How to Make Stacking Better and Faster While Also Taking Care of an Unknown Weakness

Alexander K. Seewald ALEX@SEEWALD.AT; ALEXSEE@AI.UNIVIE.AC.AT
Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Wien, Austria

Abstract

We investigated performance differences between multi-class and two-class datasets for ensemble learning schemes. We were surprised to find that Stacking, the best-known such scheme, performs worse on multi-class datasets when using class probability distributions as meta-level data. In this paper we will present results concerning this heretofore unknown weakness of Stacking. In addition we will present a new variant of Stacking, using meta-learner MLR, which is able to compensate for this weakness, improving Stacking significantly on almost half of our multi-class datasets. Two other related meta-learners could also be improved using the same idea. The dimensionality of the meta-data set is reduced by a factor equal to the number of classes, which leads to faster learning. In comparison to other ensemble learning methods this improves Stacking's lead further, making it the most successful system by a variety of measures.

1. Introduction

When faced with the decision "Which algorithm will be most accurate on my classification problem?", the predominant approach is to estimate the accuracy of the candidate algorithms on the problem and select the one that appears to be most accurate. Schaffer (1993) has investigated this approach in a small study with three learning algorithms on five UCI datasets. His conclusions are that on the one hand this procedure is on average better than working with a single learning algorithm, but, on the other hand, the cross-validation procedure often picks the wrong base algorithm on individual problems. This problem is expected to become more severe with an increasing number of classifiers.

As a cross-validation basically computes a prediction for each example in the training set, it was soon realized that this information could be used in more elaborate ways than simply counting the number of correct and incorrect predictions. One such ensemble learning method or meta-classification scheme is the family of *stacking* algorithms (Wolpert, 1992). The basic idea of Stacking is to use the predictions of the original classifiers as attributes in a new training set that keeps the original class labels.

A straightforward extension of this approach is using class probability distributions of the original classifiers[1] which convey not only prediction information, but also confidence for all classes. This approach was evaluated and found to be superior to Stacking with predictions in (Ting & Witten, 1999), provided *multi-response linear regression* (MLR) is used as meta classifier. Our variant StackingC is based on this extension.

We have investigated the performance of four common meta-classification schemes, including the mentioned Stacking variant, relative to multi-class vs. two-class datasets, in a ranking. We found no significant differences for Grading and Voting[2] – however, the mentioned Stacking variant showed a significant performance degradation for multi-class datasets. This performance degradation is also quite apparent by other accuracy-based measures. We will present these results and a variant of stacking which does not show this performance degradation.

First, we will present the basic concept behind Stacking and show how it can be extended towards StackingC for the meta-learner MLR. Then we will describe the experimental setup, base classifiers and meta-classification schemes used and investigate empirically the claims that Stacking performs worse on multi-class datasets. Afterwards we will compare StackingC to Stacking in more detail, considering significant differ-

[1] Every prediction is replaced by a vector of probabilities, one for each class.

[2] X-Valseems to be worse on two-class datasets.

$Attrs$	$Cl.$
$AttrVec_1$	a
$AttrVec_2$	b
$AttrVec_3$	b
$AttrVec_4$	c
⋮	⋮
$AttrVec_n$	a

(a) original training set

$Classifier_i$		
a	b	c
$P_{i,a1} = 0.90$	$P_{i,b1} = 0.05$	$P_{i,c1} = 0.05$
$P_{i,a2} = 0.15$	$P_{i,b2} = 0.70$	$P_{i,c2} = 0.15$
$P_{i,a3} = 0.10$	$P_{i,b3} = 0.80$	$P_{i,c3} = 0.10$
$P_{i,a4} = 0.20$	$P_{i,b4} = 0.20$	$P_{i,c4} = 0.60$
⋮	⋮	⋮
$P_{i,an} = 0.80$	$P_{i,bn} = 0.10$	$P_{i,cn} = 0.10$

(b) sample class probability distribution

$Classifier_1$		$Classifier_2$			$Classifier_N$			$class$
a	b	c	a	b	c		a	b	c	$= a?$
$P_{1,a1}$	$P_{1,b1}$	$P_{1,c1}$	$P_{2,a1}$	$P_{2,b1}$	$P_{2,c1}$	...	$P_{N,a1}$	$P_{N,b1}$	$P_{N,c1}$	1
$P_{1,a2}$	$P_{1,b2}$	$P_{1,c2}$	$P_{2,a2}$	$P_{2,b2}$	$P_{2,c2}$	...	$P_{N,a2}$	$P_{N,b2}$	$P_{N,c2}$	0
$P_{1,a3}$	$P_{1,b3}$	$P_{1,c3}$	$P_{2,a3}$	$P_{2,b3}$	$P_{2,c3}$	...	$P_{N,a3}$	$P_{N,b3}$	$P_{N,c3}$	0
$P_{1,a4}$	$P_{1,b4}$	$P_{1,c4}$	$P_{2,a4}$	$P_{2,b4}$	$P_{2,c4}$	...	$P_{N,a4}$	$P_{N,b4}$	$P_{N,c4}$	0
⋮	⋮		⋮				⋮			⋮
$P_{1,an}$	$P_{1,bn}$	$P_{1,cn}$	$P_{2,an}$	$P_{2,bn}$	$P_{2,cn}$	...	$P_{N,an}$	$P_{N,bn}$	$P_{N,cn}$	1

(c) meta training set for class a, Stacking with MLR

$Classifier_1$	$Classifier_2$		$Classifier_N$	$class$
a	a		a	$= a?$
$P_{1,a1}$	$P_{2,a1}$	...	$P_{N,a1}$	1
$P_{1,a2}$	$P_{2,a2}$	...	$P_{N,a2}$	0
$P_{1,a3}$	$P_{2,a3}$	...	$P_{N,a3}$	0
$P_{1,a4}$	$P_{2,a4}$	...	$P_{N,a4}$	0
⋮	⋮		⋮	⋮
$P_{1,an}$	$P_{2,an}$	...	$P_{N,an}$	1

(d) meta training set for class a, StackingC with MLR

Figure 1. Illustration of Stacking and StackingC on a dataset with three classes (a, b and c), n examples and N base classifiers. $P_{i,jk}$ refers to the probability given by base classifier i for class j on example number k

ences by dataset, influence of *number of classes* and shortly investigate the claim of faster learning for StackingC. At last we will discuss results about applying the same basic idea to other meta-classifiers and point towards interesting research directions for the future.

2. StackingC

The main idea behind stacking is using the output from a set of level-0 (=base) classifiers, estimated via cross-validation, to learn a level-1 (=meta) classifier which gives the final prediction.

As Ting & Witten (1999) propose, we use Stacking with *multi-response linear regression* (MLR) as level-1 classifier. Basically, MLR learns a linear regression function for each class which predicts degree of confidence in class membership and can, after normalization, be interpreted as class probability. Other level-1 classifiers do not usually learn a distinct model for each class, but instead learn a single model for all classes at once, e.g. a decision tree.

During prediction, the base classifiers are queried for their class probability distributions which are then used as input for the regression models (one for each class). The output of the linear models is renormalized to yield a proper class probability distribution.

Figure 1 shows an example with three classes (a, b and c), n examples and N base classifiers. 1(a) shows the original training set with its attribute vectors and class values.

Figure 1(b) shows how a class probability distribution of one sensible classifier may look like. The maximum probabilities are shown in *italics* and denote the classes which would be predicted for each example. There is one such set of class probability distributions for each base classifier.

Figure 1(c) shows the meta training set for Stacking which is used to learn a linear regression function to predict the probability of $class == a$. We denote $P_{i,jk}$ to signify the probability given by base classifier i for class j on example number k. The classes are mapped to an indicator variable such that only class a is mapped to 1 and all other classes to 0. In our example there are of course two other such training sets for class b and c which differ only in the last column and are thus not shown.

The proposed variant, StackingC, differs in these points: for each linear model associated with a specific class, only the partial class probability distribution which deals with this very class is used during training and testing. While stacking uses probabilities for all classes and from all component classifiers for each linear model, StackingC uses only the class probabilities associated with the class which we want our linear model to predict.[3]

[3] We also switched off the internal feature subset selection in MLR since that seemed to slightly improve perfor-

Figure 1(d) shows the corresponding meta training set for StackingC which consists only of those columns from the original meta training set which are concerned with class=a; i.e. $P_{i,ak}$ for all i and k. While the meta training sets for Stacking's meta-classifier differ only in the last attribute (the class indicator variable), those for StackingC have less attributes by a factor equal to the number of classes and also have no common attributes. Out of necessity, this leads to more diverse linear models which we believe to be one mechanism why it outperforms Stacking. Another one may simply be that with less attributes, the learning problem becomes easier to solve, as long as only irrelevant information is removed.

As can be easily seen, this modification should not change the performance for two-class datasets significantly. Since the sum of each class probability distribution has to be one, the probability of one class is one minus the probability of the other class, so one of these values is sufficient to encode the complete information[4] Thus we would expect two-class datasets to offer equally good performance under this modification, but train slightly faster[5] because of the inherent dimensionality reduction for meta-data.

3. Experimental Setup

We implemented StackingC in Java within the Waikato Environment for Knowledge Analysis (WEKA[6]). All other algorithms at the base and meta-level were already available within WEKA.

For an empirical evaluation we chose twenty-six datasets from the UCI Machine Learning Repository (Blake & Merz, 1998), shown in Table 1. These datasets include fourteen multi-class and twelve two-class problems. Reported accuracy estimates are the average of ten ten-fold stratified cross validations unless otherwise noted. Significant differences were evaluated by a t-test with significance level of 99%.[7]

mance – possibly because all features from the focussed meta-level data are already relevant.

[4] A linear model is free to use either the one attribute with a positive weight or the other with a negative weight, using appropriate constant terms.

[5] The training costs for the base-classifiers are of course unchanged.

[6] The Java source code of WEKA has been made available at `www.cs.waikato.ac.nz`

[7] The used t-test has been shown to have a high type I error e.g. in (Dietterich, 1998). Although we obtained similar results using a single ten-fold cross-validation and χ^2 test after McNemar – which should have a low type I error according to the same paper – our reported significant differences may still be too optimistic.

Table 1. The used datasets with number of classes and examples, discrete and continuous attributes, baseline accuracy (%) and entropy in bits per example (Kononenko & Bratko, 1991).

Dataset	cl	Inst	disc	cont	bL	E
audiology	24	226	69	0	25.22	3.51
autos	7	205	10	16	32.68	2.29
balance-scale	3	625	0	4	45.76	1.32
breast-cancer	2	286	10	0	70.28	0.88
breast-w	2	699	0	9	65.52	0.93
colic	2	368	16	7	63.04	0.95
credit-a	2	690	9	6	55.51	0.99
credit-g	2	1000	13	7	70.00	0.88
diabetes	2	768	0	8	65.10	0.93
glass	7	214	0	9	35.51	2.19
heart-c	5	303	7	6	54.46	1.01
heart-h	5	294	7	6	63.95	0.96
heart-statlog	2	270	0	13	55.56	0.99
hepatitis	2	155	13	6	79.35	0.74
ionosphere	2	351	0	34	64.10	0.94
iris	3	150	0	4	33.33	1.58
labor	2	57	8	8	64.91	0.94
lymph	4	148	15	3	54.73	1.24
primary-t.	22	339	17	0	24.78	3.68
segment	7	2310	0	19	14.29	2.81
sonar	2	208	0	60	53.37	1.00
soybean	19	683	35	0	13.47	3.84
vehicle	4	846	0	18	25.41	2.00
vote	2	435	16	0	61.38	0.96
vowel	11	990	3	10	9.09	3.46
zoo	7	101	16	2	40.59	2.41

We chose four meta-classification schemes, including Stacking.

- Grading is the implementation of the grading algorithm evaluated in (Seewald & Fürnkranz, 2001). It uses the instance-based classifier IBk with ten nearest neighbors as meta-level classifier.
- X-Val chooses the best base classifier on each fold by an internal ten-fold CV on the training data. This is just Selection by Crossvalidation which we mentioned in the beginning.
- Voting is a straight-forward adaptation of voting for distribution classifiers. Instead of giving its entire vote to the class it considers to be most likely, each classifier is allowed to split its vote according to the base classifier's estimate of the class probability distribution for the example. It is mainly included as a benchmark of the performance that could be obtained without resorting to the expensive CV of every other algorithm.
- Stacking is the stacking algorithm as implemented in WEKA, which follows (Ting & Witten, 1999).

Table 2. This table shows the performance of Stacking with different meta-learners by four measures which are described in the text. Performance on multi-class and two-class datasets is shown separately, in two adjacent columns. All data is based on a single ten-fold cross-validation.

	DecisionTable		J48		KernelDensity		KStar		MLR		NaiveBayes	
	2Cl	mulCl	2Cl	mulCl	2Cl	mulCl	2Cl	mulCl	2Cl	mulCl	2Cl	mulCl
Avg.acc.	0.842	0.709	0.839	0.827	0.815	0.828	0.816	0.810	0.856	0.826	0.852	0.825
by Acc_{best}	0.974	0.798	0.971	0.958	0.939	0.960	0.943	0.937	0.990	0.963	0.987	0.950
by Acc_{XVal}	0.988	0.809	0.985	0.971	0.953	0.973	0.956	0.949	1.005	0.976	1.001	0.963
by Acc_{Voting}	0.987	0.810	0.983	0.972	0.951	0.974	0.955	0.950	1.003	0.977	0.999	0.964

It constructs the meta dataset by adding entire class probability distributions instead of only the most likely class. Following (Ting & Witten, 1999), we also used MLR as the level 1 learner.

All meta-classification schemes used the following six base learners, which were chosen to cover a variety of different biases.

- DecisionTable: a decision table learner.
- J48: a Java port of C4.5 Release 8 (Quinlan, 1993)
- NaiveBayes: the Naive Bayes classifier using multiple kernel density estimation for continuous attributes.
- KernelDensity: a simple kernel density classifier.
- MLR: a multi-class learner which tries to separate each class from all other classes by linear regression (*multi-response linear regression*)
- KStar: the K* instance-based learner (Cleary & Trigg, 1995)

All algorithms are implemented in WEKA Release 3.1.8. Each of them returns a class probability distribution, i.e., they do not predict a single class, but give probability estimates for each possible class. Parameters for learning schemes which have not been mentioned were left at their default values.

4. Multi-class vs. two-class datasets

In this section, we present results concerning the inferior performance of Stacking with MLR on multi-class datasets and show that all but one meta-learner also suffer from the same weakness.

This weakness is apparent by a variety of measures. By average accuracy[8], Stacking with MLR performs slightly worse on multi-class datasets. However, since average accuracy is not a reliable measure because of the different baselines involved, we investigated three different ways of normalizating the accuracy ($\frac{Acc_{Stacking}}{Acc_{best}}$[9], $\frac{Acc_{Stacking}}{Acc_{XVal}}$, $\frac{Acc_{Stacking}}{Acc_{Voting}}$[10]) and computed the geometric mean[11] of these accuracy ratios, both for two-class and multi-class datasets separately. Detailed results can be found in Table 2, column MLR. All these measures agree that Stacking performs about 3% worse on multi-class datasets. A ranking of all meta-classification schemes based on significant differences also shows this weakness, see Table 5. While the latter may be too optimistic due to the used t-test having a high type-I error, the overall agreement of these different measures seems to make this weakness quite obvious.

When using all available base learners also as meta-learners for Stacking, i.e. running six experiments which use the same set of base learners but different meta-learners, we found that in all but one case[12], again for average accuracy and all three normalization methods, the performance on multi-class datasets was worse than on two-class datasets. So we conclude that this may be a general weakness for Stacking with probability distributions as meta-level data, i.e. of the extension proposed by Ting and Witten (1999).

There are three reasonable explanations: Firstly, since the number of classes is proportional to the number of features in the meta-level dataset, a higher number of classes makes learning harder simply because there are more features which have to be considered – i.e. the curse of dimensionality. Secondly, since Stacking with MLR as meta-learner uses almost the same meta-level data to train each linear model (i.e. only the

[8] See Table 3 and Table 2.

[9] Acc_{best} = accuracy of best base classifier by hindsight according to X-Val, i.e. estimated on the complete dataset.

[10] X-Val and Voting are the accuracies of resp. meta-classification schemes

[11] For ratio values, the geometric mean is more appropriate.

[12] When using KernelDensity as meta-learner, the performance on multi-class datasets is indeed better. However, all measures also agree that Stacking with KernelDensity performs worse than with MLR.

class indicator feature is different), a higher number of classes may decrease the diversity among those linear models. Thirdly, the base classifiers themselves may be susceptible to the curse of dimensionality and pass this susceptibility on to Stacking.

The previous result which hints at a general weakness for Stacking with probability distributions supports the first explanation and discounts the second, since e.g. J48 does not learn one model per class but one model for all classes at once and thus decreased diversity of class models cannot be used as explanationn.

If the first explanation is therefore correct, Stacking with predictions as base data should not suffer from this weakness. Accordingly, our experiments show that, when normalizing with Acc_{Voting}, there are three meta-learners which perform better on multi-class datasets and three meta-learners which perform better on two-class datasets – as would be expected by chance. The observed differences in performance are at most 1%. Average accuracy does not agree with this conclusion, but it is an unreliable indicator at best.

When we normalize with Acc_{XVal} or Acc_{best}, we also get similar conflicting results. However, according to our ranking (see Table 5), X-Val performs worse on two-class datasets while Voting offers more balanced results. Thus, when using Acc_{XVal} or related measure Acc_{best} for normalization, this leads to an systematic overestimation of the performance on two-class datasets and thus to a relative underestimation of the performance on multi-class datasets.

So we still conclude that Stacking with predictions does not seem to suffer from the mentioned weakness. This also discounts the third explanation. Thus, even though the base classifiers may still perform worse on multi-class problems, Stacking with predictions seems to be able to compensate for this bias – as is StackingC which we will see shortly.

5. StackingC versus Stacking

In this section we will compare StackingC and Stacking in detail, focussing on performance and runtime differences. Table 3 shows detailed accuracies, standard deviations and significant differences on all datasets and also average accuracy on two-class and multi-class datasets respectively. The last measure has to be interpreted carefully, since it combines problems with different baselines. Also, the ratio between runtimes[13] is given.

Table 3. This table shows accuracies ± standard deviations of Stacking and StackingC. *RRT* shows $\frac{St}{StC}Runtime$, i.e. the runtime ratio, greater than one where StackingC is faster. Diff shows + and - for significant wins resp. losses of StackingC to Stacking and is empty in case of no significant differences.

DS	Cl.	RRT	StackingC	Stacking	Diff
aud	24	2.86	82.17%	76.02%	+
aut	7	2.00	84.20%	82.20%	+
b-s	3	1.00	90.22%	89.50%	+
b-c	2	1.97	72.13%	72.06%	
b-w	2	1.08	97.38%	97.41%	
col	2	2.21	84.70%	84.78%	
c-a	2	2.30	86.22%	86.09%	
c-g	2	1.65	76.24%	76.17%	
dia	2	1.15	76.48%	76.32%	
gla	7	1.67	77.20%	76.45%	
h-c	5	1.25	84.09%	84.26%	
h-h	5	1.27	85.10%	85.14%	
h-s	2	1.67	84.30%	84.04%	
hep	2	1.13	82.97%	83.29%	
ion	2	1.23	92.82%	92.82%	
iri	3	0.90	95.40%	94.93%	
lab	2	1.08	90.88%	91.58%	
lym	4	1.19	81.82%	80.20%	+
p-t	22	17.73	47.23%	42.63%	+
seg	7	2.00	98.10%	98.08%	
son	2	1.88	85.63%	85.58%	
soy	19	2.80	93.47%	92.90%	
veh	4	1.72	79.36%	79.89%	
vot	2	1.81	96.34%	96.32%	
vow	11	2.64	99.07%	99.00%	
zoo	7	2.37	96.24%	93.96%	+
Avg(2Cl)	2	1.60	85.51%	85.54%	
Avg(mCl)	9.14	2.96	85.26%	83.94%	6+

Summarizing the table, StackingC wins six times against Stacking, always on multi-class datasets, and never loses. In all but one case, StackingC is faster – on average it is 2.3 times faster. StackingC also improves on accuracy by an average of 3.75% for multi-

[13] This ratio has to be interpreted carefully, since the experiments were run on a heterogenous cluster of linux and sun machines with a dynamic mapping of tasks to machines. However, since all tasks were run ten times, each time on a different machine, we still consider these to be rough but useful estimates.

Table 4. This table shows the improvement of StackingC over Stacking as $\frac{Acc_{StackingC}}{Acc_{Stacking}}$ for meta-learners MLR, LWR and M5Prime. All data here is based on a single ten-fold cross-validation.

Meta-learner	multi-class	two-class
MLR	1.0375	0.9983
LWR	1.0256	0.9997
M5Prime	1.0070	1.0000

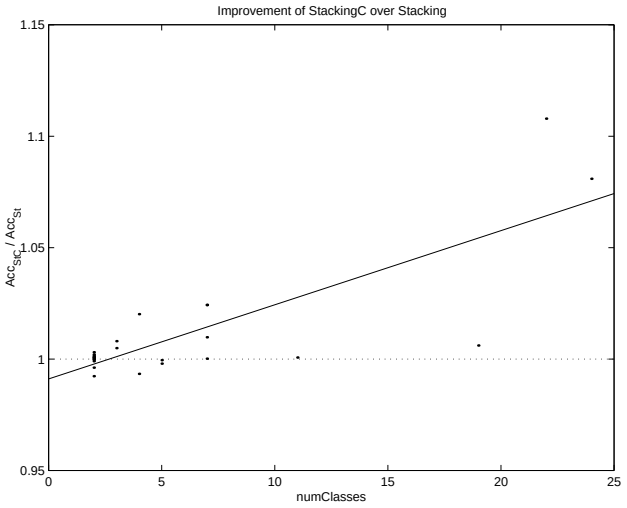

Figure 2. This figure shows the improvement of StackingC over Stacking as $\frac{Acc_{StackingC}}{Acc_{Stacking}}$ as a function of the number of classes. The solid line shows the obtained least squares fit for a one-dimensional linear model while the dotted line indicates a ratio of 1.0. Above the latter line, StackingC is better than Stacking and vice versa.

class datasets; the performance difference for two-class datasets is negligible, see Table 4.

We would also expect StackingC to improve on Stacking more, if the number of classes is increased. This is usually the case, see Figure 2. The statistical correlation coefficient for the shown relation is 0.8. Our figure also shows the fitted regression line which has a mean squared error of 2.59E-04.

Concluding, StackingC improves on Stacking in terms of significant accuracy differences, accuracy ratios and runtime. These improvements are more evident for multi-class datasets and have some tendency to become more pronounced as the number of classes increases. StackingC also resolves the weakness of Stacking in the extension proposed by Ting and Witten (1999) and offers balanced performance on two-class and multi-class datasets.

6. Discussion

To find out whether or not our approach is also able to improve other meta-learners besides MLR, we conducted additional experiments.

Essentially, we tried two different approaches. One was to use two other regression learners instead of MLR to approximate the class membership functions for each class separately. This worked quite well, see Table 4. MLR is the meta-learner which we previously used for StackingC, but we used only the first cross-validation in order to enable a fair comparison to the other learners. LWR stands for *locally weighted regression*[14] and M5Prime stands for a model tree learner. Both learners are available within WEKA. In terms of absolute performance over all datasets, LWR is slightly better than MLR and M5Prime is slightly worse.

The second approach, namely modifying common machine learning algorithms[15] to use only partial probability distributions during prediction, yielded catastrophically bad results on some datasets.

These results indicate that the source of StackingC's improvement may lie more in the diversity of class models than in the dimensionality reduction, although both play a key role (see also Section 2, paragraph 8). Using one-against-all class binarization and regression learners seems to be essential. So we intend to look into other binarization methods in the future.

Another interesting venue for future research may be to find out why X-Val performs worse on two-class datasets as our ranking indicates. A tentative explanation may be that the base classifiers perform better on two-class datasets. Thus, their accuracies are more similar on these datasets, increasing the probability that X-Val will choose a suboptimal learner by its internal CV. Further research is needed to find out if this is indeed the case. This may have far-reaching consequences because of the ubiquitousness of X-Val as a method to choose the best classifier throughout the machine learning community.

7. Related Research

Chan and Stolfo (1995) propose the use of *arbiters* and *combiners*. A combiner is more or less identical to stacking. Chan and Stolfo (1995) also investigate a related form, which they call an *attribute-combiner*. In this architecture, the original attributes are not replaced with the class predictions, but instead they are added to them. As Schaffer (1994) shows in his pa-

[14] We used parameter -W 1 for inverse kernels $\frac{1}{x}$

[15] NaiveBayes, KStar, IBk and KernelDensity.

Table 5. This table shows a ranking of all meta-classification schemes with Stacking and StackingC. We are mostly concerned with the differences between Stacking and StackingC on multi-class datasets. Separate rankings are given on multi-class and two-class datasets. Ranks are given as Wins/Losses with the wins counting for the algorithm in the row (Scheme).

Scheme		X-Val	Grading	Stacking	StackingC	Voting
X-Val	on multi-class ds.	0/0	3/3	3/3	2/6	5/5
Grading	on multi-class ds.	3/3	0/0	5/4	1/5	3/2
Stacking	on multi-class ds.	3/3	4/5	0/0	0/6	4/5
StackingC	on multi-class ds.	6/2	5/1	6/0	0/0	5/3
Voting	on multi-class ds.	5/5	2/3	5/4	3/5	0/0
X-Val	on two-class ds.	0/0	2/3	0/3	0/3	2/4
Grading	on two-class ds.	3/2	0/0	2/3	2/4	1/0
Stacking	on two-class ds.	3/0	3/2	0/0	0/0	3/2
StackingC	on two-class ds.	3/0	4/2	0/0	0/0	4/2
Voting	on two-class ds.	4/2	0/1	2/3	2/4	0/0

per about bi-level stacking, this may result in worse performance.

Cascading by Gama and Brazdil (2000) is a related variant to Stacking where the classifiers are applied in sequence and there is no dedicated level 1 classifier. Each base classifier, when applied to the data, adds his class probability distribution to the data and returns an augmented dataset, which is to be used by the next base classifier. Thus, the order in which the classifiers are executed becomes important. Cascading does not use an internal cross-validation like most other meta-classification schemes and is therefore claimed to be at least three times faster than Stacking. On the other hand in Stacking the classifier order is not important, thereby reducing the degrees of freedom and minimizing chances for overfitting. Furthermore, cascading increases the dimensionality of the dataset with each step whereas Stacking's meta-dataset has a dimensionality which is independent of the dimensionality of the dataset, i.e. the number of base classifiers multiplied with the number of classes.

Todorovski and Dzeroski (2000) introduce a novel method to combine multiple models. Instead of directly predicting the final class as all combining schemes we considered, their meta-learner MDT, based on C4.5, specifies which model to use for each example based on statistical and information theoretic measures computed from the class probability distribution. While their approach may make the combining scheme more comprehensible by learning an explicit decision tree decision tree, it is unclear whether this leads to better insight as well. Stacking and StackingC using MLR as meta-learner also allow to determine relative importance of base learners per class, simply by inspecting the weights of meta-level attributes after training – this has e.g. been done by Ting and Witten (1999).

Skalak (1997) includes an excellent overview about methods for constructing classifier ensembles. His other main contribution consists of investigating ensembles of coarse instance-based classifiers storing only a few prototypes per class.

Merz (1999) studies the use of correspondence analysis and lazy learning to combine classifiers in a stacking-like setting. He compares his approach, SCANN, to two Stacking-variants with NaiveBayes resp. a backpropagation-trained neural network as meta-learner. MLR was not considered as meta-learner. According to experiments with synthetic data, his approach is equivalent to plurality vote if the models make uncorrelated errors. However, in practice this is seldom the case. Moreover, his approach is limited to using predictions as meta-level data and would fail for the class probability distributions which we use. Still, correspondence analysis as a means of less static dimensionality reduction of stacking meta-level data may have its merits.

Ting and Witten (1999) deal with the type of generalizer suitable to derive the higher-level model and the kind of attributes it should use as input. Using probability distributions as they propose instead of just predictions is essential to our variant StackingC. They also investigated the usefulness of non-negativity constraints for feature weights within linear models, but found that it is not essential to get the best performance. However they found it may be useful to facilitate human comprehension of these models. Since our focus was on performance and not on comprehen-

sibility, we did not use a non-negativity constraint. In the future it may be interesting to look into comparing their linear models to those found by StackingC to see where they differ.

8. Conclusion

We have presented empirical evidence that Stacking in the extension proposed by (Ting & Witten, 1999) performs worse on multi-class datasets than on two-class datasets, for all but one meta-learner we investigated.

This can be explained as follows: With a higher number of classes, the dimensionality of the meta-level data is proportionally increased. This higher dimensionality makes it harder for meta-learners to find good models, since there are more features to be considered. Stacking using meta-level data consisting of predictions does not suffer from this weakness, as would be expected.

In order to improve on the status quo, we have proposed and implemented a new Stacking variant, StackingC, based on reducing the dimensionality of the meta-dataset so as to be independent of the number of classes and removing a priori irrelevant features, and shown that it resolves this previously unreported weakness, for MLR and two other related meta-learners considered. We believe that the source of this improvement lies partially in the dimensionality reduction, but more importantly in the higher diversity of class models. Using one-against-all class binarization and regression learners for each class model seems to be essential.

Acknowledgements

This research is supported by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant no. P12645-INF. The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry of Education, Science and Culture. We would like to thank Johannes Fürnkranz for comments.

References

Blake, C. L., Merz, C. J: UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html` (1998). Department of Information and Computer Science, University of California at Irvine, Irvine CA.

Chan, P. K., & Stolfo, S. J. (1995). A comparative evaluation of voting and meta-learning on partitioned data. *Proceedings of the 12th International Conference on Machine Learning (ICML-95)* (pp. 90–98). Morgan Kaufmann.

Cleary, J. G., Trigg, L. E: K*: An instance-based learner using an entropic distance measure. In Prieditis, A., Russell, S., Proceedings of the 12th International Conference on Machine Learning (1995) 108–114, Lake Tahoe, CA.

Dietterich, T.G.: Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. Neural Computation, 10 (7) 1895-1924.

Gama J., Brazdil P.: Cascade Generalization, Machine Learning *41(3)*, 315-344, 2000.

Kononenko, I., Bratko, I: Information-based evaluation criterion for classifier's performance. Machine Learning *6* (1991) 67–80.

Merz, C.J.: Using Correspondence Analysis to Combine Classifiers, Machine Learning, *36(1-2)* (1999) 33–58.

Quinlan, J. R: C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, CA.

Schaffer, C.: Selecting a classification method by cross-validation. Machine Learning *13(1)*, 135–143, 1993.

Schaffer, C.: Cross-validation, stacking and bi-level stacking: Meta-methods for classification learning, in P. Cheeseman and R. W. Oldford (Eds.), *Selecting models from data: Artificial Intelligence and Statistics IV*, Springer-Verlag, 51–59, 1994.

Seewald A.K., Fürnkranz J.: An Evaluation of Grading Classifiers, in Hoffmann F. et al. (eds.), Advances in Intelligent Data Analysis, 4th International Conference, IDA 2001, Proceedings, Springer, Berlin/Heidelberg/New York/Tokyo, pp.115-124, 2001.

Skalak, D. B: Prototype Selection for Composite Nearest Neighbor Classifiers. PhD Dissertation (1997), University of Massachusetts, Amherst, Massachusetts.

Ting, K. M., Witten, I. H: Issues in stacked generalization. Journal of Artificial Intelligence Research *10* (1999) 271–289.

Todorovski L., Dzeroski S.: Combining Multiple Models with Meta Decision Trees, in Zighed D.A. et al. (eds.), Principles of Data Mining and Knowledge Discovery, Lecture Notes in Artificial Intelligence, Springer-Verlag, pp.54-64, 2000.

Wolpert, D. H: Stacked generalization. Neural Networks *5(2)* (1992) 241–260.