In Fisher, D., ed., *Machine Learning: Proceedings of the Fourteenth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 404-411, 1997.

Instance Pruning Techniques

D. Randall Wilson, Tony R. Martinez
Computer Science Department
Brigham Young University
Provo, UT 84058
randy@axon.cs.byu.edu, martinez@cs.byu.edu

Abstract

The nearest neighbor algorithm and its derivatives are often quite successful at learning a concept from a training set and providing good generalization on subsequent input vectors. However, these techniques often retain the entire training set in memory, resulting in large memory requirements and slow execution speed, as well as a sensitivity to noise. This paper provides a discussion of issues related to reducing the number of instances retained in memory while maintaining (and sometimes improving) generalization accuracy, and mentions algorithms other researchers have used to address this problem. It presents three intuitive noise-tolerant algorithms that can be used to prune instances from the training set. In experiments on 29 applications, the algorithm that achieves the highest reduction in storage also results in the highest generalization accuracy of the three methods.

1. INTRODUCTION

The nearest neighbor algorithm (Cover & Hart, 1967; Dasarathy, 1991) has been used successfully for pattern classification on many applications. Each pattern has an input vector with one value for each of several input attributes. An instance has an input vector and an output class. A training set T is a collection of instances with known output classes.

A new pattern x is classified in the basic nearest neighbor algorithm by finding the instance y in T that is closest to x, and using the output class of y as the predicted classification for x. The instance that is *closest* to x is taken to be the one with minimum distance, using some distance function. The distance function can have a significant impact on the ability of the classifier to generalize correctly.

The nearest neighbor algorithm learns very quickly (O(n)) because it need only read in the training set without further processing. It can also generalize accurately for many applications because it can learn complex concept descriptions, and provides an appropriate bias for many applications.

The nearest neighbor algorithm also has several shortcomings. Since it stores all training instances, it has large (O(n)) memory requirements. Since it must search through all available instances to classify a new input vector, it is slow (O(n)) during classification. 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) 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, do not address the problem of noise, and often become much less effective as the dimensionality of the problem (i.e., the number of input attributes) grows.

On the other hand, when some of the instances are removed (or *pruned*) 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 while trying to maintain or even improve generalization accuracy. Much research has been done in this area, and an overview of this research is given in Section 2.

Section 3 discusses several issues related to the problem of instance set reduction, and provides motivation for further algorithms. Section 4 presents four new instance reduction techniques that were hypothesized to provide substantial instance reduction while continuing to generalize accurately, even in the presence of noise. The first is similar to the Reduced Nearest Neighbor (RNN) algorithm (Gates 1972). The second changes the order in which instances are considered for removal, and the third adds a noise-reduction step similar to that done by Wilson (1972) before proceeding with the main reduction algorithm.

Section 5 describes experiments on 29 datasets that compare the performance of each of these three reduction techniques to a basic k-nearest neighbor algorithm. The results indicate that these algorithms achieve substantial storage reduction, while maintaining good generalization accuracy. Comparisons are also made with two of the best algorithms from the review in Section 2. Section 6 provides conclusions and future research directions.

2. RELATED WORK

Several researchers have addressed the problem of training set size reduction. This section reviews previous work in reduction algorithms.

2.1. NEAREST NEIGHBOR EDITING RULES

Hart (1968) made one of the first attempts to reduce the size of the training set with his *Condensed Nearest Neighbor Rule* (CNN). This algorithm begins by randomly selecting one instance belonging to each output class from the training set T and putting it in a subset 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 algorithm is especially sensitive to noise, because noisy instances will usually be misclassified by their neighbors, and thus will be retained.

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 a member of T (instead of S) of a different class. Further, the method ensures a minimal subset satisfying these conditions. This algorithm resulted in greater reduction in the training set size as well as slightly higher accuracy than CNN in their experiments, though it is still sensitive to noise.

Gates (1972) modified this algorithm in his *Reduced Nearest Neighbor Rule* (RNN). The reduced NN 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. 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.

Wilson (1972) developed an algorithm which removes instances that do not agree with the majority of their 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 many other algorithms. Tomek (1976) extended Wilson's algorithm with his "all k-NN" method of editing by calling Wilson's algorithm repeatedly with k=1..k, though this still retains internal points.

2.2. "INSTANCE-BASED" LEARNING ALGORITHMS

Aha et. al. (1991) presented a series of *instance-based* learning algorithms that reduce storage. *IB2* is quite similar to the Condensed Nearest Neighbor (CNN) rule (Hart, 1968), and suffers from the same sensitivity to noise.

IB3 (Aha et al. 1991) addresses IB2's problem of keeping noisy instances by using a statistical test to retain only *acceptable* misclassified instances. In their experiments, 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.

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.

Cameron-Jones (1995) used an *encoding length heuristic* to determine how good the subset S is in describing T. After a growing phase similar to IB3, a random mutation hill climbing method called *Explore* is used to search for a better subset of instances, using the encoding length heuristic as a guide.

2.3. 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.

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.

Domingos (1995) introduced the RISE 2.0 system which treats each instance in T as a rule in R, and then generalizes rules until classification accuracy starts to suffer.

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.

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

3. INSTANCE REDUCTION ISSUES

From the above learning models, several observations can be made regarding the issues involved in training set reduction. This section covers the issues of instance 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 strategies.

3.1. REPRESENTATION

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

On the other hand, many models seek to retain a subset of the original instances, including the Condensed NN rule (CNN) (Hart, 1968), the Reduced NN rule (RNN) (Gates 1972), the Selective NN rule (SNN) (Ritter et. al., 1975), Wilson's rule (Wilson, 1972), the "all *k*-NN" method (Tomek, 1976), Instance-Based (IBL) Algorithms (Aha et. al. 1991), and the Typical Instance Based Learning (TIBL) algorithm (Zhang, 1992).

Another decision that affects the concept description 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 integer (e.g., 1, 3 or 5) that is odd so as to avoid "ties" in the voting of neighbors. The value of k is often determined from cross-validation.

3.2. DIRECTION OF SEARCH

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

Incremental. The incremental search begins with an empty subset S, and adds each instance in T to S if it fulfills the criteria. Using such a search direction, 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. For example, CNN begins by selecting one instance from each class at random, which gives these instances a 100% chance of being included. The next instances visited are classified only by the few instances that are already in S, while instances chosen near the end of the algorithm are classified by a much larger number of instances that have been included in S. Other incremental schemes include IB2 and IB3.

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.

Decremental. Decremental searches begins with S=T, and then search for instances to remove from S. Again the order of presentation can be 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 include RNN, SNN, and Wilson's (1972) rule. NGE and RISE can also be viewed as decremental algorithms, except that instead of simply removing instances from S, they are instead generalized into hyperrectangles or rules. Similarly, Chang's prototype rule 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 and/or increased generalization accuracy, then the extra computation during learning (which is done just once) can be well worth the computational savings during execution thereafter.

Batch. A final 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, and then removing all of them at once. For example, the "all-*k*NN" rule 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, but there are also dangers in batch processing.

For example, assume we apply a rule such as "*Remove an instance if it has the same output class as its k nearest neighbors*" to a training set. 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.

3.3. 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 the border points, and thus can be removed with relatively little effect on classification. Algorithms which tend to retain border points include CNN, RNN, IB2, and IB3.

On the other hand, some algorithms instead seek to *remove* border points. Wilson's rule and the "all *k*NN" rule are examples of this. They remove points that are noisy or do not agree with their neighbors. This removes close border points, leaving smoother decision boundaries behind. This may help generalization in some cases, but typically keeps most of the instances.

Some algorithms retain center points instead of border points. For example, the Typical Instance Based Learning algorithm attempts to retain center points and can achieve very dramatic reduction (ideally one instance per class) when conditions are right. However, when decision boundaries are complex, it may take nearly as many instances to define the boundaries using center points as it would using boundary points. Current research is addressing this question.

3.4. DISTANCE FUNCTION

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

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

where $\boldsymbol{x}$ and $\boldsymbol{y}$ 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.

When *nominal* (discrete, unordered) attributes are included in an application, a distance metric is needed that can handle them. We use a distance function based upon the Value Difference Metric (VDM) (Stanfill & Waltz, 1986) for nominal attributes. A simplified version of the VDM defines the distance between two values x and y of a single attribute a as:

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

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—we use the heterogeneous distance function *HVDM* (Wilson & Martinez, 1997), which is defined as:

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

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

$$d_a(x, y) = \begin{cases} vdm_a(x, y), & \text{if } a \text{ is nominal} \\ \dfrac{|x - y|}{4\sigma_a}, & \text{if } a \text{ is numeric} \end{cases} \quad (4)$$

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 function handles unknown input values by assigning them a large distance (1.0), and provides appropriate normalization between numeric and nominal attributes, as well as between numeric attributes of different scales.

4. NEW INSTANCE SET REDUCTION ALGORITHMS

This section presents a collection of new heuristics or rules used to decide which instances to keep and which instances to remove from a training set. 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 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.

Given the issues in Section 3 to consider, our research is 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. Sections 4.1-4.3 present three rules, called RT1-RT3, respectively, that seek to meet these goals.

4.1. REDUCTION TECHNIQUE 1 (RT1)

The first reduction technique we present, RT1, 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 it.

To see if an instance can be removed using this rule, each associate 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+1st 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 the $P.A_i.N_{k+1}$ 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.

The algorithm for RT1 proceeds as shown in Figure 1.

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.

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. RT1 also removes instances in the center of clusters, because

```
RT1(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.
            Remove P from its associates' lists of nearest neighbors, and find
                the next nearest neighbor for each of these associates.
            Remove P from its neighbors' lists of associates.
        Endif
    Return S.
```

Figure 1: Pseudo-code for RT1.

associates there are not near their enemies, and thus continue to be classified correctly without *P*.

Near the border, the removal of some instances sometimes causes 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. REDUCTION TECHNIQUE 2 (RT2)

There is a potential problem that can arise in RT1 with regards 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 criteria, 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.

RT2 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.

Using this modification, each instance in the original training set *T* continues to maintain a list of its $k + 1$ nearest neighbors in *S*, even after it 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 RT1 such as removing entire clusters. This change is made by removing line 13 from the pseudo-code for RT1 so that pruned instances will still be associates of their nearest neighbors in *S*.

RT2 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. REDUCTION TECHNIQUE 3 (RT3)

RT2 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.

RT3 therefore uses a noise-filtering pass *before* sorting the instances in *S*. This is done using a rule similar to *Wilson’s Rule* (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.

5. EXPERIMENTAL RESULTS

The reduction algorithms RT1, RT2 and RT3 were implemented using $k = 3$, and using the *HVDM* distance function described in Section 3.4. These algorithms were tested on 29 data sets from the University of California, Irvine Machine Learning Database Repository (Merz & Murphy, 1996) and compared to a *k*-nearest neighbor classifier that was identical to RT1 except that it does not remove any instances from the instance set (i.e., *S*=*T*).

Each test consisted of ten trials, each using one of ten partitions of the data randomly selected from the data sets, i.e., 10-fold cross-validation. For each trial, 90% of the training instances were used for *T*, the subset *S* was determined using each reduction technique (except for the *k*NN algorithm, which keeps all the instances), and the remaining 10% of the instances were classified using only the instances remaining in *S*. The average generalization accuracy over the ten trials for each test is given in Table 1. The average percentage of instances retained in *S* is shown in the table as well.

Table 1: Generalization accuracy and storage requirements of *k*NN, RT1, RT2, and RT3.

Database	*k*NN	(size)	RT1	(size)	RT2	(size)	RT3	(size)	H-IB3	(size)
Anneal	**93.11**	100	**87.85**	9.11	**95.36**	11.42	**93.49**	8.63	**94.98**	7.81
Australian	**84.78**	100	**82.61**	7.67	**84.64**	15.41	**84.35**	5.93	**85.99**	6.48
Breast Cancer WI	**96.28**	100	**94.00**	2.56	**96.14**	5.79	**96.14**	3.58	**95.71**	2.56
Bridges	**66.09**	100	**55.64**	20.86	**59.18**	24.11	**58.27**	18.66	**59.37**	38.67
Crx	**83.62**	100	**81.01**	6.70	**84.93**	14.11	**85.80**	5.46	**83.48**	6.86
Echocardiogram	**94.82**	100	**93.39**	9.01	**85.18**	7.51	**93.39**	9.01	**93.39**	14.85
Flag	**61.34**	100	**58.13**	24.51	**62.34**	32.30	**61.29**	20.45	**51.50**	39.18
Glass	**73.83**	100	**60.30**	26.11	**64.98**	31.52	**65.02**	23.88	**67.77**	32.92
Heart	**81.48**	100	**79.26**	12.96	**81.11**	21.60	**83.33**	13.62	**76.30**	10.33
Heart.Cleveland	**81.19**	100	**77.85**	14.26	**79.87**	20.61	**80.84**	12.76	**74.23**	10.78
Heart.Hungarian	**79.22**	100	**78.92**	11.38	**79.22**	15.98	**79.95**	10.43	**74.83**	8.88
Heart.Long Beach VA	**70.00**	100	**73.00**	11.78	**72.00**	16.33	**73.50**	4.22	**69.50**	11.67
Heart.More	**74.17**	100	**73.20**	11.20	**74.50**	16.98	**76.25**	9.10	**74.75**	13.97
Heart.Swiss	**92.69**	100	**91.15**	2.08	**93.46**	2.89	**93.46**	1.81	**84.62**	4.79
Hepatitis	**80.62**	100	**76.21**	8.67	**82.00**	13.98	**81.87**	7.81	**72.79**	8.03
Horse-Colic	**57.84**	100	**65.09**	10.89	**66.17**	17.98	**71.08**	7.42	**61.82**	17.64
Image.Segmentation	**93.10**	100	**84.76**	10.21	**92.38**	13.76	**92.62**	10.98	**90.24**	14.79
Ionosphere	**84.62**	100	**84.91**	5.67	**88.32**	12.09	**87.75**	7.06	**88.32**	13.61
Iris	**94.00**	100	**89.33**	11.70	**95.33**	16.89	**95.33**	14.81	**92.00**	10.96
LED-Creator+17	**67.10**	100	**68.80**	17.50	**70.50**	26.23	**70.40**	12.66	**52.60**	42.42
LED-Creator	**60.60**	100	**64.00**	7.78	**67.00**	11.20	**68.20**	11.57	**71.30**	24.32
Liver.Bupa	**65.57**	100	**59.66**	27.28	**65.80**	37.55	**60.84**	24.99	**55.64**	15.59
Pima Indians Diabetes	**73.56**	100	**70.96**	20.12	**73.31**	28.11	**75.01**	16.90	**67.83**	13.22
Promoters	**93.45**	100	**85.00**	8.18	**87.91**	17.82	**86.82**	16.67	**88.59**	10.02
Sonar	**87.55**	100	**69.81**	24.04	**81.86**	30.82	**78.00**	26.87	**71.67**	15.22
Soybean-Large	**88.59**	100	**79.81**	24.03	**84.99**	27.94	**85.62**	25.73	**90.23**	21.90
Vehicle	**71.76**	100	**66.21**	24.00	**68.91**	31.60	**65.85**	23.00	**66.45**	29.17
Vowel	**96.57**	100	**88.98**	43.14	**91.46**	46.91	**89.56**	45.22	**94.70**	23.42
Wine	**94.93**	100	**91.05**	8.55	**93.79**	15.23	**94.93**	16.11	**94.38**	13.42
Average	**80.78**	100	**76.93**	14.55	**80.09**	20.16	**80.31**	14.32	**77.41**	16.67

Results were also obtained for several of the methods discussed in Section 2, including CNN (Hart, 1968), SNN (Ritter et al., 1975), Wilson's Rule (Wilson, 1972), the "All *k*-NN" method (Tomek, 1976), IB2, IB3 (Aha, Kibler & Albert, 1991), and the *Explore* method (Cameron-Jones, 1995). Space does not permit the inclusion of all of the results, but the results for IB3 are included in Table 1 as a benchmark for comparison, since it is one of the most popular algorithms. This version of IB3 was modified to use the heterogeneous distance function *HVDM* and is thus labeled H-IB3.

From the results in this table, several observations can be made. RT1 had very good storage reduction on average, but also dropped in generalization accuracy by an average of almost 4%. This is likely due to the fact that this algorithm ignores those instances that have already been pruned from S when deciding whether to remove additional instances. It also prunes them in a random order, thus causing some border points to be removed too early.

RT2 had better generalization accuracy than RT1 because of its use of the additional information provided by pruned instances in determining whether to remove others. However, it also has higher storage requirements, due at least in part to the fact that noisy instances sometimes cause nearby instances to be retained even after the noisy instance is removed.

RT3 had a higher average generalization accuracy than RT1, RT2, and H-IB3, and also had the lowest storage requirements of the four. Its generalization accuracy was within one-half of a percent of the *k*NN algorithm that retained 100% of the instances, and yet it retained on average only 14.32% of the instances.

Some datasets seem to be especially well suited for these reduction techniques. For example, RT3 required less than 2% storage for the *Heart.Swiss* dataset, yet it achieved even higher generalization accuracy than the *k*NN algorithm. On the other hand, some datasets were not so appropriate. On the Vowel dataset, for example, RT3 required over 45% of the data, and dropped in generalization accuracy by 7%, suggesting that RT3 is inappropriate for this particular dataset.

The Heterogeneous IB3 algorithm was also compared to the original IB3 algorithm (i.e., with $k = 1$, and using Euclidean distance on linear attributes and overlap metric on nominal attributes), to make sure that the above comparison was fair. The original IB3 algorithm attained 75% accuracy with 18% storage on average on these 29 datasets, indicating that the heterogeneous distance function was beneficial to IB3.

As mentioned above, the results in Table 1 were also compared with those achieved by CNN, SNN, Wilson's Rule, and the All k-NN method. All of these resulted in lower generalization accuracy and higher storage requirements than RT3. IB2, CNN and SNN all had about 76% accuracy and required about 25-33% storage. Wilson's Rule and the All k-NN method both had 80.1% accuracy but retained over 75% of the instances.

The *Explore* method (Cameron-Jones, 1995) mentioned in Section 2.2 was also tested, and though its average accuracy was lower than RT3 (76.23%), it reduced storage more dramatically than the others, retaining only 2.23% of the instances on average. When modified with the *HVDM* distance function, its storage improved further (2.11%) and accuracy increased to 79.01%. This is still not as accurate as RT3, but the improved reduction may be worth the trade-off in some cases.

One factor that influences the amount of reduction achieved by RT3 is the use of $k = 3$. This causes some instances to be retained that could be removed with $k = 1$. Initial experiments suggest that the smaller value of k results in more dramatic storage reduction, but may reduce generalization accuracy slightly in RT3. One area of future research is in the use of a dynamic value of k, where k starts out with a value of 3 or 5 and is reduced as pruning continues until it is eventually reduced to a value of 1.

We were interested to see how fast generalization accuracy drops as instances are removed from S, and to see if it drops off more slowly when using an intelligent reduction technique than it does when removing instances randomly.

In order to test this, the experiments described above were modified as follows. When instances were removed from S, the order of their removal was recorded so that the training set T could be sorted by order of removal (with the instances in S at the beginning in random order, since they were not removed). Then the instances in the test set were classified using only the first 1% of the data, then 2%, and so on up to 100% of the data. Figure 2 shows the average generalization accuracy (over all 10 trials on all 29 datasets) as a function of the percentage of the training set that was used for generalization.

As can be seen from the figure, the average generalization accuracy drops off more quickly if instances are randomly removed than if they are removed using RT3, indicating that the order of removal is improved by RT3.

This can be useful if more storage is available than that required by RT3. For example, if an instance set of 10 million instances is reduced to just 100, but there is sufficient storage and computational resources to handle 1000 instances, then by using the sorted list of pruned instances, the best 1000 instances can be used. The user is thus allowed to manually trade off storage for accuracy without the higher loss in accuracy that random removal would cause.

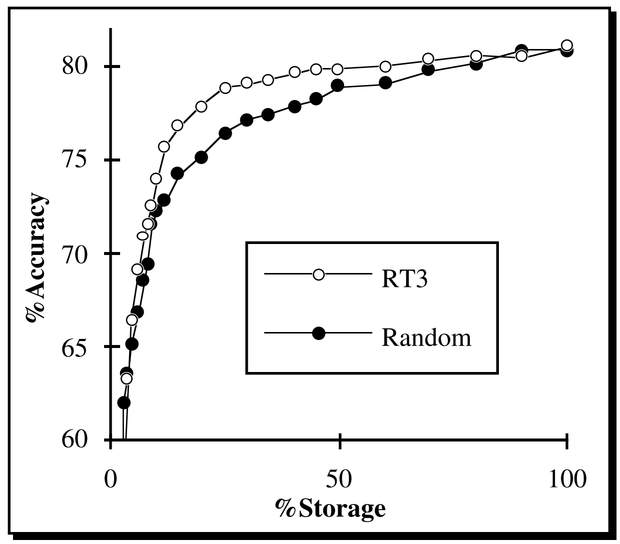

Figure 2: Generalization Accuracy for RT3 vs. Randomly removing instances.

6. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

Nearest neighbor algorithms and their derivatives are often appropriate and can provide high generalization accuracy for real-world applications, but the storage and computational requirements can be restrictive when the size of the training set is large.

This paper introduced three new instance reduction techniques which are intuitive and provide good storage reduction. In experiments on 29 datasets, the third technique, RT3, provided higher generalization accuracy and lower storage requirements than the other two methods, and its accuracy was within 0.5% of that of a nearest neighbor classifier that retained all of the instances. On average it retained under 15% of the original training set.

RT3 (and to a lesser extent the other algorithms) is designed to be robust in the presence of noise and use an estimate of generalization accuracy in making decisions, which helps it avoid removing instances that would be helpful in generalization. Since RT3 makes use of all the instances in the training set in making its decision, it is not sensitive to the order of presentation of the instances (as are incremental approaches), and is able to choose a good order of removal regardless of how the instances were ordered in the original training set.

These reduction algorithms were also among the first to use heterogeneous distance functions that are appropriate for applications with both nominal and continuous attributes (Wilson & Martinez, 1997).

Future research will focus on determining the conditions under which these algorithms are not appropriate, and will seek to overcome weaknesses in such areas. These reduction algorithms will also be integrated with feature selection algorithms and weighting techniques in order to produce comprehensive instance-based learning systems that are robust in the presence of irrelevant attributes and other difficult circumstances, thus providing more accurate learning algorithms for a wide variety of problems.

References

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

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.

Chang, Chin-Liang, (1974). "Finding Prototypes for Nearest Neighbor Classifiers," *IEEE Transactions on Computers*, vol. 23, no. 11, November 1974, pp. 1179-1184.

Cover, T. M., and P. E. Hart, (1967). "Nearest Neighbor Pattern Classification," *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, vol. 13, no. 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.

Domingos, Pedro, (1995). "Rule Induction and Instance-Based Learning: A Unified Approach," to appear in *The 1995 International Joint Conference on Artificial Intelligence (IJCAI-95)*.

Gates, G. W. (1972). "The Reduced Nearest Neighbor Rule," *IEEE Transactions on Information Theory*, vol. IT-18, no. 3, pp. 431-433.

Hart, P. E., (1968). "The Condensed Nearest Neighbor Rule," Institute of Electrical and Electronics Engineers Transactions on Information Theory, vol. 14, pp. 515-516.

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.

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

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*, vol. 21, no. 6, November 1975, pp. 665-669.

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

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

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

Tomek, Ivan, (1976). "An Experiment with the Edited Nearest-Neighbor Rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 6, June 1976, pp. 448-452.

Wettschereck, Dietrich, (1994). "A Hybrid Nearest-Neighbor and Nearest-Hyperrectangle Algorithm", *To appear in the Proceedings of the 7th European Conference on Machine Learning*.

Wettschereck, Dietrich, and Thomas G. Dietterich, (1995). "An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms," *Machine Learning*, vol. 19, no. 1, pp. 5-28.

Wilson, D. Randall, and Tony R. Martinez, (1997). "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research (JAIR)*, vol. 6, no. 1, pp. 1-34.

Wilson, Dennis L., (1972). "Asymptotic Properties of Nearest Neighbor Rules Using Edited Data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408-421.

Zhang, Jianping, (1992). "Selecting Typical Instances in Instance-Based Learning," *Proceedings of the Ninth International Conference on Machine Learning*.