Working Paper Series
ISSN 1170-487X

Stacked Generalization: When Does It Work?

by Kai Ming Ting and Ian H. Witten

Working Paper 97/3
January 1997

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

Stacked Generalization: when does it work?

Kai Ming Ting KAIMING@CS.WAIKATO.AC.NZ

Ian H. Witten IHW@CS.WAIKATO.AC.NZ

Department of Computer Science, University of Waikato, Hamilton, New Zealand.

Abstract

Stacked generalization is a general method of using a high-level model to combine lower-level models to achieve greater predictive accuracy. In this paper we resolve two crucial issues which have been considered to be a 'black art' in classification tasks ever since the introduction of stacked generalization in 1992 by Wolpert: the type of generalizer that is suitable to derive the higher-level model, and the kind of attributes that should be used as its input.

We demonstrate the effectiveness of stacked generalization for combining three different types of learning algorithms, and also for combining models of the same type derived from a single learning algorithm in a multiple-data-batches scenario. We also compare the performance of stacked generalization with published results of arcing and bagging.

Keywords: Stacking, cross-validation.

1. Introduction

Stacked generalization is a way of combining multiple models that have been learned for a classification task (Wolpert, 1992). The first step is to collect the output of each model into a new set of data. For each instance in the original training set, this data set represents every model's prediction of that instance's class, along with its true classification. During this step, care is taken to ensure that the models are formed from a batch of training data that does not include the instance in question, in just the same way as ordinary cross-validation. The new data is treated as the data for another learning problem, and in the second step a learning algorithm is employed to solve this problem. In Wolpert's terminology, the original data and the models constructed for it in the first step are referred to as *level-0 data* and *level-0 models*, respectively, while the set of cross-validated data and the second-stage learning algorithm are referred to as *level-1 data* and the *level-1 generalizer*.

In this paper, we show how to make stacked generalization work for classification tasks by addressing two crucial issues which Wolpert (1992) originally described as 'black art' and have not been resolved since. The two issues are (i) the type of attributes that should be used to form level-1 data, and (ii) the type of level-1 generalizer in order to get improved accuracy using the stacked generalization method.

Breiman (1996a) demonstrated the success of stacked generalization in the setting of ordinary regression. The level-0 models are regression trees of different sizes or linear regressions using different number of variables. But instead of selecting the single model that works best as judged by (for example) cross-validation, Breiman used the different level-0 regressors' output values for each member of the training set to form level-1 data. Then he used least-squares linear regression, under the constraint that all regression coefficients be non-negative, as the level-1 generalizer. The non-negativity constraint turned out to be crucial to guarantee that the predictive accuracy would be better than that achieved by selecting the single best predictor.

Here we show how stacked generalization can be made to work reliably in classification tasks. We do this by using the output class probabilities generated by level-0 models to form level-1 data. Then, for the level-1 generalizer we use a version of least squares linear regression adapted for classification tasks. We find the use of class probabilities to be crucial for the successful application of stacked generalization in classification tasks. However, the non-negativity constraints found necessary by Breiman in regression are irrelevant to improved predictive accuracy in our classification situation.

In Section 2, we formally introduce the technique of stacked generalization and describe pertinent details of each learning algorithm used in our experiments. Section 3 describes the results of stacking three different types of learning algorithms. Section 4 shows that stacked generalization can also be used successfully when combining multiple models derived by a single learning algorithm from different, randomly-chosen, batches of training data. Section 5 compares stacked generalization with arcing and bagging. The following section describes related work, and is followed by a summary of our conclusions.

2. Stacked Generalization

Given a data set $\mathcal{L} = \{(y_n, x_n), n = 1, \ldots, N\}$, where y_n is the class value and x_n represents the attribute values of the nth instance, randomly split the data into J almost equal parts $\mathcal{L}_1, \ldots, \mathcal{L}_J$. Define $\mathcal{L}_j$ and $\mathcal{L}^{(-j)} = \mathcal{L} - \mathcal{L}_j$ to be the test and training sets for the jth fold of a J-fold cross-validation. Given K learning algorithms, which we call *level-0 generalizers*, invoke the kth algorithm on the data in the training set $\mathcal{L}^{(-j)}$ to induce a model $\mathcal{M}_k^{(-j)}$, for $k = 1, \ldots, K$. These are called *level-0 models*.

For each instance x in $\mathcal{L}_j$, the test set for the jth cross-validation fold, let $v_k^{(-j)}(x)$ denote the prediction of the model $\mathcal{M}_k^{(-j)}$ on x. Let

$$z_{kn} = v_k^{(-j)}(x_n).$$

At the end of the entire cross-validation process, the data assembled from the outputs of the K models is

$$\mathcal{L}_{CV} = \{(y_n, z_{1n}, \ldots, z_{Kn}), n = 1, \ldots, N\}.$$

This is the *level-1 data*. Use some learning algorithm that we call the *level-1 generalizer* to derive from this data a model $\tilde{\mathcal{M}}$. This is the *level-1 model*. To complete the training process, models $\mathcal{M}_k$, $k = 1, \ldots, K$, are derived using all the data in $\mathcal{L}$.

Now let us consider the classification process, which uses the models $\mathcal{M}_k$, $k = 1, \ldots, K$, in conjunction with $\tilde{\mathcal{M}}$. Given a new instance, models $\mathcal{M}_k$ produce a vector $(z_1, \ldots, z_K)$. This vector is input to the level-1 model $\tilde{\mathcal{M}}$, whose output is the final classification result for that instance. This completes the stacked generalization method as proposed by Wolpert (1992), and also used by Breiman (1996a) and LeBlanc & Tibshirani (1993).

As well as the situation described above, which results in the level-1 model $\tilde{\mathcal{M}}$, the present paper also considers a further situation where the output from the level-0 models is a set of class probabilities rather than a single class prediction. If model $\mathcal{M}_k^{(-j)}$ is used to classify an instance x in $\mathcal{L}_j$, let $P_{ki}^{(-j)}(x)$ denote the probability of the ith output class, and write

$$z_{kin} = P_{ki}^{(-j)}(x_n).$$

As the level-1 data, assemble together the class probability vector from all K models, along with the original class:

$$\mathcal{L}'_{CV} = \{(y_n, z_{11n}, .., z_{1In}, \ldots, z_{k1n}, .., z_{kIn}, \ldots, z_{K1n}, .., z_{KIn}), n = 1, \ldots, N\}$$

(assuming there are I classes). Denote the level-1 model derived from this as $\tilde{\mathcal{M}}'$ to contrast it with $\tilde{\mathcal{M}}$.

The following two subsections describe the algorithms used as level-0 and level-1 generalizers in the experiments reported in Section 3.

2.1 Level-0 Generalizers

Three learning algorithms are used as the level-0 generalizers: C4.5, a decision tree learning algorithm (Quinlan, 1993); NB, a re-implementation of a Naive Bayesian classifier (Cestnik, 1990); and IB1, a variant of a lazy learning algorithm (Aha, Kibler & Albert, 1991) which

employs the p-nearest-neighbor method using a modified value-difference metric for nominal and binary attributes (Cost & Salzberg, 1993). For each of these learning algorithms we now show the formula that we use for the estimated output class probabilities $P_i(x)$ for an instance x (where, in all cases, $\sum_i P_i(x) = 1$).

C4.5: Consider the leaf of the decision tree at which the instance x falls. Let m_i be the number of (training) instances with class i at this leaf, and suppose the majority class at the leaf is $\hat{I}$. Let $E = \sum_{i \neq \hat{I}} m_i$. Then

$$P_{\hat{I}}(x) = 1 - \frac{E+1}{\sum_i m_i + 2},$$

$$P_i(x) = (1 - P_{\hat{I}}(x)) \times \frac{m_i}{E}, \text{ for } i \neq \hat{I}.$$

NB: Let $P(i|x)$ be the posterior probability of class i, given instance x. Then

$$P_i(x) = \frac{P(i|x)}{\sum_i P(i|x)}.$$

IB1: Suppose p nearest neighbors are used; denote them by $\{(y_s, x_s), s = 1, \ldots, p\}$ for instance x. (We use $p = 3$ in the experiments.) Then

$$P_i(x) = \frac{\sum_{s=1}^{p} f(y_s)/d(x, x_s)}{\sum_{s=1}^{p} 1/d(x, x_s)},$$

where $f(y_s) = 1$ if $i = y_s$ and 0 otherwise, and d is the Euclidean distance function.

In all three learning algorithms, the predicted class of the level-0 model, given an instance x, is that $\hat{I}$ for which

$$P_{\hat{I}}(x) > P_i(x) \text{ for all } i \neq \hat{I}.$$

2.2 Level-1 Generalizers

We compare the effect of four different learning algorithms as the level-1 generalizer: C4.5, IB1(using $p = 21$ nearest neighbors),[1] NB, and a multi-response linear regression algorithm, MLR. Only the last needs further explanation.

MLR is an adaptation of a least-squares linear regression algorithm that Breiman (1996a) used in regression settings. Any classification problem with real-valued attributes can be transformed into a multi-response regression problem. If the original classification problem has I classes, it is converted into I separate regression problems, where the problem for class l has instances with responses equal to one when they have class l and zero otherwise.

The input to MLR is level-1 data, and we need to consider the situation for the model $\tilde{\mathcal{M}}'$, where the attributes are probabilities, separately from that for the model $\tilde{\mathcal{M}}$, where

1. A large p value is used following Wolpert's (1992) advice that "...it is reasonable that 'relatively global, smooth ...' level-1 generalizers should perform well."

they are classes. In the former case, where the attributes are already real-valued, the linear regression for class l is simply

$$LR_l(x) = \sum_k^K \sum_i^I \alpha_{kil} P_{ki}(x).$$

In the latter case, the classes are unordered nominal attributes. We map them into binary values in the obvious way, setting $P_{ki}(x)$ to 1 if the class of instance x is l and zero otherwise; and then use the above linear regression.

Choose the linear regression coefficients $\{\alpha_{kil}\}$ to minimize

$$\sum_j \sum_{(y_n,x_n)\in\mathcal{L}_j} (y_n - \sum_k \sum_i \alpha_{kil} P_{ki}^{(-j)}(x_n))^2.$$

The coefficients $\{\alpha_{kil}\}$ are constrained to be non-negative, following Breiman's (1996a) discovery that this is necessary for the successful application of stacked generalization to regression problems. The non-negative-coefficient least-squares algorithm described by Lawson & Hanson (1995) is employed here to derive the linear regression for each class. We show later that, in fact, the non-negative constraint is unnecessary in classification tasks.

With this in place, we can now describe the working of MLR. To classify a new instance x, compute $LR_l(x)$ for all I classes and assign the instance to that class l which has the greatest value:[2]

$$LR_l(x) > LR_{l'}(x) \text{ for all } l' \neq l.$$

In the next section we investigate the stacking of C4.5, NB and IB1.

3. Stacking C4.5, NB and IB1

3.1 When does stacked generalization work?

The experiments in this section show that

- for successful stacked generalization it is necessary to use output class probabilities rather than class predictions—that is, $\tilde{\mathcal{M}}'$ rather than $\tilde{\mathcal{M}}$;
- only the MLR algorithm is suitable for the level-1 generalizer.

We use two artificial datasets and eight real-world datasets from the UCI Repository of machine learning databases (Merz & Murphy, 1996). Details of these are given in Table 1.

For the artificial datasets—Led24 and Waveform—each training dataset $\mathcal{L}$ is generated using a different seed. The algorithms used for the experiments are then tested on a separate dataset of 5000 instances. Results are expressed as the average error rate of ten repetitions of this entire procedure.

For the real-world datasets, W-fold cross-validation is performed. In each fold of this cross-validation, the training dataset is used as $\mathcal{L}$, and the models derived are evaluated on the test dataset. The result is expressed as the average error rate of the W-fold cross-validation. Note that this cross-validation is used for evaluation of the entire procedure,

Table 1: Details of the datasets used in the experiment.

Datasets	# Samples	# Classes	# Attr & Type
Led24	200–5000	10	10N
Waveform	300–5000	3	40C
Horse	368	2	3B+12N+7C
Credit	690	2	4B+5N+6C
Vowel	990	11	10C
Euthyroid	3163	2	18B+7C
Splice	3177	3	60N
Abalone	4177	3	1N+7C
Nettalk(s)	5438	5	7N
Coding	20000	2	15N

N-nominal; B-binary; C: Continuous.

and is quite different from the J-fold cross-validations employed as part of the stacked generalization operation. However, both W and J are set to 10 in the experiments.

Table 2 shows the average error rates, obtained using W-fold cross-validation, of C4.5, NB and IB1, and Best, which is the best of the three, selected using J-fold cross-validation. As expected, Best is almost always the classifier with the lowest error rate.

Table 3 shows the result of stacked generalization using the level-1 model $\tilde{\mathcal{M}}$, for which the level-1 data comprises the classifications generated by the level-0 models, and $\tilde{\mathcal{M}}'$, for which the level-1 data comprises the probabilities generated by the level-0 models. Results are shown for all four level-1 generalizers in each case, along with Best. The lowest error rate for each dataset is given in bold.

Table 4 summarizes the results in Table 3 in terms of a comparison of each level-1 model with Best totaled over all datasets. Clearly, the best level-1 model is $\tilde{\mathcal{M}}'$ derived using MLR. It performs better than Best in nine datasets and equally well in the tenth. The best performing $\tilde{\mathcal{M}}$ is derived from NB, which performs better than Best in seven datasets but significantly worse in two (Waveform and Vowel).

The datasets are shown in the order of increasing size. MLR performs significantly better than Best in the four largest datasets.[3] This indicates that stacked generalization is more likely to give significant improvements in predictive accuracy if the volume of data is large—a direct consequence of more accurate estimation using cross-validation.

MLR has an advantage over the other three level-1 generalizers in that its model can easily be interpreted. Examples of the combination weights it derives (for the probability-based model $\tilde{\mathcal{M}}'$) appear in Table 5 for the Splice and Abalone datasets. The weights indicate the relative importance of the level-0 generalizers for each prediction class. For example, in the Splice dataset (Table 5a), NB is the dominant generalizer for predicting class 2, NB and IB1 are both good at predicting class 3, and all three generalizers make a worthwhile contribution to the prediction of class 1. In contrast, in the Abalone dataset (Table 5b) all three generalizers contribute substantially to the prediction of all three classes.

2. The pattern recognition community calls this type of classifier a *linear machine* (Duda & Hart, 1973).
3. We regard a difference of more than two standard errors as significant (95% confidence).

Table 2: Average error rates of C4.5, NB and IB1, and the best among them selected using J-fold cross-validation.

Datasets	Level-0 Generalizers			
	C4.5	NB	IB1	Best
Led24	35.4	35.4	32.2	32.8
Waveform	31.8	17.1	26.2	17.1
Horse	15.8	17.9	15.8	17.1
Credit	17.4	17.3	28.1	17.4
Vowel	22.7	51.0	2.6	2.6
Euthyroid	1.9	9.8	8.6	1.9
Splice	5.5	4.5	4.7	4.5
Abalone	41.4	42.1	40.5	40.1
Nettalk(s)	17.0	15.9	12.7	12.7
Coding	27.6	28.8	25.0	25.0

Table 3: Average error rates for stacking C4.5, NB and IB1.

Datasets		Level-1 model, $\tilde{\mathcal{M}}$				Level-1 model, $\tilde{\mathcal{M}}'$			
	Best	C4.5	NB	IB1	MLR	C4.5	NB	IB1	MLR
Led24	32.8	34.0	32.4	35.0	33.3	41.7	35.7	**32.1**	**32.1**
Waveform	17.1	17.7	19.2	18.7	17.2	20.6	17.6	17.8	**16.8**
Horse	17.1	16.9	**14.9**	17.6	16.3	18.0	18.5	17.7	15.2
Credit	17.4	18.4	16.1	16.9	17.4	15.4	15.9	**14.3**	16.2
Vowel	2.6	2.6	3.8	3.6	2.6	2.7	7.2	3.3	**2.5**
Euthyroid	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	2.2	4.3	2.0	**1.9**
Splice	4.5	3.9	3.9	**3.8**	**3.8**	4.0	3.9	**3.8**	**3.8**
Abalone	40.1	38.5	38.5	38.2	38.1	43.3	**37.1**	39.2	37.9
Nettalk(s)	12.7	12.4	11.9	12.4	12.6	14.0	14.6	12.0	**11.5**
Coding	25.0	23.2	23.1	23.2	23.2	22.3	21.2	21.2	**20.7**

Table 4: Summary of Table 3—Comparison of Best with $\tilde{\mathcal{M}}$ and $\tilde{\mathcal{M}}'$.

	Level-1 model, $\tilde{\mathcal{M}}$				Level-1 model, $\tilde{\mathcal{M}}'$			
	C4.5	NB	IB1	MLR	C4.5	NB	IB1	MLR
#Win vs. #Loss	3-5	2-7	4-5	2-5	7-3	6-4	4-6	0-9

Table 5: (a) Weights generated by MLR (model $\tilde{\mathcal{M}}'$) for the Splice dataset

	C4.5			NB			IB1		
Class	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}	α_{31}	α_{32}	α_{33}
1	0.23	0.00	0.00	0.43	0.00	0.00	0.36	0.00	0.00
2	0.00	0.15	0.00	0.00	0.72	0.00	0.00	0.12	0.00
3	0.00	0.01	0.08	0.00	0.00	0.52	0.00	0.01	0.40

Table 5: (b) Weights generated by MLR (model $\tilde{\mathcal{M}}'$) for the Abalone dataset

	C4.5			NB			IB1		
Class	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}	α_{31}	α_{32}	α_{33}
1	0.25	0.02	0.00	0.25	0.02	0.00	0.39	0.05	0.00
2	0.11	0.27	0.07	0.00	0.20	0.10	0.00	0.25	0.02
3	0.00	0.05	0.30	0.01	0.03	0.18	0.00	0.07	0.39

3.2 Are non-negativity constraints necessary?

Both Breiman (1996a) and LeBlanc & Tibshirani (1993) use the stacked generalization method in a regression setting and report that it is necessary to constrain the regression coefficients to be non-negative in order to guarantee that stacked regression improves predictive accuracy. Here we investigate this finding in the domain of classification tasks.

To assess the effect of the non-negativity constraint on performance, three versions of MLR are employed to derive the level-1 model $\tilde{\mathcal{M}}'$:

i. each linear regression in MLR is calculated with an intercept constant (that is, $I+1$ weights for the I classes) but without any constraints;

ii. each linear regression is derived with neither an intercept constant (I weights for I classes) nor constraints;

iii. each linear regression is derived without an intercept constant, but with non-negativity constraints (I non-negative weights for I classes).

The third version is the one used for the results presented earlier. Table 6 shows the results of all three versions. They all have almost indistinguishable error rates. We conclude that in classification tasks, non-negativity constraints are not necessary to guarantee that stacked generalization improves predictive accuracy.

However, there is another reason why it is a good idea to employ non-negativity constraints. Table 7 shows an example of the weights derived by these three versions of MLR on the Euthyroid dataset. The third version, shown in row (iii), supports a more perspicuous interpretation of each level-0 generalizer's contribution to the class predictions than do the other two. In this dataset C4.5 is the dominant generalizer, as evidenced by its high weights. However, the negative weights render the interpretation of the other two versions much less clear.

Table 6: Average error rates of three versions of MLR.

Datasets	MLR with		
	No Constraints	No Intercept	Non-Negativity
Led24	34.1	34.1	32.1
Waveform	16.8	16.8	16.8
Horse	15.8	15.8	15.2
Credit	16.2	16.2	16.2
Vowel	2.4	2.4	2.5
Euthyroid	1.9	1.9	1.9
Splice	3.7	3.7	3.8
Abalone	37.9	37.9	37.9
Nettalk(s)	11.5	11.5	11.5
Coding	20.7	20.7	20.7

Table 7: Weights for the Euthyroid dataset with three versions of MLR: (i) no constraints, (ii) no intercept, and (iii) non-negativity constraints.

		Const.	C4.5		NB		IB1	
	Class	α_0	α_{11}	α_{12}	α_{21}	α_{22}	α_{31}	α_{32}
(i)	1	–0.23	0.99	0.06	–0.05	–0.03	0.30	0.20
	2	1.02	–1.03	–0.10	0.17	0.16	–0.17	–0.07
(ii)	1	–	0.99	0.05	0.08	0.09	–0.05	–0.14
	2	–	–1.01	–0.07	–0.41	–0.42	1.40	1.49
(iii)	1	–	0.93	0.00	0.00	0.00	0.09	0.00
	2	–	0.00	0.93	0.01	0.00	0.00	0.07

Table 8: Average error rates of Best, Majority Vote and MLR (model $\tilde{\mathcal{M}}'$), along with the standard error (#SE) between Best and the worst level-0 generalizers.

Dataset	#SE	Best	Majority	MLR
Horse	0.5	17.1	**15.0**	15.2
Splice	2.5	4.5	4.0	**3.8**
Abalone	3.3	40.1	39.0	**37.9**
Led24	8.7	32.8	**31.8**	32.1
Credit	8.9	17.4	**16.1**	16.2
Nettalk(s)	10.8	12.7	12.2	**11.5**
Coding	12.7	25.0	23.1	**20.7**
Waveform	18.7	17.1	19.5	**16.8**
Euthyroid	26.3	**1.9**	8.1	**1.9**
Vowel	242.0	2.6	13.0	**2.5**

3.3 How does stacked generalization compare to majority vote?

Let us now compare the error rate of $\tilde{\mathcal{M}}'$, derived from MLR, to that of majority vote, a simple decision combination method which requires neither cross-validation nor level-1 learning. Table 8 shows the average error rates of Best, majority vote and MLR. In order to see whether the relative performances of level-0 generalizers have any effect on these methods, the number of standard errors (#SE) between the error rates of the worst performing level-0 generalizer and Best is given, and the datasets are re-ordered according to this measure. Since Best almost always selects the best performing level-0 generalizer, small values of #SE indicate that the level-0 generalizers perform comparably to one another, and vice versa.

MLR compares favorably to majority vote, with seven wins versus three losses. Out of the seven wins, six have significant differences (the only exception is for the Splice dataset); whereas all three losses have insignificant differences. Thus the extra computation for cross-validation and level-1 learning seems to have paid off.

It is interesting to note that the performance of majority vote is related to the size of #SE. Majority vote compares favorably to Best in the first seven datasets, where the values of #SE are small. In the last three, where #SE is large, majority vote performs worse. This indicates that if the level-0 generalizers perform comparably, it is not worth using cross-validation to determine the best one, because the result of majority vote—which is far cheaper—is not significantly different. Although small values of #SE are a necessary condition for majority vote to rival Best, they are not a sufficient condition—see Matan (1996) for an example. The same applies when majority vote is compared with MLR. MLR performs significantly better in the five datasets that have large #SE values, but only one in the other cases.

SUMMARY

- None of the four learning algorithms used to obtain model $\tilde{\mathcal{M}}$ perform satisfactorily.
- MLR is the best of the four learning algorithms to use as the level-1 generalizer for obtaining the model $\tilde{\mathcal{M}}'$.
- When obtained using MLR, $\tilde{\mathcal{M}}'$ has lower predictive error rate than the best model selected by J-fold cross-validation, for almost all datasets used in the experiments.
- Another advantage of MLR over the other three level-1 generalizers is its interpretability. The weights α_{ki} indicates the different contributions that each level-0 model makes to the prediction classes.
- Model $\tilde{\mathcal{M}}'$ can be derived by MLR with or without non-negativity constraints. Such constraints make little difference to the model's predictive accuracy.
- The use of non-negativity constraints in MLR has the advantage of interpretability. Non-negative weights α_{ki} support easier interpretation of the extent to which each model contributes to each prediction class.
- When derived using MLR, model $\tilde{\mathcal{M}}'$ compares favorably with majority vote.

4. Stacking in a multiple-data-batches scenario

Ting & Low (1996) consider how to combine models derived by a single learning algorithm from different batches of a data set, and show how this "model combination approach" can outperform a single model induced by the same learning algorithm from all of the original data together—the "data combination approach."

They explain this by relating the performance of both approaches to the position of the datasets on the learning curve of the learning algorithm used. At the beginning of the learning curve where the training set is relatively small, data combination is expected to yield a large gain in performance because the learning curve is steep. However, near the asymptotic region of the curve, additional data yields only a marginal improvement. For example, near-asymptotic behavior occurs when doubling the training set yields little performance gain. The effect of doubling the training data size in two different regions of a learning curve is illustrated in Figure 1: doubling the data X yields a large improvement, whereas doubling Y makes nearly no difference to performance. The reverse is true for model combination, which requires some estimate to be made of the local performance of individual models. When little data is available, the estimated measure can be inaccurate; thus combining different models yields only a marginal performance gain. Large volumes of data enable more accurate estimation and therefore a greater performance gain. Experimental results bear this out (Ting & Low, 1996).

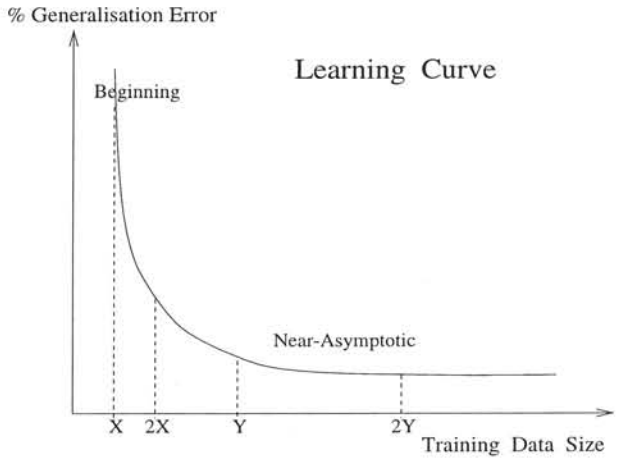

Figure 1: Performance gain yielded by doubling the training data in different regions of a learning curve.

Previous work used just one method for combining the individual models (Ting, 1996). Here we investigate the performance of stacked generalization in the same scenario. We consider the base-line behavior in which there are only two data batches. The next two subsections describe the setup necessary for the use of stacked generalization in this situation, and report experimental results.

4.1 Stacked generalization with multiple data batches

Divide the original data set $\mathcal{L}$ into two batches $\mathcal{L}A = \{(y_a, x_a), a = 1, \ldots, N\}$ and $\mathcal{L}B = \{(y_b, x_b), b = 1, \ldots, M\}$. Randomly split the first batch into J almost equal parts $\mathcal{L}A_1, \ldots, \mathcal{L}A_J$, and define $\mathcal{L}A_j$ and $\mathcal{L}A^{(-j)} = \mathcal{L}A - \mathcal{L}A_j$ to be the test and training sets respectively for the jth fold of cross-validation. Using the data in the training set, induce a model $\mathcal{M}_A^{(-j)}$ using some learning algorithm. Use the same learning algorithm on all of the data in $\mathcal{L}B$ to derive a single model, which we call $\mathcal{M}_B$.

Given a data instance x_a in $\mathcal{L}A_j$, denote by z_{Aia} the probability that $\mathcal{M}_A^{(-j)}$ assigns to the ith class. For the same instance, denote by z_{Bia} the probability that the model $\mathcal{M}_B$ assigns to the ith class.

Using this procedure, form one set of *level-1* data as

$$\mathcal{L}A'_{CV} = \{(y_a, z_{A1a}, \ldots, z_{AIa}, z_{B1a}, \ldots, z_{BIa}), a = 1, \ldots, N\},$$

where I is the number of classes. A second set of level-1 data can be obtained by randomly splitting $\mathcal{L}B$ into J almost equal parts and deriving a single $\mathcal{M}_A$ and a cross-validation set of models $\mathcal{M}_B^{(-j)}$ for use on the test sets $\mathcal{L}B_j$:

$$\mathcal{L}B'_{CV} = \{(y_b, z_{A1b}, \ldots, z_{AIb}, z_{B1b}, \ldots, z_{BIb}), b = 1, \ldots, M\}.$$

Now concatenate the two data sets $\mathcal{L}A'_{CV}$ and $\mathcal{L}B'_{CV}$ to form the joint level-1 data in this two-data-batches scenario:

$$\mathcal{L}'_{CV} = \{(y_c, z_{A1c}, \ldots, z_{AIc}, z_{B1c}, \ldots, z_{BIc}), c = 1, \ldots, N + M\}.$$

Use the MLR learning strategy on this training data to form a level-1 model $\tilde{\mathcal{M}}'$. The final classification process uses models $\mathcal{M}_A$, $\mathcal{M}_B$ and $\tilde{\mathcal{M}}'$ in just the same way as described in Section 2.

In the following experiments, we set $N = M$ and $J = 10$. Models $\mathcal{M}_A$ and $\mathcal{M}_B$ are derived using the same learning algorithm. Results are given for each of the algorithms C4.5, NB and IB1.

4.2 Does stacked generalization work in the multiple-data-batches scenario?

We use the two artificial datasets Led24 and Waveform for this experiment and proceed as follows. For each dataset, batches $\mathcal{L}A$ and $\mathcal{L}B$ are generated using different seeds. The performance of stacked generalization is assessed on a separate dataset of size 5000. The result is given as the average error rate of 10 repeated trials. We are interested in comparing the performance of $\tilde{\mathcal{M}}'$ with that of a single model derived using the joint dataset formed from $\mathcal{L}A$ and $\mathcal{L}B$, which we call $\mathcal{M}_{A+B}$. A learning curve is produced for each model.

Figures 2, 3 and 4 show the learning curves for stacked generalization using C4.5, NB, and IB1. Each figure has two parts that show the results for the Led24 and Waveform datasets. Each graph comprises three learning curves, the first showing the behavior of the model $\mathcal{M}_A$ derived from the single batch $\mathcal{L}A$, the second showing the model $\mathcal{M}_{A+B}$ derived from the two batches $\mathcal{L}A$ and $\mathcal{L}B$ together, and the third showing the level-1 model $\tilde{\mathcal{M}}'$ derived by using MLR for stacked generalization. In this section, the terms, $\tilde{\mathcal{M}}'$ and MLR are used interchangeably.

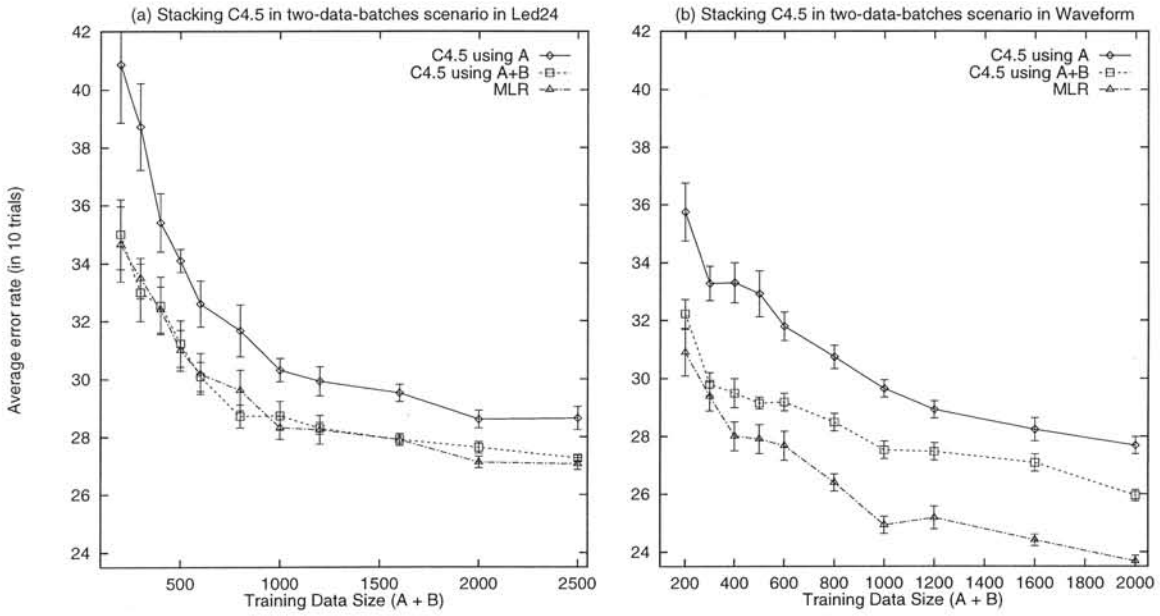

Figure 2: Learning curves for models derived by C4.5 from two data batches. Note that each point in the solid-line graph employs just half of the training data indicated.

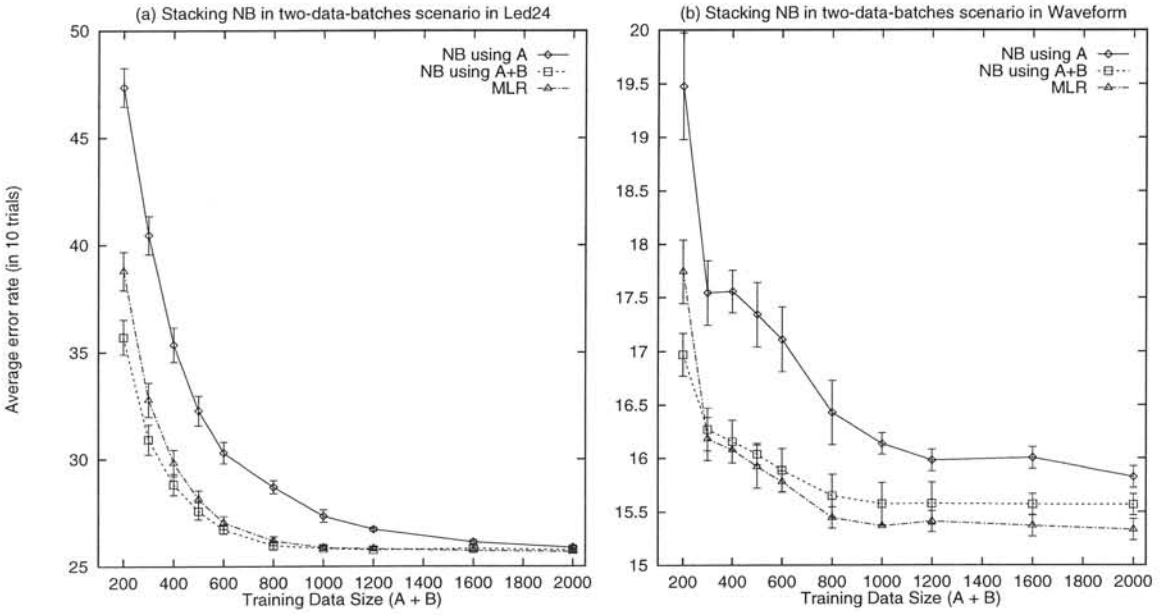

Figure 3: Learning curves for models derived by NB from two data batches.

When stacking models which are derived from C4.5, the stacked model MLR performs comparably with the single model $\mathcal{M}_{A+B}$ until $|\mathcal{L}A| + |\mathcal{L}B| = 2000$ in the Led24 dataset. From this point onwards, it begins to outperform the single model. In the Waveform dataset, MLR performs better, though not significantly better, than $\mathcal{M}_{A+B}$ even at the beginning of the curve, and becomes significantly better as the asymptote is approached.

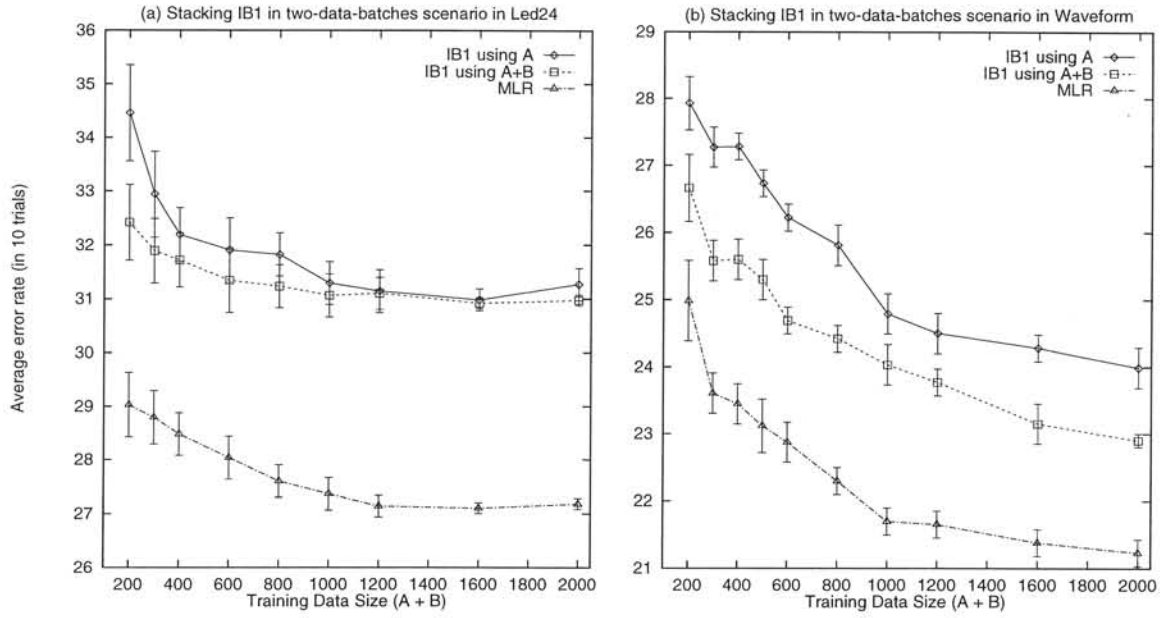

Figure 4: Learning curves for models derived by IB1 from two data batches.

Table 9: An example of combination weights for MLR when stacking C4.5 for two batches of the Waveform dataset.

	$\mathcal{M}_A$			$\mathcal{M}_B$		
Class	α_{11}	α_{12}	α_{13}	α_{21}	α_{22}	α_{23}
1	0.41	0.03	0.00	0.51	0.03	0.07
2	0.03	0.44	0.02	0.01	0.50	0.00
3	0.04	0.00	0.44	0.01	0.00	0.46

When stacking models derived using NB, shown in Figure 3, the two learning curves, for MLR and $\mathcal{M}_{A+B}$, cross over—this is particularly apparent for the Waveform dataset. At the beginning of the curve, $\mathcal{M}_{A+B}$ is significantly better than MLR, and this is reversed in the near-asymptotic region. Note that the Bayes error rate for the Led24 and Waveform datasets are 26% and 14% respectively (Breiman *et al.*, 1984), and for NB, both $\mathcal{M}_{A+B}$ and MLR approach these figures. When stacking models derived using IB1, shown in Figure 4, MLR is always significantly better than $\mathcal{M}_{A+B}$ throughout the learning curves for both the Led24 and Waveform datasets. Not surprisingly, in all cases MLR always performs better than $\mathcal{M}_A$ (and $\mathcal{M}_B$).

An example of the combination weights generated by MLR when stacking models derived using C4.5 on the Waveform dataset is shown in Table 9. It indicates that both models $\mathcal{M}_A$ and $\mathcal{M}_B$ have about the same ability to predict all classes. Both batches $\mathcal{L}A$ and $\mathcal{L}B$ contain 300 instances.

Table 10: Comparing stacking with arcing and bagging classifiers.

Dataset	#Samples	stacking	arcing	bagging
Waveform	300	16.8 ±0.2	17.8	19.3
Glass	214	28.4 ±2.9	22.0	23.2
Ionosphere	351	9.7 ±1.5	6.4	7.9
Soybean	683	4.3 ±1.1	5.8	6.8
Breast Cancer	699	2.7 ±0.8	3.2	3.7
Diabetes	768	24.2 ±1.2	26.6	23.9

We conclude that stacking models derived from a single learning algorithm on two randomly-chosen batches of data yields performance that is always better than, or comparable to, that given by combining the batches into one in the near-asymptotic region of the learning curve. The reader is referred to Ting & Low (1996) for a fuller investigation of the multiple-data-batches scenario.

5. Comparison with arcing and bagging

This section compares the results of stacking C4.5, NB and IB1 with the results of arcing (called boosting by its originator, Schapire, 1990) and bagging that are reported by Breiman (1996b; 1996c). Both arcing and bagging employ sampling techniques to modify the data distribution in order to produce multiple models from a single learning algorithm. To combine the decisions of the individual models, arcing uses a weighted majority vote and bagging uses an unweighted majority vote. Breiman reports that both arcing and bagging can substantially improve the predictive accuracy of a single model derived using their base learning algorithm.

First we describe the differences between the experimental procedures. Our results for stacking are averaged over ten-fold cross-validation for all datasets except Waveform, which is averaged over ten repeated trials. Standard errors are also shown. Results for arcing and bagging are those obtained by Breiman (1996b; 1996c), which are averaged over 100 trials. For all datasets except Waveform, each trial uses a random 9:1 split to form the training and test sets. Also note that the Waveform dataset we used has 19 irrelevant attributes, but Breiman used a version without irrelevant attributes. In both cases 300 training instances were used for this dataset, but we used 5000 test instances whereas Breiman used 1800. Arcing and bagging are done with 50 decision tree models derived from CART (Breiman *et al.*, 1984) in each trial.

The results on six datasets are given in Table 10, and indicate that the three methods are very competitive.[4] Note that stacking performs very poorly on Glass and Ionosphere, two small real-world datasets. This is not surprising, because cross-validation inevitably produces poor estimates for small datasets.

It is worth noting that arcing and bagging can be incorporated into the framework of stacked generalization by using them as level-0 models.

4. The heart dataset used by Breiman (1996b; 1996c) is omitted because it was very much modified from the original one.

Like bagging, stacked generalization is ideal for parallel computation. The construction of each level-0 model proceeds independently, no communication with the other modeling processes being necessary.

Arcing 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. Using a level-1 generalizer, stacking can work with only two or three level-0 models. Although we have not tested the use of more level-0 models, there is no reason to believe that stacking will fail in this situation.

Suppose the computation time required for a learning algorithm is C, and arcing or bagging needs h models. The learning time required is $T_a = hC$. Suppose stacking requires g models and each model employs J-fold cross-validation. Assuming that time C is needed to derive each of the g level-0 models and the level-1 model, the learning time for stacking is $T_s = (g(J+1)+1)C$. For the results given in Table 10, $h = 50$, $J = 10$, and $g = 3$; thus $T_a = 50C$ and $T_s = 34C$. However, in practice the learning time required for the level-0 and level-1 generalizers is different.

Users of stacking have a free choice of level-0 models. They may either be derived from a single learning algorithm, or from a variety of different algorithms. The example in Section 3 uses different types of learning algorithms, while the one in Section 4 uses data variation to obtain a diverse set of models. In the former case, performance may vary substantially between the level-0 models—for example NB performs very poorly in the Vowel and Euthyroid datasets compared to the other two models (see Table 2). Stacking copes well with this situation. The performance variation among the member models in bagging is rather small because they are derived from the same learning algorithm using bootstrap samples. Section 3.3 shows that a small performance variation among member models is a necessary condition for majority vote (as employed by bagging) to work well.

Breiman (1996b; 1996c) reveals that arcing and bagging can only improve the predictive accuracy of learning algorithms that are 'unstable.' An unstable learning algorithm is one for which small perturbations in the training set can produce large changes in the derived model. Decision trees and neural networks are unstable; NB and IB1 are stable. Stacking works with both.

6. Related work

Our analysis of stacked generalization was motivated by that of Breiman (1996a), discussed earlier, and LeBlanc & Tibshirani (1993). LeBlanc & Tibshirani (1993) examine the stacking of a linear discriminant and a nearest neighbor classifier and show that, for one artificial dataset, a method such as MLR performs better with non-negativity constraints than without. Our results show that these constraints are irrelevant to MLR's predictive accuracy in the classification situation.

In the earlier work on model combination in the multiple-data-batches scenario, Ting & Low (1996) report that "...though our investigation is limited to one type of combination method, we believe that the results ... are applicable to other reasonable combination methods." The present paper confirms their conjecture by showing that stacked generalization also works in this scenario.

The limitations of MLR are well-known (Duda & Hart, 1973). For a I-class problem, it divides the description space into I convex decision regions. Every region must be singly connected, and the decision boundaries are linear hyperplanes. This means that MLR is most suitable for problems with unimodal probability densities. Despite these limitations, MLR still performs better as a level-1 generalizer than IB1, its nearest competitor in deriving $\tilde{\mathcal{M}}'$. These limitations may hold the key for a fuller understanding of the behavior of stacked generalization. Jacobs (1995) reviews linear combination methods like that used in MLR.

Previous work on stacked generalization, especially as applied to classification tasks, has been limited in several ways. Some only applies to a particular dataset (e.g., Zhang, Mesirav & Waltz, 1992). Others report results that are less than convincing (Merz, 1995). Still others have a different focus and evaluate the results on just a few datasets (LeBlanc & Tibshirani, 1993; Chan & Stolfo, 1995; Kim & Bartlett, 1995; Fan *et al.*, 1996).

One might consider a degenerate form of stacked generalization that does not use cross-validation to produce data for level-1 learning. Then, level-1 learning can be done 'on the fly' during the training process (Jacobs *et al.*, 1991). In another approach, level-1 learning takes place in batch mode, after all level-0 models are derived (Ho *et al.*, 1994). In regression settings, Meir (1994) gives a mathematical account of the effect of combining several least-squares linear estimators linearly, based on their expected performance, under the multiple-data-batches scenario. Sollich and Krogh (1996) conduct a similar study under slightly different conditions. In both cases, level-1 learning is regarded a batch process.

Several researchers have worked on a still more degenerate form of stacked generalization without any cross-validation or learning at level 1. Examples are neural network ensembles (Hansen & Salamon, 1990; Perrone & Cooper, 1993; Krogh & Vedelsby, 1995), multiple decision tree combination (Kwok & Carter, 1990; Buntine, 1991; Oliver & Hand, 1995), and multiple rule combination (Kononenko & Kovačič, 1992). The methods used at level 1 are majority voting, weighted averaging and Bayesian combination. Other possible methods are distribution summation and likelihood combination. There are various forms of re-ordering class rank, and Ali and Pazzani (1996) study some of these methods for a rule learner.

7. Conclusions

We have resolved two crucial issues for the successful implementation of stacked generalization in classification tasks. First, class probabilities should be used instead of the single predicted class as input attributes for higher-level learning. Second, the multi-response least squares linear regression technique should be employed as the high-level generalizer.

When combining three different types of learning algorithms, this implementation of stacked generalization was found to achieve better predictive accuracy than both model selection based on cross-validation and majority vote; it was also found to be competitive with arcing and bagging. Unlike stacked regression, non-negativity constraints in the least-squares regression are not necessary to guarantee improved predictive accuracy in classification tasks. However, these constraints are still preferred because they increase the interpretability of the level-1 model. Stacked generalization also works in the multiple-data-batches scenario, combining two models derived from the same learning algorithm.

This paper concentrates on finding conditions under which stacked generalization works. A better understanding of why it works in this particular configuration may open up other possibilities for further improvement of stacked generalization.

Acknowledgment

The authors are grateful to the New Zealand Marsden Fund for financial support for this research.

References

Aha, D.W., D. Kibler & M.K. Albert (1991), Instance-Based Learning Algorithms, *Machine Learning, 6*, pp. 37-66.

Ali, K.M. & M.J. Pazzani (1996), Error Reduction through Learning Multiple Descriptions, *Machine Learning*, Vol. 24, No. 3, pp. 173-206.

Breiman, L. (1996a), Stacked Regressions, *Machine Learning*, Vol. 24, pp. 49-64.

Breiman, L. (1996b), Bagging Predictors, *Machine Learning*, Vol. 24, No. 2, pp. 123-140.

Breiman, L. (1996c), Bias, Variance, and Arcing Classifiers, *Technical Report 460*, Department of Statistics, University of California, Berkeley, CA.

Breiman, L., J.H. Friedman, R.A. Olshen & C.J. Stone (1984), *Classification And Regression Trees*, Belmont, CA: Wadsworth.

Cestnik, B. (1990), Estimating Probabilities: A Crucial Task in Machine Learning, in *Proceedings of the European Conference on Artificial Intelligence*, pp. 147-149.

Chan, P.K. & S.J. Stolfo (1995), A Comparative Evaluation of Voting and Meta-learning on Partitioned Data, in *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 90-98, Morgan Kaufmann.

Cost, S & S. Salzberg (1993), A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features, *Machine Learning, 10*, pp. 57-78.

Fan, D.W., P.K. Chan, S.J. Stolfo (1996), A Comparative Evaluation of Combiner and Stacked Generalization, in *Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models*, pp. 40-46.

Hansen, L.K. & P. Salamon (1990), Neural Network Ensembles, in *IEEE Transactions of Pattern Analysis and Machine Intelligence, 12*, pp. 993-1001.

Ho, T.K., J.J. Hull & S.N. Srihari (1994), Decision Combination in Multiple Classifier Systems, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 1, pp. 66-75.

Jacobs, R.A. (1995), Methods of Combining Experts' Probability Assessments, *Neural Computation 7*, pp. 867-888, MIT Press.

Jacobs, R.A., M.I. Jordan, S.J. Nowlan & G.E. Hinton (1991), Adaptive Mixtures of Local Experts, in *Neural Computation 3*, pp. 79-87.

Kim, K. & E.B. Bartlett (1995), Error Estimation by Series Association for Neural Network Systems, *Neural Computation 7*, pp. 799-808, MIT Press.

Kononenko, I. & M. Kovačič (1992), Learning as Optimization: Stochastic Generation of Multiple Knowledge, in *Proceedings of the Ninth International Conference on Machine Learning*, pp. 257-262, Morgan Kaufmann.

Krogh, A. & J. Vedelsby (1995), Neural Network Ensembles, Cross Validation, and Active Learning, in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D.S. Touretsky & T.K. Leen (Editors), pp. 231-238, MIT Press.

Kwok, S. & C. Carter (1990), Multiple Decision Trees, *Uncertainty in Artificial Intelligence 4*, R. Shachter, T. Levitt, L. Kanal and J. Lemmer (Editors), pp. 327-335, North-Holland.

Lawson C.L. & R.J. Hanson (1995), *Solving Least Squares Problems*, SIAM Publications.

LeBlanc, M & R. Tibshirani (1993), Combining Estimates in Regression and Classification, Technical Report 9318, Department of Statistics, University of Toronto.

Matan, O. (1996), On Voting Ensembles of Classifiers (extended abstract), in *Proceedings of AAAI-96 workshop on Integrating Multiple Learned Models*, pp. 84-88.

Meir, R. (1994), Bias, Variance and the Combination of Estimators: The Case of Linear Least Squares, Technical Report No. 922, Department of Electrical Engineering, Technion, Haifa, Israel.

Merz, C.J. (1995), Dynamic Learning Bias Selection, in *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL: Unpublished, pp. 386-395.

Merz, C.J. & Murphy, P.M. (1996), *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.

Oliver, J.J. & D.J. Hand (1995), On Pruning and Averaging Decision Trees, in *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 430-437. Morgan Kaufmann.

Perrone, M.P. & L.N. Cooper (1993), When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, in *Artificial Neural Networks for Speech and Vision*, R.J. Mammone (Editor), Chapman-Hall.

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

Schapire, R.E. (1990), The Strength of Weak Learnability, *Machine Learning, 5*, pp. 197-227, Kluwer Academic Publishers.

Sollich, P. & A. Krogh (1996), Learning with ensembles: How overfitting can be useful, in *Advances in Neural Information Processing Systems 8*, D.S. Touretzky, M.C. Mozer & M.E. Hasselmo (Editors) , MIT Press.

Ting, K.M. (1996), The Characterisation of Predictive Accuracy and Decision Combination, in *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 498-506, Morgan Kaufmann.

Ting, K.M. & B.T. Low (1996), Theory Combination: an alternative to Data Combination, *Working Paper 96/19*, Department of Computer Science, University of Waikato. [*http : //www.cs.waikato.ac.nz/cs/Staff/kaiming.html*].

Wolpert, D.H. (1992), Stacked Generalization, *Neural Networks*, Vol. 5, pp. 241-259, Pergamon Press.

Zhang, X., J.P. Mesirov and D.L. Waltz (1992), Hybrid System for Protein Secondary Structure Prediction, *Journal of Molecular Biology*, 225, pp. 1049-1063.