

Techniques for Identifying Mislabeled Training Examples in ILP Classification Problems

Sofie Verbaeten
Thomas Cardoen

Report CW 347, September 2002



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Techniques for Identifying Mislabeled Training Examples in ILP Classification Problems

Sofie Verbaeten
Thomas Cardoen

Report CW 347, September 2002

Department of Computer Science, K.U.Leuven

Abstract

We consider the problem of noisy training examples, more precisely mislabeled training examples, in the context of ILP classification problems. We 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 filtering techniques, some of which were proposed in the literature for attribute-value problems. We evaluate these techniques on a Bongard data set, which we artificially corrupt with different levels of classification noise.

CR Subject Classification : I.2.6

1 Introduction

In many applications of machine learning the data to learn from is imperfect. Different kinds of imperfect information exists, and several classifications are given in the literature (see e.g. [2], [11], [13]). In [11] the following types of imperfect data for Inductive Logic Programming tasks are discussed: 1) noise, that is, random errors in training examples and background knowledge, 2) too sparse training examples, from which it is difficult to reliably detect correlations, 3) inappropriate or insufficient background knowledge, and 4) missing argument values in the training examples. As pointed out in [2], one can also have, in contrast to noise or random errors, systematic errors in the data (caused e.g. by incorrect calibration of an item of measuring equipment so that it is reading consistently low). In this paper, we consider the problem of noise or random errors in training examples. We address this problem in the context of ILP classification problems.

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. Many techniques exist which allow ILP algorithms to handle noisy data and to prevent overfitting. Classical noise-handling mechanisms are based on appropriate search heuristics and stopping criteria used in the hypothesis construction, or on some form of post-processing of hypotheses. These techniques modify the learning algorithm itself to make it more noise-tolerant. Another approach 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. We will follow this approach here.

In the context of attribute-value learning there are a number of proposals for identifying and removing noisy examples from the training data. Examples are [4], [7, 6, 9, 8], and [10]. In [4] a technique is presented which detects data with a wrong class label. The idea is to use a number of (possibly different) learning methods to create classifiers that act as a filter for the training data. They present and evaluate a Single Algorithm filter, a Majority Vote filter and a Consensus filter. Although [4] evaluates the technique for the case of attribute-value learning, it is directly applicable to ILP problems. The noise detection algorithm of [7, 6, 9, 8] for attribute-value learning 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). The noise detection algorithm is called the Saturation filter since it employs the CLCH measure to test whether the training set is saturated, i.e. whether, given a selected hypothesis language, the data set contains a sufficient number of examples to induce a stable and reliable target theory. In [9] two combinations of the Saturation filter and the filters proposed in [4] are proposed: the combined Classification-Saturation filter and the Consensus Saturation filter. In [10] Robust decision trees are presented. Robust decision trees take the idea of pruning one step further: training examples which are locally uninformative and harmful, that is, examples which are misclassified by the pruned tree, are also globally uninformative. Therefore, after pruning a decision tree, the training examples which are misclassified should be removed from the training set and the tree needs to be rebuilt using this reduced set. This process is repeated until no more training examples are removed. Although [10] evaluates the Robust technique in the context of C4.5 (the resulting system is called Robust C4.5), it can be directly applied to first order decision trees. Our filtering techniques are based on the ideas in [4] and [10].

Many of the methods for filtering training data are motivated by the technique of removing outliers in regression analysis [18]. An outlier is a case that does not follow the same model as the rest of the data and appears as though it comes from a different probability distribution. As such, an outlier does not only include erroneous data but also surprising correct data. One of the difficulties in removing noisy data is to also remove correct exceptions. One work addressing this problem is [17]. We will not cover this topic here.

In classification problems noise in the training examples can be caused by erroneous argument values and/or erroneous classifications. Quinlan [14] demonstrated that, for high levels of noise, removing noise from attribute information decreases the predictive accuracy of the resulting classifier if the same attribute noise is present when the classifier is subsequently used. This is not the case for noise in the classification of examples. In this paper, we will consider misclassified examples in the training data.

Summarizing, we consider the problem of noisy training examples, more precisely erroneous classifications, in the context of ILP classification problems. We address this problem by pre-processing the training set, more precisely by identifying and removing outliers from the training set. We study a number of filtering techniques, some of which were proposed in the literature for attribute-value problems. We evaluate these techniques on

a Bongard data set, which we artificially corrupt with different levels of classification noise.

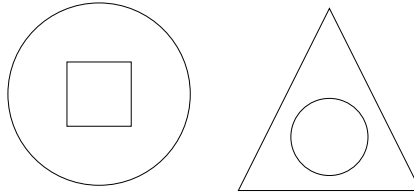
The rest of this paper is structured as follows. In the next section 2, we recall the first order decision tree induction system Tilde on which we base our filtering techniques. Tilde will also be used to validate the filtering techniques. The experiments are run on a Bongard data set, and we will say a word on such data sets in section 2. Next, in section 3, we introduce the different filtering techniques. These techniques will be evaluated in section 4 on a Bongard data set. We conclude and discuss topics of future research in section 5.

2 Tilde and Bongard Sets

Tilde (Top-down Induction of Logical Decision Trees) [1] is an ILP extension of the C4.5 decision tree algorithm. Instead of using attribute-value tests in the nodes of the tree, logical queries are used. As in C4.5, Tilde builds the decision tree in a top-down way, starting with the empty tree and all training examples. In each node, Tilde generates all possible tests and computes an heuristic value, in our experiments information gain ratio [15], for each of these tests. The test which scores best is placed in the node. The examples in the current node are then sorted down the tree: the examples passing the test are propagated to the left, the other examples to the right. The procedure is repeated for the left and right subtree. A node is turned into a leaf when the examples it covers are of a single class. After a tree is constructed, a post-pruning algorithm is used. The post-pruning algorithm that we will use here is the C4.5 post-pruning method which is based on an estimate of the error on unseen cases (see [15]).

Note that the logical queries in the nodes of a first order decision tree may contain logical variables. These variables may be shared among different nodes under the restriction that a variable that is introduced in a node (that is, it does not occur in higher nodes) must not occur in the right branch (the “no”-branch) of that node. The reason for this restriction follows from the semantics of a first order decision tree. A variable that is introduced in a node is existentially quantified within the conjunction of that node. When this conjunction fails (and we thus enter the right subtree) no further reference to that variable is needed.

We next explain on an example how a first order decision tree can be



```

circle(o1).
square(o2).
triangle(o3).
circle(o4).
in(o2,o1).
in(o4,o3).
config(o3,up).

```

Figure 1: Bongard example

used to classify examples. We consider a Bongard problem¹. Bongard data sets consist of configurations of geometrical objects, more precisely triangles, squares and/or circles. The direction in which triangles point can be up or down. An object can be inside another object. A typical Bongard example is shown in Figure 1.

A Bongard data sets consists of a number of these drawings, each classified as either positive or negative. Given a Bongard data set, Tilde will learn a decision tree which predicts the class of unclassified examples. In Figure 2 an example decision tree² is shown.

Given the decision tree of Figure 2, the unclassified example of Figure 1 gets a label as follows. The query *?triangle(A)* succeeds, so we enter the “yes”-branch of the tree. The query *?triangle(A), in(A,B)* fails, we follow the right branch. The next query is *?triangle(A), circle(I)* which succeeds and hence we follow the left branch. Finally, the query *?triangle(A), circle(I), in(I,J)* succeeds and by entering the left branch we get the classification **neg** for this example.

The equivalent Prolog program which can be derived from the decision tree of Figure 2 is shown in Figure 3.

¹We will use a Bongard data set to validate the different filtering techniques.

²This decision tree was induced from the noise-free Bongard data set which was used in our experiments.

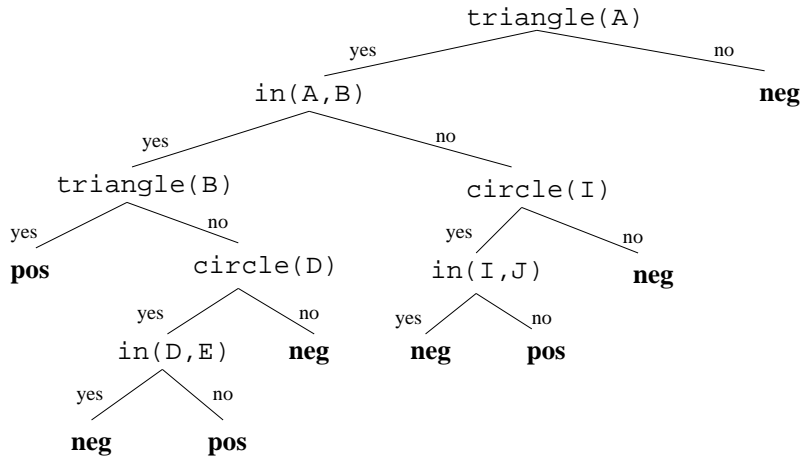


Figure 2: A decision tree for a Bongard problem

```

class(pos) :- triangle(A), in(A,B), triangle(B), !.
class(neg) :- triangle(A), in(A,B), circle(C), in(C,D), !.
class(pos) :- triangle(A), in(A,B), circle(C), !.
class(neg) :- triangle(A), in(A,B), !.
class(neg) :- triangle(A), circle(B), in(B,C), !.
class(pos) :- triangle(A), circle(B), !.
class(neg) :- triangle(A), !.
class(neg) .

```

Figure 3: The equivalent Prolog program

3 Filtering Algorithms

In this section we introduce different ILP filtering techniques. They are based on existing techniques for attribute-value learning. In the next section, we will evaluate these techniques.

3.1 Robust Tilde

The first filtering technique is based on the proposal of [10]. There, a robust version of the decision tree system C4.5 is presented, but in fact it can be applied to any decision tree learner. Here we will apply it to Tilde.

The motivation of the proposal of [10] comes from statistics. One approach to identify outliers in statistics is to look at the measure of effect the data point has on the learned model. Points with high leverage, that is, points with disproportionately high effect on the fitted model, are identified and removed from the data in order to better fit the remaining points. For decision trees, the set of training examples which are misclassified after pruning the tree approximates the set of points with high leverage. Indeed, this set contains the examples which caused a more complex tree to be constructed so that they would be classified correctly. John [10] takes the idea of pruning one step further: examples which are locally uninformative and harmful are also globally uninformative. That is, after pruning the decision tree, the examples which are misclassified should be removed from the training set and the tree needs to be rebuilt using this reduced set. This process is repeated until no more training examples are removed. The robust decision tree algorithm can be described as follows:

1. learn a decision tree from the training set,
2. prune this tree,
3. for each example in the training set, if the pruned tree misclassifies the example, then remove the example from the training set,
4. if no examples are removed from the training set in the previous step, then stop, else go to step 1.

Note that in the case of robust decision trees, the final learning algorithm is the same as the learning algorithm used during filtering. We will apply this technique to Tilde and call the resulting system (filtering and learning on the filtered training set) Robust Tilde. In the next section, Robust Tilde will be abbreviated as “R”.

3.2 Voting Filters

In [4] a general method for filtering training sets with classification noise is proposed. The idea is to use a set of learning algorithms to create classifiers that act as a filter for the training data. More precisely, the general method is as follows:

1. m learning algorithms (called filter algorithms) are chosen,
2. the training data is divided in n non-overlapping parts (n -fold cross-validation),
3. for each of the n parts, the m algorithms are learned on the other $n - 1$ parts,
4. the m resulting classifiers are used to label each instance in the excluded part (in this way each example gets m labels),
5. the filter compares the original class of each example with the m labels it got, and decides whether or not to remove the example,
6. the filtered set of training examples is given as input to the final learning algorithm, the resulting classifier is the end product.

Steps 1 to 5 constitute the filtering procedure. A variation of instances of this general scheme exists depending on the choice (and the number) of filter algorithms, the value of n and the decision procedure in step 5. In [4], different instances of this scheme are studied and empirically evaluated: the Single Algorithm filter ($m = 1$), and the Consensus and Majority Vote filter ($m > 1$). Step 5 in the Single Algorithm is simple: if the class of a training example differs from the (one) label it got, the example is removed. In the Consensus filter, a training example is removed only if all the labels it got (m) differ from its class. And in the Majority Vote filter, a training example is removed if the majority of the labels it got ($m/2$ - or $(m + 1)/2$ in case m is odd - or more) differ from its class.

We next discuss in more detail which Voting filters we will evaluate here.

3.2.1 Single Algorithm Filter

We will consider a Single Algorithm filter with Tilde as filter algorithm. We use three different values for n , namely 2, 5 and 10. In the experiments, our

Single Algorithm filter will be abbreviated as “S(2)” (for $n = 2$), “S(5)” (for $n = 5$), and “S(10)” (for $n = 10$).

Note that the Single Algorithm filter is related to the Robust decision tree method of [10]. The difference between the two approaches is that the Single Algorithm filter uses a cross-validation over the training data with one iteration whereas the Robust decision tree method deals with all the training examples and performs multiple iterations.

3.2.2 Consensus Filter and Majority Vote Filter

Our Consensus and Majority Vote filter differ from the ones described in [4] in the fact that we do not use m different filter algorithms, we only use Tilde. More precisely, we divide the training set in n parts, and for each of the combinations of $n - 1$ parts, Tilde learns on this combination. The resulting classifiers will be used to give labels to the examples in their own training sets.³ In this way, each training example gets $n - 1$ labels (since it belongs $n - 1$ times to a training set). Note that the resulting filter method might be more conservative⁴, since votes are given to examples which were used in the training set. In our experiments, we will evaluate such a Consensus and Majority Vote filter for $n = 6$ (each example gets 5 labels) and $n = 10$ (each example gets 9 labels). In the next section, our Consensus filter will be abbreviated as “C(6)” (for $n = 6$), and “C(10)” (for $n = 10$), and our Majority Vote filter as “M(6)” (for $n = 6$), and “M(10)” (for $n = 10$).

Note that our Voting filters are in a way dual to our Single Algorithm filter: instead of using the learned decision tree to classify examples in the subset excluded from the training set, the tree is used to classify the examples in its own training set. Note also that our Voting algorithms have some similarities with the Robust approach, described in subsection 3.1. More precisely, in the Robust decision tree method, the learning algorithm learns on the training data and removes those training examples that are misclassified by the learned (and pruned) tree. Our Voting algorithms also learn trees which classify the examples in their own training set. The difference is that for the Voting algorithms, these training sets consist of $n - 1$ of the n parts, and hence each example gets $n - 1$ votes. Also, the Voting algorithms are run only once, whereas Robust Tilde consists of a number of iterations (it keeps on running until no more training examples are re-

³As described in Section 2, Tilde with pruning is used; this is of course essential for this classifier.

⁴meaning that it will remove less examples

moved). In the next subsection, we present a hybrid approach combining Robust Tilde and our Voting algorithms: the Voting filters will be repeated until no more examples are removed.

As a remark, we see some similarities with bagging [3]. In bagging a combined classifier is constructed by learning n times from a bootstrap replicate of the training set. Bagging often improves the accuracy (in comparison with the original algorithm), if the base learning algorithm is sufficiently instable.⁵ Decision tree algorithms, and in particular Tilde, are instable enough to serve as underlying base algorithm in bagging. The difference with bagging is that in our Consensus and Majority Vote algorithms, no bootstrap replicate is taken, but an n -fold cross-validation is used. Moreover, the purpose of bagging is to build better, combined classifiers, whereas our technique is used only as a filtering mechanism.

3.3 Iterated Voting Filters

Finally, we present a hybrid approach, combining the Robust method with the Voting filters. The method works as follows:

1. filter the training data using a Voting filter (Single Algorithm filter, Consensus filter or Majority Vote filter),
2. if no training examples are removed in the previous step, then the reduced set of training examples is given as input to the final learning algorithm, else repeat the previous step.

We call the resulting filter algorithms, the Iterated Voting algorithms. More precisely, when the basic filtering method (of step 1) is a Single Algorithm filter, we call the resulting filter the Iterated Single Algorithm filter; when the basic filtering method is a Consensus filter, we call the resulting filter the Iterated Consensus filter; and when the basic filtering method is a Majority Vote filter, we call the resulting filter the Iterated Majority Vote filter. In the experiments, our Iterated Single Algorithm filter will be abbreviated as “IS(2)” (for $n = 2$), “IS(5)” (for $n = 5$), and “IS(10)” (for $n = 10$); our Iterated Consensus filter will be abbreviated as “IC(6)” (for $n = 6$), and “IC(10)” (for $n = 10$); and our Iterated Majority Vote filter as “IM(6)” (for $n = 6$), and “IM(10)” (for $n = 10$).

⁵meaning that small changes in the data set (by taking a bootstrap replicate) lead to significant changes in the learned hypothesis

4 Empirical Evaluation

We first describe the data set which we used in the experiments. Then we explain how the experiments were carried out, and finally we discuss the results.

4.1 Data Set and Noise Introduction

We want to evaluate how well the different filtering techniques perform on data sets with different amounts of classification noise. Since real world data sets often contain already an unknown number of noisy examples, we chose to do the experiments on a Bongard problem (see section 2). The advantage is that we can generate a noise-free Bongard data set and introduce a controlled amount of noise artificially. The Bongard data set with which the experiments were performed contains 392 examples, 128 of them are classified as positive, the other 264 as negative.

Different levels of noise were introduced. 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.⁶ We introduced noise levels of 0%, 5%, 10%, 15%, 20%, 25%, 30%, 35% and 40%.⁷

4.2 Experimental Method

All the filters are based on Tilde. Tilde was also used to evaluate the performance of the different filtering techniques.

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 all 10 folds. For each of the 10 runs, we made a training set (9 parts) and a test set (remaining 1 part). The training set was corrupted by introducing classification errors using noise levels of 0%, 5%, 10%, 15%, 20%, 25%, 30%, 35% and 40%. Each of the above described filtering techniques was then run on the (noisy⁸) training set. After filtering the training set, Tilde was used to learn a decision tree on the reduced

⁶Positive examples are made negative and vice versa, negative examples are made positive.

⁷More precisely, taking the total number of examples into account, these (rounded) percentages were 0%, 5.10%, 9.92%, 15.02%, 20.07%, 24.94%, 30.05%, 35.09% and 39.97%.

⁸For each noise level and each of the 10 training sets, the classification errors were introduced only once (in a random way), and the different filtering techniques were run on the same noisy training sets.

training set. This decision tree was 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 noise levels and each of the 10 runs, we also run Tilde directly on the (unfiltered, noisy) training set.

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

4.3 Filter Precision

In this subsection, we report results concerning the filter precision. For each of the filters and each level of introduced noise, we tabulate the following: the percentage of examples which are removed, the percentage of examples which are removed and are actually noisy, the percentage of noisy examples in the filtered data sets, and the probability of making an error of type 1 and of type 2.

A type 1 error (E_1) is made when a correct example is filtered. The probability of making a type 1 error is estimated as follows:

$$P(E_1) = \frac{\#(F \cap C)}{\#C}$$

with F the set of filtered examples and C the set of correct (non-noisy) examples.

A type 2 error (E_2) is made when a noisy example is not removed from the training set. The probability of making a type 2 error is estimated as follows:

$$P(E_2) = \frac{\#(F^c \cap N)}{\#N}$$

with F^c the complement of the set F (i.e. the examples which are not removed) and N the set of noisy examples.

For Robust Tilde and the Iterated Voting Filters, we will also report the number of rounds in the filtering process. We say that the number of rounds in an experiment using one of these filters is n if in round number $n + 1$ no training examples are removed from the training set. Note that the reported number of rounds is not always an integer, since this number is the mean taken over the 10 runs (in each run, the number of rounds is of course an integer).

4.3.1 Robust Tilde

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0.03	0	0	0.0003	NA	0.1
5.10	5.58	4.90	0.21	0.007	0.039	1.1
9.92	10.29	9.81	0.13	0.005	0.011	1
15.02	15.99	13.95	1.29	0.024	0.072	1
20.07	21.68	18.40	2.14	0.041	0.083	1.4
24.94	25.28	22.79	2.87	0.033	0.086	1.3
30.05	30.27	25.74	6.13	0.065	0.143	1.5
35.09	33.82	27.35	11.67	0.100	0.221	1.4
39.97	38.32	25.71	23.08	0.210	0.357	1.5

The Robust filter performs very well up to 25% of noise. Note that the number of rounds increases with the number of noisy examples, but remains low (1.1 rounds on average).

4.3.2 Single Algorithm Filter, $n = 2$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	2.24	0	0	0.022	NA
5.10	8.33	4.85	0.28	0.037	0.05
9.92	14.26	9.64	0.33	0.051	0.029
15.02	18.99	13.97	1.30	0.059	0.070
20.07	26.76	18.03	2.81	0.109	0.102
24.94	30.44	21.97	4.30	0.113	0.119
30.05	39.09	25.00	8.37	0.201	0.168
35.09	43.62	25.28	17.57	0.283	0.280
39.97	45.75	26.90	24.17	0.314	0.327

The Single Algorithm filter, with $n = 2$, is good at removing noisy examples up to a noise level of 20%. However, in comparison with the Robust filter, S(2) is a rather aggressive filter (i.e. removes more examples). Also, S(2) removes more correct examples ($P(E_1)$ with S(2) is always higher than $P(E_1)$ with the Robust filter), and removes less noisy examples ($P(E_2)$ with S(2) is always higher - except when there is 20% of noise in which case it is slightly less - than $P(E_2)$ with the Robust filter).

4.3.3 Single Algorithm Filter, $n = 5$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	0.17	0	0	0.002	NA
5.10	6.60	4.93	0.18	0.018	0.033
9.92	12.16	9.78	0.16	0.026	0.014
15.02	17.60	13.97	1.28	0.043	0.070
20.07	24.83	18.79	1.69	0.076	0.064
24.94	29.65	22.53	3.42	0.095	0.097
30.05	36.48	26.96	4.85	0.136	0.103
35.09	42.01	27.83	12.50	0.218	0.207
39.97	46.43	29.68	19.11	0.279	0.257

The Single Algorithm filter with $n = 5$ performs well up to a noise level of 25-30 %. S(5) is not so aggressive as S(2) and retains more examples (except for the 40% noise level). S(5) is more precise than S(2). That is, the probability of making a type 1 error as well as the probability of making a type 2 error is smaller for each noise level (in comparison with S(2)). But, in comparison with Robust Tilde, S(5) has a higher probability of making a type 1 error. The probabilities of making a type 2 error with S(5) versus the Robust filter are comparable.

4.3.4 Single Algorithm Filter, $n = 10$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	0.06	0	0	0.0006	NA
5.10	6.32	4.99	0.12	0.014	0.022
9.92	11.62	9.83	0.10	0.020	0.009
15.02	17.40	14.29	0.90	0.037	0.049
20.07	23.41	18.96	1.44	0.056	0.055
24.94	29.11	22.96	2.80	0.082	0.080
30.05	36.25	27.64	3.80	0.123	0.080
35.09	41.13	28.68	10.88	0.192	0.183
39.97	46.74	29.17	20.18	0.293	0.270

The Single Algorithm filter with $n = 10$ performs well up to a noise level of 30%. Except for the noise level of 40%, S(10) throws out (slightly) less

examples than S(5) (and hence S(2)). Also, S(10) is more precise than S(5) (and hence S(2)): $P(E_1)$ and $P(E_2)$ are smaller if you use S(10) instead of S(5) (and hence S(2)), except for the noise level of 40%. In comparison with the Robust filter, the probability of making a type 1 error with S(10) is higher, but the probability of making a type 2 error is smaller.

4.3.5 Iterated Single Algorithm Filter, $n = 2$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	6.26	0	0	0.063	NA	3.5
5.10	12.70	4.88	0.25	0.082	0.044	3.6
9.92	17.97	9.75	0.21	0.091	0.017	3.8
15.02	22.31	14.29	0.94	0.094	0.049	2.9
20.07	32.54	19.08	1.48	0.168	0.050	3.7
24.94	37.24	23.75	1.87	0.180	0.048	3.8
30.05	48.04	28.49	3.03	0.280	0.052	3.5
35.09	57.06	30.24	11.43	0.413	0.138	5
39.97	58.84	32.26	17.97	0.443	0.193	3.9

Note that this filter takes on average (over the different noise levels) 3.7 rounds, and even when there is no noise in the data set, the filter takes (on average over the 10 runs) 3.5 rounds. As can be easily understood, the Iterated Single Algorithm filter with $n = 2$ is more aggressive than the Single Algorithm filter with $n = 2$, meaning that more examples are removed from the data set. This results in a smaller probability of making an error of type 2 (retaining noisy examples) but a higher probability of making an error of type 1 (throwing out correct examples). For high noise levels (30% and higher) almost or more than half of the data set is removed. Actually, from all filters presented here, IS(2) is the most aggressive one.

4.3.6 Iterated Single Algorithm Filter, $n = 5$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0.37	0	0	0.004	NA	0.4
5.10	8.02	5.05	0.06	0.031	0.011	2.6
9.92	13.78	9.89	0.03	0.043	0.003	2.4
15.02	19.45	14.17	1.06	0.062	0.057	2.6
20.07	27.13	19.30	1.04	0.098	0.038	2.9
24.94	33.68	23.84	1.64	0.131	0.044	3.5
30.05	42.66	28.91	1.94	0.196	0.038	4.2
35.09	48.30	30.38	8.92	0.276	0.134	3.8
39.97	55.39	33.70	13.5	0.361	0.157	4.1

Similar comment as above: IS(5) is more aggressive than S(5), $P(E_1)$ is higher but $P(E_2)$ is smaller. In comparison with IS(2), IS(5) removes a smaller number of examples and has a smaller $P(E_1)$ value, but the number of noisy examples removed from the data set is higher with IS(5) than with IS(2) (except for noise level of 15% where it is slightly smaller), that is IS(5) has a smaller $P(E_2)$ value. So, IS(5) is more precise than IS(2). Note that IS(5) needs 2.9 rounds on average.

4.3.7 Iterated Single Algorithm Filter, $n = 10$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0.09	0	0	0.0009	NA	0.3
5.10	6.60	4.99	0.12	0.017	0.022	1.5
9.92	12.87	9.89	0.03	0.033	0.003	2.3
15.02	18.43	14.43	0.74	0.047	0.040	2.1
20.07	25.11	19.33	0.98	0.072	0.037	2.5
24.94	31.12	23.78	1.69	0.098	0.047	3
30.05	39.65	28.86	1.96	0.154	0.040	3.3
35.09	46.06	31.26	6.94	0.228	0.109	3.3
39.97	52.92	32.09	16.36	0.347	0.197	3

Again, IS(10) is more aggressive than S(10), and $P(E_1)$ is higher but $P(E_2)$ is smaller. IS(10) removes less examples than IS(5), and the number of noisy examples that are removed is more or less equal for both filters. Or,

$P(E_1)$ is smaller for IS(10) than for IS(5), and $P(E_2)$ is more or less the same for both filters. In comparison with the Robust filter, IS(10) is more aggressive, has smaller $P(E_2)$ values than the Robust filter, but the Robust filter has smaller $P(E_1)$ values than IS(10). On average, IS(10) takes 2.4 rounds.

4.3.8 Consensus Filter, $n = 6$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	0.03	0	0	0.0003	NA
5.10	4.79	4.62	0.50	0.002	0.094
9.92	9.10	8.87	1.15	0.003	0.106
15.02	13.12	12.50	2.90	0.007	0.168
20.07	15.79	15.70	5.13	0.001	0.218
24.94	17.77	16.69	9.95	0.014	0.331
30.05	18.25	17.83	14.81	0.006	0.407
35.09	16.84	15.16	23.85	0.026	0.568
39.97	14.54	12.30	32.26	0.037	0.692

The Consensus filter with $n = 6$ is a conservative filter. The number of examples which are removed is always less than the number of noisy examples. Most of these removed examples are noisy. However, a number of noisy examples are retained, resulting in a rather high probability of making a type 2 error. Up to the noise level of 15%, the percentage of noise in the filtered data set is low, but for higher noise levels, a lot of noisy examples remain in the filtered set. On the other hand, the probability of making a type 1 error is very low (for all noise levels). From all the filters presented here, the consensus filter has the smallest $P(E_1)$ values and the highest $P(E_2)$ values.

4.3.9 Consensus Filter, $n = 10$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	0	0	0	0	NA
5.10	4.93	4.70	0.42	0.002	0.078
9.92	8.96	8.96	1.05	0	0.097
15.02	13.12	12.75	2.61	0.004	0.151
20.07	16.61	16.44	4.33	0.002	0.181
24.94	18.05	16.92	9.67	0.015	0.322
30.05	19.11	18.54	14.09	0.008	0.383
35.09	16.21	15.28	23.60	0.014	0.565
39.97	13.72	12.07	32.22	0.027	0.698

The precision of C(10) is comparable with (slightly better, i.e. slightly smaller $P(E_2)$, than) the precision of C(6).

4.3.10 Majority Vote Filter, $n = 6$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	0.03	0	0	0.0003	NA
5.10	5.47	4.90	0.21	0.006	0.039
9.92	10.40	9.72	0.22	0.008	0.020
15.02	15.48	13.83	1.41	0.019	0.079
20.07	20.35	18.62	1.81	0.022	0.072
24.94	25.00	22.53	3.20	0.033	0.097
30.05	28.23	25.77	5.93	0.035	0.142
35.09	31.80	25.96	13.28	0.090	0.260
39.97	34.21	24.71	23.16	0.158	0.382

As could be expected, the Majority Vote filter is more aggressive than the Consensus filter. Although this results in a higher (but still acceptable) probability of making a type 1 error, this also results in a lower type 2 error, so that the resulting filtered data sets contains very few noisy examples. The filter performs well up to a noise level of 25%. The precision of the Majority Vote filter is comparable with that of the Robust filter.

4.3.11 Majority Vote Filter, $n = 10$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$
0	0.06	0	0	0.0006	NA
5.10	5.56	4.88	0.24	0.007	0.044
9.92	10.26	9.84	0.10	0.005	0.009
15.02	15.76	14.11	1.08	0.019	0.060
20.07	20.83	18.45	2.04	0.030	0.081
24.94	24.91	22.53	3.20	0.032	0.097
30.05	28.52	25.40	6.45	0.045	0.155
35.09	31.46	26.27	12.78	0.080	0.251
39.97	35.46	25.43	22.54	0.167	0.364

The precision of the Majority Vote filter with $n = 10$ is comparable with the precision of the Majority Vote filter with $n = 6$.

4.3.12 Iterated Consensus Filter, $n = 6$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0.03	0	0	0.0003	NA	0.1
5.10	5.02	4.82	0.30	0.002	0.056	1.7
9.92	9.81	9.52	0.44	0.003	0.04	1.9
15.02	14.43	13.52	1.75	0.011	0.1	2.7
20.07	18.37	18.06	2.45	0.004	0.100	2.8
24.94	22.45	20.21	6.03	0.030	0.190	4.1
30.05	23.92	22.71	9.40	0.017	0.244	3.6
35.09	28.52	24.21	15.09	0.066	0.310	5.3
39.97	31.72	21.66	26.73	0.168	0.458	6.1

IC(6) performs better than C(6). Although the probability of making a type 1 error is slightly higher (but still acceptable), the probability of making a type 2 error is smaller. IC(6) needs on average 3.1 rounds.

4.3.13 Iterated Consensus Filter, $n = 10$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0	0	0	0	NA	0
5.10	5.05	4.79	0.33	0.003	0.061	1.3
9.92	9.95	9.61	0.34	0.004	0.031	1.8
15.02	14.06	13.35	1.94	0.008	0.111	1.9
20.07	18.28	18.03	2.48	0.003	0.102	2
24.94	20.61	18.93	7.51	0.022	0.241	2.8
30.05	22.82	21.63	10.80	0.017	0.280	3.2
35.09	25.91	22.05	17.39	0.059	0.372	4.9
39.97	25.76	20.58	25.79	0.086	0.485	5.6

Similar comments as for IC(6). IC(10) needs on average 2.6 rounds.

4.3.14 Iterated Majority Vote Filter, $n = 6$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0.03	0	0	0.0003	NA	0.1
5.10	5.56	4.96	0.15	0.006	0.028	1.3
9.92	11.14	9.72	0.23	0.016	0.020	1.5
15.02	15.73	13.92	1.32	0.021	0.074	1.4
20.07	21.32	18.79	1.62	0.032	0.064	1.8
24.94	25.93	23.10	2.49	0.038	0.074	2
30.05	31.09	27.32	3.94	0.054	0.091	2.7
35.09	35.40	27.78	11.32	0.117	0.208	2.8
39.97	39.31	26.79	21.66	0.209	0.330	2.6

The precision of IM(6) is comparable with (but slightly higher $P(E_1)$ and slightly smaller $P(E_2)$ than) the precision of M(6). On average IM(6) needs 1.8 rounds.

4.3.15 Iterated Majority Vote Filter, $n = 10$

% noise	% filtered	% noise and filtered	% noise in filtered set	$P(E_1)$	$P(E_2)$	rounds
0	0.06	0	0	0.0006	NA	0.2
5.10	5.58	4.90	0.21	0.007	0.039	1.1
9.92	10.35	9.86	0.06	0.005	0.006	1.1
15.02	15.76	14.11	1.08	0.019	0.060	1
20.07	21.60	18.71	1.73	0.036	0.068	1.5
24.94	25.42	22.93	2.68	0.033	0.081	1.6
30.05	30.73	26.53	5.07	0.060	0.117	2.3
35.09	34.01	27.49	11.51	0.100	0.217	2.6
39.97	38.58	26.96	21.15	0.194	0.326	2.2

Similar remarks as for IM(6). The precision of IM(10) is comparable with the precision of IM(6). IM(10) needs 1.5 rounds on average.

4.3.16 Conclusions

Except for the (Iterated) Consensus filters, all filters perform well up to a noise level of 25% (or higher). That is, in the reduced training sets only a small percentage of the examples are noisy.

From our experiments with the Single Algorithm filters, we see that, if we are concerned with filter precision, S(10) should be preferred over S(5), which in turn should be preferred over S(2). Indeed, our experiments show that the higher the parameter n in the Single Algorithm filter, the less aggressive the filter is (i.e. throwing out less examples) and the smaller the probabilities $P(E_1)$ and $P(E_2)$ are. We might explain this by observing that the classifiers, which act as filter, are trained on $n - 1$ parts of the training set. So, the higher n , the more training data is available. However, it should be observed that this training data is noisy as well. It can be easily seen that the higher the parameter n , the more time the filtering takes. We were not concerned with time requirements in this paper.

Comparing S(10) (the “best” filter of the Single Algorithm filters) with the Robust filter, we notice that S(10) is more aggressive (removes more examples) than the Robust filter. More precisely, S(10) removes more correct examples (higher $P(E_1)$) than the Robust filter but also removes more noisy examples (smaller $P(E_2)$) than the Robust filter.

For the iterated versions of the Single Algorithm filters, we see that

IS(10) is less aggressive than IS(5), which in turn is less aggressive than IS(2). Also, IS(5) is more precise than IS(2) (smaller $P(E_1)$ and $P(E_2)$), and IS(10) is more precise than IS(5) (smaller $P(E_1)$, but $P(E_2)$ is more or less the same).

From all filters presented here, IS(2) is the most aggressive one (removes the most examples).

Comparing the Single Algorithm filters $S(n)$ with their iterated versions IS(n) ($n = 2, 5, 10$), we observe that IS(n) is more aggressive than $S(n)$, has higher $P(E_1)$ and smaller $P(E_2)$. This can be easily explained since IS(n) is just $S(n)$ run several times (until no examples are removed anymore).

Comparing IS(10) (and IS(5)) and the Robust filter: the Robust filter is less aggressive than IS(10) (and hence IS(5)), IS(5) and IS(10) have smaller $P(E_2)$ than the Robust filter, and the Robust filter has smaller $P(E_1)$ than IS(10) (and hence IS(5)).

The Consensus filters C(6) and C(10) are the most conservative (the less aggressive) filters. From all filters, they have the smallest $P(E_1)$ and highest $P(E_2)$ values. One can choose to use a Consensus filter if the training set is small and the cost of adding new training examples is high. The iterated versions of the Consensus filters are (as can be easily understood) less conservative (but in comparison with the other filters, still more conservative). This results in a slightly higher, but still acceptable $P(E_1)$, but a smaller $P(E_2)$.

The Majority Vote filters M(6) and M(10) have a precision comparable with the precision of the Robust filter. The precision of the iterated versions of the Majority Vote filters are comparable with (but slightly higher $P(E_1)$ and slightly lower $P(E_2)$ than) the precision of the Majority Vote filters.

4.4 Tree Size

We tabulate the number of nodes in the trees induced from the filtered sets. We also report the number of nodes in the trees induced from the unfiltered sets (column under “Tilde”). Note that, since we take the mean over 10 runs, the values in the tables below are not always integers. When the number of nodes in a tree induced from a filtered set is smaller than or equal to the number of nodes in the tree induced from the unfiltered data set, the number is put in bold.

% noise	Tilde	R	S(2)	S(5)	S(10)	IS(2)	IS(5)	IS(10)
0	7.2	7.2	7.7	7	7.2	6.3	7	7.2
5.10	8.8	8	7.4	8	7.5	5.4	7.4	7.4
9.92	9.2	8.3	7.1	7.9	7.9	5.1	6.9	7.3
15.02	8.3	7.3	6.2	7	7.7	4.4	5.9	6.8
20.07	8.5	7.8	7.3	6.9	7.4	3.9	5.3	6.2
24.94	9.3	8.3	6.9	8.4	6.7	4.2	5.4	6.1
30.05	12.3	10.9	7.8	8.2	6.8	3.8	4.5	5
35.09	11.6	12.2	8	8.8	8.4	2.2	4.5	5.7
39.97	9.4	10.5	8.1	7.5	6.8	2.3	4.2	3.7

% noise	Tilde	C(6)	C(10)	M(6)	M(10)	IC(6)	IC(10)	IM(6)	IM(10)
0	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
5.10	8.8	7.7	7.7	7.7	8	7.7	7.7	7.7	8
9.92	9.2	8.4	8.4	8.2	8.2	8.6	8.6	8	8.2
15.02	8.3	8.8	8.1	7.8	7.1	7.8	8.6	7.8	7.1
20.07	8.5	9.3	9.3	8.7	8.8	9.6	9.3	8.5	8.9
24.94	9.3	10.6	10.2	8.2	7.8	10.3	10.4	7.7	8.1
30.05	12.3	14.1	15.3	7.9	10.6	13	14.5	7.8	10.4
35.09	11.6	13.2	13.7	11.4	13	13.7	13.9	9.2	12
39.97	9.4	11.9	12.3	11.7	10	13.8	13.6	9.5	10.1

4.4.1 Conclusions

The trees induced by Tilde from a filtered training set are almost always smaller than the trees induced by Tilde from the noisy training set (except for the (Iterated) Consensus filters where the trees are larger for noise levels from 15-20% to 40%). Using the IS(2) filter results in the smallest trees.

From subsection 4.3 we know that, using IS(2), the number of noisy examples in the filtered training set is small. We are tempted to conclude that the reduction in tree size by using the filter IS(2) is due to the removal of noisy examples. However, there are other filters (e.g. IS(5) and IS(10)) for which the number of noisy examples in the filtered training set is also low, but for which the induced trees from the filtered training set are not so small. From subsection 4.3 we know that IS(2) is a very aggressive filter (the most aggressive one of the filters presented here): even when there are no noisy examples, IS(2) removes 6.26% examples from the training set, and for noise levels of 30% and more IS(2) removes almost and more than

50% of the training examples. When we use IS(2), we obtain the smallest filtered training set. The reduction in tree size resulting from using the IS(2) filter seems therefore not only attributable to the removal of noisy examples from the training set, but also to the reduction in the size of the training set. In [12] it is empirically shown that for many data sets there is a nearly linear relationship between the tree size and the number of training instances. Therefore it is suggested that all data reduction techniques will see some decrease in tree size simply because they are reducing the size of the training set. In order to have an idea of how much of the reduction in tree size is directly attributable to how a data reduction method selects examples to remove, a data reduction method should be compared with random data reduction. In [12] this analysis was done for Robust C4.5 [10] and it was shown that 41.67% of the decrease in tree size is attributable to reduction in training set size. The remainder is due to the removal of uninformative examples. In the appendix, we do this analysis for some of our filters (namely for IS(2), for the Robust filter, and for IM(6)).

4.5 Accuracy and Cramer’s Coefficient

We tabulate the accuracies of the trees induced from the filtered sets (on the non-noisy test sets). We compare them with the accuracies (also on the non-noisy test sets) of the trees induced from the unfiltered, noisy sets (reported in the column “Tilde”). We do the same for Cramer’s coefficient. When the accuracy, resp. Cramer’s coefficient, obtained by using Tilde on a filtered training set is equal to or higher than the accuracy, resp. Cramer’s coefficient, obtained by using Tilde on the unfiltered training set, it is put in bold.

4.5.1 Accuracy

% noise	Tilde	R	S(2)	S(5)	S(10)	IS(2)	IS(5)	IS(10)
0	1	1	0.95	0.99	1	0.93	0.99	1
5.10	0.99	0.99	0.96	0.98	0.98	0.91	0.97	0.97
9.92	0.99	0.99	0.95	0.97	0.97	0.92	0.95	0.96
15.02	0.94	0.94	0.93	0.94	0.94	0.91	0.94	0.94
20.07	0.95	0.95	0.89	0.94	0.95	0.86	0.93	0.93
24.94	0.93	0.94	0.92	0.92	0.92	0.87	0.91	0.92
30.05	0.90	0.90	0.89	0.94	0.89	0.84	0.89	0.91
35.09	0.85	0.86	0.75	0.85	0.84	0.74	0.83	0.86
39.97	0.75	0.75	0.75	0.77	0.75	0.70	0.74	0.73

% noise	Tilde	C(6)	C(10)	M(6)	M(10)	IC(6)	IC(10)	IM(6)	IM(10)
0	1	1	1	1	1	1	1	1	1
5.10	0.99	0.98	0.98	0.99	0.99	0.98	0.98	0.99	0.99
9.92	0.99	0.97	0.97	0.99	0.99	0.99	0.98	0.99	0.99
15.02	0.94	0.94	0.95	0.95	0.94	0.95	0.96	0.95	0.94
20.07	0.95	0.97	0.97	0.96	0.95	0.97	0.97	0.95	0.95
24.94	0.93	0.92	0.92	0.92	0.94	0.90	0.92	0.92	0.94
30.05	0.90	0.89	0.88	0.93	0.91	0.90	0.90	0.93	0.91
35.09	0.85	0.81	0.83	0.85	0.86	0.86	0.83	0.85	0.86
39.97	0.75	0.71	0.78	0.74	0.75	0.74	0.76	0.75	0.75

4.5.2 Cramer's Coefficient

% noise	Tilde	R	S(2)	S(5)	S(10)	IS(2)	IS(5)	IS(10)
0	1	1	0.89	0.97	1	0.83	0.97	1
5.10	0.98	0.98	0.92	0.95	0.95	0.80	0.93	0.93
9.92	0.98	0.98	0.88	0.93	0.92	0.82	0.89	0.92
15.02	0.87	0.87	0.84	0.86	0.85	0.80	0.85	0.86
20.07	0.89	0.88	0.76	0.87	0.88	0.70	0.83	0.84
24.94	0.84	0.86	0.82	0.83	0.83	0.69	0.79	0.83
30.05	0.77	0.76	0.76	0.85	0.75	0.65	0.73	0.80
35.09	0.67	0.69	0.44	0.67	0.66	0.33	0.61	0.68
39.97	0.42	0.42	0.40	0.46	0.41	0.29	0.38	0.38

% noise	Tilde	C(6)	C(10)	M(6)	M(10)	IC(6)	IC(10)	IM(6)	IM(10)
0	1	1	1	1	1	1	1	1	1
5.10	0.98	0.96	0.96	0.98	0.98	0.96	0.96	0.98	0.98
9.92	0.98	0.94	0.94	0.97	0.97	0.97	0.96	0.97	0.97
15.02	0.87	0.87	0.89	0.89	0.87	0.90	0.90	0.89	0.87
20.07	0.89	0.93	0.93	0.91	0.88	0.93	0.93	0.89	0.88
24.94	0.84	0.82	0.82	0.82	0.86	0.78	0.82	0.83	0.87
30.05	0.77	0.74	0.72	0.84	0.79	0.77	0.75	0.83	0.79
35.09	0.67	0.59	0.62	0.65	0.68	0.70	0.64	0.67	0.68
39.97	0.42	0.36	0.48	0.43	0.42	0.40	0.43	0.42	0.40

4.5.3 Conclusions

From the above tables we see that using a (Iterated) Single Algorithm filter before learning with Tilde does not increase accuracy/Cramer’s coefficient (except for a few cases). More interestingly in this respect is Robust Tilde and the (Iterated) Majority Vote filters, which perform best w.r.t. accuracy. We also observe that the (Iterated) Consensus and (Iterated) Majority Vote filters are especially useful for noise levels of 15% and 20%. Because Majority Vote filters perform better than Consensus filters, we might conclude that retaining noisy examples hinders performance (in terms of accuracy) more than throwing out correct examples. This trend is particularly important when one has an abundance of data. This was also observed in [4] for other (non-artificial) data sets. Finally, we should note that the accuracy of Tilde on an unfiltered noisy training set is still very good: Tilde is very noise-tolerant.

5 Conclusions and Future Work

We addressed the problem of training sets with mislabeled examples in ILP classification problems. We proposed a number of filtering techniques for identifying and removing noisy examples. We experimentally evaluated these techniques on a Bongard data set which we artificially corrupted with different levels of classification noise. We reported results concerning filter precision, tree size and accuracy. Concerning accuracy, we see that Tilde also performs well on an unfiltered training set; we may conclude that Tilde is very noise-tolerant.

There are several issues which remain to be studied. First of all, we only

validated the filtering techniques on one (artificial) data set. It remains to be seen if our conclusions also hold for other (non-artificial) data sets. Therefore, we first plan to validate the filters on larger Bongard data sets, to study to influence of the size of the training set. Then, we plan to use real world data sets, as in [4]. We only proposed filtering techniques for classification problems with 2 classes. We plan to extend our techniques to multi-class problems. These extended filters will then be tested on real world data sets, which we will corrupt “in a natural way” as was done in [4]. There, domain experts were consulted to identify the pairs of classes likely to be confused. Noise was then artificially introduced into the training labels between these pairs of classes (and not between all pairs of classes as this would not model the types of labelling errors that occur in practice).

Since in the different filtering algorithms a classifier formed from a noisy data set is used to determine if examples are mislabeled, the identification of mislabeled examples will lead to some errors. We estimated the probability of making type 1 and type 2 errors in our experiments. A topic of future research is to make a more formal analysis of the probabilities of making type 1 and type 2 errors for the different filtering algorithms.

When the data set is small and the cost of finding new training examples is high, one can choose to use a conservative 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. We plan to extend the filter approach to automatically correct the labelling errors in the training set. We will evaluate the performance of such extensions.

A hypothesis of interest is whether a majority vote ensemble classifier can be used instead of filtering. This hypothesis was tested in [4]. There, two majority vote ensemble classifiers were formed: one from the filtered and one from the unfiltered data. The resulting classifiers were then used to classify the uncorrupted test data. The experiments in [4] show that the majority vote classifier performed better than the individual classifiers, but that it cannot replace filtering when data are noisy. It is concluded that the best approach is to combine filtering and voting. We also want to test this hypothesis in our setting.

We were not concerned with efficiency in this paper. But it can be easily understood that some of our filters take quite some time. The Robust filter and the Iterated Voting filters run Tilde several times on slightly modified (reduced) training sets. An efficient algorithm for updating first-order trees would be beneficial in this respect.

The work in this paper is based on the Master thesis of Thomas Cardoen [5]. In this thesis, another filter, named Boosting filter, was introduced. The idea is to use boosting [16] to identify noisy examples: the examples with the highest weights are removed. The number of examples which are removed must be given as input to the Boosting filter. In [5], the actual percentage of noise was given as input. An idea is to first estimate the percentage of noisy examples (for instance by taking the percentage of examples removed by the Robust filter) and to give this estimate as input to the Boosting filter. Since boosting takes several rounds, and the learning algorithm is forced to concentrate on the “difficult” examples, the Boosting filter takes a lot of time. We did not include the Boosting filter in this paper.

A Comparison with Random Data Reduction

In [12] it is empirically shown that for many data sets there is a nearly linear relationship between training set size and tree size, even after accuracy has ceased to increase. This suggests that all data reduction techniques will see some decrease in tree size simply because they are reducing the size of the training set. Therefore, each data reduction method should be compared with random data reduction. Only then one can have an idea of how much of the reduction in tree size is directly attributable to how a data reduction method selects instances to remove.

In [12] this analysis was done for Robust C4.5 [10] and it was shown that 41.67% of the decrease in tree size is attributable to reduction in training set size. The remainder is due to the removal of uninformative examples.

We will investigate this issue for some of our filters here. To determine how much of a filter's effect on tree size for a given data set is due to reduction of training set size alone, we need to know the following (we follow the approach of [12]):

- the size of the tree that Tilde builds on the entire data set (Tilde Size),
- the size of the tree that Tilde builds on the filtered data set (Filter Size),
- the size of the tree that Tilde builds on the data set from which we randomly removed the same percentage of examples as the filter did (RDR Size).

The percentage of the filter's effect on tree size which is due to reduction in training set size can then be computed as:

$$100 * \frac{\text{Tilde Size} - \text{RDR Size}}{\text{Tilde Size} - \text{Filter Size}}$$

We do the analysis for the Iterated Single Algorithm filter with $n = 2$, the Robust filter and the Iterated Majority Vote filter with $n = 6$.

A.1 Iterated Single Algorithm filter with $n = 2$

Recall from section 4.4 that, using IS(2), the smallest trees were obtained.

% noise	Tilde Size	IS(2) Size	% Filtered	RDR Size	% of IS(2) Effect Due to RDR
0	7.2	6.3	6.26	8.1	-100
5.10	8.8	5.4	12.70	9	-5.88
9.92	9.2	5.1	17.97	8.5	17.07
15.02	8.3	4.4	22.31	7.9	10.26
20.07	8.5	3.9	32.54	8.1	8.70
24.94	9.3	4.2	37.24	8.5	15.69
30.05	12.3	3.8	48.04	7.4	57.65
35.09	11.6	2.2	57.06	9.7	20.21
39.97	9.4	2.3	58.84	6.6	39.44

We see that, for all noise levels, a great percentage of the reduction in tree size is attributable to how IS(2) selects examples to remove. For the noise level of 10% (17.97% of the examples are removed) and higher, we also see a reduction in tree size using random data reduction, but this reduction is never as big as the reduction we observe when we use IS(2).

We next tabulate the accuracies of the induced trees on the (non-noisy) test set. In the column “Tilde Acc” we report the accuracies of the trees induced on the unfiltered training set; in the column “IS(2) Acc” we report the accuracies of the trees induced on the filtered (using the IS(2) filter) training set; in the column “RDR Acc” we report the accuracies of the trees induced on the training set from which we randomly removed the same percentage of training examples as the IS(2) filter did.

% noise	Tilde Acc	IS(2) Acc	RDR Acc
0	1	0.93	1
5.10	0.99	0.91	0.97
9.92	0.99	0.92	0.98
15.02	0.94	0.91	0.94
20.07	0.95	0.86	0.94
24.94	0.93	0.87	0.90
30.05	0.90	0.84	0.83
35.09	0.85	0.74	0.72
39.97	0.75	0.70	0.69

Strangely enough, we observe that up to a noise level of 25%, the accur-

acy of Tilde using RDR is always higher than⁹ the accuracy of Tilde using the IS(2) filter. This is not what we expected, and we plan to investigate this issue further.

A.2 Robust Tilde

Below, we tabulate our results for the Robust filter.

% noise	Tilde Size	Robust Size	% Filtered	RDR Size	% of Robust Effect Due to RDR
0	7.2	7.2	0.03	7.2	no size reduction
5.10	8.8	8	5.58	8.9	-12.5
9.92	9.2	8.3	10.29	8.7	55.56
15.02	8.3	7.3	15.99	8.9	-60
20.07	8.5	7.8	21.68	8.7	-28.57
24.94	9.3	8.3	25.28	7.8	150
30.05	12.3	10.9	30.27	9.5	200
35.09	11.6	12.2	33.82	10.4	no size reduction
39.97	9.4	10.5	38.32	7.5	no size reduction

For noise levels of 5% up to 20% we see that, again, a great percentage of reduction in tree size is attributable to how the Robust filter selects examples to remove. For higher noise levels however, the size of the tree using RDR is smaller than the size of the tree using the Robust filter.

We also tabulate the accuracies of the induced trees on the (non-noisy) test set.

⁹but still not higher than the accuracy of Tilde trained on the unfiltered noisy data set

% noise	Tilde Acc	Robust Acc	RDR Acc
0	1	1	1
5.10	0.99	0.99	0.98
9.92	0.99	0.99	0.97
15.02	0.94	0.94	0.94
20.07	0.95	0.95	0.93
24.94	0.93	0.94	0.91
30.05	0.90	0.90	0.84
35.09	0.85	0.86	0.78
39.97	0.75	0.75	0.78

Here we see that the accuracy of Tilde using the Robust filter is always higher than or equal to¹⁰ the accuracy of Tilde using RDR.

A.3 Iterated Majority Vote filter with $n = 6$

We do the same for the Iterated Majority Vote filter with $n = 6$.

% noise	Tilde Size	IM(6) Size	% Filtered	RDR Size	% of IM(6) Effect Due to RDR
0	7.2	7.2	0.03	7.2	no size reduction
5.10	8.8	7.7	5.56	8.4	36.36
9.92	9.2	8	11.14	9	16.67
15.02	8.3	7.8	15.73	9.8	-300
20.07	8.5	8.5	21.32	8.1	no size reduction
24.94	9.3	7.7	25.93	9	18.75
30.05	12.3	7.8	31.09	9	73.33
35.09	11.6	9.2	35.40	11.8	-8.33
39.97	9.4	9.5	39.31	9.1	no size reduction

Also here we see that, except for the cases where there is no tree size reduction, a great percentage of reduction in tree size is attributable to how IM(6) selects examples to remove.

We also tabulate the accuracies.

¹⁰except for the noise level of 40%

% noise	Tilde Acc	IM(6) Acc	RDR Acc
0	1	1	1
5.10	0.99	0.99	0.99
9.92	0.99	0.99	0.97
15.02	0.94	0.95	0.96
20.07	0.95	0.95	0.96
24.94	0.93	0.92	0.91
30.05	0.90	0.93	0.88
35.09	0.85	0.85	0.73
39.97	0.75	0.75	0.66

Especially for noise levels of 25% and higher, the accuracy of Tilde using IM(6) is better than the accuracy of Tilde using RDR.

References

- [1] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
- [2] P. Brazdil and P. Clark. Learning from imperfect data. In P. Brazdil and K. Konolige, editors, *Machine Learning, Meta-reasoning and Logics*, pages 207–232. Kluwer Academic Press, 1990.
- [3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [4] C.E. Brodley and M.A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.
- [5] T. Cardoen. Technieken voor filteren van ruis bij op ILP gebaseerde data mining algoritmes. Master’s thesis, Department of Computer Science, Katholieke Universiteit Leuven, 2002. In Dutch.
- [6] D. Gamberger and N. Lavrač. Conditions for Occam’s razor applicability and noise elimination. In G. Widmer M. van Someren, editor, *Proceedings of the European Conference on Machine Learning (ECML)*, volume 1224 of *Lecture Notes in Computer Science*, pages 108–123. Springer, 1997.
- [7] D. Gamberger, N. Lavrač, and S. Džeroski. Noise elimination in inductive concept learning: a case study in medical diagnosis. In *Algorithmic Learning Theory: ALT '96*, volume 1160, pages 199–212. Springer-Verlag, 1996.
- [8] 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.
- [9] Dragan Gamberger, Nada Lavrač, and Ciril Grošelj. Experiments with noise filtering in a medical domain. In *Proc. 16th International Conf. on Machine Learning*, pages 143–151. Morgan Kaufmann, San Francisco, CA, 1999.
- [10] 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.

- [11] N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.
- [12] Tim Oates and David Jensen. The effects of training set size on decision tree complexity. In *Proceedings of The Fourteenth International Conference on Machine Learning*, pages 254–262. Morgan Kaufmann, Nashville, TN, 1997.
- [13] S. Parsons. Imperfect information in knowledge and databases: a survey of current approaches. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):353–372, 1996.
- [14] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [15] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann, 1993.
- [16] J.R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 725–730. AAAI Press, 1996.
- [17] A. Srinivasan, S. Muggleton, and M. Bain. Distinguishing exceptions from noise in non-monotonic learning. In *Proceedings of the 2nd International Workshop on Inductive Logic Programming*, 1992.
- [18] S. Weisberg. *Applied linear regression*. John Wiley & Sons, 1980.