Machine Learning, , 1–32 ()

Reduction Techniques for Instance-Based Learning Algorithms

D. RANDALL WILSON, TONY R. MARTINEZ randy@axon.cs.byu.edu, martinez@cs.byu.edu
Neural Network & Machine Learning Laboratory
World-Wide Web: http://axon.cs.byu.edu
Computer Science Department
Brigham Young University
Provo, Utah 84602, USA

Editor: Robert Holte

Abstract.

Instance-based learning algorithms are often faced with the problem of deciding which instances to store for use during generalization. Storing too many instances can result in large memory requirements and slow execution speed, and can cause an oversensitivity to noise. This paper has two main purposes. First, it provides a survey of existing algorithms used to reduce storage requirements in instance-based learning algorithms and other exemplar-based algorithms. Second, it proposes six additional reduction algorithms called DROP1–DROP5 and DEL (three of which were first described in Wilson & Martinez, 1997c, as RT1–RT3) that can be used to prune instances from the concept description. These algorithms and 10 algorithms from the survey are compared on 31 classification tasks. Of those algorithms that provide substantial storage reduction, the DROP algorithms have the highest average generalization accuracy in these experiments, especially in the presence of noise.

Keywords: Instance-based learning, nearest neighbor, pruning, classification.

1. Introduction

In supervised learning, a machine learning algorithm is shown a *training set*, T, which is a collection of training examples called *instances*. Each instance has an input vector and an output value. After learning from the training set, the learning algorithm is presented with additional input vectors, and the algorithm must *generalize*, i.e., it must use some inductive *bias* (Mitchell, 1980; Schaffer, 1994; Dietterich, 1989; Wolpert, 1993) to decide what the output value should be even if the new input vector was not in the training set.

A large number of machine learning algorithms compute a distance between the input vector and stored *exemplars* when generalizing. Exemplars can be *instances* from the original training set, or can be in other forms such as hyperrectangles, prototypes, or rules. Many such *exemplar-based* learning algorithms exist, and they are often faced with the problem of deciding how many exemplars to store, and what portion of the input space they should cover.

Instance-based learning (IBL) algorithms (Aha, Kibler & Albert, 1991; Aha, 1992) are a subset of exemplar-based learning algorithms that use original instances from the training set as exemplars. One of the most straightforward instance-based learning algorithms is the *nearest neighbor* algorithm (Cover & Hart, 1967; Hart, 1968; Dasarathy, 1991). During generalization, instance-based learning algorithms use a distance function to determine how close a new input vector $\vec{y}$ is to each stored instance, and use the nearest instance or instances to predict the output class of $\vec{y}$ (i.e., to *classify* $\vec{y}$).

Other exemplar-based machine learning paradigms include *memory-based reasoning* (Stanfill & Waltz, 1986), *exemplar-based generalization* (Salzberg, 1991; Wettschereck & Dietterich, 1995), and *case-based reasoning* (CBR) (Watson & Marir, 1994). Such algorithms have had much success on a wide variety of domains. There are also several exemplar-based neural network learning algorithms, including *probabilistic neural networks* (PNN) (Specht, 1992; Wilson & Martinez, 1996, 1997b) and other *radial basis function* networks (Broomhead & Lowe, 1988; Renals & Rohwer, 1989; Wasserman, 1993), as well as *counterpropagation networks* (Hecht-Nielsen, 1987), ART (Carpenter & Grossberg, 1987), and *competitive learning* (Rumelhart & McClelland, 1986).

Exemplar-based learning algorithms must often decide what exemplars to store for use during generalization in order to avoid excessive storage and time complexity, and possibly to improve generalization accuracy by avoiding noise and overfitting.

For example, the basic nearest neighbor algorithm retains all of the training instances. It learns very quickly because it need only read in the training set without much further processing, and it generalizes accurately for many applications. However, since the basic nearest neighbor algorithm stores all of the training instances, it has relatively large memory requirements. It must search through all available instances to classify a new input vector, so it is slow during classification. Also, since it stores every instance in the training set, noisy instances (i.e., those with errors in the input vector or output class, or those not representative of typical cases) are stored as well, which can degrade generalization accuracy.

Techniques such as *k-d trees* (Sproull, 1991; Wess, Althoff & Richter, 1993) and *projection* (Papadimitriou & Bentley, 1980) can reduce the time required to find the nearest neighbor(s) of an input vector, but they do not reduce storage requirements, nor do they address the problem of noise. In addition, they often become much less effective as the dimensionality of the problem (i.e., the number of input attributes) grows (Sproull, 1991).

On the other hand, when some of the instances are removed from the training set, the storage requirements and time necessary for generalization are correspondingly reduced. This paper focuses on the problem of reducing the size of the stored set of instances (or other exemplars) while trying to maintain or even improve generalization accuracy. It accomplishes this by first providing a relatively thorough survey of machine learning algorithms used to reduce the number of instances needed by learning algorithms, and then by proposing several new reduction techniques.

Section 2 discusses several issues related to the problem of instance set reduction, and provides a framework for the discussion of individual reduction algorithms. Section 3 surveys much of the work done in this area. Section 4 presents a collection of six additional algorithms called DROP1–DROP5 and DEL that are used to reduce the size of the training set while maintaining or even improving generalization accuracy. Section 5 presents empirical results comparing 10 of the surveyed techniques with the six additional techniques presented in Section 4 on 31 datasets. Section 6 provides conclusions and future research directions.

2. Issues in Instance Set Reduction

This section provides a framework for the discussion of the instance reduction algorithms presented in later sections. The issues discussed in this section include exemplar representation, the order of the search, the choice of distance function, the general intuition of which instances to keep, and how to evaluate the different reduction strategies.

2.1. Representation

One choice in designing a training set reduction algorithm is to decide whether to retain a subset of the original instances or to modify the instances using a new representation. For example, some algorithms (Salzberg, 1991; Wettschereck & Dietterich, 1995) use hyperrectangles to represent collections of instances; instances can be generalized into rules (Domingos, 1995, 1996); and prototypes can be used to represent a cluster of instances (Chang, 1974), even if no original instance occurred at the point where the prototype is located.

On the other hand, many algorithms (i.e., *instance-based* algorithms) seek to retain a subset of the original instances. One problem with using the original data points is that there may not be any data points located at the precise points that would make for the most accurate and concise concept description. Prototypes, on the other hand, can be artificially constructed to exist exactly where they are needed, if such locations can be accurately determined. Similarly, rules and hyperrectangles can be constructed to reduce the need for instances in certain areas of the input space.

2.2. Direction of Search

When searching for a subset S of instances to keep from training set T, there are also a variety of directions the search can proceed, including *incremental*, *decremental*, and *batch*.

2.2.1. Incremental. The incremental search begins with an empty subset S, and adds each instance in T to S if it fulfills some criteria. In this case the order of presentation of instances can be very important. In particular, the first few instances may have a very different probability of being included in S than they would if they were visited later.

Under such schemes, the order of presentation of instances in T to the algorithm is typically random because by definition, an incremental algorithm should be able to handle new instances as they are made available without all of them being present at the beginning. In addition, some incremental algorithms do not retain all of the previously seen instances even during the learning phase, which can also make the order of presentation important.

One advantage to an incremental scheme is that if instances are made available later, after training is complete, they can continue to be added to S according to the same criteria. Another advantage of incremental algorithms is that they can be faster and use less storage during learning than non-incremental algorithms, since they can ignore some of the discarded instances when adding others. Thus instead of $O(n^2)$ time and $O(n)$ storage during the learning phase, they can use $O(ns)$ time and $O(s)$ storage, where n is the number of training instances and s is the number of instances retained in the subset.

The main disadvantage is that incremental algorithms are sensitive to the order of presentation of the instances, and their early decisions are based on very little information, and are therefore prone to errors until more information is available. Some incremental algorithms (e.g., EACH, Salzberg, 1991) use a small number of instances (e.g., 100) in an initial "batch" phase to help alleviate these problems.

Some algorithms add instances to S in a somewhat incremental fashion, but they examine all available instances to help select which instance to add next. This makes the algorithm not truly incremental, but may improve its performance substantially.

2.2.2. Decremental. The decremental search begins with $S = T$, and then searches for instances to remove from S. Again the order of presentation is important, but unlike the incremental process, all of the training examples are available for examination at any time, so a search can be made to determine which instance would be best to remove during each step of the algorithm. Decremental algorithms discussed in Section 3 include RNN (Gates, 1972), SNN (Ritter, Woodruff, Lowry & Isenhour, 1975), ENN (Wilson, 1972), VSM (Lowe, 1995), and the SHRINK (SUBTRACTIVE) Algorithm (Kibler & Aha, 1987). RISE (Domingos, 1995) can also be viewed as a decremental algorithm, except that instead of simply removing instances from S, instances are generalized into rules. Similarly, Chang's prototype rule (Chang, 1974) operates in a decremental order, but prototypes are merged into each other instead of being simply removed.

One disadvantage with the decremental rule is that it is often computationally more expensive than incremental algorithms. For example, in order to find the nearest neighbor in T of an instance, n distance calculations must be made. On the other hand, there are fewer than n instances in S (zero initially, and some fraction of T eventually), so finding the nearest neighbor in S of an instance takes less computation.

However, if the application of a decremental algorithm can result in greater storage reduction, then the extra computation during learning (which is done just once) can be well worth the computational savings during execution thereafter. Increased generalization accuracy, if it can be achieved, is also typically worth some extra time during learning.

2.2.3. Batch. Another way to apply a training set reduction rule is in batch mode. This involves deciding if each instance meets the removal criteria before removing any of them. Then all those that do meet the criteria are removed at once. For example, the ALL K-NN rule (Tomek, 1976) operates this way. This can relieve the algorithm from having to constantly update lists of nearest neighbors and other information when instances are individually removed.

However, there are also dangers in batch processing. For example, assume the following rule is applied to an instance set:

Remove an instance if it has the same output class as its k nearest neighbors.

This could result in entire clusters disappearing if there are no instances of a different class nearby. If done in decremental mode, however, some instances would remain, because eventually enough neighbors would be removed that one of the k nearest neighbors of an instance would have to be of another class, even if it was originally surrounded by those of its own class.

As with decremental algorithms, batch processing suffers from increased time complexity over incremental algorithms.

2.3. Border points vs. central points

Another factor that distinguishes instance reduction techniques is whether they seek to retain border points, central points, or some other set of points.

The intuition behind retaining border points is that "internal" points do not affect the decision boundaries as much as border points, and thus can be removed with relatively little effect on classification.

On the other hand, some algorithms instead seek to *remove* border points. They remove points that are noisy or do not agree with their neighbors. This removes close border points, leaving smoother decision boundaries behind. However, such algorithms do not remove internal points that do not necessarily contribute to the decision boundary.

It may take a large number of border points to completely define a border, so some algorithms retain *center* points in order to use those instances which are most typical of a particular class to classify instances near them. This can dramatically affect decision boundaries, because the decision boundaries depend on not only where the instances of one class lie, but where those of other classes lie as well. Roughly speaking (i.e., assuming $k = 1$), the decision boundary lies halfway between two nearest instances of opposing classes, so center points must be chosen carefully in order to keep the decision boundaries in the correct general vicinity.

2.4. Distance Function

The distance function (or its complement, the similarity function) is used to decide which neighbors are closest to an input vector and can have a dramatic effect on an instance-based learning system.

The nearest neighbor algorithm and its derivatives usually use variants of the Euclidean distance function, which is defined as:

$$E(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \tag{1}$$

where $\vec{x}$ and $\vec{y}$ are the two input vectors, m is the number of input attributes, and x_i and y_i are the input values for input attribute i. This function is appropriate when all the input attributes are numeric and have ranges of approximately equal width. When the attributes have substantially different ranges, the attributes can be normalized by dividing the individual attribute distances by the range or standard deviation of the attribute.

A variety of other distance functions are also available for continuously-valued attributes, including the Minkowsky (Batchelor, 1978), Mahalanobis (Nadler & Smith, 1993), Camberra, Chebychev, Quadratic, Correlation, and Chi-square distance metrics (Michalski, Stepp & Diday, 1981; Diday, 1974); the Context-Similarity measure (Biberman, 1994); the Contrast Model (Tversky, 1977); hyperrectangle distance functions (Salzberg, 1991; Domingos, 1995) and others. Several of these functions are defined in Figure 1 (Wilson & Martinez, 1997a).

When *nominal* (discrete, unordered) attributes are included in an application, a distance metric is needed that supports them. Some learning algorithms have used the *overlap* metric, which defines the distance for an attribute as 0 if the values are equal, or 1 if they are different, regardless of which two values they are.

An alternative distance function for nominal attributes is the *Value Difference Metric* (VDM) (Stanfill & Waltz, 1986). Using the VDM, the distance between two values x and y of a single attribute a is given as:

$$vdm_a(x, y) = \sum_{a=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2 \tag{2}$$

where $N_{a,x}$ is the number of times attribute a had value x; $N_{a,x,c}$ is the number of times attribute a had value x and the output class was c; and C is the number of output classes. Using this distance measure, two values are considered to be closer if they have more similar classifications, regardless of the order of the values.

In order to handle heterogeneous applications—those with both numeric and nominal attributes—it is possible to use a heterogeneous distance function such as HVDM (Wilson & Martinez, 1997a), which is defined as:

$$HVDM(\vec{x}, \vec{y}) = \sqrt{\sum_{a=1}^{m} d_a^2(x_a, y_a)} \tag{3}$$

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x}-\boldsymbol{y})^T Q (\boldsymbol{x}-\boldsymbol{y}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x}-\boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x}-\boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_x} - \frac{y_i}{size_y} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \operatorname{sign}(x_i - x_j) \operatorname{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions. ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

where the function $d_a(x, y)$ is the distance for attribute a and is defined as:

$$d_a(x,y) = \begin{cases} 1, & \text{if } x \text{ or } y \text{ is unknown; otherwise...} \\ vdm_a(x,y), & \text{if } a \text{ is nominal, else} \\ \frac{|x-y|}{4\sigma_a}. & \text{if } a \text{ is numeric.} \end{cases} \tag{4}$$

where $vdm_a(x, y)$ is the function given in (2), and σ_a is the standard deviation of the values occurring for attribute a in the instances in the training set T. This distance function provides appropriate normalization between numeric and nominal attributes, as well as between numeric attributes of different scales. It handles unknown input values by assigning them a large distance so that instances with missing attributes will be less likely to be used as neighbors than those with all attributes specified. Using a constant for the distance to an unknown attribute value also serves to effectively ignore such attributes when an instance to be classified is missing a value, since the distance to that attribute will be the same for all instances in the system.

2.4.1. Weighting. Several algorithms use weighting schemes that alter the distance measurements and voting influence of each instance. In this paper we focus on training set reduction, and thus will not use any weighting schemes in our experiments other than those needed for normalization in the distance function, as explained above. A good survey of weighting schemes is given by Wettschereck, Aha and Mohri (1997).

2.5. Voting

Another decision that must be made for many algorithms is the choice of k, which is the number of neighbors used to decide the output class of an input vector. The value of k is typically a small, odd integer (e.g., 1, 3 or 5). Usually each such nearest neighbor gets exactly one vote, so even values of k could result in "ties" that would have to be resolved arbitrarily or through some more complicated scheme. There are some algorithms which give closer neighbors more influence than further ones, such as the *Distance-Weighted kNN Rule* (Dudani, 1976). Such modifications reduce the sensitivity of the algorithm to the selection of k. *Radial Basis Function* networks (Wasserman, 1993) and *Probabilistic Neural Networks* (Specht, 1992) use a Gaussian weighting of influence and allow all instances to "vote", though instances that are very far from the input have only negligible influence. This does away with the need for the k parameter, but introduces a need for weight-spreading parameters.

One common way of determining the value of k is to use *leave-one-out cross-validation*. For each of several values of k, each instance is classified by its k nearest neighbors other than the instance itself, to see if it is classified correctly. The value for k that produces the highest accuracy is chosen.

In the basic nearest neighbor rule, setting k greater than 1 decreases the sensitivity of the algorithm to noise, and tends to smooth the decision boundaries somewhat. It is also important for many pruning algorithms to have a $k > 1$. However, once pruning has taken place, it is possible that the value of k should be changed. For example, if the training set has been reduced to the point that there is only one instance representing what was formerly a cluster of instances, then perhaps $k = 1$ would be more appropriate than $k > 1$, especially if the noisy instances have been removed during the reduction process. In other cases, the value of k should remain the same. Thus, it may be appropriate to find a value of k for use during the pruning process, and then redetermine the best value for k after pruning is completed.

It may even be advantageous to update k dynamically during the pruning process. For example, if a very large value of k were used initially, the order of removal of instances from the subset might be improved.

2.6. Evaluation Strategies

In comparing training set reduction algorithms, there are a number of criteria that can be used to compare the relative strengths and weaknesses of each algorithm. These include speed increase (during execution), storage reduction, noise tolerance, generalization accuracy, time requirements (during learning), and incrementality.

2.6.1. Storage reduction. One of the main goals of training set reduction algorithms is to reduce storage requirements. It is important to note that if alternate representations are used (e.g., hyperrectangles or rules), any increase in the size of the new representation must be taken into account along with the reduction in number of instances stored.

2.6.2. Speed increase. Another main goal is to speed up classification. A reduction in the number of stored instances will typically yield a corresponding reduction in the time it takes to search through these instances and classify a new input vector. Again, more complex representations such as hyperrectangles may not need as many comparisons, but may require more computation for each comparison, and this must be taken into account.

2.6.3. Generalization accuracy. A successful algorithm will often be able to significantly reduce the size of the training set without significantly reducing generalization accuracy. In some cases generalization accuracy can increase with the reduction of instances, such as when noisy instances are removed and when decision boundaries are smoothed to more closely match the true underlying function rather than the sampling distribution.

2.6.4. Noise tolerance. Algorithms also differ with respect to how well they work in the presence of noise. In the presence of noise, there are two main problems that can occur. The first is that very few instances will be removed from the training set because many instances are needed to maintain the noisy (and thus overly complex) decision boundaries. The second problem is that generalization accuracy can suffer, especially if noisy instances are retained while good instances are removed. In such cases the reduced training set can be much less accurate than the full training set in classifying new input vectors.

2.6.5. Learning speed. The learning process is done just once on a training set, so it is not quite as important for the learning phase to be fast. However, if the learning phase takes too long it can become impractical for real applications. Ironically, it is on especially large training sets that reduction algorithms are most badly needed, so a reasonable (e.g., $O(n^2)$ or faster) time bound is desirable.

2.6.6. Incremental. In some cases it is convenient to have an incremental algorithm so that additional instances can be added over time as they become available. On the other hand, it is possible to use a non-incremental algorithm on an initial database and then employ a separate incremental algorithm once a reasonable starting point has been achieved.

Note that not all algorithms attempt to meet all of these goals. For example, a hybrid hyperrectangle and nearest-neighbor algorithm by Wettschereck (1994) saves all of the training set in addition to the hyperrectangles, and thus actually increases storage requirements. However, it uses the hyperrectangles to quickly classify most input vectors, and only uses the entire training set when necessary. Thus, it sacrifices the goal of storage reduction in favor of the goals of classification speed and maintaining or increasing generalization accuracy.

3. Survey of Instance Reduction Algorithms

Many researchers have addressed the problem of training set size reduction. This section surveys several techniques, discusses them in light of the framework presented in Section 2, and points out their interesting differences. This survey builds upon an earlier survey done by Dasarathy (1991). Most of the algorithms discussed here use a subset S of the original instances in the training set T as their representation, and though most have primarily used the Euclidean distance function in the past, they can typically make use of the HVDM distance function or other distance functions when needed. Most of the algorithms also tend to use $k = 1$ except where noted, though in most cases the algorithms can be modified to use $k > 1$.

3.1. Nearest Neighbor Editing Rules

3.1.1. Condensed Nearest Neighbor Rule. Hart (1968) made one of the first attempts to reduce the size of the training set with his *Condensed Nearest Neighbor Rule* (CNN). His algorithm finds a subset S of the training set T such that every member of T is closer to a member of S of the same class than to a member of S of a different class. In this way, the subset S can be used to classify all the instances in T correctly (assuming that T is consistent, i.e., that no two instances in T have identical inputs but different classes).

This algorithm begins by randomly selecting one instance belonging to each output class from T and putting them in S. Then each instance in T is classified using only the instances in S. If an instance is misclassified, it is added to S, thus ensuring that it will be classified correctly. This process is repeated until there are no instances in T that are misclassified. This algorithm ensures that all instances in T are classified correctly, though it does not guarantee a minimal set.

This algorithm is especially sensitive to noise, because noisy instances will usually be misclassified by their neighbors, and thus will be retained. This causes two problems. First, storage reduction is hindered, because noisy instances are retained, and because they are there, often non-noisy instances nearby will also need to be retained. The second problem is that generalization accuracy is hurt because noisy instances are usually exceptions and thus do not represent the underlying function well. Since some neighbors have probably been pruned, a noisy instance in S will often cover more of the input space than it did in T, thus causing even more misclassifications than before reduction.

3.1.2. Selective Nearest Neighbor Rule. Ritter *et al.* (1975) extended the condensed NN method in their *Selective Nearest Neighbor Rule* (SNN) such that every member of T must be closer to a member of S of the same class than to any member of T (instead of S) of a different class. Further, the method ensures a minimal subset satisfying these conditions.

The algorithm for SNN is more complex than most other reduction algorithms, and the learning time is significantly greater, due to the manipulation of an $n \times n$ matrix and occasional recursion. The SNN algorithm begins by constructing a binary $n \times n$ matrix A (where n is the number of instances in T), where A_{ij} is set to 1 when instance j is of the same class as instance i, and it is closer to instance i than i's nearest *enemy*, i.e., the nearest neighbor of i in T that is of a different class than i. A_{ii} is always set to 1.

Once this array is set up, the following 5 steps are taken until no columns remain in the array:

1. For all columns i that have exactly one bit on, let j be the row with the bit on in column i. All columns with a bit on in row j are removed, row j is removed, and instance j is added to S.
2. For all rows j, delete row j if for all (remaining) columns i and for some (remaining) row k, $A_{ji} \leq A_{ki}$. In other words, row j is deleted if for some other row k, whenever row j contains a 1, row k also contains a 1. In this case instance j is *not* added to S.
3. Delete any column i if for all (remaining) rows j and some (remaining) column k, $A_{ji} \geq A_{jk}$. In other words, column i is deleted if there is some other column k that has zeroes in every row that column i does (and possibly zeroes in other rows as well). Again instance i is *not* added to S.
4. Continue to repeat steps 1–3 until no further progress can be made. If no columns remain in the array, then S is complete and the algorithm is finished. Otherwise, go on to step 5.
5. Find the row j that when included in S requires the fewest other rows to also be included in S. This is done as follows:
   (a) For each remaining row j, assume that instance j will be added to S, and that row j and any (remaining) columns with a bit on in row j will be deleted (but do not actually remove row j or the columns yet). Subject to this assumption, find the fewest number of additional rows it would take to get at least as many 1's as there are remaining columns. From the minimums found for each row j, keep track of the absolute minimum found by any row j.
   (b) For each row j in (a) that resulted in the absolute minimum number of additional rows that *might* be needed, actually remove j and columns with

bits on in row j and call the algorithm recursively beginning with step 1. If the minimum number of rows was really used, then add j to S and stop: S is complete. Otherwise, restore row j and the removed columns, and try the next possible row j.

(c) If no row j is successful in achieving the minimum number, increment the absolute minimum and try (b) again until successful.

Note that the only steps in which instances are chosen for inclusion in S are steps 1 and 5.

This algorithm takes approximately $O(mn^2 + n^3)$ time, compared to the $O(mn^2)$ or less time required by most other algorithms surveyed. It also requires $O(n^2)$ storage during the learning phase for the matrix, though this matrix is discarded after learning is complete. The algorithm is sensitive to noise, though it will tend to sacrifice storage more than accuracy when noise is present. For an example of how this algorithm works, the reader is referred to Ritter *et al.* (1975), which also appears in Dasarathy (1991). An implementation in C is also available in the on-line appendix to this paper.

3.1.3. Reduced Nearest Neighbor Rule. Gates (1972) introduced the *Reduced Nearest Neighbor Rule* (RNN). The RNN algorithm starts with $S = T$ and removes each instance from S if such a removal does not cause any *other* instances in T to be misclassified by the instances remaining in S. It is computationally more expensive than Hart's Condensed NN rule, but will always produce a subset of CNN, and is thus less expensive in terms of computation and storage during the classification stage.

Since the instance being removed is not guaranteed to be classified correctly, this algorithm is able to remove noisy instances and internal instances while retaining border points.

3.1.4. Edited Nearest Neighbor Rule. Wilson (1972) developed the *Edited Nearest Neighbor* (ENN) algorithm in which S starts out the same as T, and then each instance in S is removed if it does not agree with the majority of its k nearest neighbors (with $k = 3$, typically). This edits out noisy instances as well as close border cases, leaving smoother decision boundaries. It also retains all internal points, which keeps it from reducing the storage requirements as much as most other reduction algorithms. The *Repeated ENN* (RENN) applies the ENN algorithm repeatedly until all instances remaining have a majority of their neighbors with the same class, which continues to widen the gap between classes and smoothes the decision boundary.

3.1.5. All k-NN. Tomek (1976) extended the ENN with his *All k-NN* method of editing. This algorithm works as follows: for $i = 1$ to k, flag as bad any instance not classified correctly by its i nearest neighbors. After completing the loop all k times, remove any instances from S flagged as bad. In his experiments, RENN produced higher accuracy than ENN, and the ALL K-NN method resulted in even higher accuracy yet. As with ENN, this method can leave internal points intact, thus limiting the amount of reduction that it can accomplish. These algorithms serve more as noise filters than serious reduction algorithms.

3.1.6. Variable Similarity Metric. Lowe (1995) presented a *Variable Similarity Metric* (VSM) learning system that produces a confidence level of its classifications. In order to reduce storage and remove noisy instances, an instance t is removed if all k of its neighbors are of the same class, even if they are of a different class than t (in which case t is likely to be noisy). This removes noisy instances as well as internal instances, while retaining border instances. The instance is only removed, however, if its neighbors are at least 60% sure of their classification. The VSM system typically uses a fairly large k (e.g., $k = 10$), and the reduction in storage is thus quite conservative, but it can provide an increase in generalization accuracy. Also, the VSM system used distance-weighted voting, which makes a larger value of k more appropriate.

3.2. "Instance-Based" Learning Algorithms

Aha *et al.* (1991; Aha, 1992) presented a series of *instance-based* learning algorithms. IB1 (Instance Based learning algorithm 1) was simply the 1-NN algorithm, and was used as a baseline.

3.2.1. IB2. The *IB2* algorithm is incremental: it starts with S initially empty, and each instance in T is added to S if it is not classified correctly by the instances already in S (with the first instance always added). An early case study (Kibler & Aha, 1987) calls this algorithm the *Growth (Additive) Algorithm.* This algorithm is quite similar to Hart's Condensed NN rule, except that IB2 does not seed S with one instance of each class, and does not repeat the process after the first pass through the training set. This means that IB2 will not necessarily classify all instances in T correctly.

This algorithm retains border points in S while eliminating internal points that are surrounded by members of the same class. Like the CNN algorithm, IB2 is extremely sensitive to noise, because erroneous instances will usually be misclassified, and thus noisy instances will almost always be saved, while more reliable instances are removed.

3.2.2. Shrink (Subtractive) Algorithm. Kibler & Aha (1987) also presented an algorithm that starts with $S = T$, and then removes any instances that would still be classified correctly by the remaining subset. This is similar to the Reduced Nearest Neighbor (RNN) rule, except that it only considers whether the *removed* instance would be classified correctly, whereas RNN considers whether the classification of *other* instances would be affected by the instance's removal. Like RNN and many of the other algorithms, it retains border points, but unlike RNN, this algorithm is sensitive to noise.

3.2.3. IB3. The *IB3* algorithm (Aha *et al.*, 1991; Aha 1992) is another incremental algorithm that addresses IB2's problem of keeping noisy instances by retaining only *acceptable* misclassified instances. The algorithm proceeds as shown below.

1. For each instance t in T
2. Let a be the nearest *acceptable* instance in S to t.
3. (if there are no acceptable instances in S, let a be a random instance in S)
4. If $class(a) \neq class(t)$ then add t to S.
5. For each instance s in S
6. If s is at least as close to t as a is
7. Then update the classification record of s
8. and remove s from S if its classification record is significantly poor.
9. Remove all non-acceptable instance from S.

An instance is *acceptable* if the lower bound on its accuracy is statistically significantly higher (at a 90% confidence level) than the upper bound on the frequency of its class. Similarly, an instance is dropped from S if the upper bound on its accuracy is statistically significantly lower (at a 70% confidence level) than the lower bound on the frequency of its class. Other instances are kept in S during training, and then dropped at the end if they do not prove to be acceptable.

The formula for the upper and lower bounds of the confidence interval is:

$$\frac{p + z^2/2n \pm z\sqrt{\frac{p(1-p)}{n} + \frac{z^2}{4n^2}}}{1 + z^2/n} \tag{5}$$

where for the *accuracy* of an instance in S, n is the number of classification attempts since introduction of the instance to S (i.e., the number of times it was at least as close to t as a was), p is the accuracy of such attempts (i.e., the number of times the instance's class matched t's class, divided by n), and z is the confidence (.9 for acceptance, .7 for dropping). For the frequency of a class, p is the frequency (i.e. proportion of instances so far that are of this class), n is the number of previously processed instances, and z is the confidence (.9 for acceptance, .7 for dropping).

IB3 was able to achieve greater reduction in the number of instances stored and also achieved higher accuracy than IB2, due to its reduced sensitivity to noise on the applications on which it was tested.

3.2.4. IB4 and IB5. In order to handle irrelevant attributes, *IB4* (Aha, 1992) extends IB3 by building a set of attribute weights for each class. It requires fewer instances to generalize well when irrelevant attributes are present in a dataset. *IB5* (Aha, 1992) extends IB4 to handle the addition of new attributes to the problem after training has already begun. These extensions of IB3 address issues that are beyond the scope of this paper, and are thus only briefly mentioned here.

3.2.5. MCS. Brodley (1993) introduced a *Model Class Selection* (MCS) system that uses an instance-based learning algorithm (which claims to be "based loosely on IB3") as part of a larger hybrid learning algorithm. Her algorithm for reducing the size of the training set is to keep track of how many times each instance was one of the k nearest neighbors of another instance (as instances were being added to the concept description), and whether its class matched that of the instance being classified. If the number of times it was wrong is greater than the number of times it was correct then it is thrown out. This tends to avoid noise, though it uses a simpler approach than IB3.

3.2.6. TIBL. Zhang (1992) used a different approach called the *Typical Instance Based Learning* (TIBL) algorithm, which attempted to save instances near the center of clusters rather than on the border. This can result in much more drastic reduction in storage and smoother decision boundaries, and is robust in the presence of noise.

The *typicality* of an instance is defined as the ratio of its average similarity to instances of the same class to its average similarity to instances of other classes. The *similarity* of two instances x and y is defined as $1 - distance(x, y)$, where

$$distance(\vec{x}, \vec{y}) = w_x \sqrt{\frac{1}{m} \sum_{i=1}^{m} \left(\frac{x_i - y_i}{max_i - min_i} \right)^2} \tag{6}$$

and m is the number of input attributes, max_i and min_i are the maximum and minimum values occurring for attribute i, respectively. For nominal attributes, the distance for that attribute is 0 if they are equal or 1 if they are different (i.e., the *overlap* metric). Each instance x has a weight w_x that is used during training and subsequent classification, but which is not used in computing typicality.

The learning algorithm proceeds as follows. Pick the most typical instance x in $T - S$ that is incorrectly classified by the instances in S. Find the most typical instance y in $T - S$ which causes x to be correctly classified, and add it to S. Note that x itself is *not* added at this point. Set y's weight to be $w_y = 1/typicality(y)$. Repeat this process until all instances in T are classified correctly.

This strategy shows great reduction in storage, especially when the application has "graded structures" in which some instances are more typical of a class than others in a fairly continuous way. The TIBL algorithm also avoids saving noisy instances. It is pseudo-incremental, i.e., it proceeds in an incremental fashion, but it uses the entire training set to determine the typicality of each instance and the range of each input attribute.

The TIBL algorithm may have difficulty on problems with complex decision surfaces, and requires modifications to handle disjoint geometric regions that belong to the same class.

3.2.7. Random Mutation Hill Climbing. Skalak (1994) used *random mutation hill climbing* (Papadimitriou & Steiglitz, 1982) to select instances to use in S. The method begins with m randomly-selected instances in S (where m is a parameter that is unfortunately not automatically selected). Then for each iteration (called a *mutation*), one randomly-selected instance in S is removed and replaced with another randomly-selected instance in $T-S$. If this strictly improves classification of the instances in T, the change is retained, otherwise it is undone. This process is repeated for n iterations, where n is another parameter provided by the user. Skalak used $n = 100$.

Since it does not determine the number m of instances to retain in the subset, this method only solves part of the problem.

3.2.8. Encoding Length. Cameron-Jones (1995) used an *encoding length heuristic* to determine how good the subset S is in describing T. The basic algorithm begins with a growing phase that takes each instance i in T and adds it to S if that results in a lower cost than not adding it. As with IB3, the growing phase can be affected by the order of presentation of the instances.

The *cost* (i.e., the value to be minimized) of the instance-based model is

$$COST(m,n,x) = F(m,n) + m\log_2(C) + F(x,n-m) + x\log_2(C-1) \qquad (7)$$

where n is the number of instances in T, m is the number of instances in S, and x is the number of *exceptions* (i.e., the number of instances *seen so far* that are misclassified by the instances in S). $F(m,n)$ is the cost of encoding which m instances of the n available are retained, and is defined as:

$$F(m,n) = \log^*\left(\sum_{j=0}^{m} C_j^n\right) = \log^*\left(\sum_{j=0}^{m} \frac{n!}{j!(n-j)!}\right) \qquad (8)$$

where $\log^*(x)$ is the sum of the positive terms of $\log_2(x)$, $\log_2(\log_2(x))$, etc.

After all instances are seen, pruning is done, where each instance i in S is removed if doing so lowers the cost of the classifier. Cameron-Jones calls this method the "Pre/All" method, since it is not truly incremental, but to better distinguish it from other techniques in this paper, we call it the *Encoding Length Grow* (ELGrow) method.

The *Explore* method (Cameron-Jones, 1995) begins by growing and pruning S using the ELGrow method, and then performs 1000 mutations to try to improve the classifier. Each mutation tries adding an instance to S, removing one from S, or swapping one in S with one in $T-S$, and keeps the change if it does not increase the cost of the classifier. The generalization accuracy of the Explore method is quite good empirically, and its storage reduction is much better than most other algorithms.

3.3. Prototypes and Other Modifications of the Instances

Some algorithms seek to reduce storage requirements and speed up classification by modifying the instances themselves, instead of just deciding which ones to keep.

3.3.1. Prototypes. Chang (1974) introduced an algorithm in which each instance in T is initially treated as a prototype. The nearest two instances that have the same class are merged into a single prototype (using a weighted averaging scheme) that is located somewhere between the two prototypes. This process is repeated until classification accuracy starts to suffer.

This method achieved good results, though it requires modification to handle applications that have one or more nominal input attributes.

3.3.2. RISE. Domingos (1995) introduced the RISE 2.0 system which treats each instance in T as a rule in R. For each rule r in R, the nearest example n in T of the same class as r is found that is not yet covered by r. The rule r is then minimally generalized to cover n, unless that harms accuracy. This process is repeated until no rules are generalized during an entire pass through all the rules in R.

During generalization, the nearest rule to an input vector is used to provide the output class. If two rules are equally close, the one with higher generalization accuracy on the training set is used.

3.3.3. EACH. Salzberg (1991) introduced the *Nested Generalized Exemplar* (NGE) theory, in which hyperrectangles are used to take the place of one or more instances, thus reducing storage requirements. The program used to implement NGE is called the *Exemplar-Aided Constructor of Hyperrectangles* (EACH). EACH seeds the system with several randomly-selected instances from the training set, after which it operates incrementally. As each instance is presented, EACH finds the distance to the nearest exemplar (i.e., a point or hyperrectangle), which is 0 if the instance is inside the hyperrectangle. A point inside multiple hyperrectangles is considered to be closest to the smallest one.

When the new instance has the same class as its nearest exemplar, the exemplar is generalized (i.e., the hyperrectangle is grown) so that it also covers the new instance. When the classes are different, EACH attempts to change the shape of the second-closest exemplar so that it becomes the closest one. If it cannot do so, then the new instance becomes a new exemplar. Weights are maintained for each exemplar that reduce the effect of noisy exemplars and irrelevant attributes.

Wettschereck & Dietterich (1995) introduced a hybrid nearest-neighbor and nearest-hyperrectangle algorithm that uses hyperrectangles to classify input vectors if they fall inside the hyperrectangle, and kNN to classify inputs that were not covered by any hyperrectangle. This algorithm must store the entire training set T, but accelerates classification by using relatively few hyperrectangles whenever possible.

4. Ordered Removal

Given the issues in Section 2 to consider, our research has been directed towards finding instance reduction techniques that provide noise tolerance, high generalization accuracy, insensitivity to the order of presentation of instances, and significant storage reduction, which in turn improves generalization speed.

This section presents a collection of new heuristics used to decide which instances to keep and which instances to remove from a training set. Unlike most previous methods, these algorithms take careful note of the order in which instances are removed. The first three methods, DROP1–DROP3, were previously introduced by the authors under the names RT1–RT3, respectively (Wilson & Martinez, 1997c).

In order to avoid repeating lengthy definitions, some notation is introduced here. A training set T consists of n instances (or prototypes) $P_{1..n}$. Each instance P has k nearest neighbors $P.N_{1..k}$ (ordered from nearest to furthest), where k is typically a small odd integer such as 1, 3 or 5. P also has a nearest *enemy*, $P.E$, which is the nearest instance with a different output class. Those instances that have P as one of their k nearest neighbors are called *associates* of P, and are notated as $P.A_{1..a}$ (sorted from nearest to furthest) where a is the number of associates that P has.

4.1. DROP1

The first new reduction technique we present is the *Decremental Reduction Optimization Procedure 1*, or *DROP1*. This algorithm is identical to RNN (Gates, 1972) with the exception that the accuracy is checked on S instead of T. It is included here mostly as a baseline for comparison with the other DROP algorithms, and to provide a framework on which to build the others.

DROP1 uses the following basic rule to decide if it is safe to remove an instance from the instance set S (where $S = T$ originally):

> Remove P if at least as many of its associates in S would be classified correctly without P.

To see if an instance P can be removed using this rule, each associate (i.e., each instance that has P as one of its neighbors) is checked to see what effect the removal of P would have on it.

Removing P causes each associate $P.A_i$ to use its $k{+}1^{\mathrm{st}}$ nearest neighbor ($P.A_i.N_{k+1}$) in place of P. If P has the same class as $P.A_i$, and $P.A_i.N_{k+1}$ has a different class than $P.A_i$, this weakens its classification, and could cause $P.A_i$ to be misclassified by its neighbors. On the other hand, if P is a different class than $P.A_i$ and $P.A_i.N_{k+1}$ is the same class as $P.A_i$, the removal of P could cause a previously misclassified instance to be classified correctly.

In essence, this rule tests to see if removing P would degrade leave-one-out cross-validation generalization accuracy, which is an estimate of the true generalization ability of the resulting classifier. An instance is removed when it results in the same level of generalization with lower storage requirements. By maintaining lists of $k+1$ neighbors and an average of $k+1$ associates (and their distances), the leave-one-out cross-validation can be computed in $O(k)$ time for each instance instead of the usual $O(mn)$ time, where n is the number of instances in the training set, and m is the number of input attributes. An $O(mn)$ step is only required once an instance is selected for removal. This efficient method is similar to the method used in RISE (Domingos, 1995).

The algorithm for DROP1 proceeds as shown below.

```
DROP1(Training set T): Instance set S.
    Let S = T.
    For each instance P in S:
        Find P.N_{1..k+1}, the k + 1 nearest neighbors of P in S.
        Add P to each of its neighbors' lists of associates.
    For each instance P in S:
        Let with = # of associates of P classified correctly with P as a neighbor.
        Let without = # of associates of P classified correctly without P.
        If (without − with) ≥ 0
            Remove P from S.
            For each associate A of P
                Remove P from A's list of nearest neighbors.
                Find a new nearest neighbor for A.
                Add A to its new neighbor's list of associates.
            For each neighbor N of P
                Remove P from N's lists of associates.
        Endif
    Return S.
```

This algorithm begins by building a list of nearest neighbors for each instance, as well as a list of associates. Then each instance in S is removed if its removal does not hurt the classification of the instances remaining in S. When an instance P is removed, all of its associates must remove P from their list of nearest neighbors, and then must find a new nearest neighbor so that they still have $k+1$ neighbors in their list. When they find a new neighbor N, they also add themselves to N's list of associates so that at all times every instance has a current list of neighbors and associates.

This algorithm removes noisy instances, because a noisy instance P usually has associates that are mostly of a different class, and such associates will be at least as likely to be classified correctly without P. DROP1 also removes instances in the center of clusters, because associates there are not near their enemies, and thus continue to be classified correctly without P.

Near the border, the removal of some instances can cause others to be classified incorrectly because the majority of their neighbors can become enemies. Thus this algorithm tends to keep non-noisy border points. At the limit, there is typically a collection of border instances such that the majority of the k nearest neighbors of each of these instances is the correct class.

4.2. DROP2: Using More Information and Ordering the Removal

There is a potential problem that can arise in DROP1 with regard to noisy instances. A noisy instance will typically have associates of a different class, and will thus be contained to a somewhat small portion of the input space. However, if its associates are removed by the above rule, the noisy instance may cover more and more of the input space. Eventually it is hoped that the noisy instance itself will be removed. However, if many of its neighbors are removed first, its associates may eventually include instances of the same class from the other side of the original decision boundary, and it is possible that removing the noisy instance at that point could cause some of its distant associates to be classified incorrectly.

DROP2 solves this problem by considering the effect of the removal of an instance on all the instances in the *original* training set T instead of considering only those instances remaining in S. In other words, an instance P is removed from S only if at least as many of its associates—including those that may have already been pruned—are classified correctly without it.

Thus, the removal criterion can be restated as:

> Remove P if at least as many of its associates in T would be classified correctly without P.

Using this modification, each instance P in the original training set T continues to maintain a list of its $k+1$ nearest neighbors in S, even after P is removed from S. This in turn means that instances in S have associates that are both in and out of S, while instances that have been removed from S have no associates (because they are no longer a neighbor of any instance). This modification makes use of additional information that is available for estimating generalization accuracy, and also avoids some problems that can occur with DROP1 such as removing entire clusters. This change is made by removing lines 15 and 16 from the pseudo-code for DROP1 in Section 4.1 so that pruned instances will still be associates of their nearest neighbors in S.

DROP2 also changes the order of removal of instances. It initially sorts the instances in S by the distance to their nearest enemy. Instances are then checked for removal beginning at the instance furthest from its nearest enemy. This tends to remove instances furthest from the decision boundary first, which in turn increases the chance of retaining border points.

4.3. DROP3: Filtering Noise

DROP2 sorts S in an attempt to remove center points before border points. One problem with this method is that noisy instances are also "border" points, and cause the order of removal to be drastically changed. One noisy point in the center of a cluster causes many points in that cluster to be considered border points, and some of these can remain in S even after the noisy point is removed.

Two passes through S can remove the dangling center points, but unfortunately, by that time some border points may have already been removed that should have been kept.

DROP3 therefore uses a noise-filtering pass *before* sorting the instances in S. This is done using a rule similar to ENN (Wilson, 1972): Any instance misclassified by its k nearest neighbors is removed. This removes noisy instances, as well as close border points, which can in turn smooth the decision boundary slightly. This helps to avoid "overfitting" the data, i.e., using a decision surface that goes beyond modeling the underlying function and starts to model the data sampling distribution as well.

After removing noisy instances from S in this manner, the instances are sorted by distance to their nearest enemy remaining in S, and thus points far from the real decision boundary are removed first. This allows points internal to clusters to be removed early in the process, even if there were noisy points nearby.

4.4. DROP4: More Carefully Filtering Noise

DROP4 is identical to DROP3 except that instead of blindly applying ENN, the noise-filtering pass removes each instance only if it is (1) misclassified by its k nearest neighbors, *and* (2) it does not hurt the classification of other instances. While DROP3 usually works well, it can in rare cases remove far too many instances in the noise-reduction pass. In one experiment, it went so far as to remove all of them. DROP4 avoids such problems and thus protects against especially poor generalization accuracy in such rare cases, at the expense of slightly higher storage requirements on average.

4.5. DROP5: Smoothing the Decision Boundary

DROP5 modifies DROP2 such that instances are considered for removal beginning with instances that are *nearest* to their nearest enemy, and proceeding outward. This serves as a noise-reduction pass, but will also cause most internal points to be removed as well. By removing points near the decision boundary first, the decision boundary is smoothed. After this pass, the furthest-to-nearest pass as done by DROP2 is done repeatedly until no further improvement can be made.

A modified version of DROP5 was used in the *Reduced Probabilistic Neural Network* (RPNN) (Wilson & Martinez, 1997b), which is a *Radial Basis Function* (RBF) network used for classification. The RPNN used a pruning technique that included a conservative nearest-to-furthest noise-filtering pass followed by a more aggressive furthest-to-nearest node pruning pass.

4.6. Decremental Encoding Length

The *Decremental Encoding Length* (*DEL*) algorithm is the same as DROP3, except that it uses the encoding length heuristic (as is used in ELGrow and Explore in Section 3.2) to decide in each case whether an instance can be removed. DEL starts with $S = T$, and begins with a noise-filtering pass in which each instance is removed if (a) it is misclassified by its k nearest neighbors, and (b) removing the instance does not increase the encoding length cost. The remaining instances are then sorted by the distance to their nearest enemy, and as long as any improvement is being made, the remaining instances are removed (starting with the instance furthest from its nearest enemy) if doing so does not increase the encoding length cost.

5. Experimental Results

Many of the reduction techniques surveyed in Section 3 and all of the techniques proposed in Section 4 were implemented and tested on 31 datasets from the Machine Learning Database Repository at the University of California, Irvine (Merz & Murphy, 1996). Those included in these experiments are CNN, SNN, ENN, RENN, All k-NN, IB2, IB3, ELGrow, Explore, DEL, and DROP1-DROP5.

These experiments were limited to those algorithms that choose a subset S from the training set T to use for subsequent classification. Therefore, the methods that modify the instances themselves were not included, i.e., rule-based, prototype, and hyperrectangle-building methods. Similarly, VSM and MCS were excluded since they are part of more complicated systems. RMHC was excluded because it does not specify how many instances to retain, and its method is subsumed by Explore. Similarly, RNN and Shrink (Subtractive) are improved upon by DROP2 and DROP1, respectively, and are thus not included for the sake of parsimony.

The basic k nearest neighbor (kNN) algorithm that retains 100% of the training set is also included for comparison.

All of the algorithms use $k = 3$, and in our experiments they all use the HVDM distance function. (Experiments were also done using a more traditional Euclidean distance metric with overlap metric for nominal attributes, but the average accuracy for every one of the algorithms was higher using HVDM.)

5.1. *Results*

Ten-fold cross-validation was used for each experiment. Each dataset was divided into 10 partitions and each pruning technique was given a training set T consisting of 9 of the partitions (i.e., 90% of the data), from which it returned a subset S. The remaining partition (i.e., the other 10% of the data) was classified using only the instances in S. Ten such trials were run for each dataset with each pruning algorithm, using a different one of the 10 partitions as the test set for each trial. The average accuracy over the 10 trials is reported for each pruning algorithm on each dataset in Table 1. The average percentage of instances in T that were included in S is also reported for each experiment under the column "%". The average accuracy and storage percentage for each method over all of the 31 datasets is shown in bold near the bottom of Table 1. Due to the size of Table 1, it is broken into three parts, but the overall average of all of the reduction techniques on each dataset and the results for the kNN algorithm are included with each part for comparison.

Table 1(a). Results for CNN, SNN, IB2, IB3 and DEL. The left column shows generalization accuracy and the right column ("%") shows what percent of the original training set was retained by the pruning algorithm.

Database	kNN	%	CNN	%	SNN	%	IB2	%	IB3	%	LED	%	Average	Avg%
Anneal	93.11	100	96.99^{--}	9.57	86.08^{++}	10.57	96.74^{--}	9.48	91.35^{++}	9.79	93.85	9.30	**92.32**	28.12
Australian	84.78	100	77.68^{++}	24.22	81.31^{+}	28.38	78.26^{++}	24.15	85.22^{-}	4.78	84.78	2.56	**82.35**	27.92
Breast Cancer(WI)	96.28	100	95.71	7.09	93.85^{++}	8.35	95.71	7.09	96.57	3.47	96.28	1.89	**94.56**	25.55
Bridges	66.09^{--}	100	61.18	49.48	61.37	52.52	62.18^{-}	48.64	64.73^{--}	28.83	64.27^{--}	35.64	**59.20**	37.74
Crx	83.62^{++}	100	79.42^{++}	24.15	81.59^{++}	27.52	79.42^{++}	24.15	86.09	4.28	83.62^{++}	3.08	**82.91**	27.85
Echocardiogram	94.82	100	85.18^{+}	14.72	48.75^{++}	26.29	85.18^{+}	14.72	72.86^{++}	11.57	93.39	6.91	**89.01**	30.31
Flag	61.34	100	53.63^{++}	50.29	53.63^{++}	51.66	53.11^{++}	49.95	49.47^{++}	34.14	56.18^{+}	45.88	**56.85**	39.50
Glass	73.83^{--}	100	68.14	38.53	64.39	42.63	66.77	39.25	62.14	33.80	69.59^{-}	38.42	**65.30**	38.95
Heart	81.48	100	70.00^{++}	26.17	77.04^{++}	33.78	70.00^{++}	26.17	80.00^{++}	13.58	78.89^{++}	4.73	**78.41**	30.68
Heart(Cleveland)	81.19	100	73.95^{++}	30.84	76.25^{++}	33.88	73.96^{++}	30.29	81.16	11.11	79.49	13.64	**79.00**	31.93
Heart(Hungarian)	79.55	100	70.40^{++}	28.87	75.84^{++}	34.01	73.87^{++}	27.44	79.20	9.90	77.18^{++}	12.28	**78.04**	30.13
Heart(Long Beach VA)	70.00	100	61.00^{++}	35.67	67.00^{++}	43.56	57.00^{++}	35.39	70.00	4.89	70.00	19.28	**70.26**	31.01
Heart(More)	73.78^{++}	100	69.69^{++}	33.21	72.22^{++}	43.64	69.69^{++}	33.21	76.31	9.36	75.15^{++}	16.81	**73.02**	30.99
Heart(Swiss)	92.69	100	91.09	11.38	92.69	15.90	90.26^{+}	11.38	93.46	3.70	92.69	4.25	**92.95**	26.05
Hepatitis	80.62	100	75.50^{++}	25.30	81.92	30.96	74.17^{++}	25.66	73.08	5.09	80.00	7.59	**78.69**	28.88
Horse Colic	57.84^{++}	100	59.90^{++}	35.66	64.47^{++}	48.65	60.24^{++}	35.36	66.75	8.49	67.73	21.82	**60.89**	27.38
Image Segmentation	93.10	100	90.00^{++}	16.61	77.38^{++}	13.02	89.52^{++}	16.93	92.14	16.01	91.90	11.11	**89.71**	30.43
Ionosphere	84.62^{++}	100	82.93^{++}	21.62	81.74^{++}	19.21	82.93^{++}	21.62	85.75	14.59	86.32	12.88	**83.99**	28.73
Iris	94.00	100	90.00^{++}	12.74	83.34^{++}	14.07	90.00^{++}	12.74	94.67	19.78	93.33^{++}	9.56	**92.27**	31.29
LED Creator+17	67.10^{++}	100	55.50^{++}	43.14	59.10^{++}	51.38	55.50^{++}	43.16	60.70^{++}	32.31	66.60^{++}	20.90	**66.21**	34.89
LED Creator	73.40^{-}	100	64.90^{++}	35.79	71.80	92.78	64.60^{++}	35.71	70.40	22.04	72.30	13.92	**70.79**	35.10
Liver (Bupa)	65.57^{-}	100	56.80	40.87	57.70	52.59	56.80	40.87	58.24	10.66	61.38	38.36	**60.33**	37.62
Pima Diabetes	73.56	100	65.76^{++}	36.89	67.97^{++}	42.95	65.76^{++}	36.89	69.78^{++}	10.97	71.61^{++}	12.64	**71.00**	33.04
Promoters	93.45^{-}	100	86.73	13.83	87.09	15.51	84.91	14.36	91.64	18.12	83.09	7.34	**88.64**	31.48
Sonar	87.55^{--}	100	74.12	32.85	79.81	28.26	80.88	33.87	69.38^{++}	12.02	83.29^{--}	29.86	**77.90**	37.66
Soybean (Large)	88.59^{-}	100	83.10	24.97	80.44^{+}	20.27	84.06	24.61	86.63	30.33	87.27	24.76	**84.81**	38.31
Vehicle	71.76^{--}	100	67.50	37.04	67.27^{-}	43.21	67.50	37.04	67.62	28.36	68.10^{-}	32.51	**66.80**	37.67
Voting	95.64	100	93.59^{++}	9.12	95.40	10.21	93.59^{++}	9.12	95.64	5.44	94.27^{+}	2.02	**94.44**	26.57
Vowel	96.57^{--}	100	86.72^{+}	30.05	78.56^{++}	19.97	87.48	29.71	89.57	36.60	93.17^{--}	36.15	**85.57**	47.48
Wine	94.93	100	92.65	14.30	96.05	14.23	92.65	14.30	91.50^{+}	16.60	94.38	9.05	**93.50**	30.91
Zoo	94.44^{--}	100	91.11	12.47	76.67^{++}	10.62	91.11	12.47	92.22	29.38	90.00	18.27	**91.05**	34.69
Average	**82.11**	100	**76.48**	26.69	**75.44**	31.63	**76.58**	26.64	**78.85**	16.13	**80.65**	16.88	**79.06**	32.54
#DROP3 better/worse	14-17	31-0	26-5	25-6	24-7	25-6	25-6	25-6	20-10	20-11	20-10	17-14		
#Sig. better/worse	5-10	31-0	19-1	25-5	20-1	25-4	19-2	25-5	8-2	13-9	9-5	15-12		
Wilcoxon	-78.99	99.5	99.50	99.50	99.50	99.50	99.50	99.50	99.32	93.53	91.97	87.55		

DROP3 seemed to have the best mix of storage reduction and generalization accuracy of the DROP methods, so it was selected for comparisons with all of the other methods. Three tests were used to see how DROP3 compared to the other reduction algorithms on this entire set of classification tasks. The first is a count of how often DROP3 was "better" and "worse" than each of the other algorithms, where "better" means higher accuracy or lower storage requirements in the respective columns. These counts are given as a pair of numbers in the row labeled "*#DROP3 better/worse.*" For example, under CNN, "26–5" in the accuracy column and "25–6" in the storage column indicates that DROP3 had higher average accuracy than CNN on 26 of the 31 datasets, and lower average storage requirements on 25 of them.

Since many differences were not statistically significant, a one-tailed paired t-test was used on the 10-fold cross-validation results for each dataset to measure whether the average accuracy for DROP3 was significantly higher (when its average accuracy was higher) or lower (when its average accuracy was lower) than each of the

Table 1(b). Accuracy and storage percentage for DROP1–DROP5.

Database	kNN	%	DROP1	%	DROP2	%	DROP3	%	DROP4	%	DROP5	%	Average	Avg%
Anneal	93.11	100	87.70^{++}	5.05	95.61^{--}	8.08	94.11	8.65	94.36	11.67	95.24	9.93	**92.32**	28.12
Australian	84.78	100	64.49^{++}	2.30	83.62	7.28	83.91	5.96	84.78	7.99	83.91	9.18	**82.35**	27.92
Breast Cancer(WI)	96.28	100	77.52^{++}	1.14	95.86	3.13	96.14	3.58	96.28	4.05	95.71	4.07	**94.56**	25.55
Bridges	66.09^{--}	100	39.64^{++}	10.17	61.18^{-}	17.30	56.36	17.60	57.36	21.28	62.82^{-}	22.22	**59.20**	37.74
Crx	83.62^{++}	100	65.94^{++}	3.75	84.64	7.31	85.80	5.46	85.51	7.33	83.77^{+}	7.68	**82.91**	27.85
Echocardiogram	94.82	100	93.39	9.61	94.82	10.51	93.39	10.66	94.82	10.96	93.39	9.16	**89.01**	30.31
Flag	61.34	100	43.18^{++}	9.05	62.79	20.62	61.29	20.45	59.58	27.09	58.13	25.26	**56.85**	39.50
Glass	73.83^{--}	100	62.97	15.47	65.04	23.10	65.02	23.88	65.91	29.54	65.45	24.81	**65.30**	38.95
Heart	81.48	100	72.59^{++}	5.47	81.85	12.22	83.33	13.62	81.85	16.71	81.11	16.67	**78.41**	30.68
Heart(Cleveland)	81.19	100	70.91^{++}	6.09	79.55	11.92	80.84	12.76	78.19	15.26	79.84	15.37	**79.00**	31.93
Heart(Hungarian)	79.55	100	72.17^{++}	5.74	78.52	8.80	80.29	9.86	79.22	11.53	79.60	11.15	**78.04**	30.13
Heart(Long Beach VA)	70.00	100	69.00	4.39	70.00^{+}	11.83	73.50	4.50	74.00	11.72	73.00	14.94	**70.26**	31.01
Heart(More)	73.78^{++}	100	63.46^{++}	3.51	73.98^{++}	10.71	76.38	9.14	74.36^{++}	13.19	74.63^{++}	14.62	**73.02**	30.99
Heart(Swiss)	92.69	100	93.46	1.81	93.46	2.53	93.46	1.81	93.46	2.35	92.63	5.42	**92.95**	26.05
Hepatitis	80.62	100	72.38^{++}	4.66	80.75	10.54	81.87	7.81	78.75^{++}	9.75	83.29	9.39	**78.69**	28.88
Horse Colic	57.84^{++}	100	59.15^{++}	1.55	70.74	8.20	70.13	10.30	67.73	20.41	68.45	14.14	**60.89**	27.38
Image Segmentation	93.10	100	81.19^{++}	6.61	92.86	10.45	92.62	10.98	94.05^{-}	12.41	89.29^{++}	11.35	**89.71**	30.43
Ionosphere	84.62^{++}	100	79.77^{++}	3.23	86.60	7.79	87.75	7.06	86.90	10.60	86.90	9.78	**83.99**	28.73
Iris	94.00	100	84.67^{++}	8.59	94.67	14.22	95.33	14.81	95.33	14.89	94.00	12.15	**92.27**	31.29
LED Creator+17	67.10^{++}	100	61.40^{++}	9.94	69.20	12.98	70.40	12.66	69.50^{+}	16.37	69.80	14.96	**66.21**	34.89
LED Creator	73.40^{-}	100	68.30^{++}	10.05	71.80	11.85	71.70	11.93	71.90	13.71	72.00	12.33	**70.79**	35.10
Liver (Bupa)	65.57^{-}	100	58.24	10.92	67.77^{--}	24.77	60.84	24.99	62.60	32.56	65.50^{--}	31.08	**60.33**	37.62
Pima Diabetes	73.56	100	65.23^{++}	6.50	70.44^{++}	17.59	75.01	16.90	72.53^{+}	21.76	73.05^{+}	21.95	**71.00**	33.04
Promoters	93.45^{-}	100	87.00	6.39	84.91	13.63	86.82	16.67	86.82	16.67	87.00	12.58	**88.64**	31.48
Sonar	87.55^{--}	100	64.93^{++}	11.38	80.88^{-}	26.60	78.00	26.87	82.81^{-}	31.20	79.88	29.81	**77.90**	37.66
Soybean (Large)	88.59^{-}	100	77.20^{++}	19.51	86.60^{-}	22.77	84.97	25.26	86.29	28.41	83.73	25.44	**84.81**	38.31
Vehicle	71.76^{--}	100	59.91^{++}	12.07	67.37	21.49	65.85	23.00	67.03	27.88	70.22^{--}	26.71	**66.80**	37.67
Voting	95.64	100	93.11^{+}	2.91	94.50^{++}	4.90	95.87	5.11	95.87	5.36	95.86	7.13	**94.44**	26.57
Vowel	96.57^{--}	100	83.31^{++}	39.16	91.08^{--}	44.66	89.56	45.22	90.70^{--}	46.02	93.36^{--}	42.66	**85.57**	47.48
Wine	94.93	100	90.98^{+}	5.74	93.24^{+}	11.42	94.93	16.11	94.93	16.17	96.08	9.74	**93.50**	30.91
Zoo	94.44^{--}	100	88.89	18.02	88.89	15.80	90.00	20.00	91.11	21.60	95.56^{--}	17.16	**91.05**	34.69
Average	**82.11**	100	**72.65**	8.41	**81.07**	14.03	**81.14**	14.31	**81.11**	17.30	**81.39**	16.09	**79.06**	32.54
#DROP3 better/worse	14-17	31-0	28-1	0-30	17-13	10-21	n/a	0-0	11-15	30-0	18-12	25-6		
#Sig. better/worse	5-10	31-0	24-0	0-29	5-6	8-11	n/a	0-0	4-3	26-0	4-5	22-6		
Wilcoxon	-78.99	99.5	99.50	-99.50	70.23	-91.03	n/a	-50.00	56.25	99.50	62.42	99.50		

other methods. In Table 1, the superscripts "+" and "++" indicate that DROP3's average accuracy was significantly higher than the other method's average accuracy at a 90% and 95% confidence level, respectively. Similarly, "–" and "– –" indicate that DROP3 had significantly lower average accuracy than the other method at a 90% and 95% confidence level, respectively.

The row labeled "*#sig. better/worse*" gives a count of how often DROP3 was significantly "better" and "worse" than each of the other algorithms at a 90% or higher confidence level. A t-test was also done to test the significance of differences in storage requirements for each experiment, and the results are summarized in this same row, though the "+'s" and "–'s" were not included in the storage column due to space constraints.

In a further effort to verify whether differences in accuracy and storage requirements on this entire set of classification tasks were statistically significant, a one-tailed Wilcoxon Signed Ranks test (Conover, 1971; DeGroot, 1986) was used to compare DROP3 with each of the other pruning techniques. The confidence level of a significant difference is shown in the "Wilcoxon" row of Table 1. Positive values indicate the confidence that DROP3 is "better" (i.e., higher accuracy or lower stor-

Table 1(c). Accuracy and storage percentage for ENN, RENN, All k-NN, ELGrow, and Explore.

Database	kNN	%	ENN	%	RENN	%	AllKNN	%	ELGrow	%	Explore	%	Average	Avg%
Anneal	93.11	100	89.73^{++}	91.99	89.48^{++}	90.70	89.98^{++}	92.43	88.35^{++}	0.70	91.11^{+}	0.75	**92.32**	28.12
Australian	84.78	100	84.49	86.49	84.20	84.80	86.09^{--}	78.07	83.62	0.32	85.80^{-}	0.32	**82.35**	27.92
Breast Cancer(WI)	96.28	100	97.00^{-}	96.80	96.86^{-}	96.61	97.00^{-}	94.58	89.86^{++}	0.32	96.71	0.32	**94.56**	25.55
Bridges	66.09^{--}	100	59.46	68.23	58.36^{-}	65.93	59.36	58.48	56.27	5.35	57.18	5.67	**59.20**	37.74
Crx	83.62^{++}	100	85.36	86.17	85.80	85.64	85.07	78.82	85.22	0.32	85.51	0.32	**82.91**	27.85
Echocardiogram	94.82	100	93.39	92.94	93.39	92.94	93.39	92.18	93.39	3.01	94.82	3.01	**89.01**	30.31
Flag	61.34	100	63.32	67.07	62.76	62.83	61.24	55.67	55.50	2.00	56.16	2.06	**56.85**	39.50
Glass	73.83^{--}	100	65.91	70.82	64.00	69.06	67.75	65.89	50.54^{++}	2.28	63.98	3.53	**65.30**	38.95
Heart	81.48	100	81.11^{++}	83.13	81.11^{++}	81.98	81.85	72.02	74.44^{++}	0.82	81.85	0.82	**78.41**	30.68
Heart(Cleveland)	81.19	100	82.49	83.46	82.16	82.51	81.51	72.72	81.52	0.73	82.15	0.73	**79.00**	31.93
Heart(Hungarian)	79.55	100	80.28	82.12	79.25	79.93	80.62	70.79	80.61	0.75	82.30	0.75	**78.04**	30.13
Heart(Long Beach VA)	70.00	100	74.00	75.44	74.00	72.72	74.00	62.28	72.50^{+}	0.84	74.50	1.11	**70.26**	31.01
Heart(More)	73.78^{++}	100	76.31	76.58	76.51	74.41	75.60	66.97	67.48^{++}	0.14	73.13^{++}	0.14	**73.02**	30.99
Heart(Swiss)	92.69	100	93.46	93.49	93.46	93.49	93.46	88.71	93.46	0.90	93.46	0.90	**92.95**	26.05
Hepatitis	80.62	100	81.25	83.73	80.58	82.80	81.33	75.20	76.67^{++}	1.00	78.67^{+}	1.29	**78.69**	28.88
Horse Colic	57.84^{++}	100	45.89^{++}	58.21	32.91^{++}	27.87	45.89^{++}	52.16	67.09	0.37	67.09	0.37	**60.89**	27.38
Image Segmentation	93.10	100	91.90	92.72	91.43	91.77	92.14	91.46	85.95^{++}	2.22	89.76^{++}	2.43	**89.71**	30.43
Ionosphere	84.62^{++}	100	84.04^{++}	84.24	84.04^{++}	82.27	84.05^{++}	82.18	73.77^{++}	0.63	80.89^{++}	0.63	**83.99**	28.73
Iris	94.00	100	95.33	94.74	95.33	94.67	95.33	93.78	88.67^{++}	2.30	92.67	2.30	**92.27**	31.29
LED Creator+17	67.10^{++}	100	71.00^{-}	71.42	70.90	70.00	70.90	58.98	71.20	1.66	72.20^{--}	1.40	**66.21**	34.89
LED Creator	73.40^{-}	100	72.10	73.88	72.00	72.86	71.80	72.07	70.40	1.53	72.10	1.52	**70.79**	35.10
Liver (Bupa)	65.57^{-}	100	61.12	68.15	58.77	63.13	60.24	52.34	56.74^{++}	0.55	57.65	0.64	**60.33**	37.62
Pima Diabetes	73.56	100	75.39	76.37	75.91	74.52	74.88	64.61	67.84^{++}	0.29	75.27	0.29	**71.00**	33.04
Promoters	93.45^{-}	100	93.45^{-}	96.33	93.45^{-}	96.33	93.45^{-}	95.07	88.82	2.10	91.36^{-}	2.10	**88.64**	31.48
Sonar	87.55^{--}	100	81.79	84.35	78.38	81.79	80.36	80.29	70.24^{++}	1.07	70.29^{++}	1.07	**77.90**	37.66
Soybean (Large)	88.59^{-}	100	86.61^{--}	89.90	85.97	87.41	86.62^{-}	88.24	82.70	7.35	85.92	7.78	**84.81**	38.31
Vehicle	71.76^{--}	100	69.52^{--}	73.81	69.05^{--}	69.75	70.21^{--}	64.74	58.15^{++}	2.25	60.76^{++}	2.47	**66.80**	37.67
Voting	95.64	100	95.41	95.84	95.41	95.81	95.41	94.35	88.99^{++}	0.51	94.25^{+}	0.51	**94.44**	26.57
Vowel	96.57^{--}	100	92.40^{--}	96.57	91.27^{-}	95.94	93.54^{--}	96.70	50.20^{++}	4.69	57.77^{++}	6.65	**85.57**	47.48
Wine	94.93	100	94.93	95.57	94.93	95.57	94.93	94.76	81.47^{++}	1.93	95.46	2.12	**93.50**	30.91
Zoo	94.44^{--}	100	91.11	92.96	91.11	92.59	93.33^{--}	94.07	94.44^{--}	7.90	95.56^{--}	8.40	**91.05**	34.69
Average	**82.11**	100	**80.95**	83.34	**80.09**	80.92	**81.01**	77.44	**75.68**	1.83	**79.24**	2.01	79.06	32.54
#DROP3 better/worse	14-17	31-0	10-18	31-0	10-17	31-0	12-16	31-0	24-5	0-31	16-14	0-31		
#Sig. better/worse	5-10	31-0	4-6	31-0	4-5	31-0	3-7	31-0	17-1	0-31	9-4	0-31		
Wilcoxon	-78.99	99.5	-91.85	99.50	-57.88	99.50	-84.48	99.50	99.50	-99.50	95.62	-99.50		

age requirements) than the other method on these datasets, while negative values indicate the confidence that DROP3 is "worse." Confidence values with a magnitude of at least 90% can be considered significant differences, and throughout the remainder of this paper, the word "significant" will be used to refer to statistical significance with at least a 90% confidence level.

The accuracy and storage percentages for each of the 10 trials for each method on each dataset are available in an on-line appendix, along with standard deviations, t-test confidence values, source code and data files (see the Appendix of this paper for details).

5.2. *Analysis of Results*

Several observations can be made from the results in this table. CNN and IB2 both achieve almost identical results (less than 1% difference in both size and accuracy in most cases), due to the similarity of their algorithms. SNN had lower accuracy and higher storage requirements on average when compared to CNN and IB2, and the SNN algorithm is much more complex and substantially slower than the others as well. IB3 was able to achieve higher accuracy and lower storage than SNN, CNN and IB2, with the only disadvantage being a learning algorithm that is somewhat more complex (though not much slower) than CNN or IB2.

DROP3 had significantly higher accuracy than all of these four methods at over a 99% confidence level (according to the Wilcoxon test) on these datasets. It also had significantly lower storage requirements than all of these methods at over a 99% confidence level, except on IB3, where the confidence of lower storage requirements was still over 90%.

As expected, ENN, RENN and All k-NN all retained over 75% of the instances, due to their retention of internal (non-border) instances. They all had fairly good accuracy, largely because they still had access to most of the original instances. In agreement with Tomek (1976), the All k-NN method achieved better reduction and higher accuracy than RENN, which in turn had higher reduction (though slightly lower accuracy) than ENN. All three of these methods had higher average accuracy than DROP3, though only ENN's accuracy was significantly higher, but this is mostly due to retaining most of the instances.

The ELGrow and Explore techniques had by far the best storage reduction of any of the algorithms. The ELGrow algorithm achieved the lowest average reduction (1.67%) but also suffered a significant drop in generalization accuracy when compared to the non-pruned system. However, the Explore method achieved better average accuracy with only a slight increase in storage over ELGrow, indicating that the random mutation hill climbing step was successful in finding a better subset S after the growing and pruning phases were complete. DROP3 had significantly better accuracy than both of these algorithms, but their storage requirements were significantly better than that of DROP3.

The ordered reduction techniques DROP2–DROP5 all had average generalization accuracy that was within 1% of the full kNN classifier. Their average accuracy was higher than any of the other reduction methods. DROP2 and DROP3 both had average storage requirements of about 14%, which is lower than any of the other methods except ELGrow and Explore. DROP1 retained about half as many instances as the other ordered reduction techniques (significantly better than DROP3), but had the worst generalization accuracy of any of them (significantly worse than DROP3), because it fails to use information provided by pruned instances in determining whether further instances should be pruned.

The DEL approach also had good accuracy, but was not quite as high as DROP2–DROP5, and its storage requirements were not quite as low as most of the DROP methods. DROP3 had lower storage and significantly higher accuracy than DEL.

5.3. Effect of Noise

Since several of these algorithms are designed to be robust in the presence of noise, the same experiments were repeated with 10% noise artificially added to each dataset. This was done by randomly changing the output class of 10% of the

Table 2. Average accuracy and storage requirements in the presence of 10% noise.

Algorithm	Clean	Size%	Noisy	Size%
kNN	**82.11**	100.00	**78.93**	100.00
CNN	**76.48**	26.69	**68.14**	38.29
SNN	**75.44**	31.63	**74.60**	48.60
IB2	**76.58**	26.64	**67.80**	38.27
IB3	**78.85**	16.13	**72.09**	18.85
LED	**80.65**	16.88	**78.16**	8.57
DROP1	**72.65**	8.41	**71.24**	8.00
DROP2	**81.07**	14.03	**79.99**	14.75
DROP3	**81.14**	14.31	**80.00**	11.49
DROP4	**81.11**	17.30	**79.57**	14.74
DROP5	**81.39**	16.09	**79.95**	15.52
Wilson	**80.95**	83.34	**80.19**	74.91
Wilson2	**80.09**	80.92	**79.65**	72.53
AllKNN	**81.01**	77.44	**79.98**	64.58
ELGrow	**75.68**	1.83	**73.67**	1.88
Explore	**79.24**	2.01	**77.96**	2.03
Average	**79.06**	32.54	**76.36**	32.83

instances in the training set to an incorrect value (with an equal probability for each of the incorrect classes). The output class of the instances in the *test* set are not noisy, so the results indicate how well each model is able to predict the correct output even if some of its training data is mislabeled.

Table 2 shows the average accuracy and storage requirements over all 31 datasets for each algorithm, including the unpruned kNN algorithm.

As can be seen from Table 2, the accuracy for the kNN algorithm dropped just over 3% on average. Note that some of the effect of noise is already handled by the use of $k = 3$ in these experiments. Otherwise the drop in accuracy would be more on the order of 8% (i.e., 10% of the 82% already classified correctly).

As expected, CNN and IB2 increased storage and suffered large reductions in accuracy in the presence of noise. SNN dropped only slightly in accuracy when noise was added, but it retained almost half of the instances in the training set due to its strict (and noise intolerant) requirements as to which instances must be in S.

In agreement with Aha's results (1992), IB3 had higher accuracy and lower storage requirements in the presence of noise than IB2, though it still suffered a dramatic decrease in accuracy (and a slight increase in storage) when compared to its performance in the noise-free case. In our experiments we found that when the number of instances in the training set was small, IB3 would occasionally end up with an *empty* subset S, because none of the instances gets enough statistical strength to be *acceptable*. This problem worsens in the presence of noise, and thus more training data (or a modification of the algorithm) is required to handle small, noisy datasets.

DROP1 did not fall much in accuracy, but its accuracy was already poor to begin with. However, all of the other DROP methods (DROP2–DROP5) achieved accuracy *higher* than the kNN method, while using less than one-sixth of the original instances. DROP3 had the highest accuracy of the DROP methods, and had the lowest storage of the accurate ones (DROP2–DROP5), using less than 12% of the original instances.

The ENN, RENN, and All k-NN methods also achieved higher accuracy than kNN, since they were designed specifically for noise filtering. They also required about 10% less storage than in the noise-free case, probably because they were throwing most of the noisy instances (as well as a few good instances that were made to appear noisy due to the added noise).

The encoding-length heuristic methods all dropped about 2% in accuracy when noise was added leaving them closer to—but still below—the kNN method in terms of accuracy. ELGrow had fairly poor accuracy compared to the others, but Explore was within 1% of the kNN method in terms of accuracy while using only about 2% of the instances for storage.

6. Conclusions and Future Research Directions

Many techniques have been proposed to reduce the number of instances used for classification in instance-based and other exemplar-based learning algorithms. In experiments on 31 datasets, the results make possible the division of the tested algorithms into several groups. The first group consists of algorithms which had low generalization accuracy and are thus mostly of historical significance. This group includes CNN, SNN, IB2 (which led to the development of IB3) and DROP1 (which led to the more successful DROP algorithms). These had low generalization even before noise was introduced, and dropped further when it was. Of this group, only DROP1 kept less than 25% of the instances on average, so the storage reduction did not make up for the lack of accuracy.

The second group consists of the three similar noise-filtering algorithms: ENN, RENN, and All k-NN. These had high accuracy but also kept most of the instances. In the noise-free environment, they achieved slightly lower accuracy than kNN, but when noise was added, their accuracy was higher than kNN, indicating that they are successful in the situation for which they were designed. These algorithms are useful when noise is expected in the data and when it is reasonable to retain most of the data. Of this group, All k-NN had the highest accuracy and lowest storage requirements in the presence of noise.

The third group consists of two algorithms, ELGrow and Explore, that were able to achieve reasonably good accuracy with only about 2% of the data. ELGrow had the lowest storage (about 1.8%) but its accuracy was somewhat poor, especially in the noisy domain. The Explore method had fairly good accuracy, especially in the noisy arena, though it was not quite as accurate as the DROP methods. However, its aggressive storage reduction would make this trade-off acceptable in many cases.

The final group consists of algorithms which had high accuracy and reasonably good storage reduction. These include IB3, DROP2–DROP5 and DEL. IB3 was designed to overcome the noise-sensitivity of IB2, and in our experiments it had better accuracy and storage reduction than IB2, especially in the noisy case. However, its accuracy still dropped substantially in the noisy experiments, and it had lower average accuracy and higher average storage than the EXPLORE method both with and without noise.

The algorithms DROP2–DROP5 had higher average accuracy than IB3 on the original data, and had much higher average accuracy in the noisy case. They also improved in terms of average storage reduction as well. DROP2–DROP5 all had an accuracy within about 1% of kNN on the original data, and were about 1% higher when noise was added, with storage ranging from 11% to 18%. DEL had slightly lower accuracy than the DROP2–DROP5 methods, but had lower storage in the noisy domain.

DROP3 seemed to have the best mix of generalization accuracy and storage requirements of the DROP methods. DROP3 had significantly higher accuracy and lower storage than any of the algorithms in the first group; somewhat lower accuracy but significantly lower storage than any of the algorithms in the second group; and significantly worse storage but significantly better accuracy than the algorithms in the third group.

This paper has reviewed much of the work done in the area of reducing storage requirements in instance-based learning systems. The effect of uniform output class noise on many of the algorithms has also been observed on a collection of datasets. Other factors that influence the success of each algorithm must still be identified. Continued research should help determine under what conditions each of these algorithms is successful so that an appropriate algorithm can be automatically chosen when needed. Current research is also focused on combining the reduction techniques proposed here with various weighting techniques in order to develop learning systems that can more dynamically adapt to problems of interest.

Acknowledgments

The authors would like to thank Mike Cameron-Jones for providing code needed to implement the encoding-length heuristic techniques. Thanks also to David Aha who provided his code on-line and supplied many useful pointers to references relevant to this research. Pedro Domingos provided suggestions on including significance results which have strengthened the conclusions of this paper. Thanks also go out to Dan Ventura for his proofreading and to the anonymous reviewers for their help in bringing this paper into its final form.

Appendix

An on-line appendix is available at the following location.

`ftp://axon.cs.byu.edu/pub/randy/ml/drop/`

This site contains the complete source code used for these experiments, including the cost function for the encoding-length methods and the code used to generate t-test and Wilcoxon test statistics. The FTP site also contains all of the data sets and complete experimental results, including the accuracy for all 10 trials of each experiment, standard deviations, etc.

References

Aha, David W. (1992). Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, *36*, pp. 267–287.

Aha, David W., Dennis Kibler, Marc K. Albert (1991). Instance-Based Learning Algorithms. *Machine Learning*, *6*, pp. 37–66.

Batchelor, Bruce G. (1978). *Pattern Recognition: Ideas in Practice*. New York: Plenum Press.

Biberman, Yoram (1994). A Context Similarity Measure. In *Proceedings of the European Conference on Machine Learning (ECML-94)*. Catania, Italy: Springer Verlag, pp. 49–63.

Brodley, Carla E. (1993). Addressing the Selective Superiority Problem: Automatic Algorithm/Model Class Selection. *Proceedings of the Tenth International Machine Learning Conference*, Amherst, MA, pp. 17–24.

Broomhead, D. S., and D. Lowe (1988). Multi-variable functional interpolation and adaptive networks. *Complex Systems*, *2*, pp. 321–355.

Cameron-Jones, R. M. (1995). Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing. In *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, pp. 99–106.

Carpenter, Gail A., and Stephen Grossberg (1987). A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. *Computer Vision, Graphics, and Image Processing*, *37*, pp. 54–115.

Chang, Chin-Liang (1974). Finding Prototypes for Nearest Neighbor Classifiers. *IEEE Transactions on Computers*, *23–11*, November 1974, pp. 1179–1184.

Conover, W. J. (1971). *Practical Nonparametric Statistics*. New York: John Wiley, pp. 206–209, 383.

Cover, T. M., and P. E. Hart (1967). Nearest Neighbor Pattern Classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, *13–1*, January 1967, pp. 21–27.

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

DeGroot, M. H. (1986). *Probability and Statistics* (Second Edition). Reading, MA: Addison-Wesley.

Diday, Edwin (1974). Recent Progress in Distance and Similarity Measures in Pattern Recognition. *Second International Joint Conference on Pattern Recognition*, pp. 534–539.

Dietterich, Thomas G. (1989). Limitations on Inductive Learning. In *Proceedings of the Sixth International Conference on Machine Learning*. San Mateo, CA: Morgan Kaufmann, pp. 124–128.

Domingos, Pedro (1995). Rule Induction and Instance-Based Learning: A Unified Approach. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada: Morgan Kaufmann, pp. 1226–1232.

Domingos, Pedro (1996). Unifying Instance-Based and Rule-Based Induction. *Machine Learning*, *24*, pp. 141–168.

Dudani, Sahibsingh A. (1976). The Distance-Weighted k-Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man and Cybernetics*, *6–4*, April 1976, pp. 325–327.

Gates, G. W. (1972). The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, *IT–18–3*, pp. 431–433.

Hart, P. E. (1968). The Condensed Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, *14*, pp. 515–516.

Hecht-Nielsen, R. (1987). Counterpropagation Networks. *Applied Optics*, *26–23*, pp. 4979–4984.

Kibler, D., and David W. Aha (1987). Learning Representative Exemplars of Concepts: An Initial Case Study. *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine, CA: Morgan Kaufmann, pp. 24–30.

Lowe, David G. (1995). Similarity Metric Learning for a Variable-Kernel Classifier. *Neural Computation*. *7–1*, pp. 72–85.

Merz, C. J., and P. M. Murphy (1996). *UCI Repository of Machine Learning Databases.* Irvine, CA: University of California Irvine, Department of Information and Computer Science. Internet: http://www.ics.uci.edu/ ~mlearn/ MLRepository.html.

Michalski, Ryszard S., Robert E. Stepp, and Edwin Diday (1981). A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts. *Progress in Pattern Recognition*, *1*, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland, pp. 33–56.

Mitchell, Tom M., (1980). The Need for Biases in Learning Generalizations. In J. W. Shavlik & T. G. Dietterich (Eds.), *Readings in Machine Learning*, San Mateo, CA: Morgan Kaufmann, 1990, pp. 184–191.

Nadler, Morton, and Eric P. Smith (1993). *Pattern Recognition Engineering.* New York: Wiley.

Papadimitriou, C. H., and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, NJ.

Papadimitriou, Christos H., and Jon Louis Bentley (1980). A Worst-Case Analysis of Nearest Neighbor Searching by Projection. *Lecture Notes in Computer Science*, *85*, Automata Languages and Programming, pp. 470–482.

Renals, Steve, and Richard Rohwer (1989). Phoneme Classification Experiments Using Radial Basis Functions. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'89)*, *1*, pp. 461–467.

Ritter, G. L., H. B. Woodruff, S. R. Lowry, and T. L. Isenhour (1975). An Algorithm for a Selective Nearest Neighbor Decision Rule. *IEEE Transactions on Information Theory*, *21–6*, November 1975, pp. 665–669.

Rumelhart, D. E., and J. L. McClelland (1986). *Parallel Distributed Processing*, MIT Press, Ch. 8, pp. 318–362.

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

Schaffer, Cullen (1994). A Conservation Law for Generalization Performance. In *Proceedings of the Eleventh International Conference on Machine Learning (ML'94)*, New Brunswick, NJ: Morgan Kaufmann, pp. 259–265.

Skalak, D. B. (1994). Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning (ML94)*. Morgan Kaufmann, pp. 293–301.

Specht, Donald F. (1992). Enhancements to Probabilistic Neural Networks. In *Proceedings International Joint Conference on Neural Networks (IJCNN '92)*, *1*, pp. 761–768.

Sproull, Robert F. (1991). Refinements to Nearest-Neighbor Searching in k-Dimensional Trees. *Algorithmica*, *6*, pp. 579–589.

Stanfill, C., and D. Waltz (1986). Toward memory-based reasoning. *Communications of the ACM*, *29*, pp. 1213–1228.

Tomek, Ivan (1976). An Experiment with the Edited Nearest-Neighbor Rule. *IEEE Transactions on Systems, Man, and Cybernetics*, *6–6*, pp. 448–452.

Tversky, Amos (1977). Features of Similarity. *Psychological Review*, *84–4*, pp. 327–352.

Wasserman, Philip D. (1993). *Advanced Methods in Neural Computing.* New York, NY: Van Nostrand Reinhold, pp. 147–176.

Watson, I., and F. Marir (1994). Case-Based Reasoning: A Review. *The Knowledge Engineering Review*, *9–4*, Cambridge, UK: Cambridge University Press.

Wess, Stefan, Klaus-Dieter Althoff, and Michael M. Richter (1993), Using k-d Trees to Improve the Retrieval Step in Case-Based Reasoning. *Topics in Case-Based Reasoning, First European Workshop (EWCBR-93)*, Springer-Verlag, pp. 67–181.

Wettschereck, Dietrich (1994). A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm, In *Proceedings of the 7th European Conference on Machine Learning*, LNAI-784, F. Bergadano and L. de Raedt (editors), pp. 323–335.

Wettschereck, Dietrich, and Thomas G. Dietterich (1995). An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms. *Machine Learning*, *19–1*, pp. 5–28.

Wettschereck, Dietrich, David W. Aha, and Takao Mohri (1997). A Review and Comparative Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. *Artificial*

Intelligence Review, *11*, issues 1–5, Special Issue on *Lazy Learning*, Klewer Academic Publishers, pp. 273–314.

Wilson, D. Randall, and Tony R. Martinez (1996). Heterogeneous Radial Basis Functions. *Proceedings of the International Conference on Neural Networks (ICNN'96)*, *2*, pp. 1263–1267.

Wilson, D. Randall, and Tony R. Martinez (1997a). Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research (JAIR)*, *6–1*, pp. 1–34.

Wilson, D. Randall, and Tony R. Martinez (1997b). Improved Center Point Selection for Radial Basis Function Networks. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA'97)*.

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

Wilson, Dennis L. (1972). Asymptotic Properties of Nearest Neighbor Rules Using Edited Data. *IEEE Transactions on Systems, Man, and Cybernetics*, *2–3*, pp. 408–421.

Wolpert, David H. (1993). On Overfitting Avoidance as Bias. Technical Report SFI TR 92-03-5001. Santa Fe, NM: The Santa Fe Institute.

Zhang, Jianping (1992). Selecting Typical Instances in Instance-Based Learning. *Proceedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland: Morgan Kaufmann, pp. 470–479.