SCIMA 2003 - International Workshop on
Soft Computing Techniques in Instrumentation, Measurement and Related
Applications Provo, Utah, USA, 17 May 2003

A Noise Filtering Method Using Neural Networks

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

Abstract* – During the data collecting and labeling process it is possible for noise to be introduced into a data set. As a result, the quality of the data set degrades and experiments and inferences derived from the data set become less reliable. In this paper we present an algorithm, called ANR (automatic noise reduction), as a filtering mechanism to identify and remove noisy data items whose classes have been mislabeled The underlying mechanism behind ANR is based on a framework of multi-layer artificial neural networks. ANR assigns each data item a soft class label in the form of a class probability vector, which is initialized to the original class label and can be modified during training. When the noise level is reasonably small ($< 30\%$), the non-noisy data is dominant in determining the network architecture and its output, and thus a mechanism for correcting mislabeled data can be provided by aligning class probability vector with the network output. With a learning procedure for class probability vector based on its difference from the network output, the probability of a mislabeled class gradually becomes smaller while that of the correct class becomes larger, which eventually causes a correction of mislabeled data after sufficient training. After training, those data items whose classes have been relabeled are then treated as noisy data and removed from the data set. We evaluate the performance of the ANR based on 12 data sets drawn from the UCI data repository. The results show that ANR is capable of identifying a significant portion of noisy data. An average increase in accuracy of 24.5% can be achieved at a noise level of 25% by using ANR as a training data filter for a nearest neighbor classifier, as compared to the one without using ANR.

I. INTRODUCTION

Although much effort can be made to avoid noise during data collection, it is often difficult and sometimes impossible to completely exclude noise, which could be introduced during one of collecting tasks such as selecting, measuring, and recording. In the field of machine learning, neural networks and pattern recognition, a typical approach to evaluate the performance of a classifier is to test it on some real-word data sets. Clearly, the reliability of the evaluation depends on the quality of data sets, and it will inevitably be degraded by the noise included in the data sets.

This issue was previously addressed in the area of instance-based learning, whose performance is particularly sensitive to noise in training data. To eliminate noise in a training set, Wilson used a 3-NN (Nearest Neighbor) classifier as a filter (or pre processor) to eliminate those instances that were misclassified by the 3-NN, and then applied 1-NN as the classifier on the filtered data [1]. Several versions of edited nearest neighbor algorithms applied certain criteria to select only a faction of original data, which serves as a mechanism to remove potential noise and reduce storage [2, 3, 4]. Aha et. al. proposed an algorithm to remove noise and reduce storage by retaining only those items that had good classification records [5, 6]. Wilson and Martinez presented several instance-pruning techniques that were capable of removing noise and reducing storage requirement [7, 8].

The idea of using selected instances in training data has also been applied to other types of classifiers. Winston proposed an approach to learn structural descriptions by selecting "near miss" instances while constructing the classifier [9]. John [10] presented a method that first removed those instances pruned by a C4.5 tree [11] and then built a new tree based on filtered data. Gamberger proposed a noise detection and elimination method based on compression measures and the Minimum Description Length principle [12]. Brodley and Friedl applied an ensemble of classifiers as a filter to identify and eliminate mislabeled training data [13]. Teng [14, 15] employed a procedure to identify and correct noise based on predictions from C4.5 decision trees. In previous work [16], we proposed a neural network based method to identify and correct mislabeled data.

In this paper we present a noise filtering algorithm, called ANR (automatic noise reduction), to identify and remove mislabeled instances in a data set. ANR is based on the framework and mechanism of multi-layer neural networks trained by backpropagation. It assigns a class probability vector to each pattern in the data set. While the network itself is trained by standard backpropagation, the class probability vector is constantly updated using a learning procedure such that it becomes closer to the output of the network. The output of the network is determined by the architecture and weight settings of the network, which are the results of previous training based on all patterns in the whole data set. For a small mislabeled level (e.g. $< 30\%$), the network will be predominantly determined by those correctly labeled patterns. Thus, through training, the output of the network becomes more consistent with the class probability vector of correctly labeled patterns and less consistent with those of mislabeled patterns. With this mechanism, the class probability vector of mislabeled patterns can be gradually modified and eventually changed to the correct class label after sufficient training. After training, ANR treats those patterns whose class labels have been changed as noise and removes them from the data set. There are three reasons to choose removing them from the data set instead of keeping them with a newly assigned class label. First, a small fraction of correctly labeled patterns could be mistakenly identified as noise, and thus it would add extra noises if keeping them in the data set. Second, even if a mislabeled pattern is identified, it may be assigned to an incorrect class label if there are more than two classes. Third, removing a small portion of a data set has a small impact on the quality of a classifier that is trained on this data set, unless the data set is very small. However, keeping even a small fraction of noise is more detrimental to the quality of a classifier.

The ANR procedure presented in this work has the following distinct features compared to previous approaches for similar tasks. (i). In previous work [*e.g.*, 13, 14, 15], a data set was first divided into a training set S and a test set T. The noise in T was identified through predictions made by classifiers constructed from S. However, because S consists of the same percentage of noise as T, those predictions may not be reliable. In contrast, ANR includes all instances in the process and allows every instance to change its class label, without relying on a pre-constructed classifier. (ii). By using a continuous class probability vector, ANR allows a large number of hypotheses about class labeling to interact and compete with each other simultaneously, and thus enables them to smoothly and incrementally converge to an optimal or near-optimal solution. This type of search strategy has been shown efficient on a large solution-space for NP-class optimization problems using relaxation-type neural networks [17]. (iii). Using multi-layer feed-forward networks as classifiers can take advantage of their high capacity for fitting the target function. (iv). Both nominal and numerical attributes can be easily handled by ANR (in contrast, each attribute needs to be nominal in [14, 15]).

We have tested the performance of ANR on 12 data sets drawn from the UCI data repository. The results show that for most data sets, ANR is capable of identifying more than half of the mislabeled items while keeping a small error rate of misidentifying non-noisy data. Using ANR as a noise filter for training data can increase the accuracy of a nearest neighbor classifier by an average of 24.5% on the 12 data sets.

II. AUTOMATIC NOISE REDUCTION ALGORITHM

Let S be an input data set in which some instances have been mislabeled, Our task is to find a procedure to identify and remove those mislabeled instances and then output a filtered data set $\hat{S}$. Let α be the non-mislabeled fraction and β $(= 1 - \alpha)$ the mislabeled fraction of input data set S. Let $S^{(c)}_{sub}$ be the correctly labeled subset and $S^{(m)}_{sub}$ the mislabeled subset ($S^{(c)}_{sub} + S^{(m)}_{sub} = S$). The instances in $S^{(c)}_{sub}$ have a tendency of strengthening the regularities possessed in S, while those in $S^{(m)}_{sub}$ 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 $S^{(c)}_{sub}$ will be dominant. The strategy of ANR is to apply the regularities discovered by a neural network in $S^{(c)}_{sub}$ to correct those mislabeled instances in $S^{(m)}_{sub}$.

We choose a multi-layer neural network as the underlying classifier to capture the regularities contained in S because neural networks with one hidden-layer have the capability of approximating any function [18] and they have demonstrated capability of detecting and representing regularities in a data set. The format and procedure applied in ANR are the same as those of standard backpropagation networks except for the following.

In the standard procedure, each instance v in S has the following format:

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

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

In ANR, a *class probability vector* is attached to each instance v in S:

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

where $\mathbf{p} = (p_1, p_2, ..., p_c)$ is the class probability vector of v and c is the number of class labels. In addition, there are an input U_i and output V_i assigned for each node. V_i is determined by U_i through the *sigmoid* activation function:

$$V_i = \frac{1}{2}(1 + tanh(\frac{U_i}{u_0})) \tag{3}$$

where $u_0 = 0.02$ is the amplification parameter that reflects the steepness of the activation function.

The weights in the network are updated using the standard backpropagation procedure. For each instance the class probability vector $\mathbf{p}$ is initially set to the original class label. A learning procedure is applied to update $\mathbf{p}$ during training. For each node representing a class i, the input U_i is first updated based on the difference between p_i and the network output, and the output V_i is then updated from U_i according to Eq. (3). The class probability vector $\mathbf{p}$ is updated by assigning V_i to p_i and then normalized to 1. Thus after each update, $\mathbf{p}$ gets closer to the output node value. As long as the noise level is reasonable small, the regularities of a data set can be reflected in the architecture and weight setting of the network after sufficient training. Thus a mislabeled class probability vector can be gradually changed to a correct one through this learning procedure.

The basic steps of ANR are explained in the following (for each parameter we tried different settings and chose the value with the best performance):

- The weights of the network are initially set randomly with uniform distribution in the range $[-0.05, 0.05]$. The initial number of hidden nodes is set to be 1.
- For each instance $v = (\mathbf{x}, y, \mathbf{p})$ (where y is the initial class label), its output vector $\mathbf{V}$ is initially set as follows. V_y (the output probability for class y) is set to be a large fraction D (=0.95), and then (1-D) is divided equally into the other $(C-1)$ output components. The input U_i is then determined from the corresponding output using the inverse *sigmoid* function.
- For each training instance v, the weights in the network and the probability vector $\mathbf{p}$ for v are updated using the following procedure:
  (i). Update the network weights by standard backpropagation (learning rate $L_n = 0.2$, momentum $M_n = 0.5$).
  (ii). For each class i, the input U_i is updated using formula:
  $$U_i = U_i + L_p(O_i - p_i) \tag{4}$$
  where O_i is the value of output node i for instance v, and L_p (= 0.0005) is the learning rate for *probability* vector. V_i is calculated from U_i by Eq. (3). $\mathbf{p}$ is then updated by:
  $$p_i = V_i \tag{5}$$
  and then $\mathbf{p}$ is normalized to 1 (sum of p_i is equal to 1).
  (iii). The class label y for instance v $(= (\mathbf{x}, y, \mathbf{p}))$ is also updated using the following formula:
  $$y = argmax_i\{p_i|(i = 1, 2, ...C)\} \tag{6}$$
- After every N_e (= 20) epochs, the sum of squared errors (SSE) is calculated to monitor the progress of the training. Instead of using SSE directly, we use an *adjusted* version $SSE^{(adj)}$, which is calculated by
  $$SSE^{(adj)} = SSE^{(std)} + SSE^{(hn)} + SSE^{(dist)} \tag{7}$$
  where $SSE^{(std)}$ is the *standard* SSE. $SSE^{(hn)}$ is an additional term taking into account the effects of the number of hidden nodes. More hidden nodes can usually lead to a smaller $SSE^{(std)}$, but with a higher possibility of overfitting. To reduce this effect, we add an additional error term $SSE^{(hn)}$ using an empirical formula:
  $$SSE^{(hn)} = A_1(H-1)N(C-1)/C \qquad (H \leq I)$$
  $$= (A_1(I-1)+A_2(H-I))N(C-1)/C \quad (H > I) \tag{8}$$
  where H is the number of hidden nodes, I is the number of input nodes, N is the number of instances in the data set, and C is the number of classes. A_1 (= 0.05) and A_2 (= 0.2) are two empirical parameters. $SSE^{(hn)}$ increases with H (it increases relatively slow when $H \leq I$ and more rapidly when $H > I$).
  $SSE^{(dist)}$ is another term that takes into account the deviation of the current class *distribution* from the initial (original) one. It is based on the assumption that mislabeling has a random nature and thus the class distribution is expected to be same before and after the correcting procedure. An error term $SSE^{(dist)}$ is added to reflect the difference between the distributions. Let $\mathbf{q}$ be the class distribution vector $\mathbf{q}$ defined by
  $$\mathbf{q} = (q_1, q_2, ..., q_C) = (\frac{N_1}{N}, \frac{N_2}{N}, ...\frac{N_C}{N}) \tag{9}$$
  where N is the total number of instances in the data set, and N_i is the number of instances labeled with class i $(i = 1, 2, ..., C)$. Let $\mathbf{q^{(init)}} = (q_1^{(init)}, q_2^{(init)}, ...q_C^{(init)})$ and $\mathbf{q^{(curr)}} = (q_1^{(curr)}, q_2^{(curr)}, ...q_C^{(curr)})$ be the initial and current class distribution vector respectively. Then $SSE^{(dist)}$ is calculated using the formula
  $$SSE^{(dist)} = \frac{N(C-1)}{C} * \sum_{i=1}^{C} B_i D_i * max(q_i^{(curr)}, q_i^{(init)}) \tag{10}$$
  where $D_i = |q_i^{(curr)} - q_i^{(init)}|/q_i^{(init)}$ is the difference fraction between $q_i^{(curr)}$ and $q_i^{(init)}$. B_i is an empirical parameter determined by: $B_i = 0.1$ when $D_i < 0.05$; $B_i = 1.0$ when $D_i \geq 0.05$.
- For a fixed number of hidden nodes, H $SSE^{(adj)}$ is compared to the stored best (minimum) of the previous $SSE^{(adj)}$ in our experiment), the calculated error $SSE^{(adj)}$ is compared with the stored best (minimum) of the previous $SSE^{(adj)}$ after each N_e (= 20) epochs. If it is smaller, then it will replace the previous one as the new best $SSE^{(adj)}$ and be stored for future comparison and retrieval, along with the current network configuration and class probability vectors. If no better $SSE^{(adj)}$ is found after N_m (= 10) of N_e epochs (equivalent to $N_e * N_m = 20 * 10 = 200$ epochs), we assume that the best configuration for fixed H hidden nodes has been found and then we begin the training with $H + 1$ hidden nodes.
- If two consecutive additions of hidden nodes do not yield a better result, it is assumed that the best configuration has been found for the data set and then training is stopped.
- The items whose class labels have been changed from their initial labels are identified, and they are then treated as noise and removed from the data set.

III. EXPERIMENTS

We tested ANR on 12 data sets drawn from the UCI machine learning data repository [19]. The performance was evaluated based on accuracy of identifying mislabeled items, error of incorrectly identifying noisy data, and accuracy comparison between a nearest neighbor classifier with ANR (as a noise filter for training data) and one without ANR. For each parameter setting with a given data set, 20 stratified 10-fold cross-validations were conducted to achieve reliable performance estimations.

In each of 10 iterations for one stratified 10-fold cross-validation, 9 folds of data are used as the training set S and the other fold as the testing set T. We obtain a mislabeled training set S_m by mislabeling a fraction β of output classes in S using the following process. For each class i (i=1,2,...,C), βN_i instances (N_i is the 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 (same as q_j defined in Eq. (9)). With this procedure, S_m keeps the same class distribution as S, which is consistent with the assumption of random mislabeling. ANR is then applied to filter noise in S_m.

One performance measurement for ANR is the accuracy of noise identification, which is defined as the ratio of identified mislabeled items over all mislabeled items:

$$ACC_{ni} = \frac{M_{anr}^{(1)}}{\beta N} \tag{11}$$

where ACC_{ni} represents the accuracy of noise identification (ni), $M_{anr}^{(1)}$ is the number of mislabeled items that have been correctly identified by ANR as noise, and βN is the total number of mislabeled items.

Another performance measurement is the error of incorrectly identifying noisy data, which is defined as the ratio of items that have been incorrectly identified by ANR as noise over all non-mislabeled items:

$$ERR_{ni} = \frac{M_{anr}^{(2)}}{(1-\beta)N} \tag{12}$$

where ERR_{ni} represents the error of noise identification (ni), and $M_{anr}^{(2)}$ is the number of items that were not mislabeled but have been mistakenly identified by ANR as noise. Note that $(M_{anr}^{(1)} + M_{anr}^{(2)})$ is the total number of items that have been identified by ANR as noise.

The performance of ANR was also evaluated by comparing the test-set accuracies of the following two classifiers using the nearest neighbor rule [2, 20]: classifier NNR_c based on the training set S_c that have been filtered by ANR and classifier NNR_m based on the mislabeled set S_m without using ANR (both using 1-nearest neighbor). Both NNR_c and NNR_m use the same T as the testing set.

The accuracy for one stratified 10-fold cross-validation is the total number of correctly classified instances in all the 10 iterations divided by the total number of instances in the data set ($|S| + |T|$). For each data set, we conduct 20 such stratified 10-fold cross-validations and then average them.

Table 1 shows the size and other properties of the data sets; **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; **#class** is the number of classes.

Table 1: Description of 12 UCI data sets

Data Set	size	#attr	#num	#symb	#class
australian	690	14	6	8	2
balance	625	4	0	4	3
crx	690	15	6	9	2
echoc	131	9	7	2	2
ecoli	336	7	7	0	8
hayes	93	4	0	4	3
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
led7	200	7	0	7	10

Figure 1 shows simulation results on the 12 tested data sets. Four curves are displayed for each graph: "ni-acc-anr" is the accuracy of noise identification as defined by ACC_{ni} in Eq. (11); "ni-err-anr" is the error of noise identification as defined by ERR_{ni} in Eq. (12); "nnr-anr" is the accuracy of nearest neighbor classifier using ANR as a filter for its training set while "nnr-no-anr" is the one without using ANR. The graphs show how these quantities vary with different mislabeling levels (β). Each data point represents the accuracy averaged over 20 stratified 10-fold cross-validations, along with the corresponding error bar with a 95% confidence level.

The results show that ANR performs well for most of these data sets. ANR is capable of identifying a large fraction of noise while keeping a low noise identification error rate. The accuracies of the nearest neighbor classifiers using ANR are significantly higher than those without using ANR (difference between "nnr-no-anr" and "nnr-anr") for most data sets.

The amount of the performance improvement depends on the noise level. As the noise level increases, the noise identification accuracy ACC_{ni} decreases while the noise identification error ERR_{ni} increases. The accuracies of both nearest neighbor classifiers (with and without using ANR) decrease as the noise level increases, but the most performance gain (accuracy difference between the two classifiers) occurs in the middle range of noise levels (between 20% to 30%). At the noise level of 25%, the average increase in accuracy over the 12 data sets is 24.5% by using ANR for a classifier.

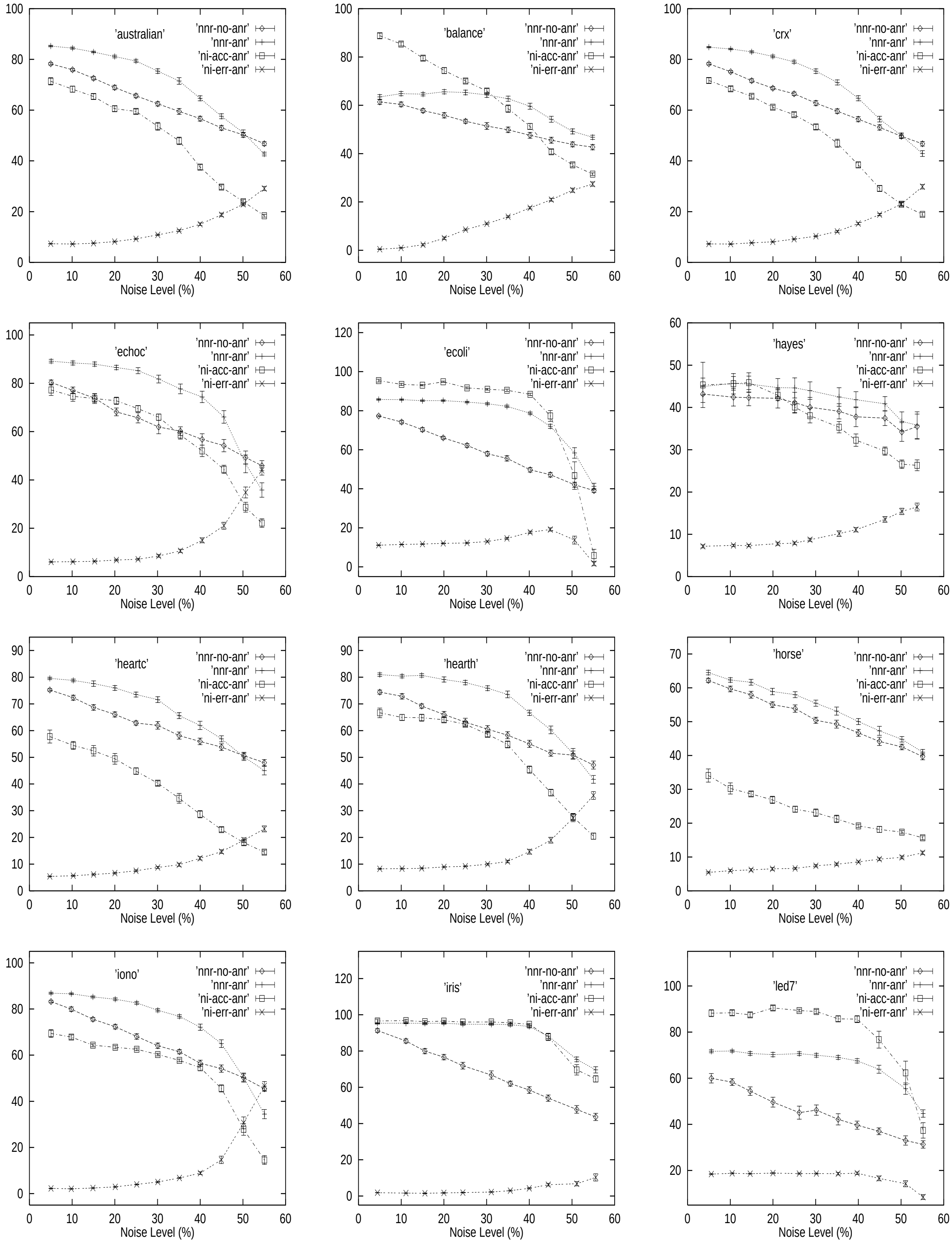

Figure 1: Simulation results on 12 data sets from UCI, where "nnr-no-anr" and "nnr-anr" represent test-set accuracies (%) of nearest neighbor classifiers without ANR and with ANR respectively, "ni-acc-anr" represents accuracy (%) of noise identification by ANR (ACC_{ni} defined in Eq. (11)), and "ni-err-anr" represents error (%) of ANR (ERR_{ni} defined in Eq. (12)).

For most data sets, ANR performs well when the noise level is in the range not exceeding 30%. However, for several data sets (*e.g., ecoli, iris, led7*), even when the noise level exceeds 30% (but smaller than 45%) ANR still performs well. When the noise level is reasonably small, non-noisy data in the training set is dominant and is capable of controlling the formation of the network architecture, which can be used by ANR to correct and identify noisy data. When the noise level reaches or exceeds 50% (half of training data is noise), as can be expected, the performance degrades significantly (as shown in Fig. 1). The performance of ANR varies with different data sets, but the improvement by using ANR as a noise filter is significant for most data sets.

IV. SUMMARY

In summary, we have presented a neural network based approach – ANR – to filter noise in data sets. In this approach, a class probability vector is attached to each instance and evolves during neural network training. ANR combines the backpropagation network with a relaxation mechanism for training. A learning algorithm is proposed to update the class probability vector based on the difference of its current value and the network output value. When the noise level in a data set is reasonable small ($< 30\%$), the architecture, weight settings and output values of the network are determined predominantly by those non-nosy data. This provides a mechanism to correct noisy data by aligning the class probability vector with the neural network output.

We have tested the performance of ANR on 12 data sets drawn from the UCI data repository by evaluating its capacity of identifying noise and by comparing the accuracies of two versions of nearest neighbor classifiers, one using the training set filtered by ANR and the other using the training set without filtering. The results show that ANR is able to remove a large fraction of noise with a small error of misidentifying non-noise data. An average increase in accuracy of 24.5% can be achieved at a noise level of 25% by using ANR as a training data filter for a nearest neighbor classifier, as compared to the one without using ANR.

References

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

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

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

[4] 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.

[5] Aha, D. W., & 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.

[6] Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6, 37-66.

[7] Wilson, D. Randall, and Martinez, Tony R. (1997). Instance Pruning Techniques. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kaufmann Publishers, San Francisco, CA, 403-411.

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

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

[10] 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.

[11] Quinlan, J. R., (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufman, Los Altos, CA.

[12] 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.

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

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

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

[16] Zeng, Xinchuan and Martinez, Tony R. (2001). An Algorithm for Correcting Mislabeled Data. *Intelligent Data Analysis*, 5, 491-502.

[17] Hopfield, J. J. & Tank, D. W. (1985). Neural Computations of Decisions in Optimization Problems. *Biological Cybernetics*, 52, 141-152.

[18] Hornik, K., Stinchcombe M. and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 359-366.

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

[20] Cover, T. M., & Hart, P. E. (1967). Nearest Neighbor Pattern Classification. Identifying and eliminating mislabeled training instances. *IEEE Transactions on Information Theory*, 13, 21-27.