A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm

Dietrich Wettschereck

Dearborn Hall 303
Department of Computer Science
Oregon State University
Corvallis, OR 97331-3202
USA
wettscd@cs.orst.edu

Abstract. Algorithms based on Nested Generalized Exemplar (NGE) theory [10] classify new data points by computing their distance to the nearest "generalized exemplar" (i.e. an axis-parallel multidimensional rectangle). An improved version of NGE, called BNGE, was previously shown to perform comparably to the Nearest Neighbor algorithm. Advantages of the NGE approach include compact representation of the training data and fast training and classification. A hybrid method that combines BNGE and the k-Nearest Neighbor algorithm, called KBNGE, is introduced for improved classification accuracy. Results from eleven domains show that KBNGE achieves generalization accuracies similar to the k-Nearest Neighbor algorithm at improved classification speed. KBNGE is a fast and easy to use inductive learning algorithm that gives very accurate predictions in a variety of domains and represents the learned knowledge in a manner that can be easily interpreted by the user.

1 Introduction

Salzberg [10] describes a family of learning algorithms based on nested generalized exemplars (NGE). In NGE, an exemplar is a single training example and a generalized exemplar is an axis-parallel hyperrectangle that may cover several training examples. These hyperrectangles may overlap or nest. The NGE algorithm grows the hyperrectangles incrementally as training examples are processed. Once the generalized exemplars are learned, a test example can be classified by computing the distance between the example and each of the generalized exemplars. If an example is contained inside a generalized exemplar, the distance to that generalized exemplar is zero. The class of the nearest generalized exemplar is assigned to the test example.

The NGE approach can be viewed as a hybrid of nearest neighbor methods and propositional Horn clause rules. Like nearest neighbor methods, a distance metric is applied to match test examples to training examples. But like Horn clause rules, training examples can be generalized to be axis-parallel hyperrectangles. Advantages of the NGE approach over other methods include fast training, few user-defined parameters (one for NGE and none for BNGE, see below), compact representation of the training data, and the ability to interpret hyperrectangles as prototypes of the task. These prototypes can be used to justify or explain decisions made by the classifier. The NGE algorithm, as described in this paper, differs from Nearest Neighbor methods, including those with reduced exemplar sets [3, Chapter 6], through its rectangular bias and its ability to distinguish queries that fall inside of some hyperrectangles from those that are not covered by any hyperrectangle. It differs from Parzen Windows [9] in that hyperrectangles can be of varying sizes and edge lengths may be unequal.

Salzberg [10] achieved promising classification results with NGE in three domains. However, Wettschereck & Dietterich [14] have reported that when tested in 11 additional domains, NGE does not perform as well as the nearest neighbor (NN) algorithm in 6 out of the 11. Wettschereck & Dietterich [14] point out some weaknesses of NGE and suggest ways to improve its performance. The single most successful improvement in predictive accuracy can be achieved by elimination of overlapping hyperrectangles. A weakness of the NGE algorithm is the need for expensive cross-validation to estimate the best value for the one user-defined parameter of NGE. This necessity for cross-validation can be avoided by training NGE in batch mode. This improved version of NGE, called BNGE ("Batch" NGE), is either superior to or indistinguishable from NGE in all domains tested. It was concluded that creation of overlapping hyperrectangles should be avoided when training NGE and that BNGE should be employed in place of NGE in any situation where batch learning is appropriate. The main advantages of BNGE over NGE are that BNGE often yields better data compression and has no user-specified parameters thus making it easier to use. BNGE compares favorably to approaches based on neural networks in training time and experimenter's effort required to construct a classifier. For example, it takes only approximately 1 hour on a Sparc-2 workstation to construct a BNGE classifier for the 16000 training examples of the Letter Recognition domain described in Table 1. A classifier constructed by BNGE is generally faster than a Nearest Neighbor approach while neural networks often find more compact representations of the training data thus making them faster at classification time.

The potential advantages of NGE algorithms (data compression, fast learning and classification, interpretability of exemplars) are significant, but classification accuracy is still not satisfactory. This paper proposes and tests two additional modifications to the NGE algorithm. First, the amount of memory required by NGE algorithms can be further reduced after the classifier is constructed by pruning hyperrectangles that were not generalized during the training period. These trivial hyperrectangles contribute little to NGE's predictive accuracy, but they consume memory and increase the time needed for classification. Second, to achieve better classification accuracy, a hybrid algorithm that uses BNGE in areas of the input space that are covered by hyperrectangles and that uses kNN otherwise, is introduced and evaluated. We call this algorithm KBNGE. It is shown that, while BNGE is significantly inferior to the kNN algorithm in 6 out of 11 domains, KBNGE is inferior to the kNN algorithm in only 1 domain and yields better predictive accuracy in 2 other domains. Furthermore, KBNGE is shown to be faster than kNN in all domains. The KBNGE algorithm can be seen as a generalized Nearest Neighbor algorithm. Nearest Neighbor algorithms play an important role in inductive machine learning because of their simplicity and their ability to give highly accurate predictions after a short learning phase. The KBNGE algorithm shares these advantages and offers, in addition, fast classification and a compact representation of the most salient parts of the task learned.

1.1 The NGE Algorithm

Figure 1 summarizes the NGE algorithm following closely Salzberg's definition of NGE. NGE constructs hyperrectangles by processing the training examples sequentially. It is initialized by randomly selecting a user-defined number of seed training examples and constructing trivial (point) hyperrectangles for each seed. Each subsequent training example is first classified according to the existing set of hyperrectangles by computing the distance from the example to each hyperrectangle. If the class of the nearest hyperrectangle and the training example coincide, then the nearest hyperrectangle is extended to include the training example, otherwise the second nearest hyperrectangle is tried (this is called the second match heuristic). Should both the first and second nearest hyperrectangles have different classes than the training example, then the training example is stored as a new (trivial) hyperrectangle. A query is classified according to the class of the nearest hyperrectangle. Distances are computed as follows: If an example lies outside of all existing hyperrectangles, a distance is computed according to a distance metric. If the example falls inside a hyperrectangle, its distance to that hyperrectangle is zero. If the example is equidistant to several hyperrectangles, the smallest of these is taken to be the "nearest" hyperrectangle.

In our implementation of NGE, we first make a pass over the training examples and normalize the values of each feature into the interval [0,1] (linear normalization [1]). Features of values in the test set are normalized by the same scaling factors (but note that they may fall outside the [0,1] range). Aside from this scaling pass, the algorithm is entirely incremental.

The original NGE algorithm was designed for continuous features only. Discrete and symbolic features require a modification of the distance computation for NGE. We adopted for NGE the policy that for each symbolic or discrete feature the set of covered feature values is stored for each hyperrectangle (analogous to storing the range of feature values for continuous features). A hyperrectangle then covers a certain feature value if that value is a member of the covered set. If a hyperrectangle is generalized to include a missing discrete or symbolic feature, then a flag is set such that the corresponding feature of the hyperrectangle will cover any feature value in the future.

Each hyperrectangle H^j is labeled with an output class. The hyperrectangle is represented by its lower left corner (H^j_{lower}) and its upper right corner (H^j_{upper}) for continuous features and by the set of values (H^j) covered for symbolic or discrete features. The distance between H^j and an example E with features f_1 through $f_{nFeatures}$ is defined as follows:

$$D(E,H^j) = \sqrt{\sum_{i=1}^{nFeatures} d_{f_i}(E,H^j)^2}$$

where:

$$\text{if } (f_i \text{ continuous}) \qquad d_{f_i}(E,H^j) = \begin{cases} E_{f_i} - H^j_{upper,f_i} & \text{if } E_{f_i} > H^j_{upper,f_i} \\ H^j_{lower,f_i} - E_{f_i} & \text{if } H^j_{lower,f_i} > E_{f_i} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{else} \qquad d_{f_i}(E,H^j) = \begin{cases} 1 & \text{if } E_{f_i} \in H^j \\ 0 & \text{otherwise} \end{cases}$$

Choice of the distance metric can significantly influence the performance of any distance-based machine learning algorithm in domains with continuous features [15]. Euclidean distance (L^2-norm) is used in this paper for NGE. Note that the decision whether a query is inside or outside of a hyperrectangle is independent of the metric. On the other hand, the metric may heavily influence the number and shape of hyperrectangles constructed.

1.2 The Nearest Neighbor Algorithm

One of the most venerable algorithms in machine learning is the nearest neighbor algorithm (NN, see [3] for a survey of the literature). The entire training set is stored in memory. To classify a new example, the Euclidean distance (possibly weighted) is computed between the example and each stored training example, and the new example is assigned the class of the nearest neighboring example. Better classification accuracy can often be achieved by using more than the first nearest neighbor to classify a query. The number k of neighbors to be considered is usually determined via leave-one-out cross-validation [13]. Aha [1] describes several space-efficient variations of nearest-neighbor algorithms.

1.3 Experimental Methods and Test Domains

To measure the performance of the NGE and nearest neighbor algorithms, we employed the training set/test set methodology. Each data set was randomly partitioned into a training set containing approximately 70% of the patterns and a test set containing the remaining patterns (see also Table 1). After training on the training set, the percentage of correct classifications on the test set was measured. The procedure was repeated a total of 25 times to reduce statistical variation. In each experiment, the algorithms being compared were trained (and tested) on identical data sets to ensure that differences in performance were due entirely to the algorithms.

```
1.  Build an NGE classifier (input: number $s$ of seeds):
2.    Initialization:        /* assume training examples are given in random order */
3.      for each of the first $s$ training examples $E^s$ call createHyperrectangle($E^s$)
4.    Training:
5.      for each remaining training example $E$:
6.        find the two $H^j$ with $D(E, H^j)$ minimal
7.                       /* in case of ties, choose the two $H^j$ with minimal area */
8.          call these hyperrectangles $H^{closest}$ and $H^{second\ closest}$
9.          if (class($E$) == class($H^{closest}$))                generalize($H^{closest}$,$E$)
10.         else if (class($E$) == class($H^{second\ closest}$)) generalize($H^{second\ closest}$,$E$)
11.         else                                              createHyperrectangle($E$)

12. Generalize a hyperrectangle:
13.     generalize($H$, $E$)
14.       for all features of E do:
15.         $H_{upper,f_i} = \max(H_{upper,f_i}, E_{f_i})$
16.         $H_{lower,f_i} = \min(H_{lower,f_i}, E_{f_i})$
17.       replMissFeatures($H$,$E$)

18. Create a hyperrectangle:
19.     createHyperrectangle($E$)
20.       $H_{upper} = E$
21.       $H_{lower} = E$
22.       replMissFeatures($H$, $E$)

23. Replace missing features in a hyperrectangle:
24.     replMissFeatures($H$,$E$)
25.       for all features of $E$ do:
26.         if (feature $i$ of $E$ is missing)
27.           $H_{upper,f_i} = 1$
28.           $H_{lower,f_i} = 0$

29. Classification of a test example:
30.     classify($E$)
31.       output: class($H^j$) with $j$ = argmin$_i$ D($E$,$H^i$)
32.                   /* in case of ties, choose $H^j$ out of all ties with minimal area */
```

Fig. 1. Pseudo-code describing construction of an NGE classifier and classification of test examples. H generally denotes a hyperrectangle and E an example.

We have reported the average percentage of correct classifications and its standard error. Two-tailed paired t-tests were conducted to determine the level of significance at which one algorithm outperformed another. A performance difference was considered significant when the p-value was smaller than 0.05.

Eleven domains of varying size and complexity were used to compare the performance of NGE to nearest neighbor. The first three data sets are two dimensional data sets especially constructed in Wettschereck & Dietterich [14] to evaluate NGE. The decision boundaries in Tasks A and C are rectangular, while in Task B the boundary is diagonal. The data sets for the other eight domains were obtained from the UC-Irvine repository [1, 6] of machine learning databases. Table 1 describes some of the characteristics of the domains used. There are a few important points to note: (a) the Waveform-40 domain is identical to the Waveform-21 domain with the addition of 19 irrelevant features (having random values), (b) the Cleveland database [4] contains some missing features, and (c) many input features in the Hungarian database [4] and the Voting Record database are missing.

Table 1. Domain characteristics (modified from Aha (1990)). B = Boolean, C = Continuous, N = Nominal.

Domain	Training Set Size	Test Set Size	Number and Kind of Features	Number of Classes
Task A	350	150	2 C	2
Task B	350	150	2 C	2
Task C	350	150	2 C	10
Iris	105	45	4 C	3
Led-7 Display	200	500	7 B	10
Waveform-21	300	100	21 C	3
Waveform-40	300	100	40 C	3
Cleveland	212	91	5 C, 3 B, 5 N	2
Hungarian	206	88	5 C, 3 B, 5 N	2
Voting	305	130	16 B	2
Letter recog.	16000	4000	16 C	26

2 Pruning

One of the main advantages of NGE and its variations when compared to the Nearest Neighbor algorithm is that NGE often finds a more compact representation of the data. For example, if all training patterns of one class can be described by a single rectangle, then BNGE will find that rectangle. Often, however, NGE and BNGE store trivial point-hyperrectangles. Since these hyperrectangles cover no significant part of the input space, they may contribute little to the generalization accuracy of NGE while using up memory and slowing down the classifier during classification. Figure 2 describes the effect on the performance of BNGE if hyperrectangles that cover only one training example (BNGE_{p1}), at most two training examples (BNGE_{p2}), or at most five training examples (BNGE_{p5}) were removed from the classifier prior to classification of the test examples. Pruning of exemplars that cover only one training example (i.e. were never generalized) significantly decreased the performance of BNGE only in the Cleveland domain. However, in the remaining ten domains, pruning of un-generalized hyperrectangles had little effect on the predictive accuracy of BNGE (and NGE, experiments not shown). The largest reduction in memory was achieved by removal of un-generalized exemplars in the Waveform domains. BNGE stored approximately 140 hyperrectangles in these domains. On average, 4 hyperrectangles remained after pruning in these domains, while no loss in predictive accuracy was observed. More than 75% of the hyperrectangles could be pruned in the Letter Recognition domain, also with no significant loss in predictive accuracy. A significant reduction in storage was observed in all domains except in Task A, where BNGE had found the smallest possible representation of the training data without pruning. The slight improvement in performance in the Hungarian and Voting domains indicates that un-generalized exemplars may often represent noisy training examples. Pruning could therefore be used to filter out noisy exemplars to improve speed and accuracy of the classifier.

It is important to note that the main purpose of the pruning technique described here is to find a more compact BNGE classifier with somewhat similar classification accuracy. Since this approach is a modification of BNGE's bias, it may also suffer from the same problems as other pruning techniques [11] with respect to classification accuracy. However, pruning never increases the amount of storage required by BNGE.

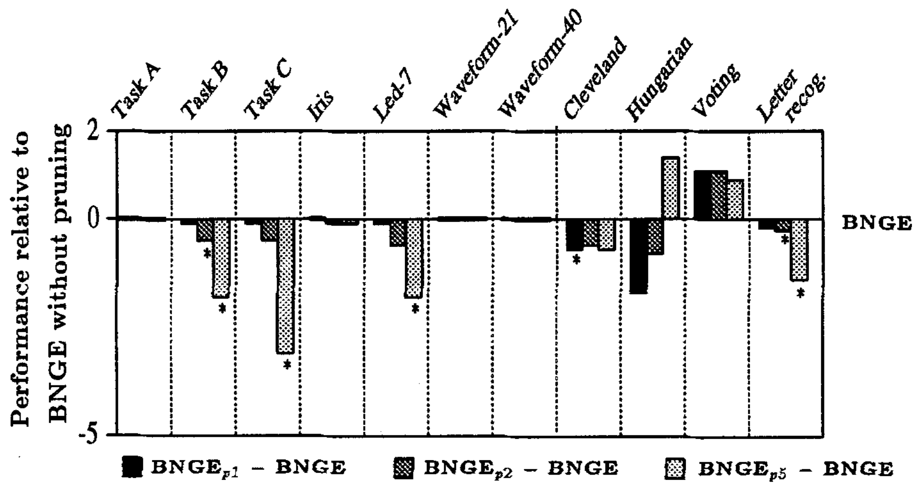

Fig. 2. Performance differences between BNGE without pruning and BNGE with different levels of pruning on the test set. The subscript px indicates that hyperrectangles which cover at most x training examples were removed before the classifier was tested. Performance relative to BNGE without pruning is shown. These differences (*) are statistically significant ($p < 0.05$).

Through inspection of hyperrectangles that were constructed by BNGE$_{p5}$ in the Hungarian domain, we could, for example, determine that 4 of the 13 input features in this domain are completely irrelevant and that the typical patient who is likely to suffer from heart disease can be described as a middle-aged male experiencing atypical angina or asymptomatic chest pains with exercise-induced angina and a medium to high ST depression induced by exercise relative to rest. After pruning in the Voting Records domain only one hyperrectangle for Republicans and one for Democrats was left to describe the voting patterns of the members of the US congress in the legislative period described in that data set. In particular, the votes on adoption of the budget and the physician fee freeze were most informative, and 11 of the 16 features were irrelevant.

3 A Hybrid Algorithm – KBNGE

Figure 3 (and Table 4) compares the performances of the Nearest Neighbor algorithm (NN), BNGE, and KBNGE (see below) to those of the k-Nearest Neighbor algorithm (kNN, k determined via leave-one-out cross-validation [13]) in eleven domains. Shown are relative performance differences between kNN and the other algorithms compared. An asterisk appears when the difference is statistically significant. BNGE (without pruning) outperforms the first Nearest Neighbor algorithm (NN) in 3 domains and is outperformed by NN in 4 other domains. The k-Nearest Neighbor algorithm outperforms NN and BNGE in 6 domains, and BNGE shows better generalization performance than kNN in Tasks A and C.

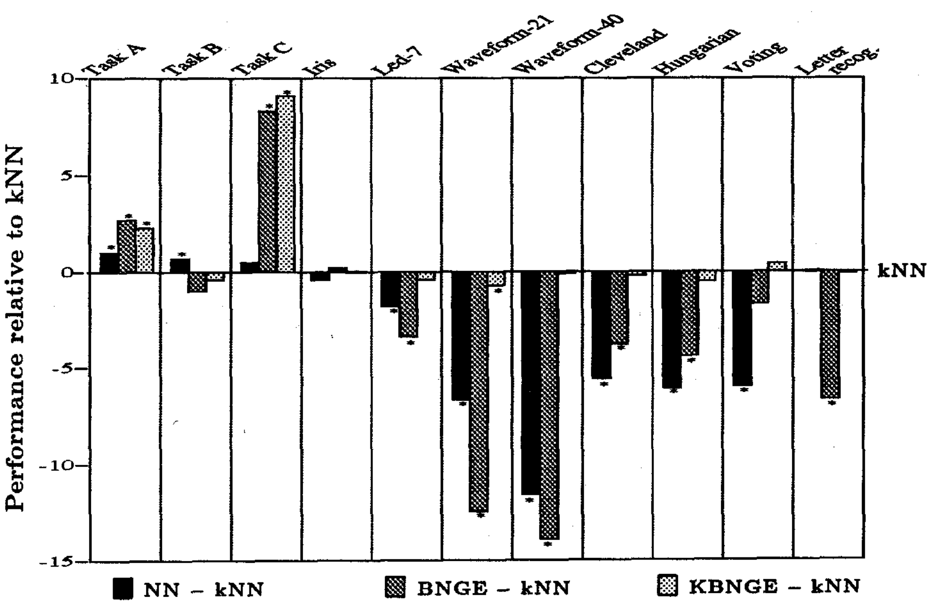

Fig. 3. Performance of NN, BNGE, and KBNGE relative to kNN. An * indicates that the performance difference between kNN and the other algorithms is statistically significant ($p < 0.05$). See Table 4 in appendix for detailed numbers.

Table 2 shows results from a set of experiments that were conducted to determine where BNGE would commit most of its errors. Displayed are the percentages of test examples that were covered by at least one hyperrectangle (column 2), the percentage of these test examples that were misclassified (column 3), the percentage of test examples that were outside of all hyperrectangles (column 4), and the percentage of these "outside"-test examples that were misclassified (column 5). BNGE commits significantly more errors when predicting

Table 2. Comparison of correctness of classifications made by BNGE inside of hyperrectangles versus outside. Numbers are based on a single repetition.

	Percentage of test examples			
	classified		classified	
Domain	inside	of these incorrect	outside	of these incorrect
Task A	93.3	0.0	6.7	20.0
Task B	80.7	4.1	19.3	10.3
Task C	76.0	0.9	24.0	38.9
Iris	80.0	0.0	20.0	22.2
Led-7	89.4	26.0	10.6	56.6
Wave-21	26.0	11.5	74.0	33.8
Wave-40	12.0	8.3	88.0	35.2
Cleveland	30.8	3.6	69.2	33.3
Hungarian	45.5	12.5	54.5	22.9
Voting	83.0	2.8	17.0	13.6
Letter recogn.	68.4	1.0	31.6	32.8

the class of test examples that are not inside any hyperrectangles than when predicting the class of test examples that are inside hyperrectangles. Hence, we decided to use a different classifier for any queries that are not covered by hyperrectangles. In the experiments described in Fig. 3, the k-Nearest Neighbor algorithm was used as that classifier.[1] Un-generalized exemplars were pruned to accelerate the classifier. We call this hybrid method KBNGE to indicate that it is a combination of BNGE$_{p1}$ and kNN.

KBNGE has two main advantages over kNN: 1) Areas that clearly belong to only one class are represented by only one hyperrectangle. This can often lead to significantly faster classification times. The computationally more expensive kNN classifier is only used to classify queries in areas with complex decision boundaries or high levels of noise. 2) The hyperrectangles constructed can be inspected and interpreted by the user. This may lead to a higher acceptance of the decisions made by KBNGE than of those made by kNN or by neural networks, for example. Figure 3 indicates that KBNGE has the same predictive accuracy as kNN in 8 domains, is outperformed by kNN in the Cleveland domain, and outperforms kNN in Tasks A and C (level of significance $p < 0.05$). KBNGE is faster than kNN at classification time if the following rough formula is satisfied:

$$\begin{aligned}&\#\textit{(training examples)} \; > \\ &\qquad 2 \times \#\textit{(hyperrectangles)} + (1-x) \times \#\textit{(training examples)} \qquad (1)\end{aligned}$$

where x is the percentage of test cases classified by BNGE. All other $(1-x)\%$ of the test cases are classified by the kNN classifier. The value of x differs from domain to domain (see Table 3) and must be determined empirically. The justification for the formula is that kNN has to compare each query to all training

[1] Once again, k values were determined via leave-one-out cross-validation [13]. Values of k varied significantly across domains and for different random partitions of the training data within most domains (see also Table 5).

examples, while KBNGE must compare each query to all hyperrectangles (a comparison to a hyperrectangle is approximately twice as expensive as a comparison to a training example), and if the query is not covered by any hyperrectangles (which happens in $(1-x)\%$ of the cases), KBNGE must compare the query to all training examples.[2] Formula (1), evaluated with the data displayed in Table 3, shows that KBNGE is faster than kNN in all domains tested. It can also be seen from that table that in domains with large amounts of noise (Waveform, Cleveland, and Hungarian), kNN is used very often, which indicates that a noise tolerant version of BNGE should help to improve the speed of KBNGE even further.

The number of training (and test) examples covered by any hyperrectangle differs significantly within and across domains. For example, a single hyperrectangle is always constructed in the Iris domain to cover all instances of Iris Setosa, while in the Cleveland and Hungarian domains a single hyperrectangle never covers more than approximately 30% of the training (20% of the test) examples of its class.

Table 3. Complexity of KBNGE. Shown are the number of hyperrectangles constructed by the BNGE part of KBNGE, the ratio of the number of hyperrectangles to the number of training examples (in parentheses), and the average percentage of test examples which were covered by at least one hyperrectangle. Numbers are means (± standard error) over 25 experiments.

Domain	Number of hyperrectangles constructed by BNGE$_{p1}$		Percentage of test examples classified by BNGE$_{p1}$*
Const A	4.0±0.0	(1%)	96.7%
Const B	18.2±0.6	(6%)	82.6%
Const C	22.4±0.7	(7%)	83.8%
Iris	4.6±0.3	(4%)	77.9%
Led-7	31.0±0.6	(16%)	77.0%
Wave-21	4.0±0.1	(1%)	22.7%
Wave-40	3.1±0.1	(1%)	14.4%
Cleveland	23.0±0.8	(11%)	35.8%
Hungarian	25.3±0.5	(12%)	49.8%
Voting	12.3±0.7	(4%)	78.0%
Letter recogn.	663.5±2.7	(4%)	68.3%

* All other test examples were classified by kNN

[2] Formula (1) assumes that retrieval of training data is not conducted more efficiently with methods such as k-d trees [5] or box-trees [7]. In domains with many relevant features, neither k-d trees nor box-trees provide significant speedups over serial search. In domains where they do provide speedups, KBNGE could also be accelerated by storing the hyperrectangles in a box tree.

4 Conclusions and Discussion

A batch version of NGE without overlapping hyperrectangles, called BNGE, was introduced in Wettschereck & Dietterich [14] and shown to significantly outperform NGE in most domains tested. A simple pruning technique, which significantly reduces the amount of storage required by NGE and BNGE, is introduced in this paper. This significant simplification of the classifier had no negative effect on the predictive accuracy of BNGE (and NGE) in 10 of the 11 domains tested. A very compact representation of the training data is found after a classifier is constructed with BNGE and pruned. This representation can be used to do the following:

- Re-evaluate the representation. For example, we were able to determine in several domains through inspection of the hyperrectangles after training and pruning that some of the input features were irrelevant.
- Learn about the task. If only a few hyperrectangles are necessary to describe a task, then it can be said that it has a low level of noise and that one might be able to construct a rule-based system from the hyperrectangles to solve the task. If a large number of small hyperrectangles is necessary, then the task at hand is either extremely complex or the input representation is not powerful enough and should be modified.
- Assign levels of confidence to decisions. Queries that fall inside of hyperrectangles constructed by BNGE are significantly more likely to be classified correctly than queries outside of all hyperrectangles.
- Determine which regions of the input space are not adequately covered by training examples. This could prompt the experimenter either to collect more data or to clearly define which inputs can be processed by the system and which should be rejected. The ability for the user to easily interpret exemplars as prototypes of the task to be learned is a significant advantage of hyperrectangular based methods over such methods as kNN, neural networks, or decision trees.

A hybrid method, called KBNGE, that uses BNGE in areas that clearly belong to one output class and kNN otherwise was introduced and shown to have accuracy similar to kNN at improved classification speed in a large number of applications. In the majority of the domains tested, over 70% of the test examples were classified by the hyperrectangular based part of KBNGE, thus making it significantly faster than kNN at classification time and enabling the system to justify most of its decisions in a manner that can be easily understood by the user. Note that the pruning technique used by KBNGE (un-generalized hyperrectangles are removed) influences the classification accuracy of KBNGE only for queries that perfectly match a given trivial hyperrectangle and only if $k \neq 1$. In all other cases, pruning only affects the speed of KBNGE.

A flaw of the current version of BNGE is that it constructs hyperrectangles only in those parts of the input space that contain no noisy patterns. Future work will introduce noise tolerance into the BNGE algorithm by introducing a mechanism for accepting merges of hyperrectangles even if examples of other classes would be covered. A conceivable approach would be Omohundro's bottom-up model merging approach [8], for example.

The KBNGE algorithm exhibits classification accuracies comparable to the best known accuracies, it is fast in training and testing time, and it is easy to use. We believe KBNGE is an important tool to include in the set of commonly used machine learning algorithms.

Acknowledgements

I thank Steven Salzberg for providing assistance during the implementation of NGE. I also thank Thomas Dietterich, Kathy Astrahantseff, Bill Langford, and the anonymous reviewers for comments made on earlier drafts of this manuscript. This research was supported in part by NSF Grant IRI-8657316, NASA Ames Grant NAG 2-630, and gifts from Sun Microsystems and Hewlett-Packard.

References

1. Aha, D.W.: A Study of Instance-Based Algorithms for Supervised Learning Tasks. Technical Report, University of California, Irvine (1990)
2. Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.h., Rosen, D.B.: Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps. IEEE Transactions on Neural Networks **3** (1992) 698–713
3. Dasarathy, B.V.: Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. IEEE Computer Society Press (1991)
4. Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, K., Sandhu, S., Guppy, K., Lee, S., Froelicher, V.: Rapid searches for complex patterns in biological molecules. American Journal of Cardiology **64** (1989) 304–310
5. Friedman, J.H., Bentley J.L., Finkel, R.A.: An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Transactions on Mathematical Software. **3** (1977) 209–226
6. Murphy, P.M., Aha, D.W.: UCI Repository of machine learning databases [Machine-readable data repository]. Technical Report, University of California, Irvine (1991)
7. Omohundro, S.M.: Five Balltree Construction Algorithms. Technical Report, International Computer Science Institute, Berkeley, CA (1989)
8. Omohundro, S.M.: Best-First Model Merging for Dynamic Learning and Recognition. Neural Information Processing Systems **4** San Mateo California: Morgan Kaufmann Publishers, INC. (1992) 958–965
9. Parzen, E.: An estimation of a probability density function and mode. Ann. Math. Stat. **33** (1962) 1065–1076
10. Salzberg, S.: A Nearest Hyperrectangle Learning Method. Machine Learning **6** (1991) 277–309
11. Schaffer, C.: Overfitting Avoidance as Bias. Machine Learning **10** (1993) 153–178
12. Simpson, P.K.: Fuzzy min-max neural networks: 1. Classification. IEEE Transactions on Neural Networks **3** (1992) 776–786
13. Weiss, S.M., Kulikowski, C.A.: Computer Systems that learn. San Mateo California: Morgan Kaufmann Publishers, INC. (1991)

14. Wettschereck, D., Dietterich, T.G.: An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangles Algorithms. Machine Learning (to appear)
15. Wettschereck, D.: A Study of Distance-Based and Local Machine Learning Algorithms. Ph.D. Thesis. Oregon State University, OR (to appear)

Appendix

Table 4. Percent accuracy (± standard error) on test set. Shown are mean performances over 25 repetitions, standard error. These ($\star$ † •) differences to kNN are statistically significant.

	Performance			
Domain	NN	kNN	BNGE	KBNGE
Const A	97.7±0.4*	96.7±0.4	99.4±0.1†	99.0±0.2†
Const B	97.9±0.3$^{\bullet}$	97.2±0.5	96.2±0.4	96.8±0.3
Const C	83.5±0.7	83.0±0.7	91.3±0.4†	92.1±0.3†
Iris	95.2±0.4	95.6±0.5	95.8±0.4	95.6±0.4
Led-7	70.5±0.6†	72.3±0.6	68.9±0.6†	71.9±0.6
Wave-21	75.2±1.1†	81.9±0.9	69.4±1.1†	81.2±0.7$^{\bullet}$
Wave-40	69.1±0.8†	80.7±1.1	66.8±1.1†	80.6±1.0
Cleveland	77.8±0.9†	83.4±0.5	79.6±1.1*	83.2±0.6
Hungarian	75.9±0.8†	82.0±1.0	77.6±1.1†	81.5±1.0
Voting	87.3±0.7†	93.3±0.5	91.6±1.7	93.7±0.5
Letter recognition	95.8±0.1	95.8±0.1	89.1±0.1†	95.7±0.0

†: $p < 0.001$, *: $p < 0.005$, $^{\bullet}$: $p < 0.05$

Table 5. Values of k used by KBNGE.

	k value		
Domain	min	max	average
Const A	1	99	24.7±6.7
Const B	1	27	6.5±1.5
Const C	1	5	1.6±0.2
Iris	1	18	8.0±0.8
Led-7	2	7	4.3±0.4
Wave-21	7	92	34.4±4.2
Wave-40	14	93	43.3±5.0
Cleveland	3	57	18.6±3.5
Hungarian	21	57	37.2±2.3
Voting	3	10	6.1±0.5
Letter recognition	1	1	1.0±0.0