Ensemble Methods for Noise Elimination in Classification Problems

Sofie Verbaeten and Anneleen Van Assche

Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200 A, B-3001 Heverlee, Belgium
{sofie.verbaeten,anneleen.vanassche}@cs.kuleuven.ac.be

Abstract. Ensemble methods combine a set of classifiers to construct a new classifier that is (often) more accurate than any of its component classifiers. In this paper, we use ensemble methods to identify noisy training examples. More precisely, we consider the problem of mislabeled training examples in classification tasks, and address this problem by pre-processing the training set, i.e. by identifying and removing outliers from the training set. We study a number of filter techniques that are based on well-known ensemble methods like cross-validated committees, bagging and boosting. We evaluate these techniques in an Inductive Logic Programming setting and use a first order decision tree algorithm to construct the ensembles.

1 Introduction

In many applications of machine learning the data to learn from is imperfect. Different kinds of imperfect information exist, and several classifications are given in the literature (see e.g. [9]). In this paper, we consider the problem of noise or random errors in training examples.

One of the problems created by learning from noisy data is overfitting, that is, the induction of an overly specific hypothesis which fits the (noisy) training data well but performs poor on the entire distribution of examples. Classical noise-handling mechanisms modify the learning algorithm itself to make it more noise-tolerant. Another approach, which we explore in this paper, is to pre-process the input data before learning. This approach consists of filtering the training examples (hopefully removing the noisy examples), and applying a learning algorithm on the reduced training set. As pointed out in [7], this separation of noise detection and hypothesis formation has the advantage that noisy examples do not influence the hypothesis construction, making the induced hypothesis less complex and more accurate.

Many of the methods for filtering training data are in fact removing outliers from the training data. An outlier is a case that does not follow the same model as the rest of the data[1]. For instance in [3], the basic idea is to use a set of

[1] Note that, as such, an outlier does not only include erroneous data but also surprising correct data.

T. Windeatt and F. Roli (Eds.): MCS 2003, LNCS 2709, pp. 317–325, 2003.

classifiers (induced by a number of possibly different learning methods) formed from part of the training data to test whether instances in the remaining part of the training data are mislabeled. By taking a consensus or majority vote of these classifiers, it is decided whether or not to remove a particular instance. The noise detection algorithm of [7] is based on the observation that the elimination of noisy examples, in contrast to the elimination of examples for which the target theory is correct, reduces the CLCH value of the training set (CLCH stands for the Complexity of the Least Complex correct Hypothesis). In [8] robust decision trees are presented. Robust decision trees take the idea of pruning one step further: training examples which are misclassified by the pruned tree are removed from the training set and the tree is rebuilt using this reduced set. This process is repeated until no more training examples are removed. In [12], we presented filter techniques for Inductive Logic Programming (ILP) that are based on the idea of [3]. We also applied the robust decision tree technique of [8] to the ILP setting. We already obtained some good results with the filters proposed in [12].

In this paper, we further explore a number of other, new techniques. We propose filter techniques that are based on well-known ensemble methods [4], namely cross-validated committees, bagging and boosting. We present two approaches: (1) filtering based on (unweighted) voting of classifiers that are built on different subsets of the training set (obtained by either cross-validation or bagging), (2) filtering based on removing training examples that obtain high weights in the boosting process. We introduce these filter techniques in the next section, and evaluate them in an ILP setting in section 3. We conclude and discuss topics for future research in section 4.

2 Filter Algorithms

2.1 Base Classification Algorithm

The filters that are presented below make use of a learning algorithm for classification. With L we denote this base classification algorithm.

In our experiments, we evaluate the different filters in an ILP setting, and use Tilde [1] as the base learning algorithm L. Tilde (Top-down Induction of Logical Decision Trees) is an ILP extension of the C4.5 decision tree algorithm [11]. Instead of using attribute-value tests in the nodes of the tree, logical queries (which may contain logical variables) are used. The test to select the best query in each node is information gain ratio. After a tree is constructed, a post-pruning algorithm, based on an estimate of the error on unseen cases, is used.

2.2 Voting Filters

Voting filters are (as many other filter methods) based on the idea of removing outliers from a training set: an instance is removed if it can not be classified correctly by all, or the majority of, the classifiers built on parts of the training

set. A motivation for using ensembles for filtering is pointed out in [3]: when we assume that some instances in the data have been mislabeled and that the label errors are independent of the particular model being fit to the data, collecting information from different models will provide a better method for detecting mislabeled instances than collecting information from a single model. As noted in many articles (see e.g. [4]), constructing ensembles of classifiers by manipulating the training examples works especially well for unstable learning algorithms. Decision tree algorithms, like Tilde, are unstable. We expect that ensembles of decision trees will act well as a filter for noisy data sets.

The general scheme of our voting filters is as follows:

1. L induces n classifiers on different subsets of the training set,
2. these n classifiers give labels to every example in the training set,
3. the filter compares the original class of each example with the n labels it has, and decides whether or not to remove the example.

A variation of instances of this general scheme exists depending on the way these n classifiers are induced, the value of n and the decision procedure in step 3.

Concerning step 3, we consider two possibilities: (1) a *consensus* filter (C filter), where a training example is removed only if all the n labels it has differ from its class; (2) a *majority vote* filter (M filter), where a training example is removed if the majority of the labels it has differ from its class.

Concerning step 1, we present two approaches for building these n classifiers. In the first approach, the training set is partitioned in n subsets of (approximately) equal size. L is trained n times, each time leaving out one of the subsets from the training set. This results in n classifiers. Such a filter is called a *cross-validated committees* filter (X filter). In the second approach, n bootstrap replicates are taken from the training set, and L learns on these n sets. Such a filter is called a *bagging* filter (Ba filter). In [5], a motivation is found for using bagging as a filter. In that paper, it is experimentally shown that bagged C4.5 gains advantage over C4.5 when noise is added to the training sets. More precisely, it is observed that for most data sets, noise improves the diversity of bagging, which permits it to perform better.

We performed experiments with cross-validated committees - consensus (XC) filters, cross-validated committees - majority vote (XM) filters, bagging consensus (BaC) and bagging majority vote (BaM) filters. The parameter n was set to 5, 9, 15, and 25.

2.3 Boosting Filters

Boosting is known to perform poorly with respect to noise. According to [5] a plausible explanation for the poor response of boosting to noise is that mislabeled training examples will tend to receive very high weights in the boosting process. Hence, after a few iterations, most of the training examples with high weights will be mislabeled examples. This gives a good motivation to use boosting as a noise filter.

The idea is to use Adaboost [6] and to remove, after a number of rounds (n), the examples with the highest weights. Since the filter has no idea what the exact percentage of noise is, we chose to give this as input to the filter[2]. Note that there might be several examples with the same weight (especially in the first rounds of the boosting process), so the filter might remove more examples than is given as input[3].

We tested the boosting (Bo) filter with 3, 5, 10 and 17 rounds.

3 Experiments

We evaluate the different filters in an ILP setting. As opposed to propositional or attribute-value learning systems that use a single table to represent the data set, ILP systems use a first order representation. This makes ILP very suitable for dealing with complex data structures.

We first describe the data sets that we used in the experiments. Then we explain how the experiments were carried out, and finally we discuss the results. For more details, we refer to [13].

3.1 Data Sets and Noise Introduction

We want to evaluate how well the different filter techniques perform on data sets with different amounts of classification noise. We therefore considered noise-free ILP data sets, and artificially introduced different levels of classification noise.

We considered the following noise-free data sets: an (artificial) Bongard data set [2] (392 examples), three (artificial) eastbound/westbound trains data sets [10] (200, 400, and 800 examples), and a (non-artificial) KRK data set for learning illegal positions in a chess endgame [9] (200 examples). These are all 2-class problems.

We introduced different levels of classification noise in the data sets. A noise level of $x\%$ means that for a randomly chosen subset of $x\%$ of the training examples, the class-value of these examples was flipped[4]. We introduced noise levels of 0%, 5%, 10%, 15%, 20%, 25%, 30%, 35%, and 40%.

3.2 Experimental Method

In order to obtain a more reliable estimate of the performance of the filters, all experiments were carried out in 10-fold cross-validation and the results were averaged over the 10 folds. For each of the 10 runs, the data set was divided in a training set (9 parts) and a test set (remaining 1 part). The training set was then corrupted by introducing classification errors. Each of the above described

[2] In a later stage, we will try to estimate this.

[3] Another possibility is to remove less examples than is given as input; we chose the greedy approach here.

[4] Positive examples are made negative and vice versa.

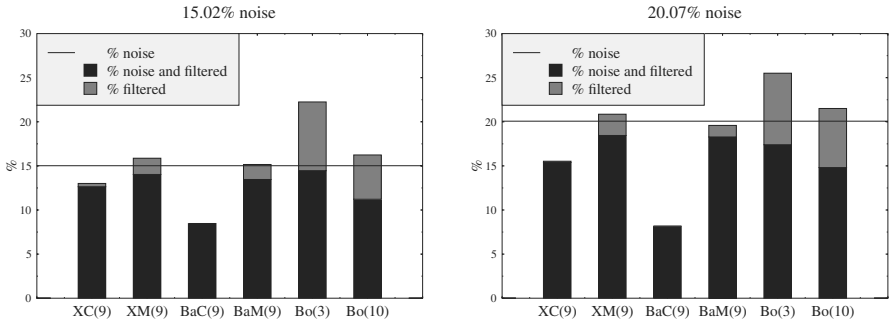

Fig. 1. Filter precision on the Bongard data set.

filter techniques was then run on the (noisy[5]) training set. After filtering the training set, Tilde was used to learn a decision tree on the reduced training set. This decision tree was then validated on the (noise-free) test set. Results were obtained by taking the mean of the results of the 10 runs. For each of the 9 noise levels and each of the 10 runs, we also run Tilde directly on the (unfiltered) training set.

In the next subsections, we report results concerning filter precision, tree size and accuracy.

3.3 Filter Precision

We evaluate the different filters by looking at the percentage of examples that are removed, as well as the percentage of examples that are removed and are actually noisy[6].

In Fig. 1, we show the precision of some of our filters on the Bongard data set with 15% and 20% of noise.

In general, we observe the following. The BaC filters are the most conservative filters: they remove the fewest examples. But, especially for the artificial data sets, these filters are also very precise: almost no correct examples are removed. One might choose to use a BaC filter when data is sparse. Also the XC filters are rather conservative. The XM and BaM filters perform well. On the KRK data set the Ba and X filters are not as precise as on the artificial data sets: relatively more correct examples are also removed. The best results concerning precision are obtained on the Bongard data sets. This can be explained by the fact that Tilde reaches an accuracy of 100% on the noise-free Bongard data set, whereas the other (noise-free) data sets, especially the KRK data set, are harder to learn from.

[5] For each noise level and each of the 10 training sets, the classification errors were introduced only once (in a random way), and the different filter techniques were run on the same noisy training sets.

[6] For the detailed results, including an estimated probability of making type 1 and type 2 errors for all filters, we refer to [13].

In general, the precision of the Bo filters is not very good: a great deal of filtered examples is not noisy. This shows that in the boosting process, also correct examples get high weights.

The influence of the number of classifiers n is as follows. For the XC(n) filters, when there is not too much noise, the higher n, the higher the number of filtered examples, and also the higher the number of filtered and noisy examples. Indeed, when n increases, the training sets on which the n classifiers are built become more similar, and hence the classifiers will obtain a consensus in more cases. This is especially the case when there is not too much noise (up to 20%). For the XM(n) filters, the parameter n does not have much influence.

For the BaC(n) filters, the higher n, the lower the number of filtered examples, and also the lower the number of filtered and noisy examples. This is exactly the opposite behaviour as for the XC(n) filters. This is because with bagging very diverse classifiers are built (especially when there is a lot of noise, see [5]). When the number (n) of such diverse classifiers increases, it becomes more difficult to obtain a consensus, hence less examples will be removed. The BaM(n) filters seem to improve their precision when n is increased, meaning that more noisy examples, and at the same time less correct examples, are removed.

Despite the fact that the exact percentage of noisy examples is given as input to the Bo(n) filters, these filters will remove more examples. This is because there might be examples that have the same weight. Since this is especially the case in the first rounds of the boosting process, we observe that, the higher n, the lower the number of filtered examples. At the same time we observe that, the higher n, the less noisy examples are filtered. When we look at the ratio of noisy examples in the set of examples that are filtered, we notice that, for noise levels up to 15%, this ratio is smaller in the first rounds of the boosting process. For higher noise levels however, the situation is completely reversed: when n is small, there are relatively more noisy examples in the set of filtered examples. This can be explained as follows. If there is a lot of noise in the training set, boosting will first force the learning algorithm to concentrate on these noisy examples. But, by doing this, the correct examples become harder to classify, and will receive higher weights further on in the boosting process. So when n increases, more correct examples will receive high weights and will be filtered.

Finally, by looking at the results on the three trains data sets, we observe that the more training examples we have at our disposal, the more precise the filters are.

3.4 Tree Size

A decision tree that is built from a noisy training set might be overly complex due to overfitting of this set. Therefore, it can be expected that the sizes of the trees induced from a filtered (and hopefully non-noisy) training set are smaller than the sizes of the trees induced from the non-filtered, noisy training set.

In Fig. 2, we show the number of nodes in the decision trees induced from (un)filtered Bongard training sets. More precisely, we report the results for the XM(9), BaM(9) and Bo(10) filters.

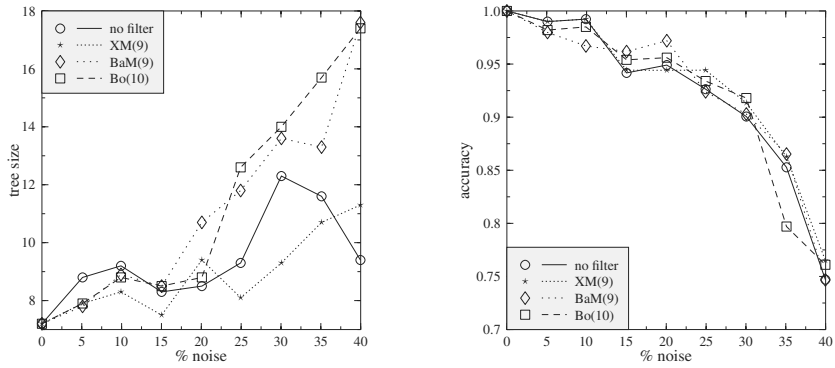

Fig. 2. Results concerning tree size (left) and accuracy (right) on the Bongard data set.

For noise levels up to 15% it is indeed the case that the sizes of the trees induced from a filtered training set are smaller than the sizes of the trees induced from the unfiltered set. For higher noise levels however there is, for many of the cases, no decrease in tree size if a filtered training set is used. One plausible explanation is that, for high noise levels, the filters still leave some amount of noise in the training sets. Also, we should note that Tilde with pruning is used (both in the filter algorithms and for inducing decision trees from the (un)filtered training sets), so the effect of overfitting is already largely reduced.

3.5 Accuracy

Decision trees built on a non-noisy training set will (in general) be more accurate (on a separate test set) than trees induced from a noisy training set. We compare the accuracies of the trees induced from the filtered sets (on the non-noisy test sets) with the accuracies (also on the non-noisy test sets) of the trees induced from the unfiltered, noisy sets.

In Fig. 2, we show our results for the XM(9), BaM(9) and Bo(10) filters on the Bongard data set. For this data set and also for the trains data sets, we observe that for noise levels up to 10%, Tilde still performs well on an unfiltered training set. For higher noise levels, it seems better to first filter the training set. No one filter outperforms the other filters in this respect. For the KRK data set, it is better to also filter the training set for low noise levels.

A hypothesis of interest is whether a majority vote ensemble classifier can be used instead of filtering, or whether the best method is to first filter the training set and then use a majority vote ensemble classifier. This hypothesis was tested in [3]. It was concluded that (for the filters proposed in [3]) a majority vote ensemble classifier can not replace filtering. We did some initial experiments to test this hypothesis in our setting. More experiments are needed to see if this conclusion also holds in our setting.

4 Conclusions and Future Work

We addressed the problem of training sets with mislabeled examples in classification tasks. We proposed a number of filter techniques, based on ensemble methods, for identifying and removing noisy examples. We experimentally evaluated these techniques on noise-free ILP data sets which we artificially corrupted with different levels of classification noise. We reported results concerning filter precision, tree size and accuracy.

Both the BaM and XM filters have a good precision. Surprisingly, the Bo filters did not perform so well. We plan to investigate in more detail how the boosting process can be used/modified to obtain a noise filter. We also plan to evaluate the proposed filters on more data sets.

When the data set is small and the cost of finding new training examples is high, one can choose to use a conservative filter, e.g. a BaC or XC filter. A better solution would be to detect and also correct labelling errors (and thus not removing any example). One way to do this is to present the suspicious data to a human expert and ask what to do with it. Another way is to automatically switch the class labels of the examples which are identified as noise. We will evaluate the performance of such an extension.

Acknowledgements

Sofie Verbaeten is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.- Vlaanderen). Anneleen Van Assche is supported by the GOA/2003/08(B0516) on Inductive Knowledge Bases.

References

1. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.
2. M. Bongard. *Pattern Recognition*. Spartan Books, 1970.
3. C.E. Brodley and M.A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.
4. T.G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems, First International Workshop*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.
5. T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.
6. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In L. Saitta, editor, *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.
7. D. Gamberger, N. Lavrač, and S. Džeroski. Noise detection and elimination in data preprocessing: experiments in medical domains. *Applied Artificial Intelligence*, 14:205–223, 2000.

8. G.H. John. Robust decision trees: Removing outliers from databases. In U.M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 174–179. AAAI Press, 1995.
9. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
10. R.S. Michalski and J.B. Larson. Inductive inference of VL decision rules. Paper presented at Workshop in Pattern-Directed Inference Systems, Hawaii, 1977. SIGART Newsletter, ACM, 63, 38-44.
11. J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann, 1993.
12. S. Verbaeten. Identifying mislabeled training examples in ILP classification problems. In M. Wiering and W. de Back, editors, *Twelfth Dutch-Belgian Conference on Machine Learning*, pages 1–8, 2002.
13. S. Verbaeten and A. Van Assche. Ensemble methods for noise elimination in classification problems. Technical report, Department of Computer Science, K.U.Leuven, Belgium,
http://www.cs.kuleuven.ac.be/publicaties/rapporten/cw/CW358.abs.html, 2003.