Machine Learning, 41, 315–343, 2000

Cascade Generalization

JOÃO GAMA jgama@ncc.up.pt
PAVEL BRAZDIL pbrazdil@ncc.up.pt
LIACC, FEP, University of Porto, Rua Campo Alegre, 823 4150 Porto, Portugal*

Editor: Raul Valdes-Perez

Abstract. Using multiple classifiers for increasing learning accuracy is an active research area. In this paper we present two related methods for merging classifiers. The first method, Cascade Generalization, couples classifiers loosely. It belongs to the family of stacking algorithms. The basic idea of Cascade Generalization is to use sequentially the set of classifiers, at each step performing an extension of the original data by the insertion of new attributes. The new attributes are derived from the probability class distribution given by a base classifier. This constructive step extends the representational language for the high level classifiers, relaxing their bias. The second method exploits tight coupling of classifiers, by applying Cascade Generalization locally. At each iteration of a *divide and conquer* algorithm, a reconstruction of the instance space occurs by the addition of new attributes. Each new attribute represents the probability that an example belongs to a class given by a base classifier. We have implemented three *Local Generalization Algorithms*. The first merges a linear discriminant with a decision tree, the second merges a naive Bayes with a decision tree, and the third merges a linear discriminant and a naive Bayes with a decision tree. All the algorithms show an increase of performance, when compared with the corresponding single models. *Cascade* also outperforms other methods for combining classifiers, like *Stacked Generalization*, and competes well against *Boosting* at statistically significant confidence levels.

Keywords: multiple models, constructive induction, combining classifiers, merging classifiers

1. Introduction

The ability of a chosen classification algorithm to induce a good generalization depends on the appropriateness of its representation language to express generalizations of the examples for the given task. The representation language for a standard decision tree is the DNF formalism that splits the instance space by axis-parallel hyper-planes, while the representation language for a linear discriminant function is a set of linear functions that split the instance space by oblique hyper planes. Since different learning algorithms employ different knowledge representations and search heuristics, different search spaces are explored and diverse results are obtained. In statistics, Henery (1997) refers to *rescaling* as a method used when some classes are over-predicted leading to a bias. Rescaling consists of applying the algorithms in sequence, the output of an algorithm being used as input to another algorithm. The aim would be to use the estimated probabilities $W_i = P(C_i \mid X)$ derived from a learning algorithm, as input to a second learning algorithm the purpose of which is to produce an unbiased estimate $Q(C_i \mid W)$ of the conditional probability for class C_i.

*http://www.ncc.up.pt/liacc/ML.

The problem of finding the appropriate bias for a given task is an active research area. We can consider two main lines of research: on the one hand, methods that try to select the most appropriate algorithm for the given task, for instance Schaffer's selection by cross validation (Schaffer, 1993), and on the other hand, methods that combine predictions of different algorithms, for instance Stacked Generalization (Wolpert, 1992). The work presented here near follows the second line of research. Instead of looking for methods that fit the data using a single representation language, we present a family of algorithms, under the generic name of *Cascade Generalization*, whose search space contains models that use different representation languages. Cascade generalization performs an iterative composition of classifiers. At each iteration a classifier is generated. The input space is extended by the addition of new attributes. These are in the form of probability class distributions which are obtained, for each example, by the generated classifier. The language of the final classifier is the language used by the high level generalizer. This language uses terms that are expressions from the language of low level classifiers. In this sense, Cascade Generalization generates a unified theory from the base theories generated earlier.

Used in this form, Cascade Generalization performs a loose coupling of classifiers. The method can be applied *locally* at each iteration of a divide-and- conquer algorithm generating a tight coupling of classifiers. This method is referred to as *Local Cascade Generalization*. In our implementation, it generates a decision tree, which has interesting relations with multivariate trees (Brodley & Utgoff, 1995) and neural networks, namely with the Cascade correlation architecture (Fahlman, 1991). Both Cascade Generalization and Local Cascade Generalization are described and analyzed in this paper. The experimental study shows that this methodology usually improves accuracy and decreases theory size at statistically significant levels.

In the next Section we review previous work in the area of multiple models. In Section 3 we present the framework of *Cascade Generalization*. In Section 4 we discuss the strengths and weaknesses of the proposed method in comparison to other approaches to multiple models. In Section 5 we perform an empirical evaluation of Cascade Generalization using UCI data sets. In Section 6 we define a new family of multi-strategy algorithms that apply Cascade Generalization *locally*. In Section 7, we empirically evaluate *Local Cascade Generalization* using UCI data sets. In Section 8, we examine the behavior of Cascade Generalization providing insights about why it works. The last Section summarizes the main points of the work and discusses future research directions.

2. Related work on combining classifiers

Voting is the most common method used to combine classifiers. As pointed out by Ali and Pazzani (1996), this strategy is motivated by the Bayesian learning theory which stipulates that in order to maximize the predictive accuracy, instead of using just a single learning model, one should ideally use all of the models in the hypothesis space. The vote of each hypothesis should be weighted by the posterior probability of that hypothesis given the training data. Several variants of the voting method can be found in the machine learning literature, from uniform voting where the opinion of all base classifiers contributes to the

final classification with the same strength, to weighted voting, where each base classifier has a weight associated, that could change over the time, and strengthens the classification given by the classifier.

Another approach to combine classifiers consists of generating multiple models. Several methods appear in the literature. In this paper we analyze them through *Bias-Variance* analysis (Kohavi & Wolpert, 1996): methods that mainly reduce variance, such as *Bagging* and *Boosting*[1], and methods that mainly reduce *bias*, such as *Stacked Generalization* and *Meta-Learning*.

2.1. Variance reduction methods

Breiman (1998) proposes *Bagging*, that produces replications of the training set by sampling with replacement. Each replication of the training set has the same size as the original data but some examples do not appear in it while others may appear more than once. From each replication of the training set a classifier is generated. All classifiers are used to classify each example in the test set, usually using a uniform vote scheme.

The *Boosting* algorithm of Freund and Schapire (1996) maintains a weight for each example in the training set that reflects its importance. Adjusting the weights causes the learner to focus on different examples leading to different classifiers. Boosting is an iterative algorithm. At each iteration the weights are adjusted in order to reflect the performance of the corresponding classifier. The weight of the misclassified examples is increased. The final classifier aggregates the learned classifiers at each iteration by weighted voting. The weight of each classifier is a function of its accuracy.

2.2. Bias reduction methods

Wolpert (1996) proposed *Stacked Generalization*, a technique that uses learning at two or more levels. A learning algorithm is used to determine how the outputs of the base classifiers should be combined. The original data set constitutes the level zero data. All the base classifiers run at this level. The level one data are the outputs of the base classifiers. Another learning process occurs using as input the level one data and as output the final classification. This is a more sophisticated technique of cross validation that could reduce the error due to the bias.

Chan and Stolfo (1995b) present two schemes for classifier combination: *arbiter* and *combiner*. Both schemes are based on meta learning, where a meta-classifier is generated from meta data, built based on the predictions of the base classifiers. An arbiter is also a classifier and is used to arbitrate among predictions generated by different base classifiers. The training set for the arbiter is selected from all the available data, using a selection rule. An example of a selection rule is "*Select the examples whose classification the base classifiers cannot predict consistently*". This arbiter, together with an arbitration rule, decides a final classification based on the base predictions. An example of an arbitration rule is "*Use the prediction of the arbiter when the base classifiers cannot obtain a majority*". Later (Chan & Stolfo, 1995a), this framework was extended using *arbiters/combiners* in an hierarchical fashion, generating *arbiter/combiner* binary trees.

Skalak (1997) presents a dissertation discussing methods for combining classifiers. He presents several algorithms most of which are based on *Stacked Generalization* which are able to improve the performance of *Nearest Neighbor* classifiers.

Brodley (1995) presents *MCS*, a hybrid algorithm that combines, in a single tree, nodes that are *univariate tests*, *multivariate tests* generated by *linear machines* and *instance based learners*. At each node MCS uses a set of *If-Then* rules to perform a hill-climbing search for the best hypothesis space and search bias for the given partition of the dataset. The set of rules incorporates knowledge of experts. *MCS* uses a dynamic search control strategy to perform an automatic model selection. *MCS* builds trees which can apply a different model in different regions of the instance space.

2.3. Discussion

Results of *Boosting* or *Bagging* are quite impressive. Using 10 iterations (i.e. generating 10 classifiers) Quinlan (1996) reports reductions of the error rate between 10% and 19%. Quinlan argues that these techniques are mainly applicable for unstable classifiers. Both techniques require that the learning system not be stable, to obtain different classifiers when there are small changes in the training set. Under an analysis of bias-variance decomposition of the error (Kohavi & Wolpert, 1996) the reduction of the error observed with Boosting or Bagging is mainly due to the reduction in the variance. Breiman (1998) reveals that Boosting and Bagging can only improve the predictive accuracy of learning algorithms that are "unstable".

As mentioned in Bauer and Kohavi (1998) the main problem with Boosting seems to be robustness to noise. This is expected because noisy examples tend to be misclassified, and the weight will increase for these examples. They present several cases were the performance of Boosting algorithms degraded compared to the original algorithms. They also point out that Bagging improves in *all* datasets used in the experimental evaluation. They conclude that although Boosting is on average better than Bagging, it is *not* uniformly better than Bagging. The higher accuracy of Boosting over Bagging in many domains was due to a reduction of bias. Boosting was also found to frequently have higher variance than Bagging. *Boosting* and *Bagging* require a considerable number of member models because they rely on varying the data distribution to get a diverse set of models from a single learning algorithm.

Wolpert (1992) says that successful implementation of *Stacked Generalization* for classification tasks is a "*black art*", and the conditions under which stacking works are still unknown:

> *For example, there are currently no hard and fast rules saying what* $level_0$ *generalizers should we use, what* $level_1$ *generalizer one should use, what k numbers to use to form the* $level_1$ *input space, etc*.

Recently, Ting and Witten (1997) have shown that successful stacked generalization requires the use of output class distributions rather than class predictions. In their experiments only the MLR algorithm (a linear discriminant) was suitable for level-1 generalizer.

3. Cascade generalization

Consider a learning set $D = (\vec{x_n}, y_n)$ with $n = 1, \ldots, N$, where $\vec{x_i} = [x_1, \ldots, x_m]$ is a multidimensional input vector, and y_n is the output variable. Since the focus of this paper is on classification problems, y_n takes values from a set of predefined values, that is $y_n \in \{Cl_1, \ldots, Cl_c\}$, where c is the number of classes. A classifier $\Im$ is a function that is applied to the training set D to construct a model $\Im(D)$. The generated model is a mapping from the input space X to the discrete output variable Y. When used as a predictor, represented by $\Im(\vec{x}, D)$, it assigns a y value to the example $\vec{x}$. This is the traditional framework for classification tasks. Our framework requires that the predictor $\Im(\vec{x}, D)$ outputs a vector representing conditional probability distribution $[p1, \ldots, pc]$, where p_i represents the probability that the example $\vec{x}$ belongs to class i, i.e. $P(y = Cl_i \mid \vec{x})$. The class that is assigned to the example $\vec{x}$ is the one that maximizes this last expression. Most of the commonly used classifiers, such as *naive Bayes* and *Discriminant*, classify examples in this way. Other classifiers (e.g., *C4.5* (Quinlan, 1993)), have a different strategy for classifying an example, but it requires few changes to obtain a probability class distribution.

We define a constructive operator $\varphi(\vec{x}, \mathcal{M})$ where $\mathcal{M}$ represents the model $\Im(D)$ for the training data D, while $\vec{x}$ represents an example. For the example $\vec{x}$ the operator φ concatenates the input vector $\vec{x}$ with the output probability class distribution. If the operator φ is applied to all examples of dataset D' we obtain a new dataset D''. The cardinality of D'' is equal to the cardinality of D' (i.e. they have the same number of examples). Each example in $\vec{x} \in D''$ has an equivalent example in D', but augmented with $\#c$ new attributes, where $\#c$ represents the number of classes. The new attributes are the elements of the vector of class probability distribution obtained when applying classifier $\Im(D)$ to the example $\vec{x}$. This can be represented formally as follows:

$$D'' = \Phi(D', \mathcal{A}(\Im(D), D')) \tag{1}$$

Here $\mathcal{A}(\Im(D), D')$ represents the application of the model $\Im(D)$ to data set D' and represents, in effect, a dataset. This dataset contains all the examples that appear in D' extended with the probability class distribution generated by the model $\Im(D)$.

Cascade generalization is a sequential composition of classifiers, that at each generalization level applies the Φ operator. Given a training set L, a test set T, and two classifiers $\Im_1$, and $\Im_2$, Cascade generalization proceeds as follows. Using classifier $\Im_1$, generates the $Level_1$ data:

$$Level_1 train = \Phi(L, \mathcal{A}(\Im(L), L)) \tag{2}$$

$$Level_1 test = \Phi(T, \mathcal{A}(\Im(L), T)) \tag{3}$$

Classifier $\Im_2$ learns on $Level_1$ training data and classifies the $Level_1$ test data:

$$\mathcal{A}(\Im_2(Level_1 train), Level_1 test)$$

These steps perform the basic sequence of a Cascade Generalization of classifier $\Im_2$ after classifier $\Im_1$. We represent the basic sequence by the symbol ∇. The previous composition

could be represented succinctly by:

$$\Im_2 \nabla \Im_1 = \mathcal{A}(\Im_2(Level_1 Train), Level_1 Test)$$

which, by applying Eqs. (2) and (3), is equivalent to:

$$\Im_2 \nabla \Im_1 = \mathcal{A}(\Im_2(\Phi(L, \mathcal{A}(\Im_1(L), L))), \Phi(T, \mathcal{A}(\Im_1(L), T)))$$

This is the simplest formulation of *Cascade Generalization*. Some possible extensions include the composition of *n* classifiers, and the parallel composition of classifiers.

A composition of *n* classifiers is represented by:

$$\Im_n \nabla \Im_{n-1} \nabla \Im_{n-2} \cdots \nabla \Im_1$$

In this case, Cascade Generalization generates $n-1$ levels of data. The final model is the one given by the $\Im_n$ classifier. This model could contain terms in the form of conditions based on attributes build by the previous built classifiers.

A variant of cascade generalization, which includes several algorithms in parallel, could be represented in this formalism by:

$$\begin{aligned}\Im_n \nabla [\Im_1, \ldots, \Im_{n-1}] = \mathcal{A}(\Im_n(\Phi_p(L, [\mathcal{A}(\Im_1(L), L), \ldots, \\ \mathcal{A}(\Im_{n-1}(L), L)])), (\Phi_p(T, [\mathcal{A}(\Im_1(L), T), \ldots, \\ \mathcal{A}(\Im_{n-1}(L), T)])))\end{aligned}$$

The algorithms $\Im_1, \ldots, \Im_{n-1}$ run in parallel. The operator

$$\Phi_p(L, [\mathcal{A}(\Im_1(L), L), \ldots, \mathcal{A}(\Im_{n-1}(L), L)])$$

returns a new data set L' which contains the same number of examples as L. Each example in L' contains $(n-1) \times \#cl$ new attributes, where *#cl* is the number of classes. Each algorithm in the set $\Im_1, \ldots, \Im_{n-1}$ contributes with *#cl* new attributes.

3.1. An illustrative example

In this example we will consider the UCI (Blake, Keogh, & Merz, 1999) data set *Monks-2*. The *Monks* data sets describe an artificial robot domain and are quite well known in the Machine Learning community. The robots are described by six different attributes and classified into one of two classes. We have chosen the *Monks-2 problem* because it is known that this is a difficult task for systems that learn decision trees in attribute-value formalism. The decision rule for the problem is: "The robot is O.K. if exactly two of the six attributes have their *first* value." This problem is similar to *parity* problems. It combines different attributes in a way that makes it complicated to describe in DNF or CNF using the given attributes only.

Some examples of the original training data are presented:

head,	**body,**	**smiling,**	**holding,**	**color,**	**tie,**	**Class**
round,	round,	yes,	sword,	red,	yes,	not_Ok
round,	round,	no,	balloon,	blue,	no,	OK

Using ten-fold cross validation, the error rate of *C4.5* is 32.9%, and of *naive Bayes* is 34.2%. The composite model C4.5 after *naive Bayes*, $C4.5\nabla$*naive Bayes*, operates as follows. The $Level_1$ data is generated, using the *naive Bayes* as the classifier. Naive Bayes builds a model from the original training set. This model is used to compute a probability class distribution for each example in the training and test set. The $Level_1$ is obtained by extending the train and test set with the probability class distribution given by the naive Bayes. The examples shown earlier take the form of:

head,	**body,**	**smiling,**	**holding,**	**color,**	**tie,**	**P(OK),**	**P(not Ok),**	**Class**
round,	round,	yes,	sword,	red,	yes,	0.135,	0.864,	not_Ok
round,	round,	no,	balloon,	blue,	no,	0.303,	0.696,	OK

where the new attribute $P(OK)$ ($P(not\ OK)$) is the probability that the example belongs to class *OK*(*not OK*).

C4.5 is trained on the *Level_1* training data, and classifies the *Level_1* test data. The composition $C4.5\nabla$*NaiveBayes*, obtains an error rate of 8.9%, which is substantially lower than the error rates of both *C4.5* and *naive Bayes*. None of the algorithms in isolation can capture the underlying structure of the data. In this case, Cascade was able to achieve a notable increase of performance. Figure 1 presents one of the trees generated by $C4.5\nabla$*naiveBayes*.

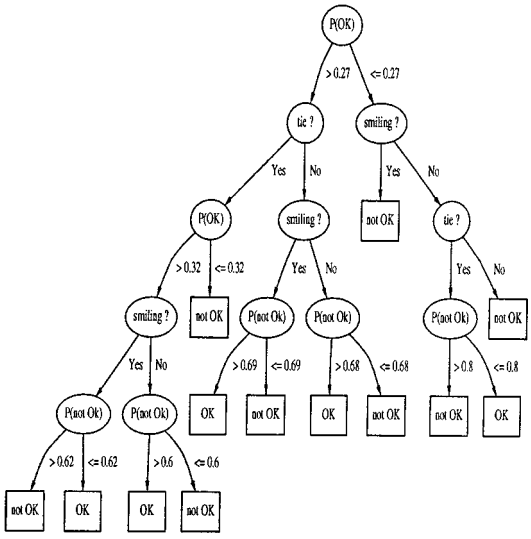

Figure 1. Tree generated by C4.5∇Bayes.

The tree contains a mixture of some of the original attributes (smiling, tie) with some of the new attributes constructed by *naive Bayes* ($P(OK)$, $P(not\ Ok)$). At the root of the tree appears the attribute $P(OK)$. This attribute represents a particular class probability (Class = OK) calculated by *naive Bayes*. The decision tree generated by C4.5 uses the constructed attributes given by Naive Bayes, but redefining different thresholds. Because this is a two class problem, the Bayes rule uses $P(OK)$ with threshold 0.5, while the decision tree sets the threshold to 0.27. Those decision nodes are a kind of function given by the Bayes strategy. For example, the attribute $P(OK)$ can be seen as a function that computes $p(Class = OK \mid \vec{x})$ using the Bayes theorem. On some branches the decision tree performs more than one test of the class probabilities. In a certain sense, this decision tree combines two representation languages: that of naive Bayes with the language of decision trees. The constructive step performed by *Cascade* inserts new attributes that incorporate new knowledge provided by naive Bayes. It is this new knowledge that allows the significant increase of performance verified with the decision tree, despite the fact that naive Bayes cannot fit well complex spaces. In the *Cascade* framework lower level learners delay the decisions to the high level learners. It is this kind of collaboration between classifiers that Cascade Generalization explores.

4. Discussion

Cascade Generalization belongs to the family of stacking algorithms. Wolpert (1992) defines Stacking Generalization as a general framework for combining classifiers. It involves taking the predictions from several classifiers and using these predictions as the basis for the next stage of classification.

Cascade Generalization may be regarded as a special case of Stacking Generalization mainly due to the layered learning structure. Some aspects that make Cascade Generalization novel, are:

- The new attributes are continuous. They take the form of a probability class distribution. Combining classifiers by means of categorical classes looses the strength of the classifier in its prediction. The use of probability class distributions allows us to explore that information.
- All classifiers have access to the original attributes. Any new attribute built at lower layers is considered exactly in the same way as any of the original attributes.
- Cascade Generalization does not use internal Cross Validation. This aspect affects the computational efficiency of Cascade.

Many of these ideas has been discussed in literature. Ting and Witten (1997) has used probability class distributions as level-1 attributes, but did not use the original attributes. The possibility of using the original attributes and class predictions as $level_1$ attributes as been pointed out by Wolpert in the original paper of Stacked Generalization. Skalak (1997) refers that Schaffer has used the original attributes and class predictions as $level_1$ attributes, but with disappointing results. In our view this could be explained by the fact that he combines three algorithms with similar behavior from a bias-variance analysis: decision

trees, rules, and neural-networks (see Section 8.2 for more details on this point). Chan and Stolfo (1995a) have used the original attributes and class predictions in a scheme denoted *class-attribute-combiner* with mixed results.

Exploiting all these aspects is what makes Cascade Generalization succeed. Moreover, this particular combination implies some *conceptual* differences.

- While Stacking is parallel in nature, Cascade is sequential. The effect is that intermediate classifiers have access to the original attributes plus the predictions of low level classifiers. An interesting possibility, that has not been explored in this paper, is to provide the classifier$_n$ with the original attributes plus the predictions provided by classifier$_{n-1}$ only.
- The ultimate goal of Stacking Generalization is combining predictions. The goal of Cascade Generalization is to obtain a model that can use terms in the representation language of lower level classifiers.
- *Cascade Generalization* provides rules to choose the low level classifiers and the high level classifiers. This aspect will be developed in the following sections.

5. Empirical evaluation

5.1. The algorithms

Ali and Pazzani (1996) and Tumer and Ghosh (1996) present empirical and analytical results that show that "*the combined error rate depends on the error rate of individual classifiers and the correlation among them*." They suggest the use of "*radically different types of classifiers*" to reduce the correlation errors. This was our criterion when selecting the algorithms for the experimental work. We use three classifiers that have different behaviors: a naive Bayes, a linear discriminant, and a decision tree.

5.1.1. Naive Bayes. Bayes theorem optimally predicts the class of an unseen example, given a training set. The chosen class is the one that maximizes: $p(C_i \mid \vec{x}) = p(C_i)\,p(\vec{x} \mid C_i)/p(\vec{x})$. If the attributes are independent, $p(\vec{x} \mid Ci)$ can be decomposed into the product $p(x_1 \mid C_i) * \cdots * p(x_k \mid C_i)$. Domingos and Pazzani (1997) show that this procedure has a surprisingly good performance in a wide variety of domains, including many where there are clear dependencies between attributes. In our implementation of this algorithm, the required probabilities are estimated from the training set. In the case of nominal attributes we use counts. Continuous attributes were discretized into equal size intervals. This has been found to produce better results than assuming a Gaussian distribution (Domingos & Pazzani, 1997; Dougherty, Kohavi, & Sahami, 1995). The number of bins used is a function of the number of different values observed on the training set: $k = \max(1; 2 * \log(nr.\,\textit{different values}))$. This heuristic was used by Dougherty, Kohavi, and Sahami (1995) with good overall results. Missing values were treated as another possible value for the attribute. In order to classify a query point, a *naive Bayes* classifier uses all of the available attributes. Langley (1996) states that *naive Bayes* relies on an important assumption that the variability of the dataset can be summarized by a single probabilistic description, and that these are sufficient to distinguish between classes. From an analysis of *Bias-Variance*, this implies that *naive Bayes* uses a

reduced set of models to fit the data. The result is low variance but if the data cannot be adequately represented by the set of models, we obtain large bias.

5.1.2. Linear discriminant. A linear discriminant function is a linear composition of the attributes that maximizes the ratio of its between-group variance to its within-group variance. It is assumed that the attribute vectors for the examples of class C_i are independent and follow a certain probability distribution with a probability density function f_i. A new point with attribute vector $\vec{x}$ is then assigned to that class for which the probability density function $f_i(\vec{x})$ is maximal. This means that the points for each class are distributed in a cluster centered at μ_i. The boundary separating two classes is a hyper-plane (Michie, Spiegelhalter, & Taylor, 1994). If there are only two classes, a unique hyper-plane is needed to separate the classes. In the general case of q classes, $q-1$ hyper-planes are needed to separate them. By applying the linear discriminant procedure described below, we get $q-1$ hyper-planes. The equation of each hyper-plane is given by:

$$H_i = \alpha_i + \sum_j \beta_{ij} * x_j \quad \text{where } \alpha_i = -\frac{1}{2}\mu_i^T S^{-1}\mu_i \text{ and } \beta_i = S^{-1}\mu_i$$

We use a Singular Value Decomposition (SVD) to compute S^{-1}. SVD is numerically stable and is a tool for detecting sources of collinearity. This last aspect is used as a method for reducing the features of each linear combination. A linear discriminant uses all, or almost all, of the available attributes when classifying a query point. Breiman (1998) states that from an analysis of Bias-Variance, Linear Discriminant is a stable classifier. It achieves stability by having a limited set of models to fit the data. The result is low variance, but if the data cannot be adequately represented by the set of models, then we obtain large bias.

5.1.3. Decision tree. *Dtree* is our version of a univariate decision tree. It uses the standard algorithm to build a decision tree. The splitting criterion is the gain ratio. The stopping criterion is similar to C4.5. The pruning mechanism is similar to the *pessimistic error* of C4.5. *Dtree* uses a kind of smoothing process that usually improves the performance of tree based classifiers. When classifying a new example, the example traverses the tree from the root to a leaf. In *Dtree*, the example is classified taking into account not only the class distribution at the leaf, but also all class distributions of the nodes in the path. That is, all nodes in the path contribute to the final classification. Instead of computing class distribution for all paths in the tree at classification time, as it is done in Buntine (1990), *Dtree* computes a class distribution for all nodes when growing the tree. This is done recursively taking into account class distributions at the current node and at the predecessor of the current node, using the recursive Bayesian update formula (Pearl, 1988):

$$P(C_i \mid e_n, e_{n+1}) = P(C_i \mid e_n)\frac{P(e_{n+1} \mid e_n, C_i)}{P(e_{n+1} \mid e_n)}$$

where $P(e_n)$ is the probability that one example falls at node n, that can be seen as a shorthand for $P(e \in E_n)$, where e represents the given example and E_n the set of examples in node n. Similarly $P(e_{n+1} \mid e_n)$ is the probability that one example that falls at node n goes

to node *n+1*, and $P(e_{n+1} \mid e_n, C_i)$ is the probability that one example from class C_i goes from node *n* to node $n+1$. This recursive formulation, allows *Dtree* to compute efficiently the required class distributions. The smoothed class distributions influence the pruning mechanism and the treatment of missing values. It is the most relevant difference from C4.5.

A decision tree uses a subset of the available attributes to classify a query point. Kohavi and Wolpert (1996), Breiman (1998) among other researchers, note that decision trees are unstable classifiers. Small variations on the training set can cause large changes in the resulting predictors. They have high variance but they can fit any kind of data: the bias of a decision tree is low.

5.2. The experimental methodology

We have chosen 26 data sets from the UCI repository. All of them were previously used in other comparative studies. To estimate the error rate of an algorithm on a given dataset we use 10 fold stratified cross validation. To minimize the influence of the variability of the training set, we repeat this process ten times, each time using a different permutation of the dataset.[2] The final estimate is the mean of the error rates obtained in each run of the cross validation. At each iteration of CV, all algorithms were trained on the same training partition of the data. Classifiers were also evaluated on the same test partition of the data. All algorithms where used with the default settings.

Comparisons between algorithms were performed using *paired t-tests* with significance level set at 99.9% for each dataset. We use the Wilcoxon matched-pairs signed-ranks test to compare the results of the algorithms across datasets.

Our goal in this empirical evaluation is to show that *Cascade Generalization* are plausible algorithms, that compete quite well against other well established techniques. Stronger statements can only be done after a more extensive empirical evaluation.

Table 1 presents the error rate and the standard deviation of each base classifier. Relative to each algorithm a $+(-)$ sign on the first column means that the error rate of this algorithm, is significantly better (worse) than *Dtree*. The error rate of *C5.0* is presented for reference. These results provide evidence, once more, that no single algorithm is better overall.

5.3. Evaluation of Cascade Generalization

Tables 2 and 3 presents the results of all pairwise combinations of the three base classifiers and the most promising combination of the three models. Each column corresponds to a *Cascade Generalization* combination. For each combination we have conducted *paired t-tests*. All composite models are compared against its components using *paired t-tests* with significance level set to 99.9%. The $+(-)$ signs indicate that the combination (e.g. C4∇Bay) is significantly better than the component algorithms (i.e. C4.5 and Bayes).

The results are summarized in Tables 4 and 5. The first line shows the arithmetic mean across all datasets. It shows that the most promising combinations are *C4.5*∇ *Discrim*, *C4.5*∇ *naive Bayes*, *C4.5*∇ *Discrim*∇ *naive Bayes*, and *C4.5*∇ *naive Bayes*∇ *Discrim*. This is confirmed by the second line that shows the geometric mean. The third line that

Table 1. Data characteristics and results of base classifiers.

Dataset	#Classes	#Examples	Dtree	Bayes	Discrim	C4.5	C5.0
Adult	2	48842	13.93 ± 0.4	$(-)17.40 \pm 0.7$	$(-)21.93 \pm 0.4$	13.98 ± 0.6	13.86 ± 0.6
Australian	2	690	14.13 ± 0.6	14.48 ± 0.4	14.06 ± 0.1	14.71 ± 0.6	14.17 ± 0.7
Balance	3	625	22.35 ± 0.7	$(+)8.57 \pm 0.3$	$(+)13.35 \pm 0.3$	22.10 ± 0.7	22.34 ± 0.8
Banding	2	238	21.35 ± 1.3	23.24 ± 1.2	23.20 ± 1.4	23.98 ± 1.8	24.16 ± 1.4
Breast (W)	2	699	5.77 ± 0.8	$(+)2.65 \pm 0.1$	$(+)4.13 \pm 0.1$	5.46 ± 0.5	5.30 ± 0.5
Cleveland	2	303	20.66 ± 1.8	$(+)16.06 \pm 0.7$	$(+)16.07 \pm 0.5$	21.87 ± 1.9	22.21 ± 1.5
Credit	2	690	14.28 ± 0.6	14.53 ± 0.3	14.23 ± 0.1	14.28 ± 0.6	14.30 ± 0.6
Diabetes	2	768	26.46 ± 0.7	$(+)23.87 \pm 0.5$	$(+)22.71 \pm 0.2$	26.14 ± 0.8	25.70 ± 1.0
German	2	1000	27.93 ± 0.7	$(+)24.39 \pm 0.4$	$(+)23.03 \pm 0.5$	28.63 ± 0.7	28.53 ± 0.9
Glass	6	213	30.14 ± 2.4	$(-)37.43 \pm 1.5$	$(-)36.65 \pm 0.8$	31.96 ± 2.6	$(-)33.26 \pm 2.2$
Heart	2	270	23.90 ± 1.8	$(+)15.63 \pm 0.8$	$(+)16.37 \pm 0.4$	22.85 ± 2.0	21.64 ± 1.9
Hepatitis	2	155	19.54 ± 1.5	17.31 ± 1.0	21.60 ± 2.2	20.42 ± 1.6	20.82 ± 2.1
Ionosphere	2	351	9.45 ± 1.1	10.64 ± 0.6	$(-)13.38 \pm 0.8$	10.47 ± 1.2	10.47 ± 1.1
Iris	3	150	4.67 ± 0.9	4.27 ± 0.6	$(+)2.00 \pm 0.0$	4.80 ± 0.9	5.01 ± 1.0
Letter	26	20000	13.17 ± 1.0	$(-)40.34 \pm 0.7$	$(-)29.82 \pm 1.3$	$(+)12.02 \pm 0.7$	$(+)11.57 \pm 0.5$
Monks-1	2	432	6.76 ± 2.1	$(-)25.00 \pm 0.0$	$(-)33.31 \pm 0.0$	$(+)3.52 \pm 1.8$	$(+)1.09 \pm 1.1$
Monks-2	2	432	32.90 ± 0.0	$(-)34.19 \pm 0.6$	$(-)34.21 \pm 0.3$	32.87 ± 0.0	32.90 ± 0.0
Monks-3	2	432	0.00 ± 0.0	$(-)2.77 \pm 0.0$	$(-)22.80 \pm 0.3$	0.00 ± 0.0	0.00 ± 0.0
Mushroom	2	8124	0.00 ± 0.0	$(-)3.85 \pm 0.0$	$(-)6.86 \pm 0.0$	0.00 ± 0.0	0.00 ± 0.0
Satimage	6	6435	13.47 ± 0.2	$(-)19.05 \pm 0.1$	$(-)16.01 \pm 0.1$	13.65 ± 0.4	13.50 ± 0.2
Segment	7	2310	3.55 ± 0.3	$(-)10.20 \pm 0.1$	$(-)8.41 \pm 0.1$	3.29 ± 0.2	3.15 ± 0.3
Sonar	2	208	28.38 ± 2.5	24.95 ± 1.2	25.26 ± 1.2	27.96 ± 3.4	24.71 ± 1.2
Vehicle	4	846	27.48 ± 1.0	$(-)38.73 \pm 0.6$	$(+)22.16 \pm 0.1$	27.10 ± 1.0	26.82 ± 1.1
Votes	2	435	3.34 ± 0.6	$(-)9.74 \pm 0.2$	$(-)5.43 \pm 0.2$	3.65 ± 0.4	3.47 ± 0.3
Waveform	3	2581	24.28 ± 0.8	$(+)18.72 \pm 0.2$	$(+)14.94 \pm 0.2$	24.66 ± 0.4	24.89 ± 0.6
Wine	3	178	7.06 ± 0.6	$(+)2.37 \pm 0.6$	$(+)1.13 \pm 0.5$	6.93 ± 0.6	7.12 ± 0.9

shows the average rank of all base and cascading algorithms, computed for each dataset by assigning rank 1 to the most accurate algorithm, rank 2 to the second best and so on. The remaining lines compares a cascade algorithm against the top-level algorithm. The fourth line shows the number of datasets in which the top-level algorithm was more accurate than the corresponding cascade algorithm, versus the number in which it was less. The fifth line considers only those datasets where the error rate difference was significant at the 1% level, using paired t-tests. The last line shows the *p-values* obtained by applying the Wilcoxon matched-pairs signed-ranks test.

All statistics show that the most promising combinations use a decision tree as high-level classifier and naive Bayes or Discrim as low-level classifiers. The new attributes built by *Discrim* and *naive Bayes* express relations between attributes, that are outside the scope

Table 2. Results of Cascade Generalization. Composite models are compared against its components.

Dataset	*Bay*∇*Bay*	*Bay*∇*Dis*	*Bay*∇*C4.5*	*Dis*∇*Dis*	*Dis*∇*Bay*	*Dis*∇*C4.5*
Adult	(−)18.90 ± 0.7	(++)17.07 ± 0.7	(+−)16.85 ± 0.6	21.93 ± 0.4	(−)21.93 ± 0.4	(−)21.93 ± 0.4
Australian	14.69 ± 0.5	(++)13.61 ± 0.2	(+)14.16 ± 0.6	14.06 ± 0.1	(++)12.72 ± 0.4	(+)14.15 ± 0.7
Balance	7.06 ± 1.1	(+)8.37 ± 0.1	(−−)23.38 ± 1.0	(+)8.41 ± 0.1	(+−)11.44 ± 0.8	(−)22.23 ± 0.8
Banding	22.36 ± 0.9	21.99 ± 0.8	(++)18.76 ± 1.2	23.28 ± 1.4	22.01 ± 1.6	(+)22.33 ± 1.7
Breast	2.83 ± 0.1	(−+)3.26 ± 0.1	(−+)3.42 ± 0.2	4.13 ± 0.1	(+)2.75 ± 0.1	(−)5.08 ± 0.4
Cleveland	17.28 ± 0.9	16.03 ± 0.4	(−+) 20.35 ± 1.5	16.07 ± 0.5	16.35 ± 0.5	(−) 21.77 ± 1.9
Credit	14.91 ± 0.4	(++)13.35 ± 0.3	13.97 ± 0.6	14.22 ± 0.1	(++)13.59 ± 0.4	14.34 ± 0.3
Diabetes	24.71 ± 0.6	(+)22.44 ± 0.3	(−)25.33 ± 0.8	22.71 ± 0.2	23.51 ± 0.6	(−)25.99 ± 0.8
German	(−)25.48 ± 0.6	(+)23.17 ± 0.6	(−)28.56 ± 0.5	23.03 ± 0.5	23.73 ± 0.6	(−)28.58 ± 0.7
Glass	37.48 ± 1.7	35.79 ± 1.7	(+)30.63 ± 2.8	36.25 ± 1.4	35.89 ± 1.6	(+)31.63 ± 2.8
Heart	16.67 ± 0.7	16.30 ± 0.5	(−)21.74 ± 1.5	16.37 ± 0.4	(+)15.56 ± 0.6	(−)22.89 ± 1.9
Hepatitis	15.95 ± 1.6	(+)17.52 ± 1.3	18.44 ± 1.9	21.60 ± 2.0	(+)16.30 ± 1.2	21.15 ± 1.8
Ionosphere	9.76 ± 0.7	(++)9.14 ± 0.3	(++)8.57 ± 0.8	13.38 ± 0.8	(+)10.42 ± 0.4	(+)10.47 ± 1.2
Iris	3.80 ± 0.5	(−)3.27 ± 0.9	(+)3.67 ± 1.0	2.00 ± 0.0	(−+)3.00 ± 0.4	(−)4.73 ± 0.9
Letter	(+)36.59 ± 1.1	(++)25.69 ± 1.0	(+)11.87 ± 0.7	(+)28.14 ± 1.3	(+−)36.42 ± 0.9	(+)11.94 ± 0.8
Monks-1	25.21 ± 0.4	(−)33.33 ± 0.0	(+)3.56 ± 1.8	(−) 41.07 ± 1.5	(+)25.01 ± 0.0	(+−)20.21 ± 4.2
Monks-2	(+)30.31 ± 3.0	34.33 ± 0.9	(−)34.19 ± 0.6	35.06 ± 0.8	34.07 ± 0.6	(−)34.21 ± 0.3
Monks-3	1.76 ± 0.8	(−+)14.13 ± 0.3	(+)0.00 ± 0.0	(+)20.76 ± 0.8	(+)2.77 ± 0.0	(−)22.80 ± 0.3
Mushroom	(+)1.85 ± 0.0	(+)3.13 ± 0.0	(+)0.00 ± 0.0	6.86 ± 0.0	(++)1.77 ± 0.0	(−)6.86 ± 0.0
Satimage	(+)18.82 ± 0.1	(+−)16.57 ± 0.1	(+−)15.61 ± 0.1	(+)15.59 ± 0.1	(++)14.84 ± 0.1	(+)13.63 ± 0.4
Segment	(+)9.41 ± 0.2	(++)7.91 ± 0.1	(+−)3.78 ± 0.3	(+)7.93 ± 0.1	(−+)9.29 ± 0.1	(+)3.27 ± 0.2
Sonar	25.59 ± 1.4	(++)23.72 ± 1.1	(++)21.84 ± 2.0	24.81 ± 1.2	(+)24.93 ± 1.4	25.96 ± 2.1
Vehicle	39.16 ± 1.0	(+−)25.34 ± 0.7	(+−)28.52 ± 0.8	22.00 ± 0.3	(−+)23.54 ± 0.9	(−+)26.33 ± 1.2
Votes	10.00 ± 0.3	(+)5.28 ± 0.2	(+−)4.41 ± 0.3	5.43 ± 0.2	(+)5.43 ± 0.2	(+)3.56 ± 0.5
Waveform	(+)16.42 ± 0.3	(+)15.24 ± 0.2	(−+)21.80 ± 0.7	(+)4.45 ± 0.2	(−+)16.93 ± 0.4	(−)24.65 ± 0.4
Wine	2.62 ± 0.6	1.06 ± 0.6	(−+)4.14 ± 0.9	1.31 ± 0.7	2.01 ± 0.7	(−)5.83 ± 1.2

of DNF algorithms like C4.5. These new attributes systematically appear at the root of the composite models.

One of the main problems when combining classifiers is: *Which algorithms should we combine?* The empirical evaluation suggests:

- Combine classifiers with different behavior from a *Bias-Variance* analysis.
- At low level use algorithms with low variance.
- At high level use algorithms with low bias.

On *Cascade* framework lower level learners delay the final decision to the high level learners. Selecting learners with low *bias* for high level, we are able to fit more complex decision surfaces, taking into account the "*stable*" surfaces drawn by the low level learners.

Given equal performance, we would prefer fewer component classifiers, since training, and application times will be lower for smaller number of components. Larger number of

Table 3. Results of Cascade Generalization. Composite models are compared against its components.

Dataset	*C4.5∇C4.5*	*C4.5∇Dis*	*C4.5∇Bay*	*C4.5∇Disc∇Bay*	*C4.5∇Bay∇Disc*	*Stacked Gen.*
Adult	(+)13.85 ± 0.5	(+)13.92 ± 0.5	(+)13.72 ± 0.3	(++)13.71 ± 0.3	(++)13.76 ± 0.2	13.96 ± 0.6
Australian	14.74 ± 0.5	13.99 ± 0.9	15.41 ± 0.8	14.24 ± 0.5	15.34 ± 0.9	13.99 ± 0.4
Balance	21.87 ± 0.7	(++)5.42 ± 0.7	(++)4.78 ± 1.1	(+++)5.34 ± 1.0	(+++)6.77 ± 0.6	(−)7.76 ± 0.9
Banding	23.77 ± 1.7	21.73 ± 2.5	22.75 ± 1.8	21.48 ± 2.0	22.18 ± 1.5	21.45 ± 1.2
Breast	5.36 ± 0.5	(+)4.13 ± 0.1	(+)2.61 ± 0.1	(++)2.62 ± 0.1	(++)2.69 ± 0.2	2.66 ± 0.1
Cleveland	22.27 ± 2.2	(−)19.93 ± 1.0	(+)18.31 ± 1.1	18.25 ± 2.2	(−−)20.54 ± 1.6	(+)16.75 ± 0.9
Credit	14.21 ± 0.6	13.85 ± 0.4	15.07 ± 0.7	14.84 ± 0.4	13.75 ± 0.6	(+)13.43 ± 0.6
Diabetes	26.05 ± 1.0	(+−)24.51 ± 0.9	(−)26.06 ± 0.7	(−)25.02 ± 0.9	(−−)25.48 ± 1.4	
German	28.61 ± 0.7	(+−)24.60 ± 1.0	(+−)26.20 ± 1.1	(+−−)26.28 ± 1.0	(+−)25.92 ± 1.0	(+)24.72 ± 0.3
Glass	32.02 ± 2.4	36.09 ± 1.8	33.60 ± 1.6	34.68 ± 1.8	35.11 ± 2.5	31.28 ± 1.9
Heart	23.19 ± 1.9	(+)18.48 ± 1.5	(−)19.30 ± 1.9	(+−−)18.67 ± 1.0	(+−−)18.89 ± 1.1	(++)16.19 ± 0.9
Hepatitis	20.60 ± 1.5	19.89 ± 2.3	(+)16.63 ± 1.3	(++)15.97 ± 1.9	17.21 ± 2.1	15.33 ± 1.1
Ionosphere	10.21 ± 1.3	(+)10.79 ± 0.8	11.55 ± 1.0	(+)11.28 ± 0.8	(+)10.98 ± 0.6	9.63 ± 1.2
Iris	4.80 ± 0.9	(−)3.53 ± 0.8	5.00 ± 0.8	(−)4.53 ± 0.9	(−)4.13 ± 1.1	4.13 ± 1.0
Letter	11.79 ± 0.7	(−+)13.72 ± 0.8	(−+)13.69 ± 0.8	(−++)14.41 ± 0.5	(−++)14.99 ± 0.9	(++)11.91 ± 0.7
Monks-1	2.70 ± 0.8	(+)3.52 ± 1.8	(+)1.49 ± 1.7	(+++)0.78 ± 1.2	(++)1.20 ± 1.2	(−)3.74 ± 2.0
Monks-2	32.87 ± 0.0	(+)32.87 ± 0.0	(++)8.99 ± 2.6	(+++)8.99 ± 2.6	(+++)8.89 ± 2.8	(−−)32.87 ± 0.0
Monks-3	0.00 ± 0.0	(+)0.00 ± 0.0	(−+)0.60 ± 0.4	(−++)0.60 ± 0.4	(−++)0.81 ± 0.5	(−−)2.24 ± 0.8
Mushroom	0.00 ± 0.0	(+)0.00 ± 0.0	(−+)0.14 ± 0.0	(−++)0.15 ± 0.0	(−++)0.04 ± 0.0	(−−)2.93 ± 0.04
Satimage	13.58 ± 0.5	(++)12.18 ± 0.4	(+)13.06 ± 0.4	(+++)12.83 ± 0.3	(+++)12.22 ± 0.3	(−)13.11 ± 0.4
Segment	3.21 ± 0.2	(+)3.09 ± 0.1	(+)3.67 ± 0.3	(++)3.44 ± 0.3	(++)3.40 ± 0.2	3.32 ± 0.2
Sonar	28.02 ± 3.2	24.75 ± 2.9	24.36 ± 1.9	24.45 ± 1.8	23.83 ± 2.1	24.81 ± 1.0
Vehicle	26.96 ± 0.8	(+)22.36 ± 0.9	(+)28.35 ± 1.3	(+−+)23.97 ± 0.9	(++−)24.28 ± 1.0	(−−)27.72 ± 0.8
Votes	(+)3.17 ± 0.5	(−+)4.41 ± 0.5	(+)3.70 ± 0.3	(−+)4.62 ± 0.7	(−++)4.45 ± 0.5	(++)3.65 ± 0.4
Waveform	24.66 ± 0.3	(+−)16.86 ± 0.3	(++)17.30 ± 0.5	(+−+)16.75 ± 0.4	(+−+)15.94 ± 0.3	(−)17.14 ± 0.4
Wine	6.99 ± 0.7	(+−)4.25 ± 0.6	(+)2.97 ± 0.9	(+−)2.50 ± 0.6	(+−)2.26 ± 0.7	2.23 ± 0.9

Table 4. Summary of results of Cascade Generalization.

Measure	Bayes	Bay∇Bay	Bay∇Dis	Bay∇C4	Disc	Disc∇Disc	Disc∇Bay	Disc∇C4
Arithmetic mean	17.62	17.29	16.42	15.29	17.80	17.72	16.39	17.94
Geometric mean	13.31	12.72	12.61	10.71	13.97	13.77	12.14	14.82
Average rank	9.67	9.46	6.63	7.52	9.06	8.77	7.23	10.29
Nr. of wins	–	14/12	8/18	9/16	–	4/10	10/16	14/9
Significant wins	–	2/6	3/13	8/13	–	1/6	4/10	10/7
Wilcoxon Test	–	0.49	0.02	0.16	–	0.19	0.16	0.46

Table 5. Summary of results of Cascade Generalization.

Measure	*C4.5*	*C4.5*∇*C4.5*	*C4.5*∇*Bay*	*C4.5*∇*Dis*	*C4.5*∇*Dis*∇*Bay*	*C4.5*∇*Bay*∇*Dis*
Arithmetic mean	15.98	15.98	13.44	14.19	13.09	13.27
Geometric mean	11.40	11.20	8.25	9.93	7.95	7.81
Average rank	9.83	9.04	7.85	6.17	6.46	6.69
Nr. of wins	–	7/15	4/19	11/15	8/18	8/18
Significant wins	–	0/2	3/8	2/9	4/11	4/9
Wilcoxon Test	–	0.17	0.04	0.005	0.005	0.008

components has also adverse affects in comprehensibility. In our study the version with three components seemed perform better than the version with two components. More research is needed to establish the limits of extending this scenario.

5.4. *Comparison with Stacked Generalization*

We have compared various versions of Cascade Generalization to Stacked Generalization, as defined in Ting and Witten (1997). In our re-implementation of *Stacked Generalization* the *level_0* classifiers were C4.5 and Bayes, and the *level_1* classifier was *Discrim*. The attributes for the *level_1* data are the probability class distributions, obtained from the *level_0* classifiers using a 5-fold stratified cross-validation.[3] Table 3 shows, in the last column, the results of *Stacked Generalization*. *Stacked Generalization* is compared, using *paired t-tests*, to *C4.5*∇ *Discrim*∇ *naive Bayes* and *C4.5*∇ *naive Bayes*∇ *Discrim* in this order. The $+(-)$ sign indicates that for this dataset the Cascade model performs significantly better (worse). Table 6 presents a summary of results. They provide evidence that the generalization ability of Cascade Generalization models is competitive with Stacked Generalization that computes the *level_1* attributes using internal cross-validation. The use of internal cross-validation affects of course the learning times. Both Cascade models are at least three times faster than Stacked Generalization.

Table 6. Summary of comparison against Stacked Generalization.

	Stacked Generalization	*C4.5*∇ *Discrim*∇*Bayes*	*C4.5*∇*Bayes*∇*Discrim*
Arithmetic mean	13.87	13.09	13.27
Geometric mean	10.13	7.95	7.81
Average rank	1.8	2.1	2.1
	C4.5∇ *Disc*∇*Bay vs. Stack.G.*		*C4.5*∇*Bay*∇*Disc vs. Stack.G.*
Number of wins	11/15		10/16
Significant wins	6/5		6/4
Wilcoxon Test	0.71		0.58

Cascade Generalization exhibits good generalization ability and is computationally efficient. Both aspects lead to the hypothesis: *Can we improve Cascade Generalization by applying it at each iteration of a divide-and-conquer algorithm?* This hypothesis is examined in the next section.

6. Local Cascade Generalization

Many classification algorithms use a divide and conquer strategy that resolve a given complex problem by dividing it into simpler problems, and then by applying recursively the same strategy to the subproblems. Solutions of subproblems are combined to yield a solution of the original complex problem. This is the basic idea behind the well known decision tree based algorithms: ID3 (Quinlan, 1986), CART (Breiman et al., 1984), ASSISTANT (Kononenko et al., 1987), C4.5 (Quinlan, 1993). The power of this approach derives from the ability to split the hyper space into subspaces and fit each subspace with different functions. In this Section we explore *Cascade Generalization* on the problems and subproblems that a *divide and conquer* algorithm generates. The intuition behind this proposed method is the same as behind any *divide and conquer* strategy. The relations that can not be captured at global level can be discovered on the simpler subproblems.

In the following sections we present in detail how to apply *Cascade Generalization* locally. We will only develop this strategy for decision trees, although it should be possible to use it in conjunction with any *divide and conquer* method, like *decision lists* (Rivest, 1987).

6.1. The local Cascade Generalization algorithm

Local Cascade Generalization is a composition of classification algorithms that is elaborated when building the classifier for a given task. In each iteration of a divide and conquer algorithm, *Local Cascade Generalization* extends the dataset by the insertion of new attributes. These new attributes are propagated down to the subtasks. In this paper we restrict the use of *Local Cascade Generalization* to decision tree based algorithms. However, it should be possible to use it with any *divide-and-conquer* algorithm. Figure 2 presents the general algorithm of *Local Cascade Generalization*, restricted to a decision tree. The method will be referred to as *CGTree*.

When growing the tree, new attributes are computed at each decision node by applying the Φ operator. The new attributes are propagated down the tree. The number of new attributes is equal to the number of classes appearing in the examples at this node. This number can vary at different levels of the tree. In general deeper nodes may contain a larger number of attributes than the parent nodes. This could be a disadvantage. However, the number of new attributes that can be generated decreases rapidly. As the tree grows and the classes are discriminated, deeper nodes also contain examples with a decreasing number of classes. This means that as the tree grows the number of new attributes decreases.

In order to be applied as a predictor, any *CGTree* must store, in each node, the model generated by the base classifier using the examples at this node. When classifying a new example, the example traverses the tree in the usual way, but at each decision node it is

Input: A data set D, a classifier $\Im$
Output: A decision tree

Function CGtree(D, $\Im$)
  If stopping criteria(D) = TRUE
    return a leaf with class probability distribution
  Else
    $D' = \Phi(D, \mathcal{A}(\Im(D), D))$
    Choose the attribute A_i that maximizes splitting criterion on D'
    For each partition of examples based on the values of attribute A_i
      generate a subtree: $Tree_i$ = CGtree(D'_i, $\Im$)
    return Tree containing a decision node based on attribute Ai,
      storing $\Im(D)$ and descendant subtrees $Tree_i$
  EndIf
End

Figure 2. Local cascade algorithm based on a decision tree.

extended by the insertion of the probability class distribution provided the base classifier predictor at this node.

In the framework of local cascade generalization, we have developed a *CGLtree*, that uses the $\Phi(D, \mathcal{A}(Discrim(D), D))$ operator in the constructive step. Each internal node of a *CGLtree* constructs a discriminant function. This discriminant function is used to build new attributes. For each example, the value of a new attribute is computed using the linear discriminant function. At each decision node, the number of new attributes built by *CGLtree* is always equal to the number of classes taken from the examples at this node. In order to restrict attention to well populated classes, we use the following heuristic: we only consider a $class_i$ if the number of examples, at this node, belonging to $class_i$ is greater than N times the number of attributes.[4] By default N is 3. This implies that at different nodes, different number of classes will be considered leading to addition of a different number of new attributes. Another restriction to the use of the constructive operator $\mathcal{A}(\Im(D), D)$, is that the error rate of the resulting classifier should be less than 0.5 in the training data.

In our empirical study we have used two other algorithms that apply *Cascade Generalization* locally. The first one is *CGBtree* that uses as constructive operator

$$\Phi(D, \mathcal{A}(naiveBayes(D), D)),$$

and the second one is *CGBLtree* that uses as constructive operator:

$$\Phi_p(D, [\mathcal{A}(naiveBayes(D), D), \mathcal{A}(Discrim(D), D)]),$$

In all other aspects these algorithms are similar to *CGLtree*.

There is one restriction to the application of the $\Phi(D', \mathcal{A}(\Im(D), D'))$ operator: the induced classifier $\Im(D)$ must return the corresponding probability class distribution for each

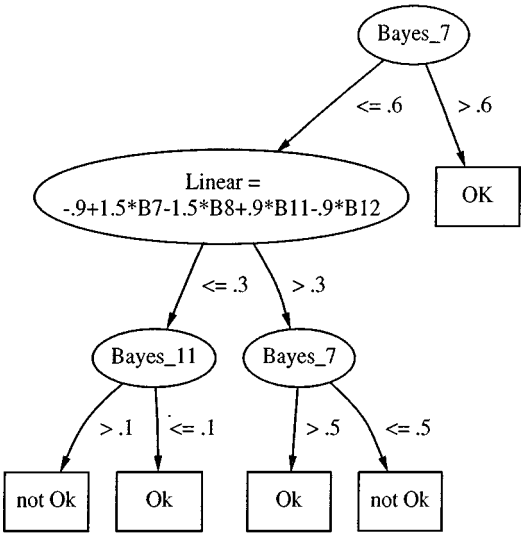

Figure 3. Tree generated by a CGTree using Discrim∇Bayes as constructive operator.

$\vec{x} \in D'$. Any classifier that satisfies these requisites could be applied. It is possible to imagine a *CGTree*, whose internal nodes are trees themselves. For example, small modifications to C4.5.[5] enables the construction of a *CGTree* whose internal nodes are trees generated by C4.5.

6.2. An illustrative example

Figure 3 represents the tree generated by a *CGTree* on the *Monks-2* problem. The constructive operator used is: $\Phi(D, Discrim\nabla Bayes(\vec{x}, D))$. At the root of the tree the *naive Bayes* algorithm provides two new attributes—*Bayes_7* and *Bayes_8*. The *linear discriminant* uses continuous attributes only. There are only two continuous attributes, those built by the *naive Bayes*. In this case, the coefficients of the linear discriminant shrink to zero by the process of variable elimination used by the discriminant algorithm. The *gain ratio* criterion chooses the *Bayes_7* attribute as a test. The dataset is split into two partitions. One of them contains only examples from class *OK*: a leaf is generated. In the other partition two new *Bayes* attributes are built (Bayes_11, Bayes_12) and so a linear discriminant is generated based on these two *Bayes* attributes and on those built at the root of the tree. The attribute based on the linear discriminant is chosen as test attribute for this node. The dataset is segmented and the process of tree construction proceeds.

This example illustrate two points:

- The interactions between classifiers: The linear discriminant contains terms built by *naive Bayes*. Whenever a new attribute is built, it is considered as a regular attribute. Any attribute combination built at deeper nodes can contain terms based on the attributes built at upper nodes.

- Re-use of attributes with different thresholds. The attribute *Bayes_7*, built at the root, is used twice in the tree with different thresholds.

6.3. Relation to other work on multivariate trees

With respect to the final model, there are clear similarities between *CGLtree* and *Multivariate trees* of Brodley and Utgoff (1995). Langley refers that any multivariate tree is topologically equivalent to a three-layer *inference network* Langley (1996). The constructive ability of our system is similar to the *Cascade Correlation Learning architecture* of Fahlman and Lebiere (1991). Also the final model of *CGBtree* is related with the *recursive naive Bayes* presented by Langley (1996). This is an interesting feature of *Local Cascade Generalization*: it unifies in a single framework several systems from different research areas. In our previous work (Gama & Brazdil, 1999) we have compared system *Ltree*, similar to *CGLtree*, with *Oc1* of Murthy et al. (1994), *LMDT* of Brodley et al. (1995), and *CART* of Breiman et al. The focus of this paper is on methodologies for combining classifiers. As such, we only compare our algorithms against other methods that generate and combine multiple models.

7. Evaluation of local Cascade Generalization

In this section we evaluate three instances of local Cascade Algorithms: *CGBtree*, *CGLtree*, and *CGBLtree*. We compare the local versions against its corresponding global models, and against two standard methods to combine classifiers: Boosting and Stacked Generalization. All the implemented *Local Cascade Generalization* algorithms are based on *Dtree*. They use exactly the same splitting criteria, stopping criteria, and pruning mechanism. Moreover they share many minor heuristics that individually are too small to mention, but collectively can make difference. At each decision node, *CGLtree* applies the *Linear discriminant* described above, while *CGBtree* applies the *naive Bayes* algorithm. *CGBLtree* applies the *Linear discriminant* to the *ordered* attributes and the *naive Bayes* to the *categorical* attributes. In order to prevent *overfitting* the construction of new attributes is constrained to a depth of 5. In addition, the level of pruning is greater than the level of pruning in *Dtree*.

Table 7 presents the results of *local Cascade Generalization*. Each column corresponds to a local Cascade Generalization algorithm. Each algorithm is compared against its similar *Cascade* model using *paired t-tests*. For example, *CGLtree* is compared against *C4.5*∇ *Discrim*. A $+(-)$ sign means that the error rate of the composite model is, at statistically significant levels, lower (higher) than the correspondent model. Table 8 presents a comparative summary of the results between local Cascade Generalization and the corresponding global models. It illustrates the benefits of applying Cascade Generalization locally.

System *CGBLtree* is compared to *C5.0Boosting*, a variance reduction method.[6] and to *Stacked Generalization*, a bias reduction method. Table 7 presents the results of *C5.0Boosting* with the default parameter of 10, that is aggregating over 10 trees, and *Stacked Generalization* as it is defined in Ting and Witten (1997) and described in an earlier section. Both Boosting and Stacked are compared against *CGBLtree*, using *paired t-tests* with the significance level set to 99.9%. A $+(-)$ sign means that Boosting or *Stacked* performs significantly

Table 7. Results of Local Cascade Generalization, Boosting and Stacked, Boosting a Cascade algorithm. The footnote indicates the models used in comparison.

Dataset	*CGBtree*[a]	*CGLtree*[a]	*CGBLtree*[a]	*C5.0Boost*[b]	*Stacked*[b]	*C5B∇ Bayes*[c]
Adult	13.46 ± 0.4	13.56 ± 0.3	13.52 ± 0.4	-14.33 ± 0.4	13.96 ± 0.6	14.41 ± 0.5
Australian	14.45 ± 0.7	-14.69 ± 0.9	13.95 ± 0.7	13.21 ± 0.7	13.99 ± 0.4	13.96 ± 0.9
Balance	5.32 ± 1.1	8.24 ± 0.5	-8.08 ± 0.4	-20.03 ± 1.0	7.75 ± 0.9	$(+)4.25 \pm 0.7$
Banding	20.98 ± 1.2	23.60 ± 1.2	20.69 ± 1.2	$(+)17.39 \pm 1.7$	21.45 ± 1.2	18.38 ± 1.8
Breast (W)	2.62 ± 0.1	$(+)3.23 \pm 0.4$	2.66 ± 0.1	-3.34 ± 0.3	2.66 ± 0.1	3.03 ± 0.2
Cleveland	$(+)15.10 \pm 1.4$	$(+)16.54 \pm 0.8$	16.50 ± 0.8	-18.95 ± 1.3	16.75 ± 0.9	17.86 ± 1.1
Credit	15.35 ± 0.5	14.41 ± 0.8	14.52 ± 0.8	13.41 ± 0.8	13.43 ± 0.6	13.57 ± 0.9
Diabetes	25.37 ± 1.5	24.43 ± 0.9	24.48 ± 0.9	24.58 ± 0.9	23.62 ± 0.4	24.71 ± 1.1
German	25.37 ± 1.1	24.78 ± 1.1	$(+)24.88 \pm 0.8$	25.36 ± 0.8	24.72 ± 0.3	25.20 ± 1.1
Glass	32.08 ± 2.5	34.71 ± 2.3	32.35 ± 2.0	$(+)25.06 \pm 2.0$	31.28 ± 1.8	-29.23 ± 1.6
Heart	$(+)16.37 \pm 1.0$	16.85 ± 1.2	$(+)16.81 \pm 1.1$	-19.94 ± 1.3	16.19 ± 0.9	$(+)17.24 \pm 1.5$
Hepatitis	16.87 ± 1.1	$(+)16.87 \pm 1.1$	16.87 ± 1.1	16.67 ± 1.5	15.33 ± 1.1	15.93 ± 1.3
Ionosphere	9.62 ± 0.9	11.06 ± 0.6	11.00 ± 0.7	$(+)6.57 \pm 1.1$	9.63 ± 1.2	7.71 ± 0.7
Iris	4.73 ± 1.3	2.80 ± 0.4	$(+)2.80 \pm 0.4$	-5.68 ± 0.6	4.13 ± 1.0	5.07 ± 0.8
Letter	13.47 ± 0.9	12.97 ± 0.9	$(+)13.06 \pm 0.9$	$(+)5.16 \pm 0.4$	$(+)11.91 \pm 0.7$	-6.91 ± 0.5
Monks-1	-9.39 ± 3.5	6.80 ± 3.3	-8.53 ± 3.0	$(+)0.00 \pm 0.0$	$(+)3.74 \pm 2.0$	0.33 ± 0.3
Monks-2	-14.87 ± 3.3	33.19 ± 1.7	11.88 ± 3.3	-35.76 ± 1.0	-32.87 ± 0.0	$(+)3.64 \pm 1.7$
Monks-3	0.39 ± 0.4	-0.92 ± 0.5	0.39 ± 0.4	0.00 ± 0.0	-2.24 ± 0.8	0.63 ± 0.3
Mushroom	0.22 ± 0.1	-0.96 ± 0.1	-0.24 ± 0.0	$(+)0.00 \pm 0.0$	-2.93 ± 0.0	0.01 ± 0.0
Satimage	$(+)11.86 \pm 0.3$	11.99 ± 0.3	$(+)11.99 \pm 0.3$	$(+)9.25 \pm 0.2$	-13.11 ± 0.4	9.21 ± 0.2
Segment	-4.36 ± 0.2	3.19 ± 0.3	3.20 ± 0.3	$(+)1.79 \pm 0.1$	3.32 ± 0.2	-2.09 ± 0.1
Sonar	26.23 ± 1.7	25.26 ± 1.5	25.50 ± 1.6	$(+)19.25 \pm 2.2$	24.81 ± 1.0	23.02 ± 1.2
Vehicle	28.75 ± 0.8	21.21 ± 0.9	$(+)21.32 \pm 0.9$	-23.71 ± 0.7	-27.72 ± 0.8	24.29 ± 1.3
Votes	3.29 ± 0.4	4.30 ± 0.5	$(+)3.26 \pm 0.4$	4.11 ± 0.4	3.65 ± 0.4	4.30 ± 0.4
Waveform	16.50 ± 0.6	$(+)15.74 \pm 0.5$	16.12 ± 0.5	-17.45 ± 0.4	-17.14 ± 0.4	$(+)15.58 \pm 0.3$
Wine	2.31 ± 0.6	$(+)1.20 \pm 0.6$	$(+)1.20 \pm 0.6$	-3.45 ± 1.0	2.23 ± 0.9	2.94 ± 0.7

[a]vs. Corresponding Cascade Models.
[b]vs. *CGBLtree*.
[c]vs. *C5.0Boost*.

better (worse) than *CGBLtree*. In this study, *CGBLtree* performs significantly better than Stacked, in 6 datasets and worse in 2 datasets.

7.1. A step ahead

Comparing with *C5.0Boosting*, *CGBLtree* significantly improves in 10 datasets and loses in 9 datasets. It is interesting to note that in 26 datasets there are 19 significant differences. This is evidence that Boosting and Cascade have different behavior. The improvement observed with Boosting, when applied to a decision tree, is mainly due to the reduction of the

Table 8. Summary of results of local Cascade Generalization.

	CGBtree	*CGLtree*	*CGBLtree*	*C5.0Boost*	*Stacked G.*	*C5B∇ Bayes*
Arithmetic mean	13.43	13.98	12.92	13.25	13.87	11.63
Geometric mean	8.70	9.46	8.20	8.81	10.13	6.08
Average rank	3.90	3.92	3.29	3.27	3.50	3.12
		C4.5∇ Bay *vs* *CGBtree*	C4.5∇ Dis *vs* *CGLtree*	C4∇ Bay∇ Dis *vs* *CGBLtree*	CGBLtree *vs* *C5.0Boost*	CGBLtree *vs* *Stacked G.*
Number of wins		10–16	12–14	7–19	13–13	15–11
Significant wins		3–3	3–5	3–8	10–9	6–2
Wilcoxon Test		0.18	0.49	0.07	0.86	0.61

variance component of the error rate while, with Cascade algorithms, the improvement is mainly due to the reduction on the *bias* component. Table 7 presents the results of Boosting a *Cascade* algorithm. In this case we have used the global combination C5.0 Boost∇ naive Bayes. It improves over *C5.0Boosting* on 4 datasets and loses in 3. The summary of the results presented in Table 8 evidence a promising result, and we intend, in the near future, to *boost CGBLtree*.

7.2. Number of leaves

Another dimension for comparisons involves measuring the number of leaves. This corresponds to the number of different regions into which the instance space is partitioned by the algorithm. Consequently it can be seen as an indicator of the model complexity. In almost all datasets,[7] any Cascade tree splits the instance space into half of the regions needed by *Dtree* or *C5.0*. This is a clear indication that Cascade models capture better the underlying structure of the data.

7.3. Learning times

Learning time is the other dimension for comparing classifiers. Here comparisons are less clear as results may strongly depend on the implementation details as well on the underlying hardware. However at least the order of magnitude of time complexity is a useful indicator.

C5.0 and *C5.0Boosting* have run on a *Sparc 10* machine.[8] All the other algorithms have run on a *Pentium 166 MHz, 32 Mb* machine under *Linux*. Table 9 presents the average time

Table 9. Relative learning times of base and composite models.

Bayes	*Discrim*	*C4.5*	*Bay∇Dis*	*Dis∇Dis*		*Dtree*	*Bay∇Bay*	*Dis∇Bay*	*Bay∇C4*	*Dis∇C4*
1	*1.04*	*2.35*	*2.67*	*2.75*	*C5.0 2.77*		*2.86*	*3.31*	*3.37*	*3.59*	*3.65*

C4∇Dis	*C4∇C4*	*C4∇Bay*	*CGBtree*	*C4∇Dis∇Bay*	*CGLtree*	*CGBLtree*	*C5.0Boost*	*Stacked*
4.1	4.55	4.81	6.70	6.85	7.72	11.08	15.16	15.29

needed by each algorithm to run on all datasets, taking the time of *naive Bayes* as reference. Our results demonstrate that any *CGTree* is faster than *C5.0Boosting*. *C5.0Boosting* is slower because it generates 10 trees with increased complexity. Also, any *CGTree* is faster than *Stacked Generalization*. This is due to the internal cross validation used in *Stacked Generalization*.

8. Why does Cascade Generalization improve performance?

Both Cascade Generalization and Local Cascade Generalization transforms the instance space into a new, high-dimensional space. In principle this could turn the given learning problem into a more difficult one. This phenomenon is known as the *curse of dimensionality* Mitchell (1997). In this section we analyze the behavior of Cascade Generalization through three dimensions: the error correlation, the bias-variance analysis, and Mahalanobis distances.

8.1. Error correlation

Ali and Pazzani (1996) have shown that a desirable property of an ensemble of classifiers is *diversity*. They use the concept of *error correlation* as a metric to measure the degree of diversity in an ensemble. Their definition of *error correlation* between two classifiers is defined as the probability that both make the same error. Because this definition does not satisfy the property that the correlation between an object and itself should be 1, we prefer to define the error correlation between two classifiers as the conditional probability of the two classifiers make the same error given that one of them makes an error. This definition of *error correlation* lies in the interval $[0:1]$ and the correlation between one classifier and itself is 1. Formally:

$$\phi_{ij} = p(\hat{f}_i(x) = \hat{f}_j(x) \mid \hat{f}_i(x) \neq f(x) \vee \hat{f}_j(x) \neq f(x)). \tag{4}$$

The formula that we use provides higher values than the one used by Ali and Pazzani. As it was expected the lowest degree of correlation is between *decision trees* and *Bayes* and between *decision trees* and *discrim*. They use very different representation languages. The error correlation between *Bayes* and *discrim* is a little higher. Despite the similarity of the two algorithms, they use very different search strategies.

Table 10 presents the results of this analysis. These results provide evidence that the decision tree and any discriminant function make uncorrelated errors, that is each classifier make errors in different regions of the instance space. This is a desirable property for combining classifiers.

Table 10. Error correlation between base classifiers.

	C4 vs. Bayes	*C4 vs. Discrim*	*Bayes vs. Discrim*
Average	0.32	0.32	0.40

8.2. Bias-variance decomposition

The *bias-variance* decomposition of the error is a tool from the statistics theory for analyzing the error of supervised learning algorithms.

The basic idea, consists of decomposing the expected error into three components:

$$E(C) = \sum_x P(x)\left(\sigma^2 + bias_x^2 + variance_x\right) \tag{5}$$

To compute the terms *bias* and *variance* for zero-one loss functions we use the decomposition proposed by Kohavi and Wolpert (1996). The *bias* measures how closely average guess of the learning algorithm matches the target. It is computed as:

$$bias_x^2 = \frac{1}{2}\sum_{y\in Y}\left(P(Y_F = y) - P(Y_H = y)\right)^2 \tag{6}$$

The *variance* measures how much the learning algorithm's guess "bounces around" for the different sets of the given size. This is computed as:

$$variance_x = \frac{1}{2}\left(1 - \sum_{y\in Y} P(Y_H = y)^2\right) \tag{7}$$

To estimate the bias and variance, we first split the data into training and test sets. From the training set we obtain ten bootstrap replications used to build ten classifiers. We ran the learning algorithm on each of the training sets and estimated the terms of the variance Eq. (7) and bias[9] Eq. (6) using the generated classifier for each point x in the evaluation set E. All the terms were estimated using frequency counts.

The base algorithms used in the experimental evaluation have different behavior under a Bias-Variance analysis. A decision tree is known to have low bias but high variance, and naive Bayes and linear discriminant are known to have low variance but high bias.

Our experimental evaluation has shown that the most promising combinations use a decision tree as high level classifier, and naive Bayes or linear discriminant as low level classifiers. To illustrate these results, we measure the bias and the variance of C4.5, naive Bayes and C4.5∇naive Bayes in the datasets under study. These results are shown in figure 4. A summary of the results is presented in Table 11. The benefits of the Cascade

Table 11. Bias variance decomposition of error rate.

	C4.5	*Bayes*	*C45∇Bayes*
Average variance	4.8	1.59	4.72
Average bias	11.53	15.19	8.64

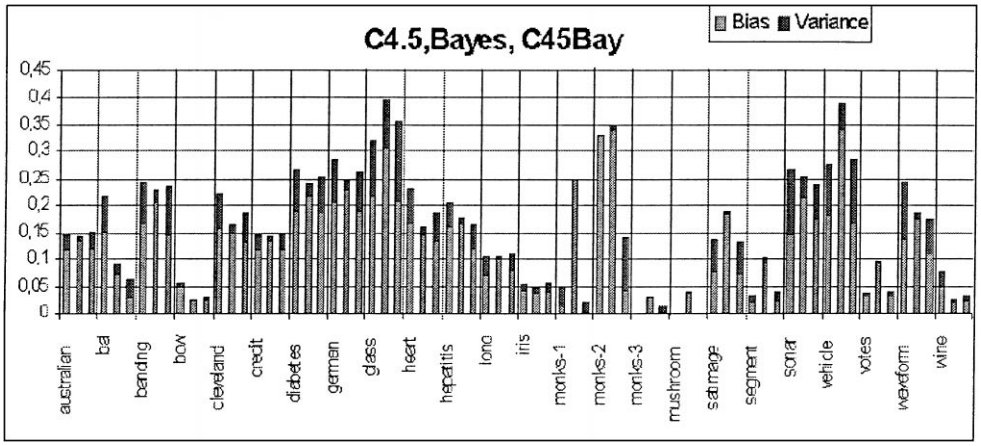

Figure 4. Bias-Variance decomposition of the error rate for C4.5, Bayes and C4.5∇Bayes for different datasets.

composition are well illustrated in datasets like Balance-scale, Hepatitis, Monks-2, Waveform, and Satimage. Comparison between Bayes and C4.5∇Bayes shows that the latter combination obtain a strong reduction of the bias component at costs of increasing the variance component. C4.5∇Bayes reduces both bias and variance when compared to C4.5. The reduction of the error is mainly due to the reduction of bias.

8.3. Mahalanobis distance

Consider that each class defines a single cluster[10] in an *Euclidean space*. For each class i, the centroid of the corresponding cluster is defined as the vector of attribute means $\bar{x}_i$, which is computed from the examples of that class. The shape of the cluster is given by the covariance matrix S_i.

Using the *Mahalanobis* metric we can define two distances:

1. The *within-class* distance. It is defined as the *Mahalanobis* distance between an example and the centroid of its cluster. It is computed as:

$$\left(\vec{\bar{x}}_i - \vec{x}\right)^T S_i^{-1} \left(\vec{\bar{x}}_i - \vec{x}\right) \tag{8}$$

   where $\vec{x}$ represents the example attribute vector, $\vec{\bar{x}}_i$ denotes the centroid of the cluster corresponding to class i, and S_i is the covariance matrix for class i.
2. The *between-classes* distance. It is defined as the *Mahalanobis* distance between two clusters. It is computed as:

$$\left(\vec{\bar{x}}_i - \vec{\bar{x}}_j\right)^T S_{pooled}^{-1} \left(\vec{\bar{x}}_i - \vec{\bar{x}}_j\right) \tag{9}$$

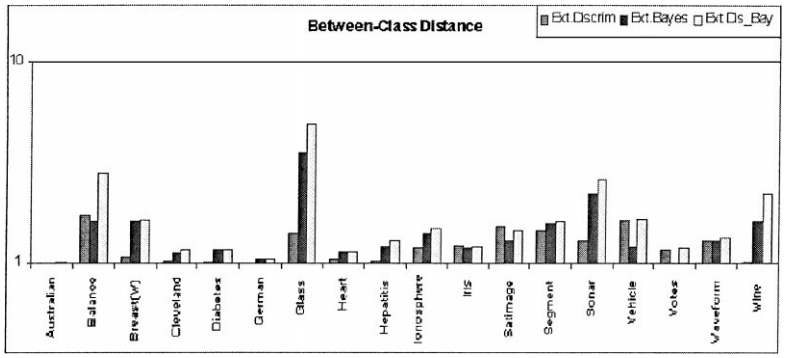

Figure 5. Average increase of between-class distance.

where $\vec{x}_i$ denotes the centroid of the cluster corresponding to class i, and S_{pooled} is the pooled covariance matrix using S_i and S_j.

The intuition behind the within-class distance is that smaller values leads to more compact clusters. The intuition behind the between-classes distance is that larger values would lead us to believe that the groups are sufficiently spread in terms of separation of means.

We have measured the between-classes distance and the within-class distance for the datasets with all numeric attributes. Both distances have been measured in the original dataset and in the dataset extended using a Cascade algorithm. We observe that while the within-class distance remains almost constant, the between-classes distance increases. For example, when using the constructive operator Discrim∇Bay the between-classes distance almost doubles. Figure 5 shows the average increase of the between-class distance, with respect to the original dataset, after extending it using Discrim, Bayes and Discrim∇Bayes, respectively.

9. Conclusions and future work

This paper provides a new and general method for combining learning models by means of constructive induction. The basic idea of the method is to use the learning algorithms in sequence. At each iteration a two step process occurs. First a model is built using a base classifier. Second, the instance space is extended by the insertion of new attributes. These are generated by the built model for each given example. The constructive step generates terms in the representational language of the base classifier. If the high level classifier chooses one of these terms, its representational power has been extended. The *bias* restrictions of the high level classifier is relaxed by incorporating terms of the representational language of the base classifiers. This is the basic idea behind the *Cascade Generalization* architecture.

We have examined two different schemes of combining classifiers. The first one provides a loose coupling of classifiers while the second one couples classifiers tightly:

1. *Loose coupling: Base classifier(s) pre-process data for another stage*. This framework can be used to combine most of the existing classifiers without changes, or with rather small changes. The method only requires that the original data is extended by the insertion of the probability class distribution that must be generated by the base classifier.
2. *Tight coupling through local constructive induction*. In this framework two or more classifiers are coupled locally. Although in this work we have used only *Local Cascade Generalization* in conjunction with decision trees the method could be easily extended to other *divide-and-conquer* systems, such as *decision lists*.

Most of the existing methods such as *Bagging* and *Boosting* that combine learned models, use a voting strategy to determine the final outcome. Although this leads to improvements in accuracy, it has strong limitations—loss in interpretability. Our models are easier to interpret particularly if classifiers are loosely coupled. The final model uses the representational language of the high level classifier, possibly enriched with expressions in the representational language of the low level classifiers. When Cascade Generalization is applied locally, the models generated are more difficult to interpret than those generated by loosely coupled classifiers. The new attributes built at deeper nodes, contain terms based on the previously built attributes. This allows us to built very complex decision surfaces, but it affects somewhat the interpretability of the final model. Using more powerful representations does not necessarily lead to better results. Introducing more flexibility can lead to increased instability (variance) which needs to be controlled. In *local Cascade Generalization* this is achieved by limiting the depth of the applicability of the constructive operator and requiring that the error rate of the classifier used as constructive operator should be less than 0.5. One interesting feature of *local Cascade Generalization* is that it provides a single framework, for a collection of different methods. Our method can be related to several paradigms of machine learning. For example there are similarities with multivariate trees (Brodley & Utgoff, 1995), neural networks (Fahlman, 1990), recursive Bayes (Langley, 1993), and multiple models, namely Stacked Generalization (Wolpert, 1992). In our previous work (Gama & Brazdil, 1999) we have presented system *Ltree* that combines a decision tree with a discriminant function by means of constructive induction. Local Cascade combinations extend this work. In *Ltree* the constructive operator was a single discriminant function. In Local Cascade composition this restriction was relaxed. We can use *any* classifier as constructive operator. Moreover, a composition of several classifiers, like in *CGBLtree*, could be used.

The unified framework is useful because it overcomes some superficial distinctions and enables us to study more fundamental ones. From a practical perspective the user's task is simplified, because his aim of achieving better accuracy can be achieved with a single algorithm instead of several ones. This is done efficiently leading to reduced learning times.

We have shown that this methodology can improve the accuracy of the base classifiers, competing well with other methods for combining classifiers, preserving the ability to provide a single, albeit structured model for the data.

9.1. Limitations and future work

Some open issues, which could be explored in future, involve:

- From the perspective of *bias-variance* analysis the main effect of the proposed methodology is a reduction on the bias component. It should be possible to combine the Cascade architecture with a *variance* reduction method, like Bagging or Boosting.
- Will Cascade Generalization work with other classifiers? Could we use neural networks or nearest neighbors? We think that the methodology presented will work for this type of classifier. We intend to verify it empirically in future.

Other problems that involve basic research include:

- Why does *Cascade Generalization* improve performance? Our experimental study suggests that we should combine algorithms with complementary behavior from the point of view of *bias-variance* analysis. Other forms of complementarity can be considered, for example the *search bias*. So, one interesting issue to be explored is: given a dataset, can we *predict* which algorithms are complementary?
- When does *Cascade Generalization* improve performance? In some datasets *Cascade* was not able to improve the performance of base classifiers. Can we characterize these datasets? That is, can we *predict* under what circumstances *Cascade Generalization* will lead to an improvement in performance?
- How many base classifiers should we use? The general preference is for a smaller number of base classifiers. Under what circumstances can we reduce the number of base classifiers without affecting performance?
- The *Cascade Generalization* architecture provides a method for designing algorithms that use multiple *representations* and multiple *search* strategies within the induction algorithm. An interesting line of future research should explore *flexible* inductive strategies using several diverse representations. It should be possible to extend *Local Cascade Generalization* to provide a dynamic control and this make a step in this direction.

Acknowledgments

Gratitude is expressed to the financial support given by the FEDER and PRAXIS XXI, project ECO, the Plurianual support attributed to LIACC, and Esprit LTR METAL project. Thanks also to Pedro Domingos, all anonymous reviewers, and my colleagues from LIACC for the valuable comments.

Notes

1. The effect of Boosting depends on the learning algorithm used. Here we consider decision trees.
2. Except in the case of Adult and Letter datasets, where a single 10-fold cross-validation was used.
3. We have also evaluated Stacked Generalization using C4.5 at top level. The version that we have used is somewhat better. Using C4.5 at top level the average mean of the error rate is 15.14.

4. This heuristic was suggested by Breiman et al. (1984).
5. Two different methods are presented in Ting and Witten (1997) and Gama (1998).
6. We have preferred C5.0Boosting (instead of Bagging) because it is available for us and allows cross-checking of the results. There are some differences between our results and those previous published by Quinlan. We think that this may be due to the different methods used to estimate the error rate.
7. Except on Monks-2 dataset, where both *Dtree* and *C5.0* produce a tree with only one leaf.
8. The running time of *C5.0* and *C5.0Boosting* were reduced by a factor of 2 as suggested in: *www.spec.org*.
9. The intrinsic noise in the training dataset will be included in the bias term.
10. This analysis assumes that there is a single dominant class for each cluster. Although this may not always be satisfied, it can give insights about the behavior of Cascade composition.

References

Ali, K. M. & Pazzani, M. J. (1996). Error reduction through learning multiple descriptions. *Machine Learning, 24*, 173–202.

Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning, 36*, 105–139.

Blake, C., Keogh, E., & Merz, C. (1999). UCI repository of Machine Learning databases. Department of Information and Computer Science, University of California at Irvine, Irvine, CA.

Breiman, L. (1998). Arcing classifiers. *The Annals of Statistics, 26*(3), 801–849.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Wadsworth International Group.

Brodley, C. E. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning, 20*, 63–94.

Brodley, C. E. & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning, 19*, 45–77.

Buntine, W. (1990). A theory of learning classification rules. Ph.D. Thesis, University of Sydney.

Chan, P. & Stolfo, S. (1995a). A comparative evaluation of voting and meta-learning on partitioned data. In A. Prieditis & S. Russel (Eds.), *Machine Learning, Proc. of 12th International Conference*. Morgan Kaufmann.

Chan, P. & Stolfo, S. (1995b). Learning arbiter and combiner trees from partitioned data for scaling machine learning. In U. M. Fayyad & R. Uthurusamy (Eds.), *Proc. of the First Intern. Conference on Knowledge Discovery and Data Mining*. AAAI Press.

Dillon, W. & Goldstein, M. (1984). *Multivariate analysis, methods and applications*. J. Wiley and Sons, Inc.

Domingos, P. & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning, 29*, 103–129.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In A. Prieditis & S. Russel (Eds.), *Machine Learning, Proc. of 12th International Conference*. Morgan Kaufmann.

Fahlman, S. E. (1991). The recurrent cascade-correlation architecture. In R. P. Lippmann, J. E. Moody, & D. S. Touretzky (Eds.), *Advances in neural information processing systems* (Vol. 3, pp. 190–196). Morgan Kaufmann Publishers, Inc.

Freund, Y. & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In L. Saitta (Ed.), *Machine Learning, Proc. of the 13th International Conference*. Morgan Kaufmann.

Gama, J. (1998). Combining classifiers with constructive induction. In C. Nedellec & C. Rouveirol (Eds.), *Proc. of European Conf. on Machine Learning ECML-98*. LNAI 1398, Springer Verlag.

Gama, J. & Brazdil, P. (1999). Linear tree. *Intelligent Data Analysis, 3*(1), 1–22.

Henery, B. (1997). Combining classification procedures. In R. Nakhaeizadeh, C. Taylor (Ed.), *Machine learning and statistics: The Interface*. John Wiley, Sons, Inc.

Kohavi, R. & Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In L. Saitta (Ed.), *Machine Learning, Proceedings of the 13th International Conference*. Morgan Kaufmann.

Langley, P. (1993). Induction of recursive Bayesian classifiers. In P. Brazdil (Ed.), *Proc. of European Conf. on Machine Learning: ECML-93*. LNAI 667, Springer Verlag.

Langley, P. (1996). *Elements of machine learning*. Morgan Kaufmann.

Michie, D., Spiegelhalter, D., & Taylor, C. (1994). *Machine learning, neural and statistical classification*. Ellis Horwood.

Mitchell, T. (1997). *Machine learning*. MacGraw-Hill Companies, Inc.
Murthy, S., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research, 2*, 1–32.
Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Morgan Kaufmann Publishers, Inc.
Quinlan, R. (1986). Induction of decision trees. *Machine Learning, 1*, 89–106.
Quinlan, R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, Inc.
Quinlan, R. (1996). Bagging, Boosting and C4.5 In *Proc. 13th American Association for Artificial Intelligence*. AAAI Press.
Rivest, R. L. (1987). Learning decision lists. *Machine Learning, 2*, 229–246.
Schaffer, C. (1993). Selecting a classification method by cross-validation. *Machine Learning, 13*, 135–143.
Skalak, D. (1997). *Prototype selection for composite nearest neighbor classifiers*. Ph.D. Thesis, University of Massachusetts Amherst.
Ting, K. & Witten, I. (1997). Stacked generalization: When does it work? In *Proc. International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.
Tumer, K. & Ghosh, J. (1996). Error correlation and error reduction in ensemble classifiers. *Connection Science, Special Issue on Combining Artificial Neural Networks: Ensemble approaches, 8*(34), 385–404.
Wolpert, D. (1992). Stacked generalization. *Neural networks* (Vol. 5, pp. 241–260). Pergamon Press.

Received December 10, 1998
Accepted March 10, 2000
Final manuscript March 10, 2000