Kernel Based Detection of Mislabeled Training Examples

Hamed Valizadegan* Pang-Ning Tan †

Abstract

The problem of identifying mislabeled training examples has been examined in several studies, with a variety of approaches developed for editing the training data to obtain better classifiers. Many of these approaches involve applying an individual or an ensemble of classifiers to the training set and filtering the mislabeled examples based on their consistency with respect to the classifier's outputs. In this study, we formulate mislabeled detection as an optimization problem and introduce a kernel-based approach for filtering the mislabeled examples. Experimental results using a variety of data sets from the UCI data repository demonstrate the effectiveness of our proposed method, compared to existing nearest-neighbor and ensemble-based filtering schemes.

1 Introduction

The presence of noise, in the form of mislabeled training examples, often has an adverse effect on the performance of classifiers. As the amount of mislabeled examples increases, the classifiers become more susceptible to the model overfitting problem. This has led to considerable interest in developing techniques for identifying and eliminating mislabeled examples from training data.

According to Brodley and Friedl [1], mislabeling occurs due to a variety of reasons, including the subjective nature of the labeling task, the mistakes made during data entry, and the lack of information to determine the true label of a given example. Subjective mislabeling may happen, for instance, when experts are given the task of rating a particular observation according to their personal judgement. The assessments provided by some experts may disagree with the general consensus, which lead to mislabeling errors. Another potential cause for mislabeling is due to data entry mistakes which occur when transforming information on paper to computerized forms due to illegible or unclear handwriting. Finally, mislabeling errors may also arise when there is insufficient amount of information to determine the true class label of a given example. For example, in the medical domain, a physician may not be able to make the right diagnosis unless certain expensive medical procedures have been performed on a patient.

The problem of identifying and eliminating mislabeled training examples has been widely studied particularly in the pattern recognition and machine learning literature [1, 2, 3, 4, 5]. While some of these algorithms attempt to edit the mislabeled examples during model building, others apply noise filtering as a preprocessing step prior to executing their model building algorithms. The latter group of algorithms, which is the focus of this study, differs in terms of the strategy used to determine whether a training example is mislabeled.

Some algorithms employ local learning methods such as k-nearest neighbor (kNN) classification to measure the amount of inconsistencies between the label of a training example and the labels of its surrounding neighbors. If there is strong evidence of discrepancy among the labels, the training example is tagged as mislabeled. One problem with this approach is that it does not propagate the mislabeling information to the detection of other training examples. Each training example is examined independently, without considering the decisions made for other examples. Another group of algorithms applies an ensemble of classifiers to the training examples and detects whether the class label assigned to each example is consistent with the output of these classifiers. The main problem with this approach is that the classifiers used to detect mislabeled examples are constructed from a training set containing mislabeled examples. As a result, the induced model may be biased towards the mislabeled examples. The problem becomes even more pronounced when the level of noise in the training data is high.

The use of ensemble learning approaches for mislabeled detection has become increasingly popular in recent years [1, 5, 3]. The performance of such approaches strongly rely on the following two fundamental assumptions—(1) that the errors committed by the base models are independent of each other and (2) that the error rates of the base models should be less than 50%. Guaranteeing that both of these assumptions hold is not a trivial task. This is because there are two types of errors often associated with mislabeled detection. A Type 1 error corresponds to declaring a correctly labeled example as mislabeled, while a Type 2 error corresponds to declaring a mislabeled example as correctly labeled. As will be shown in our experiments, the Type 2 errors in the base models may exceed 50% for many data sets, especially when the noise level is sufficiently high.

*Michigan State University. Email: valizade@cse.msu.edu
†Michigan State University. Email: ptan@cse.msu.edu

Kernel methods are currently at the heart of many machine learning algorithms. These methods are based on the idea of projecting the data into a higher dimensional embedding, known as the Hilbert space, and searching for linear relations in the transformed space. The projection is performed implicitly and specified by a kernel function that facilitates the efficient computation of inner product between any two given examples. These methods have shown great promise in solving a variety of clustering, classification, and dimensionality reduction problems.

In this paper, we present a novel kernel-based approach for detecting mislabeled training examples. Unlike other approaches, we directly formulate mislabeled detection as a constrained optimization problem and introduce a kernel-based strategy to find mislabeled examples. Experimental results indicate that our proposed method can effectively produce higher detection rate and lower false alarm rate compared to existing ensemble and nearest neighbor methods, when applied to many real world data sets.

The remainder of the paper is organized as follows. Section 2 presents some preliminary background with related work. Section 3 describes our proposed kernel-based mislabeled detection approach. Experimental results are reported in Section 4. Finally, we conclude with a summary of our contributions and directions for future research.

2 Background

Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ be a collection of n training examples, where each $\mathbf{x}_i$ is an instance of the input space $\Re^d$ and $y_i \in \{-1, +1\}$ denotes its corresponding class label. The objective of mislabeled detection is to identify a subset of training examples in $\mathcal{D}$ whose labels are incorrect. In the remainder of this section, we briefly review the related work in this area.

2.1 Related Work

Recent years have witnessed growing interest in developing effective methods for detecting mislabeled examples in the training data. Similar to our proposed approach, most of the methods are designed to deal with random noise, instead of systematic mislabeling errors. The existing methods differ in terms of the strategies used to determine the true classification of the training examples.

Local learning methods [2, 6] assume that the class labels of mislabeled examples tend to disagree with the class labels of other examples in their surrounding neighborhood. Sanchez et al. [2] used several variations of the kNN approach, including depuration and nearest centroid neighborhood (NCN), to detect mislabeled examples. Depuration, which was initially introduced by Marques et al. in [6], is an iterative process to modify examples whose class labels disagree with the class labels for most of their nearest-neighbors and to remove other suspicious examples whose class labels cannot be verified with high certainty. In NCN [7], the nearest neighbors are chosen not only based on their proximity, but also whether they are symmetrically placed around the given example. One problem with local learning methods is that they do not propagate the mislabeling information to other examples in the data set. In other words, each training example is examined separately, without considering its effect on other examples.

Ensemble-based methods [8, 5, 3] assume that the mislabeled examples often produce conflicting class labels when multiple, independent classifiers are applied. Algorithms that belong to this category vary in terms of how the different classifiers are constructed. Brodley and Friedl [1] used leave-one-out cross validation to generate an ensemble of classifiers. They also proposed two approaches to detect mislabeled examples—consensus filter and majority vote. A consensus filter tags an example as being mislabeled only if it is misclassified by all the classifiers in the ensemble. A less conservative method is to consider an example to be mislabeled if it disagrees with the majority vote of the classifiers. Verbaeten and Assche [9] considered an ensemble method for mislabeled detection based on boosting and bagging. John proposed an iterative method to remove examples by pruning a decision tree algorithm and then to rebuild a new tree from the reduced data set [4]. Zhu et al. [5] introduced a method based on partitioning a large data set into smaller subsets and building a corresponding classifier for each subset. Jiang et al. [3] considered an ensemble of neural networks for mislabeled detection while Venkataraman et al. [10] employed an ensemble approach using support vector machine trained on different feature subsets of the data. One problem with these ensemble-based methods is that the underlying models are built from a training set that contains mislabeled examples. Another problem is the requirement that each base classifier must be independent and has type 1 and type 2 error rates of less than 50%, which may not hold when the level of noise is sufficiently high.

In the past decade, kernel-based learning has found wide applications in a variety of data mining and machine learning problems, including classification, clustering, and dimensionality reduction [11, 12, 13, 14, 15]. Through the use of a positive semi-definite kernel function, kernel-based learning exploits the inherent structure of the data and thus produces highly effective results. Nevertheless, to date, we are not aware of any work that uses kernel-based learning for mislabeled detection.

One challenging aspect of kernel-based learning is the difficulty of choosing the appropriate kernel function as well as its corresponding kernel parameter [16, 17, 14, 18]. Although many algorithms require the user to set the parameters manually, there have been several recent studies devoted to the automatic selection of kernel parameters. Shi and Malik [17] recommended using a kernel width of 10% to 20% of the maximum distance between examples. Zelnik-Manor and Perona [16] proposed a self-tuning approach to compute the local kernel width of a data point $\mathbf{x}_i$ based on its distance to the kth nearest neighbor. Another promising strategy is to learn the kernel matrix directly from data. Examples of such methods include kernel alignment [19, 20], semi-definitive programming [21], spectral graph partitioning [14] and unsupervised learning of kernel matrix[15].

3 Methodology

We begin this section with an introduction to the weighted k-nearest neighbor classifier, which is the building block of our proposed kernel-based mislabeled detection algorithm. We also show how WkNN can be extended for handling the mislabeled detection problem. Finally, in Section 3.2 we discusses the limitations of using WkNN for mislabeled detection, leading to the development of our proposed kernel-based learning algorithm called *KBDMS*.

3.1 Weighted-kNN approach (WkNN)

A k nearest neighbor (kNN) classifier predicts the class label of an example based on the majority vote of class labels among its k nearest neighbors. One potential drawback with this approach is that finding the appropriate value for k is not a trivial task. Many algorithms employ the leave-one-out cross-validation approach to determine the right value for k, but such an approach is not only time-consuming, they do not guarantee that the optimal k is chosen for the particular data set.

Instead of using simple majority voting, the weighted kNN approach takes into account the similarity between examples. More specifically, the class label for the training example x_i is determined as follows:

$$\hat{y}_i = \frac{\sum_{j \in N(x_i)} y_j w_{ij}}{\sum_{j \in N(x_i)} w_{ij}} \tag{3.1}$$

where $W = [w_{ij}]$ corresponds to the similarity matrix between all the training examples and $N(x_i)$ is the neighborhood formed by the k closest training examples of x_i. The predicted class $\hat{y}_i$ ranges between -1 and +1. Note that, depending on the choice of similarity function, it may also be possible to extend the summation over all the training examples, instead of restricting it only to the k nearest neighbors.

To measure the extent of which the predicted class of x_i departs from its actual class y_i, the following margin-like quantity [22] can be defined:

$$\begin{aligned} \delta_i &= y_i \frac{\sum_{j \in N(x_i)} y_j w_{ij}}{\sum_{j \in N(x_i)} w_{ij}} \\ &= y_i \hat{y}_i \end{aligned} \tag{3.2}$$

This quantity also ranges between [-1,1] and can be interpreted as follows:

1. When δ_i is close to -1, the class labels of the nearest neighbors strongly disagree with y_i.
2. When δ_i is close to +1, the class labels of the nearest neighbors strongly agree with y_i.
3. When δ_i is close to zero, the nearest neighbors have equal representations from both classes.

Therefore, δ_i can be used as a measure for detecting mislabeled examples. The smaller the value of δ_i, the more likely x_i is mislabeled.

3.2 Kernel-based Detection of Mislabeled Samples (KBDMS)

The weighted kNN approach described in the previous section has several inherent limitations. First, according to Equation (3.1), it predicts the class of a given example based on the assumption that other examples in its local neighborhood are correctly labeled. Second, as can be seen from Equation (3.3), the decision whether a given example is mislabeled has no impact on the analysis of other examples[1]. In other words, mislabeled detection is performed locally without propagating the decision to other examples. This makes the detection process simple but not quite as effective. This is because the incorrect label of a previously known mislabeled example is still being used to detect mislabeling of other examples in its neighborhood. A more effective strategy is to propagate information about the correctness of a class label to all the neighboring examples, but this is not a trivial task.

One way to propagate the decision is to replace each y_j on the right-hand side of Equations (3.1) and (3.3) with its corresponding predicted class, $\hat{y}_j$. However, since this affects the prediction of other examples, the process must be repeated for all the examples. Instead of iteratively computing $\hat{y}_i$ in each round, we formulate mislabeled detection directly as a quadratic optimization problem.

Let p_i be the probability that a training example x_i is mislabeled. Therefore, the expected value for y_i is:

$$E[y_i] = -p_i y_i + (1 - p_i) y_i = (1 - 2p_i) y_i, \tag{3.3}$$

which $-y_i$ is the actual class if x_i is mislabeled.

We may derive a similar quantity as Equation (3.3) which includes the probability that the training examples are mislabeled. However, we need to consider the following four situations:

1. Both x_i and its neighbor x_j are correctly labeled.

[1]Otherwise, the right hand side of Equation (3.3) should involve the δ_js or $\hat{y}_j$s of its neighbors.

2. x_i is wrongly labeled but x_j is correctly labeled.

3. x_i is correctly labeled but not x_j.

4. Neither x_i nor x_j are correctly labeled.

This can be accomplished by replacing each y_i with its corresponding expected value. The quantity δ_i becomes:

$$\delta_i = (1-2p_i)y_i \frac{\sum_{j \in N(x_i)}(1-2p_j)y_j w_{ij}}{\sum_{j \in N(x_i)} w_{ij}} \tag{3.4}$$

The above formulation can be generalized by extending the neighborhood $N(x_i)$ to include all the training examples.

Note that, by replacing each y_i with its expected value, the quantity δ_i becomes a measure of consistency between the expected value of y_i and the expected value of the predicted class, $E(\hat{y}_i)$. Our objective now is to learn the probability vector, $P = (p_1, p_2, ..., p_n)'$ such that δ_i is maximized for all n training examples. In other words, we would like to solve the following optimization problem:

$$\begin{aligned} & \max_{P \in [\mathbf{0},\mathbf{1}]^n} \sum_i \delta_i \\ = \; & \max_{P \in [\mathbf{0},\mathbf{1}]^n} \sum_i (1-2p_i)y_i \frac{\sum_j (1-2p_j)y_j w_{ij}}{\sum_j w_{ij}} \\ = \; & \max_{P \in [\mathbf{0},\mathbf{1}]^n} \sum_i \sum_j \frac{(1-2p_i)y_i w_{ij} y_j (1-2p_j)}{\sum_j w_{ij}} \end{aligned} \tag{3.5}$$

To compute the solution for the preceding optimization problem, it is useful to rewrite the objective function into the following matrix notation:

$$\max_{P \in [\mathbf{0},\mathbf{1}]^n} (1-2P)' diag(Y) W D^{-1} diag(Y)(1-2P)$$

where W corresponds to an $n \times n$ similarity (kernel) matrix, Y is an $n \times n$ diagonal matrix whose diagonal element $Y_{ii} = y_i$, and D is an $n \times n$ diagonal matrix with diagonal elements

$$D_{ii} = \sum_{j=1}^{n} w_{ij}, \; i = 1, 2, \ldots, n.$$

To simplify the expression, let

$$A = diag(Y) W D^{-1} diag(Y).$$

Therefore the objective function to be maximized can be written as follows:

$$\max_{P \in [\mathbf{0},\mathbf{1}]^n} (1-2P)' A (1-2P) \tag{3.6}$$

It is important to realize that the matrix A in (3.6) is not symmetric. However, following the approach in [22], we may approximate it with a symmetric matrix according to the following transformation: $(A + A')/2$. The resulting symmetric matrix is positive semi-definite, which leads to a convex objective function. Nevertheless, because this is a maximization problem, the solution to (3.6) lies along the boundary, i.e., $\forall i : p_i \in \{0, 1\}$. Such a formulation yields a hard classification of the training examples, i.e., each x_i is classified as either mislabeled or correctly labeled.

It is also worth noting that the objective function given in (3.6) is maximized by declaring all the examples from one of the two classes as mislabeled. This is because, if all the training examples are from the same class, there is perfect consistency between the class label of an example and the class labels of its surrounding neighborhood. Clearly this is not our desired solution. To overcome this problem, a regularization penalty term is introduced to avoid declaring too many examples as being mislabeled. Our objective function is now modified as follows:

$$\max_{P \in [\mathbf{0},\mathbf{1}]^n} (1-2P)' A (1-2P) - C||P||_1, \tag{3.7}$$

where C is a parameter that controls the tradeoff between maximizing the consistency of the class labels with their corresponding neighborhood examples (i.e., the first term) and minimizing the number of mislabeled examples (i.e., the second term).

Rearranging the terms in (3.7), we obtain:

$$\max_{P \in [\mathbf{0},\mathbf{1}]^n} 4P'AP - (2eA + 2eA' + Ce)P \tag{3.8}$$

where e is a row vector with all its elements being one. Notice that the objective function in (3.8) is in quadratic form, which can be solved efficiently using the Newton method described in [23]. Specifically, because the constraints in this formulation involve only the lower bound and upper bound for P (it does not include other equality or inequality constraints), it can be solved very efficiently using the algorithm presented in [23] for large-scale problems.

In addition, because the formulation stated in (3.8) depends only on the similarity between training examples, it is essentially a kernel-based method, allowing the problem to be solved in a high dimensional space known as the Hilbert space.

3.3 Model Parameters

Our kernel-based mislabeled detection method requires the user to specify two parameters, the constant C and the kernel parameter used to compute the similarity matrix W. For example, if we use an RBF kernel function, the latter parameter corresponds to the kernel width, σ. Note that both of these parameters (C and kernel parameter) are also present in other kernel-based learning formulations, including support vector machines (SVM). In this section, we describe the roles of these parameters in our formulation and present some of the possible methodologies for estimating them.

As previously noted, the parameter C controls the number of mislabeled examples in KBDMS. Setting $C = 0$ would result in declaring all the training examples from one of the two classes as mislabeled. This is not the desired solution because it assumes all the training examples should belong to the same class. On the other hand, choosing a very large value for C yields a solution where all the training examples are assumed to be correctly labeled, i.e., $\forall i : p_i = 0$.

In support vector machines, the parameter C is typically chosen using model selection methods such as ten-fold cross-validation on training data. Unfortunately, such a strategy might not be directly applicable to mislabeled detection unless some of the mislabeled examples are known. However, it may be easier to identify a sample of correctly labeled examples and then artificially inject noise into the sample. We can then apply KBDMS to the data using different parameter values of C. The value for C that gives the highest detection rate of the artificially injected mislabeled examples could be selected as our parameter. For our experiments, we apply the same parameter value, $C = 1$, on all the data sets. As will be shown in Section 4, we still obtain very accurate results even without tuning the parameter C.

As mentioned in Section 2.1, there are many ways to choose the width parameter σ of an RBF kernel function. We apply the following two strategies in this paper:

Fixed Width In this approach the kernel width is fixed at 10% of the variance of the distance between examples.

Variable Width Unlike the previous strategy, this approach employs a local kernel width associated with each example x_i. As a result, the kernel width depends on the pair of examples whose similarity value is to be computed. To understand the rationale behind this approach, let σ_i be the scaling parameter corresponding to example x_i. The scaled distance from x_i to x_j as measured by x_i is $d(x_i, x_j)/\sigma_i$, while the same scaled distance as measured by x_j is $d(x_i, x_j)/\sigma_j$. Therefore, the squared distance d^2 between x_i and x_j can be written as

$$d(x_i, x_j)/\sigma_i \times d(x_i, x_j)/\sigma_j = d^2(x_i, x_j)/(\sigma_i \sigma_j).$$

Zelnik-Manor and Perona [16] proposed a method for selecting the local scale σ_i based on the local statistics in the neighborhood of x_i. In our experiments, we use $\sigma_i = d(x_i, x_K)$ which is the distance between x_i and its k-th nearest neighbor (where $k = 7$ for all our experiments). With this approach, the similarity between x_i and x_j is given by:

$$w_{ij} = \exp\left[- d^2(x_i, x_j)/\sigma_i \sigma_j \right].$$

As a result, there are two versions of the kernel-based learning algorithm investigated in this study: *KBDMS1*, which uses the fixed kernel width approach described above, and *KBDMS2*, which uses the variable width approach.

4 Experimental Evaluation

This section presents our experimental setup and the results obtained when applying KBDMS to a variety of data sets. The objectives of our experiments are: (1) to compare the performance of KBDMS against other existing algorithms (Section 4.2), (2) to illustrate the effect of applying different noise levels (Section 4.3), (3) to illustrate the improvements in classification accuracy when the mislabeled examples are removed (Section 4.4), and (4) to illustrate the effect of Type-1 and Type-2 errors in ensemble methods on mislabeled detection (Section 4.5).

4.1 Experimental Setup

To evaluate the performance of our method, we used a variety of data sets from the UCI data repository [24]. Table 1 shows a summary of the data sets. We artificially inject noise to the data set by modifying the class labels for some randomly chosen training examples. Similar to other previous works, we do not consider mislabeling due to systematic errors. Such type of mislabeling error is beyond the scope of this paper. Each of our experiments is repeated ten times and the average performance is reported.

4.1.1 Algorithms The following mislabeled detection algorithms are chosen for our experiments:

WkNN: This corresponds to the weighted kNN algorithm described in Section 3.1. We used the RBF kernel function as the underlying similarity measure and fixed the kernel width to 10% of the variance of the samples distance. After applying the algorithm, wkNN will tag a training example x_i as mislabeled if its corresponding margin δ_i (see Equation 3.3) is negative.

Table 1: Description of data set.

	# of instances	# of attributes
Ionosphere	351	34
Vote	435	16
Flare	1066	10
Digits08	352	64
Digits38	357	64
Digits56	363	64
Digits17	361	64
Digits27	356	64
Covertype	2526	10

KBDMS1: This corresponds to the KBDMS with fixed kernel width method[2] as described in Section 3.2. Note that $C = 1$ for all our experiments.

KBDMS2: This corresponds to the KBDMS with variable width method as described in Section 3.2. Note that $C = 1$ for all our experiments.

CCTree: CCTree stands for Consensus Cross-Validation Tree Filter [1]. It partitions the training examples into 10 folds and uses 9 of the folds to build a decision tree. The tree building step is then repeated 9 more times, each time leaving out a different partition. At the end of the process, an ensemble of 10 tree classifiers is obtained. The ensemble is then used to classify each of the training examples. A given example is declared as mislabeled if its class label disagrees with the predictions made by all the classifiers in the ensemble.

MCTree: MTree stands for Majority Cross-Validation Tree Filter [1]. It generates an ensemble of classifiers using the same approach as CCTree, except it uses majority vote to decide whether a given example is mislabeled. More specifically, if the class label of the example disagrees with more than half of the predictions made by the classifiers, the example is declared as mislabeled.

CBTree: CBTree stands for Consensus Bagging Tree Filter [9]. This approach is very similar to CCTree except it uses bootstrapping to create an ensemble of classifiers.

MBTree: MBTree stands for Majority Bagging Tree Filter [9]. This approach is very similar to MCTree, except it uses bootstrapping to create an ensemble of classifiers.

4.1.2 Evaluation Metrics

Let $\mathcal{D}$ be a collection of training examples. Suppose $M \subseteq \mathcal{D}$ corresponds to the set of training examples that was wrongly labeled. Furthermore, let $M' \subseteq \mathcal{D}$ be the set of examples tagged by the algorithm as being mislabeled.

We evaluated the performance of a mislabeled detection algorithm in terms of its True Positive Rate (TPR) and False Positive Rate (FPR), which are defined as follows:

$$\mathbf{TPR} = \frac{|M' \cap M|}{|\mathcal{M}|}$$

$$\mathbf{FPR} = \frac{|M' - (M' \cap M)|}{|D - M|}$$

Another possibility is to summarize TPR and FPR into a single metric using the well-known F_1-measure, which is a widely used metric in information retrieval.

[2] For the Digits38 data, the default width parameter in KBDMS1 and WkNN produces a kernel matrix with most of its elements being zero! So we used 20 percent of variance of the distance between samples as the width parameter.

4.2 Comparison among Mislabeled Detection Algorithms

Tables 2 and 3 show the performance comparison (in terms of TPR and FPR) between KBDMS1 and KBDMS2 against other baseline methods when 10% and 30% of the training examples are mislabeled. Note that KBDMS2 outperforms other competing methods for all the data sets. The performance improvements of KBDMS2 (in terms of both TPR and FPR) are observed in every data set except for the Vote data, in which KBDMS2 has a high TPR but also a relatively high FPR. Comparing the results of both tables, the improvements observed in KBDMS2 becomes even more significant when the number of mislabeled samples increases.

In contrast, the performance of KBDMS1 is not as good as KBDMS2, which shows that using a local kernel width is more effective than using a global kernel width. This analysis shows that, similar to other kernel-based learning methods, the performance of the method is quite sensitive to the choice of kernel function and its corresponding parameters.

For most of the data sets, the TPR values for the ensemble method are very poor. There are two interesting points worth noting regarding this method. First, the lower values of TPR in both CCTree and CBTree suggest that consensus filters have worse detection rates than majority vote filters due to their conservative decision making. The second point is that, in many cases, the TPR of MBTree and MCTree is less than 50%, especially when the level of noise is high. This could explain the poor performance observed in the ensemble method as its Type II error (which is equivalent to 1 - TPR) is more than 50%. We will revisit this issue in Section 4.5.

4.3 Robustness to Noise Level

Our next experiment is to investigate the effect of noise level on the performance of various algorithms. To make the plots easier to visualize, we only show the results for KBDMS2, MBTree, and MCTree (knowing that KBDMS2 is better than KBDMS1 and wKNN, and majority vote filters are better than consensus filters). We use the F_1-measure to compare the performance of the algorithms.

Due to space considerations, we only show the results for four data sets. Note that the results on other data sets also look very similar. For each data set, we vary the amount of mislabeled examples from 5% up to 50%. Each experiment is repeated 10 times and we report the average values of their corresponding F_1-measure. From Figure 1, it can be clearly seen that KBDMS2 is generally superior than the other two ensemble methods, except for the Vote data with noise levels below 20%. KBDMS2 also attains 100% TPR on the digits data sets when the level of noise is less than 35%. However, similar to the other approaches, its performance starts to

Table 2: Performance comparison among various algorithms (10% of examples are randomly mislabeled) .

	KBDMS1	KBDMS2	WkNN	CBTree	MBTree	CCTree	MCTree
	TPR, FPR						
Ionosphere	.74, .08	**.88, .03**	.77, .19	.006, 0	.45, .02	.02, .001	.24, .01
Vote	**.78, .05**	**.83, .08**	**.87, .10**	.15, 0	**.77, .03**	.38, .004	.76, .03
Flare	.70, .13	**.83, .17**	.80, .20	.56, .05	.72, .17	.59, .08	.70, .15
Digits08	.78, .02	**.99, 0**	.90, .10	0, 0	.44, .006	.02, 0	.37, .01
Digits38	.83, 0.02	**.88, 0.005**	.91, .10	.003, 0	.43, .005	.02, 0	.32, .007
Digits56	.80, .03	**.99, 0**	.90, .09	0, 0	.54, .007	.008, .0003	.35, .01
Digits17	.76, .03	**1.0, 0**	.89, .10	.003, 0	.53, .008	.04, 0	.24, .01
Digits27	.77, .02	**.97, 0.002**	.91, .10	.003, 0	.50, .01	.014, 0	.35, .007
Covertype	.986, .005	**.9998, .0001**	.998, .002	.02, 0	.35, .001	.007, 0	.26, .001

Table 3: Performance comparison among various algorithms (30% of examples are randomly mislabeled) .

	KBDMS1	KBDMS2	WkNN	CBTree	MBTree	CCTree	MCTree
	TPR, FPR						
Ionosphere	.62, .11	**.83, .05**	.68, .35	.002, 0	.18, .03	.007, 0	.09, .007
Vote	.52, .10	**.80, .10**	.71, .25	.004, .001	.45, .06	.12, .004	.42, .06
Flare	.76, .45	**.83, .17**	.67, .32	.17, .02	.63, .20	.46, .06	.65, .16
Digits08	.56, .08	**.97, 0**	.67, .30	0, 0	.20, .013	0, 0	.05, .007
Digits38	.56, .07	**.84, .005**	.67, .28	0, 0	.17, .02	0, 0	.31, .05
Digits56	.55, .09	**.98, 0**	.70, .29	0, 0	.19, .01	0, 0	.07, .007
Digits17	.57, .08	**1.0, 0.0**	.69, .29	0, 0	.25, .015	0, 0	.08, .006
Digits27	.56, .08	**.98, .004**	.68, .27	0, 0	.18, .02	.006, .0003	.094, .01
Covertype	.863, .025	**.96, .007**	.847, .11	.0003, 0	.25, .02	.0004, .0001	.11, .01

degrade when the noise level increases. It is also interesting to observe that MBTree usually performs better than MCTree suggesting that bootstrapping might be more effective to create ensemble models for mislabeled detection compared to the 10-fold data partitioning approach.

4.4 The Effect of Removing Mislabeled Examples

So far, we have shown that KBDMS2 has high detection rate and low false alarm rate even when the level of noise is high. The question is, how do the improvements in detection rates and false alarm rates translate to making better classifiers?

To investigate this issue, we split the original data into two parts: a training set and a test set. The training set contains 80% of the overall examples while the test set contains the remaining 20%. Mislabeling errors are then randomly introduced to the training set (i.e., the test set remains unchanged).

To illustrate the effect of removing mislabeled examples, we consider the following classifiers:

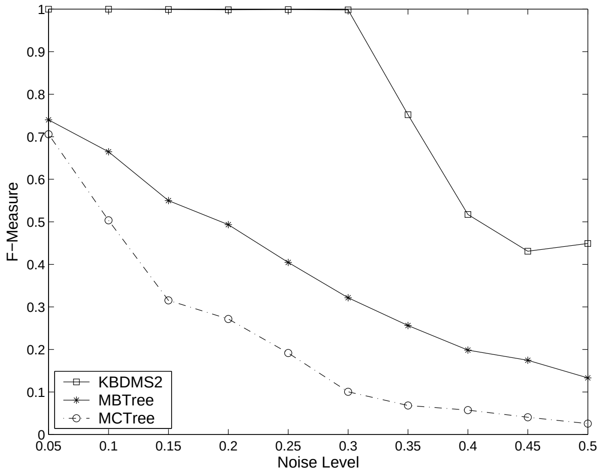
No Mislabeled This corresponds to the baseline approach in which the training set remains unperturbed. We expect such an approach to produce classifiers with the highest accuracy.

KBDMS2 In this approach, we discard any examples that are found to be mislabeled by KBDMS2. We then train a new classifier using only the remaining training examples.

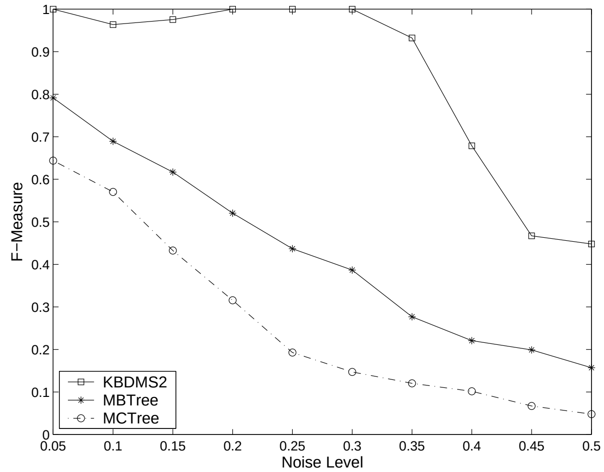
MBTree This is analogous to the previous KBDMS2 approach, except we use MBTree for mislabeled detection. Again, we choose MBTree because it has better performance than MCTree.

No Filter This is another baseline approach, where we train a classifier on the training set without removing the mislabeled examples beforehand.

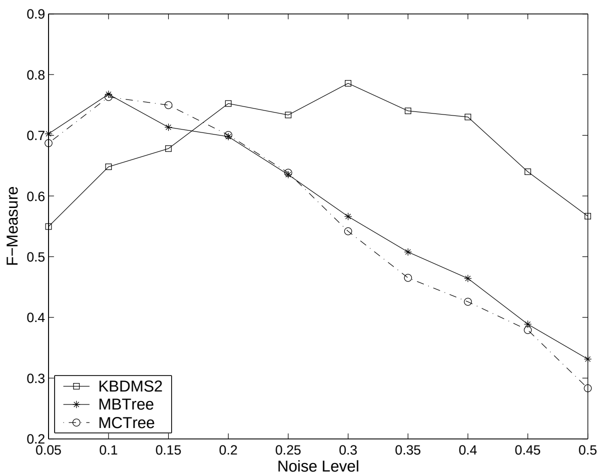
For this experiment, we also vary the amount of mislabeled examples from 5% to 50% to illustrate the effect of noise level. We repeated the experiment 10 times for each noise level and obtained the average accuracy over 10 runs. Figure 2 shows the results of the experiments. It is clear that editing the training set by removing mislabeled examples is effective for both KBDMS2 and MBTree, but KBDMS2 makes better classifiers (even in the case of the Vote data set) due to its better performance at detecting mislabeled examples.



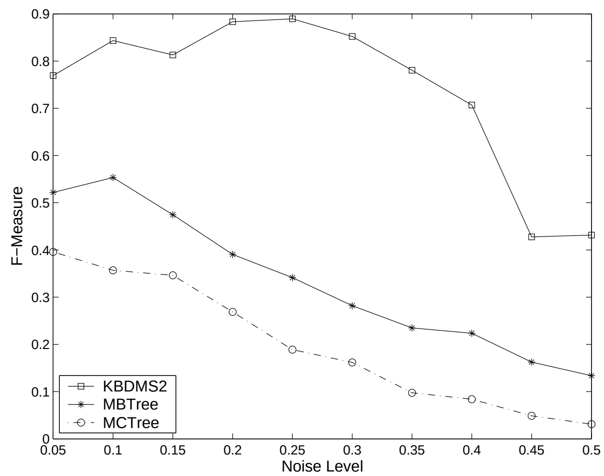
(a) Digitis08



(b) Digitis17



(c) Vote



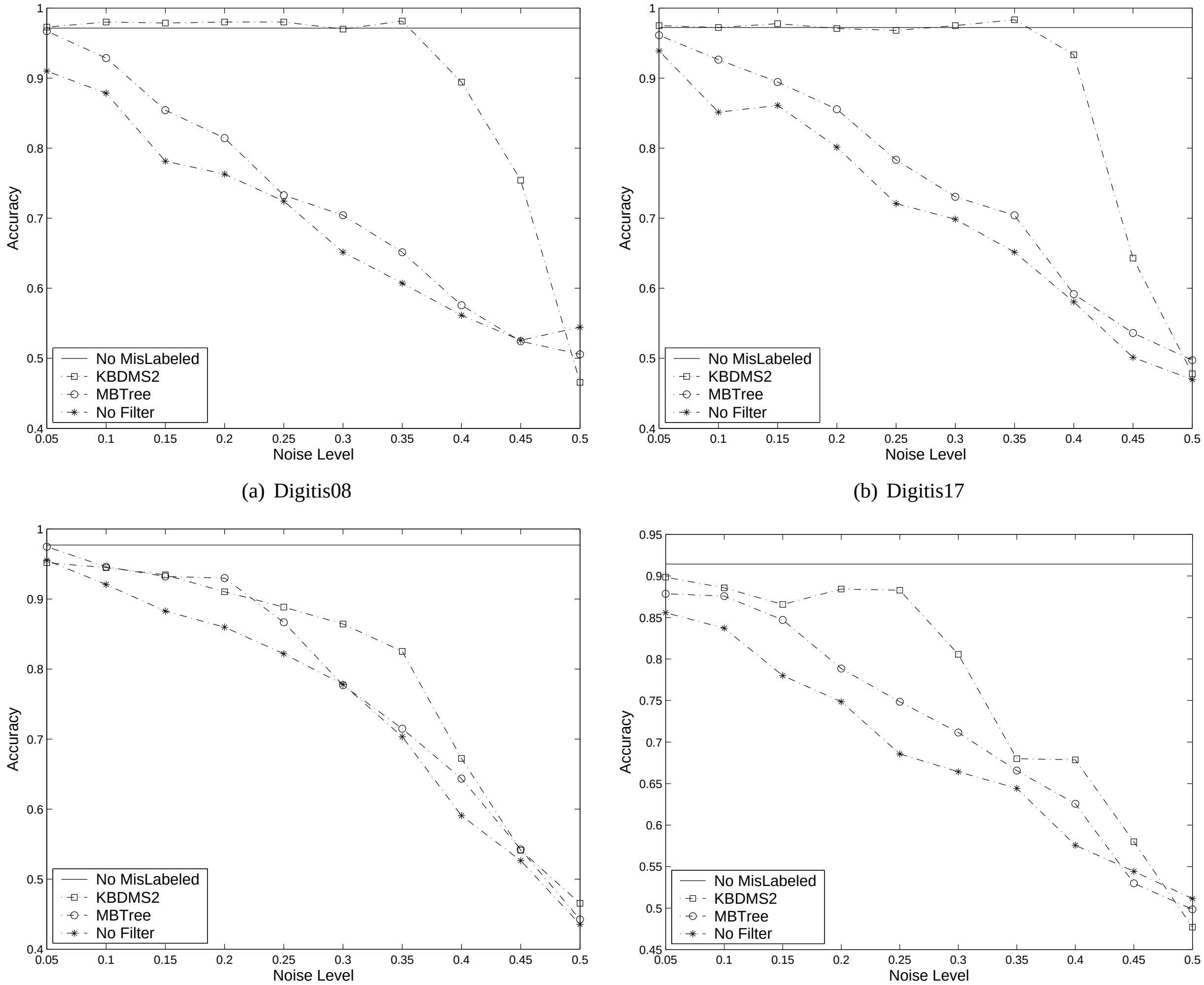
(d) Ionosphere

Figure 1: F-measure comparison of different algorithms for datasets with different noise levels

4.5 Analysis of ensemble filters

In our previous experiments, we were quite surprised to observe the poor performance of the ensemble methods, especially in terms of their TPR. We conjectured that this is because the Type 2 errors of the classifier might be greater than 50%. Assuming a null hypothesis in which an example is correctly labeled, a Type 1 error corresponds to declaring a correctly labeled example as mislabeled (i.e., rejecting the null hypothesis even though it is true). Meanwhile, a Type 2 error corresponds to declaring a mislabeled example as being correctly labeled (i.e., accepting the null hypothesis even though it is false).

To verify our conjecture, we have created 10 different training sets using the bootstrap approach. We then build 10 different classifiers and compute their Type 1 and Type 2 error rates (in terms of detecting mislabeled examples). To capture their variability, we use a boxplot to show the distribution of errors committed on the different training sets. Each box in the boxplot summarizes the following descriptive statistics: median, upper and lower quartiles, minimum and maximum data values. We show the boxplots for both Type 1 and Type 2 errors on four different data sets in Figure 3. For each data set, we again vary the level of noise from 5% to 50%. Notice that the Type II error exceeds 50% most of the time especially when the level of noise is high even though its Type I error is almost always less that 50%. This observation is consistent with the results obtained in the previous sections. While the ensemble-based filters have moderately low false alarm rates, their TPR are very poor. This explains the poor performance of the ensemble method on the given data sets.



(a) Digitis08 (b) Digitis17

(c) Vote (d) Ionosphere

Figure 2: Accuracy comparison of different algorithms for data sets with different noise levels

5 Conclusion

Mislabeling of training examples is a pervasive problem underlying many real-world data. This paper presents a novel kernel-based learning algorithm for detecting mislabeled examples. Unlike other existing methods, which have the drawback of building models from incorrectly labeled data or not propagating their results to other examples, we address these problems by formalizing it as an optimization problem and developing an iterative approach for solving it. We show that our proposed algorithm can effectively detect mislabeled examples and is very robust even when the level of noise is very high. We also demonstrated that the poor performance of the ensemble method is due to their large Type 2 errors, which violates the assumption for most ensemble methods (that the errors should be less than 50%).

For future work, we plan to extend our methodology to multi-class problems. One possibility is to use the one-versus-one or one-versus-all methods often associated with multi-class learning. Another possible research direction is to incorporate the multi-class information directly into the objective function. Finally, we would like to expand the analysis to mislabeling resulting from systematic errors.

6 Acknowledgement

We thank the anonymous reviewers for their valuable comments about the paper.

References

[1] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research 11*, pages 131–167, 1999.

[2] A. I. Marques R. Alejo J. Badenas. J.S. Sanchez, R. Barandela. Analysis of new techniques to obtain quality train-

ing sets. Pattern Recognition Letteres 24, pages 1015–1022, 2003.

[3] Z.-H. Zhou. Y. Jiang. Editing training data for knn classifiers with neural network ensemble. In Lecture Notes in Computer Science 3173, pages 356–361, 2004.

[4] G. H. John. Robust decision trees: Removing outliers from databases. In Proceeding of International Conference on Knowledge Discovery and Data Mining, pages 174–179, 1995.

[5] X. WU X. Zhu and Q. Chen. Eliminating class noise in large datasets. In *Proceeding of International Conference on Machine Learning (ICML2003)*, 2003.

[6] A. I. Marques R. Alejo J. Badenas. J.S. Sanchez, R. Barandela. Decontamination of training data for supevised pattern recognition. In Advances in Pattern Recognition Lecture Notes in Computer Science 1876, pages 621–630, 2000.

[7] B.B. Chaudhuri. A new definition of neighborhood of a point in multi-dimensional space. Pattern Recognition Letters 17, pages 11–17, 1996.

[8] C. E. Brodley and M. A. Friedl. Improving autmated land cover mapping by identifying and eliminating mislabeled observations from training data. In Geoscience and Remote Sensing Symposium, pages 1379–1381, 1996.

[9] S. Verbaeten and A. V. Assche. Ensemble methods for noise elimination in classification problems. In Proceeding of 4th International Workshop on Multiple Classifier Systems, 2003.

[10] D. Metxas D. Fradkin C. Kulikowski. S. Venkataraman. Distinguishing mislabeled data from correctly labeled data in classifier design. In 16th IEEE International Conference on Tools with Artificial Intelligence, 2004.

[11] G. Ratsch K. Tsuda B. Scholkopf. K.R. Muller, S. Mika. An introducion to kernel-based learning algorithms. IEEE Transaction on Neural Networks 12, pages 181–201, 2001.

[12] G. Ratsch K. Tsuda B. Scholkopf. K.R. Muller, S. Mika. Support vector networks. Machine Learning 20, pages 273–297, 1995.

[13] J. Weston B. Schlkopf A. J. Smola S. Mika, G. Rtsch and K.-R. Mller. Invariant feature extraction and classification in kernel spaces. In Advances in Neural Information Processing Systems 12, pages 526–532, 2000.

[14] J. Weston B. Schlkopf A. J. Smola S. Mika, G. Rtsch and K.-R. Mller. Learning spectral clustering. In Advances in Neural Information Processing Systems 16, 2004.

[15] H. Valizadegan and R. Jin. Generalized maximum margin clustering and unsupervised kernel learning. In Advances in Neural Information Processing Systems 19, 2006.

[16] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608, 2005.

[17] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[18] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, 2001.

[19] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. S. Kandola. On kernel-target alignment. In *NIPS*, pages 367–373, 2001.

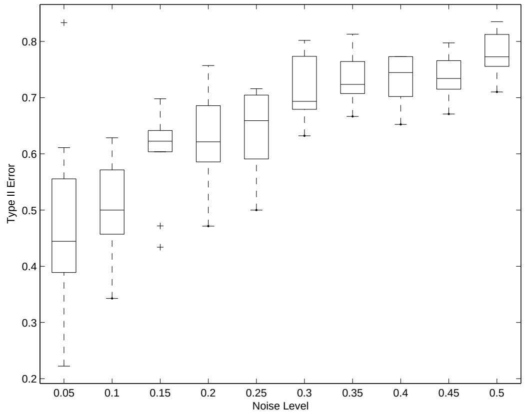
[20] X. Zhu, J. Kandola Z. Ghahramani, and J. Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In *Advances in Neural Information Processing Systems 17*, pages 1641–1648, 2005.

[21] G. R. G. Lanckriet, N. Cristianini, P. L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

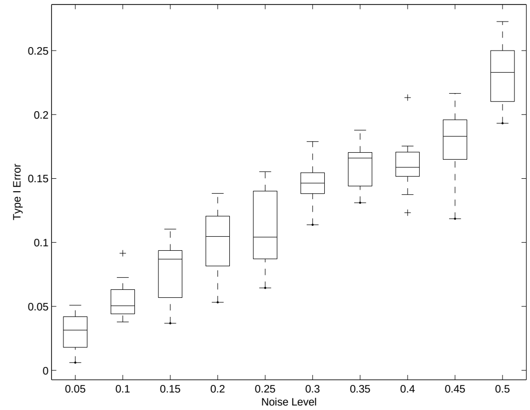
[22] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. S. Kandola. Transductive learning via spectral graph partitioning. In *Proceeding og Twentieth International Conference on Machine Learning*, 2003.

[23] S. Verbaeten and A. V. Assche. A reflective newton method for minimizing a quadratic function subject to bounds on some of the variables. pages 1040–1058, 1996.

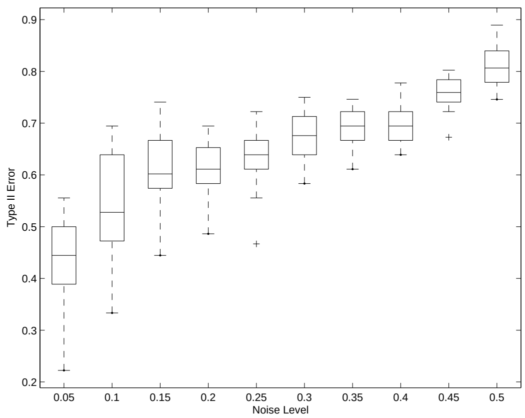
[24] UCIrvine. Uci data repository. In *http://www.ics.uci.edu/ mlearn/MLRepository.html*.



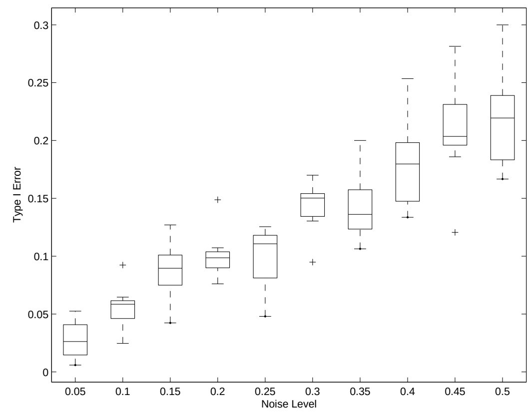
(a) Digitis08 Error II



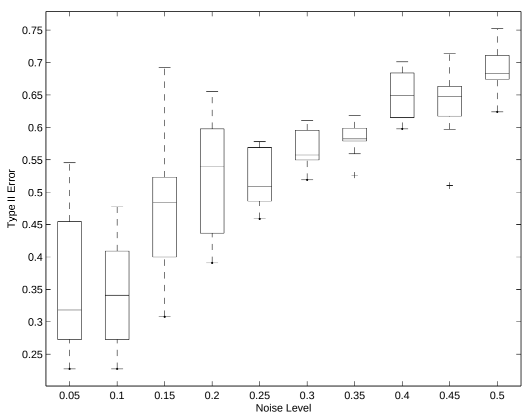
(b) Digitis08 Error I



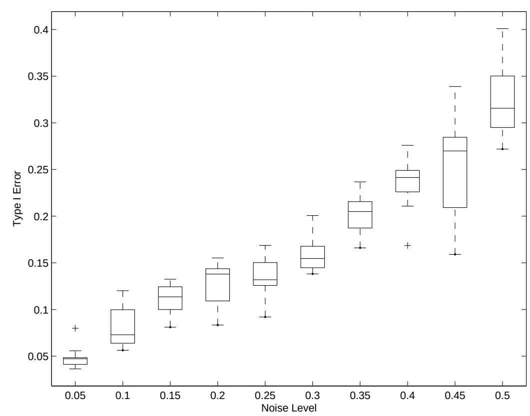
(c) Digitis17 Error II



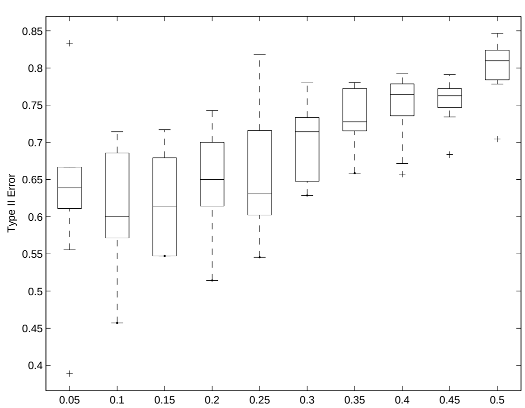
(d) Digitis17 Error I



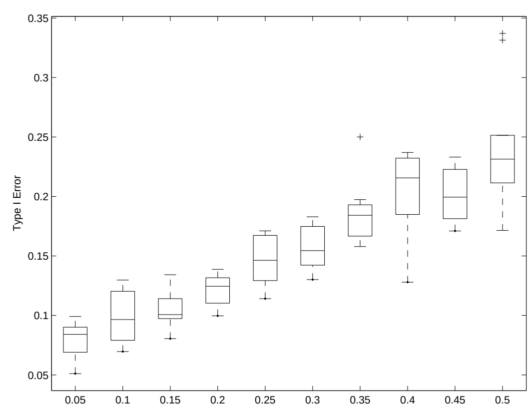
(e) Vote Error II



(f) Vote Error I



(g) Ionosphere Error II



(h) Ionosphere Error I

Figure 3: Type 1 and Type 2 errors of the base classifiers for the ensemble method.