The Multiscale Classifier

Brian C. Lovell
lovell@elec.uq.oz.au

Andrew P. Bradley
bradley@elec.uq.oz.au

Cooperative Research Centre for Sensor Signal and Information Processing
and the Dept. of Electrical and Computer Engineering,
University of Queensland 4072
Australia

April 24, 1995

Abstract

In this paper we propose a *rule-based* inductive learning algorithm called *Multiscale Classification* (MSC). It can be applied to any N-dimensional real or binary classification problem to classify the training data by successively splitting the feature space in half. The algorithm has several significant differences from existing rule-based approaches: learning is incremental, the tree is non-binary, and backtracking of decisions is possible to some extent.
The paper first provides background on current machine learning techniques and outlines some of their strengths and weaknesses. It then describes the MSC algorithm and compares it to other inductive learning algorithms with particular reference to ID3, C4.5, and back-propagation neural networks. Its performance on a number of standard benchmark problems is then discussed and related to standard learning issues such as generalization, representational power, and over-specialization.
Index Terms — Multiscale Classification, decision tree, inductive machine learning, tree pruning.

1 Introduction

In this paper we introduce a new machine learning strategy which we call Multiscale Classification (MSC). The paper describes the Multiscale Classifier algorithm and compares it to other decision tree classifiers and neural networks to highlight similarities and differences. It then demonstrates the ability of the algorithm to generalize to unseen data by comparing the performance of the proposed algorithm with these other methods on a few benchmark problems.

1.1 Background

Although there are many techniques for machine learning, most techniques fall into one of the following three classes: statistical classifiers, parameter adjustment systems, or signature table systems [10]. Statistical classifiers such as the linear and quadratic discriminant functions [9] have been studied for many years. Their major disadvantage is that they rely on simplifying assumptions about the probability distributions and decision surfaces which may not be valid [12]. They also perform poorly when presented with nonlinear relationships.

The remaining non-parametric techniques can cope with nonlinear relationships and may be classed on the basis of how they store their knowledge. Parameter adjustment systems such as nearest neighbour classifiers [11,12], the Perceptron [24], and the back-propagation neural network [16] adjust the parameters or coefficients of a discriminant function until it is optimal, or at least satisfactory, according to predefined criteria. Signature table systems such as Michie's Boxes [17] and Samuel's checker-playing program [26] divide each input variable into a number of levels using threshold points. The output class is then obtained by table lookup. Thus a parameter adjustment approach attempts to determine a function (hyper-surface) which defines the boundary between two classes, whereas the signature table method produces a jagged decision line which resembles a staircase.

Quinlan's ID3, its successor C4.5 and CART [4,20,22,23] are a significant step forward from signature table methods. Instead of creating a decision table, it creates a decision tree. The decision tree can be thought of as a compressed signature table. ID3 works by first finding the variable (or test) which is most discriminatory, and then partitioning the data with respect to that variable. Having divided the data into two subsets on the basis of the most discriminatory variable, each subset is recursively partitioned in a similar way until it contains examples of only one class. The result is a binary decision tree which can be used to classify new examples.

The proposed Multiscale algorithm uses a decision tree to store the knowledge it learns, but borrows ideas from logic minimization, arithmetic coding, and the binary search algorithm to produce a fundamentally different approach to learning.

2 The MSC Algorithm

All N inputs are considered to be real numbers in the range [0,1). There is no loss of generality in this step, since all physical quantities must have some upper and lower bounds on their range, and so suitable linear or non-linear transformations to the interval [0,1) can always be found. This means that the entire feature space is mapped to the inside of a unit hypercube.

The inputs being in the range [0,1) means that these real numbers can be expressed as binary fractions. This is convenient because each successive bit position corresponds to a successive halving of the feature space. In other words, the most significant bit indicates if an input is greater or less than 0.5; the second most significant bit increases this resolution to 0.25, the third to 0.125, and so on. By performing the classification one bit position at a time, the algorithm uses finer and finer levels of resolution to determine the eventual classification — hence the name Multiscale.[1] As classification occurs one bit position at a time, even non-terminating numbers such as 0.6, which in binary is $0.1\overline{0011}$ recurring, are handled with ease as they are converted to the minimum number of bits required for successful classification.

MSC is based on a tree architecture. Each node of the tree may have up to 2^N branches or leaves — thus a two input system may take the form of a quadtree [25]. Normally there will be far fewer than 2^N branches or leaves at each level of the tree, as rules can have "don't care" terms which cover large regions of feature space. Here "don't care" terms indicate irrelevant inputs which may be exploited in the same manner as occurs in logic minimisation [29]. Classification is performed by simultaneously examining the most significant bit of each of the N inputs. This either yields the output class directly (a leaf of the tree), or tells us that we must examine the next most significant bit (descend down a branch of the tree) to determine the output class. The next bit then either yields the output class, or tells us to examine the following bit, and so on. In this way the training data is classified using the minimum level of input resolution required to separate the output classes by examining only one bit of each input at each level of the tree. For example, to separate the inputs 0.8 and 0.3 only the most significant bit is required, but to separate the inputs 0.6 and 0.601 several levels of resolution will be required.

Before data can be classified the decision tree must first be constructed. This is done in the training phase which is similar to classification, but allows for the incremental modification of the decision tree to accommodate new data. The tree consists of decision rules, branches to rules, and counters which store rule usage information.

If, when a leaf of the tree is reached, the rule class is different from the class of the current training point, the corresponding leaf is either:

1. repartitioned at the current level to accommodate the new training point (see Section 2.2). The rule usage is incremented;
2. split in half, exploiting "don't care" relationships, so that the current point is classified correctly. The usage of this new rule is then set to 1;
3. if all the "don't cares" have been used, the leaf is converted to a branch to the next lower level in the tree. This new universal (all "don't cares") leaf is given the class of the current point and its usage is set to 1. The previous point which generated the leaf at the higher

[1] The concept of treating a data stream as a large binary fraction is also fundamental to arithmetic coding compaction algorithms [2].

level is now forgotten until the next pass through the training data when its position at this new lower level can be determined;

4. or, if all the "don't cares" have been used **and** this is the maximum level of the tree allowed, defined by the required resolution, no new leaf is created, but the count of the number of errors this leaf has made is incremented. The error and rule usage counts can then be used as part of a tree pruning strategy (see section 2.3).

```
WHILE NOT class_found DO
    IF rule fits input data THEN
        IF rule is a leaf THEN
            /* Learning Phase */
            IF learning THEN
                IF rule class equals input class THEN
                    class_found := TRUE
                    increment rule_usage
                ELSE IF maximum tree level
                    increment other_usage
                    class_found := TRUE
                ELSE
                    split current rule
                ENDIF
            /* Classification Phase */
            ELSE
                class_found := TRUE
                return input class
            ENDIF
        ELSE
            go to next level down the tree
        ENDIF
    ELSE
        go to next rule at this level
    ENDIF
ENDWHILE
```

Figure 1: Pseudo-code for the basic Multiscale algorithm.

The pseudo-code in Figure 1 demonstrates this process, while the evolution of the classification tree as the algorithm learns is illustrated by an example in two dimensions (two inputs) shown in Figures 2 through 6.

2.1 A Simple Classification Problem

Figure 7 shows the training data points for a simple two class classification problem. The locations of these points in the unit square are used to train MSC giving the decision boundaries as shown in Figure 8. This problem took three passes through the training set (epochs) to train and required three levels of resolution, meaning that the training points could be fully separated by looking at the first three bits of the inputs.

The rules inductively learnt for the solution of this problem are illustrated as a decision tree in Figure 9 and the resultant rule file is shown in Figure 10. The rule file shows "don't care" terms with either an 'x' or an 'X' for the cases when the rule data bits (the training point learnt) are 0 and 1 respectively. The *confidence* (*i.e.,* rule usage) numbers correspond to the number of training points that the rule has correctly classified and the *prior confidence* levels correspond to the number correctly classified before conversion to a branch. They give a measure of a rule's importance. These confidence numbers should be viewed relative to the number of training points the classifier has seen. In this case the classifier has seen 84 training points, which is three passes through the 28 point training set.

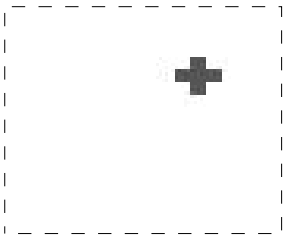

Figure 2: The first training point is learnt as one universal rule

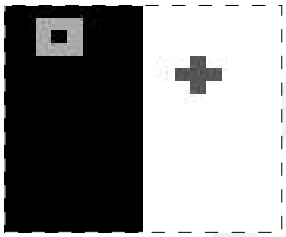

Figure 3: The second point is learnt, the feature space is split in half

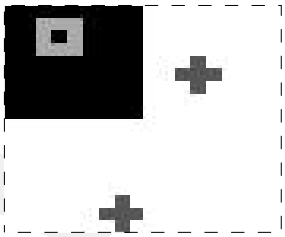

Figure 4: The third training point leads to a further rule split

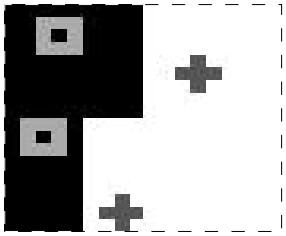

Figure 5: The fourth point requires going to the next level of resolution

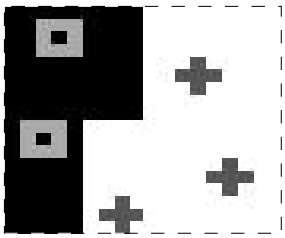

Figure 6: The training data matches the rule, so no rules need changing

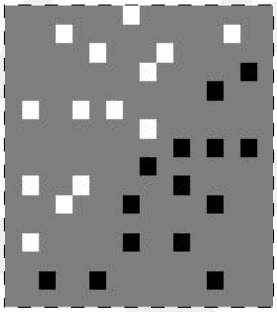

Figure 7: A simple two class classification problem.

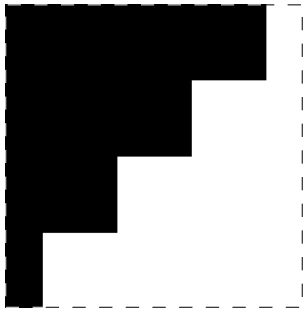

Figure 8: The classification rules learnt on this simple problem.

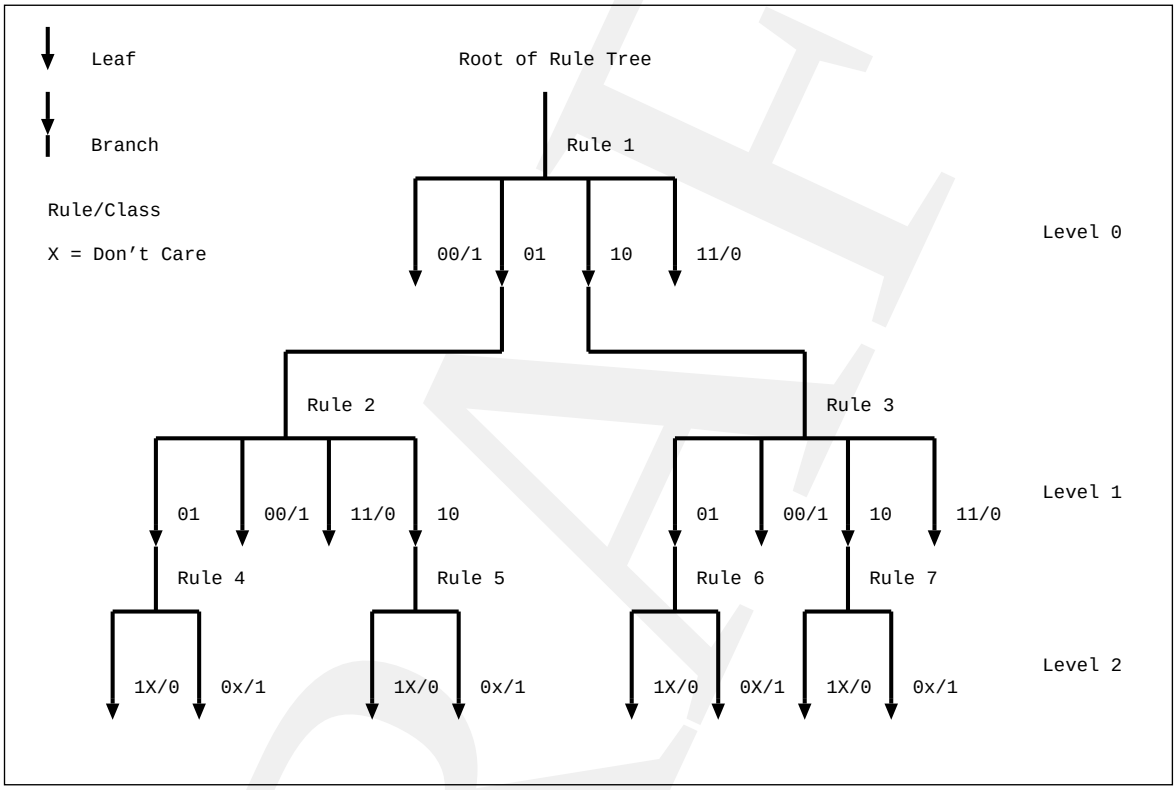

Figure 9: Diagram of the rule tree learnt for the simple classification problem.

```
RULE 1 DATA = 00 CLASS = 1 WITH CONFIDENCE 21;
       DATA = 01 BRANCH TO RULE 2 PRIOR CONFIDENCE 1;
       DATA = 10 BRANCH TO RULE 3 PRIOR CONFIDENCE 2;
       DATA = 11 CLASS = 0 WITH CONFIDENCE 18;
       ( Level 0 )
RULE 2 DATA = 01 BRANCH TO RULE 4 PRIOR CONFIDENCE 1;
       DATA = 00 CLASS = 1 WITH CONFIDENCE 4;
       DATA = 11 CLASS = 0 WITH CONFIDENCE 6;
       DATA = 10 BRANCH TO RULE 5 PRIOR CONFIDENCE 1;
       ( Level 1 )
RULE 3 DATA = 01 BRANCH TO RULE 6 PRIOR CONFIDENCE 1;
       DATA = 00 CLASS = 1 WITH CONFIDENCE 4;
       DATA = 10 BRANCH TO RULE 7 PRIOR CONFIDENCE 1;
       DATA = 11 CLASS = 0 WITH CONFIDENCE 8;
       ( Level 1 )
RULE 4 DATA = 1X CLASS = 0 WITH CONFIDENCE 3;
       DATA = 0x CLASS = 1 WITH CONFIDENCE 2;
       ( Level 2 )
RULE 5 DATA = 1X CLASS = 0 WITH CONFIDENCE 2;
       DATA = 0x CLASS = 1 WITH CONFIDENCE 1;
       ( Level 2 )
RULE 6 DATA = 1X CLASS = 0 WITH CONFIDENCE 3;
       DATA = 0X CLASS = 1 WITH CONFIDENCE 2;
       ( Level 2 )
RULE 7 DATA = 1X CLASS = 0 WITH CONFIDENCE 2;
       DATA = 0x CLASS = 1 WITH CONFIDENCE 1;
       ( Level 2 )

   TOTAL NUMBER OF TRAINING POINTS = 84
```

Figure 10: An example rule file generated for the simple problem.

2.2 Degrees of Freedom Rotations

When the feature space needs to be split in half to accommodate new data points there are often a number of possible splits that can be chosen. A split is possible wherever there is a bit difference between the example which generated the current rule and the new example at that level. An illustration of this problem is shown in Figures 11 through 13. In this example, for the second point to be learnt, the feature space can either be split in the horizontal direction or the vertical direction as shown in Figures 11 and 12. In this case a split is initially chosen in the vertical direction but both possible splits are marked with a "degree of freedom." This means that the selected split is flagged as temporary since the split could have been made on another variable. It is not until the third point arrives that this split is found to be inappropriate so the decision boundary is "rotated" and a permanent split is then made in the horizontal direction, as shown in Figure 13. A permanent split is one that does not have a degree of freedom associated with it and so cannot be removed at some future point. Although this example was performed in two dimensions for clarity, the problem of being presented with several choices for splits arises much more frequently when the number of dimensions (inputs) is large.

Rotations using degrees of freedom allows good splits to be chosen, reducing the number of rule leaves in the decision tree and helping the classifier to home in on the *relevant* input dimensions.

The actual implementation is as follows. For simplicity, we assume that there are only two output classes, but the extension to many output classes is trivial since only two classes will be present at most class boundaries. Three N-bit binary words are used to define the hyper-volume enclosed by a leaf. They are:

1. **Data:** At the first level (level 0) of the tree this word stores the most significant bit of

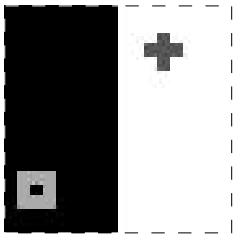

Figure 11: One possible split to learn the second point

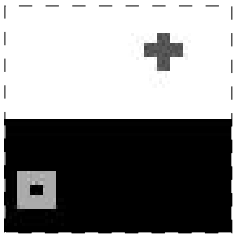

Figure 12: The other possible split to learn the second point

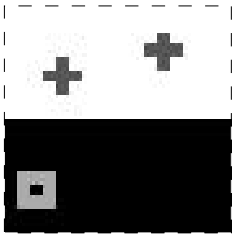

Figure 13: The third point is used to confirm the correct split

each of the N inputs of the example that caused the leaf to be created (the generating point). At the next level, the word stores the second most significant bits, and so on down the tree. This means that the position of the generating point is stored in the tree but only to $(L+1)$-bits precision, where L is the level of the leaf in the tree.

2. **Mask:** This word flags the "don't care" inputs, marking the **Data** bits that are not decision bits.
3. **Dof:** The degrees of freedom (dof) binary word is used to flag the alternative inputs which can be used for rotations with no forgetting of seen examples, *i.e.,* they mark which of the **Data** bits are temporary decision bits.

We use alphanumeric notation to summarize the values of the three words as shown in Table 1.

rule.data	0	0	0	0	1	1	1	1
rule.mask	0	0	1	1	0	0	1	1
rule.dof	0	1	0	1	0	1	0	1
Notation for Data Mask	0	d	x	z	1	D	X	Z

Table 1: Conversion of the three words to alphanumeric notation.

The first of the alternative inputs is always chosen to be the initial temporary split.[2] This means that only one 'D' or 'd' can appear in the data mask and it will always precede 'Z' or 'z' entries (*e.g.,* 1DxXzZ is a valid mask). Each time a new example is seen, the *dof* mask for both the corresponding leaf and it's buddy (the new leaf formed by a temporary split) are updated in the following manner.

- If the leaf correctly classifies the new point, the degrees of freedom are removed where there is a bit difference between the rule data word (*i.e.,* position of the generating point) and the new data point using:

  rule.dof = (rule.dof AND (NOT(rule.data XOR new_point.data))).

  Splits on these differing bit positions are no longer desirable as the rule has seen both states (0 or 1) for that bit (at this level of resolution) and so splitting would lead to forgetting of seen points.

[2]MSC always chooses the first alternative because, at this stage, it does not yet seen enough examples to make a better choice (although the selection could be made randomly). Non-incremental classifiers such as C4.5 can determine the best input for partitioning from the entire training set.

- If the new point is wrongly classified by the rule and **rule.data** $\neq$ **new_point.data**, the degrees of freedom are removed where the rule and new data point are the same using

  rule.dof = (rule.dof AND (rule.data XOR new_point.data))

  Splits on those bit positions are no longer desirable because the rule has seen points with different classes with the same value for that bit position.

- If the new training point is incorrectly classified and **rule.data** = **new_point.data**, the degrees of freedom are retained but the region corresponding to rule.data is converted to a branch to a new universal rule at the next level of resolution with the same class as the new point. The generating point will be forgotten until the next epoch because its exact position at the next level is unknown.

Note that leaves are created with a certain number of degrees of freedom and that they can only lose degrees of freedom as new examples are seen. When a new training point cannot be incorporated by rotating two rules, one of the rules must be split in half. The degrees of freedom are then reset to cover all the possible "don't care" terms and these two rules become *buddies*. In this way the process of creating degrees of freedom is reset on each new rule split.

The method is best illustrated with the use of a simple example. In this case there are three binary inputs with a classification of either 0 or 1.

The first training point produces one universal rule.

Point 0 1 1 Class 1 $\Rightarrow$ RULE 1 DATA = zZZ CLASS = 1 WITH CONFIDENCE 1;

The second training point of the same class removes a degree of freedom from the third bit. This is required because these two training points differ in the third bit position.

Point 0 1 0 Class 1 $\Rightarrow$ RULE 1 DATA = zZX CLASS = 1 WITH CONFIDENCE 2;

The third training point produces a temporary split on the first bit, these two rule leaves are now buddies. Note that because of the rule split, the degrees of freedom have been reset to cover all the "don't care" terms. In this case, resetting the degrees of freedom for the third input may lead to the second point being forgotten until the next epoch.

Point 1 0 0 Class 0 $\Rightarrow$ RULE 1 DATA = dZZ CLASS = 1 WITH CONFIDENCE 2;
DATA = Dzz CLASS = 0 WITH CONFIDENCE 1;

The fourth training point moves the rule split to be on the second bit and updates the degrees of freedom.

Point 0 0 0 Class 0 $\Rightarrow$ RULE 1 DATA = xDZ CLASS = 1 WITH CONFIDENCE 2;
DATA = Xdz CLASS = 0 WITH CONFIDENCE 2;

The fifth training point fits the second leaf so the degrees of freedom are updated and copied to its buddy, the first leaf. Note: this results in only one degree of freedom which is then removed because no alternate splits are possible.

Point 0 0 1 Class 0 $\Rightarrow$ RULE 1 DATA = x1X CLASS = 1 WITH CONFIDENCE 2;
DATA = X0x CLASS = 0 WITH CONFIDENCE 3;

Thus updating the degrees of freedom as each example is seen turns out to be a elegant and efficient means to provide some backtracking in the classification algorithm; instead of storing the position of every example which falls on the current leaf, we only store the generating point and update the dof word. Lack of backtracking is one of the disadvantages of decision tree algorithms such as ID3.

The usefulness of backtracking with degrees of freedom is demonstrated in the solution of the first Monk's problem in Section 3.4. In this 17-dimensional problem, the addition of degrees of freedom rotations improved classification accuracy on the test set from 96% to 100% and reduced the number of rules by a factor of more than two. Moreover, with backtracking the classifier identified the three relevant inputs for the correct solution since all other input dimensions become "don't cares."

2.3 Decision Tree Pruning

MSC is termed a *consistent* learning algorithm [1], in that it constructs a tree to fit the training data exactly. Conventional decision trees stop creating new nodes when the information gained by that node falls below a certain threshold. When the training data is noisy or uncertain the decision trees grown tend to be very large, as nodes are created for few training examples. This is known as over-specialisation, the resulting decision tree usually having poor performance on new, previously unseen data.

There are two solutions to this problem, either tree growth is terminated before it over-specialises (*pre-pruning*), or, the over-specialised tree is pruned to remove the nodes which are not necessary for the solution of the underlying problem (*post-pruning*). Empirical evidence has shown that *pre-pruning* tends not to add a branch when it seems to add no information, but subsequently it is found very useful when combined with another branch. The second method, post-pruning, does not suffer from this problem of localised information and has been shown to be superior [4]. Post-pruning was therefore implemented for MSC. There are a number of post-pruning algorithms detailed in the literature [4,7,21,23], however, in order to evaluate MSC's tree construction techniques a simple pruning algorithm was developed.

If at a branch in the decision tree there are *only* leaves and these leaves are of different output classes then the branch must be on a classification boundary. If there is a small, low-confidence region in feature space that is surrounded by leaves of other output classes then it may be desirable to *merge* it with those other leaves, thus removing it from the decision tree. This low confidence leaf was probably formed because of a single point in the training data that was either wrong due to noise, or is a low probability point. Removal of this leaf decreases tree complexity and will usually improve generalization performance [3].

In depth first pruning this concept is applied recursively from the bottom of the tree and the confidence numbers are adjusted accordingly. The leaves lost during pruning were originally created because of examples present in the training set, so the pruned tree will no longer classify the training set with 100% accuracy. However, the total number of points whose classification is "lost" during pruning can be counted and pruning stopped when a certain number of training points have been lost, say 5%. If leaves with a low confidence are pruned the number of misclassifications on the training set can be kept to a minimum, whilst gaining a large reduction in decision tree size. More sophisticated pruning criteria such as *Cost-Complexity* [4] or *Reduced-Error* [21] could be applied to MSC decision trees. Mingers's [18] finding, that there is no significant relationship between tree creation method and pruning performance is important, because it means that these methods of estimating errors in decision trees can also be applied to MSC [5].

Note that it is possible to assign probabilistic class outputs to the merged trees. For example, if a leaf which classified two points as class 0 is merged with a leaf which classified 8 points as class 1, the merged leaf could be described as "class 1: 80%, with class 0: 20%." This would be useful to suggest alternative classifications. Similar processing has been suggested for ID3 [10, page 64]. The C4.5 algorithm converts confidence numbers (number of training points correctly classified by a rule) into confidence intervals by applying a transformation based on the sampling statistics of the binomial distribution [23].

2.4 Relation to Other Work

The proposed Multiscale classifier is most sensibly compared to ID3, C4.5 [20,22,23] and CART [4] since these classifiers all use trees for their knowledge representation. However, the tree is used in a very different manner in MSC and so it is somewhat misleading to refer to it as a decision tree in the conventional sense. Nevertheless, the algorithm shares a number of

advantages that decision tree algorithms enjoy when compared to alternative strategies:

1. The rules it generates are understandable;
2. Learning from the training set is extremely fast;
3. There is little need for data analysis or fine-tuning;
4. They can cope with non-linearly-separable problems;
5. Learning is deterministic;[3]
6. They can often identify the inputs which are relevant to the classification problem at hand.

MSC is distinguished from other decision tree classifiers by the following features:

1. Learning is incremental since the tree is modified with each example seen — even examples which confirm the existing rules. Each example is learned immediately with little forgetting of previous examples.
2. The tree is non-binary and allows complex logical relationships to be expressed at each node of the tree.
3. A certain amount of backtracking is possible — repartitioning of variables may be possible as new data is presented.
4. Although MSC operates on real inputs, once the inputs are mapped to the range [0,1), only shift and logic operations are required. Thus a dedicated digital hardware implementation would be simple to construct using standard logic gates and shift registers.

3 Benchmark Results

3.1 The Twin Spirals Problem

The "twin spirals" benchmark was first proposed by Alexis Wieland of the MITRE Corporation. It consists of two continuous-valued inputs and a single output. The training set consists of 194 X-Y values, half of which are classified as **black** (shown as '+') and the other half as **white** (shown as 'o'). These training points are arranged in two interlocking spirals that go around the origin three times as shown in Figure 14. This problem is not linearly separable and has been shown to be extremely difficult for conventional backprop neural network algorithms to solve [8].

Figures 15, 16, and 17 show the development of the MSC rules during training. At the fourth epoch the algorithm correctly classifies all the training data and has converged to a solution. For comparison, Lang and Witbrock [15] claim that, with appropriate choice of network, standard backprop solves this problem in 20,000 epochs, backprop with a modified error function requires 12,000 epochs, and Quickprop requires 8000 epochs. Fahlman [8] states that the Cascade-Correlation algorithm needs about 1700 epochs. As decision trees such as ID3 and C4.5 are not incremental but examine the statistics of the whole data set at once the notion of training epochs does not apply.

A measure of the generalization performance can be obtained from the twin-spirals benchmark by using another set of 194 test points which fall on the spirals midway between the training set points. The MSC algorithm classifies 95% of this test data correctly, compared to the test set performance of Cascade-Correlation which achieves between 85 and 93% correct.

[3] Though MSC is dependent on the order in which the training examples are presented.

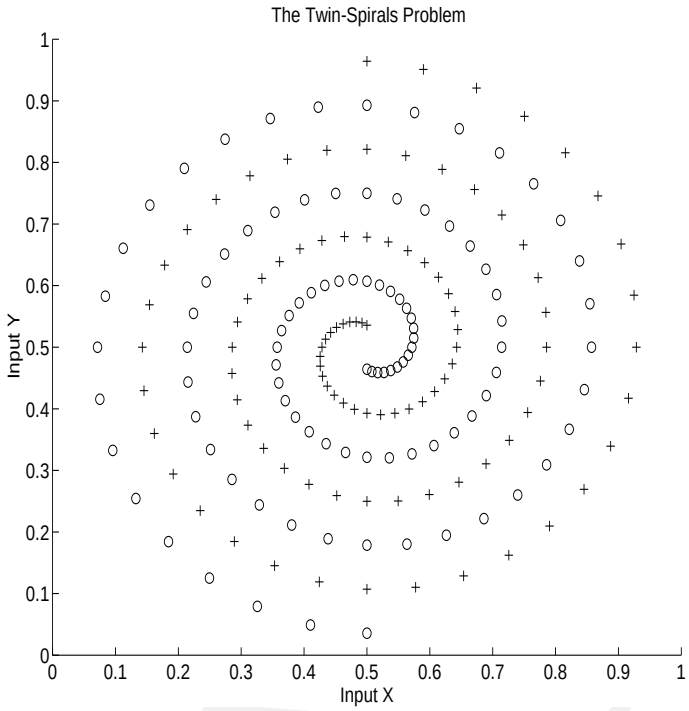

Figure 14: The "twin spirals" training data

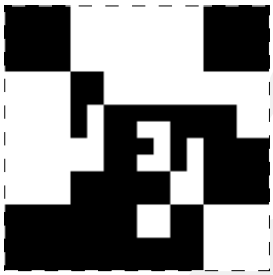

Figure 15: twin spirals classification after 1 training epoch

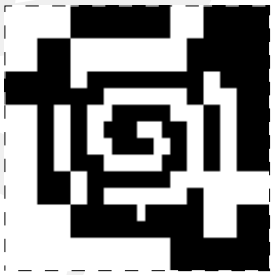

Figure 16: twin spirals classification after 2 training epochs

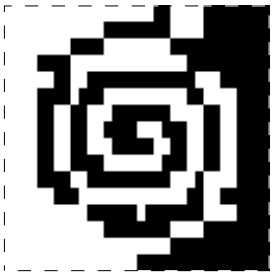

Figure 17: twin spirals classification after 4 training epochs

Classifier	Training	Test
Basic MSC	100%	95%
MSC with Stochastic Gen.	100%	100%
Cascade-Correlation	100%	85–93%
C4.5 (decision tree)	90%	81%
C4.5 (production rules)	92%	84%

Table 2: Performance of classifiers on twin-spirals problem.

C4.5 achieves around 81% using the decision tree and 84% using the production rules[4]. Note that, just like C4.5, the MSC algorithm is deterministic, and so will always yield the same decision tree when presented with deterministic data, in the same order. Cascade-Correlation like other back-propagation neural networks, yields a slightly different network every time it is run. Both MSC and Cascade Correlation algorithms obtain 100% correct classification on the training data, while C4.5 stops growing the tree when around 90–92% of the training data has been correctly classified. This is done to stop over-specialization to the training data by not having leaves in the tree that only cover only a small number of training examples. If C4.5 is allowed to achieve 100% on the training set, the decision tree grows to around twice the size and actually performs slightly worse on the test data. These results are summarized in Table 2.

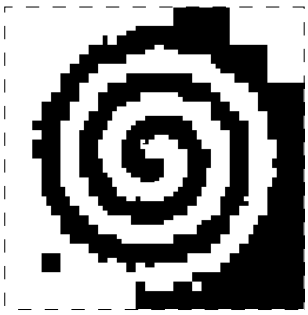

Figure 18: Twin Spirals classification after 90 noisy training epochs

Figure 19: Confidence plot of Twin Spiral classification

Figure 20: Pruned Twin Spirals classification after 90 noisy training epochs

Figures 18 through 20 show the resultant decision tree after the continued training of MSC with uniform random noise added to the twin spirals data. Originally MSC required five levels to correctly classify the training data, so the noise added was bounded[5] to be less than or equal to the resolution of the fifth level (*i.e.,* The diameter of the noise sphere $\leq 2^{-5}$).

This process of adding noise to the training data and then continuing to train the decision tree is called stochastic generalization. In problem domains where the classes cluster but do not overlap, with limited training data, and attributes that are non-binary, the decision boundaries can be "pushed" away from the training points by effectively surrounding them with "noise

[4]Production rules are derived from the decision tree and are designed to be simple and therefore understandable to a human expert.

[5]For this example, the noise was uniformly distributed in both radius and angle.

spheres" [2, page 235]. A similar effect could be obtained by parsing the decision tree and moving the decision boundaries to be midway between clusters of training points. However, this approach would be computationally complex and be counter to the incremental nature of MSC.

Figure 19 shows the confidence plot of the rule tree generated for the twin-spirals problem. Increasing grey level shows increasing confidence of a black classification, while lighter grey levels show increasing confidence of a white classification. The white and black areas show rules where more than 80% of the training points are from the corresponding class.

Figure 18 shows the classification tree before pruning, while Figure 20 shows the rule tree after it has been pruned. Here, any rules that had seen less than two training points were made available to be pruned.

Before stochastic generalization MSC achieved 95% correct classification on the twin spiral test set after only 4 epochs of training. After a further 90 epochs of stochastic generalization, it achieved 100%. After pruning, it still obtained 100% but the rule tree was reduced in size by about 25%. This performance is far better than either C4.5 (81%), Cascade-Correlation (85–93%), or backprop neural networks with a low computational burden.[6]

3.2 Overlapping Classes

It is extremely common to find classification problems where the output classes in the training set overlap. In order to investigate this type of problem MSC was trained to separate two Gaussian noise sources. The first noise source had a classification of **black** (shown as '+'), with a mean at (X, Y) input co-ordinates of $(0.25, 0.25)$, and a standard deviation of 0.2. The second noise source had a classification of **white** (shown as 'o'), with mean $(0.75, 0.75)$ and the same standard deviation. There was a training set of 2000 examples and a test set of another 2000 examples. A sample of the training data is shown in Figure 21.

As shown in Figure 22 the training data was learnt in just five epochs. Since MSC learns the training data perfectly, the rules can be seen to be over-specialized to the training set. This leads to the small isolated rules on the other side of the *ideal* Bayes [11,12,30] classification boundary[7]. The Bayes classifier is able to achieve 96% accuracy on the test set for these two noise sources — no classifier can be expected to do better than this Figure on unseen data. The performance of Quinlan's C4.5 algorithm on this data set is shown in Table 4. This data was also classified using Cascade Correlation, obtaining an accuracy of around 95% on the training data and 93% on the test. In this case the number of hidden units was restricted to 1 so that the algorithm did not add extra units in an attempt to reach the stopping criterion error of 0.2%. Standard back-propagation obtained 95.7% on the test set but this result requires the *a priori* knowledge that no hidden units are required, because a straight line is the best solution. Performance, as expected, reduces on the test set as hidden units are added and the network over-specializes to the training set.

After MSC was trained, the rule tree was then pruned using *depth first pruning* (see Section 2.3). A table of classification accuracy on the test and training set and the total number of rules in the decision tree for different levels of pruning is shown in Table 3.

Rules at the deepest level of the tree are pruned off only if they have a confidence number below a certain threshold. In this case a confidence threshold of 10 points was used, so that only the deepest levels of the tree near to the decision boundary, where many training points fall, are kept. It should also be noted that in this example the decision tree was pruned to almost a third of its original size while its generalization performance increased.

[6] Only 94 epochs instead of 1,700 for Cascade-Correlation, or 20,000 for standard backprop.

[7] a diagonal line from (0,1) to (1,0)

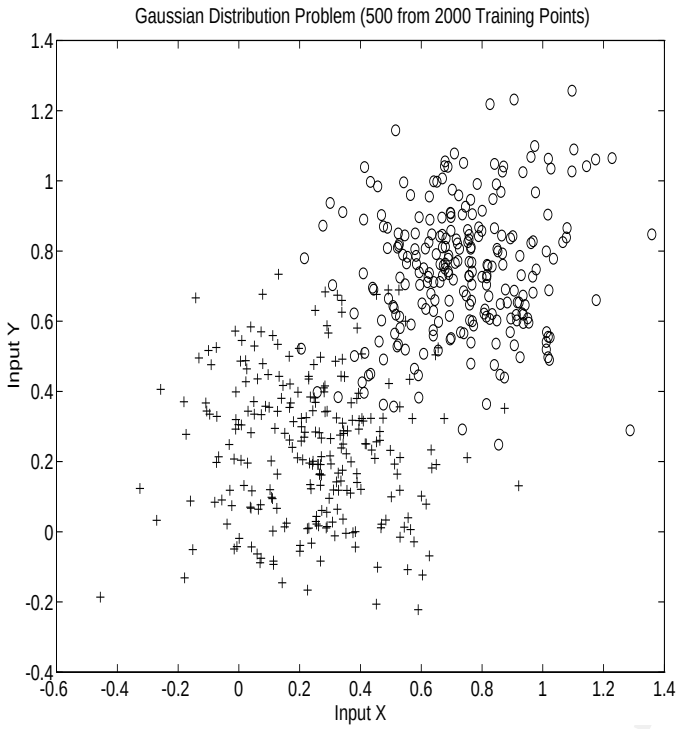

Figure 21: Overlapping Gaussian noise sources training data

Tree Depth	level 8	level 7	level 6	level 5	level 4	level 3
training set	100%	99.85%	99.6%	97.8%	93.5%	89.6%
test set	93.05%	93.05%	92.95%	93%	93.5%	95.3%
num rules	467	463	453	409	293	138

Table 3: Classification accuracy of MSC on overlapping Gaussian distributions with rule pruning.

C4.5	training set	test set	number rules
unpruned tree	98%	94.2%	97
pruned tree	97%	94.9%	35
production rules	97%	95%	11

Table 4: Classification accuracy of C4.5 on overlapping Gaussian distributions.

Yet even the accuracy of the unpruned tree is still near optimal due to the very small area of the islands of classification space on the wrong side of the decision boundary. This behaviour is a natural consequence of the Multiscale strategy since the classifier must descend many levels to separate these exceptional examples from nearby examples of the correct class — thus forming very small leaves with low confidence. As pruning progresses these islands are in effect averaged out and the overall classification approaches the optimal diagonal line. Figure 23 shows the classification boundary when the decision tree has been pruned back to the fourth level. The decision boundary strays off the diagonal towards the corners of the unit square as only few examples are seen in this area.

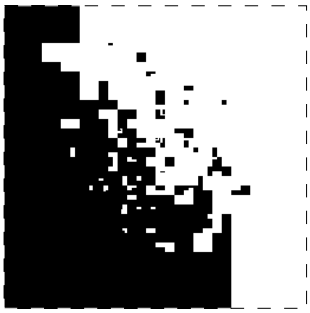

Figure 22: Overlapping Gaussian classification after 5 training epochs

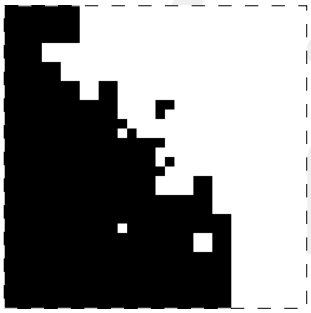

Figure 23: Overlapping Gaussian classification after pruning back to the fourth Level

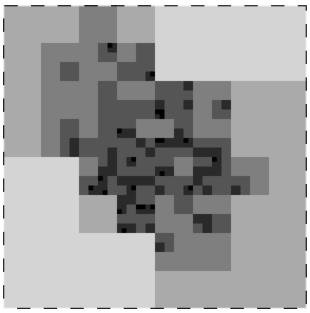

Figure 24: Plot of the levels of resolution used for overlapping Gaussian problem

Figure 24 shows a plot of the levels of resolution required in the Gaussian noise sources problem, a darker grey level indicating a higher level of resolution in the decision tree. Note, that many more levels of resolution are required nearer to the classification boundary.

3.3 The Exclusive-OR Problem

MSC performs quite well on many binary input problems, even though it was specifically developed to take advantage of the resolution redundancy of multi-bit data. The Exclusive-OR (XOR or n-parity) problem is examined for completeness because it is a pathological non-linear problem in many ways. If the XOR problem is presented to MSC, the algorithm effectively tabulates every training example and generalization is poor — the classifier is unable to find any redundancy in the training data because there is no clustering, so the XOR problem is represented by a tree with 2^N leaves and no branches. This is a natural consequence of the sum-of-products type representation.

Unlike MSC, back-propagation neural networks can generalize to an XOR, or near XOR, functional relationship from seeing only 25 to 50% of the training set. This appears to be a telling point in favour of the neural network approach, so it requires further consideration.

One of the reasons neural networks solve this problem is because the network is usually given implicit *a priori* information about the problem, by specifying a small fixed architecture network. So, if a solution is found, it is guaranteed to be compact. As long as the XOR function is within the representational power of the small network, even a random search of the relatively compact weight space would eventually find something close to the XOR solution. Conversely,

if a larger architecture is specified (more hidden units) the network is likely to find a solution that fits the training data, but is not the XOR rule.

The architecture of MSC gives an implicit generalization strategy that "Nearby points in feature space should have the same classification" which happens to run counter to the XOR rule. A way around this problem is to add an extra input feature based on the original inputs. Indeed, if we add the XOR of all the existing inputs as an additional input, the degrees of freedom rotations will quickly home in on this variable and all other inputs will remain "don't cares." An optimally compact tree with only two leaves will result and perfect generalization from the training set will occur with few training samples.

This approach seems like cheating, but it can be useful for finding compact solutions to problems that are close to XOR since the classifier will attempt to fit XOR mappings to the data. In general, it is known [19] that any function can be made linearly separable by the addition of an *appropriate* new feature.

3.4 The Monks Problems

Some more interesting binary problems are the Monk's problems, originally designed to enable comparison of a number of different algorithms in the 2^{nd} European Summer school on Machine Learning, July 1991 [28]. They consist of three classification problems for different types of robots, they are all based on six attributes:

attribute		*possible values*
x1: head shape	$\in$	round, square, octagon
x2: body shape	$\in$	round, square, octagon
x3: is smiling	$\in$	yes, no
x4: holding	$\in$	sword, balloon, flag
x5: jacket colour	$\in$	red, yellow, green, blue
x6: has tie	$\in$	yes, no

The three problems are fundamentally different in the type of classification problem they pose.

- The first problem is a sum-of-products type problem where only three of the input dimensions are relevant to the output classification.
  (head shape = body shape) OR (jacket colour = red)
- The second states that exactly two of the six attributes must have their *first* value. This is similar to an XOR relationship.
- The third is again a sum-of-products type problem but has 5% misclassifications present in the training set.
  (jacket colour = green AND holding = sword) OR
  (jacket colour $\neq$ blue AND body shape $\neq$ octagon)

All three problems have a total test set size of 432 examples, the training data being randomly selected from this set. For further details on the problems and the performance of the other learning algorithms on these problems, refer to [28].

The 6 attributes in the Monks problems can have 3, 3, 2, 3, 4, and 2 different values respectively and so the inputs can be mapped to either 6 multi-valued inputs or 17 binary inputs with each bit corresponding to a particular attribute value being set. This 17 input binary approach was used by Thrun [28] for his solution using a back-propagation neural network. Results from both approaches are given here for completeness.

3.4.1 Monk #1

num dimensions	training epochs	basic MSC	MSC with rotations
6	3	79.2%	79.2%
17	1	95.8%	100%

Table 5: Classification accuracy of MSC on Monk #1 test set

These results show that on the first Monk's problem MSC with rotations learnt the problem with 100% accuracy on the test set. Figure 25 also shows that the rule set was compact, although not minimal.

```
RULE 1 DATA = 1zz1zzZzZzzzzZzZz CLASS = 1 WITH CONFIDENCE 9;
       DATA = 0Xx1xxXxXxxdxXxXx CLASS = 0 WITH CONFIDENCE 20;
       DATA = 0Xx1xxXxxXxDxxxXx CLASS = 1 WITH CONFIDENCE 6;
       DATA = 1xx0XxXxXxx0XxxXx CLASS = 0 WITH CONFIDENCE 31;
       DATA = 01x01xXxXxx0XxxXx CLASS = 1 WITH CONFIDENCE 12;
       DATA = 00X01xXxXxx0xxXxX CLASS = 0 WITH CONFIDENCE 5;
       DATA = 01x00XXxxXx0xXxXx CLASS = 0 WITH CONFIDENCE 6;
       DATA = 00X00XXxXxx0XxxXx CLASS = 1 WITH CONFIDENCE 14;
       DATA = Xxx0XxXxxXx1xxxXx CLASS = 1 WITH CONFIDENCE 21;
       ( Level 0 )

   TOTAL NUMBER OF TRAINING POINTS = 125
```

Figure 25: Rule file for Monk #1 problem using MSC with rotations.

The rule file also illustrates which of the input dimensions are *relevant* to the solution of the problem. In this case inputs 1 and 2 (both 3 bits) and the first bit of input 5 are needed to solve the problem. This is an important feature of the degrees of freedom rotations that enables further understanding of the rules learnt for the given problem. In this case examining the rule file allows us to confirm the rule for the first Monks problem.

When the classifier was used without rotations, the rule set was 3 times larger and did not identify the relevant inputs.

3.4.2 Monk #2

num dimensions	training epochs	basic MSC	MSC with rotations
6	3	89.6%	91.2%
17	3	76.2%	88.7%

Table 6: Classification accuracy of MSC on Monk #2 test set

This problem does not have clusters in feature space and so runs contrary to the MSC's generalization strategy. This is particularly noticeable for the case when the inputs are 17 input binary. However, for the 6 input real case the data clusters better.

3.4.3 Monk #3

num dimensions	training epochs	basic MSC	MSC with rotations
6	3	73.1%	73.1%
17	2	80.6%	91.2%

Table 7: Classification accuracy of MSC on Monk #3 test set

This problem is made more difficult because of the presence of 5% misclassifications or "noise" in the training set. With this taken into consideration it would be impossible to achieve 100% on the test set unless some strategy is employed to effectively ignore training points that don't fit a general rule — this could be achieved using tree pruning.

If the test and training set are now switched, the classifier will now learn the problem perfectly and be able to distinguish which of the examples in the original training set have had noise added to them. Figure 26 shows the points which appear to be the noisy training points deliberately added to the third Monks problem. These points are in agreement with those found by Fahlman [28] using Cascade Correlation in his trials with the Monks problems.

```
Misclassified point   1 0 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0   Predicted 1 Should be 0
Misclassified point   1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 1 0   Predicted 1 Should be 0
Misclassified point   0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0   Predicted 1 Should be 0
Misclassified point   0 1 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0   Predicted 1 Should be 0
Misclassified point   0 1 0 0 1 0 1 0 0 0 1 0 0 1 0 0 1   Predicted 1 Should be 0
Misclassified point   0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0   Predicted 0 Should be 1
```

Figure 26: The noisy training points in Monk #3 Problem.

3.4.4 Comparison with Other Learning Strategies

Algorithm	Monk #1	Monk #2	Monk #3
ID3	83.2%	69.1%	95.6%
ID3 with windowing	98.6%	67.9%	94.4%
Assistant Professional	100%	81.3%	100%
C4.5, production rules	75.7%	65%	97.2%
C4.5, attribute grouping	100%	70.4%	100%
Back-propagation	100%	100%	93.1%
Backprop with weight decay	100%	100%	97.2%
Cascade Correlation	100%	100%	97.2%
MSC	100%	88.7%	91.2%

Table 8: Brief comparative results on the Monks problems

Table 8 shows a comparison of the results obtained using MSC and some of the other learning strategies mentioned. These results are a selection taken from Thrun's report [28] and Quinlan's

book on C4.5 [23]. It shows that even though MSC is only using one level of resolution to solve the problems it does as well as, if not better than, many more complex strategies. It should also be noted that exactly the same algorithm was used to solve each problem with no *a priori* knowledge being required to select a suitable network architecture or to set learning parameters. The performance on Monk #3 is weaker because it was obtained from the basic MSC with no attempt to remove spurious points, unlike most of the other strategies. "Assistant Professional" [6] is also shown here as it is a descendant of ID3 with improved probability estimates and binarization of attributes. It also has built-in tree pruning which was used on the third problem to remove the "noisy" training examples.

3.5 Other Results

The MSC algorithm has been applied to a number of other machine learning data domains [5]. The datasets were chosen so that there was a variety of attribute noise, classification noise, residual variation, number of training points, output classes, discrete and continuous input variables. The domains chosen, together with some previously reported results, were:

1. LED:Recognizing digits on a faulty 7-segment LED display. Optimal Bayes classification rate of 74% and CART [4] obtained an accuracy of 71% using resubstitution.
2. IRIS:Recognizing flower types from physical measurements. Classification rates of between 84% (second order Bayes) and 98% (linear discriminant) have been reported under cross-validation [30].
3. GLASS:Recognizing glass types from forensic data. Accuracies of 53.8%, 56.3% and 62.2% have been reported for a *single* attribute decision tree [13].
4. BREAST:Diagnosing breast cancer as either benign or malignant. Accuracies of 93.5% and 95.9% have been reported for a multisurface method [31], on a single train-and-test partition.
5. PIMA:Diagnosing diabetes among female Pima Indians. Reported accuracy, using the ADAP learning algorithm, of 76% on a single train-and-test partition [27].

The actual datasets and further information regarding them can be obtained from a machine-readable data repository at the University of Southern California, Department of Information and Computer Science (ics.uci.edu:pub/machine-learning-databases).

Dataset	LED	IRIS	GLASS	BREAST	PIMA
MSC Unpruned	71.2%	96.1%	62.1%	94.7%	66.3%
MSC Minerr	N.A.	94.8%	63.7%	94.9%	68.6%
MSC Pessim	N.A.	92.1%	62.2%	94.8%	66.4%
C4.5 Unpruned	73.5%	94.7%	70.5%	89.5%	70.2%
C4.5 Pruned	72.3%	94.7%	69.6%	90.4%	72.5%
C4.5 Production	71.6%	94%	67.2%	93.1%	71.6%

Table 9: Brief comparative results on five other machine learning datasets

Table 9 shows the accuracy of MSC on these datasets, for the unpruned tree, trees pruned using the revised form of Minimum Error [7], and Pessimistic Pruning [23]. It should be noted that the tree generated for the LED dataset is not pruned because all of the input attributes

are binary, and so it is a subtree of all leaves[8]. If this tree were pruned it would produce a single leaf of the most frequently occurring class. The error of this single leaf would be higher than the original subtree and so the pruning does not occur. Results are also shown in Table 9 for C4.5 unpruned and pruned decision trees, and production rules. The results shown for both MSC and C4.5 are for 10-fold cross validation [30], and so are an almost unbiased estimate of the true accuracy. The results illustrate that MSC compares well with the other techniques in these data domains, though its performance on the datasets with large numbers of output classes (LED and GLASS have 10 and 7 output classes respectively), attribute noise and residual variation are not as favourable.

For a further comparison of the proposed method, results from a large number of other classification algorithms on these and other datasets can be found in [13]. In addition the Authors have arranged for the current Microsoft Windows executable version of The Multiscale Classifier to be available via anonymous ftp (enterprise.cssip.elec.uq.oz.au:pub/cssip/software/msc), so that independent researchers can have access to the algorithm.

4 Further Comments

4.1 Reject Set

A problem with many classifiers is that they do not test their data to see if it is valid. For example, a classifier that is trained to distinguish between the silhouettes of ships and aircrafts may classify the silhouette of a person as a ship — it has to choose one of the two classes. One way to get around this problem is to have a large training set of reject data which should be classified as "rejected." However, because of its very nature,[9] it is hard to find enough reject examples to completely envelop the interesting regions of feature space.

In the case of MSC, we can synthesize a very large reject set by randomly generating points in feature space and classing them as rejected. These examples will generate large reject class leaves in sparsely populated areas and very small leaves (which will often be forgotten) in densely populated areas. Pruning will eliminate leaves due to reject examples in the densely populated regions and ensure that the interesting part of feature space is entirely surrounded by reject class leaves.

4.2 Missing Input Data

In many *real world* multi-dimensional problems, where the information has been collated to be examined by a human expert, there are often features or bits of information missing or unknown in some of the examples. Humans handle this problem well and are still able to make some sort of classification based on the available information.

MSC can handle this problem by trying all possible routes down the decision tree with this piece of information missing. This is possible because at each level of the tree each input is stored as one bit in the rule data. At rules where this missing input bit is not a "don't care" term, both of the possibilities can be followed down the tree. This, of course, may lead to a number of possible classifications for that example the final decision being based on the confidence levels associated with those classifications. As long as the missing bit of information is not one of the most important input dimensions, the number of paths to follow down the tree

[8] This is because problems with binary inputs (0 or 1) can be solved using only the most significant bit of each input.

[9] Since the reject set is set of of examples that have no common feature other than being different from objects of interest, it is huge and varied.

will be relatively small. If the missing bit of information is one of the most important input dimensions then it may not be wise to attempt a classification on that example anyway.

5 Future Work

Further work is proceeding on techniques to improve and measure generalization performance, in the areas of: re-sampling techniques [30], pruning strategies [4,7,23], cost-sensitive pruning [14], and the addition of synthetic input dimensions.

The ability of MSC to perform fast, incremental learning makes it ideally suited for on-line expert knowledge acquisition through a point-and-click interface. For example, in the development of an automated pap-smear system to screen for cervical cancer, we envisage an interface where a cytologist would click on a cell image and MSC would determine the cell type. If the classifier were wrong, the expert would correct the classification and MSC would update its rule tree. The nature of MSC ensures that each example is learnt immediately. Here MSC offers a significant advantage over classifiers such as C4.5 and CART which must rebuild their decision trees from scratch each time new examples are added. On the other hand, neural networks would learn new examples too slowly to be satisfactory in this application.

The MSC technique is currently being used to examine a number of problems in medical diagnostics where it is ideally suited because of its rule-based nature and confidence level output.

6 Conclusions

The *Multiscale Classifier* (MSC) has been proposed and in the results reported here it offered fast learning and comparable (if not better) generalization performance to both decision tree classifiers and backprop neural networks. Of particular interest is the ability of the Multiscale Classifier to learn incrementally, cope with conflicting data, and determine the relevant inputs by backtracking.

7 Acknowledgments

The Authors are grateful to Alvin Mok for for his work in taking initial research code and turning it into a user-friendly, well documented software package. Thanks are also due to the anonymous referees for helpful comments on earlier drafts of this paper.

8 References

[1] M. Anthony and N. Biggs, in *Computational learning theory : an introduction*. Cambridge University Press, 1992.

[2] R. E. Blahut, in *Digital Transmission of Information*. Reading, Mass.: Addison-Wesley, pp. 306–313, 1990.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, Vol. 24, pp. 377–380, 1987.

[4] L. Breiman, J. Freidman, R. Olshen and C. Stone, in *Classification and Regression Trees*. Belmont: Wadsworth, 1984.

[5] R. Burnett, "Theory and Application of the Multiscale Classifier," Dept. Electrical and Computer Engineering, University of Queensland, Honours Thesis, 1994.

[6] B. Cestnik, I. Kononenko and I. Bratko, "Assistant 86: A Knowledge-Elicitation tool for sophisticated users," in *Progress in Machine learning*, I. Bratco and N. Lavrac, Eds. Wilmslow: Sigma Press, 1987.

[7] B. Cestnik and I. Bratko, "On Estimating Probabilities in Tree Pruning," in *Machine Learning: EWSL-91: European Working Session on Learning*, Y. Kodratoff, Ed. Porto, Portugal: Lecture notes in Artificial Intelligence: 482, pp. 138–150, 1991.

[8] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Ed. San Mateo: Morgan Kaufmann, pp. 524–532, 1990, Volume 2.

[9] R. Fisher, "The use of multiple measurements in taxinomic problems," *Annals of Eugenics*, 7, pp. 179–188, 1936.

[10] R. Forsyth and R. Rada, in *Machine Learning: Applications in Expert Systems and Information Retrieval.* West Sussex: Ellis Horwood Limited, 1986.

[11] A. D. Gordon, in *Classification.* Chapman and Hall, 1981.

[12] D. J. Hand, in *Discrimination and Classification.* John Wiley and Sons, 1981.

[13] R. C. Holte, "Very simple Classification Rules Perform Well on Most Commonly Used Datasets," *Machine Learning*, Vol. 11, pp. 63–91, 1993.

[14] U. Knoll, G. Nakhaeizadeh and B. Tausend, "Cost Sensitive Pruning of Decision Trees ," in *Machine Learning: Proceedings of ECML-94.* pp. 383–386, 1994.

[15] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in *Proceedings of the 1988 Connectionist Summer School.* Morgan Kaufman, 1988.

[16] J. L. McClelland and D. E. Rumelhart, in *Explorations in parallel distributed processing.* Cambridge: MIT Press, pp. 121–137, 1988.

[17] D. Michie and R. Chambers, "Boxes: an experiment in adaptive control," in *Machine Inteligence 2*, Dale and Michie, Eds. Edinburgh University Press, 1968.

[18] J. Mingers, "An empirical comparison of pruning methods for decision tree induction," *Machine Learning*, Vol. 4, pp. 227–243, 1989.

[19] M. L. Minsky and S. A. Papert, in *Perceptrons.* Cambridge, Mass: MIT Press, 1988, Expanded Edition.

[20] J. R. Quinlan, "Semi-autonomous acquisition of pattern-based knowledge," in *Introductory Readings in Expert Systems*, D. Michie, Ed. Gordon and Breach, 1982.

[21] J. R. Quinlan, "Simplifying decision trees," *International journal of man-machine studies*, Vol 27, pp. 221–234, 1987 .

[22] J. R. Quinlan, "Inductive Knowledge aquisition: a case study," in *Applications of expert systems*, J. R. Quinlan, Ed. North Ryde, N.S.W. : Addison-Wesley, pp. 157–173, 1989b.

[23] J. R. Quinlan, in *C4.5:Programs for Machine Learning.* San Mateo: Morgan Kaufmann, 1993 .

[24] F. Rosenblatt, "The perceptron: a probabalistic model for information storage and organization in the brain," *Psychological Review*, Vol 65, 1958.

[25] H. Samet, "The Quadtree and related hierarchical data structures," *Computing surveys*, Vol 16, 1984.

[26] A. Samuel, "Some studies in machine learning using the game of checkers, part II," *IBM Journal of Research and Development*, Vol 11, no. No. 4, pp. 601–618, 1967.

[27] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler and R. S. Johannes, "Using the (ADAP) Learning Algorithm to Forecast the Onset of Diabetes Mellitus," in *Proceedings of the Symposium on Computer Applications and Medical Care*. IEEE Computer Society Press, pp. 261–265, 1988.

[28] S. B. Thrun, "The MONK's problem: A performance comparison of different learning algorithms," Carnegie-Mellon University, Technical Report, 1991.

[29] R. F. Tinder, in *Digital Engineering Design*. New Jersey: Prentice-Hall, 1991.

[30] S. M. Weiss and C. A. Kulikowski, in *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Networks, Machine Learning, and Expert Systems*. San Mateo: Morgan Kaufmann, 1991.

[31] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology," in *Proceedings of the National Academy of Sciences, U.S.A.*, vol. Vol. 87, no. 12. pp. 9193–9196, 1990.