Wrapping Boosters against Noise

Bernhard Pfahringer, Geoffrey Holmes, and Gabi Schmidberger

University of Waikato, Hamilton, New Zealand,
{bernhard, geoff, gabi}@cs.waikato.ac.nz,
WWW home page: http://www.cs.waikato.ac.nz/~ml

Abstract. Wrappers have recently been used to obtain parameter optimizations for learning algorithms. In this paper we investigate the use of a wrapper for estimating the correct number of boosting ensembles in the presence of class noise. Contrary to the naive approach that would be quadratic in the number of boosting iterations, the incremental algorithm described is linear.
Additionally, directly using the k-sized ensembles generated during k-fold cross-validation search for prediction usually results in further improvements in classification performance. This improvement can be attributed to the reduction of variance due to averaging k ensembles instead of using only one ensemble. Consequently, cross-validation in the way we use it here, termed wrapping, can be viewed as yet another ensemble learner similar in spirit to bagging but also somewhat related to stacking.

Keywords: machine learning.

1 Introduction

Boosting can be viewed as an induction method that sequentially generates a set of classifiers by reweighting the training set in accordance with the performance of each intermediate set of classifiers. Theoretical attempts at explaining boosting's superior performance, based on so-called *margins* [17], would imply the following relationship between predictive performance of an ensemble and its size: given sufficiently expressive base classifiers, in the limit (i.e. the ensemble consists of infinitely many classifiers) each training example will have a margin of 1. This infinite ensemble will also be optimal in terms of predictive error on new test examples.

Obviously, one would expect this relationship to hold only in noise-free cases, and quite a few recent studies (e.g. [3]) have shown boosting's potential for over-fitting noisy data. Consequently, quite a few authors have proposed and investigated various modifications of the original AdaBoostM1 algorithm [5].

Some of these attempts focus on the reweighting policy, which in the original algorithm utilises an exponential function. Modified reweighting policies try to be less aggressive [4, 6]. Usually the modified algorithm includes an additional parameter for regularization, which could, for example, be an estimate of the optimal ensemble size, or the maximal percentage of training examples, that the ensemble is allowed to misclassify. Also, Friedman's additive regression

interpretation of boosting [8] adding a *shrinkage* parameter, can be seen as a counter-measure to exponential fast fitting of the training data. Alternatively, others have attempted to counter noise problems by using some kind of bagging *around* boosting. Whereas [12] directly bags boosted ensembles, MultiBoost [19] utilizes a bagging variant called *wagging* which simulates bagging's sample-with-replacement by poisson-distributed weights.

What all these methods have in common is their *indirect* approach of solving the anticipated problem: all force the user to specify additional parameters. Most of these parameters have an obvious interpretation, so we can expect the user to supply reasonable values. Take the case of BrownBoost as an example: the user is supposed to supply the true noise-level c as a parameter, thus allowing the algorithm to classify $c\%$ of the training examples incorrectly. Thus, we have only shifted the burden of selecting a reasonable ensemble size upfront to estimating another parameter. Still, there is no guarantee that the supplied estimates are effective for a given dataset.

Alternatively, in this paper we investigate a more direct approach: we try to estimate the appropriate size of the boosted ensemble directly by standard cross-validation. We will show how cross-validation can be computed efficiently for boosters, and we will also show that it naturally leads to ensembles of boosters at no extra induction time cost. The next section defines the algorithm. In section 3 we report on experiments involving various boosters and various levels of noise in datasets. Section 4 discusses our findings and in section 5 we draw our final conclusions.

2 Wrapping

Usually, boosting seems to be pretty stable even in the presence of noise: if enough boosting iterations are performed, a boosted ensemble outperforms an unboosted base-level learner most of the time, sometimes by an impressive amount. Even in the presence of noise the behaviour does not seem to deteriorate too much. But judging by the results cited above, boosting could have performed better in cases with noise. Recently, it has been shown that if the optimal Bayes error for some dataset is different from zero, the boosted ensemble will *not* be optimal in the limit, but some initial prefix of the same ensemble will [9].

We try to directly estimate this optimal size of the boosting ensemble by simple cross-validation. This is reminiscent of so-called early stopping in neural network induction (see e.g. [16]), where some portion of the training set is set aside and used as an independent evaluation set for judging whether performance is still improving or not. Standard k-fold cross-validation seems to be a more principled estimator, but of course involves a k-fold higher runtime cost. Trying to optimize parameters by cross-validation is not new, either. Most importantly, it has been formalized and called the *wrapper* approach for feature subset selection [10]. Also, in [7] cross-validation is used to determine the optimal size of an ADTree for a given dataset.

Table 1. Pseudo-code for two ways of estimating the optimal ensemble size: standard cross-validation, which is $O(k * N^2)$, and wrapping - incremental cross-validation - which is $O(k * N)$.

Standard CV

```
func estimateSize(k,data,booster,N)

let bestSize = 0
let minError = 1.0

for T from 1 upto N
   error =
     cvEstimate(k,data,booster,T)
   if (error =< minError)
      minError = error
      bestSize = T
   endif
endfor

return bestSize
```

Wrapping (incremental CV)

```
func Wrap(k,data,booster,N)

let bestSize = 0
let minError = 1.0
let boosters = new booster[k]

for i from 1 upto k
   boosters[i].initCV(data,i,k)
endfor

for T from 1 upto N
   let error = 0.0
   for i from 1 upto k
      boosters[k].iterate()
      error += boosters[k].estimate()
   endfor
   error = error/k
   if (error < minError)
      minError = error
      bestSize = T
   endif
endfor

return bestSize
```

Simple-minded application of cross-validation is hampered by excessive runtime needs. The issue here is not the multiplication due to k folds being used, but the fact that using simple-minded cross-validation for determining the right ensemble size shows quadratic behaviour in the number of calls to the underlying base-level learner: one call for an ensemble of size one, two calls for an ensemble of size two, and so on, yielding a total of $k * (n-1) * n/2$ calls for estimating all ensemble sizes from 1 up to n using k-fold cross-validation. This will clearly be a prohibitive cost for most datasets.

Luckily, there is a simple remedy available, due to the additive nature of boosting ensembles: for a given set of examples, the ensemble of size m is the union of the ensemble of size $m-1$ plus one more base-level classifier. Both ensembles use exactly the same $m-1$ classifiers. So the smart way to implement cross-validation is to do it incrementally, simply adding one base-level classifier after the other, interleaving these steps with performance estimation on the respective test-fold. Thus we can reduce the complexity of our estimation down

Table 2. Pseudo-code for the ensemble-returning variant of wrapping.

$W_{ensemble}$

```
func WrapEnsemble(k,data,booster,N)

let minError = 1.0
let boosters = new booster[k]

for i from 1 upto k
   boosters[i].initCV(data,i,k)
endfor

let bestBoosters = boosters.clone()    // <== CHANGED

for T from 1 upto N
   let error = 0.0
   for i from 1 upto k
      boosters[k].iterate()
      error += boosters[k].estimate()
   endfor
   error = error/k
   if (error < minError)
      minError = error
      bestBoosters = boosters.clone()  // <== CHANGED
   endif
endfor

return bestBoosters                     // <== CHANGED
```

to $k * n$ calls of the base-level learner, i.e. a linear number of calls. Therefore this cost is identical to the cost of just *one* k-fold cross-validation for the maximal ensemble size n, meaning that we get all the other estimates for sizes 1,2, up to $n - 1$ at *no additional cost*. Also, bagging k times a boosted ensemble of size n would involve exactly the same cost.

We will call this improved implementation of cross-validation *wrapping*. Basically, we are estimating a single integer parameter in the range from 1 to N, where N is user-specified. This improved version is applicable whenever the algorithm in question is incremental in that parameter, which is obviously true for boosted ensembles due to their additive nature.

Pseudo-code comparing standard cross-validation to its incremental variant termed wrapping is given in Table 1. We assume that incremental boosters implement an interface that supports at least the following operations[1]:

[1] Of course, in practise the interface may also include additional functions for bookkeeping, cleanup, general outputting, and so on.

- `initCV(data,i,k)`: which initializes the data-structures of the respective boosting algorithm, as well as separates the training data into the ith-fold for later testing and the remaining data as the ith-fold training set (it is sufficient to just keep the index and a local boosting weight for each training example, full copies are not required here).
- `iterate()`: which performs one boosting iteration, e.g. adding the next best test to an ADTree, or adding another C4.5 generated tree to an AdaBoosted C4.5 ensemble. Each iteration will only use its ith-fold training data subset for induction.
- `estimate()`: which returns an estimate of the predictive error rate of the ensemble at the current size, using the previously set-aside ith-fold test set.

Additionally, we can further improve the utility of wrapping in the following way: with the above scheme we first estimate a good size, and then induce one ensemble of exactly that size using *all* of the training data. Alternatively, quite similar to bagging [2], we can also just directly use these k ensembles of optimal size m as computed during cross-validation, yielding a $k * m$ size ensemble reminiscent of a bagged boosting ensemble. In that case we don't even have to perform the final induction step over all the training data. This variant is depicted in Table 2. As an alternative implementation, this variant might extract the best-sized ensemble for each fold separately, i.e. estimate the best size for each fold independently from all other folds. Thus the ensembles of each fold might vary in size. We have experimented with this alternative variant as well, but found that the estimates computed in such a manner were a lot more unstable, consequently causing inferior predictive behaviour.

So, in summary, wrapping (using k-fold cross-validation) allows us to both estimate the best size m for a boosted ensemble as well as compute a k-sized ensemble of such m-sized boosted ensembles, all in one go. Best of all, the total runtime cost of wrapping is about the same as that of k-times bagging the booster to the respective maximal size n.

In the next section we will investigate the utility of wrapping in terms of predictive error rates.

3 Experiments

This section compares the performance of three different boosting algorithms to their bagged and wrapped versions respectively. The methods are only compared for predictive error rates. Runtimes are as expected, i.e. all bagged and wrapped versions consume about ten times as much runtime due to the additional size-10 layer of computation.

The datasets used and their properties are listed in Table 3. All these sixteen datasets are taken from the UCI repository [1]. The datasets were evaluated using five times ten-fold cross validation. All datasets are two-class problems only, as some of the boosters we use are limited to such problems (currently). The noisy variants of these datasets were generated as follows: as we only deal

with class-noise here, X% of the examples were chosen at random and their class-value was *flipped*. This noisification was done in a preprocessing step prior to experimentation, i.e. both training and testing was done on noisy data. This specific noise model (which has been used previously, e.g. in [14]) was chosen because of both its simplicity and its guaranteed noise levels: if X% is specified, exactly X% of all examples will be given a new, different class label.

Table 3. Datasets used for the experiments

Dataset	Instances	Missing values (%)	Numeric attributes	Nominal attributes
breast-cancer	286	0.3	0	9
breast-wisc	699	0.2	9	0
cleveland	303	0.2	6	7
credit	690	0.6	6	9
diabetes	768	0.0	8	0
hepatitis	155	5.4	6	13
hypothyroid	3772	5.4	7	22
ionosphere	351	0.0	34	0
kr-vs-kp	3196	0.0	0	36
labor	57	33.6	8	8
promoters	106	0.0	0	57
sick-euthyroid	3163	6.5	7	18
sonar	208	0.0	60	0
splice	3190	0.0	0	61
vote	435	5.3	0	16
vote1	435	5.5	0	15

To investigate the sensitivity of bagging and wrapping with respect to the underlying booster, we have conducted experiments with three different boosters:

- ADTree (our WEKA version of it [13]) using randomized search and a maximal ensemble size of 100.
- AdaBoostM1 over C4.5 (in their respective WEKA incarnations) with a maximal ensemble size of 10.
- An ADTree variant that triples the size of the tree at each iteration, with a maximal ensemble size of 8.

Unfortunately, it is somewhat tricky to depict all the variations along all axes available for comparison: noise-levels, algorithms, datasets. Therefore we will only give exemplary full tables of results for one base algorithm, namely for ADTree induction, for noise-free data and data with 30% class noise (the extreme cases), and only summary tables for everything else. So, Tables 4 and 5 depict predictive error rates (all tabulated results are considered significant if the difference between two pairs is statistically significant at the 1% level according to a paired two-sided t-test) for the following four versions of ADTree induction:

Table 4. Predictive error, no noise. The best entry in each line is set in boldface, a prefix star marks values that are significantly different from the value in the first column.

Dataset	ADTree	Bagging	Wrapping	$W_{ensemble}$
BREAST-CANCER	31.59	* 28.57	* 26.02	* **25.32**
BREAST-W	3.83	* **3.49**	* 4.32	3.63
CLEVE	21.78	* 17.15	* 17.28	* **16.03**
CREDIT-A	15.10	* **13.22**	15.68	15.07
CREDIT-G	25.50	* **23.58**	26.62	24.60
DIABETES	26.22	* **24.55**	26.45	* 24.58
HEART-STATLOG	20.30	* 18.00	* **17.11**	* 17.33
HEPATITIS	18.45	* **17.27**	17.53	18.09
IONOSPHERE	8.25	* 7.46	8.43	* **7.40**
KR-VS-KP	0.86	0.79	0.84	* **0.73**
LABOR	12.33	**10.47**	13.27	12.33
PROMOTERS	**6.76**	7.71	* 9.22	5.84
SICK	1.13	* 1.44	* 1.37	**1.11**
SONAR	13.66	14.50	* 16.64	**12.86**
VOTE	4.05	**3.96**	4.32	4.28
VOTE1	9.33	* 8.60	9.43	* **8.43**

- ADTree: using randomized search boosted 100 times. In ADTrees one boosting iteration adds exactly one test to the current tree.
- Bagged ADTrees: generate an ensemble of 10 ADTrees, each of size 100, by means of bagging.
- Wrapped ADTree: use wrapping to determine the optimal ADTree size of up to 100 tests, then induce a single ADTree of that size using the full training set.
- Wrapped ADTree ensemble: use wrapping to determine the optimal ADTree size of up to 100 tests, but instead of consequently inducing another tree, simply use the ensemble of the 10 trees of optimal size generated during cross-validation as the final ensemble.

Table 6 depicts the number of significant wins, draws, and losses over all noise levels and base-learners in a pair-wise manner for the two pairs we think are the most reasonable pairwise competitors: the sole booster versus its simple wrapped form, as well as the bagged booster versus the wrapped ensemble. Finally, Table 7 depicts the number of significant wins, draws, and losses for all pairwise combinations.

We can summarize all the figures of these tables into the following qualitative findings:

- Wrapping seems to be able to choose a reasonable size for the underlying boosting algorithm, as it rarely performs significantly worse than the booster itself.

Table 5. Predictive error, 30% noise. See Table 4 for more explanation.

Dataset	ADTree	Bagging	Wrapping	$W_{ensemble}$
BREAST-CANCER	48.82	* 45.83	* **44.34**	* 44.69
BREAST-W	36.71	* 35.34	* 34.43	* **33.71**
CLEVE	43.87	42.08	44.04	* **41.37**
CREDIT-A	40.03	* 37.91	* 37.42	* **36.20**
CREDIT-G	44.78	43.72	**43.40**	* **43.40**
DIABETES	45.65	**44.61**	45.42	45.03
HEART-STATLOG	47.85	* 45.04	* 41.56	* **40.96**
HEPATITIS	36.80	* 34.12	35.13	* **33.24**
IONOSPHERE	44.45	* 40.97	* 41.31	* **39.84**
KR-VS-KP	33.63	* 32.44	* 31.97	* **31.91**
LABOR	50.73	**49.60**	52.20	51.53
PROMOTERS	52.44	**51.36**	51.45	52.60
SICK	33.92	* 32.51	* 31.36	* **31.14**
SONAR	37.89	38.66	* 40.26	**37.56**
VOTE	38.20	* 35.77	* 36.28	* **34.75**
VOTE1	36.68	* 35.13	* 30.61	* **30.43**

– Bagging also improves performance over just boosting most of the time, and it seems to perform better than simple wrapping.
– Wrapped ensemble performs as well as Bagging at low noise levels, and even better at higher noise levels.

Interestingly, there is also the odd dataset where boosting simply outperforms every method, even in the presence of noise. We suspect that this behaviour will occur mostly in situations were the available training set is actually too small. As it has been shown in [18], usual learning curves are pretty steep initially up to a point where they finally level out asymptotically to the best value a specific algorithm can achieve for a particular dataset. Now if the size of the given training set lies within this first steep region of the learning curve, a few additional examples can make a big difference. So bagging, which on average only includes about two-thirds of the training set in each bag, may be disadvantaged. Similarly, ten-fold cross-validation only uses 90% of the training set for inducing a classifier for each fold, so it too may be disadvantaged, but to a lesser degree. Still, cross-validation seems to exhibit an over-fitting tendency for smaller training sets.

We have repeated the same experimental setup (original algorithm, bagged version, wrapped version, wrapped ensemble) for two other boosting algorithms as well: AdaBoostM1 over C4.5, as well as a variant of ADTree induction, where instead of choosing the globally best test at each boosting iteration all locally (at each prediction node) best tests are added, thus tripling the size of the tree at each iteration. Consequently, we have limited the total number of boosting iterations to 8 for this variant, and set it to 10 for AdaBoostM1, which seems to be a reasonable value reported for AdaBoostM1 over C4.5 induction [15].

Table 6. Significant wins, draws, and significant losses for various pair-wise comparisons at various noise-levels. An entry "i-j-k" means that the first algorithm wins significantly i times, draws j times, and significantly looses k times against the second algorithm.

Noise	ADTree vs. Wrapping	Bagging vs. $W_{ensemble}$
00%	4-9-3	3-8-5
10%	0-10-6	2-12-2
20%	0-4-12	2-7-7
30%	1-6-9	0-10-6
	AdaBoost vs. Wrapping	Bagging vs. $W_{ensemble}$
00%	3-9-4	3-11-2
10%	2-6-8	0-5-11
20%	1-8-7	0-14-2
30%	1-9-6	0-13-3
	tADTree vs. Wrapping	Bagging vs. $W_{ensemble}$
00%	3-9-4	5-5-6
10%	1-7-8	4-7-5
20%	0-8-8	1-6-9
30%	0-6-10	0-8-8

4 Discussion

In this section we discuss two further opportunities offered by wrapping. First, wrapping allows for a more interactive approach to ensemble induction. As error estimates are computed sequentially for increasing ensemble sizes, these estimates can be displayed or graphed online, providing immediate feedback. This allows a user to immediately withdraw from investigating larger ensembles once the error estimates are either good enough or seem to have reached a plateau. Such interaction is valuable in exploratory data analysis.

Second, the additive nature of both wrapping and bagging also allows for further compression of ensembles, provided the underlying base-level learner itself is also additive. This is not the case for Adaboost in general, but it is certainly the case for ADTrees. Ensembles of ADTrees can be merged into a single ADTree, which usually reduces the total size by 20 to 30% (we compare the total number of prediction nodes in a wrapped ensemble of ADTrees to the total number of prediction nodes in the equivalent single merged ADTree). We are currently looking into more sophisticated ways of merging, thus hopefully compressing ensembles even further.

Regarding the apparent success of wrapped ensembles over bagging in high noise cases, an explanation of this fact would be most welcome, as both methods seem to be quite similar. Wrapping seems to enjoy a better variance reduction than bagging under these circumstances. At least, some experimental bias plus variance decompositions that we have computed in the same way as described in [11] seem to indicate that. On the other hand, we are reluctant to put too much

trust in these results, as their overall error sums seem to be of high variance, varying considerably with the specific split chosen for estimation.

5 Conclusions

Our empirical investigation of wrapping seems to indicate that wrapping is a viable (and efficient) alternative to bagging boosters, especially when we suspect considerable levels of class-noise in our data. Simple wrapping allows us to choose an appropriate size, and the wrapped ensemble variant looks even more promising: at zero noise they are equivalent to bagged ensembles, and at higher noise levels they significantly outperform bagged ensembles, and their induction times are about equal. So consequently, wrapped ensembles provide an effective and efficient safeguard for boosters against noise.

In future work we hope to compare wrapping with some of the more sophisticated regularization approaches we have mentioned in the introduction, especially a comparison with BrownBoost and MultiBoost should be most interesting. Furthermore, we want to concentrate more on larger KDD-class datasets. Such experiments might be able to further strengthen our hypothesis that cross-validation is prone to overfitting small datasets. Additionally, we want to investigate the potential of merging, especially as a means of reducing total ensemble sizes, thus hopefully improving the comprehensibility of these merged ensembles. Furthermore, we are researching the applicability of wrapping to general complexity class estimation, a problem that is obviously not limited to boosting algorithms alone. Perhaps the most valuable achievement would be to find a way of replacing the currently user-specified maximal ensemble-size by some principled estimation. Unfortunately, our attempts in that direction have not been successful so far. We believe that something better than just presetting the maximal ensemble size to some ridiculously high value must exist.

A WRAPPER1D class as well as an appropriate interface for iterative classifiers, and a few exemplar iterative classifiers will all be included in the next version of the WEKA machine learning workbench [20], which is available[2] under the Gnu Public License.

References

1. Blake, C. L., Keogh, E., Merz, C.J.: UCI Repository of Machine Learning Data-Bases. Irvine, CA: University of California, Department of Information and Computer Science. [http://www.ics.uci.edu/ mlearn/MLRepository.html] (1998).
2. Breiman L.: Bagging Predictors, Machine Learning, 24(2), 1996.
3. Dietterich T.G.: An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, Machine Learning, 40(2), 139-158, 2000.

[2] WEKA can be downloaded from `http://www.cs.waikato.ac.nz/~ml`

4. Domingo C., Watanabe O.: MadaBoost: A Modification of AdaBoost, in Proceedings of the Thirteenth Annual Conference on Computational Learning Theory", Morgan Kaufmann, San Francisco, 2000.
5. Freund Y., Schapire R.E.: Experiments with a New Boosting Algorithm, in Saitta L.(ed.), Proceedings of the 13th International Conference on Machine Learning (ICML'96), Morgan Kaufmann, Los Altos/Palo Alto/San Francisco, pp.148-156, 1996.
6. Freund, Y.: An adaptive version of the boost by majority algorithm. In Proceedings of the Twelfth Annual Conference on Computational Learning Theory, Morgan Kaufmann, San Francisco, 1999.
7. Freund, Y., Mason, L.: The alternating decision tree learning algorithm. Proceedings of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, (1999) 124-133.
8. Friedman J., Hastie T., Tibshirani R.: Additive logistic regression: a statistical view of boosting. Technical Report, 1998.
9. Jiang W.: Some theoretical aspects of boosting in the presence of noisy data, in Proceedings: The Eighteenth International Conference on Machine Learning (ICML-2001), Morgan Kaufmann, 2001.
10. Kohavi R., John G.H.: Wrappers for feature subset selection, in Subramanian D., et al.(eds.), Special Issue on Relevance, Artificial Intelligence, 97(1-2), 273-324, 1997.
11. Kohavi R., Wolpert D.: Bias plus variance decomposition for zero-one loss functions, in Proc. of the Thirteenth International Machine Learning Conference, Morgan Kaufmann, 1996.
12. Pfahringer B.: Winning the KDD99 Classification Cup: Bagged Boosting, SIGKDD explorations, 1(2), 65-66, 2000.
13. Pfahringer B., Holmes G., Kirkby R.: Optimizing ADTrees, Proceedings of the Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2001.
14. Quinlan J.R.: The Minimum Description Length Principle and Categorical Theories, in Cohen W.W. and Hirsh H.(eds.), Proceedings of the 11th International Machine Learning Conference (ICML'94), Rutgers University, Newark, NJ, pp.233-241, 1994.
15. Quinlan J.R.: Bagging, Boosting, and C4.5, in Proceedings of the 13th National Conference on Artificial Intelligence, AAAI Press/MIT Press, Cambridge/Menlo Park, 1996.
16. Raudys S., Cibas T.: Regularization by Early Stopping in Single Layer Perceptron Training, in Malsburg C.van der, et al.(eds.), Artificial Neural Networks - ICANN 96, Springer, pp.77-82, 1996.
17. R.E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee.: Boosting the margin: A new explanation for the effectiveness of voting methods. The Annals of Statistics, 26(5):1651-1686, October 1998.
18. Ting K.M., Low B.T.: Model Combination in the Multiple-Data-Batches Scenario, in Someren M.van and Widmer G.(eds.), Machine Learning: ECML-97, Springer, Berlin/Heidelberg/New York/Tokyo, pp.250-265, 1997.
19. Webb G.I.: MultiBoosting: A Technique for Combining Boosting and Wagging, Machine Learning, 40(2), 159-196, 2000.
20. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, San Francisco, California (2000).

Table 7. Significant wins, draws, and significant losses for all four noise-levels and the following four algorithms: ADTree, bagged ADTree, wrapped ADTree, and a wrapped ensemble over ADTree. An entry "i-j-k" means that the row algorithm wins significantly i times, draws j times, and significantly looses k times against the column algorithm.

ADTree							
no noise	Bagging	Wrapping	$W_{ensemble}$	10% noise	Bagging	Wrapping	$W_{ensemble}$
ADTree	1-5-10	4-9-3	0-9-7	ADTree	0-3-13	0-10-6	0-3-13
Bagging		8-7-1	3-8-5	Bagging		8-6-2	2-12-2
Wrapping			0-6-10	Wrapping			0-5-11
20% noise	Bagging	Wrapping	$W_{ensemble}$	30% noise	Bagging	Wrapping	$W_{ensemble}$
ADTree	0-2-14	0-4-12	0-1-15	ADTree	0-6-10	1-6-9	0-4-12
Bagging		3-9-4	2-7-7	Bagging		0-11-5	0-10-6
Wrapping			0-7-9	Wrapping			0-10-6

AdaBoost							
no noise	Bagging	Wrapping	$W_{ensemble}$	10% noise	Bagging	Wrapping	$W_{ensemble}$
AdaBoost	2-2-12	3-9-4	1-4-11	AdaBoost	0-2-14	2-6-8	0-5-11
Bagging		8-6-2	3-11-2	Bagging		6-9-1	1-14-1
Wrapping			1-8-7	Wrapping			0-12-4
20% noise	Bagging	Wrapping	$W_{ensemble}$	30% noise	Bagging	Wrapping	$W_{ensemble}$
AdaBoost	0-6-10	1-8-7	0-6-10	AdaBoost	0-15-1	1-9-6	0-8-8
Bagging		5-9-2	0-14-2	Bagging		2-10-4	0-13-3
Wrapping			0-13-3	Wrapping			0-15-1

tADTree							
no noise	Bagging	Wrapping	$W_{ensemble}$	10% noise	Bagging	Wrapping	$W_{ensemble}$
tADTree	2-4-10	3-9-4	0-7-9	tADTree	0-3-13	1-7-8	0-5-11
Bagging		7-8-1	5-5-6	Bagging		6-7-3	4-7-5
Wrapping			0-8-8	Wrapping			0-8-8
20% noise	Bagging	Wrapping	$W_{ensemble}$	30% noise	Bagging	Wrapping	$W_{ensemble}$
tADTree	0-6-10	0-8-8	0-4-12	tADTree	0-12-4	0-6-10	0-5-11
Bagging		1-11-4	1-6-9	Bagging		1-7-8	0-8-8
Wrapping			0-10-6	Wrapping			0-14-2