Intelligent Data Analysis 8 (2004) 61–78
IOS Press

Weighted Instance Typicality Search (WITS): A nearest neighbor data reduction algorithm

Brent D. Morring[a] and Tony R. Martinez[b]
Computer Science Department, Brigham Young University, Provo, UT 84602, USA
[a]*E-mail: morringb@axon.cs.byu.edu*
[b]*E-mail: martinez@cs.byu.edu*

Received 26 November 2002
Revised 28 December 2003
Accepted 13 February 2003

Abstract. Two disadvantages of the standard nearest neighbor algorithm are 1) it must store all the instances of the training set, thus creating a large memory footprint and 2) it must search all the instances of the training set to predict the classification of a new query point, thus it is slow at run time. Much work has been done to remedy these shortcomings. This paper presents a new algorithm WITS (Weighted-Instance Typicality Search) and a modified version, Clustered-WITS (C-WITS), designed to address these issues. Data reduction algorithms address both issues by storing and using only a portion of the available instances. WITS is an incremental data reduction algorithm with $O(n^2)$ complexity, where n is the training set size. WITS uses the concept of Typicality in conjunction with Instance-Weighting to produce minimal nearest neighbor solutions. WITS and C-WITS are compared to three other state of the art data reduction algorithms on ten real-world datasets. WITS achieved the highest average accuracy, showed fewer catastrophic failures, and stored an average of 71% fewer instances than DROP-5, the next most competitive algorithm in terms of accuracy and catastrophic failures. The C-WITS algorithm provides a user-defined parameter that gives the user control over the training-time vs. accuracy balance. This modification makes C-WITS more suitable for large problems, the very problems data reductions algorithms are designed for. On two large problems (10,992 and 20,000 instances), C-WITS stores only a small fraction of the instances (0.88% and 1.95% of the training data) while maintaining generalization accuracies comparable to the best accuracies reported for these problems.

Keywords: Instance-based learning, nearest-neighbor, instance reduction, pruning, classification

1. Introduction

The nearest neighbor algorithm [1,2,6,7] is a conceptually simple machine learning technique that has proven to be successful over a wide range of classification problems. The simplest form of the algorithm consists of storing all n instances of the training set T. Each instance x consist of a vector of input values and a corresponding output class. When a new instance or query point y needs to be classified, the distance between instance y and each instance in the training set T is calculated according to a distance function $D(y, x)$ defined over the input space. The k nearest neighbors to the point y vote to determine y's classification. The simplest algorithm uses $k = 1$. The nearest neighbor algorithm has the power to represent very complex decision surfaces and is very fast at train time because it simply stores the training set T. Another advantage is the ability for humans to understand why the algorithm predicted a particular class for a given query point. With many other algorithms (Neural Nets, Rules, Statistical Classifiers) it is more difficult for humans to understand why or exactly how the algorithm made its decision. Its

disadvantages include that it requires large storage to hold T and it is slow at run-time because the distance between the query point and each instance in T must be calculated. This paper presents a new algorithm WITS (Weighted-Instance Typicality Search) and a modified version, Clustered-WITS (C-WITS), designed to address these issues. WITS and C-WITS are compared to three other state of the art data reduction algorithms on ten real-world datasets. WITS achieved the highest average accuracy, showed fewer catastrophic failures, and stored an average of 71% fewer instances than DROP-5, the next most competitive algorithm in terms of accuracy and catastrophic failures. The C-WITS algorithm provides a user-defined parameter that gives the user control over the training-time vs. accuracy balance. This modification makes C-WITS more suitable for large problems, the very problems data reductions algorithms are designed for. On two large problems (10,992 and 20,000 instances), C-WITS stores only a small fraction of the instances (0.88% and 1.95% of the training data) while maintaining generalization accuracies comparable to the best accuracies reported for these problems.

A number of issues must be addressed when designing the standard algorithm or any of its many variants. These issues are discussed in greater detail in Section 2. An appropriate distance function must be chosen. In many cases the input vector contains real, discrete numeric, and nominal data simultaneously. The distance function must handle this situation. Many useful distance functions have been developed [21,26,28]. In many applications the various inputs have differing relevance to the classification task at hand. The extreme case is when a given input feature contains no useful information for the classification task. Irrelevant or less relevant features dilute the effectiveness of the distance function and can degrade classification accuracy. Various feature weighting schemes [23] have been developed to mitigate this problem. Since the effects of irrelevant features is not the focus of this paper we have chosen problems that are not plagued by this problem and have employed no feature weighting schemes in our algorithms.

Two general approaches to speedup the run-time of nearest neighbor algorithms are known. One approach is to index and structure the stored instances in order to perform a quicker search for the nearest neighbors [10]. These types of approaches improve the run-time but do nothing to reduce storage requirements. The second approach is to reduce the set of instances stored and used for classification [29]. These techniques are often referred to as pruning or data reduction. Data reduction techniques have the advantage of reducing storage and run-time simultaneously.

Another modification to the original algorithm is Instance-Weighting [5]. This method associates a weight value with each instance in the solution. The weight value dictates an effective sphere of influence for that instance. When the distance between a query point y and a given instance x is calculated, the weight w associated with x is multiplied by $D(y, x)$. If w is small it will make the effective distance between x and any other point smaller thus making x more influential in the classifier. On the other hand if w is large, then the distance between x and any other point is effectively larger and x has less of an influence in the space. For a given set of points, instance weighting adds complexity to the model (more degrees of freedom) making it more powerful in the sense of its ability to fit data.

The before mentioned issues and modifications have often been investigated in isolation. This is done to illuminate the effect of that particular issue and may be necessary to keep the computational complexity reasonable. It stands to reason that better solutions could be found if various parameters and modifications were optimized and integrated in a simultaneous fashion. WITS uses the concept of Typicality in conjunction with Instance-Weighting to produce minimal nearest neighbor solutions.

Section 2 discusses various issues encountered when using nearest neighbor algorithms. Section 3 presents the WITS and C-WITS algorithms with corresponding results and analysis on various artificial and real-world problems. Section 4 provides a conclusion and future works discussion.

2. Nearest neighbor learners

There are many variants of the nearest neighbor algorithm and a number of design issues that must be addressed. This section discusses modifications to the basic algorithm and points out some of the strengths and weaknesses.

2.1. Distance metrics

The distance metric employed is clearly a key ingredient in the nearest neighbor (NN) philosophy. A poorly chosen distance function could severely impair the generalization ability of the algorithm. The original NN algorithm was designed to work with real valued inputs and employed a Euclidean distance metric. The simple Overlap metric, which returns a zero if the attribute values are the same and a one if different, has often been used for linear and nominal data. [21] introduced the Value Difference Metric (VDM), which is a significant improvement over the Overlap metric. The VDM determines the distance between two values of a given input variable based on how similarly they classify the output variable. For example, if 'Age' is a nominal input variable with {young, middle, old, very old} as possible values and the output variable is 'Safe Driver' with values {yes, no} then the distance between 'young' and 'very old' would be small because they tend to have the same classification of 'no' on 'Safe Driver'. The distance between 'young' and 'middle' would be larger because they have greater difference in classification on the output variable. Variants of the VDM have appeared in recent years and the VDM is often used with Euclidean metrics in a heterogeneous fashion [27]. A further improvement has been made by discretizing real valued inputs and applying the VDM (or one of its variants) to them also [26].

The algorithms in this paper use the Heterogeneous Value Difference Metric (HVDM) as presented in [28]. The HVDM uses the VDM for nominal valued inputs and Euclidean distance for real valued inputs. This metric was chosen because it shows good performance over a range of problems and is simple to implement.

2.2. Feature weighting

By allowing all the input features to contribute to the distance metric equally, irrelevant, redundant or noisy features can degrade classification accuracy. This is a major shortcoming of the original NN algorithm. Several NN variants have been developed to mitigate this problem. A nice review of these variants is presented in [23]. In that review the feature weighting methods are classified along five dimensions: Bias – (Performance, Preset), Weight Space – (Continuous, Binary), Representation – (Given, Transformed), Generality – (Global, Local), and Knowledge – (Poor, Intensive). Performance biases make use of how well the weight settings actually perform on the given system in the weight adjustment phase. Performance biases are expected to outperform preset biases because they work with the inductive bias of the learner while a preset bias may conflict with it. The weight space dimension refers to the nature of the weights assigned to the features. Continuous refers to the range of real numbers while binary allows only the values zero and one and performs feature selection. The representation dimension refers to whether or not the original features are weighted or are transformed to some new feature space and then weighted. Generality distinguishes between methods that apply a feature weight to the whole instance space in a global fashion or allow a more complex setting of weights to distinguish relevance within local sub regions of the instance space. After empirical testing they conclude that weighting methods that use performance to set the weight values exhibit a number of advantages over other methods. They require less pre-processing, work better with interacting features, and require less training data to find good weight values. They also found that a continuous weight space outperforms feature selection methods in the cases with less-relevant features.

2.3. Data reduction

One method of reducing the storage cost and classification time of a nearest neighbor algorithm is to store and search only a fraction of the training instances. This approach produces an advantage in data storage and classification speed simultaneously while the methods mentioned in the following section only improve the classification speed. When using a data reduction technique, one expects the classification accuracy to degrade somewhat, but in some cases, the classification accuracy can actually improve. The goal in designing a data reduction method is to store as little data as possible while maintaining the best accuracy possible or an appropriate trade-off between the two. The accuracy is often compared to a k-NN system that uses 100% of the training data as the solution. Even though classification accuracy generally degrades, the reduced storage requirements may make a data reduction technique the best choice for many situations.

A nice review of many data reduction techniques is presented in [29]. In this paper a framework is suggested for classifying data reduction algorithms. Three dimensions are particularly useful: representation, direction of search and border vs. central points. Representation refers to whether the algorithm stores the chosen instances in their original form or stores instances in a new representational form. The direction of search can proceed in an incremental, decremental or batch mode. An incremental mode is when the solution S begins empty and then instances are added as some criteria are met. A decremental algorithm begins with all the instances belonging to S and then instances are removed as some criteria are met. Both the incremental and decremental modes can be susceptible to the presentation order of the training set T. In batch mode the criteria for adding or removing an instance is made for each instance, taking into account the full training set T, before any instances are added or removed. This mitigates the presentation order problem but may require increased training time. The third dimension distinguishes between algorithms that tend to save instances close to the border of a different class or instances surrounded by other instances of its own class, and thus represent the whole cluster. Saving the central points creates smoother surfaces and presents greater potential for data reduction but may perform worse than border saving algorithms on problems with very complex decision surfaces.

2.4. Speed classification

Speed classification techniques attempt to improve classification speed by performing a more efficient search for the nearest neighbor(s) of the query point. This is generally accomplished by placing the training instances into various data structures that facilitate the efficient search. Projection and k-d trees [20] are examples of speed classification techniques. [7] presents a good review of many speed classification techniques as well as the historical development. These techniques do not sacrifice classification accuracy because the full training set is stored. On the down side, the large storage requirements for these techniques make them unsuitable for some applications. It has also been observed that their speedup performance degrades as the dimensionality of the problem grows larger [20].

2.5. Instance weighting

Instance weighting should not be confused with feature weighting. An instance weight is a real number in the range [0,inf) assigned to a particular instance, which indicates the influence of that instance in the space. When the distance between a new query point and an instance is calculated the instance's weight is multiplied by the distance to give the final distance. Instances with small weights are more likely to be the nearest neighbor of new query points than instances with large weights, since they influence a

larger region of the space. Without instance weights all the instances have equal influence in the space. A more complex decision surface can be created with fewer instances if instance weights are employed. To illustrate this we consider the case of a region of space predominately of class A with a smaller exceptional region of class B within the class A region. Without instance weighting many instances of class A surrounding the exceptional region must be stored in order to contain the influence of the stored class B instances. With instance weights, only two instances need be stored in this case, one class A instance, with a small weight, to dominate the region and one class B instance, with a large weight, to define the exception space.

Class noise and exceptions in training data are common in many problems and need to be handled appropriately by any machine learning algorithm. As previously shown, instance weighting provides a natural way to deal with noise or exceptions. If it is possible to identify the noise then it is advantageous to leave it out of the solution. If noise identification is difficult or impossible, instance weighting allows the influence of the stored noisy instances to be controlled.

3. WITS algorithm

The Weighted-Instance Typicality-Search (WITS) Algorithm uses instance weighting in conjunction with an aggressive data reduction technique based on the idea of the typicality of instances. This section explains the concept of typicality, outlines the details of the WITS algorithm, and concludes with a summary and analysis of the results of three artificial data sets and nine real world data sets.

3.1. Typicality

Typicality is a measure of how representative a given instance is of its class. The concept was introduced in [30]. Typicality is defined as the ratio of the intra-class similarity to the inter-class similarity. Intra-class similarity of a point x is defined as the average of $1 - D(x, y)$, where the average is computed over all y such that Class$(y) =$ Class(x). Inter-class similarity has the same definition but the average is over all y such that Class$(y) \neq$ Class(x). A larger intra-class similarity indicates a larger typicality while a large inter-class similarity produces a smaller typicality. Central points of clusters have the highest typicality scores, border points have intermediate values while noisy or exception points have the lowest typicality scores.

3.2. The WITS algorithm

The primary purpose of the WITS algorithm is to produce instance-based solutions exhibiting excellent generalization performance while using very few instances in order to decrease storage and run time. This algorithm tends to store central points first, then accepts intermediate and border points if they are necessary and exhibit an acceptable degree of generalization. Because WITS tends to store very few points and those stored are central points that represent whole classes or major regions of a class, it makes sense to use the single nearest neighbor to classify novel query points, thus it is a $k = 1$ algorithm. Storing central points also has the advantage of creating smoother decision boundaries that often exhibit better generalization performance.

There are three principle components to WITS, a search order through the training data based on typicality, a method for assigning weights to instances, and a generalization criterion for accepting potential instances into the solution. The next three sections explain each of these components.

3.2.1. Search order

WITS is an incremental method so the solution S starts empty. As a preprocessing step, the training data T is divided into buckets according to class (we call these class buckets) and within each class bucket they are sorted in a Largest Typicality First order (LTF). The algorithm loops while 1) there are instances that are misclassified by the current S and 2) there are instances that have not been considered for inclusion into S yet. Instances that match both of those conditions are candidates for inclusion into S and are considered in a LTF order. The candidate instances pertaining to the class bucket with the greatest number of errors are searched first. As an instance is considered it either passes the generalization criteria (explained in 3.2.3) and is added to the solution S or it fails the criteria and is marked 'Tried' so it is no longer a candidate. Each time an instance is successfully added to S the errors per class bucket are re-evaluated.

3.2.2. Weight assignment

Once a candidate instance has been chosen for possible inclusion into S an appropriate weight value must be assigned to it. The weights come from the positive real numbers. Negative weights are nonsensical in this situation. Remember that a small weight value corresponds to a large influence in the space and larger weight values produce smaller spheres of influence. The desired weight value is the one that maximizes accuracy on the training data, thus the task is to find the maximum of a function, which extends over the positive half of the reals, whose characteristics are only partially known. When the weight value reaches a certain magnitude, the sphere of influence about that instance ceases to include any other instances from the training set and testing larger weight values serves no purpose. This problem of finding the weight value that maximizes the accuracy is a complex problem in and of itself because there are local maxima involved. This weight assignment step must be performed efficiently because it is executed $O(n)$ times during training, where n is the training set size. The WITS algorithm uses a naïve weight assignment approach. It simply searches through a predefined set of weight values and chooses the value that maximizes the training set accuracy. WITS uses a two parameter exponential function to define the set of weight values. The values are created from $f = b^k$ where b is a parameter in the range $(1, \infty)$ and $k = (0, 1, 2, \ldots, m)$ where m is a user set parameter. For example, if $b = 1.1$ and $m = 20$ then the weight value set is [1.0, 1.1, 1.21, 1.331, ..., 6.116, 6.727]. Although this is clearly an area where improvement could be made, the results while using this naïve approach succeed in demonstrating the usefulness of instance weighting in finding low data solutions. Parameter values of $b = 1.11$ and $m = 30$ were used in all problems presented in this paper.

3.2.3. Generalization criteria

After the best weight value for the candidate instance has been determined it must pass a generalization condition before it is added to S. The extended solution, S′, must not only improve accuracy over the training set but also do so by a margin that exhibits generalizing ability. For this purpose, accuracy is not measured as a percentage but as the 'number of training instances correctly classified' (NumCorrect). NumCorrect is measured in whole number units. A generalizing parameter, G, designates the size of the margin in whole numbers. A G value of one means that the extended solution, S′, must correctly classify at least one more instance of the training set than S. This exhibits a minimal amount of generalization. This is an important mechanism to prevent the memorization and subsequent overfit of the training data. This mechanism also prevents noisy instances from being included in the solution. One can bias the algorithm against noise and toward generalization by increasing the parameter G. By making G a function of the training set size one keeps the solution S from growing larger and larger as more data is made available. In WITS, G is set as a percentage of the training set size. G is not allowed to have a value of zero. A G value of (0.005*trainset size) was used for the results reported below.

```
InitializeTypicality()
SortAccordingToTypicality()
MarkDataUntried()
While (TrainAccuracy() < 100%) and (UntriedInstancesExist == 'true'):
    AddInstanceToSolution(traindata)
    SetNewInstanceWeight()
    If (GeneralizationCriteria(solution) == 'not pass'):
        ResetSolution()
        MarkInstanceTried()
```

Fig. 1. Pseudocode for the WITS algorithm.

Table 1
Dataset characteristics

Dataset	# Nominal inputs	# Linear/real inputs	# Output classes	Majority class %	Size of dataset
Glass	0	10	7	36	214
Ionosphere	0	34	2	64	351
Iris	0	4	3	33	150
Liver (Bupa)	0	6	2	60	345
Pima (Diabetes)	0	8	2	65	768
Promoters	58	0	2	50	106
Segment	0	19	7	14	2310
Sonar	0	60	2	53	208
Vote	16	0	2	61	435
Zoo	1	6	7	41	101

3.3. Results

The WITS algorithm has been tested on three artificial datasets and ten real world datasets chosen from the Machine Learning Database Repository at the University of California, Irvine [14]. The three artificial datasets are presented to show graphically how the algorithm works. For this reason, the artificial datasets are two dimensional in the inputs and have only two output classes. The ten real world datasets have been chosen so as to have differing input and output characteristics. Table 1 indicates the input and output characteristics of the ten datasets as well as the dataset size.

3.3.1. Artificial problems: Isthmus, Ring, and Gaussian

The artificial datasets were created in a two dimensional real space bound to the region of [0,1] in each dimension. They also contain only two output classes in order to make the graphing and visualization easy. Figures 2 and 4 show the underlying distributions for the 'Isthmus' and 'Ring' problems along with the data points generated. The 'Gaussian' problem's underlying distribution consist of two 2-dimensional gaussians centered at (0.25,0.25) and (0.75,0.75) respectively. Each gaussian has a variance of 0.2 in both the x and y dimensions. Figure 6 shows the data generated for the gaussian problem and bisecting line between the two gaussians. The 'Gaussian' problem differs from the 'Isthmus' and 'Ring' problems in that its underlying distribution has an overlapping region instead of mutually exclusive regions. Figures 3, 5 and 7 show the decision boundaries learned by the WITS algorithm on these problems.

The ovals imposed on Fig. 2 are not part of the distribution. The ovals encircling the corners in Fig. 2 highlight the lack of 'diamond' instances in those regions, which explains the 'squares' incursion into

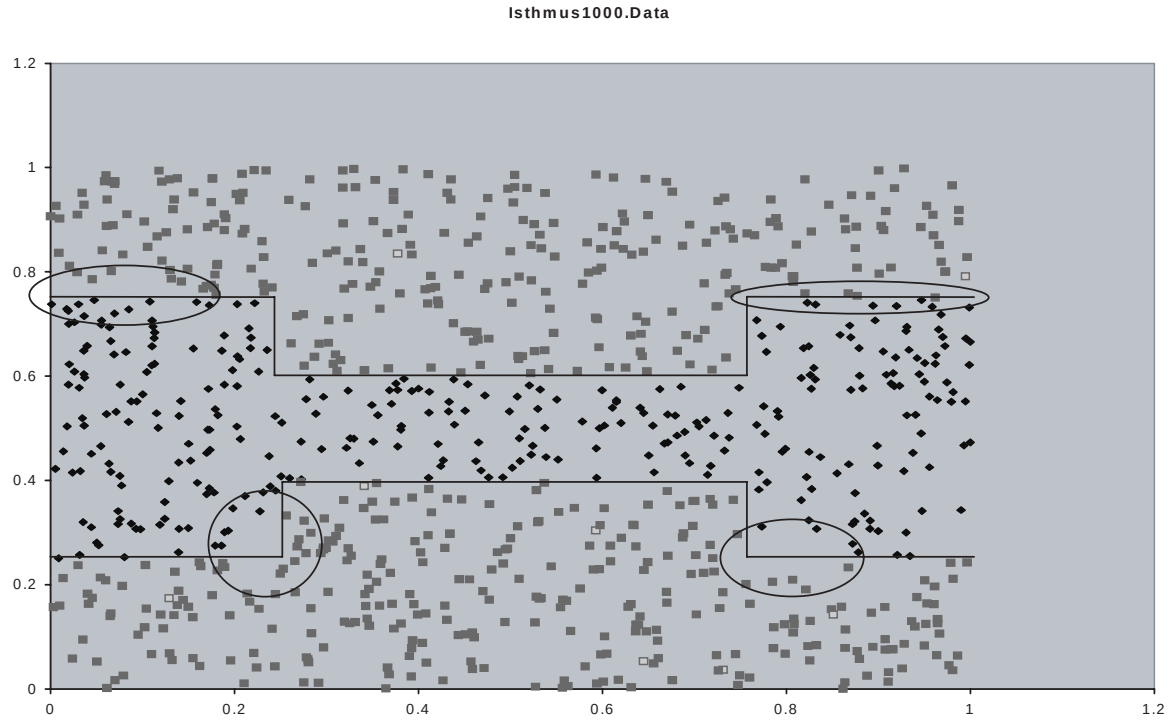

Fig. 2. Isthmus Dataset with the underlying distribution shown.

that region in Fig. 3. It is difficult for any algorithm to properly place a decision surface when there is no data in the region. The same phenomena occurs in the upper left oval while a nice straight-line decision surface is formed in the upper right region where there is sufficient data defining the boundary. The upper right oval highlights the data in that region.

To illustrate the importance of the generalization parameter (G-Parameter) on the size of the resulting solution we ran WITS on multiple instances of the 'Gaussian' and 'Isthmus' problems varying the size of the dataset. In one set of runs the generalization parameter is set as a percentage of the training set size while in the other set of runs the parameter has a fixed value of one. Table 2 shows the resulting solution sizes. With a scalable generalization parameter the solution size stays relatively constant. With a fixed generalization parameter the solution size grows linearly with the training data.

Figure 8 shows the decision boundaries generated by WITS when the generalization parameter is fixed to the value one. The data set contains 1000 instances. Comparing Figs 7 and 8 shows that with the G-Parameter equal to one an overly complex solution is generated (i.e. WITS stores instances representing sampling anomalies). We have not explored further the effect of the G-Parameter on the results of WITS. In all the real world problems presented in the next section a G-Parameter value of 0.005*|TrainData| was used. One would expect the accuracy results to improve if an appropriate G-Parameter value can be determined for each particular problem.

3.3.2. Real world problems

Ten real world problems taken from the Machine Learning Database Repository at the University of California, Irvine [14] were used to test the WITS algorithm. The results of the WITS algorithm on these problems are compared to three other data reduction instance-based algorithms in Table 3. All the results

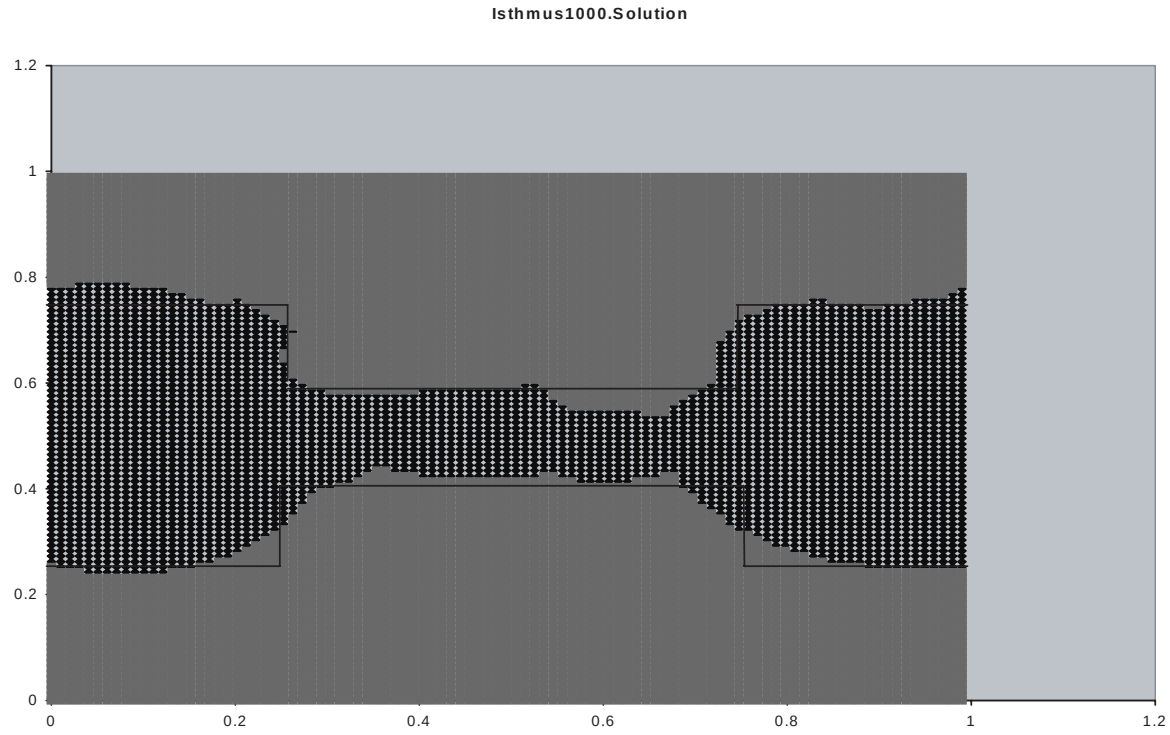

Fig. 3. WITS decision surface for the Isthmus problem.

Table 2
The growth of solution size: Scalable vs. fixed G-Parameter

Gaussian	Data Size 250	Data Size 500	Data Size 1000	Data Size 2000
Scalable G-Parameter	2	3	3	2
Fixed G-Parameter	2	6	9	12
Isthmus				
Scalable G-Parameter	15	12	14	15
Fixed G-Parameter	15	16	25	36

reported are the average of ten trials of ten-fold cross validation. The results for DROP-3, DROP-5 and Explore were taken directly from [29].

DROP-3 and DROP-5 are decremental algorithms the details of which are explained in [29]. The Explore algorithm uses a minimum encoding length heuristic to determine the 'best' minimal solution to the problem. The details of this algorithm are found in [4]. These algorithms are currently three of the best at finding a good compromise between the size and accuracy of the solution. Table 3 lists the accuracy and size of solution of five different algorithms on the chosen ten problems. The kNN column is the naïve nearest neighbor algorithm where the three nearest neighbors determine the classification of the query point. The kNN algorithm stores all of the training data and thus the size column is 100%. The kNN algorithm gives a baseline against which to compare the other algorithms. It is expected, in general, that data reduction algorithms will have poorer average accuracy than the kNN while making gains in storage requirements and classification speed.

The highest accuracy achieved by any algorithm (excluding kNN) is bolded. WITS achieves the highest average accuracy of 83.38% with DROP-5 at a comparable 83.25%. DROP-3 is competitive at

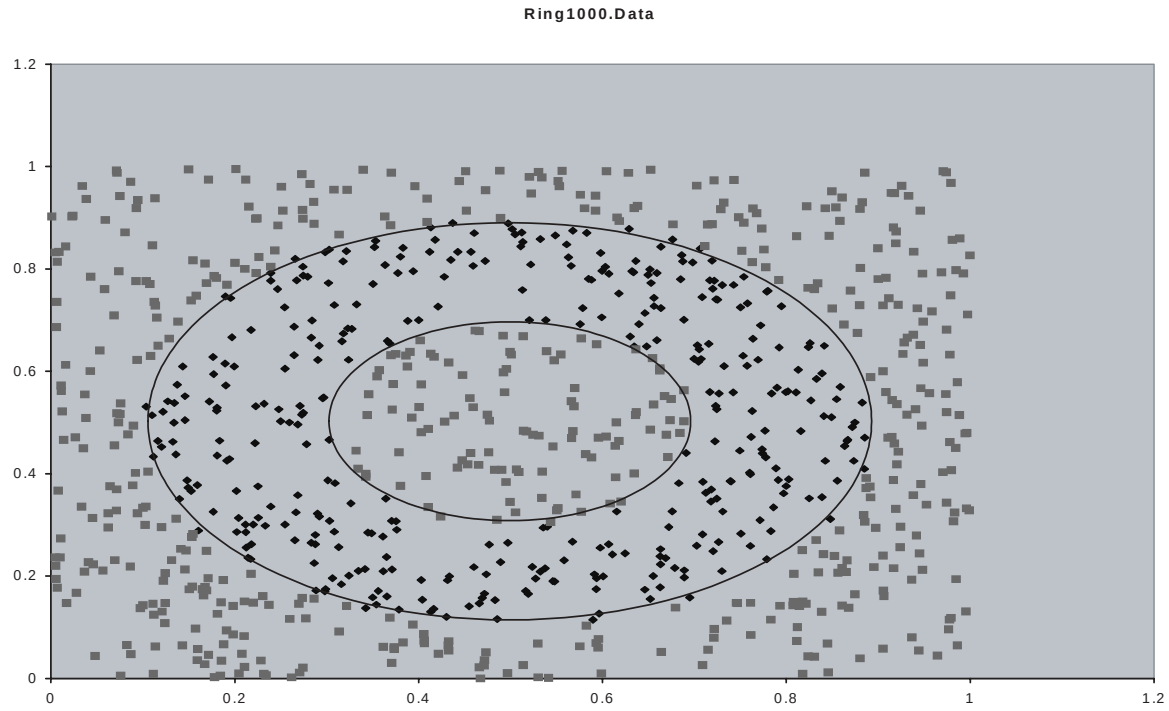

Fig. 4. Ring Dataset with the underlying distribution shown.

Table 3
Dataset accuracy and storage comparison chart

Dataset	kNN	Size %	DROP-3 Accuracy	Size %	DROP-5 Accuracy	Size %	Explore Accuracy	Size %	WITS Accuracy	Size %
Glass	73.83	100	65.02**	23.88	65.45**	24.81	63.98**	3.53	**65.79****	9.03
Ionosphere	84.62	100	87.75	7.06	86.90	9.78	80.89**	0.63	**88.06**	7.33
Iris	94.00	100	**95.33**	14.81	94.00	12.15	92.67	2.3	93.53	4.64
Liver (Bupa)	65.57	100	60.84**	24.99	65.50	31.08	57.65**	0.64	64.93	7.79
Pima (Diabetes)	73.56	100	75.01	16.9	73.05	21.95	**75.27**	0.29	74.48	1.56
Promoters	93.45	100	86.82**	16.67	87.00**	12.58	**91.36**	2.1	90.28**	3.03
Segment	93.10	100	**92.62**	10.98	89.29**	11.35	89.76**	2.43	90.91	1.49
Sonar	87.55	100	78.00**	26.87	**79.88****	29.81	70.29**	1.07	76.92**	7.58
Vote	95.64	100	**95.87**	5.11	95.86	7.13	94.25	0.51	94.83	1.36
Zoo	94.44	100	90.00**	20	**95.56**	17.16	**95.56**	8.4	94.06	7.62
Average	**85.58**	**100**	**82.73**	**16.73**	**83.25**	**17.78**	**81.17**	**2.19**	**83.38**	**5.14**

82.73% while Explore exhibits a significant drop in average accuracy down to 81.17%. The ** marking in Table 3 indicates a catastrophic failure. The criteria for a catastrophic failure is a $>$ 3% drop in accuracy compared to the kNN baseline. WITS shows three catastrophic failures over this set of ten problems while DROP-5 shows four and DROP-3 and Explore each show five.

In terms of size, the Explore algorithm shows the lowest average solution size at 2.19%. WITS stores an average of 5.14% which is $\sim$ 70% less than the storage of DROP-3 (16.73%) and DROP-5 (17.78%), the two algorithms that are competitive in terms of accuracy. Not only is this advantageous in applications where the footprint is critical but it also represents a corresponding improvement in classification speed.

Table 4 summarizes the results just discussed. The table shows the number of highest accuracies, the

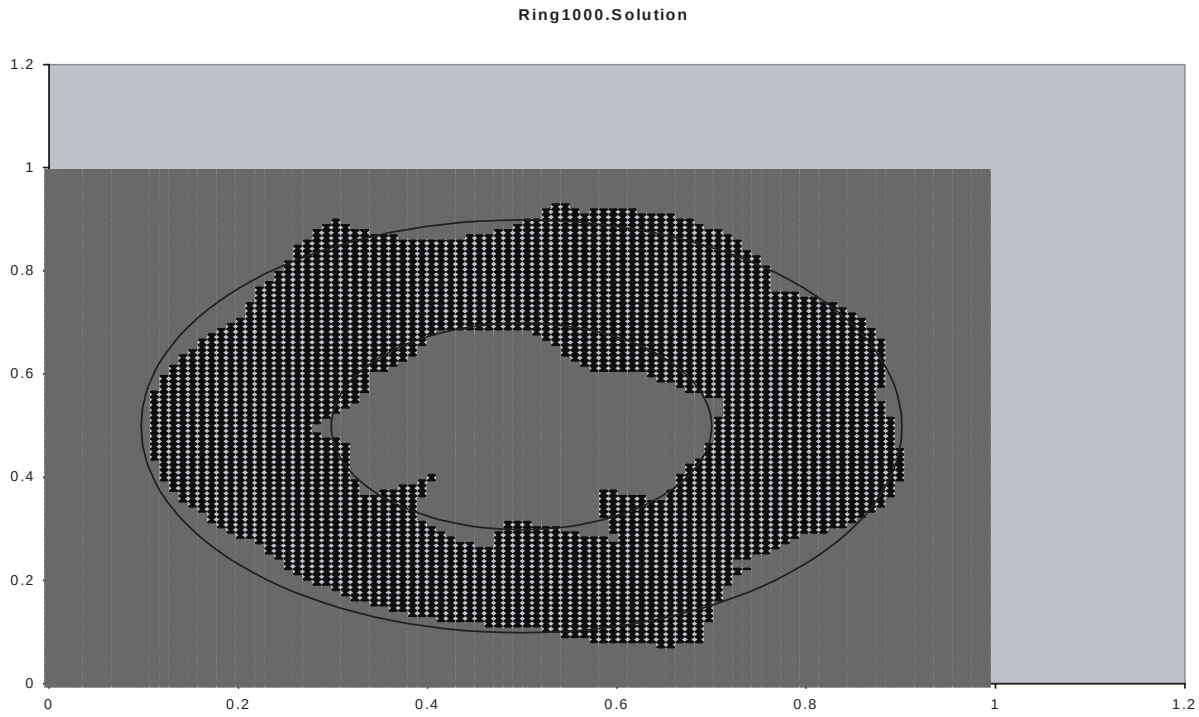

Fig. 5. WITS decision surface for the Ring problem.

Table 4
Summary of results

Algorithms	Average accuracy %	Highest accuracy	Catastrophic failures	Average solution size (%)
WITS	83.38	2	3	5.14
DROP-5	83.25	3	4	17.78
DROP-3	82.73	3	5	16.73
Explore	81.17	3	5	2.19

number of catastrophic failures and the average solution size (as a percentage of training set size) of the four algorithms. WITS shows a nice balance in these areas. The fact that it performs the best, in terms of accuracy, on some of the problems indicates that its inductive bias has the capability to outperform others within certain domains. The fact that WITS experiences fewer catastrophic failures on the problems examined is significant. Catastrophic failures indicate a serious weakness of the algorithm in certain domains. While the Explore algorithm shows the best average performance with regards to solution size, note that WITS makes significant improvements over the DROP algorithms, in this area, with fewer catastrophic failures. A closer look at the results reveals that the two catastrophic failures of the Explore algorithm, on the Ionosphere and Liver problems, coincide with the greatest discrepancies between the solution size of Explore and the WITS algorithms. The WITS solutions on these two problems contains more than ten times the instances that Explore stores. We see here a case where the Explore algorithm traded accuracy for a smaller solution size and WITS did not. The same thing can be seen on the Sonar problem also.

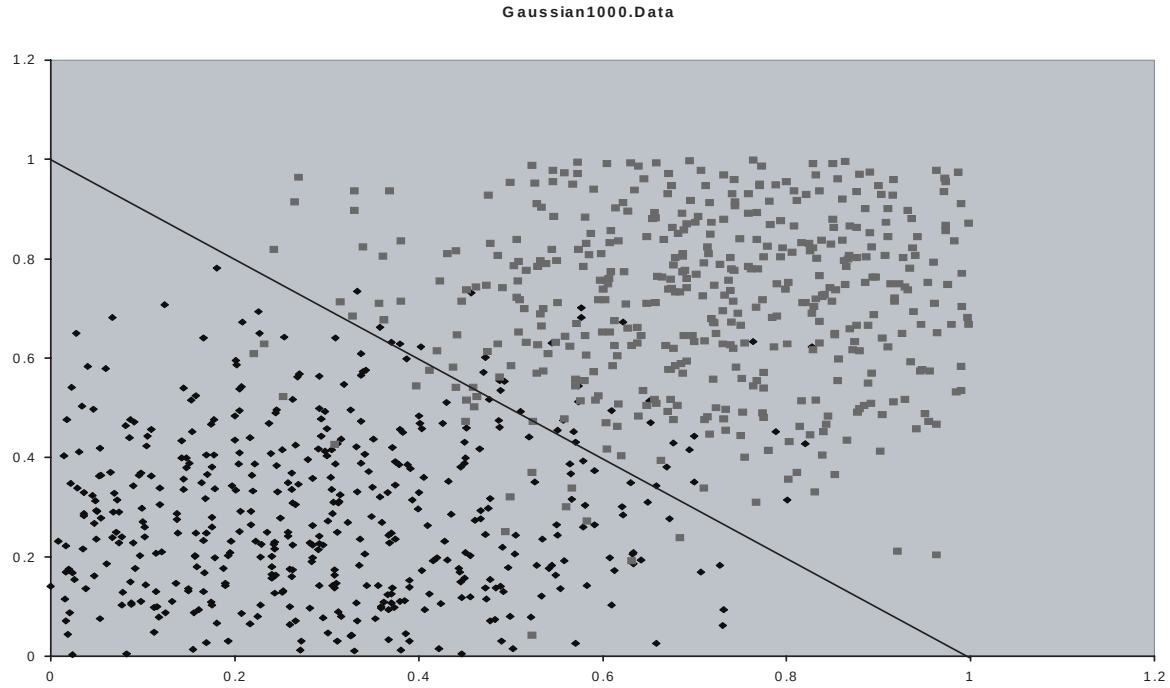

Fig. 6. Gaussian Dataset with the Bisector shown.

3.4. C-WITS

All the algorithms compared up to this point are $O(n^2)$. Although $O(n^2)$ is generally considered acceptable, especially during the learning phase of an algorithm as opposed to the execution phase, we are presented with an interesting situation. Data reduction algorithms are specifically designed to be useful on very large problems but it is exactly on very large problems that $O(n^2)$ becomes less and less acceptable. For this reason, we now present Clustered WITS (C-WITS), a modified version of WITS.

There are two parts of the WITS algorithm that make it $O(n^2)$; the 'InitializeTypicality' function that calculates the typicality values and the main 'while' loop that adds candidate instances to the solution and tests for generalization improvement. The 'InitializeTypicality' function is truly $O(n^2)$ because the distance from a given instance to all the others must be calculated in order to calculate that instances' typicality. The main 'while' loop of the algorithm is actually $O(ns)$ where s is (1 – Percent Accuracy) of the problem at hand. This is a result of marking correctly classified instances each time a new instance is added to the solution and then excluding the marked instances from possible inclusion within the solution.

Another aspect of the WITS algorithm that increases the computational load is the naïve approach to setting the weight values of the candidate instances. The naïve approach is to try a predetermined set of values and simply choose the value that shows the best generalization performance. This approach adds a large constant to the complexity term of the main loop. All the problems presented in this paper use a set of 30 distinct values and thus the computational complexity of the main loop of each problem is 30 * ns.

The C-WITS algorithm contains a user-defined parameter, called the Cluster-parameter (C-parameter), which allows the user to determine how much computation is performed in the 'InitializeTypicality'

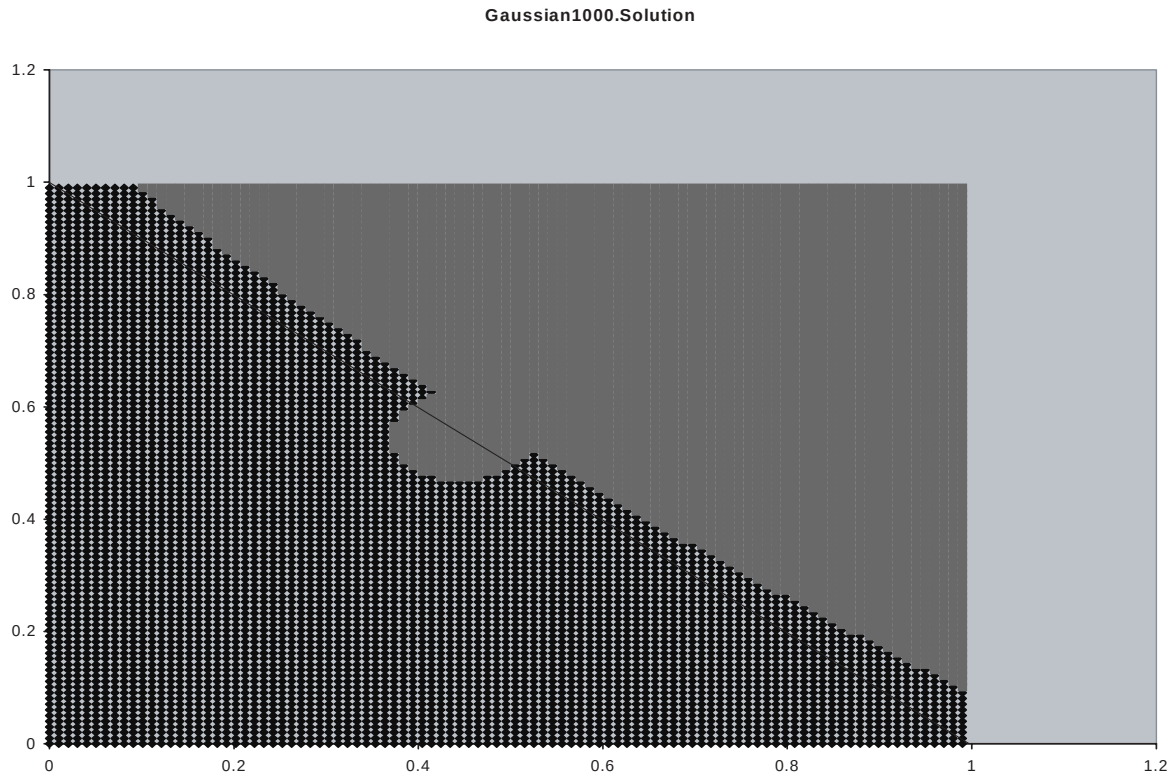

Fig. 7. WITS decision surface for the Gaussian problem.

function and potentially reduces the computation performed within the main 'while' loop. The essence of C-WITS is to change the 'InitializeTypicality' function from a deterministic function that calculates the exact typicality of each instance to a stochastic function that approximates most of the typicality values. C-WITS works as follows. A percentage of instances (seed instances), specified by the C-parameter, are randomly chosen from the training set and have their exact typicality values calculated. Next, the algorithm loops through the remaining instances and sets their typicality values to be equal to the value of the nearest neighbor of the exactly calculated instances (seed instances). The result is that each 'seed instance' is surrounded by a cluster of instances with identical typicality values. There is no further ordering performed to distinguish between instances of a given cluster, nevertheless, the various clusters of the training set are ordered according to their typicality values. This procedure allows the user to reduce the computational load of the 'InitializeTypicality' step by an arbitrary constant.

As presented here, the modified 'Initialize Typicality' of C-WITS is still $O(n^2)$ with the computation being reduced by a user-defined constant, C. The 'Initialize Typicality' step could be made to be sub-$O(n^2)$ if the parameter C is set as a sub-linear function of n, such as $\log(n)$. C was set as a linear function of n (a simple percentage) in this paper for pedagogical purposes.

The main loop of the algorithm continues to work in the same way except that only one instance from any given cluster (instances associated with the same seed instance) can be permanently added to the solution S. Once an instance is found to meet the generalization criteria, and subsequently added to the solution S, all the other instances of the same cluster are marked and are no longer solution candidates. This adjustment provides another potential speedup within the main loop, which is already $O(ns)$. The

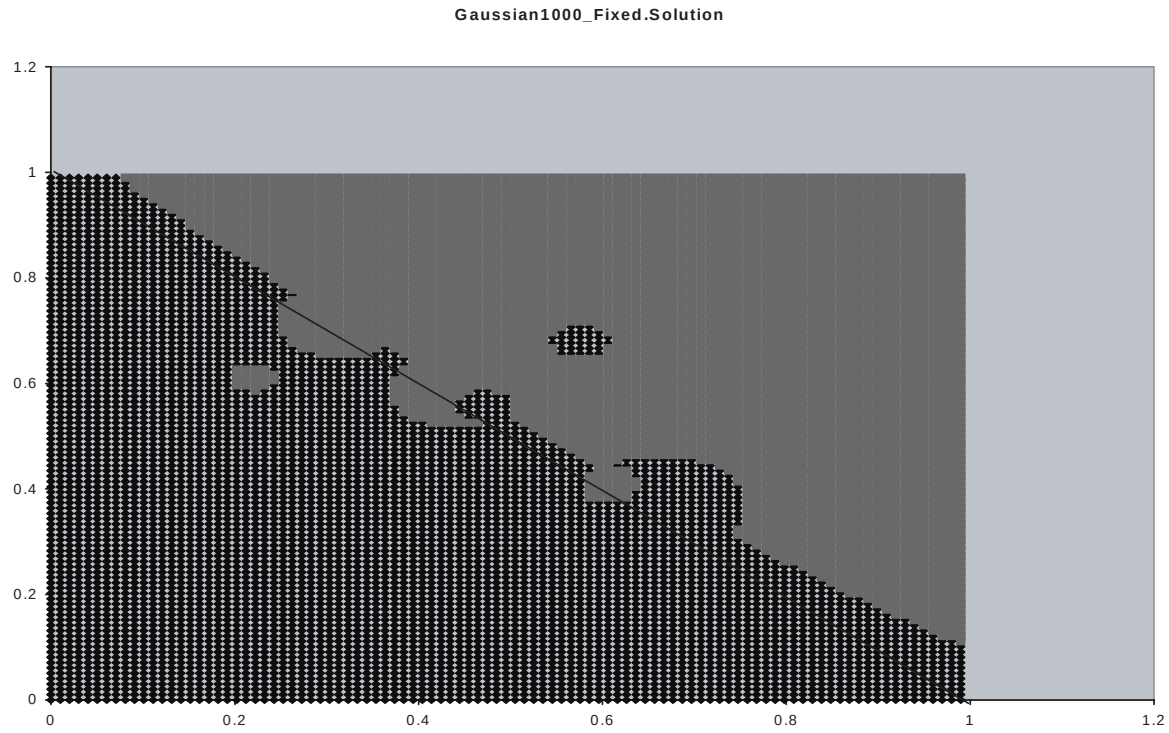

Fig. 8. WITS decision surface for the Gauss problem with parameter G = 1.

Table 5
Results of C-WITS with varied computational loads

Dataset	C = 0.1	Size %	C = 0.25	Size %	C = 0.5	Size %	C = 0.75	Size %	C = 0.9	Size %	C = 1.0 WITS	Size %
Glass	61.26	6.03	65.75	8.14	66.31	9.28	64.72	9.22	**66.64**	9.21	65.79	9.03
Ionosphere	81.54	3.67	84.99	5.66	88.03	6.76	87.86	7.23	88.03	7.23	**88.06**	7.33
Iris	90.87	4.22	93.13	4.67	**93.6**	4.81	93.33	4.87	93.33	4.72	93.53	4.64
Liver (Bupa)	60.93	5.69	60.38	7.49	62.49	8.04	63.57	8.09	63.51	7.86	**64.93**	7.79
Pima (Diabetes)	73.07	1.66	**75.13**	1.66	74.58	1.53	73.54	1.59	73.83	1.57	74.48	1.56
Promoters	89.62	3.07	89.53	3.16	90.57	2.9	90.09	3.03	**90.94**	3.05	90.28	3.03
Segment	90.09	1.6	90.39	1.51	90.56	1.53	90.26	1.56	90.52	1.55	**90.91**	1.49
Sonar	75.91	5.81	77.88	7.33	**79.13**	8.01	78.46	7.83	77.74	7.82	76.92	7.58
Vote	94.92	1.24	94.92	1.33	94.85	1.34	**95.31**	1.37	94.62	1.35	94.83	1.36
Zoo	92.87	7.59	93.27	7.69	93.47	7.6	93.96	7.66	93.96	7.65	**94.06**	7.62
Average	**81.11**	**4.06**	**82.58**	**4.86**	**83.36**	**5.18**	**83.11**	**5.25**	**83.31**	**5.20**	**83.38**	**5.14**

C-WITS algorithm does not change the naïve weight setting routine and thus does not mitigate the large constant introduced by that routine.

Table 5 shows the results of the C-WITS algorithm on the ten real-world problems. The columns are labeled according to the value of the C-parameter, which is the percentage of the training data used as 'seed instances'. The highest accuracies are bolded. It is interesting to see that using the exact typicality values for all of the training data, in other words, using the standard WITS algorithm did not always produce the highest accuracy. The stochastic nature of the C-WITS algorithm explains this phenomenon. The average for each percentage level is reported on the last row and shows the general trend of accuracy

Table 6
Results of C-WITS sorted by problem size

Dataset	C = 0.1	Size %	C = 0.25	Size %	C = 0.5	Size %	C = 0.75	Size %	C = 0.9	Size %	C = 1.0 WITS	Size %
Zoo	92.87	7.59	93.27	7.69	**93.47**	7.6	93.96	7.66	93.96	7.65	94.06	7.62
Promoters	89.62	3.07	89.53	3.16	90.57	2.9	90.09	3.03	90.94	3.05	90.28	3.03
Iris	**90.87**	4.22	93.13	4.67	93.6	4.81	93.33	4.87	93.33	4.72	93.53	4.64
Sonar	**75.91**	**5.81**	77.88	7.33	79.13	8.01	78.46	7.83	77.74	7.82	76.92	**7.58**
Glass	**61.26**	**6.03**	65.75	8.14	66.31	9.28	64.72	9.22	66.64	9.21	65.79	**9.03**
Liver (Bupa)	60.93	**5.69**	60.38	7.49	**62.49**	8.04	63.57	8.09	63.51	7.86	64.93	**7.79**
Ionosphere	**81.54**	**3.67**	84.99	5.66	88.03	6.76	87.86	7.23	88.03	7.23	88.06	**7.33**
Vote	94.92	1.24	94.92	1.33	94.85	1.34	95.31	1.37	94.62	1.35	94.83	1.36
Pima (Diabetes)	73.07	1.66	75.13	1.66	74.58	1.53	73.54	1.59	73.83	1.57	74.48	1.56
Segment	90.09	1.6	90.39	1.51	90.56	1.53	90.26	1.56	90.52	1.55	90.91	1.49
Average	**81.11**	**4.06**	**82.58**	**4.86**	**83.36**	**5.18**	**83.11**	**5.25**	**83.31**	**5.20**	**83.38**	**5.14**

Table 7
Results of C-WITS on the PenDigit dataset

PenDigit	C = 0.01	Size %	C = 0.05	Size %	C = 0.1	Size %	C = 0.25	Size %
Accuracy	91.45	0.61	95.9	0.85	96.82	0.95	97.91	0.88
Training time	5.1 hrs		10.4 hrs		12.6 hrs		13 hrs	

Table 8
Results of C-WITS on the Letter dataset

Letter	C = 0.005	Size %	C = 0.01	Size %	C = 0.02	Size %	C = 0.05	Size %
Accuracy	49.75	0.43	57.15	0.89	71.95	1.47	79.5	1.95
Training time	8.3 hrs		19.3 hrs		74.2 hrs		175.8 hrs	

degradation as greater amounts of computation are sacrificed. Within this trend we also see the stochastic effects in that average accuracies for the 0.75 and 0.9 columns are lower than the 0.5 column. It is also worth noting here that, on average, the solution sizes do not drastically change.

The same results are shown in Table 6 but the problems have been sorted according to problem size (training set size) from the smallest, 'Zoo', to the largest, 'Segment'. The onset of significant accuracy degradation is indicated by bold characters instead of the highest accuracy as in Table 5. On only two of the problems does degradation occur before the 10 percent level is reached. On the three largest problems, Vote (435 instances), Pima (768 instances) and Segment (2310 instances) accuracy degradation does not occur even at the 10 percent level. This supports the thesis that minimal solutions to large problems, exactly the kinds of problems that data reduction algorithms are designed for, can be found using an efficient algorithm.

Even though the average solution size does not undergo a drastic change, it does on specific problems. The solution size has been bolded on the four problems that experienced a major change. In all four cases the drop in solution size corresponds to a significant drop in accuracy.

To further explore the capabilities of the C-WITS algorithm it was tested on two larger datasets; the PenDigit dataset with 10,992 instances and the Letter dataset with 20,000 instances. Because of the size of the datasets only a single training set and test set were used in this series of experiments instead of a full ten-fold cross validation. According to the Machine Learning Database Repository the best accuracies reported for these problems are 97.87% for the PenDigit problem and 80% for the Letter Problem. Tables 7 and 8 show the performance of C-WITS on these two datasets along with the solution size and training time. C-WITS was able to match the best accuracies reported for these problems while

retaining only a very small fraction of the instances, 0.88% and 1.95% respectively. These two examples indicate that the percentage of the dataset needed as seed instances during training decreases as the dataset size grows larger. The solution size continues to decrease with the increase of dataset size. This coincides with the idea that the problem at hand has an inherent complexity that is independent from the (arbitrary) amount a data collected. The solution size of WITS generated solutions corresponds roughly to the inherent complexity of the problem and not to the dataset size.

4. Conclusion and future work

The WITS (Weighted-Instance Typicality Search) algorithm is an incremental instance-based algorithm. Added degrees of freedom are introduced through Instance-Weighting, which increases the representational power of a fixed set of instances. The standard measure of Typicality is used to guide the search for instances to be included in the solution. Candidate instances must pass a Generalization Test, indicating that they are not noise, before they are added to the final solution. WITS seeks to find minimal instance-based solutions to complex real world problems. A minimal solution is useful in applications where a small footprint is needed and also drastically reduces classification time over the standard nearest neighbor algorithm.

WITS has been shown to compare very favorably with other state of the art instance-based data reduction algorithms. Across ten real world problems WITS achieved the highest average generalization accuracy, showed fewer catastrophic failures, and stored an average of 71% fewer instances than DROP-5, the next most competitive algorithm in terms of accuracy and catastrophic failures.

Being $O(n^2)$ algorithms, WITS and the DROP algorithms are slow when trained on very large problems. Clustered WITS, or C-WITS, is a version of WITS that gives the user active control over the train time vs. accuracy tradeoff while maintaining the high accuracy/small solution size advantages of WITS.

Future work under consideration for the WITS algorithm includes an improved method for setting the instance weight values and an intelligent choice of the generalization parameter G. An improved method for setting the instance weight values would have little effect on the accuracy results of the algorithm but could lead to a significant computational speedup. The choice of the G-parameter directly affects the creation of the decision surface and thus has the possibility of improving accuracy results. The results presented in this paper indicate acceptable performance even when G is set rather arbitrarily (a simple manual process of trial and error to determine a value that works reasonably well across many problems). With a more intelligent choice of G, accuracy on any given problem could improve. Another direction to explore is a process where G starts out large during the early phases of learning, when more general rules are being learned, and is subsequently decreased as learning proceeds in order to capture rules of less general scope.

Another variant of WITS is Gaussian Approximator Typicality Search (GATS). GATS uses a gaussian, centered on the instance, with a variable variance to govern the instances' influence in the space. It is easy to see that the decision surfaces created by the gaussians are hyperspheres, just as with the weighted instances in WITS. By using gaussians, a close relationship between GATS, Reduced Coulomb Energy Network [18] and Radial Basis Function networks [16] is manifest. In turn, this points out the close relationship between WITS and the before mentioned algorithms. These methods all have comparable representational power by virtue of the hyperspherical surfaces but they differ in the algorithms themselves. By switching to the gaussian, GATS does not deal directly with the decision surface, as WITS does, but tries to model the underlying probability distribution. The decision surface

is then derived at run time from the relative strengths of the gaussians. Though initial experiments with GATS did not produce accuracies comparable to WITS, it remains an option for the future.

References

[1] D.W. Aha, D. Kibler and M.K. Albert, Instance-Based Learning Algorithms, *Machine Learning* **6** (1991), 37–66.

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

[3] M. Cameron-Jones, Minimum Description Length Instance-Based Learning, *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence*, 1992, pp. 368–373.

[4] M. Cameron-Jones, Instance Selection by Encoding Length Heuristic with Random Mutation Hill Climbing, *Proceedings of the Eighth Australian Joint Conference on Artificial Intelligence*, 1995, pp. 293–301.

[5] S. Cost and S. Salzberg, A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features, *Machine Learning* **10** (1993), 57–78.

[6] T.M. Cover and P.E. Hart, Nearest Neighbor Pattern Classification, *IEEE Transactions on Information Theory* **13**(1) (January 1967), 21–27.

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

[8] P. Datta and D. Kibler, Learning Symbolic Prototypes, *Proceedings of the Fourteenth International Conference on Machine Learning*, 1997, pp. 75–82.

[9] P. Datta and D. Kibler, Symbolic Nearest Mean Classifier, *Proceedings of the Fourteenth National Conference of Artificial Intelligence*, 1997, pp. 82–87.

[10] K. Deng and A.W. Moore, Multiresolution Instance-Based Learning, *Proceedings of the International Joint Conference on Artificial Intelligence* (*IJCAI'95*), 1995.

[11] P. Domingos, Rule Induction and Instance-Based Learning: A Unified Approach, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (*IJCAI-95*), 1995, pp. 1226–1232.

[12] P. Domingos, Unifying Instance-Based and Rule-Based Induction, *Machine Learning* **24** (1996), 141–168.

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

[14] C.J. Merz and P.M. Murphy, *UCIRepository 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, 1996.

[15] T.R. Payne and P. Edwards, Implicit Feature Selection with the Value Difference Metric, *Proceedings of the 13th European Conference on Artificial Intelligence*, 1998, pp. 450–454.

[16] M.J.D. Powell, Radial Basis Functions for multivariable interpolation: A Review, in: *Algorithms for approximation*, J. Mason and M. Cox, eds, Oxford: Clarendon Press, 1987, pp. 143–167.

[17] J. Rachlin, S. Kasif, S. Salzberg and D.W. Aha, Towards a Better Understanding of Memory-Based Reasoning Systems, *Proceedings of the Eleventh International Conference on Machine Learning*, 1994, pp. 242–250.

[18] D.L. Reilly, L.N. Cooper and C. Elbaum, A Neural Model for Category Learning, *Biological Cybernetics* **45** (1982), 35–41.

[19] D.B. Skalak, Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms, *Proceedings of the Eleventh International Conference on Machine Learning*, 1994, pp. 293–301.

[20] R.F. Sproull, Refinements to Nearest-Neighbor Searching in k-Dimensional Trees, *Algorithmica* **6** (1991), 579–589.

[21] C. Stanfill and D. Waltz, Toward Memory-Based Reasoning, *Communications of the ACM* **29** (December 1986), 1213–1228.

[22] K.M. Ting, The Problem of Atypicality in Instance-Based Learning, *Proceedings of the Third Pacific Rim International Conference on Artificial Intelligence*, Beijing, 1994, pp. 360–366.

[23] D. Wettschereck, D.W. Aha and T. Mohri, A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms, *Technical Report AIC-95-012*, Washington, DC: Naval Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, 1997.

[24] D. Wettschereck and T.G. Dietterich, An Experimental Comparison of Nearest-Neighbor and Nearest-Hyperrectangle Algorithms, *Machine Learning* **19**(1) (1995), 5–28.

[25] R.D. Wilson and T.R. Martinez, Instance-Based Learning with Genetically Derived Attribute Weights, *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks* (*AIE'96*), 1996, pp. 11–14.

[26] R.D. Wilson and T.R. Martinez, Value Difference Metrics for Continuously Valued Attributes, *Proceedings of the International Conference on Artificial Intelligence, Expert Systems, and Neural Networks* (*AIE'96*), August 1996, pp. 74–78.

[27] R.D. Wilson and T.R. Martinez, Instance Pruning Techniques, *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997, pp. 404–411.

[28] R.D. Wilson and T.R. Martinez, Improved Heterogeneous Distance Functions, *Journal of Artificial Intelligence Research* (*JAIR*) **6**(1) (1997), 1–34.
[29] R.D. Wilson and T.R. Martinez, Reduction Techniques for Instance-Based Learning Algorithms, *Machine Learning* **38**(3) (2000), 257–286.
[30] J. Zhang, Selecting Typical Instances in Instance-Based Learning, *Proceedings of the Ninth International Conference on Machine Learning*, 1992, pp. 470–479.