Using Decision Trees and Soft Labeling to Filter Mislabeled Data

Xinchuan Zeng and Tony Martinez
Department of Computer Science
Brigham Young University, Provo, UT 84602
E-Mail: zengx@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract

In this paper we present a new noise filtering method, called soft decision tree noise filter (SDTNF), to identify and remove mislabeled data items in a data set. In this method, a sequence of decision trees are built from a data set, in which each data item is assigned a soft class label (in the form of a class probability vector). A modified decision tree algorithm is applied to adjust the soft class labeling during the tree building process. After each decision tree is built, the soft class label of each item in the data set is adjusted using the decision tree's predictions as the learning targets. In the next iteration, a new decision tree is built from a data set with the updated soft class labels. This tree building process repeats iteratively until the data set labeling converges. This procedure provides a mechanism to gradually modify and correct mislabeled items. It is applied in SDTNF as a filtering method by identifying data items whose classes have been relabeled by decision trees as mislabeled data. The performance of SDTNF is evaluated using 16 data sets drawn from the UCI data repository. The results show that it is capable of identifying a substantial amount of noise for most of the tested data sets and significantly improving performance of nearest neighbor classifiers at a wide range of noise levels. We also compare SDTNF to the consensus and majority voting methods proposed by Brodley and Friedl [1996, 1999] for noise filtering. The results show SDTNF has a more efficient and balanced filtering capability than these two methods in terms of filtering mislabeled data and keeping non-mislabeled data. The results also show that the filtering capability of SDTNF can significantly improve the performance of nearest neighbor classifiers, especially at high noise levels. At a noise level of 40%, the improvement on the accuracy of nearest neighbor classifiers is 13.1% by the consensus voting method and 18.7% by the majority voting method, while SDTNF is able to achieve an improvement by 31.3%.

Key Words: decision trees, soft labeling, mislabeled data.

1. INTRODUCTION

The quality of data sets is an important issue in machine learning and pattern recognition. For instance, a data set containing significant noise can lead to poor performance in a trained classifier. However, it is often difficult to completely avoid noise in real-world data sets, which noise might occur during measurement, labeling, or recording. In this paper we focus on dealing with mislabeled data.

This issue has been addressed previously by many researchers, especially in the field of Nearest Neighbor Classifiers [Cover and Hart, 1967], whose performance is particularly sensitive to noise. One strategy is to apply a filtering mechanism to detect and remove mislabeled samples. For example, Wilson [1972] applied a 3-NN (Nearest Neighbor) classifier as a filter, which identifies noisy data as those items that are misclassified by the 3-NN. The filtered data was then applied as the training set for a 1-NN classifier. Several researchers [Hart 1968; Gates, 1972; Dasarathy, 1980, 1991] proposed various edited versions of the nearest neighbor algorithm by removing a significant portion of the original data. They not only reduced the storage requirement for the training data, but also removed a large portion of noise. Aha et. al. [Aha and Kibler, 1989; Aha, Kibler and Albert, 1991] proposed an algorithm that identified noisy data as those items with poor classification records. Wilson and Martinez [1997, 2000] applied several pruning techniques that can reduce the size of training sets as well as remove potential noise.

This issue has also been addressed in other learning algorithms, such as the C4.5 decision tree algorithm [Quinlan, 1993]. John [1995] presented a method that first removed those instances pruned by C4.5 and then applied the filtered data to build a new tree. Gamberger et. al. [1996] proposed a noise filtering mechanism that was based on the Minimum Description Length principle and some compression measures. Brodley and Friedl [1996, 1999] applied an ensemble of classifiers (C4.5, 1-NN and a linear machine) as a filter, which identifies mislabeled data based on consensus or majority voting in the ensemble. Teng [1999, 2000] proposed a procedure to identify and correct noise based on predictions from C4.5 decision trees. In previous work we proposed neural network based methods to identify and correct mislabeled data [Zeng and Martinez, 2001], and to filter mislabeled data in data sets [Zeng and Martinez, 2003].

In this paper we present a new noise filtering method, called *soft decision tree noise filter* (SDTNF). In this method, a soft class labeling scheme (in the form of a class probability vector), instead of fixed class labeling, is used for building a sequence of C4.5 decision trees for noise filtering. Each component of a class probability vector represents the probability of a given class. A sequence of C4.5 decision trees are recursively built from a data set with soft class labeling. In each iteration, class probability vectors are learned using predictions of the previous decision tree as the learning targets. This tree building process is repeated recursively until class probability vectors for all instances converge. Noise is then identified as those items whose final learned class labels are different from their original class labels.

SDTNF has the following distinct features compared to previous methods for similar tasks. (i). In previous work [Brodley and Friedl, 1996, 1999; Teng, 1999, 2000], a data set was first divided into two disjoint subsets: a training set and a test set. The training set was applied to construct a classifier (or an ensemble of classifiers), and the constructed classifier was then applied to identify noise in the test set. However, because the training set consists of the same percentage of noise as the test set, a classifier constructed in this way may not achieve high levels of accuracy (especially for data sets with a high degree of noise). Thus it could make inaccurate predictions about the noise in the test set. In contrast, SDTNF includes all instances in the process and allows every instance to change its class label, without relying on a pre-constructed classifier. (ii). By utilizing a vector class label (instead of a binary class label), SDTNF allows a large number of hypotheses about class labeling to interact and compete with each other simultaneously, enabling them to smoothly and incrementally converge to an optimal or near-optimal labeling. This type of search strategy has been shown efficient on large solution-spaces for NP-class optimization problems [Hopfield and Tank, 1985].

We test the performance of SDTNF on 16 data sets drawn from the UCI data repository, and also compare it to the methods proposed by Brodley and Friedl [1996, 1999]. The performance of SDTNF is evaluated at different noise levels (with mislabeled classes) by testing its capability to filter mislabeled data and its capability to keep non-mislabeled data. To evaluate its impact on the quality of constructed classifiers, we compare the test set accuracies of two nearest neighbor classifiers – one constructed using a training set that includes mislabeled instances and the other constructed using the same training set that is filtered by SDTNF. Stratified 10-fold cross-validation is applied to estimate their accuracies. The results show that for most data sets, SDTNF is capable of filtering a large fraction of noise and significantly improving the performance of nearest neighbor classifiers at a wide range of noise levels. We also compare SDTNF to the consensus and majority voting methods proposed by Brodley and Friedl [1996, 1999]. The results show that on average SDTNF performs better than these two methods, especially at noise levels from 20% to 40%. SDTNF has a more efficient and balanced filtering capability in terms of filtering mislabeled data and keeping non-mislabeled data. SDTNF is also able to achieve a better improvement on the accuracy of nearest neighbor classifiers. At a noise level of 40%, the improvement on the accuracy of nearest neighbor classifiers is 13.1% by the consensus voting method and 18.7% by the majority voting method, while SDTNF is able to achieve an improvement by 31.3%.

2. SOFT DECISION TREE NOISE FILTERING ALGORITHM

Let S be an input data set in which some instances have been mislabeled. The task of SDTNF is to identify and remove those mislabeled instances and then output a filtered data set $\hat{S}$. Let α be the correctly labeled fraction and β $(= 1 - \alpha)$ the mislabeled fraction of input data set S. Let S_c be the correctly labeled subset and S_m the mislabeled subset $(S_c \cup S_m = S)$. The instances in S_c have a tendency of strengthening the regularities possessed in S, while those in S_m have a tendency of weakening the regularities due to the random nature of mislabeling. However, if the mislabeled fraction is small (i.e., $\beta << \alpha$), the trend of maintaining the regularities due to S_c will be more dominant than that due to S_m (because of significantly larger number of instances in S_c that tend to maintain the regularities in S). The strategy of SDTNF is to apply the regularities discovered by a sequence of decision trees in S_c to correct those mislabeled instances in S_m.

The format for representing data in our procedure is different from a typical one. In a typical presentation, each instance w in S has the following format:

$$w = (\mathbf{x}, y) \tag{1}$$

where $\mathbf{x} = (x_1, x_2, ..., x_f)$ is the feature vector of w, f is the number of features, and y is the class (category) label of w.

In SDTNF, a *class probability vector* is attached to each instance w in S:

$$w = (\mathbf{x}, y, \mathbf{p}) \tag{2}$$

where $\mathbf{p} = (p_1, p_2, ..., p_{|C|})$ is the class probability vector of w and $|C|$ is the number of class labels. In addition, there is an input u_i assigned for each component p_i $(i = 1, 2, ..., |C|)$ of the probability vector $\mathbf{p}$. p_i is determined by u_i through the *sigmoid* activation function:

$$p_i = \frac{1}{2}(1 + tanh(\frac{u_i}{b})) \tag{3}$$

where $b = 0.02$ is the amplification parameter that reflects the steepness of the activation function. The reason for using the *sigmoid* function as the mapping between u_i and p_i is that it is able to provide a smooth transition during the learning process. Its effectiveness has been demonstrated in the Hopfield network for solving optimization problems [Hopfield and Tank, 1985].

For each instance the class probability vector **p** is initially set to characterize the original class label. A learning procedure is applied to learn and update **p** during the building of a sequence of decision trees. After building each decision tree, the outputs of the decision tree are applied as learning targets for the class probability vector **p**. The input u_i is first updated based on the difference between p_i and the output of the decision tree, which is controlled by a learning rate for a smooth learning process. p_i is then updated from u_i according to Eq. (3) and then normalizing to 1. After each update, **p** gets closer to the outputs of the most recent decision tree which reflects the regularities in the data set. In the next iteration, a new decision tree is built based on a data set with the updated class probability vector **p**. This tree building and learning process repeats until the class probability vectors in the data set converge.

We make the following modifications to the standard C4.5 decision tree learning algorithm [Quinlan, 1993] to take into account the probabilistic labeling format of Eq. (2). The procedure for building a decision tree is similar to that described in [Quinlan, 1993]. However, the method for calculating the class frequency at each tree node (which is used to calculate the information gain) is different. In [Quinlan, 1993] class frequency $freq(C_i, S)$ at a given node t is calculated by counting the number of instances in data set S that belong to class C_i (where S is the data set that reaches the node t during the building of a decision tree). We modify the calculation of class frequency to account for information represented in the class probability vector. In our calculation, the class frequency $freq(C_i, S)$ is the sum of probabilities that belong to class C_i for all items in the data set S:

$$freq(C_i, S) = \sum_{j=1}^{|S|} p_i^j \tag{4}$$

where p_i^j is the probability of instance j belonging to class i. A corresponding modification is also made for the class distribution that is recorded at each node t.

The following explains the basic steps of SDTNF. The related parameters are set empirically based on our experiments. Our experiments show that the performance of SDTNF is quite robust over a wide range of parameter settings.

- For each instance $w = (\mathbf{x}, y, \mathbf{p})$ (where y is the initial class label), its probability vector **p** is initially set as follows. p_y (the probability for class y) is set to be a large fraction d (= 0.95), and then (1-d) is divided equally among the other ($|C| - 1$) probability components. The input u_i is then determined from p_i using the inverse *sigmoid* function.

- Build a decision tree using the modified version (as described above). Each tree is pruned using the standard pruning procedure as described in [Quinlan, 1993].

- After building a decision tree, each training item w updates its probability vector $\mathbf{p}$ using the following procedure:

  (i). Feed w into the decision tree (start at the root node) and then follow branches (according to the value of the tested attribute at each tree node) until a leaf node is reached. If w has an unknown attribute at a given node t, the path is divided and each branch is followed. In this case one branch is propagated into multiple branches with different weights. The weight for a given branch is proportional to the size of data set that passes through the branch when this decision tree was built (which is the same algorithm for handling unknown attributes as in [Quinlan, 1993]). Thus if w has an unknown attribute, the propagation of w may reach multiple leaf nodes in the tree.

  (ii). Obtain a target probability vector $\mathbf{q} = (q_1, q_2, ...q_{|C|})$ for w by using the formula:

$$q_i = \sum_{j=1}^{L} (w_j \times e_i^j) \tag{5}$$

  where L is the number of leaf nodes that are reached by w, w_j is the weight when the propagation of w reaches leaf node j (note that w_j is in the range [0, 1] when there is an unknown attribute), and e_i^j is the fraction of those items that belong to class i among all data items that reach the leaf node j when the tree was built (note that a leaf node may contain data items with different classes, depending on pruning criteria and other parameter settings, to avoid over-fitting). The probability estimation method applied here is based on simple frequency counting. There are several other probability estimation methods (for decision trees) that apply different probability smoothing procedures [Zadrozny and Elkan, 2001; Foster and Domingos 2003].

  (iii). Learning for probability vector $\mathbf{p}$ is based on its difference from the target probability vector $\mathbf{q}$. For each class i, the first step is to update the input u_i from iteration k to $k + 1$ using the formula:

$$u_i(k + 1) = u_i(k) + l_p \times (q_i(k) - p_i(k)) \tag{6}$$

  where l_p (set to 0.02 in our experiment) is the *learning rate* for the probability vector. The next step is to update p_i from u_i using Eq. (3), and then normalize it:

$$\sum_{i=1}^{|C|} p_i = 1 \tag{7}$$

(iv). The class label y for instance w $(= (\mathbf{x}, y, \mathbf{p}))$ is also updated using the following formula:

$$y = argmax_i\{p_i|(i = 1, 2, ...|C|)\} \tag{8}$$

For some data items, the updated labels may differ from the original labels. In this case, they are relabeled to the different classes. This provides a mechanism to identify mislabeled data items.

- After building each tree, check if any of following three stoppage criteria is reached: (i). The error of the decision tree is 0. (ii). The total number of iterations of building trees reaches the maximum limit (set to 50 in our experiment). (iii). If the average change $\Delta p(k)$ in the probability vector at the current tree iteration k (compared to last tree k-1) is smaller than a threshold (set to 0.03 in our experiment). $\Delta p(k)$ is defined as:

$$\Delta p(k) = \frac{\sum_{i=1}^{|T|}\sum_{j=1}^{|C|}|p_j^i(k) - p_j^i(k-1)|}{|T| \times |C|} \tag{9}$$

  where $p_j^i(k)$ is component j of the class probability vector for item i at iteration k, and $|T|$ is the size of the training set T. If $\Delta p(k)$ is very small, it means that the probability vectors are stable and therefore no further iteration is needed.

- If any of the above conditions is met, then stop; otherwise build a new tree using the data set with the updated probability vector $\mathbf{p}$ and repeat the above procedure.

The above describes the basic steps that build a sequence of decision trees. After convergence of the SDTNF algorithm, we compare the final class label to the initial label for each item. Those items whose class has been relabeled are identified as noise and removed from the data set.

The reason to use pruned trees instead of the original trees (without pruning) is that original trees can over-fit mislabeled data while pruned trees can provide a mechanism to correct mislabeled data by modifying probability vectors. Our experiments show that the performance of SDTNF varies slightly with two C4.5 parameters: the confidence level (a parameter that controls the amount of tree pruning) and minimum objects (a parameter that sets a lower bound for tree branching: once a node contains the minimum objects it will not be further branched to avoid over-fitting). In C4.5, the default value is 0.25 for the confidence level and is 2 for minimum objects. In our experiment, we set the same confidence level but set 4 for minimum objects (which gave slightly better overall performance than other parameter settings).

The complexity of this algorithm is

$$O(SDTNF) = M \times (O(C4.5Build) + |T| \times O(C4.5Run) + |T| \times |C|) \tag{10}$$

where M is the number of iterations for SDTNF to converge, $O(C4.5Build)$ is the complexity of building a decision tree for training set T, and $O(C4.5Run)$ is the complexity of running a decision tree to classify one item. $|T| \times O(C4.5Run)$ is the complexity to classify all items in T. $|T| \times |C|$ is an extra term required by SDTNF to update the probability vector $\mathbf{p}$ (with $|C|$ components) for each item in T. $O(C4.5Run)$ depends on the depth of the tree and it usually has a same order of magnitude as the number of class $|C|$ (usually in the order of 10). So $|T| \times O(C4.5Run)$ and $|T| \times |C|$ have a same order of complexity. Since the term $O(C4.5Build)$ is dominant as compared to the other terms, the complexity of SDTNF is about M times that of building a C4.5 tree. From our experiments (shown in the next section), the number of iterations M to converge is usually smaller than 10 (the average number of iterations over 16 data sets is 7.4). C4.5 has been demonstrated to be a fast and scalable algorithm in various applications. These features allow SDTNF to achieve a fast convergence and to scale to large-size data sets.

3. EXPERIMENTS

We test the performance of SDTNF on 16 data sets drawn from the UCI machine learning data repository [Merz and Murphy, 1996]. Table 1 shows the properties of the 16 tested data sets. In the table, **size** is the number of instances, **#attr** is the number of attributes, **#num** is the number of numerical (continuous) attributes, **#symb** is the number of symbolic (nominal) attributes, and **#class** is the number of output classes.

Table 1: Description of 16 UCI data sets

Data Set	size	#attr	#num	#symb	#class
Anneal	898	38	6	32	6
Australian	690	14	6	8	2
Balance	625	4	0	4	3
Car	1728	6	0	6	4
Ecoli	336	7	7	0	8
Glass	214	9	9	0	7
HeartC	303	13	5	8	2
HeartH	294	13	5	8	2
Horse	366	26	12	14	3
Iono	351	34	34	0	2
Iris	150	4	4	0	3
Pima	768	8	8	0	2
Soybean	683	35	0	35	19
Vehicle	846	18	18	0	4
Voting	435	16	0	16	2
Vowel	990	10	10	0	11

We apply a stratified 10-fold cross-validation to estimate the performance for each parameter setting. In a 10-fold cross-validation, a data set is randomly divided into 10 folds with an equal number of items. Each cross-validation consists of 10 runs. In each run, 9 folds of data are used as the training set S and the other fold as the test set T. In the next run, the folds are rotated such that each of 10 runs uses a different test set. The result of a 10-fold cross-validation is the average of the 10 runs. A stratified version [Breiman et. al. 1984; Kohavi, 1995] of 10-fold cross-validation (in which 10 folds are divided on per-class bases so that each fold has a class distribution similar to that in the original data set) is applied to achieve a more reliable evaluation, especially for small data sets.

To quantitatively estimate how the performance of SDTNF varies with the noise level, we artificially mislabel a part of the original training data. We obtain a mislabeled training set S_m by mislabeling a fraction β of data instances in S using the following process. For each class i ($i = 1, 2, ..., |C|$), βN_i instances (N_i is the total number of instances of class i) are randomly mislabeled to one of the other ($|C|$-1) classes (i.e., classes 1, 2,...,i-1, i+1,...,$|C|$). Among the βN_i instances, the number of instances labeled to class j is proportional to the population of class j. SDTNF is then applied to filter noise in S_m.

Two error terms E_1 and E_2 are applied to measure the performance of SDTNF. E_1 is defined by

$$E_1 = \frac{M_1}{(1-\beta)|T|} \tag{11}$$

where M_1 is the number of items that are not mislabeled but are mistakenly identified by SDTNF as mislabeled data. $(1-\beta)|T|$ is the total number of items that are not mislabeled ($|T|$ is total number of items in a data set). Thus E_1 represents the probability of mistakenly identifying data items that are actually not mislabeled.

A second error term E_2 is defined as

$$E_2 = \frac{M_2}{\beta|T|} \tag{12}$$

where M_2 is the number of items that are mislabeled but are not identified by SDTNF. $\beta|T|$ is the total number of items that are mislabeled. Thus E_2 represents the probability of not being able to identify mislabeled data.

The performance of SDTNF is also evaluated by comparing the test-set accuracies of two classifiers using the nearest neighbor rule [Cover and Hart, 1967]. One classifier is based on a training set that has been filtered by SDTNF, while the other classifier is based on the original training set without using SDTNF as a filter (both using 1-nearest neighbor). Both classifiers use the same T as the test set. The reason to choose the nearest neighbor classifier for our experiment is that its performance is particularly sensitive to noise.

We also compare SDTNF to the methods proposed by Brodley and Friedl [1996, 1999]. In their methods, the training set is divided into n parts. An ensemble of classifiers (C4.5, 1-NN and a linear machine) is trained on n-1 parts and is applied for filtering noise on the other part (with n rotations of the training data). One method they proposed is consensus voting: if all three classifiers in the ensemble classify an item to a class that is different from its original label, then the item is identified as noise and is removed from the training set. Another proposed method is majority voting (requiring at least two votes). We implement their algorithms based on their description in [Brodley and Friedl, 1996; 1999].

Tables 2 and 3 show the results (averaged over 10 runs of stratified 10-fold cross-validation) for the data set *Ecoli* and *Vowel*. The first row ($NoiseLevel$) is the targeted noise level (percentage of mislabeled data in the training set). The second row ($ActualNoise$) is the actual noise level, which is slightly different from the targeted level due to limited data set sizes and the difference in class distributions. $Iterations$ is the number of iterations for convergence. $TreeSize$ is the number of nodes in the decision trees. In the table, SDTNF represents the proposed filtering algorithm, while CF and MF are the consensus and majority filtering methods proposed by Brodley and Friedl [1996, 1999]. E_1 and E_2 are the error terms defined in Eq.(11) and Eq.(12). For a noise level of 0%, E_2 is not applicable (as denoted by "*"). 1-NN(Acc) is the accuracy of a 1-nearest neighbor classifier (1-NN) using the original training set without any noise filtering. SDTNF(Acc) is the accuracy of 1-NN using a training set that is filtered by SDTNF. CF(Acc) and MF(Acc) are the accuracies of 1-NN using a training set that is filtered by the consensus and majority filtering methods respectively. Each reported accuracy is in percentage, along with the confidence interval of 95%. The last column is the average over all noise levels.

Table 2: Result for the data set **Ecoli**.

Noise Level Actual Noise	0 0.0	5 4.7	10 10.1	20 19.8	30 30.2	40 40.2	**Avg.**
Iterations	3.9	5.6	8.0	10.1	10.2	11.7	8.3
Tree Size	23.4	22.6	21.8	21.0	22.4	29.6	23.5
SDTNF(E_1)	0.063	0.088	0.097	0.113	0.116	0.142	0.103
CF(E_1)	0.071	0.065	0.054	0.041	0.045	0.043	0.053
MF(E_1)	0.145	0.138	0.154	0.165	0.195	0.289	0.181
SDTNF(E_2)	$*$	0.147	0.105	0.130	0.185	0.228	0.159
CF(E_2)	$*$	0.281	0.292	0.443	0.621	0.750	0.477
MF(E_2)	$*$	0.105	0.066	0.105	0.191	0.271	0.148
1-NN(Acc)	80.7 ± 4.5	78.0 ± 5.0	76.5 ± 6.1	67.9 ± 8.1	61.6 ± 5.3	53.0 ± 6.3	69.6
SDTNF(Acc)	85.4 ± 2.5	85.1 ± 4.1	84.8 ± 4.1	82.1 ± 3.6	81.2 ± 3.1	72.9 ± 4.0	81.9
CF(Acc)	86.9 ± 3.7	86.0 ± 4.0	82.7 ± 5.3	79.5 ± 5.3	70.2 ± 6.3	57.4 ± 5.6	77.1
MF(Acc)	84.5 ± 3.1	85.1 ± 4.3	83.6 ± 6.4	81.0 ± 4.9	73.8 ± 4.8	61.9 ± 5.1	78.3

Table 3: Result for the data set **Vowel**.

Noise Level Actual Noise	0 0.0	5 4.9	10 9.9	20 19.8	30 29.6	40 39.5	 **Avg.**
Iterations	1.0	4.5	7.0	9.3	11.5	12.3	7.6
Tree Size	146.8	151.4	144.2	153.2	163.4	178.6	156.3
SDTNF(E_1)	0.000	0.041	0.066	0.078	0.088	0.104	0.063
CF(E_1)	0.002	0.002	0.004	0.008	0.008	0.014	0.006
MF(E_1)	0.065	0.072	0.091	0.111	0.136	0.178	0.109
SDTNF(E_2)	$*$	0.291	0.170	0.168	0.219	0.280	0.226
CF(E_2)	$*$	0.541	0.623	0.695	0.778	0.853	0.698
MF(E_2)	$*$	0.111	0.140	0.195	0.283	0.333	0.212
1-NN(Acc)	98.9 ± 0.8	93.7 ± 2.2	90.1 ± 2.5	79.3 ± 3.3	69.6 ± 3.1	59.1 ± 2.6	81.8
SDTNF(Acc)	98.9 ± 0.8	94.5 ± 2.4	94.4 ± 2.6	91.5 ± 2.4	84.9 ± 3.3	79.2 ± 2.5	90.6
CF(Acc)	98.1 ± 0.5	96.2 ± 1.6	92.1 ± 3.0	82.2 ± 2.5	75.1 ± 3.3	63.3 ± 2.7	84.5
MF(Acc)	89.7 ± 2.5	89.1 ± 2.2	86.9 ± 3.6	83.2 ± 2.2	75.2 ± 3.3	66.7 ± 4.7	81.8

The results for the other 14 data sets are shown in Appendix A. Table 4 shows the average result over the 16 data sets. The results show that all the three noise filtering algorithms (SDTNF, CF, MF) are able to identify a large amount of mislabeled data and improve the nearest neighbor classifiers for most data sets at various noise levels. But they behave differently in terms of error terms and accuracy.

From the last column (average over all 16 data sets and over all noise levels) in Table 4, we can see the following patterns in terms of errors E_1 (probability of mistakenly identifying data that is not mislabeled) and E_2 (probability of not identifying data that is mislabeled). (i). The consensus voting algorithm CF has a small E_1 (0.054) but a large E_2 (0.525). It tends to be conservative in identifying mislabeled data because of its consensus nature. (ii). In contrast, the majority voting algorithm MF has a larger E_1 (0.179) but a smaller E_2 (0.196). It has a stronger tendency to identify a data item as mislabeled than CF since it only needs majority votes (instead of consensus votes). (iii). SDTNF has errors E_1 (0.130) and E_2 (0.271) that are between those for CF and MF.

In terms of improvement on accuracy of the nearest neighbor classifiers, SDTNF performs slightly better than CF and MF at noise levels of 0%, 5% and 10%, as shown in Table 4. At noise levels of 20%, 30%, and 40%, SDTNF performs significantly better than CF and MF. This difference in performance is correlated to errors E_1 and E_2. For example, at a noise level of 40%, CF has a very large E_2 (0.716) although it has a very small E_1 (0.071). A large value of 0.716 for E_2 means that CF only filters out 28.4% of all mislabeled data (40% of the whole data set). The large amount of unfiltered mislabeled data significantly limits its ability to improve the nearest neighbor classifiers. Although MF has a smaller E_2 (0.302) as compared to E_2 (0.622) from CF at the noise level 40%, it has a large E_1 (0.302). This means that MF mistakenly filters out 23.5% of all non-mislabeled data (60% of the whole data set), which significantly affects its ability to improve the nearest neighbor classifiers. In comparison, SDTNF has a more balanced E_1 (0.184) and E_2 (0.326) that enable SDTNF to achieve better performance than CF and MF in terms of improving 1-NN. At the noise level 40%, SDTNF is able to achieve an improvement on the accuracy of 1-NN by 31.3% (70.9% vs. 54.0%), in comparison to an improvement of 13.1% (61.1% vs. 54.0%) by CF and an improvement of 18.7% (64.1% vs. 50.0%) by MF.

Table 4: Average result over 16 data sets.

Noise Level	0	5	10	20	30	40	**Avg.**
Iterations	5.4	6.2	7.0	8.0	8.6	8.8	7.4
Tree Size	45.1	45.0	42.8	42.3	42.9	46.7	44.1
SDTNF(E_1)	0.101	0.108	0.115	0.127	0.145	0.184	0.130
CF(E_1)	0.047	0.048	0.049	0.053	0.056	0.071	0.054
MF(E_1)	0.142	0.150	0.163	0.191	0.235	0.302	0.197
SDTNF(E_2)	$*$	0.273	0.269	0.225	0.260	0.326	0.271
CF(E_2)	$*$	0.364	0.416	0.509	0.622	0.716	0.525
MF(E_2)	$*$	0.128	0.147	0.174	0.228	0.302	0.196
1-NN(Acc)	80.7	77.9	74.2	67.0	59.1	54.0	68.8
SDTNF(Acc)	82.3	81.2	80.3	79.0	75.9	70.9	78.3
CF(Acc)	81.3	81.1	79.4	75.6	68.6	61.1	74.5
MF(Acc)	79.9	80.0	78.5	76.5	71.5	64.1	75.1

From the results of 16 tested data sets, we can also see that the three filtering methods perform differently on different data sets in terms of the improvement on the accuracy of 1-NN. For eight data sets (*Anneal*, *HeartC*, *HeartH*, *Horse*, *Pima*, *Soybean*, *Vehicle*, *Voting*) SDTNF has similar performance as CF and MF in terms of the average accuracy (over different noise levels), as shown in the last column of the table for each data set. In these data sets, the accuracy difference between SDTNF and the best of CF and MF is less than 3.0%. On the other eight data sets (*Australian*, *Balance*, *Car*, *Ecoli*, *Glass*, *Iono*, *Iris*, *Vowel*), SDTNF performs significantly better (more than 3.0%) than the best of CF and MF.

One explanation for the performance difference in filtering capabilities among the three methods is as follows. In CF and MF, a data set is divided into two subsets: S_1 and S_2. S_1 is applied for constructing an ensemble of classifiers (which is then utilized for predicting noise in S_2). Because the training data S_1 contains the same level of noise as in S_2, it can generate a trained classifier with degraded capacity for making accurate noise predictions (especially at a high level of noise). In contrast, SDTNF includes all instances (in both S_1 and S_2) in the process and allows every instance to change its class label, without relying on a pre-constructed classifier. Another factor that may contribute to the performance difference is that SDTNF uses a soft class label (class probability vector) instead of a hard (binary) class label. This allows a large number of hypotheses about class labeling to interact and compete with each other simultaneously. SDTNF can provide smoother convergence and increase the opportunity for accurate labeling.

Another observation is that SDTNF can significantly improve the accuracy of the nearest neighbor classifiers for some data sets even at a noise level of 0%. For example, SDTNF is able to improve the accuracy of 1-NN significantly (by more than 3%) on several data sets (*Australian*, *Balance*, *Ecoli*, *HeartH*, *Pima*) at a noise level of 0%. This shows that, even without artificially adding mislabeled data, SDTNF still has capability to filter potential mislabeled data in some data sets and improve the performance of the nearest neighbor classifiers.

From table 2, we can see that SDTNF needs only an average of 7.4 iterations to converge. Because SDTNF is based on C4.5 (which is a relatively fast and scalable algorithm), it has the ability to achieve a reasonably fast convergence with a good scalability to large-size data sets.

4. SUMMARY

In summary, we have presented a decision tree based approach – SDTNF – to filter noise in data sets. In this approach, a sequence of decision trees are built whose outputs are applied as the learning targets for identifying mislabeled data. In SDTNF, a soft class label (class probability vector) is applied to each instance and its values are learned during the process of building decision trees. The learning is based on the difference between the class probability vector and outputs of decision trees, and is controlled by a learning rate. This learning procedure provides a mechanism for aligning the class probability vector with the decision tree outputs, which can be applied for identifying and filtering mislabeled data.

We have tested the performance of SDTNF on 16 data sets drawn from the UCI data repository by evaluating its ability to filter mislabeled data as well as its ability to keep non-mislabeled data. We also evaluate its performance by evaluating and comparing the accuracies of two versions of nearest neighbor classifiers, one using the training set filtered by SDTNF and the other using the training set without filtering. The results show that SDTNF is able to filter a large portion of mislabeled data and significantly improve the performance of a nearest neighbor classifier over a wide range of noise levels.

We also compared SDTNF to the consensus and majority voting methods proposed by Brodley and Friedl [1996; 1999]. The results show that SDTNF compared favorably to these methods, especially at high noise levels.

APPENDIX A. ADDITIONAL RESULTS

Table 5: Result for the data set **Anneal**.

Noise Level Actual Noise	0 0.0	5 5.0	10 10.1	20 19.9	30 30.1	40 40.0	 **Avg.**
Iterations	4.5	7.7	8.4	9.5	8.4	6.5	7.5
Tree Size	66.4	51.3	48.0	45.3	19.0	6.8	39.5
SDTNF(E_1)	0.045	0.065	0.075	0.096	0.158	0.192	0.105
CF(E_1)	0.004	0.005	0.012	0.018	0.033	0.052	0.021
MF(E_1)	0.035	0.041	0.055	0.079	0.121	0.173	0.084
SDTNF(E_2)	$*$	0.072	0.111	0.109	0.156	0.175	0.125
CF(E_2)	$*$	0.175	0.237	0.333	0.457	0.563	0.353
MF(E_2)	$*$	0.030	0.050	0.063	0.118	0.161	0.084
1-NN(Acc)	95.7 ± 1.7	91.1 ± 1.9	85.1 ± 3.4	78.2 ± 3.4	66.8 ± 1.8	59.9 ± 3.6	79.5
SDTNF(Acc)	93.8 ± 1.6	92.3 ± 1.8	91.1 ± 0.9	89.8 ± 1.6	81.5 ± 2.0	79.4 ± 1.1	88.0
CF(Acc)	95.1 ± 1.8	94.7 ± 1.6	92.1 ± 1.9	89.6 ± 2.8	81.4 ± 2.8	74.2 ± 3.6	87.9
MF(Acc)	94.0 ± 1.7	93.7 ± 1.6	91.4 ± 1.8	89.8 ± 1.3	85.4 ± 1.7	79.2 ± 1.9	88.9

Table 6: Result for the data set **Australian**.

Noise Level Actual Noise	0 0.0	5 5.0	10 10.1	20 20.0	30 30.1	40 40.0	 **Avg.**
Iterations	5.3	6.6	7.0	5.8	8.0	7.7	6.7
Tree Size	20.3	20.2	21.8	14.9	40.3	69.1	31.1
SDTNF(E_1)	0.109	0.117	0.110	0.126	0.124	0.165	0.125
CF(E_1)	0.072	0.072	0.069	0.074	0.076	0.120	0.081
MF(E_1)	0.143	0.145	0.149	0.184	0.280	0.377	0.213
SDTNF(E_2)	$*$	0.158	0.185	0.190	0.275	0.469	0.255
CF(E_2)	$*$	0.294	0.367	0.466	0.615	0.741	0.497
MF(E_2)	$*$	0.094	0.144	0.186	0.260	0.385	0.214
1-NN(Acc)	81.7 ± 3.1	76.5 ± 3.3	74.9 ± 4.5	69.9 ± 2.9	64.5 ± 3.5	55.2 ± 3.3	70.5
SDTNF(Acc)	86.4 ± 1.8	84.5 ± 3.8	85.2 ± 3.7	84.9 ± 3.4	79.9 ± 4.0	69.9 ± 8.7	81.8
CF(Acc)	84.6 ± 3.1	85.4 ± 2.6	84.3 ± 3.3	80.6 ± 4.4	73.9 ± 3.0	59.4 ± 4.4	78.0
MF(Acc)	84.3 ± 3.0	82.8 ± 3.3	85.1 ± 3.7	81.9 ± 2.9	74.8 ± 4.0	61.7 ± 3.7	78.4

Table 7: Result for the data set **Balance**.

Noise Level Actual Noise	0 0.0	5 5.0	10 10.0	20 20.1	30 30.1	40 40.2	 **Avg.**
Iterations	9.3	9.7	9.4	8.7	8.5	8.8	9.1
Tree Size	41.0	43.5	36.0	30.5	28.0	33.0	35.3
SDTNF(E_1)	0.238	0.233	0.243	0.266	0.276	0.279	0.256
CF(E_1)	0.004	0.019	0.038	0.065	0.069	0.070	0.044
MF(E_1)	0.145	0.185	0.216	0.263	0.317	0.373	0.250
SDTNF(E_2)	$*$	0.379	0.353	0.357	0.355	0.396	0.368
CF(E_2)	$*$	0.575	0.579	0.619	0.700	0.777	0.650
MF(E_2)	$*$	0.146	0.182	0.218	0.270	0.334	0.230
1-NN(Acc)	63.8 ± 5.6	62.1 ± 4.0	58.2 ± 5.8	56.8 ± 2.9	49.9 ± 5.5	48.2 ± 6.2	56.5
SDTNF(Acc)	68.2 ± 5.4	69.4 ± 4.5	66.4 ± 4.4	63.2 ± 2.8	60.6 ± 6.0	59.7 ± 5.4	64.6
CF(Acc)	60.3 ± 3.0	61.8 ± 2.5	61.6 ± 3.8	57.3 ± 2.0	51.5 ± 2.3	47.4 ± 2.9	56.6
MF(Acc)	60.0 ± 3.3	61.6 ± 3.4	63.2 ± 4.8	63.2 ± 3.3	59.2 ± 6.2	58.7 ± 3.7	61.0

Table 8: Result for the data set **Car**.

Noise Level Actual Noise	0 0.0	5 5.0	10 10.0	20 20.0	30 30.1	40 40.0	 **Avg.**
Iterations	6.0	5.8	7.7	9.0	9.0	9.0	7.8
Tree Size	147.4	139.4	133.4	123.6	115.9	103.1	127.1
SDTNF(E_1)	0.039	0.045	0.051	0.063	0.082	0.140	0.070
CF(E_1)	0.007	0.011	0.022	0.035	0.049	0.085	0.035
MF(E_1)	0.069	0.094	0.144	0.188	0.207	0.257	0.160
SDTNF(E_2)	$*$	0.097	0.112	0.136	0.169	0.245	0.152
CF(E_2)	$*$	0.322	0.340	0.375	0.450	0.535	0.404
MF(E_2)	$*$	0.094	0.155	0.180	0.190	0.240	0.172
1-NN(Acc)	78.9 ± 1.7	74.2 ± 1.4	70.9 ± 2.1	63.4 ± 3.0	57.2 ± 3.6	53.9 ± 3.4	66.4
SDTNF(Acc)	78.5 ± 1.7	77.2 ± 1.2	77.3 ± 2.3	76.0 ± 2.1	75.3 ± 2.6	72.7 ± 1.9	76.2
CF(Acc)	75.5 ± 1.0	76.0 ± 1.8	74.4 ± 1.7	74.0 ± 1.0	72.0 ± 1.5	70.4 ± 2.0	73.7
MF(Acc)	74.1 ± 0.6	74.7 ± 2.3	73.4 ± 1.8	72.3 ± 0.6	70.7 ± 1.6	70.6 ± 1.5	72.6

Table 9: Result for the data set **Glass**.

Noise Level Actual Noise	0 0.0	5 4.7	10 10.4	20 20.2	30 30.2	40 39.6	 **Avg.**
Iterations	7.7	7.8	9.9	9.8	12.0	12.0	9.9
Tree Size	28.0	29.2	29.0	30.6	27.8	32.4	29.5
SDTNF(E_1)	0.149	0.151	0.148	0.155	0.163	0.206	0.162
CF(E_1)	0.070	0.072	0.075	0.073	0.071	0.069	0.072
MF(E_1)	0.264	0.273	0.272	0.313	0.332	0.371	0.304
SDTNF(E_2)	$*$	0.400	0.345	0.354	0.357	0.407	0.373
CF(E_2)	$*$	0.589	0.595	0.687	0.758	0.816	0.689
MF(E_2)	$*$	0.222	0.215	0.226	0.275	0.334	0.254
1-NN(Acc)	67.8 ± 7.3	67.8 ± 10.7	65.4 ± 6.9	55.6 ± 8.3	49.5 ± 10.1	40.7 ± 5.8	57.8
SDTNF(Acc)	67.3 ± 6.4	66.4 ± 10.1	67.3 ± 6.3	64.5 ± 8.5	62.1 ± 9.2	53.3 ± 5.8	63.5
CF(Acc)	65.0 ± 7.8	65.9 ± 10.7	67.8 ± 6.1	57.9 ± 7.5	54.7 ± 8.7	44.9 ± 8.7	59.4
MF(Acc)	62.6 ± 4.2	67.8 ± 7.7	64.5 ± 6.8	61.2 ± 6.0	57.0 ± 5.4	45.8 ± 9.3	59.8

Table 10: Result for the data set **HeartC**.

Noise Level Actual Noise	0 0.0	5 4.8	10 10.3	20 20.0	30 30.1	40 40.0	 **Avg.**
Iterations	6.7	7.0	7.0	7.0	8.0	8.2	7.3
Tree Size	32.1	28.9	27.9	31.7	34.9	35.8	31.9
SDTNF(E_1)	0.133	0.147	0.150	0.143	0.146	0.210	0.155
CF(E_1)	0.084	0.088	0.075	0.088	0.113	0.133	0.097
MF(E_1)	0.198	0.219	0.217	0.257	0.317	0.416	0.271
SDTNF(E_2)	$*$	0.377	0.407	0.468	0.459	0.569	0.456
CF(E_2)	$*$	0.362	0.471	0.601	0.663	0.750	0.569
MF(E_2)	$*$	0.185	0.246	0.288	0.290	0.414	0.285
1-NN(Acc)	77.2 ± 6.7	74.9 ± 5.5	75.6 ± 4.7	67.0 ± 6.8	55.1 ± 7.9	58.7 ± 5.2	68.1
SDTNF(Acc)	79.2 ± 4.2	78.5 ± 4.9	77.9 ± 3.3	74.3 ± 4.1	68.3 ± 5.4	65.7 ± 6.5	74.0
CF(Acc)	79.5 ± 5.5	78.9 ± 4.6	74.6 ± 3.9	75.2 ± 5.0	60.1 ± 7.7	61.4 ± 7.0	71.6
MF(Acc)	78.5 ± 3.7	77.9 ± 4.6	70.0 ± 5.4	73.3 ± 3.4	66.0 ± 5.9	56.4 ± 6.6	70.3

Table 11: Result for the data set **HeartH**.

Noise Level Actual Noise	0 0.0	5 5.0	10 10.2	20 20.0	30 30.2	40 40.1	 **Avg.**
Iterations	4.9	6.1	5.0	5.7	6.5	6.5	5.8
Tree Size	7.8	10.8	7.8	12.6	11.9	12.7	10.6
SDTNF(E_1)	0.164	0.155	0.163	0.159	0.185	0.224	0.175
CF(E_1)	0.095	0.085	0.083	0.090	0.103	0.086	0.090
MF(E_1)	0.188	0.186	0.193	0.235	0.296	0.348	0.241
SDTNF(E_2)	$*$	0.265	0.296	0.292	0.310	0.443	0.321
CF(E_2)	$*$	0.364	0.430	0.523	0.620	0.715	0.530
MF(E_2)	$*$	0.214	0.222	0.208	0.270	0.375	0.258
1-NN(Acc)	76.5 ± 6.8	75.2 ± 5.1	74.1 ± 3.1	64.6 ± 5.5	63.9 ± 5.8	55.4 ± 8.6	68.3
SDTNF(Acc)	81.6 ± 7.4	80.6 ± 5.0	80.6 ± 4.5	77.9 ± 6.1	75.2 ± 5.4	69.0 ± 9.6	77.5
CF(Acc)	80.3 ± 7.2	80.6 ± 5.4	78.6 ± 3.2	75.5 ± 5.8	73.1 ± 6.6	63.9 ± 7.9	75.3
MF(Acc)	81.3 ± 7.7	80.6 ± 5.9	77.2 ± 4.4	78.2 ± 6.0	68.7 ± 4.7	68.0 ± 4.8	75.7

Table 12: Result for the data set **Horse**.

Noise Level Actual Noise	0 0.0	5 4.9	10 10.0	20 19.9	30 30.1	40 40.0	 **Avg.**
Iterations	8.8	8.7	9.8	10.1	8.6	8.6	9.1
Tree Size	24.7	24.0	22.4	24.7	18.7	11.3	21.0
SDTNF(E_1)	0.216	0.231	0.238	0.249	0.280	0.333	0.258
CF(E_1)	0.090	0.091	0.089	0.096	0.081	0.104	0.092
MF(E_1)	0.294	0.292	0.307	0.343	0.358	0.383	0.329
SDTNF(E_2)	$*$	0.312	0.306	0.328	0.332	0.334	0.322
CF(E_2)	$*$	0.575	0.615	0.658	0.742	0.763	0.671
MF(E_2)	$*$	0.250	0.252	0.273	0.312	0.343	0.286
1-NN(Acc)	64.5 ± 6.2	63.4 ± 6.8	59.6 ± 3.5	57.9 ± 6.3	48.6 ± 5.6	47.0 ± 6.2	56.8
SDTNF(Acc)	66.1 ± 3.5	64.8 ± 3.8	65.0 ± 5.4	62.3 ± 3.9	61.2 ± 5.5	61.7 ± 3.8	63.5
CF(Acc)	64.8 ± 6.0	64.5 ± 5.7	61.5 ± 2.8	62.3 ± 6.5	56.3 ± 8.0	53.8 ± 7.1	60.5
MF(Acc)	65.3 ± 5.2	64.5 ± 5.0	63.4 ± 3.8	57.9 ± 4.7	56.3 ± 5.2	54.9 ± 5.0	60.4

Table 13: Result for the data set **Iono**.

Noise Level Actual Noise	0 0.0	5 5.1	10 9.8	20 20.1	30 30.1	40 40.0	 **Avg.**
Iterations	4.4	4.9	4.5	6.5	5.7	6.4	5.4
Tree Size	14.0	14.2	11.6	11.0	15.0	9.0	12.5
SDTNF(E_1)	0.053	0.052	0.068	0.076	0.119	0.242	0.102
CF(E_1)	0.039	0.040	0.038	0.048	0.063	0.093	0.053
MF(E_1)	0.098	0.113	0.123	0.158	0.231	0.321	0.174
SDTNF(E_2)	$*$	0.237	0.216	0.239	0.317	0.340	0.270
CF(E_2)	$*$	0.262	0.332	0.446	0.561	0.681	0.456
MF(E_2)	$*$	0.138	0.126	0.164	0.219	0.297	0.189
1-NN(Acc)	86.9 ± 2.0	84.9 ± 5.0	80.1 ± 4.0	69.2 ± 7.3	63.5 ± 4.6	55.6 ± 3.2	73.4
SDTNF(Acc)	87.2 ± 3.1	86.0 ± 4.1	84.0 ± 3.5	84.0 ± 3.2	82.3 ± 5.4	70.4 ± 3.9	82.3
CF(Acc)	86.9 ± 2.2	86.3 ± 3.4	85.2 ± 3.2	83.2 ± 4.6	73.5 ± 3.7	59.0 ± 5.8	79.0
MF(Acc)	85.5 ± 2.6	84.6 ± 2.6	82.6 ± 4.8	78.3 ± 5.2	75.5 ± 3.4	66.7 ± 5.7	78.9

Table 14: Result for the data set **Iris**.

Noise Level Actual Noise	0 0.0	5 4.4	10 11.1	20 20.0	30 31.1	40 40.0	 **Avg.**
Iterations	4.4	3.9	4.6	6.1	6.4	6.2	5.3
Tree Size	6.6	6.6	6.2	6.0	8.2	9.0	7.1
SDTNF(E_1)	0.024	0.024	0.037	0.029	0.032	0.064	0.035
CF(E_1)	0.016	0.022	0.022	0.022	0.019	0.043	0.024
MF(E_1)	0.049	0.050	0.065	0.093	0.147	0.258	0.110
SDTNF(E_2)	$*$	0.017	0.087	0.085	0.117	0.181	0.097
CF(E_2)	$*$	0.050	0.200	0.381	0.583	0.654	0.374
MF(E_2)	$*$	0.000	0.013	0.056	0.140	0.206	0.083
1-NN(Acc)	95.3 ± 3.9	92.7 ± 2.7	83.3 ± 5.6	79.3 ± 6.1	55.3 ± 6.0	62.0 ± 7.1	78.0
SDTNF(Acc)	96.0 ± 3.3	95.3 ± 2.3	94.0 ± 4.2	92.7 ± 6.1	91.3 ± 5.5	88.7 ± 6.8	93.0
CF(Acc)	97.3 ± 3.3	95.3 ± 2.3	95.3 ± 3.9	88.7 ± 6.0	75.3 ± 6.4	69.3 ± 5.1	86.9
MF(Acc)	97.3 ± 3.3	94.7 ± 4.4	93.3 ± 5.9	90.7 ± 5.6	84.0 ± 5.1	68.7 ± 13.1	88.1

Table 15: Result for the data set **Pima**.

Noise Level Actual Noise	0 0.0	5 5.1	10 10.0	20 20.0	30 30.0	40 40.0	 **Avg.**
Iterations	6.0	6.5	5.8	5.2	6.9	6.7	6.2
Tree Size	14.4	13.2	13.4	6.0	9.2	2.0	9.7
SDTNF(E_1)	0.203	0.218	0.211	0.237	0.272	0.325	0.244
CF(E_1)	0.108	0.109	0.109	0.103	0.081	0.107	0.103
MF(E_1)	0.245	0.253	0.265	0.275	0.331	0.393	0.294
SDTNF(E_2)	$*$	0.340	0.317	0.296	0.379	0.362	0.339
CF(E_2)	$*$	0.483	0.516	0.588	0.696	0.781	0.613
MF(E_2)	$*$	0.237	0.259	0.280	0.329	0.384	0.298
1-NN(Acc)	70.6 ± 4.0	68.4 ± 3.5	64.8 ± 3.4	60.7 ± 3.1	55.5 ± 2.5	50.8 ± 4.2	61.8
SDTNF(Acc)	74.5 ± 2.9	75.3 ± 3.1	74.7 ± 2.8	74.5 ± 3.2	67.6 ± 4.5	64.7 ± 5.5	71.9
CF(Acc)	74.7 ± 2.7	73.2 ± 2.9	72.9 ± 3.1	67.1 ± 4.4	63.2 ± 3.1	57.4 ± 5.3	68.1
MF(Acc)	72.7 ± 2.5	73.6 ± 3.0	72.5 ± 4.8	71.5 ± 3.6	67.6 ± 2.5	60.7 ± 6.7	69.8

Table 16: Result for the data set **Soybean**.

Noise Level Actual Noise	0 0.0	5 5.2	10 10.2	20 20.6	30 29.6	40 39.9	 **Avg.**
Iterations	1.0	1.0	1.0	5.7	7.2	9.1	4.2
Tree Size	66.4	71.8	74.3	70.7	72.3	74.3	71.6
SDTNF(E_1)	0.000	0.000	0.000	0.033	0.048	0.057	0.023
CF(E_1)	0.021	0.019	0.015	0.018	0.015	0.015	0.017
MF(E_1)	0.062	0.077	0.074	0.093	0.105	0.126	0.090
SDTNF(E_2)	$*$	1.000	1.000	0.096	0.089	0.092	0.455
CF(E_2)	$*$	0.181	0.255	0.407	0.504	0.616	0.393
MF(E_2)	$*$	0.038	0.043	0.093	0.126	0.186	0.097
1-NN(Acc)	90.0 ± 2.1	85.9 ± 2.5	81.3 ± 3.0	71.0 ± 4.6	61.6 ± 3.1	53.1 ± 4.3	73.8
SDTNF(Acc)	90.0 ± 2.1	85.9 ± 2.5	81.3 ± 3.0	89.2 ± 3.4	88.0 ± 3.2	87.0 ± 3.0	86.9
CF(Acc)	89.2 ± 2.9	90.2 ± 1.7	89.0 ± 2.5	84.0 ± 2.1	75.5 ± 4.0	68.5 ± 4.5	82.7
MF(Acc)	88.0 ± 2.6	88.4 ± 2.1	88.0 ± 3.6	86.1 ± 2.0	81.7 ± 3.0	74.7 ± 5.3	84.5

Table 17: Result for the data set **Vehicle**.

Noise Level	0	5	10	20	30	40	
Actual Noise	0.0	5.1	10.1	20.0	30.1	40.0	**Avg.**
Iterations	8.0	8.6	10.6	12.7	13.9	14.0	11.3
Tree Size	74.2	84.6	81.0	88.2	89.2	117.0	89.0
SDTNF(E_1)	0.143	0.131	0.149	0.163	0.180	0.180	0.158
CF(E_1)	0.050	0.047	0.047	0.043	0.041	0.040	0.045
MF(E_1)	0.227	0.221	0.222	0.234	0.251	0.297	0.242
SDTNF(E_2)	$*$	0.213	0.222	0.281	0.313	0.412	0.288
CF(E_2)	$*$	0.569	0.577	0.638	0.735	0.795	0.663
MF(E_2)	$*$	0.123	0.174	0.184	0.242	0.297	0.204
1-NN(Acc)	70.6 ± 4.1	67.8 ± 3.1	64.7 ± 3.0	56.0 ± 4.1	53.4 ± 2.9	48.7 ± 3.4	60.2
SDTNF(Acc)	70.7 ± 3.6	70.4 ± 3.5	69.3 ± 3.0	65.4 ± 4.5	63.1 ± 3.3	58.0 ± 4.9	66.2
CF(Acc)	70.0 ± 3.7	68.9 ± 2.8	66.8 ± 3.3	62.2 ± 4.6	58.9 ± 2.5	54.4 ± 2.8	63.5
MF(Acc)	68.6 ± 4.2	67.7 ± 3.3	67.8 ± 3.8	64.4 ± 4.6	61.6 ± 2.7	57.2 ± 3.6	64.5

Table 18: Result for the data set **Voting**.

Noise Level	0	5	10	20	30	40	
Actual Noise	0.0	5.1	10.0	19.9	29.9	39.9	**Avg.**
Iterations	5.2	5.0	5.9	6.2	7.2	7.8	6.2
Tree Size	8.8	7.8	6.4	6.2	11.0	22.8	10.5
SDTNF(E_1)	0.032	0.034	0.037	0.043	0.043	0.088	0.046
CF(E_1)	0.022	0.023	0.026	0.026	0.024	0.057	0.030
MF(E_1)	0.045	0.048	0.053	0.072	0.138	0.273	0.105
SDTNF(E_2)	$*$	0.065	0.067	0.073	0.130	0.280	0.123
CF(E_2)	$*$	0.200	0.231	0.286	0.472	0.662	0.370
MF(E_2)	$*$	0.065	0.062	0.064	0.140	0.266	0.119
1-NN(Acc)	92.6 ± 2.8	89.0 ± 3.4	83.0 ± 4.2	74.9 ± 7.2	70.3 ± 5.0	62.5 ± 4.2	78.7
SDTNF(Acc)	92.4 ± 2.5	92.6 ± 3.0	91.5 ± 3.2	91.5 ± 2.9	91.5 ± 2.3	81.6 ± 7.5	90.2
CF(Acc)	92.2 ± 2.5	92.9 ± 3.0	92.2 ± 3.0	90.6 ± 3.7	83.2 ± 5.9	72.4 ± 5.2	87.3
MF(Acc)	91.3 ± 3.1	93.3 ± 2.7	92.6 ± 3.1	91.7 ± 3.6	85.7 ± 2.6	74.3 ± 6.6	88.1

REFERENCES

Aha, D. W. and Kibler, D. 1989. Noise-tolerant instance-based learning algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI: Morgan Kaufmann, 794-799.

Aha, D. W., Kibler, D. and Albert, M. K. 1991. Instance-based learning algorithms. *Machine Learning*, **6**, 37-66.

Brodley, C. E. and Friedl, M. A. 1996. Identifying and eliminating mislabeled training instances. In *Proceedings of Thirteenth National Conference on Artificial Intelligence*, 799-805.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. 198). *Classification and Regression Trees*, Wadsworth International Group.

Brodley, C. E. and Friedl, M. A. 1999. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, **11**, pp. 131-167.

Cover, T. M. and Hart, P. E. 1967. Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, **13**, 21-27.

Dasarathy, B. V. 1980. Nosing around the neighborhood: A new system structure and classification rule for recognition in partial exposed environments. *Pattern Analysis and Machine Intelligence*, **2**, 67-71.

Gamberger, D., Lavrac, N. and Saso Dzeroski, S. 1996. Noise elimination in inductive concept learning. In *Proceedings of 7th International Workshop on Algorithmic Learning Theory*, 199-212.

Gates, G. W. 1972. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 431-433.

Hart, P. E. 1968. The condensed nearest neighbor rule. *Institute of Electrical and Electronics Engineers and Transactions on Information Theory*, **14**, 515-516.

Hopfield, J. J. and Tank, D. W. 1985. Neural Computations of Decisions in Optimization Problems. *Biological Cybernetics*, **52**, 141-152.

John, G. H. 1995. Robust decision tree: Removing outliers from data. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Montreal, Quebec, 174-179.

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1137-1143.

Merz, C. J. and Murphy, P. M. 1996. UCI repository of machine learning databases. *http://www.ics.uci.edu/∼mlearn/MLRepository.html*

Provost, F. and Domingos, P. 2003. Tree Induction for Probability-Based Ranking *Machine Learning*, **52**, 199-216.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*, Morgan Kaufman, Los Altos, CA.

Teng, C. M. 1999. Correcting noisy data. In *Proceedings of 16th International Conference on Machine Learning*, 239-248.

Teng, C. M. 2000. Evaluating noise correction. In *Proceedings of 6th Pacific Rim International Conference on Artificial Intelligence*, Lecture Notes in AI, Springer-Verlag.

Wilson, D. 1972. Asymptotic properties of nearest-neighbor rule. *IEEE Transactions on Systems, Man and Cybernetics*, **6(6)**, 448-452.

Wilson, D. R. and Martinez, T. R. 1997. Instance Pruning Techniques. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers, San Francisco, CA, 403-411.

Wilson, D. R. and Martinez, T. R. 2000. Reduction Techniques for Exemplar-Based Learning Algorithms. *Machine Learning*, **38(3)**, 257-286.

Winston, P. H. 1975. Learning structural descriptions from examples. In Winston, P. H. (Ed.), *The Psychology of Computer Vision*, McGraw-Hill, New York.

Zadrozny, B. and Elkan, C. 2001. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML'01)*, 609-616.

Zeng, X. and Martinez, T. R. 2001. An Algorithm for Correcting Mislabeled Data. *Intelligent Data Analysis*, **5**, 491-502.

Zeng, X. and Martinez, T. R. 2003. A Noise Filtering Method Using Neural Networks. In *Proceedings of IEEE International Workshop on Soft Computing Techniques in Instrumentation, Measurement and Related Applications*, 26-31.