Machine Learning, ??, 1–25 (??)

An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms

DIETRICH WETTSCHERECK wettscd@cs.orst.edu
THOMAS G. DIETTERICH tgd@cs.orst.edu
Computer Science Department, Oregon State University, Corvallis, Oregon, OR 97331

Editor: Richard Sutton

Abstract. Algorithms based on Nested Generalized Exemplar (NGE) theory (Salzberg, 1991) classify new data points by computing their distance to the nearest "generalized exemplar" (i.e., either a point or an axis-parallel rectangle). They combine the distance-based character of nearest neighbor (NN) classifiers with the axis-parallel rectangle representation employed in many rule-learning systems. An implementation of NGE was compared to the k-nearest neighbor (kNN) algorithm in 11 domains and found to be significantly inferior to kNN in 9 of them. Several modifications of NGE were studied to understand the cause of its poor performance. These show that its performance can be substantially improved by preventing NGE from creating overlapping rectangles, while still allowing complete nesting of rectangles. Performance can be further improved by modifying the distance metric to allow weights on each of the features (Salzberg, 1991). Best results were obtained in this study when the weights were computed using mutual information between the features and the output class. The best version of NGE developed is a batch algorithm (BNGE FW_{MI}) that has no user-tunable parameters. BNGE FW_{MI}'s performance is comparable to the first-nearest neighbor algorithm (also incorporating feature weights). However, the k-nearest neighbor algorithm is still significantly superior to BNGE FW_{MI} in 7 of the 11 domains, and inferior to it in only 2. We conclude that, even with our improvements, the NGE approach is very sensitive to the shape of the decision boundaries in classification problems. In domains where the decision boundaries are axis-parallel, the NGE approach can produce excellent generalization with interpretable hypotheses. In all domains tested, NGE algorithms require much less memory to store generalized exemplars than is required by NN algorithms.

Keywords: Exemplar-based learning, instance-based learning, nested generalized exemplars, nearest neighbors, feature weights.

1. Introduction

Salzberg (1991) 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 Euclidean 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 output as the predicted class of 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, Euclidean distance is applied to match test examples to training examples. But like Horn clause rules, the training examples can be generalized to be axis-parallel hyperrectangles.

Salzberg reported promising classification results in three domains. However, as we report below, when NGE is tested in 11 additional domains, it gives less accurate predictions in many of these domains when compared to the k-nearest neighbor (kNN) algorithm. The goal of this paper is to demonstrate this performance, understand its causes, and test algorithm modifications that might improve NGE's performance.

The first part of the paper is devoted to a study that compares NGE and kNN.[1] To compare these algorithms fairly, it is necessary to find optimal settings for various parameters and options in NGE and kNN. Hence, we first describe a series of experiments that study how the performance of NGE (and to a lesser extent, kNN) is determined by several key parameters and options including the number of starting seeds, the treatment of un-generalized exemplars, and the treatment of nominal feature values. We then present results showing that NGE (under the best parameter settings) is substantially inferior to kNN in 9 of the 11 domains we tested and superior to kNN in 2 of these domains.

The second part of the paper attempts to diagnose and repair the causes of this performance deficit. We present several hypotheses including (a) inappropriateness of the nested hyperrectangle bias, (b) inappropriateness of the overlapping of hyperrectangle bias, and (c) poor performance of the search algorithm and heuristics for constructing hyperrectangles. Experiments are then presented that test each of these hypotheses. A version of NGE (called NONGE) that disallows overlapping rectangles while retaining nested rectangles and the same search procedure is uniformly superior to NGE in all 11 domains and significantly better in 6 of them. A batch algorithm (OBNGE) that incorporates an improved search algorithm and disallows nested rectangles (but still permits overlapping rectangles) is only superior to NGE in one domain (and worse in two). These and other experiments lead us to conclude that a major source of problems in NGE is the creation of overlapping rectangles.

We also present a batch version of NONGE, called BNGE, that is very efficient and requires no user tuning of parameters. We recommend that BNGE be employed in domains where batch learning is appropriate.

The third part of the paper takes up the issue of learning feature weights for a weighted Euclidean distance. Salzberg (1991) proposed an online weight adjustment algorithm. Data are presented showing that this algorithm often performs poorly and erratically. An alternative feature weight algorithm, based on mutual information, is shown to work well with NN, NGE, and BNGE.

The final part of the paper compares the best version of NGE (BNGE FW_{MI}) to the best version of kNN (kNN_{cv} FW_{MI}). The comparison shows that despite the improvements in NGE, it still is significantly inferior to kNN in 7 domains and significantly superior in 2 domains. When compared to single nearest neighbor (NN), the best version of NGE fares better: it is significantly superior in 3 domains and significantly inferior in 4.

The ideal behavior for a hybrid algorithm like NGE is that in domains where axis-parallel rectangles are appropriate, NGE should take advantage of them to find concise, interpretable representations of the learned knowledge. However, in domains where axis-parallel rectangles are not appropriate, NGE should behave more like a nearest neighbor algorithm. The versions of NGE that we have developed do take advantage of hyperrectangles, but they perform poorly in domains where hyperrectangles are inappropriate. Further research is needed to develop an NGE-like algorithm that can be robust in situations where axis-parallel hyperrectangles are inappropriate.

2. Algorithms and Experimental Methods

2.1. *The NGE algorithm*

Figure 1 summarizes the NGE algorithm following closely Salzberg's definition of NGE. NGE constructs hyperrectangles by processing the training examples one at a time. It is initialized by randomly selecting a user-defined number of seed training examples and constructing trivial (point) hyperrectangles for each seed. Each new 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 weighted Euclidean distance is computed. 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 chosen.

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, Aha 1990). 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 basic algorithm is entirely incremental.

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}). The distance between H^j and an example E with features f_1 through $f_{nFeatures}$ is

```
Build an NGE classifier (input: number s of seeds):
 Initialization:                  /* assume training examples are given in random order */
  for each of the first s training examples E^s call createHyperrectangle(E^s)
 Training:
  for each remaining training example E:
    find the two H^j with D(E,H^j) minimal
                                  /* in case of ties, choose the two H^j with minimal area */
      call these hyperrectangles H^closest and H^second closest
      if (compare(H^closest,E))                  generalize(H^closest,E)
      else if (compare(H^second closest,E))      generalize(H^second closest,E)
      else                                       createHyperrectangle(E)

Compare classes of a hyperrectangle and an example:
  compare(H,E)
    if (class(E) == class(H)) return true else return false

Generalize a hyperrectangle:
  generalize(H,E)
    for all features of E do:
      H_upper,f_i = max(H_upper,f_i , E_f_i)
      H_lower,f_i = min(H_lower,f_i , E_f_i)
    replMissFeatures(H,E)

Create a hyperrectangle:
  createHyperrectangle(E)
    H_upper = E
    H_lower = E
    H_area = 0
    replMissFeatures(H, E)

Replace missing features in a hyperrectangle:
  replMissFeatures(H,E)
    for all features of E do:
      if (feature i of E is missing)
        H_upper,f_i = 1
        H_lower,f_i = 0

Classification of a test example:
  classify(E)
    output: class(H^j) with j = argmin_i D(E,H^i)
                            /* in case of ties, choose H^j out of all ties with minimal area */
```

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

defined as follows:

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

where:

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

w_{f_i} weight of feature i (see Section 6)

w_{H^j} weight of hyperrectangle j, computed as:

$$w_{H^j} = \frac{\text{number of times compare}(H^j,\, E_x) \text{ was called}}{\text{number of times compare}(H^j,\, E_x) \text{ returned true}}$$

The original NGE algorithm was designed for continuous features only. Discrete and symbolic features require a modification of the distance and area 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.
The area of non-trivial hyperrectangles is then computed as follows:[2]

$$area(H) = \prod_{i=1}^{nFeatures} size(H_{f_i})$$

with $size(H_{f_i})$ computed as follows:

37. if (H_{f_i} has been generalized to include a missing feature) $size(H_{f_i}) = 1$
38. else if (f_i is continuous)
39. if ($H_{upper,f_i} == H_{lower,f_i}$) $size(H_{f_i}) = 1$
40. else $size(H_{f_i}) = H_{upper,f_i} - H_{lower,f_i}$
41. else /* f_i is a discrete or symbolic feature */
42. $size(H_{f_i}) = \frac{\text{number of values of } f_i \text{ covered by } H}{\text{number of possible values of } f_i}$

Note that the maximum possible size of a hyperrectangle is therefore 1. Furthermore, the probability of line 39 being executed should be very low, since it is unlikely that two continuous feature values match exactly. We therefore deemed it unnecessary to adjust the area of hyperrectangles for this case.

The original NGE paper also did not specify a policy for handling examples containing missing features. In the context of nearest neighbor algorithms, Aha (1990, Section 5.2.1) evaluated three methods for distance computation with missing features. We adopted his *Ignore* method. This is one of the simplest methods for dealing with missing features: If a feature of an example is missing, then the distance for that feature is 0. Furthermore, the total distance over all features is divided by the number of known features to distinguish a perfect match from a missing feature (both have distance 0).

We incorporated this methodology into the generalization procedure of NGE as follows: Whenever a hyperrectangle in NGE is extended to include an example with missing features then the range of the hyperrectangle is extended for each missing feature to cover the entire input space for that feature (see lines 20, 26, and 27-32 in Figure 1).

2.2. *The Nearest Neighbor Algorithm*

One of the most venerable algorithms in machine learning is the nearest neighbor algorithm (NN, see Dasarathy 1991 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. More generally, the k nearest neighbors are computed, and the new example is assigned the class that is most frequent among these k neighbors (we will denote this as kNN). Aha (1990) describes several space-efficient variations of nearest-neighbor algorithms.

As with NGE, we adopted Aha's *Ignore* method for handling training examples with missing features. The reader should note that the performance of both NGE and NN may change substantially if a different missing-values policy is used.

2.3. *Data Sets*

In our study, we have employed eleven data sets, of which three are synthetic and the remaining eight are drawn from the UC-Irvine repository (Murphy & Aha, 1994, Aha 1990) of machine learning databases.

The synthetic data sets were constructed to test the sensitivity of NGE to the shape of the decision boundaries and to the number of classes (Figure 2). Tasks A and C have axis-parallel decision boundaries, while Task B has a diagonal decision boundary. Tasks A and B are 2-class problems, while Task C has 10 classes.

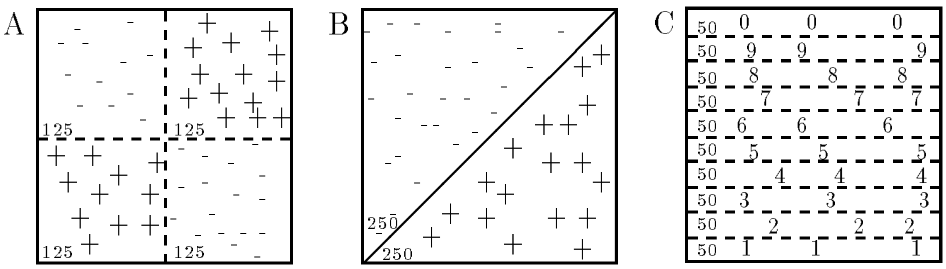


Figure 2. Artificial data sets. In A and B, + (−) indicates the location of a positive (negative) example. In C, digits indicate locations of examples from each class. The number of examples in each decision region is shown in the lower left corner of each region.

The eight Irvine data sets are summarized in Table 1. 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 (Detrano et al., 1989) contains some missing features, and (c) many input features in the Hungarian database (Detrano et al., 1989) and the Voting Record database are missing.

2.4. *Experimental Methods*

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 is 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. To generate learning curves, we follow the same procedure except that only a subset of the training set was used. The test set along each learning curve was constant, while each larger training set contained all smaller ones.

We report the average percentage of correct classifications and its standard error. Two-tailed paired t-tests were conducted to determine at what level of significance one algorithm out-performs the other. We conclude that one algorithm significantly outperforms another algorithm if the p-values obtained from the t-test are smaller than 0.05.

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
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 recognition	16000	4000	16 C	26

3. Experiments on Parameter Sensitivity

We explored the sensitivity of NGE and kNN to their user-specified parameters. For NGE, the parameters of interest are (a) the number of starting seeds, (b) the treatment of un-generalized exemplars, and (c) order of presentation of the examples. For kNN, the only parameter of interest is the number of nearest neighbors (k).

3.1. *Number of starting seeds*

Figure 3 shows the performance of NGE on Tasks A, B, and C, for several different numbers of starting seeds. The performance is shown relative to the performance of simple nearest neighbor. For Tasks A and B, where the number of classes is small, NGE's performance is particularly poor for small numbers of seeds. This contradicts Salzberg's findings (1991, page 257, first paragraph), where he states that the performance of NGE was not found to be sensitive to the size of the seed set.

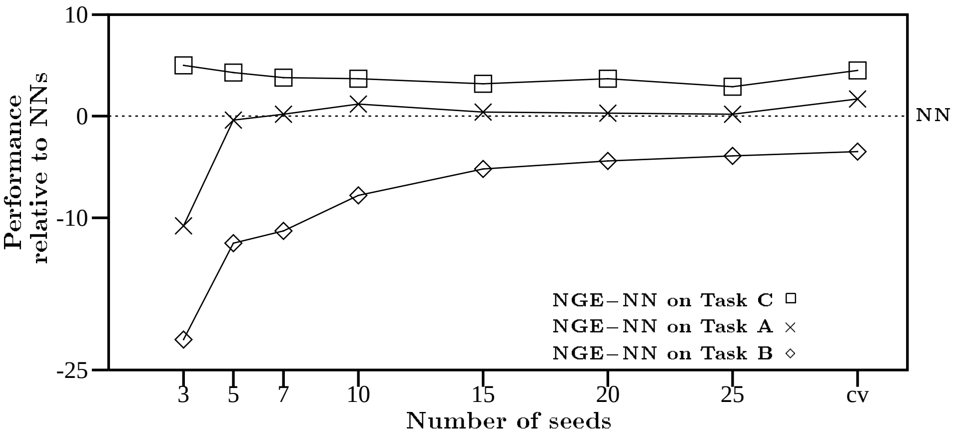


Figure 3. Performance of NGE relative to NN when NGE is initialized with varying numbers of seeds (cv: leave-one-out cross-validation). Base performance for NN is 97.6% correct in Task A, 97.0% in Task B, and 82.4% in Task C. Datapoints represent means over 25 replications with 350 training examples and 150 test examples. See Table *A1* in appendix for detailed numbers.

Figure 3 also shows, not surprisingly, that NGE performs better on Tasks C and A, where the decision-boundaries are axis-parallel, than on Task B, where the boundary is diagonal. On Task B, simple NN outperforms NGE.

At the right end of the figure (over the label "cv"), we show the performance that is obtained if leave-one-out cross-validation (Weiss & Kulikowski, 1991) is employed to determine the optimal number of seeds. This strategy worked very well, so we adopted it in all subsequent experiments (unless otherwise noted).[3] The following number of seeds was tested during each leave-one-out cross-validation run: 3, 5, 7, 10, 15, 20, and 25. Cross-validation is inherently non-incremental, so a cost of using cross-validation is that it destroys the incremental nature of NGE.

Note that if NGE is given a sufficiently large number of seeds, the algorithm becomes the simple nearest-neighbor algorithm. In the limit, there is one seed for every data point. This limit is not reached in these three tasks, however. NGE needed only approximately 6% (Task A), 13% (Task B), and 28% (Task C) of the storage that was required by NN to store the entire training set (detailed numbers are provided in Table *A1* in the appendix).

3.2. Treatment of ungeneralized exemplars

In NGE, hyperrectangles are initialized to points in the input space and should therefore have size 0 before they are generalized to non-trivial hyperrectangles (see pseudo-code in Figure 1, lines 7 & 25). We have found that in the Led-7 domain, however, initialization of the size of hyperrectangles to 1 led to a significant performance improvement (from 43.0±1.4% correct to 59.8±1.0%). This is an artifact of the Led-7 domain. (Specifically, it results from the fact that Led-7 has large numbers of training examples with identical feature vectors belonging to different classes.) The initial size of hyperrectangles had no effect on NGE's performance in any of the other domains. In the experiments reported in the remainder of the paper, we chose to initialize the size of the hyperrectangles to 0, except in the Led-7 domain, where we initialized the size to 1.

Table 2. Performance of NGE on one specific training/test set partition. Numbers shown indicate the performance of NGE when run on 25 random permutations of the same training set.

Domain	Mean	Median	Min	Max
Iris	91.8±0.8	93.3	84.4	97.8
Hungarian	78.0±0.5	77.3	71.6	83.0
Voting	93.3±0.4	93.1	89.2	96.2

3.3. Order of presentation of training data

NGE is sensitive to the order in which the training examples are presented. Table 2 shows the results of an experiment in which the training set/test set partitions were fixed while the order of presentation of the training set was randomly varied. We can see that the performance varies widely across these domains. This is a serious drawback of the NGE algorithm.[1]

Unfortunately, it is difficult to choose a "good" order for the training set. We could not find an effective way to apply cross-validation methods, for example, to select a good order. In the results reported below, a random order was selected for each run of NGE, and (as with all of the other algorithms) the mean of 25 runs is reported.

3.4. Value of k for the k-nearest neighbor algorithm

It is well-established that in noisy domains, the k-nearest neighbor algorithm performs better than simple nearest neighbor. Hence, we chose k to optimize the leave-one-out cross-validation performance of the algorithm on the training set. We tried all possible values of k (this can be done relatively efficiently). Ties were broken in favor of the smaller value of k.

4. Comparison of NGE and kNN

Figure 4 compares the performance of the k-nearest neighbor algorithm (kNN), NGE$_{cv}$ (number of seeds chosen via leave-one-out cross-validation), NGE with 3 seeds (NGE$_{3\ seeds}$)[4], and NGE when the number of seeds was increased to at most

50% of the training data (NGE$_{limit}$). The rationale behind NGE$_{limit}$ is that the amount of storage required for each hyperrectangle is twice the amount of storage required to store a single data point. Hence, when the number of seeds equals 50% of the training data, the total space required by NGE equals the space required by kNN (assuming that similar methods for dealing with ties and missing features are used). Beyond that point, NGE has no data compression advantage over kNN.[5]

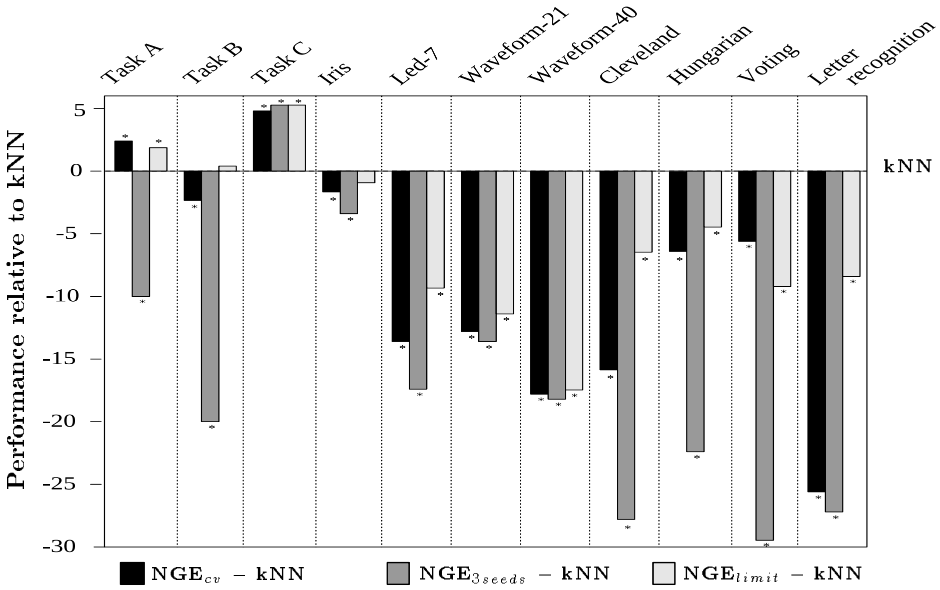


Figure 4. Performance of NGE$_{cv}$, NGE$_{3seeds}$, and NGE$_{limit}$ relative to kNN. Shown are percentage point differences between NGE$_{cv}$ and kNN, between NGE$_{3seeds}$ and kNN, and between NGE$_{limit}$ and kNN. Significance of difference between kNN and different NGE versions is indicated by a *. See Table *A2* in appendix for detailed numbers.

K-nearest neighbor outperforms NGE$_{cv}$ by a statistically significant amount in all of the eight non-constructed domains as displayed in Figure 4. In all domains, NGE$_{cv}$ achieved a significant (i.e. between 60% and 85%) compression of the data. By significantly increasing the number of seeds (NGE$_{limit}$ in Figure 4), it was possible to significantly improve the performance of NGE$_{cv}$ in Task B, and in the Led-7, Cleveland, Hungarian, and Letter recognition domains. However, NGE$_{limit}$ is still significantly inferior to kNN in performance in all but one non-constructed domain. The drop in performance in the Voting domain and Task A is due to the fact that in these domains, leave-one-out cross-validation over a small number of different seed set sizes is more beneficial than increasing the size of the seed set.[6] However, the improvement in performance by NGE$_{limit}$ comes at a high cost: In all cases where NGE's performance improved, it also used more memory than kNN.

Figure 5 shows learning curves for all of the domains. Generally, these curves have the shape that we expect from most inductive learning algorithms: Performance increases with the number of training examples and the increase levels off after

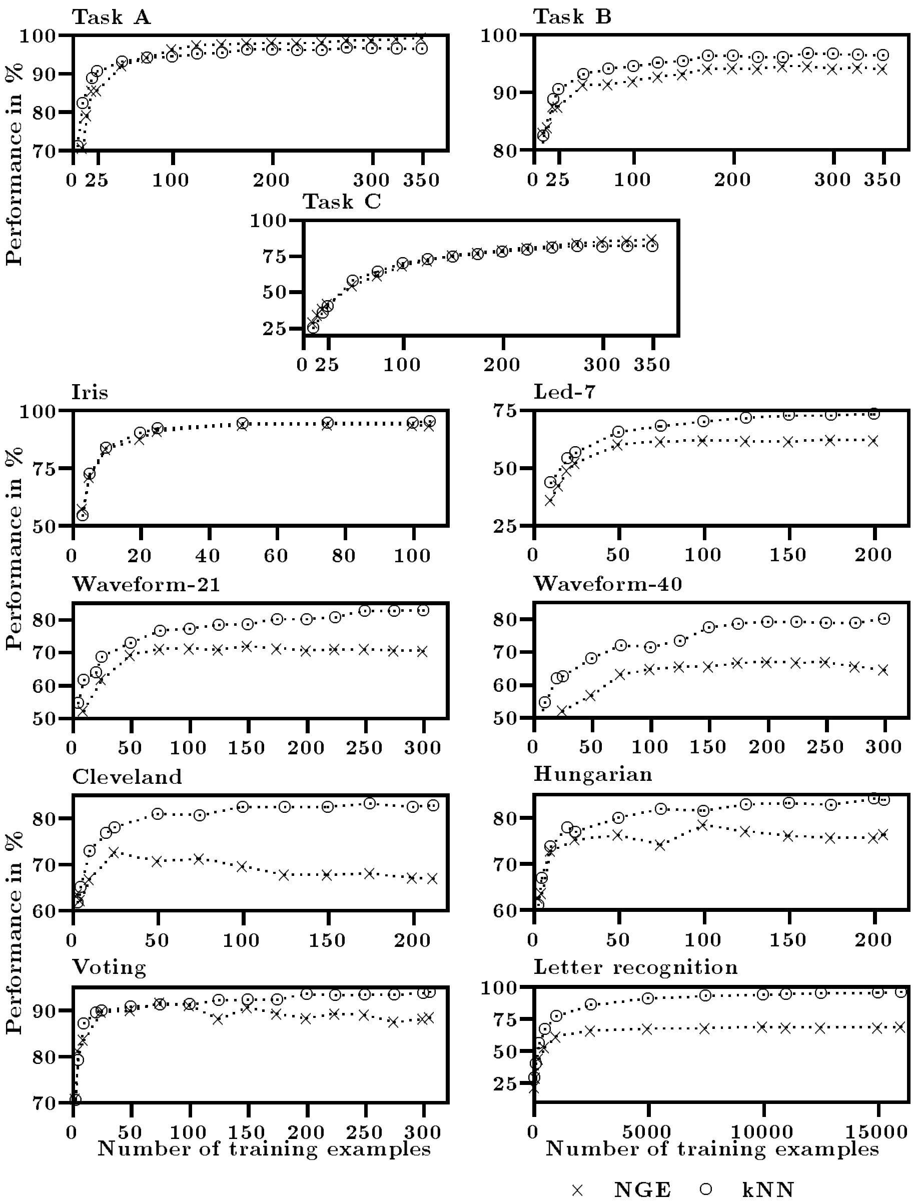


Figure 5. Performance of NGE and kNN for different numbers of training examples. Each data point denotes the mean of 25 experiments. Note the different scales on both axes of these graphs.

the training set has reached a certain size. In the Waveform, Led-7, and Letter Recognition domains, the performance of NGE$_{cv}$ levels off much earlier than kNN's. Furthermore, the graphs for the Cleveland, Hungarian, and Voting domains show some erratic behavior for NGE$_{cv}$. In the Hungarian and Voting domains, NGE$_{cv}$

reaches its (near) peak performance after only 25 training examples have been seen. For more than 25 training examples, performance of NGE_{cv} varies within two standard errors in these domains. In the Cleveland domain, the performance of NGE_{cv} peaks also at 25 examples with 72.6±1.9% correct, but then drops down to 66.9±1.8%. Through inspection of the number and sizes of hyperrectangles constructed by NGE_{cv} in these domains we were able to determine the cause of this unusual behavior: The number of hyperrectangles stored by NGE_{cv} does not grow linearly with the number of training examples. Although that is a desirable property of any machine learning algorithm, it may cause problems for NGE_{cv} since existing hyperrectangles may be generalized (extended) too often. This means that every time a hyperrectangle is enlarged it may actually become less relevant. We conclude that this behavior constitutes a serious deficiency in NGE's search and generalization procedure.

5. Possible Explanations for Inferior Performance of NGE

Given the close relationship between NGE and kNN, it is surprising that NGE performs so much worse than kNN.

With any learning algorithm, there can be two fundamental sources of problems. First, the bias of the algorithm may be inappropriate for the application domains. Second, the implementation of that bias may be poor (e.g., because poor search algorithms are employed).

Salzberg never formally defines the bias of NGE. Let us define it to be "find the minimum number of axis-parallel hyperrectangles (possibly nested or overlapping) that correctly classifies the training data."

There is some evidence that this bias is inappropriate. We know that in Task B (non-axis-parallel decision boundary), the axis-parallel bias is inappropriate (see Figure 3), but this is an artificially-constructed domain. However, Aha (1990) reports the performance of C4.5 (Quinlan, 1992) in six of the domains which are also used in this paper. C4.5 also has a rectangular bias and performs, under similar conditions, significantly better than NGE in these six domains (Aha, 1990, Section 4.3.3).[7] This suggests that the axis-parallel bias is not the cause of NGE's poor performance.

Examination of the learned hyperrectangles in several of the other domains suggests that permitting rectangles to nest and overlap is a problem. The most common form of nesting is that a large, generalized hyperrectangle is created and then many single-point rectangles are nested inside it as exceptions. This can be seen in Figure 6, which plots the number of hyperrectangles created and the number that are actually generalized to be non-point rectangles. We can see that the overwhelming majority of hyperrectangles are never generalized.

These single-point hyperrectangles are virtually never used for classifying new test examples, because if a test example falls inside a large hyperrectangle, the distance to that hyperrectangle is zero. A single-point hyperrectangle will not be used unless

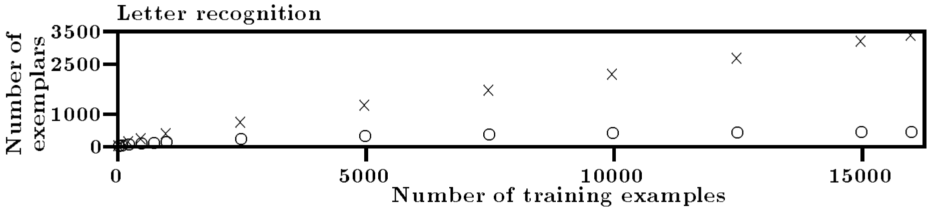


Figure 6. Number of exemplars stored by NGE when trained with 25 seeds on differently sized training sets from the Letter recognition task. Shown is the total number of hyperrectangles that were stored during training ($\times$) and the number of hyperrectangles which were generalized at least once ($\circ$). Each data point denotes the mean over 25 experiments.

either (a) the test example exactly coincides with the single-point rectangle or (b) the single-point rectangle is not nested inside another rectangle.

NGE also permits generalized rectangles to overlap, even if they don't nest. This may be a problem as well. One situation in which overlapping rectangles will be created is if the distributions of examples from two classes, A and B, overlap. The optimal decision rule (under a uniform loss function; cf. Duda & Hart, 1973) is to place the decision boundary at the point where the probability density of examples from class A equals the probability density of examples from class B. However, NGE instead arbitrarily assigns all examples in this overlapping region to one of the classes—the one which has the smaller rectangle.

In addition to these hypotheses about the bias of NGE, there is considerable evidence that the bias is not implemented well by NGE's incremental heuristic procedure. From the sensitivity experiments, we know that NGE is very sensitive to the order in which the training examples are presented. For some orders, it does very well.

In Tasks A and C, we can determine the optimal set of hyperrectangles by inspection (4 and 10, respectively). NGE does not find this optimal solution, but instead constructs an average of 10.8 ± 1.1 and 49.6 ± 1.2. In Task B, on the other hand, the optimal solution involves a large number of rather small, overlapping rectangles (one for every training example that lies near the decision boundary). However, NGE does not find this solution either. It constructs some rectangles that are too large, and then nests the smaller ones in it. Figure 7 displays the rectangles that were constructed by NGE$_{cv}$ in representative experiments in Tasks A, B, and C.

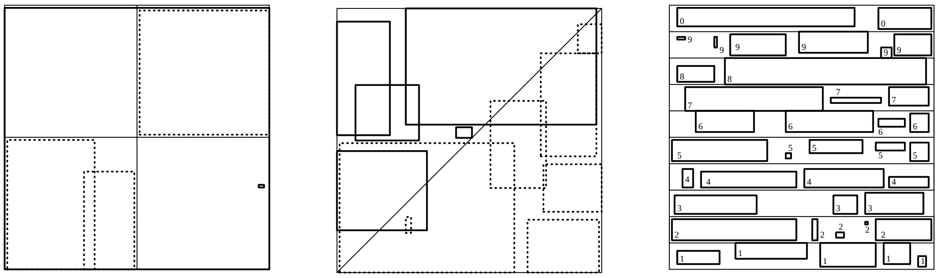
In summary, we have three hypotheses that can explain why NGE is performing poorly relative to kNN:

H1. nested rectangles,

H2. overlapping rectangles, and

H3. poor search algorithm.

To test these hypotheses, we conducted a series of experiments in which we modified NGE to eliminate one or more of these suspected problems and measured the resulting change in performance.



A B C

Figure 7. Rectangles constructed by NGE_{cv} in Tasks A, B, and C in one representative experiment. In A and B, dashed (solid) lines indicate the location of rectangles representing positive (negative) examples. In C, digits indicate the class each rectangle represents. Trivial (point) rectangles not displayed. Note that in Task A a single rectangle of class 0 covers the entire input space.

In the first experiment, we tested H1 by modifying NGE so that it produces relatively few nested rectangles but still permits overlapping rectangles. We did not otherwise change the search procedure.

In the second experiment, we tested H2 by modifying NGE so that it produces no overlapping rectangles of different classes (with the exception of rectangles entirely nested inside one another). We did not otherwise change the search procedure.

In the third experiment, we tested H3 by making a simple modification to incremental NGE to improve upon the second-match heuristic, with the goal of finding fewer hyperrectangles.

Finally, in the fourth experiment, we tested all of the hypotheses simultaneously by implementing an entirely different search procedure that completely eliminates nested rectangles and overlapping rectangles and also reduces the total number of rectangles constructed.

We now describe these experiments and their results.

5.1. *Greedy NGE (avoid nesting)*

To test H1, we want to construct a variant of NGE that avoids nesting rectangles. A major cause of nested rectangles is the second match heuristic (line 9 in Figure 1). If the nearest rectangle is of the wrong class but the second nearest rectangle is of the right class, then the second-nearest rectangle is expanded to cover the new example. In many cases, it will also cover the nearest rectangle (which could be a single point), and thus create nesting.

Salzberg (1991, Section 3.5) introduced and tested a version of NGE, called Greedy NGE, that does not have the second match heuristic. This greedy version stores an example as a new hyperrectangle whenever the closest previously stored hyperrectangle is of a different class than the example.

According to Salzberg, the second match heuristic in NGE is necessary to construct nested or overlapping hyperrectangles. This is not true: NGE may still construct overlapping or nested hyperrectangles even if its second match heuristic is disabled, because it can "grow" a hyperrectangle until it overlaps or covers another hyperrectangle. In fact, Greedy NGE did construct overlapping hyperrectangles (quite frequently) and nested hyperrectangles (in a few cases) in the experiments that we conducted.

Figure 8 shows that the predictive accuracy of Greedy NGE is significantly better than NGE's in three domains (Cleveland, Hungarian, and Voting) and significantly worse in 4 others (Task A, Task C, Waveform-21, and Waveform-40). The results in Task C and Waveform-40 are particularly poor.

Based on these, there is not much evidence that nested rectangles are a major problem for NGE.

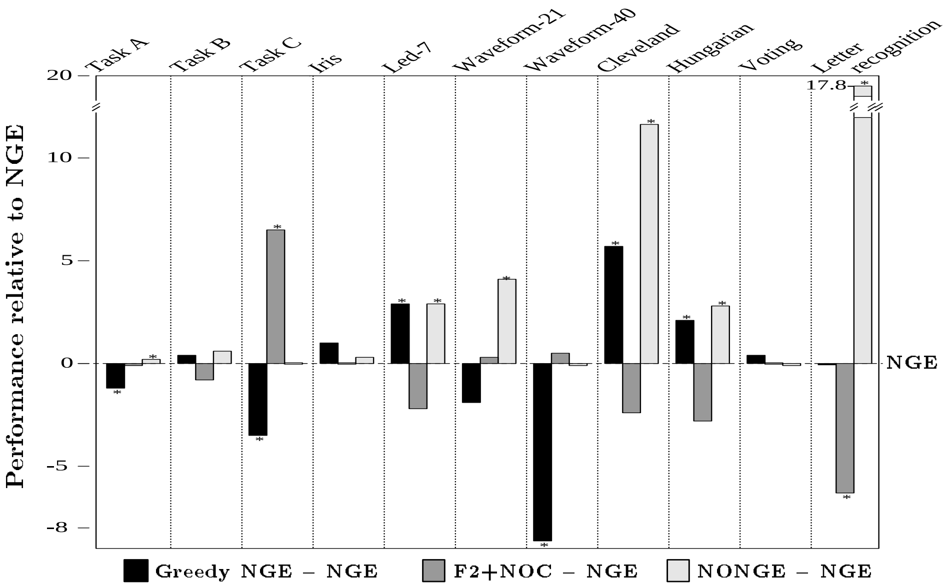


Figure 8. Performance of Greedy NGE, NGE with an additional matching heuristic (F2+NOC: first two matches and the nearest exemplar from the examples own class are considered), and NONGE relative to NGE. A * indicates that the performance difference between NGE and its modification is statistically significant ($p < 0.05$). See Table *A2* in appendix for detailed numbers.

5.2. NGE without Overlapping Hyperrectangles (NONGE)

To test H2, we want to construct a variant of NGE that avoids overlapping rectangles. This can be accomplished as follows. Let us define P to be the potential new hyperrectangle that is constructed by the calls to "generalize" in lines 9 and 10 of Figure 1. Rectangle P is the rectangle formed by extending either the first match or the second match rectangle so that it covers the training example.

In No-Overlap NGE (NONGE), we construct P and then check whether it would intersect with any hyperrectangle from any other class. If P would intersect another rectangle, then we reject P and create a new, single-point rectangle instead. However, if P would be completely contained within another hyperrectangle, we accept P. This way, nested rectangles are permitted, but overlapping (non-nesting) rectangles are forbidden.

In Figure 8, we see that NONGE is significantly better than NGE in 6 of the 11 domains, and it is never significantly worse than NGE. This strongly supports hypothesis H2—that overlapping rectangles cause problems for NGE.

5.3. *A better merge heuristic for NGE?*

NGE stores a training example as a new hyperrectangle whenever the two nearest hyperrectangles have different output classes than the example. In some cases, however, this can create unnecessary new rectangles. Consider Figure 9. Here, rectangle C is further away from point P than either rectangle A or rectangle B. However, because rectangle C has the same class as point P, it could be extended to cover point P without overlapping either of rectangles A or B. By extending rectangle C in this way, we avoid creating a new generalized exemplar for point P.

We developed a modified version of NGE, called F2+NOC, that detects this situation. If the first two matches (to the nearest and second-nearest hyperrectangles) fail, F2+NOC finds the nearest hyperrectangle having the same class as the example. It then extends that nearest hyperrectangle to include the new example if the expanded hyperrectangle would not cover any hyperrectangles from any other classes. Otherwise it stores the example as a new hyperrectangle. This gives NGE another chance to generalize and should in general reduce the amount of memory required by NGE.

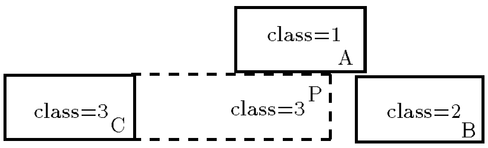


Figure 9. Example showing that rectangle C can be extended to cover point P.

F2+NOC can be considered as a weak test of hypothesis H3 (that the search algorithm of NGE needs improvement). Table *A2* indicates that this additional matching heuristic indeed achieves a reduction in storage in most domains. Hence, it is a better implementation of the NGE bias. However, as shown in Figure 8, F2+NOC performs significantly better than NGE only in Task C, while the reduction in storage is directly related to a loss in predictive accuracy in five domains.

Hence, this improvement to NGE's search algorithm does not explain the poor performance of NGE relative to kNN.

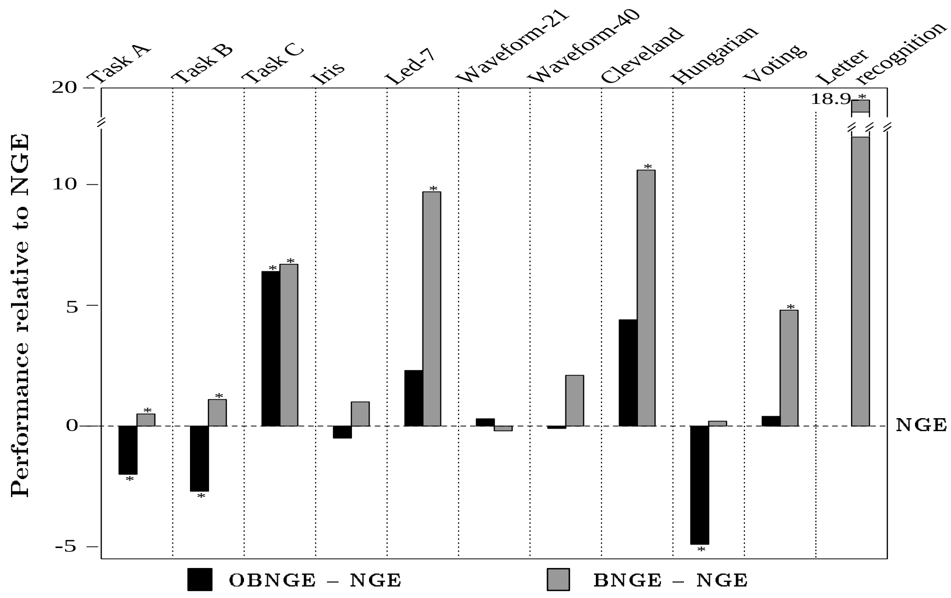


Figure 10. Performance of OBNGE and BNGE relative to NGE. Shown are percentage point differences between OBNGE and NGE and between BNGE and NGE. A * indicates that the performance difference between NGE and its modification is statistically significant ($p < 0.05$). See Table *A2* in appendix for detailed numbers.

5.4. Batch NGE

To obtain a better test of H3, we constructed two batch algorithms (OBNGE and BNGE) for the NGE bias. These algorithms begin with all training examples in memory as point hyperrectangles and progressively merge them to form generalized hyperrectangles. At each step, the two hyperrectangles nearest to one another are merged subject to one of the following constraints:

OBNGE Only merge if that merge would not cause misclassification of any training examples. This algorithm requires testing of the entire training set for each potential merge. It permits overlapping but no nesting of rectangles. We call it OBNGE (Overlapping Batch NGE).

BNGE Only merge if the new hyperrectangle does not cover (or overlap with) any hyperrectangles from any other classes. This algorithm requires intersection of each potential merge with all hyperrectangles from all other classes. It does not permit overlapping or nesting. We call it BNGE (Batch NGE).

The merging process in both algorithms is repeated until no more merges can be found. Note that these algorithms are somewhat dependent on the order in which potential merges are considered. They are greedy in that a merge is accepted as soon as the above mentioned conditions are satisfied.

These algorithms are more conservative in generalizing beyond the training data than the original NGE algorithm, since they generate hyperrectangles only in those parts of the input space which clearly belong to a certain class. Furthermore, due to the fact that BNGE and OBNGE repeatedly pass over the training data, they may also significantly reduce the number of hyperrectangles that remain at the end. BNGE is also faster and easier to use than NGE, since no cross-validation of free parameters is required. OBNGE, however, is not feasible for large training sets.

Numbers displayed in Figure 10 and Table *A2* show that BNGE significantly outperforms NGE in 7 of the 11 domains tested. In all cases, the performance of BNGE is better than NGE. On the other hand, OBNGE is significantly better than NGE only in Task C, and it is significantly worse than NGE in three domains (Task A, Task B, and Hungarian).

This provides additional strong evidence that overlapping rectangles are an inappropriate bias for these domains (H2).

To test H3, we can examine first whether BNGE implements a better search algorithm. For Tasks A and C, BNGE attains the optimal solution (4 hyperrectangles in Task A, 10 in Task C). Furthermore, BNGE also uses only one hyperrectangle to cover the Iris Setosa class in the Iris domain. This is good evidence for the quality of the BNGE search procedure.

The incremental version of NGE most similar to BNGE is NONGE (NGE without overlapping rectangles). By comparing Figures 8, 10, and Table *A2*, it can be seen that BNGE out-performs NONGE in four domains, while NONGE out-performs BNGE in two domains. This gives only weak evidence that the improved search algorithm of BNGE is responsible for the improved performance. Note that BNGE can also be trained incrementally at the cost of storing all training examples. Incremental 'BNGE' would split and re-build hyperrectangles whenever they cover a new example from a different class.

5.5. *Discussion*

From these experiments, we can see that there is weak support for H3 and strong support for H2 as explanations for the poor performance of NGE relative to the nearest neighbor algorithm. There is no support for H1. Versions of NGE that do not permit overlapping rectangles perform consistently better than NGE in all domains tested. The batch algorithm, BNGE, that does not permit nested or overlapping rectangles of different classes, performs quite well and avoids the need to choose the number of seeds for NGE by cross-validation.

6. Feature Weights

In all of the experiments we have conducted thus far, we have treated all features as equally important in computing the Euclidean distance to the nearest hyperrectangles (and nearest neighbors). However, many of the 11 domains involve noisy

Table 3. Percentage of (non-seed) examples covered by at least one hyperrectangle when NGE was initialized with 3 (25) seeds.

Domain	training		testing	
	3 seeds	25 seeds	3 seeds	25 seeds
Iris	64	29	87	57
Led-7	79	81	93	91
Waveform-21	43	38	72	68
Waveform-40	23	20	51	45
Cleveland	80	62	97	91
Hungarian	83	52	98	81
Voting	97	93	100	100
Letter recognition	79	79	95	94

or completely random features. A way to improve performance of both NGE and kNN is to introduce some mechanism for learning which features are important and ignoring unimportant (or noisy) features in all distance computations.

Salzberg (1991 [Section 3.3, last paragraph]) describes a method for online learning of feature weights in NGE. Assume that a new example E is misclassified by an exemplar H. For each input feature f_i, if E_{f_i} matches H_{f_i}, the weight of f_i (w_{f_i}) is increased by multiplying it by $(1 + \Delta_f)$; if E_{f_i} does not match H_{f_i}, the weight w_{f_i} is decreased by multiplying it by $(1 - \Delta_f)$. Δ_f is the global feature-adjustment rate (usually set to 0.2). If E is classified correctly by H, then the feature weights are adjusted in the opposite direction.

There are two problems with this heuristic. First, consider tasks in which one class is much more frequent than another. In such tasks, new examples will tend to be classified correctly by chance, and the feature weights will change exponentially: features that always match will have weights of zero, and features that are random will receive infinite weight. Salzberg (personal communication & 1991 [pseudo-code]) suggested adjusting feature weights only after both matches (i.e., to the nearest and second-nearest hyperrectangles) failed. We found empirically that with this policy, feature weights were adjusted quite infrequently. For example, in the Iris task, feature weights were adjusted for only 1% of the training examples. In Waveform-40, feature weights were adjusted for 7% of the training examples.

The second, more serious problem with the use of feature weights in NGE is that a high percentage of the test cases fall inside at least one hyperrectangle, which means that the distance to the nearest hyperrectangle is zero, and feature weights have no effect on the distance calculation. Table 3 shows the percentage of test and training cases in which this occurs. This suggests that there are limits to the performance improvement that can be obtained by using feature weights with nested hyperrectangles.

In experiments with Salzberg's method for computing feature weights, we found that performance was almost always decreased (see below). We therefore considered another procedure for computing feature weights that has given promising results in other exemplar-based learning methods (Bakiri, 1991).

6.1. Determining Weights by Mutual Information

The purpose of a feature weight mechanism is to give low weight to features that provide no information for classification (e.g., very noisy or irrelevant features) and to give high weight to features that provide reliable information. Hence, a natural quantity to consider is the mutual information between the values of a feature and the class of the examples. If a feature provides no information about the class, the mutual information will be 0. If a feature completely determines the class, the mutual information will be proportional to the log of the number of classes. Let

- $P(C = c)$ be the probability that the class of any training example equals c.
- $P(f_j \in Q(i))$ be the probability that the value of feature j of any example falls into the interval $Q(i) = [\frac{i-1}{nIntervals}, \frac{i}{nIntervals}]$ (i = 1,..,$nIntervals$).
- $P(C = c \ \wedge \ f_j \in Q(i))$ be the joint probability of these two events.

Then the mutual information between feature f_j and the classification C is

$$I(f_j; C) = \sum_{i=1}^{nIntervals} \sum_{c=1}^{nClasses} P(C = c \ \wedge \ f_j \in Q(i)) \times log \frac{P(C = c \ \wedge \ f_j \in Q(i))}{P(C = c) \times P(f_j \in Q(i))}.$$

For discrete features, $nIntervals$ is equal to the number of possible distinct inputs. For continuous features, $nIntervals$ was chosen to be 5. The probabilities were then estimated from the training data; missing values were ignored. The mutual information measure is also known in the machine learning literature as the "information gain" used as a splitting criterion in ID3 and C4.5 (Quinlan, 1992).

6.2. Experiments with Feature Weights

Figure 11 (and Table *A2*) shows the effect of including feature weights and compares the two different procedures for computing the weights. Salzberg's method gives a statistically significant increase in performance in the Hungarian domain, a statistically significant decrease in Task C, and has no significant effect in any of the other domains. The mutual information feature weights generally give slight, though statistically insignificant, improvements in domains without irrelevant features, while the improvements can be substantial in domains with irrelevant features. They give a statistically significant improvement in the Cleveland, Hungarian, Voting ($p < 0.01$), and Waveform-40 domains as well as in Task C. A small ($p < 0.05$) decrease in performance is observed for mutual information feature weights in the Letter recognition domain. Mutual information feature weights had a similar positive effect on the performance of simple nearest neighbor and, to a lesser extent, kNN (see Table A2). The mutual information weights were very small for irrelevant inputs in all domains. Furthermore, feature weights did not differ substantially from one random partition of the data sets to another. In contrast, the weights computed

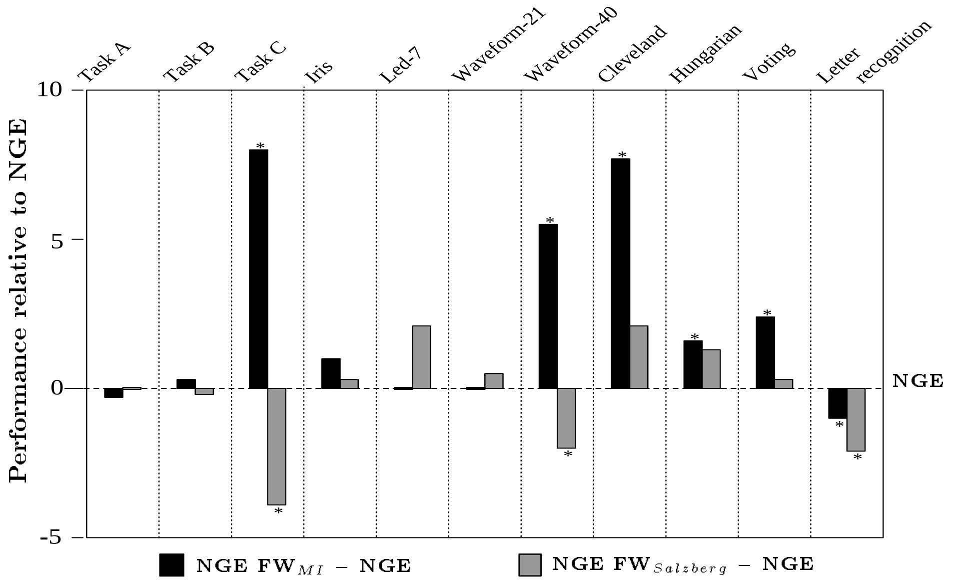


Figure 11. Performance of NGE FW$_{MI}$ and NGE FW$_{Salzberg}$ relative to NGE without feature weights. Shown are percentage point differences between NGE FW$_{MI}$ and NGE, and between NGE FW$_{Salzberg}$ and NGE. A * indicates significance of difference between NGE and its modifications. See Table *A2* in appendix for detailed numbers.

by Salzberg's method differed substantially from one partition to another (varying by as much as a factor of 1000). Within a given training set/test set partition, the features were more or less equally weighted.

From the experiments in Section 6, we conclude that Salzberg's weight procedure has no significant impact on NGE's behavior in most domains and that the mutual information weight procedure performs well in domains that have a large number of irrelevant features. Furthermore, since the mutual information weight procedure is independent of the algorithm it is used for, it is a procedure that could be used effectively by many inductive learning algorithms to filter out irrelevant features.

7. Comparison of the Best Variants of NGE and kNN

We have now developed several modifications to NGE that uniformly improve its performance. The algorithm that best combines these is batch NGE with mutual information feature weights (BNGE FW$_{MI}$). The best corresponding version of the nearest neighbor algorithm is k nearest neighbors (with cross-validation to determine k) and mutual information feature weights (kNN FW$_{MI}$). In this section, we compare these two algorithms to determine whether the modifications to NGE make it competitive with kNN.

Figure 12 shows the results of this comparison. The main conclusion to draw is that BNGE FW$_{MI}$ is significantly inferior to kNN FW$_{MI}$ in 7 domains and significantly superior in only 2. The two domains are both domains where we know

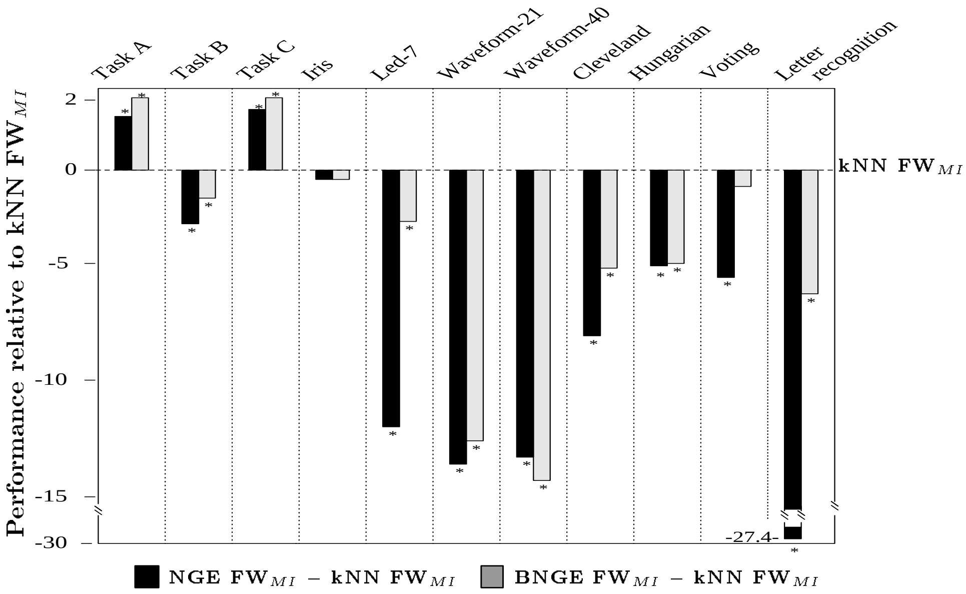


Figure 12. Performance of NGE FW$_{MI}$ and BNGE FW$_{MI}$ relative to kNN FW$_{MI}$. Shown are percentage point differences between NGE FW$_{MI}$ and BNGE FW$_{MI}$ and kNN FW$_{MI}$. These differences (*) were statistically significant. See Table *A2* in appendix for detailed numbers.

that the axis-parallel rectangle bias is appropriate. This shows that when such hyperrectangles are appropriate, BNGE FW$_{MI}$ is able to exploit them. However, in domains where such rectangles are evidently not appropriate, BNGE FW$_{MI}$'s performance suffers, while kNN FW$_{MI}$ is robust under these situations. This shows that further research is still needed to develop an NGE algorithm that is robust in such situations.

8. Related Work

Simpson (Simpson, 1992) introduced an incremental algorithm which is extremely similar to NGE called Fuzzy Min-Max Neural Networks. The main differences between NGE and a Fuzzy Min-Max Classifier (FMMC) are: (a) Hyperrectangles in FMMC are bounded in size, (b) FMMC always extends the nearest hyperrectangle of the same class as the given example to include the example as long the size of the new hyperrectangle is not larger than a user-defined value, and (c) FFMC shrinks hyperrectangles to eliminate overlap of hyperrectangles from different classes.

Carpenter et al. (1992) introduced a neural network architecture based on fuzzy logic and adaptive resonance theory (ART) neural networks. The category boxes used by fuzzy ARTMAP with complement coding are comparable to hyperrectangles. Hyperrectangles in fuzzy ARTMAP with complement coding grow monotonically during learning; their maximum size is bounded by a vigilance parameter. See Carpenter et al. (1992) for a short comparison of NGE, FFMC, and Fuzzy ARTMAP. Neither FMMC nor fuzzy ARTMAP use feature weights in the same sense as discussed in this paper.

9. Summary and Discussion

An extensive study of the NGE algorithm has been conducted. The basic algorithm and a number of modifications were evaluated in eleven domains. NGE was found to be quite sensitive to the number of starting seeds and to the order of presentation of the examples.

The performance of NGE was compared to the performance of the k-Nearest Neighbor algorithm and found to be substantially worse in several domains, even when cross-validation was applied to optimize the number of starting seeds in NGE.

Three hypotheses were introduced to explain this difference in performance: (a) nested rectangles provide a poor bias, (b) overlapping rectangles provide a poor bias, and (c) the incremental search algorithm of NGE needs improvement. Experimental modifications of NGE were made in order to test these hypotheses. Two versions of NGE that avoid nested rectangles (but permit overlapping rectangles) did not perform substantially better than NGE itself. However, an algorithm, called NONGE, that permits nested rectangles but avoids overlapping rectangles performed uniformly better than NGE in all eleven domains (the improvement was statistically significant in 6 of the domains). A batch algorithm, BNGE, that implements a better search algorithm and does not allow nested or overlapping rectangles also performs uniformly better than NGE. It performs better than NONGE in 4 domains and worse in 2.

From these experiments, we conclude that overlapping rectangles are the primary source of difficulty for NGE and that BNGE was the best variant of NGE that we studied. Further experiments reported in Wettschereck (1994) show that BNGE commits most of its errors outside of all hyperrectangles. By using kNN to classify all test examples that fall outside any hyperrectangle, a hybrid method of BNGE and kNN attains classification accuracy comparable to kNN alone – but with a large improvement in classification speed.

All versions of NGE were effective at compressing the data when compared to kNN.

We also studied whether the NGE algorithms could be improved by incorporating feature weights into the distance metric computed by the algorithms. The feature weight mechanism introduced by Salzberg (1991) was shown never to provide a significant improvement over the performance of NGE without feature weights. Indeed, it was significantly worse than simple NGE in three of the domains. On the other hand, a feature weight mechanism based on computing the mutual information between each feature and the output class was shown to be significantly better than NGE in five domains and significantly worse in only one. This mechanism is independent of NGE and can therefore be used as a pre-processing step for any inductive learning algorithm.

10. Conclusions

The data presented strongly support the conclusion that the NGE algorithm as described by Salzberg (1991) should be modified in a number of ways. First, construction of overlapping hyperrectangles should be avoided. Second, if the entire training set is available at once and can be stored in memory, then the classifier should be trained in batch mode to eliminate computationally expensive cross-validation on the number of initial seeds. Third, mutual information should be used to compute feature weights prior to running NGE.

With these modifications, NGE gives superior performance in domains where the axis-parallel hyperrectangle bias is appropriate. However, in other domains, NGE does not perform as well as kNN. Hence, if generalization performance and robustness are critical, kNN is the algorithm of choice. If, on the other hand, understandability and memory compression are important, then NGE (as modified) can be recommended as a fast, easy-to-use inductive learning algorithm.[8]

Acknowledgements

We thank Steven Salzberg for providing assistance during the implementation of NGE. We thank one of the anonymous reviewers for pointing out the heuristic for computing the size of symbolic features to us. We also thank Steven Salzberg, the anonymous reviewers, and Bill Langford for helpful comments on earlier drafts of this note. 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

Aha, D.W., (1990). *A Study of Instance-Based Algorithms for Supervised Learning Tasks.* Technical Report, University of California, Irvine.

Bakiri, G., (1991). *Converting English Text to Speech: A Machine Learning Approach.* Ph.D. Thesis, Oregon State University, Corvallis, Oregon.

Carpenter, G.A., Grossberg, S., Markuzon, N., Reynolds, J.H., & Rosen, D.B., (1992). Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps. *IEEE Transactions on Neural Networks*, 3, 698–713.

Dasarathy, B.V., (1991). *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press.

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, K., Sandhu, S., Guppy, K., Lee, S. & Froelicher, V., (1989). Rapid searches for complex patterns in biological molecules. *American Journal of Cardiology*, 64, 304–310.

Duda, R.O., & Hart, P.E., (1973). *Pattern Classification and Scene Analysis.* New York: Wiley.

Holte, R.C., (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, 11, 63–90.

Murphy, P.M. & Aha, D.W., (1994). *UCI Repository of machine learning databases [Machine-readable data repository].* Technical Report, University of California, Irvine.

Quinlan, J.R. (1992). *C4.5: Programs for Empirical Learning.* San Mateo, CA: Morgan Kaufmann Publishers, INC.

Salzberg, S., (1991). A Nearest Hyperrectangle Learning Method. *Machine Learning*, 6, 277–309.

Simpson, P.K. (1992). Fuzzy min-max neural networks: 1. Classification. *IEEE Transactions on Neural Networks*, 3, 776–786.

Weiss, S.M., & Kulikowski, C.A., (1991). *Computer Systems that learn.* San Mateo CA: Morgan Kaufmann Publishers, INC.

Wettschereck, D. (1994). A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm. *Proceedings of the 7th European Conference on Machine Learning.* In press.

Appendix

Table A1. Percent accuracy (± standard error) of nearest neighbor (NN) and NGE when trained on 350 examples and tested on 150 examples (25 replications). Numbers in parentheses show: number of seeds (first column, cv: leave-one-out cross-validation), average number of hyperrectangles constructed in all other columns.

	Task A		Task B		Task C	
Method	Accuracy	Num Rect.	Accuracy	Num Rect.	Accuracy	Num Rect.
Nge (3)	86.8±4.0	(4.4±0.2)	75.6±1.7	(3.5±0.1)	87.4±0.5	(47.4±1.1)
Nge (5)	97.2±1.4	(7.0±0.3)	85.1±1.3	(6.1±0.2)	86.7±0.5	(48.8±1.0)
Nge (7)	97.8±0.7	(8.5±0.3)	86.3±1.3	(8.1±0.3)	86.2±0.5	(49.9±1.0)
Nge (10)	98.8±0.3	(11.6±0.3)	89.8±0.7	(11.1±0.3)	86.1±0.5	(49.8±1.0)
Nge (15)	98.0±0.4	(16.7±0.3)	92.4±0.5	(16.5±0.3)	85.6±0.5	(53.4±1.3)
Nge (20)	97.9±0.3	(21.8±0.3)	93.2±0.5	(21.6±0.3)	86.1±0.5	(57.0±1.1)
Nge (25)	97.8±0.3	(27.4±0.3)	93.7±0.6	(26.7±0.3)	85.3±0.5	(59.8±1.1)
Nge (cv)	99.3±0.2	(10.8±1.1)	94.1±0.5	(22.6±1.0)	86.9±0.5	(49.6±1.3)
NN	97.6±0.3		97.0±0.2		82.4±0.6	

Notes

1. The focus of this paper is classification accuracy. Issues such as incremental versus batch learning and training and classification speed are only touched on.
2. The size of trivial point hyperrectangles is assumed to be 0.
3. Leave-one-out cross-validation is computationally very expensive for NGE since even the smartest implementation would have to process approximately $\frac{n(n-1)}{2}$ examples for each cross-validation run.
4. The results for the Iris domain differ slightly from those reported by Salzberg, because he employed leave-one-out cross-validation rather than repeated train/test partitions. The results with leave-one-out are 95.3% for nearest neighbor and 92.8% for NGE.
5. One could improve on this by allocating space for the lower and upper corner of each hyperrectangle only if the hyperrectangle is non-trivial.
6. Leave-one-out cross-validation over more than, say, 10 different numbers of seeds is not computationally feasible.
7. There are many other differences between NGE and C4.5. However, Aha's results indicate that a rectangular bias may be of no hindrance given the proper search algorithm.
8. Source code for NGE and some of its modifications is available on request from the first author.

Table A2. Percent accuracy (± standard error) on test set. Shown is mean performance over 25 repetitions, standard error (S.E.), significance of difference to NN ($\star$), NGE$_{cv}$ ($*$), NN FW$_{MI}$ ($\circ$), and NGE$_{cv}$ FW$_{MI}$ ($\bullet$) as well as amount of memory (in percent of training data, if less than 100%) required by NGE (M column).

Method	Domain TaskA		TaskB		TaskC		Iris	
	Perf.±S.E.	M	Perf.±S.E.	M	Perf.±S.E.	M	Perf.±S.E.	M
NN	97.6 ± 0.3		97.0 ± 0.2		82.4 ± 0.6		94.8 ± 0.5	
kNN$_{cv}$	$96.9\pm0.3^{\star\star\star}_{*****}$		$96.4\pm0.3^{\star\star\star}_{*****}$		$82.1\pm0.6_{*****}$		$95.4\pm0.5_{**}$	
NGE$_{cv}$	$99.3\pm0.2^{\star\star\star\star\star}$	6	$94.1\pm0.5^{\star\star\star\star\star}$	13	$86.9\pm0.5^{\star\star\star\star\star}$	28	93.7 ± 0.6	31
NGE$_{3seeds}$	$86.8\pm4.0^{\star\star}_{****}$	3	$75.6\pm1.7^{\star\star\star\star\star}_{*****}$	2	$87.4\pm0.5^{\star\star\star\star\star}$	27	$92.0\pm0.6^{\star\star\star\star\star}_{*}$	9
NGE$_{limit}$	$98.8\pm0.3^{\star\star\star\star}_{**}$	7	$96.8\pm0.2_{*****}$		$87.4\pm0.5^{\star\star\star\star\star}$	27	94.4 ± 0.6	50
Greedy NGE$_{cv}$	$98.1\pm0.2^{\star}_{*****}$	15	$94.5\pm0.4^{\star\star\star\star\star}$	23	$83.4\pm0.6_{*****}$	60	94.7 ± 0.5	31
F2+NOC$_{cv}$	$99.2\pm0.2^{\star\star\star\star\star}$	6	$93.3\pm0.6^{\star\star\star\star\star}$	12	$93.4\pm0.3^{\star\star\star\star\star}_{*****}$	6	$93.7\pm0.7^{\star}$	33
NONGE$_{cv}$	$99.5\pm0.2^{\star\star\star\star\star}_{**}$	5	$94.7\pm0.4^{\star\star\star\star\star}$	18	$86.9\pm0.5^{\star\star\star\star\star}$	28	94.0 ± 0.6	33
OBNGE	$97.3\pm0.5_{*****}$	2	$91.4\pm0.5^{\star\star\star\star\star}_{*****}$	12	$93.3\pm0.2^{\star\star\star\star\star}_{*****}$	7	$93.2\pm0.6^{\star\star}$	8
BNGE	$99.8\pm0.1^{\star\star\star\star\star}_{****}$	2	$95.2\pm0.4^{\star\star\star\star\star}_{***}$	12	$93.6\pm0.5^{\star\star\star\star\star}_{*****}$	3	94.7 ± 0.5	12
NGE$_{cv}$ FW$_{MI}$	$99.0\pm0.2^{\circ\circ\circ\circ\circ}$	5	$94.4\pm0.5^{\circ\circ\circ\circ\circ}$	13	$94.9\pm0.3^{\circ\circ\circ\circ\circ}_{*****}$	10	$94.7\pm0.5_{*}$	34
NGE$_{cv}$ FW$_{S}$	$99.3\pm0.2^{\circ\circ\circ\circ\circ}$	6	$93.9\pm0.5^{\circ\circ\circ\circ\circ}$	13	$83.0\pm0.7^{\circ\circ\circ\circ\circ}_{*****}$	32	$94.0\pm0.6^{\circ\circ}$	35
BNGE FW$_{MI}$	$99.8\pm0.1^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$	2	$95.5\pm0.4^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet}$	12	$95.4\pm0.2^{\circ\circ\circ\circ\circ}$	6	94.7 ± 0.6	11
NN FW$_{MI}$	$97.2\pm0.2^{\star\star}_{\bullet\bullet\bullet\bullet\bullet}$		$97.0\pm0.2_{\bullet\bullet\bullet\bullet\bullet}$		$92.0\pm0.4^{\star\star\star\star\star}_{\bullet\bullet\bullet\bullet\bullet}$		$95.9\pm0.4^{\star\star\star\star\star}_{\bullet\bullet}$	
kNN$_{cv}$ FW$_{MI}$	$96.7\pm0.3^{\circ}_{\bullet\bullet\bullet\bullet\bullet}$		$96.7\pm0.3_{\bullet\bullet\bullet\bullet\bullet}$		$92.3\pm0.4_{\bullet\bullet\bullet\bullet\bullet}$		$95.1\pm0.5^{\circ\circ}$	
	Led-7		Waveform-21		Waveform-40		Cleveland	
NN	71.0 ± 0.5		73.8 ± 0.7		69.5 ± 1.0		76.4 ± 0.9	
kNN$_{cv}$	$73.5\pm0.4^{\star\star\star\star\star}_{*****}$		$82.9\pm0.8^{\star\star\star\star\star}_{*****}$		$80.2\pm1.0^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$		$82.8\pm0.7^{\star\star\star\star\star}_{*****}$	
NGE$_{cv}$	$59.8\pm1.0^{\star\star\star\star\star}$	67	$70.0\pm0.9^{\star\star\star\star}$	17	$64.6\pm1.2^{\star\star\star\star\star}$	24	$66.9\pm1.8^{\star\star\star\star\star}$	29
NGE$_{3seeds}$	$56.0\pm1.2^{\star\star\star\star\star}_{**}$	61	$69.3\pm0.8^{\star\star\star\star\star}$	9	$64.2\pm1.2^{\star\star\star\star}$	14	$55.0\pm1.2^{\star\star\star\star\star}_{*****}$	5
NGE$_{limit}$	$64.2\pm0.7^{\star\star\star\star\star}_{*****}$		$71.5\pm0.9^{\star}_{*}$	56	$64.9\pm1.2^{\star\star\star\star}$	14	$76.3\pm1.2_{*****}$	
Greedy NGE$_{cv}$	$62.7\pm0.9^{\star\star\star\star\star}_{****}$	87	$68.1\pm1.6^{\star\star\star\star}$	65	$56.0\pm1.1^{\star\star\star\star\star}_{*****}$	86	$72.6\pm1.3^{\star\star\star}_{**}$	63
F2+NOC$_{cv}$	$57.6\pm1.1^{\star\star\star\star\star}$	50	$70.3\pm0.9^{\star\star\star\star}$	10	$65.1\pm1.1^{\star\star\star\star}$	15	$64.5\pm1.5^{\star\star\star\star\star}$	23
NONGE$_{cv}$	$62.7\pm0.9^{\star\star\star\star\star}_{**}$	71	$74.1\pm1.0_{****}$	77	$64.5\pm1.4^{\star\star\star\star}$	23	$78.5\pm0.8^{\star\star}_{*****}$	54
OBNGE	$62.1\pm0.9^{\star\star\star\star\star}$	84	$70.3\pm0.9^{\star\star\star\star}$	6	$64.5\pm1.2^{\star\star\star\star\star}$	3	$71.3\pm1.2^{\star\star\star\star\star}_{*}$	59
BNGE	$69.5\pm0.5^{\star\star\star\star\star}_{*****}$	73	$69.8\pm1.5^{\star\star\star\star}$		$66.7\pm1.4^{\star}$	91	$77.5\pm1.1_{*****}$	71
NGE$_{cv}$ FW$_{MI}$	$59.8\pm1.1^{\circ\circ\circ\circ\circ}$	64	$70.0\pm1.1^{\circ\circ\circ\circ\circ}$	13	$70.1\pm1.0^{\circ\circ\circ\circ\circ}_{*****}$	13	$74.6\pm1.4^{\circ}_{****}$	34
NGE$_{cv}$ FW$_{S}$	$61.9\pm1.2^{\circ\circ\circ\circ\circ}$	66	$70.5\pm0.8^{\circ\circ\circ\circ\circ}$	16	$62.6\pm1.2^{\circ\circ\circ\circ\circ}_{**}$	20	$69.0\pm1.3^{\circ\circ\circ\circ\circ}$	33
BNGE FW$_{MI}$	$68.6\pm0.4_{\bullet\bullet\bullet\bullet\bullet}$	74	$71.0\pm1.1^{\circ\circ\circ\circ}_{\bullet\bullet}$	97	$69.2\pm1.2^{\circ\circ\circ\circ\circ}$	90	$77.5\pm0.9_{\bullet}$	50
NN FW$_{MI}$	$68.6\pm0.4^{\star\star\star\star\star}_{\bullet\bullet\bullet\bullet\bullet}$		$76.3\pm0.7^{\star\star\star\star\star}_{\bullet\bullet\bullet\bullet\bullet}$		$78.3\pm0.8^{\star\star\star\star\star}_{\bullet\bullet\bullet\bullet\bullet}$		$77.9\pm1.0^{\star\star}_{\bullet\bullet\bullet\bullet\bullet}$	
kNN$_{cv}$ FW$_{MI}$	$70.8\pm0.5^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$		$82.6\pm0.9^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$		$82.4\pm0.6^{\star\star\star\star\star}_{*****}$		$81.7\pm0.6^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$	
	Hungarian		Voting		Letter recog.			
NN	76.5 ± 0.6		88.3 ± 0.8		95.8 ± 0.1			
kNN$_{cv}$	$83.8\pm0.8^{\star\star\star\star\star}_{*****}$		$94.0\pm0.4^{\star\star\star\star\star}_{*****}$		$95.8\pm0.1_{*****}$			
NGE$_{cv}$	76.5 ± 0.9	30	88.4 ± 1.2	16	$70.2\pm0.4^{\star\star\star\star\star}$	41		
NGE$_{3seeds}$	$61.3\pm2.4^{\star\star\star\star\star}_{*****}$	7	$64.5\pm3.2^{\star\star\star\star\star}_{*****}$	13	$68.6\pm0.4^{\star\star\star\star\star}_{***}$	43		
NGE$_{limit}$	$79.3\pm0.8^{\star\star\star\star\star}$		$84.8\pm1.1^{\star\star\star\star}_{****}$	22	$87.4\pm0.2^{\star\star\star\star\star}_{*****}$			
Greedy NGE$_{cv}$	$78.6\pm0.9^{\star\star\star\star}_{****}$	49	88.8 ± 1.2	34	$70.1\pm0.4^{\star\star\star\star\star}$	63		
F2+NOC$_{cv}$	$73.7\pm1.1^{\star}_{*}$	21	88.4 ± 1.2	14	$63.9\pm0.4^{\star\star\star\star\star}_{*****}$	24		
NONGE$_{cv}$	$79.3\pm0.7^{\star\star\star\star}_{****}$	43	88.3 ± 1.6	22	$88.0\pm0.2^{\star\star\star\star\star}_{*****}$	39		
OBNGE	$71.6\pm1.4^{\star\star\star}_{***}$	40	88.8 ± 2.4	25				
BNGE	76.7 ± 1.0	36	$93.2\pm0.5^{\star\star\star\star\star}_{*****}$	47	$89.1\pm0.1^{\star\star\star\star\star}_{*****}$	39	$\star\star\star\star\star$ p < 0.001	
NGE$_{cv}$ FW$_{MI}$	$78.1\pm0.6_{**}$	28	$90.8\pm0.8^{\circ\circ}_{***}$	15	$69.2\pm0.4^{\circ\circ\circ\circ\circ}_{**}$	35	$\star\star\star\star$ p < 0.005	
NGE$_{cv}$ FW$_{S}$	77.8 ± 1.0	30	88.7 ± 1.1	16	$68.1\pm0.5^{\circ\circ\circ\circ\circ}_{*****}$	45	$\star\star\star$ p < 0.01	
BNGE FW$_{MI}$	78.2 ± 0.8	31	$94.7\pm0.4^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$	30	$91.3\pm0.1^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$	32	$\star\star$ p < 0.05	
NN FW$_{MI}$	$78.9\pm0.6^{\star\star\star\star\star}$		$89.0\pm0.8^{\star\star}$		$96.6\pm0.0^{\star\star\star\star\star}_{\bullet\bullet\bullet\bullet\bullet}$		$\star$ p < 0.1	
kNN$_{cv}$ FW$_{MI}$	$82.2\pm0.9^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$		$95.4\pm0.4^{\circ\circ\circ\circ\circ}_{\bullet\bullet\bullet\bullet\bullet}$		$96.6\pm0.0_{\bullet\bullet\bullet\bullet\bullet}$			