A comparison of stacking with MDTs to bagging, boosting, and other stacking methods

Bernard Ženko, Ljupčo Todorovski, and Sašo Džeroski

Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, Ljubljana, Slovenia

Abstract. In this paper, we present an integration of the algorithm MLC4.5 for learning meta decision trees (MDTs) into the Weka data mining suite. MDTs are a method for combining multiple classifiers. Instead of giving a prediction, MDT leaves specify which classifier should be used to obtain a prediction. The algorithm is based on the C4.5 algorithm for learning ordinary decision trees. An extensive performance evaluation of stacking with MDTs on twenty-one data sets has been performed. We combine base-level classifiers generated by three learning algorithms: an algorithm for learning decision trees, a nearest neighbor algorithm and a naive Bayes algorithm. We compare MDTs to bagged and boosted decision trees, and to combined classifiers with voting and three different stacking methods: with ordinary decision trees, with naive Bayes algorithm and with multi-response linear regression as a meta-level classifier. In terms of performance, stacking with MDTs gives better results than other methods except when compared to stacking with multi-response linear regression as a meta-level classifier; the latter is slightly better than MDTs.

1 Introduction

The task of constructing ensembles of classifiers [3] can be broken down into two sub-tasks. We first have to *generate* a diverse set of base-level classifiers. Once the base-level classifiers have been generated, the issue of how to *combine* their predictions arises.

Several approaches to *generating* base-level classifiers are possible. One approach is to generate classifiers by applying different learning algorithms (with heterogeneous model representations) to a single data set (see, e.g., [7]). Another possibility is to apply a single learning algorithm with different parameters settings to a single data set. Finally, methods like bagging [2] and boosting [4] generate multiple classifies by applying a single learning algorithm to different versions of a given data set. Two different methods for manipulating the data set are used: random sampling with replacement (also called bootstrap sampling) in bagging and re-weighting of the misclassified training examples in boosting.

Techniques for *combining* the predictions obtained from the multiple base-level classifiers can be clustered in three combining frameworks: voting (used in bagging and boosting), stacked generalization or stacking [14] and cascading [5].

In voting, each base-level classifier gives a vote for its prediction. The prediction receiving the most votes is the final prediction. In stacking, a learning algorithm is used to learn how to combine the predictions of the base-level classifiers. The induced meta-level classifier is then used to obtain the final prediction from the predictions of the base-level classifiers. Cascading is an iterative process of combining classifiers: at each iteration, the training data set is extended with the predictions obtained in the previous iteration.

The work presented here focuses on combining the predictions of base-level classifiers induced by applying different learning algorithms to a single data set. It adopts the stacking framework, where we have to learn how to combine the base-level classifiers. We apply meta decision trees (MDTs) [11] for the task of combining base-level classifiers. The difference between meta and ordinary decision trees (ODTs) is that MDT leaves specify which base-level classifier should be used, instead of predicting the class value directly. The attributes used by MDTs are derived from the class probability distributions predicted by the base-level classifiers for a given example.

In this paper, we present an integration of the algorithm for inducing MDTs [11], in the Weka data mining suite [13]. For this purpose we implemented MLJ4.8, a modification of J4.8, Weka re-implementation of C4.5 [9] algorithm for induction of ODTs. MDTs and MLJ4.8 are described in more detail in Section 2.

The performance of MDTs is compared to the performance of other standard combining methods implemented in Weka. These include majority voting, boosting and bagging of decision trees and stacking with three different meta-level learners. The comparison is made on a collection of twenty-one data sets. Classifiers induced with three base-level learning algorithms, implemented in Weka, are combined: the J4.8 tree induction algorithm, the k-nearest neighbor (k-NN) algorithm IBk [1] and the naive Bayes algorithm with kernel density estimators [6].

Section 3 reports on the experimental methodology and discusses the result of the comparison. The last section presents conclusions, discusses the related work on combining classifiers and gives directions for further work.

2 Meta Decision Trees

In this section, we first introduce meta decision trees (MDTs). We then discuss the possible sets of meta-level attributes used to induce MDTs. Finally, we present an algorithm for inducing MDTs, named MLJ4.8.

2.1 What are Meta Decision Trees

The structure of a meta decision tree is identical to the structure of an ordinary decision tree. A decision (inner) node specifies a test to be carried out on a single attribute value and each outcome of the test has its own branch leading to the appropriate subtree. In a leaf node, a MDT predicts which classifier is to be

used for classification of an example, instead of predicting the class value of the example directly (as an ODT would do).

The difference between ordinary and meta decision trees is illustrated with the example in Table 1. In the meta-level data set M (Table 1a), the meta-level attributes C_1 and C_2 are the class value predictions of two base-level classifiers C_1 and C_2 for a given example. The two additional meta-level attributes $Conf_1$ and $Conf_2$ measure the confidence of the predictions of C_1 and C_2 for a given example.

The meta decision tree induced using the meta-level data set M is given in Table 1b). The MDT is interpreted as follows: if the confidence $Conf_1$ of the base-level classifier C_1 is high, then C_1 should be used for classifying the example, otherwise the base-level classifier C_2 should be used. The ordinary decision tree induced using the same meta-level data set M (given in Table 1c) is much less comprehensible, despite the fact that it reflects the same rule for choosing among the base-level predictions. Note that both the MDT and the ODT need the predictions of the base-level classifiers in order to make their own predictions.

Table 1. The difference between ordinary and meta decision trees.

a) Meta-level data set M

$Conf_1$	C_1	$Conf_2$	C_2	*true class*
0.875	0	0.625	0	0
0.875	0	0.375	1	0
0.875	1	0.375	0	1
0.875	1	0.625	1	1
0.125	0	0.625	0	0
0.125	0	0.625	1	1
0.125	1	0.625	0	0
0.125	1	0.625	1	1

b) The MDT induced from M (by MLC4.5)

```
Conf1 <= 0.125: C2
Conf1 > 0.125: C1
```

c) The ODT induced from M (by C4.5)

```
C1 = 0:
|    Conf1 > 0.125 : 0
|    Conf1 <= 0.125 :
|    |    C2 = 0: 0
|    |    C2 = 1: 1
C1 = 1:
|    Conf1 > 0.125 : 1
|    Conf1 <= 0.125 :
|    |    C2 = 0: 0
|    |    C2 = 1: 1
```

Note that in the process of inducing meta decision trees two types of attributes are used. *Ordinary* attributes are used in the decision (inner) nodes of the MDT (e.g., attributes $Conf_1$ and $Conf_2$ in the example meta-level data set M). The role of these attributes is identical to the role of attributes used for inducing ordinary decision trees. *Class* attributes (e.g., C_1 and C_2 in M), on the other hand, are used in the leaf nodes only. Each base-level classifier has its class attribute: the values of this attribute are the predictions of the base-level classifier. Thus, the class attribute assigned to the leaf node of the MDT decides which base-level classifier will be used for prediction. When inducing ODTs for combining classifiers, the class attributes are used in the same way as ordinary attributes.

We use properties of the class probability distributions predicted by the base-level classifiers as ordinary attributes for inducing MDTs. These properties reflect the certainty and confidence of the predictions. Details about the set of meta-level attributes are given in the following subsection.

2.2 Meta-Level Attributes

We calculate the properties of the class probability (CDP) distributions predicted by the base-level classifiers that reflect the certainty and confidence of the predictions.

First, $\texttt{maxprob}(x, C)$ is the highest class probability (i.e. the probability of the predicted class) predicted by the base-level classifier C for example x:

$$\texttt{maxprob}(x, C) = \max_{i=1}^{k} p_C(c_i|x).$$

Next, $\texttt{entropy}(x, C)$ is the entropy of the class probability distribution predicted by the classifier C for example x:

$$\texttt{entropy}(x, C) = -\sum_{i=1}^{k} p_C(c_i|x) \cdot \log_2 p_C(c_i|x).$$

Finally, $\texttt{weight}(x, C)$ is the fraction of the training examples used by the classifier C to estimate the class distribution for example x. For decision trees, it is the weight of the examples in the leaf node used to classify the example. For rules, it is the weight of the examples covered by the rule(s) which was used to classify the example. This property was not used for the nearest neighbor and naive Bayes classifiers, as it does not apply to them in a straightforward fashion.

The entropy and the maximum probability of a probability distribution reflect the certainty of the classifier in the predicted class value. If the probability distribution returned is highly spread, the maximum probability will be low and the entropy will be high, indicating the classifier is not very certain in its prediction of the class value. On the other hand, if the probability distribution returned is highly focused, the maximum probability is high and the entropy low, thus indicating the classifier is certain in the predicted class value. The weight quantifies how reliable is the predicted class probability distribution. Intuitively, the weight corresponds to the number of training examples used to estimate the probability distribution: the higher the weight, the more reliable the estimate.

An example MDT, induced in the image domain, is given in Table 2. The leaf denoted by an asterisk `(*)` specifies that the C4.5 classifiers is to be used to classify an example, if (1) the maximum probability in the class probability distribution predicted by k-NN is smaller than 0.77; (2) the fraction of the examples in the leaf of the C4.5 tree used for prediction is bigger than 0.4% of all the examples in the training set; and (3) the entropy of the class distribution predicted by C4.5 is less then 0.14. In sum, if the k-NN classifier is not very confident in its prediction (1) and the C4.5 classifier is very confident in its prediction

Table 2. A meta decision tree induced in the image domain using class distribution properties as ordinary attributes.

```
knn_maxprob > 0.77147 : KNN
knn_maxprob <= 0.77147 :
|       c45_weight <= 0.00385 : KNN
|       c45_weight > 0.00385 :
|       |       c45_entropy <= 0.14144 : C4.5 (*)
|       |       c45_entropy > 0.14144 : LTREE
```

(3 and 2), the leaf recommends using the C4.5 prediction; this is consistent with common-sense knowledge in the domain of classifier combination.

Note here another important property of MDTs: they are domain independent in the sense that the same language for expressing meta decision trees is used in all domains once we fix the set of base-level classifiers to be used. This means that a MDT induced in one domain can be used in any other domain for combining the same set of base-level classifiers (although it may not perform very well). In part, this is due to the fact that the set of meta-level attributes is domain independent. It depends only on the set of base-level classifiers. However, an ODT built from the same set of meta-level attributes is still domain dependent for two reasons. First, it uses tests on the class values predicted by the base-level classifiers (e.g., the tests `C1 = 0` / `C1 = 1` in the root node of the ODT from Table 1c). Second, an ODT predicts the class value itself, which is clearly domain dependent.

In sum, there are three reasons for the domain independence of MDTs: (1) the set of meta-level attributes; (2) not using class attributes in the decision (inner) nodes and (3) predicting the base-level classifier to be used instead of predicting the class value itself.

2.3 MLJ4.8 - a Re-implementation of Meta Decision Trees in Java

In this subsection, we present MLJ4.8, a re-implementation of MLC4.5 [11] algorithm for induction of MDTs in Java. MLJ4.8 is based on J4.8 [13], a re-implementation of Quinlan's C4.5 [9] algorithm for inducing ordinary decision trees. There are several minor implementation differences between J4.8 and C4.5: in such cases we decided to use C4.5 as a reference, rather than J4.8. MLJ4.8 takes as input a meta-level data set consisted of ordinary and class attributes. The only differences between MLJ4.8 and J4.8 are:

1. Only ordinary attributes are used in internal nodes;
2. Assignments of the form $C = C_i$ (where C_i is a class attribute) are made by MLJ4.8 in leaf nodes, as opposed to assignments of the form $C = c_i$ (where c_i is a class value);
3. The goodness-of-split for internal nodes is calculated differently (as described below);
4. MLJ4.8 does not post-prune the induced MDTs.

The rest of the MLJ4.8 algorithm is identical to the original J4.8 algorithm. Below we describe J4.8's and MLJ4.8's measures for selecting attributes in internal nodes.

J4.8 is a greedy divide and conquer algorithm for building classification trees. At each step, the best split according to the gain (or gain ratio) criterion is chosen from the set of all possible splits for all attributes. The gain criterion is based on the entropy of the class probability distribution of the examples in the current subset S of training examples:

$$\mathit{info}(S) = -\sum_{i=1}^{k} p(c_i, S) \cdot \log_2 p(c_i, S)$$

where $p(c_i, S)$ denotes the relative frequency of examples in S that belong to class c_i. The gain criterion selects the split that maximizes the decrement of the *info* measure.

In MLJ4.8, we are interested in the accuracies of each of the base-level classifiers C from set $\mathcal{C}$ on the examples in data set S, i.e., the proportion of examples in S that have a class equal to the class attribute C. The measure, used in MLJ4.8, is defined as:

$$\mathit{info}_{\mathcal{ML}}(S) = 1 - \max_{C \in \mathcal{C}} \texttt{accuracy}(C, S),$$

where $\texttt{accuracy}(C, S)$ denotes the relative frequency of examples in S that are correctly classified by the base-level classifier C. The vector of accuracies does not have probability distribution properties (its elements do not sum to 1), so the entropy can not be calculated. This is the reason for replacing the entropy based measure with an accuracy based one.

3 Experimental comparison

In order to compare the performance of meta decision trees with other combining schemes, we performed experiments on a collection of twenty-one data sets from the UCI Repository of Machine Learning Databases and Domain Theories [8]. These data sets have been widely used in other comparative studies.

Three learning algorithms were used in the base-level experiments: tree learning algorithm J4.8, which is a re-implementation of C4.5, k-nearest neighbor (k-NN or IBk) algorithm and naive Bayes (NB) algorithm. We used implementations in the Java programming language incorporated in the Weka data mining suite [13]. J4.8 was used with the default settings. For the IBk algorithm, k was selected with cross validation in the range from 1 to 77 and inverse distance weighting was used. Naive Bayes algorithm used kernel density estimation. The output of each base-level classifier for each example consists of predicted class and class probability distribution. The classification errors of the base-level classifiers are presented in Table 3.

In all experiments presented here, classification errors are estimated using 10-fold stratified cross validation. Cross validation is repeated ten times using

Table 3. Classification errors (in %) of base-level classifiers J4.8, IBk and naive Bayes and of bagging and boosting. Bagging and boosting used J4.8 as base learning algorithm.

Data set	J4.8	IBk	Naive Bayes	Bagging J48	Boosting J48
australian	14.54±0.55	13.45±0.29	18.65±0.12	13.67±0.40	15.58±0.32
balance	22.43±0.35	9.90±0.20	8.48±0.14	17.31±0.31	21.49±0.39
breast-w	5.39±0.20	4.28±0.13	2.69±0.05	4.98±0.24	3.71±0.37
bridges-td	14.71±0	16.57±1.11	14.02±0.45	14.90±0.39	19.41±1.43
car	7.44±0.25	5.83±0.15	14.40±0.15	6.78±0.26	4.16±0.16
chess	0.60±0.05	2.87±0.11	12.16±0.06	0.61±0.04	0.38±0.05
diabetes	26.26±0.50	25.55±0.30	24.70±0.12	24.62±0.55	28.53±0.58
echo	34.58±2.39	34.73±1.55	27.33±0.44	31.68±1.35	33.89±2.01
german	28.82±0.88	26.01±0.51	25.43±0.19	26.37±0.37	29.23±0.50
glass	32.24±0.91	29.67±0.86	49.86±0.60	26.03±1.60	23.18±1.40
heart	22.19±1.24	18.52±0.48	15.67±0.34	19.78±0.89	21.78±1.32
hepatitis	20.90±0.97	17.29±0.63	15.35±0.29	17.68±1.22	18.26±0.88
hypo	0.72±0.02	2.79±0.09	1.81±0.02	0.78±0.04	1.05±0.05
image	3.18±0.16	2.84±0.07	14.29±0.06	2.55±0.17	1.84±0.12
ionosphere	10.26±0.47	13.28±0.33	8.15±0.21	7.83±0.44	6.41±0.37
iris	5.33±0.44	4.67±0.40	4.07±0.16	5.73±0.29	5.80±0.60
soya	7.54±0.35	8.96±0.26	7.12±0.10	7.23±0.41	7.07±0.35
tic-tac-toe	15.11±0.45	0.96±0.06	30.22±0.12	6.80±0.63	3.43±0.53
vote	3.54±0.17	6.97±0.22	9.82±0.07	3.93±0.26	4.48±0.43
waveform	23.62±0.25	14.42±0.10	19.24±0.05	18.00±0.13	18.58±0.13
wine	6.57±1.05	3.26±0.47	2.64±0.17	5.11±0.70	4.04±0.64
Average	14.57±0.55	12.52±0.40	15.53±0.19	12.49±0.51	12.97±0.60

different random generator seeds resulting in ten different sets of folds. The same folds (random generator seeds) were used in all experiments.

On the meta-level, the performances of seven algorithms for combining classifiers are compared. Two of them, bagging and boosting, apply the same learning algorithm to different versions of a given data set in order to obtain a set of diverse base-level classifiers. The third combining algorithm is majority voting. The other four combining algorithms are different versions of stacking. A short description of each of them follows.

Bagging uses random sampling with replacement in order to obtain different versions of a given data set. The size of each sampled data set equals the size of the original data set. On each of these versions of the data set the same learning algorithm, J4.8 in our case, is applied. Classifiers obtained in this manner are then combined with majority voting. For more information see [2].

Boosting first builds a classifier with some learning algorithm (again J4.8 in our case) from the original data set. The weights of the misclassified examples are then increased and another classifier is built using the same learning algorithm. The procedure is repeated several times. Classifiers derived in this manner are

Table 4. Classification errors (in %) of voting, stacking with J4.8, with naive Bayes, with MLR and with meta decision trees. Base learning algorithms for all were J4.8, IBk and naive Bayes.

Data set	Voting	Stacking j48	Stacking NB	Stacking MLR	Stacking MDT
australian	13.81±0.34	14.61±0.51	14.35±0.36	14.16±0.42	13.77±0.38
balance	8.91±0.36	6.02±0.52	9.17±0.42	9.47±0.18	8.51±0.19
breast-w	3.46±0.13	2.78±0.13	2.89±0.09	2.73±0.07	2.69±0.07
bridges-td	15.78±0.80	16.76±1.32	17.35±1.31	14.12±0.67	16.08±0.84
car	6.49±0.16	1.63±0.21	2.65±0.14	5.61±0.23	5.02±0.27
chess	1.46±0.08	0.75±0.09	0.75±0.08	0.60±0.05	0.60±0.05
diabetes	24.01±0.33	25.73±0.68	25.25±0.42	23.78±0.56	24.74±0.54
echo	29.24±1.46	26.56±1.16	28.02±1.44	28.63±0.61	27.71±0.76
german	25.19±0.53	25.47±0.72	27.09±0.24	24.36±0.20	25.60±0.30
glass	29.67±0.80	38.60±2.16	50.61±1.51	30.93±1.28	31.78±1.19
heart	17.11±0.76	17.59±0.80	16.93±0.48	15.30±0.64	16.04±0.46
hepatitis	17.42±0.88	18.90±0.94	17.35±0.78	15.68±0.69	15.87±0.84
hypo	1.32±0.02	0.83±0.07	1.17±0.04	0.72±0.02	0.79±0.07
image	2.94±0.14	3.29±0.23	6.52±0.43	2.84±0.14	2.53±0.09
ionosphere	7.18±0.39	6.10±0.65	7.12±0.44	7.35±0.42	8.83±0.62
iris	4.20±0.28	6.13±0.97	5.00±0.41	4.47±0.35	4.73±0.42
soya	6.75±0.18	8.02±0.29	6.56±0.21	7.22±0.30	7.06±0.14
tic-tac-toe	9.24±0.33	0.42±0.10	1.21±0.11	0.58±0.09	0.96±0.06
vote	7.10±0.19	4.07±0.34	5.06±0.10	3.54±0.17	3.54±0.17
waveform	15.90±0.15	14.38±0.13	15.03±0.11	14.33±0.12	14.40±0.11
wine	1.74±0.11	3.82±0.76	3.48±0.44	2.87±0.35	3.26±0.60
Average	11.85±0.40	11.55±0.61	12.55±0.46	10.92±0.36	11.17±0.39

then combined using weighted voting. The AdaBoost.M1 variant of boosting was used in our experiments. For more information see [4].

Voting is a simple majority voting algorithm.

Stacking J4.8 uses J4.8 as a meta-level learning algorithm. Meta-level data consists of class probability distribution for each base-level classifier along with the actual class. For more information see [14] and [10]. Note here that in the previous study of MDTs [11] a different set of meta-level attributes for stacking with ODTs was used. Instead of using class probability distributions directly, only their aggregations (maximal probability and entropy) were used. Thus, the results presented here are not directly comparable to the ones presented in [11].

Stacking NB uses naive Bayes as a meta-level learning algorithm. The structure of meta-level data is the same as the one for Stacking J4.8.

Stacking MLR uses a multi-response linear regression algorithm (MLR) as a meta-level learning algorithm. MLR transforms a classification problem into a set of regression problems; one problem for each class value. Then, linear regression is used to predict the probability of the selected class value. If there are discrete attributes in the data set, they are transformed to binary ones. For more information see [10].

Table 5. Relative improvement in accuracy (in %) of stacking with meta decision trees when compared to bagging, boosting, voting, stacking with J4.8, with naive Bayes and with MLR and its significance (+/– means significantly better/worse, x means insignificant).

	Bag. J48			Boo. J48			Voting			Sta. J48			Sta. NB			Sta. MLR		
Data set	rel. im.		sig.	rel. im.		sig.	rel. im.		sig.	rel. im.		sig.	rel. im.		sig.	rel. im.		sig.
australian	-0.74		x	11.63		x	0.31		x	5.75		x	4.04		x	2.76		x
balance	50.83		+	60.39		+	4.49		+	-41.49		–	7.16		+	10.14		+
breast-w	45.98		+	27.41		+	22.31		+	3.09		+	6.93		+	1.57		+
bridges-td	-7.89		–	17.17		+	-1.86		–	4.09		+	7.34		+	-13.89		–
car	25.96		+	-20.75		–	22.73		+	-208.54		–	-89.30		–	10.62		+
chess	1.55		x	-56.55		x	59.10		x	20.42		x	20.42		x	0.00		x
diabetes	-0.48		x	13.28		x	-3.04		x	3.85		x	2.01		x	-4.05		x
echo	12.53		+	18.24		+	5.22		+	-4.31		–	1.09		+	3.20		+
german	2.92		+	12.42		+	-1.63		–	-0.51		–	5.50		+	-5.09		–
glass	-22.08		–	-37.10		–	-7.09		–	17.68		+	37.21		+	-2.72		–
heart	18.91		+	26.36		+	6.28		+	8.84		+	5.25		+	-4.84		–
hepatitis	10.22		x	13.07		x	8.89		x	16.04		x	8.55		x	-1.23		x
hypo	-1.62		x	24.62		x	40.09		x	4.56		x	32.34		x	-9.61		x
image	0.68		x	-37.65		x	13.72		x	22.92		x	61.16		x	10.82		x
ionosphere	-12.73		–	-37.78		–	-23.02		–	-44.86		–	-24.00		–	-20.16		–
iris	17.44		+	18.39		+	-12.70		–	22.83		+	5.33		+	-5.97		–
soya	2.43		x	0.21		x	-4.55		x	12.04		x	-7.59		x	2.23		x
tic-tac-toe	85.87		+	72.04		+	89.60		+	-130.00		–	20.69		+	-64.29		–
vote	9.94		+	21.03		+	50.16		+	12.99		+	30.00		+	0.00		x
waveform	20.00		+	22.50		+	9.44		+	-0.15		–	4.20		+	-0.53		–
wine	36.26		+	19.44		+	-87.10		–	14.71		+	6.45		+	-13.73		–
Average	19.89			14.78			18.34			-4.24			10.59			-4.07		
W/L	11+/3–			11+/3–			8+/6–			7+/7–			12+/2–			4+/9–		

Stacking MDT uses meta decision trees described in Section 2 and [11] as a meta-level learning algorithm. Meta-level data consists of the maximal class probability and entropy of the class probability distribution for each base-level classifier (as ordinary attributes) and classes predicted by base-level classifiers (as class attributes) along with the actual class.

The classification errors of the combining algorithms averaged over ten runs of ten-fold cross validation are presented in tables 3 and 4. Assessment of performance is based on the calculation of relative improvement and the paired t-test, as described below. In order to evaluate the accuracy improvement achieved using classifier C_1 as compared to using classifier C_2 we calculate the relative improvement: $1 - \texttt{error}(C_1)/\texttt{error}(C_2)$. In the analysis presented here, we compare the performance of meta decision trees to other approaches: C_1 will thus refer to combiners with meta decision trees. The average relative improvement is calculated using geometric mean: $1 - \texttt{geometric\_mean}(\texttt{error}(C_1)/\texttt{error}(C_2))$.

The statistical significance of the difference in classification errors is tested using the paired t-test (exactly the same folds are used for C_1 and C_2) with significance level of 95%. $+/-$ in table 5 means that the classifier C_1 (MDT in our case) is significantly better/worse than C_2 and x means that the difference is insignificant.

3.1 Stacking with MDTs vs. bagging and boosting

Tables 4 and 5 compare the performance of bagging and boosting of decision trees to the performance of stacking with MDTs. The latter performs significantly better than bagging on eleven and significantly worse on three out of 21 data sets. The overall relative improvement of accuracy is 20%. Stacking with MDTs performs significantly better than boosting on eleven and significantly worse on three data sets. The overall relative improvement of accuracy is 15%. It is therefore evident that stacking with MDTs outperforms both bagging and boosting of decision trees.

3.2 Stacking with MDTs vs. voting

The results of the performance comparison between voting and stacking with MDTs can be found in Table 4. Both methods use the same base-level classifiers. Stacking with MDTs performs significantly better on eight and significantly worse on six data sets, but four out of six relative decreases in accuracy are less than 5%. The overall relative improvement of accuracy is therefore still high, 18%. These findings are consistent with the previous study of MDTs [11].

3.3 Stacking with MDTs vs. stacking with J4.8, naive Bayes and MLR

The results of the performance comparison between three versions of stacking with J4.8, naive Bayes and MLR on one side and stacking with MDTs on the other, can be found in tables 4 and 5.

Stacking with MDTs performs significantly better than stacking with J4.8 on seven and significantly worse on seven out of 21 data sets. There is a 4% overall relative decrease in accuracy (calculated as a geometric mean), but this is mostly due to data sets car and tic-tac-toe, where all combining methods perform very well. If we exclude these two data sets a 7% overall relative improvement is obtained. In another three data sets the relative decreases in accuracy are below 5%. Besides, MDTs are much smaller than ODTs induced with J4.8, and can therefore be interpreted by humans. These results are consistent with the previous study of MDTs [11].

Note here that the improvement of performance is a bit smaller than the one reported in [11]. This can be due to the fact that different set of meta-level attributes was used by J4.8 as compared to the one used by C4.5 in [11]. In this study meta-level data consisted of class probability distribution for each base-level classifier along with the actual class, while in [11] meta-level data consisted

of maximal class probability and entropy of the class probability distribution for each base-level classifier and classes predicted by base-level classifiers along with the actual class.

Stacking with MDTs performs significantly better than stacking with naive Bayes on twelve and significantly worse on only two data sets. There is an 11% overall relative improvement in accuracy. It is therefore evident that stacking with MDTs outperforms stacking with naive Bayes.

Stacking with MDTs performs significantly better than stacking with MLR on four and significantly worse on nine domains. There is a 4% overall relative decrease in accuracy which can not be pinpointed to a few domains only. We can say that stacking with MLR performs better than stacking with MDTs, though the difference in performance is small. The good performance of stacking with MLR is no surprise, since MLR is recommended by [10] as the best meta-level classifier.

4 Conclusions and further work

We have presented the integration of the algorithm for building meta decision trees (MDTs) [11] into Weka data mining suite [13]. This enables experiments with MDTs using different sets of base-level classifiers and their straightforward comparison with stacking methods that use different meta-level classifiers.

Furthermore, we performed an extensive experimental comparison of stacking with MDTs to other ensemble methods: to bagging and boosting of decision trees, to voting and to three versions of stacking: with ordinary decision trees (J4.8), with naive Bayes (NB) and with multi-response linear regression (MLR).

Stacking with MDTs performs better than bagging an boosting of decision trees which are the state of the art methods for learning ensembles of classifiers. The previous study of MDTs [11] showed that meta decision trees preform better than voting and stacking with ordinary decision trees. Our study confirms this findings and proves that MDTs are independent of a specific implementation (we used their re-implementation in Java programming language) and independent of the set of base-level classifiers (we used a different number of different classifiers). Performance comparison between stacking with MDTs and stacking with naive Bayes shows that naive Bayes algorithm can not be successfully used as meta-level classifier for stacking.

Finally, we compared performance of stacking with MDTs and stacking with multi-response linear regression (MLR). The latter slightly outperforms MDTs (a 4% relative improvement in accuracy). Stacking with MDTs performs comparably while using less information (only aggregate data on the class probability distribution and not the actual class probability distribution). These attributes are domain independent once we fix the set of base-level classifiers and therefore induced MDTs are also domain independent. Another strong side of MDTs is their understandability as they provide information about areas of expertise for each base-level classifier. This information too is domain independent and can, in principle, be transferred to other domains.

There are several possible directions for further work. Since MDTs are, in principle, transferable across different data sets, we can induce them using examples originating from several different data sets. Preliminary investigations of this approach has been already presented in [12]. However, extensive experimental evaluation of this approach is yet to be done. Another possibility is building MDTs with bagged and boosted ordinary decision trees as base-level classifiers. Since the performance of bagged and boosted ordinary decision trees is better than plain ordinary decision trees, better overall performance of MDTs can be expected. Performance of MDTs with other sets of base-level classifiers should be also explored. This was one of the main reasons for the integration of MDTs in the Weka data mining suite, as it provides a large set of various classifiers that can be used for this purpose.

References

1. Aha, D. and D. Kibler (1991) Instance-based learning algorithms. In *Machine Learning*, 6: 37–66.
2. Breiman, L. (1996) Bagging Predictors. *Machine Learning* 24(2): 123–140.
3. Dietterich, T. G. (1997) Machine-Learning Research: Four Current Directions. *AI Magazine* 18(4): 97–136.
4. Freund, Y. and Schapire, R. E. (1996) Experiments with a New Boosting Algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148-156. Morgan Kaufmann.
5. Gama, J. (1998) Combining Classifiers by Constructive Induction. In *Proceedings of the Ninth European Conference on Machine Learning.*
6. John, G. H. and Langley, P. (1995) Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338-345. Morgan Kaufmann.
7. Merz, C. J. (1999) Using Correspondence Analysis to Combine Classifiers. *Machine Learning* 36(1/2): 33–58. Kluwer Academic Publishers.
8. Murphy, P. M. and Aha, D. W. (1994) *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.
9. Quinlan, J. R. (1993) *C4.5: Programs for Machine Learning.* Morgan Kaufmann.
10. Ting, K. M. and Witten, I. H. (1999) Issues in Stacked Generalization. In *Journal of Artificial Intelligence Research*, 10: 271–289.
11. Todorovski, L. and Džeroski, S. (2000) Combining multiple models with meta decision trees. In *Proceedings of the Fourth European Conference on Principles of Data Mining and Knowledge Discovery*: 54–64.
12. Todorovski, L. and Džeroski, S. (2000) Combining two aspects of meta-learning with heterogeneous meta decision trees. In *Proceedings of the Fifth International Workshop on Multistrategy Learning*: 221–232.
13. Witten, I. H. and Frank, E. (1999) *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann.
14. Wolpert, D. (1992) Stacked Generalization. *Neural Networks* 5(2): 241–260.