compared to traditional bagging.

1 Introduction

The supervised classification (SC) problem consists of finding a formal rule that classifies patterns with unknown class membership into one of a finite number of classes C as accurate as possible. Such a formal rule is called a classifier. Patterns are whatever needs to be classified. For each pattern, the values of P measurements are known. In order to design a classifier, a design dataset $\mathcal{D}$ consisting of a finite number of patterns N with known class membership is given. Let $x_{np} \in \mathrm{IR}$ with $n \in \{1, 2, ..., N\}$ and $p \in \{1, 2, ..., P\}$ be the value of measurement p of pattern n and $c_n \in \{1, 2, ..., C\}$ the class to which pattern n belongs. Overviews of the currently available methods for designing a classifier are given in textbooks such as [4, 8, 13].

This variety of SC methods gives rise to a simple question: can we find ways to aggregate classifiers such that the aggregated classifier has more desirable characteristics than any of the classifiers separately?

Given are a finite number of classifiers $g_1, g_2, ..., g_L$. These L classifiers are called the component classifiers. Let $g_{ln} \in \{1, 2, ..., C\}$ be the class that component classifier g_l assigns to design dataset pattern n. If $g_{ln} = c_n$, the pattern is correctly classified, otherwise it is misclassified by component classifier g_l. The design dataset error rate of classifier g_l is denoted by h_l. Assume without loss of generality that all $h_l > 0$ and that all the component classifiers are different, i.e. there exist no two classifiers g_{l_1} and g_{l_2} such that $g_{l_1 n} = g_{l_2 n}$ for all $n \in \{1, 2, ..., N\}$. Component classifiers can be obtained in several ways. One might apply different SC methods to the same design dataset or, alternatively, use the same SC method on sub-samples from the design dataset. These sub-samples can be taken with or without replacement.

Each component classifier will be given a weight α_l such that $\sum_{l=1}^{L} \alpha_l = 1$ and all $\alpha_l \geq 0$. Let $I(.)$ be the indicator function. $I(.) = 1$ if the argument is true and $= 0$ otherwise. The aggregated classifier classifies a pattern with measurements $x_1, ..., x_P$ into the class c^* for which $c^* = \arg\max_c \{\sum_{l=1}^{L} \alpha_l I(g_l(x_1, ..., x_P) = c)\}$ where $g_l(x_1, ..., x_P)$ denotes the class that component classifier g_l assigns to the pattern. Ties are solved randomly. $\sum_{l=1}^{L} \alpha_l I(g_l(x_1, ..., x_P) = c)$ can be seen as an estimate for $pr_C(\mathrm{C} = c \,|\, \mathrm{X}_1 = x_1, ..., \mathrm{X}_P = x_P)$, the conditional probability that a pattern with measurements $x_1, ..., x_P$ belongs to class c. Conditional probability estimates are

greatly appreciated in practice because they give an idea of the reliability of a prediction, e.g. in credit scoring [7, 11]. In majority vote (MV) aggregating, each component classifier is given equal weight, i.e. $\alpha_l = \frac{1}{L}$ for all $l \in \{1, 2, ..., L\}$ [3]. In AdaBoost.M1, the weight of the l-th component classifier is chosen as $\alpha_l = \log((1 - h_l)/h_l)$ [8]. The constraints $\sum_{l=1}^{L} \alpha_l = 1$ and all $\alpha_l \geq 0$ do not apply in AdaBoost.M1. However, this is not always the best choice for the weights α_l. It might be better to fix the values for the weights α_l according to some criterion function defined on the available data.

In Section 2, we will present and discuss the mathematical programming (MP) formulation that can be used to aggregate component classifiers. Ways to reduce the possible computational difficulties are discussed in Section 3. Section 4 illustrates how to use the approach on four real world datasets. The last section presents the conclusions.

2 Mathematical Programming Formulation

Associate with each design dataset pattern a binary variable $y_n = 1$ if pattern n is misclassified by the aggregated classifier, $= 0$ otherwise. Denote by $\mathcal{G}_{nc} = \{ l \mid g_l(x_{n1}, ..., x_{nP}) = c\}$ the set of component classifiers that classify pattern n in class c. The problem specific expense of misclassifying pattern n is e_n. ϵ is a user-defined constant, $0 \leq \epsilon \leq 1$. The aggregated classifier that minimizes the design dataset misclassification expense is found by solving the following mixed integer linear programming (MILP) formulation.

$$\text{MILP-}\epsilon: \min_{\alpha_l, y_n} \sum_{n=1}^{N} e_n y_n$$

subject to

$$\sum_{l \in \mathcal{G}_{nc_n}} \alpha_l - \sum_{l \in \mathcal{G}_{nc}} \alpha_l + (1+\epsilon) y_n \geq \epsilon \qquad n \in \{1, ..., N\} \quad c \in \{1, ..., C\} \quad c \neq c_n$$

$$\sum_{l=1}^{L} \alpha_l = 1$$

$$0 \leq \alpha_l \leq 1 \qquad l \in \{1, ..., L\}$$

$$\alpha_l \in \text{IR} \qquad l \in \{1, ..., L\}$$

$$y_n \in \{0, 1\} \qquad n \in \{1, ..., N\}$$

ϵ can be interpreted as the minimum difference there should be between the maximal conditional probability estimate and the second largest before the MILP-ϵ is allowed to consider the pattern as correctly classified. Setting $\epsilon = 0$ allows for ties in the maximal conditional probability estimates. For $\epsilon > 1$, the objective function value of any feasible solution is $\sum_{n=1}^{N} e_n$ and the formulation has no sense. As the MILP-ϵ problem always has a nonempty feasible region and cannot have an unbounded solution, an aggregated classifier that minimizes the design dataset misclassification expense always exists.

The MILP-ϵ formulation is a special case of a more general formulation. Define for each design dataset pattern C binary variables $y_{nc} = 1$ if pattern n is classified into class c by the aggregated classifier, $= 0$ otherwise. Let e_{nc} be the problem specific expense of classifying pattern n into class c. ϵ is the same as in the MILP-ϵ formulation.

general-MILP-ϵ : $\min_{\alpha_l, y_{nc}} \sum_{n=1}^{N} \sum_{c=1}^{C} e_{nc}\, y_{nc}$

subject to

$$\sum_{l \in \mathcal{G}_{nc}} \alpha_l - \sum_{l \in \mathcal{G}_{nd}} \alpha_l + (1+\epsilon)(1 - y_{nc}) \geq \epsilon \qquad n \in \{1, ..., N\} \quad c, d \in \{1, ..., C\} \quad c \neq d$$

$$\sum_{c=1}^{C} y_{nc} = 1 \qquad n \in \{1, ..., N\}$$

$$\sum_{l=1}^{L} \alpha_l = 1$$

$$0 \leq \alpha_l \leq 1 \qquad l \in \{1, ..., L\}$$

$$\alpha_l \in \mathrm{IR} \qquad l \in \{1, ..., L\}$$

$$y_{nc} \in \{0, 1\} \qquad n \in \{1, ..., N\} \quad c \in \{1, ..., C\}$$

In this model, the user is offered a large flexibility to model his SC problem. For example, suppose one wants to find the expense minimizing classifier given that at least 90% of the design dataset patterns in class 2 should be correctly classified and that none of the class 1 design dataset patterns are put in class 3. It suffices to add the constraints $\sum_{n \in \mathcal{D}_2} y_{n2} \geq 0.90 |\mathcal{D}_2|$ and $\sum_{n \in \mathcal{D}_1} y_{n3} \leq 0$ where $\mathcal{D}_c$ is the set of design dataset patterns that belong to class c, $\mathcal{D}_c = \{n \,|\, c_n = c\} \subset \mathcal{D}$. Note that adding such constraints might make the formulation infeasible. The price for this flexibility is that the models tend to be difficult to solve to optimality when the design dataset or the number of component classifiers gets larger.

In the remainder of this work, we will focus on the MILP-ϵ formulation using a simple 0-1 expense structure, i.e. $e_n = 1$ for all $n \in \{1, 2, ..., N\}$.

3 Pre-processing

Solving MILP-ϵ to optimality may be very time consuming, even impossible to solve within a reasonable time using classical branch and bound procedures. Therefore it is important to reduce the size of MILP-ϵ as much as possible. Especially reducing the number of binary variables y_n will be helpful in reducing the overall time to solve MILP-ϵ. Below a number of reduction rules are proposed that reduce considerably the difficulty of solving MILP-ϵ. This will be illustrated by the real world examples in Section 4.

Let ϵ be sufficiently small. Proposition 1 is a trivial observation.

> **Proposition 1 :** A design dataset pattern that is classified correctly (incorrectly) by all L component classifiers g_l will be classified correctly (incorrectly) by the aggregated classifier for all possible values of $(\alpha_1, \alpha_2, ..., \alpha_L)$.

A reduction rule follows directly from this proposition.

> **Reduction Rule 1 (RR1) :** All patterns that are classified correctly (incorrectly) by all component classifiers can be left out from MILP-ϵ.

When applying RR1, it is clear that the objective function value of MILP-ϵ has to be increased by 1 for every misclassified pattern that is left out. The optimal solution to MILP-ϵ is unaffected by RR1. Consider the following example with three classes, five component classifiers and three patterns that all belong to class 1. Let $(g_{11}, g_{21}, g_{31}, g_{41}, g_{51}) = (1, 1, 1, 1, 1)$, $(g_{12}, g_{22}, g_{32}, g_{42}, g_{52}) = (2, 2, 2, 2, 2)$ and $(g_{13}, g_{23}, g_{33}, g_{43}, g_{53}) = (3, 2, 3, 3, 2)$. For all feasible values of $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$, the first pattern will be classified correctly, the second and the third incorrectly.

A second reduction rule will be derived from Proposition 2.

> **Proposition 2 :** Let $g_{\tilde{l}}$ with $\tilde{l} = \arg\min_l\{h_l \mid l \in \{1, 2, ..., L\}\}$ be a component classifier with the smallest design dataset error rate among the L component classifiers. When component classifier k has $h_k > h_{\tilde{l}}$ then for an optimal solution $(\alpha_1^*, \alpha_2^*, ..., \alpha_L^*)$ to MILP-ϵ, it cannot hold that $\alpha_k^* > \sum_{l=1, l\neq k}^{L} \alpha_l^*$.

Proof : When $\alpha_k > \sum_{l=1,l\neq k}^{L} \alpha_l$, the aggregate classifier will classify the design dataset patterns in the same way as component classifier g_k, hence will have a larger design dataset error rate than component classifier $g_{\tilde{l}}$ which is a contradiction since $g_{\tilde{l}}$ provides a feasible solution to MILP-ϵ. (Q.E.D.)

Take a five component example with $(h_1, h_2, h_3, h_4, h_5) = (0.35, 0.36, 0.29, 0.32, 0.29)$. The lowest error rate is 0.29, hence $\tilde{l} \in \{3, 5\}$. The error rate of any solution for which $\alpha_1 > \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5$ is 0.35. Hence, such solution can never be optimal as $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) = (0,0,1,0,0)$ is a feasible solution with a lower objective function value. The same is true for all solutions for which $\alpha_2 > \alpha_1 + \alpha_3 + \alpha_4 + \alpha_5$ or $\alpha_4 > \alpha_1 + \alpha_2 + \alpha_3 + \alpha_5$.

Reduction Rule 2a (RR2a) : Assume classifiers $g_{\tilde{l}}$ and g_k as in Proposition 2. For any pattern n correctly classified by all component classifiers except for classifier k, it holds that $y_n^* = 0$ in an optimal solution of MILP-ϵ.

Proof : The restrictions for pattern n in MILP-ϵ include

$$\sum_{l=1,l\neq k}^{L} \alpha_l - \alpha_k + (1+\epsilon) y_n \geq \epsilon.$$

An optimal value $y_n^* = 1$ implies that $\alpha_k^* > \sum_{l=1,l\neq k}^{L} \alpha_l^*$. By Proposition 2, this leads to a contradiction. (Q.E.D.)

Again, take the five component example with $(h_1, h_2, h_3, h_4, h_5) = (0.35, 0.36, 0.29, 0.32, 0.29)$ as before but now let $(g_{11}, g_{21}, g_{31}, g_{41}, g_{51}) = (1, 1, 1, 3, 1)$. Pattern 1 belongs to class 1. Assume we classify the pattern incorrectly. This is only possible if $\alpha_4 > \alpha_1 + \alpha_2 + \alpha_3 + \alpha_5$. Any solution for which this is true has a design dataset error rate of 0.32 which cannot be optimal. Hence, in an optimal solution, it holds that $y_1^* = 0$. The constraint $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_5 - \alpha_4 \geq \epsilon$ stays in the formulation.

Reduction Rule 2b (RR2b) : Assume classifiers $g_{\tilde{l}}$ and g_k as in Proposition 2. For any pattern n incorrectly classified *in the same class* by all component classifiers except for component classifier k, it holds that $y_n^* = 1$ in an optimal solution of MILP-ϵ.

Proof : Two cases are possible.

(i) Component classifier k misclassifies pattern n. In that case Proposition 1 applies and $y_n^* = 1$.

(ii) Component classifier k classifies pattern n correctly. The restrictions for pattern n in MILP-ϵ include

$$\alpha_k - \sum_{l=1, l \neq k}^{L} \alpha_l + (1+\epsilon) y_n \geq \epsilon.$$

An optimal value $y_n^* = 0$ implies that $\alpha_k^* > \sum_{l=1, l \neq k}^{L} \alpha_l^*$. By Proposition 2, this leads to a contradiction. (Q.E.D.)

Take the same example as before but now let $(g_{11}, g_{21}, g_{31}, g_{41}, g_{51}) = (1, 2, 2, 2, 2)$. Assume we classify the pattern correctly. This implies that $\alpha_1 > \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5$. Any solution for which this is true has a design dataset error rate of 0.35 which cannot be optimal. Hence, in an optimal solution, it holds that $y_1^* = 1$. The constraint that corresponds to pattern 1 can be dropped from the formulation as it will be satisfied for all possible values of $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$.

After applying reduction rules 1, 2a and 2b, some design dataset patterns will be left in MILP-ϵ with an unknown value for y_n^*. Let $\mathcal{A}_n$ be the set of restrictions for any such pattern n. For some design dataset patterns n and m, it may hold that $\mathcal{A}_n = \mathcal{A}_m$ and the next reduction rule is then obvious.

Reduction Rule 3 (RR3) : When $\mathcal{A}_n = \mathcal{A}_m$ for two different design dataset patterns n and m in MILP-ϵ, one set of restrictions, say set $\mathcal{A}_m$, can be dropped from the formulation provided coefficient of y_n in the objective function is increased by 1.

Assume pattern 1 and 2 belong to class 1 and pattern 3 belongs to class 2. Assume five component classifiers classify these three patterns as $(g_{11}, g_{21}, g_{31}, g_{41}, g_{51}) = (1, 1, 3, 4, 4)$, $(g_{12}, g_{22}, g_{32}, g_{42}, g_{52}) = (1, 1, 2, 3, 3)$ and $(g_{13}, g_{23}, g_{33}, g_{43}, g_{53}) = (2, 2, 1, 3, 3)$. It is easy to check that $\mathcal{A}_1 = \mathcal{A}_2 = \mathcal{A}_3 = \{\alpha_1 + \alpha_2 - \alpha_3 + (1+\epsilon) y_1 \geq \epsilon, \alpha_1 + \alpha_2 - \alpha_4 - \alpha_5 + (1+\epsilon) y_1 \geq$

$\epsilon, \alpha_1 + \alpha_2 - 0 + (1+\epsilon)y_1 \geq \epsilon\}$. Hence, the restrictions of pattern 2 and 3 can be left out of MILP-ϵ by applying RR3 twice provided the objective function coefficient of y_1 is increased by 2.

Combined with pre-processing, standard branch and bound methods as implemented in commercial software are capable of solving fairly large problem instances within reasonable time limits.

4 Real World Examples

In datarich conditions, a methodological sound way of working with the MILP-ϵ formulation would be to divide the available data at random in three independent datasets: one to design the component classifiers, one to design the aggregated classifier and one dataset to evaluate both the component and aggregated classifiers. For MV aggregating and boosting, a separate dataset to design the aggregated classifier is not needed [3, 8].

If data are scarce, a different approach is necessary to use the available data more efficiently. The real world examples illustrate such schemes.

All four datasets are publicly available and described at the UCI repository at http:/kdd.ics.uci.edu [2].

Our aim on the tic-tac-toe dataset is to illustrate in detail how pre-processing can successfully reduce the computation time to solve MILP-ϵ. The component classifiers will be obtained by applying different SC methods to the same design dataset. We will compare the performance of the MILP-ϵ classifier to that of the MV classifier.

4.1 Tic-tac-toe

The tic-tac-toe dataset consists of 958 patterns each having 9 measurements. The patterns are legal tic-tac-toe endgame boards and the two classes are: $\times$ wins or $\times$ does not win, respectively referred to as class 1 and class 2. The data are randomly divided into a design dataset $\mathcal{D}$ of 638 patterns and a testing dataset $\mathcal{T}$ of 320. Component classifiers are obtained by using different SC methods as given in Table 1. The entry in the column "linear?" is yes if the classifier is linear in the measurements, no if not. The LDA, QDA, NN, KER and MLR classifiers are calculated with SAS [9]. The MSID-4A [5], GOCH [6] and BENN [1] classifiers are found with LINDO [10].

Table 1: SC Methods

SC method	linear?	implementation details
majority rule (MR)	no	-
linear discriminant analysis (LDA)	yes	proportional priors
quadratic discriminant analysis (QDA)	no	proportional priors
nearest neighbour (NN)	no	$k = 64$
kernel (KER)	no	$r = 3$, pool = yes, kernel = normal
multinomial logistic regression (MLR)	no	-
linear programming (MSID-4A)	yes	$H = 1$, $K = 0$, $s = 1$
linear programming (GOCH)	yes	-
linear programming (BENN)	yes	-

Denote by $h_{\mathcal{D}}$ the error rate on the design dataset and by $h_{\mathcal{T}}$ the error rate on the testing dataset. $h_{\mathcal{D}_1}$, $h_{\mathcal{D}_2}$ and $h_{\mathcal{T}_1}$, $h_{\mathcal{T}_2}$ are the error rates per class, respectively on the design and the testing dataset. The error rates are expressed as percentages. Table 2 shows the aggregation results when aggregating only the four classifiers that are linear in the measurements are aggregated. ϵ was set to 0.0001.

Table 2: Aggregating Linear Classifiers

SC method	$h_{\mathcal{D}}$	$h_{\mathcal{D}_1}$	$h_{\mathcal{D}_2}$	$h_{\mathcal{T}}$	$h_{\mathcal{T}_1}$	$h_{\mathcal{T}_2}$
LDA	35.5	34.9	31.0	34.4	33.3	36.1
MSID-4A	25.7	17.3	42.9	28.3	16.2	47.5
GOCH	25.9	13.8	50.5	29.9	13.1	56.6
BENN	32.0	32.6	31.0	33.1	31.3	36.1
MV aggregating	32.0	32.5	31.0	33.1	31.3	36.1
MILP-ϵ aggregating	23.7	17.1	37.1	26.9	16.2	44.3

Given the component classifiers, the MILP-ϵ aggregated classifier outperforms all other classifiers in terms of accuracy both on the design and the testing dataset. The MILP-ϵ aggregated classifier performs (much) better than the MV classifier and also strictly better than any of the individual classifiers. The MV classifier does not perform well. Due to pre-processing only 5 binary variables are left in the final MILP-ϵ formulation. As many as 485 patterns could be left out of the MILP-ϵ model due to RR1. 38 binary variables could be left out due to RR2a and 16 due to RR2b. Another 94 binary variables can be left out due to RR3. The resulting MILP-ϵ is as simple as

$$\min 16y_1 + 66y_2 + 3y_3 + y_4 + 13y_5 + 135$$

subject to

$$\begin{aligned}
\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 + (1+\epsilon)y_1 &\geq \epsilon \\
\alpha_3 + \alpha_4 - \alpha_1 - \alpha_2 + (1+\epsilon)y_2 &\geq \epsilon \\
\alpha_4 - \alpha_1 - \alpha_2 - \alpha_3 + (1+\epsilon)y_3 &\geq \epsilon \\
\alpha_2 + \alpha_4 - \alpha_1 - \alpha_3 + (1+\epsilon)y_4 &\geq \epsilon \\
\alpha_1 + \alpha_2 - \alpha_3 - \alpha_4 + (1+\epsilon)y_5 &\geq \epsilon \\
\alpha_1 &\leq \alpha_2 + \alpha_3 + \alpha_4 \\
\alpha_3 &\leq \alpha_1 + \alpha_2 + \alpha_4 \\
\alpha_4 &\leq \alpha_1 + \alpha_2 + \alpha_3 \\
\sum_{l=1}^{4} \alpha_l &= 1 \\
0 \leq \alpha_l &\leq 1 \qquad l \in \{1, ..., 4\} \\
\alpha_l &\in \mathrm{IR} \qquad l \in \{1, ..., 4\} \\
y_n &\in \{0, 1\} \qquad n \in \{1, ..., 3\}
\end{aligned}$$

The optimal weights $(\alpha_1^*, \alpha_2^*, \alpha_3^*, \alpha_4^*)$ = (0,0.49995,0.49995, 0.0001) are obtained in 0.1 CPU seconds on a pentium 530 MHz computer. Without preprocessing, LINDO needs 3.6 CPU seconds to find the same optimal solution.

Table 3: Aggregating All Classifiers

SC method	$h_{\mathcal{D}}$	$h_{\mathcal{D}_1}$	$h_{\mathcal{D}_2}$	$h_{\mathcal{T}}$	$h_{\mathcal{T}_1}$	$h_{\mathcal{T}_2}$
MR	32.9	0	100	38.1	0	100
LDA	35.5	34.9	31.0	34.4	33.3	36.1
QDA	20.1	24.4	11.4	23.1	22.7	23.8
NN	30.1	37.7	14.8	29.7	37.4	17.2
KER	34.2	39.6	23.3	35.6	36.9	33.6
MLR	29.8	15.9	58.1	28.1	8.6	59.8
MSID-4	25.7	17.3	42.9	28.3	16.2	47.5
GOCH	25.9	13.8	50.5	29.9	13.1	56.6
BENN	32.0	32.6	31.0	33.1	31.3	36.1
MV aggregating	25.5	22.9	31.0	28.4	21.2	40.2
MILP-ϵ aggregating	20.1	24.4	11.4	23.1	22.7	23.8

Table 3 gives the aggregation results when all classifiers are aggregated. The MILP-ϵ aggregated classifier again outperforms the MV classifier, both on the design and the testing dataset. The optimal solution to MILP-ϵ is obtained as $(\alpha_1^*, \alpha_2^*, \alpha_3^*, \alpha_4^*, \alpha_5^*, \alpha_6^*, \alpha_7^*, \alpha_8^*, \alpha_9^*)$ = (0,0,0.50005,0,0,0,0,0,0.49995). Its performance is identical to the performance of QDA which also follows from the observation that $\alpha_3^* > 0.5$ (classifier 3 corresponds to QDA). After pre-processing, 47 binary variables remained in the final MILP-ϵ model: 228 patterns are left out due RR1. RR2a removed 103 binary variables and RR2b 43. Another 228 binary variables were removed due to RR3. It took LINDO 0.1 second to find the optimal weights. Without pre-processing, after 24 hours CPU time, LINDO still had not terminated its search. This illustrates the importance of pre-processing. □

In boosting, say Adaboost.M1, the component classifiers are obtained in a specific way [8]. Therefore, it was not possible to compare the performance of the MILP-ϵ classifier to that of a boosted classifier on the tic-tac-toe dataset. On the handwritten digits dataset, the component classifiers will be obtained by using the Adaboost.M1 procedure. The component classifiers will be aggregated with all three procedures: boosting, MV and MILP-ϵ. The handwritten digits dataset is also substantially larger than the other real world datasets. Our aim is to show that our approach, thanks to pre-processing, can also be successful on large real life datasets.

4.2 Optical Recognition of Handwritten Digits

The handwritten digits data gives 64 discrete valued measurements of normalized bitmaps of 5620 handwritten digits. The ten possible classes are 0,1,...,9. Separate design and testing datasets are given, consisting respectively of 3823 and 1797 patterns.

We have applied AdaBoost.M1 [8] using a prototype classifier. The prototype classifier puts a pattern in the class that has the closest design dataset average to it, in terms of Euclidian distance. Ties are solved randomly. The boosting procedure stabilized already after eight iterations. Hence, we have eight component classifiers. In the AdaBoost.M1, component classifiers with a low design dataset error rate are given a relatively large weight α_l. However, the eight component classifiers could also be aggregated by means of a MV or the MILP-ϵ formulation. The results are shown in Table 4. The classifier aggregated with MILP-ϵ performs best on the design dataset as well as

the testing dataset. Pre-processing was able to reduce the number of binary variables from 3823 to 103. The final MILP-ϵ formulation was solved in 14 seconds. These results shows that our approach can also be successful on fairly large real life datasets. □

Table 4: Results of Aggregated Classifiers

	$h_{\mathcal{D}}$	$h_{\mathcal{T}}$
AdaBoost.M1	8.1	10.6
MV	7.8	10.6
MILP-ϵ	7.1	9.4

These two examples show that, although methodologically questionable, using the same dataset to get the component classifiers and aggregate them might still give good results. In general though, such an approach is expected to easily overfit the design dataset. On the tic-tac-toe dataset, the MILP-ϵ aggregated classifier overfits the design dataset though not dramatically. Previous studies [12] suggest that the tic-tac-toe dataset is highly irregular. Also on the handwritten digits dataset, there is some overfitting but it is not problematic.

On the image segmentation dataset below, we propose another way to get the component classifiers and subsequently aggregate them. In contrast to the tic-tac-toe example, the component classifiers will be obtained by applying the same SC method. Variety in the component classifiers will be obtained by changing the data onto which the SC method is run. The SC method we opt for is k-nearest neighbor as this method is known to work well on image segmentation data. The aim is to see if aggregation works when the component classifiers are already good. We will compare MV and MILP-ϵ aggregation.

4.3 Image Segmentation Dataset

In the image segmentation dataset, one is asked to classify regions of nine pixels based on 19 continuous measurements. The pixel regions are taken from outdoor images. The seven possible classes for each pixel region are brickface, sky, foliage, cement, window, path and grass. There are 30 patterns per class in the design dataset and 300 instances per class in the

testing dataset. The large testing dataset will *only* be used for evaluation. Hence, despite the large testing dataset, we are in datascarce conditions: as few as 210 patterns are available to design the component classifiers and the aggregated classifier in a 19-dimensional measurement space.

First, from the design dataset, 10 sub-samples of 210 patterns are randomly drawn *with* replacement. It can be verified that the expected number of different patterns in a sub-sample equals 132.93. A component classifier is calculated on each of the 10 sub-samples using the same SC method (k-nearest neighbor). Next, the component classifiers are aggregated with MILP-ϵ using all 210 patterns of the design dataset. Hence, in the aggregation, each component classifier is up against an expected 77.07 new patterns. To end, the error rate of the classifiers is evaluated. Each of the 10 component classifiers is evaluated on the sub-sample used for its design ($\circ$), on the design dataset ($\diamond$) and the testing dataset ($\triangle$). The component classifier error rates are averaged to indicate the performance of the average component classifier (indicated by filled symbols). The MV classifier (full line) and the MILP-ϵ aggregated classifier (dashed line) are evaluated on the design dataset and the testing dataset.

The component classifiers are calculated in SAS using the k- nearest neighbor method. In order to see if the aggregation results change if one varies the SC method to design the component classifier, the procedure is repeated for $k \in \{1, 2, 3, 5, 6, 7, 10, 15, 30\}$. ϵ was set to 0.0001. Figure 1 shows the results.

Both the component and the aggregated classifiers overfit the dataset used in their design. Surprisingly, on this dataset, the best testing dataset error rates are obtained by using the 1-nearest neighbor method. Although the performance on the design dataset differs, the testing dataset performance of the MV and MILP-ϵ aggregated classifier is very similar. The accuracy gain of aggregating (albeit with MV or MILP-ϵ) compared to the average component classifier is one or two percentages. Of course, on the design dataset, the MILP-ϵ aggregated classifier is always at least as good as any component classifier. Pre-processing was able to keep on average 85.4% of the design dataset patterns out of the MILP-ϵ formulation and strongly reduced the computational effort needed. The results on this dataset illustrate that the MILP-ϵ does not always outperform the MV aggregated classifier on unseen data despite superior performance on the design dataset. □

In all three examples, a (large) testing dataset was available. Often, one

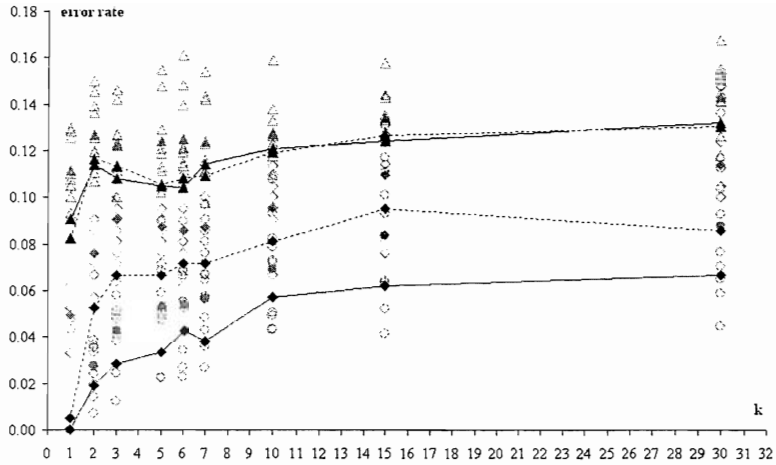


Figure 1: Results for the Image Segmentation Dataset

does not have a testing dataset available or is unwilling to use it for evaluation purposes only. An alternative that uses the available data more efficiently would be to estimate the accuracy of the aggregated classifier with K-fold cross-validation. The last example shows how to get a 10-fold cross-validation estimate of the accuracy of the MV aggregated as well as the MILP-ϵ aggregated classifier. Also, we will illustrate how to deal with extra, problem specific constraints in MILP-ϵ. To end, we will also have a look at the reliability estimates obtained by both aggregation methods.

4.4 Dermatology Dataset

The aim of the dermatology dataset is to diagnose one of six possible types of eryhemato-squamous diseases. 12 clinical and 24 histopathological measurements of the patient are given. Only the 12 clinical measurements are used. The dataset contains 366 patterns, which are to be used for both classifier design and testing. As in the image segmentation example, datascarce conditions apply. Eight patterns suffered from missing values and were

left out. The remaining 358 form the design dataset.

From the design dataset, 30 sub-samples of 358 patterns are randomly drawn *with* replacement. The expected number of different patterns in such a sub-sample is 226.48. For each of these sub-samples, a component classifier was calculated in SAS using the LDA method with proportional priors and tested on the design dataset patterns that were not in the sub-sample. Figure 2 shows the error rate per class of the 30 component classifiers on the design dataset ($\diamond$) and the testing dataset ($\triangle$). Also the error rate over all classes is shown.

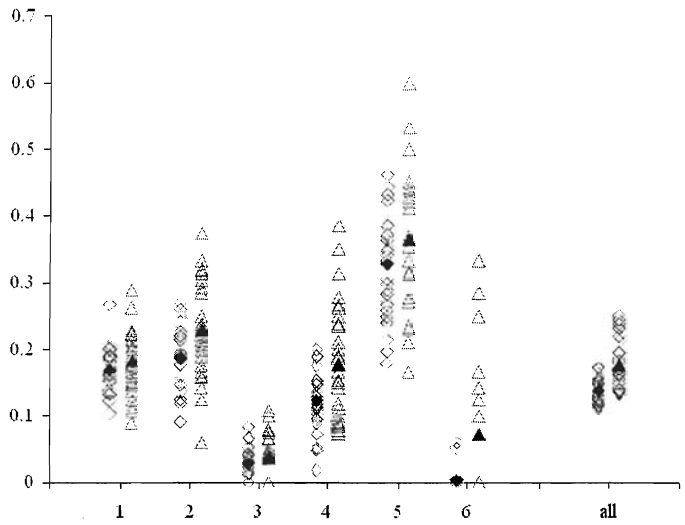


Figure 2: Component Classifier Results for Dermatology

Assume class 5 patterns are of special interest. The component classifiers have difficulties to classify class 5 patterns correctly. Can we find an aggregated classifier that has a better accuracy than the component classifiers, especially for patterns from class 5?

There are several ways to adjust the standard MILP-ϵ model to take this problem specific demand into account. Notice that this is not possible in MV

aggregation or boosting.

An obvious way of working would be to include a constraint of the type $\sum_{n \in \mathcal{D}_5} y_n \leq \delta |\mathcal{D}_5|$ where $\mathcal{D}_5$ is the set of design dataset patterns that belong to class 5 and δ is the maximum allowable error rate for class 5 on the design dataset. However, such constraint may render the model infeasible and one has to be careful in setting δ if one pre-processes.

We have implemented another idea. By changing the objective function coefficients e_n, it is possible to pre-emptively model the objective to minimize the number of class 5 misclassifications. E.g., set the objective function coefficient $e_n = N$ if pattern n belongs to class 5 and $e_n = 1$ otherwise. The advantage of this way of working is that the model cannot run into an infeasibility. The reduction rules remain valid. The only difference is that the objective function value of MILP-ϵ has to be increased with N for every misclassified pattern of class 5 that is left out and by 1 for every other misclassified pattern that is removed.

As no testing dataset is available, we will use 10-fold cross-validation to estimate the accuracy of the aggregated estimators.

The design dataset was randomly split into 10 subsets $\mathcal{D}_k$ of almost equal size with $k \in \{1, 2, ..., 10\}$. The component classifiers are aggregated on the 10 sets $\mathcal{D} \setminus \mathcal{D}_k$ and tested on the corresponding set $\mathcal{D}_k$. Averaging of the testing results gives the 10-fold cross-validation estimate of the aggregated classifier on $\mathcal{D}$. Table 5 shows the results.

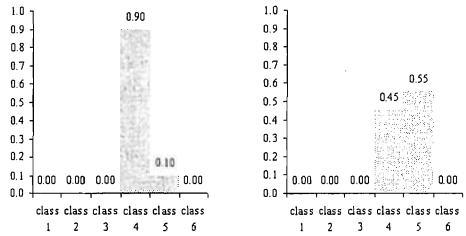
Table 5: Aggregating LDA Classifiers

	h_1	h_2	h_3	h_4	h_5	h_6	h_{all}
LDA component (averaged)	0.18	0.23	0.04	0.18	0.37	0.07	0.18
MV aggregated	0.17	0.23	0.02	0.14	0.38	0.00	0.16
MILP-ϵ aggregated	0.15	0.22	0.02	0.19	0.30	0.06	0.15

The error rates of the MILP-ϵ aggregated classifier have improved, especially for class 5, but not drastically. Even on the sets $\mathcal{D} \setminus \mathcal{D}_k$, the average class 5 error rate was still 0.19. The reason for this is that 9 out the 48 design dataset patterns from class 5 are misclassified by *all* component classifiers. This illustrates the more general statement of Breiman [3] in the context of MV aggregation: "Bagging unstable classifies usually improves them. Bagging stable classifiers is not a good idea." Indeed, if the component classifiers

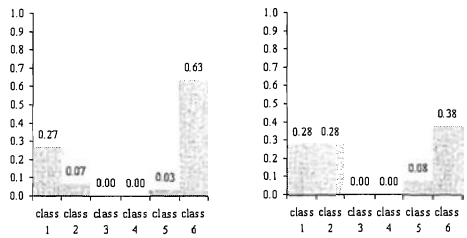
tend to yield the same classification, there is only little improvement to be made by aggregating.

In the context of medical diagnosis, it might be interesting to have an idea about the reliability of the predicted class, i.e. to have an estimate of the conditional probabilities $pr_C(\mathrm{C} = c|\mathrm{X}_1 = x_1, ..., \mathrm{X}_P = x_P)$. Both MV and MILP-$\epsilon$ aggregation directly provide such estimates.



(a) MV aggregated (b) MILP-ϵ aggregated

Figure 3: Conditional Probability Estimates for a Pattern from Class 5



(a) MV aggregated (b) MILP-ϵ aggregated

Figure 4: Conditional Probability Estimates for a Pattern from Class 6

Figures 3 and 4 give the class conditional probability estimates for two patterns. Panel (a) of Figure 3 shows that the pattern is incorrectly classified with a high reliability by the MV aggregated classifier. The MILP-ϵ aggregated classifier on the other hand classifies the pattern correctly as shown in panel (b), though the reliability of the classification is not so high. In Figure

4, both aggregated classifiers make a correct classification but the reliability differs. □

5 Conclusions

With regard to the use of MV aggregating, Breiman (1994) argues that "Aggregating can (...) transform good predictors into nearly optimal ones. On the other hand, (...), poor predictors can be transformed into worse ones." Aggregating with mathematical programming guarantees that poor predictors cannot be transformed into worse ones as the aggregated classifier will be at least as good as the best predictor on the design dataset for a given criterion function. The real world examples suggest that this is often also true on unseen data and that our approach can outperform both bagging and boosting. For practicioners, the approach is particularly interesting because it offers the user the flexibility to include extra problem specific constraints into the design of the aggregated classifier and gives an idea of the reliability of the final classification.

References

[1] Bennett K.P. and Mangasarian O.L., 1993. Multicategory Discrimination via Linear Programming, Optimization Methods and Software 3, pp. 27-39.

[2] Blake C.L. and Merz C.J., 1998. UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[3] Breiman L., 1994. Bagging Predictors. University of California, Department of Statistics, Technical Report 421.

[4] Duda R.O., Hart P.E. and Strok D.H., 2001. Pattern Recognition, 2nd Edition. Wiley-Interscience New York.

[5] Erenguc S.S. and Koehler G.J., 1990. Survey of Mathematical Programming Models and Experimental Results for Linear Discriminant Analysis, Managerial and Decision Economics, Volume 11, pp. 215-225.

[6] Gochet W., Stam A., Srinivasan V. and Chen S., 1997. Multigroup Discriminant Analysis Using Linear Programming, Operations Research, Volume 45, Number 2, pp. 213-225.

[7] Hand D.J., 1997. Construction and Assessment of Classification Rules. Wiley Chichester.

[8] Hastie T., Tibshirani R. and Friedman J., 2002. The Elements of Statistical Learning. Springer Series in Statistics, Springer.

[9] SAS Institute Inc. SAS/STATTM Users's Guide, Release 6.03 Edition. Cary, NC: SAS Institute Inc., 1998

[10] Schrage, L. 1995. LINDO: Optimization Software for Linear Programming. Lindo Systems Inc., Chicago, IL.

[11] Thomas L.C., Edelman D.B. and Crook J.N., 2002. Credit Scoring and Its Applications. Society for Industrial and Applied Mathematics.

[12] Van Gestel T., Suykens J., Baesens B., Viaene S., Vanthienen J., Dedene G., De Moor B. and Vandewalle J., 2002. Benchmarking Least Squares Support Vector Machine Classifiers, Machine Learning, forthcoming.

[13] Webb A., 1999. Statistical Pattern Recognition. Arnold London.