Joint feature selection and hierarchical classifier design

Cecille Freeman, Dana Kulić, Otman Basir
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada

***Abstract*—This work presents a method for improving classifier accuracy through joint feature selection and hierarchical classifier design with genetic algorithms. The hierarchical classifier divides the classification problem into a set of smaller problems using multiple feature-selected classifiers in a tree configuration to separate the data into progressively smaller groups of classes. This allows the use of more specific feature sets for each set of classes. Several existing performance measures for evaluating the feature sets are investigated, and a new measure, count-based RELIEF is proposed. The joint feature selection and hierarchical classifier design method is tested on two artificial data sets. Results indicate that the feature selected hierarchical classifiers are able to achieve better accuracy than a non-hierarchical classifier using feature selection alone. The newly proposed performance measure is also tested and shown to provide a better indication of classifier performance than existing methods.**

***Index Terms*—Classification algorithms, Genetic algorithms, Input variables**

I. Introduction

The performance of a classifier is affected by a number of different factors including classifier type, the input features and the desired outputs. This work aims to improve classifier performance by jointly selecting the features and the classifier structure, to create a feature-selected hierarchical classifier.

Feature selection can improve performance in a number of ways. Reducing the number of input features can reduce classifier complexity and improve classification speed. Feature selection can also help minimize the impact of noisy features, which can cause false association between relatively random features and the classifier output. It can also help to identify redundant or correlated features, which increase classifier complexity without adding much additional information, adversely affecting performance [1]. However, identifying noisy, redundant and correlated features is not a straightforward task.

The evaluation of redundant and correlated features is complicated by the fact that multi-way redundancies and correlations can exist. Features that are not useful alone can provide a significant advantage when paired with other features. Hence feature selection requires not just individual or pairwise feature evaluation, but evaluations of the entire subset [1].

In some cases, different classes can be best separated using different sets of features. The standard approach to multi-class classification is to use a single classifier with many outputs. However, another option is to split the problem into a set of smaller classification problems, each using its own feature set.

Splitting the problem can increase the overall accuracy of the classifier for a number of reasons. It allows the use of a more specific set of features for each set of classes. Additionally, for some classifiers, better generalization can be achieved by using a smaller classifier [2].

This work presents a method for joint feature selection and hierarchical classifier design using genetic algorithms. The algorithm constructs a hierarchical classifier by arranging several "base classifiers" into a tree formation. Each base classifier separates the data set into a progressively smaller set of classes. Features are selected individually for each base classifier, and the design of the tree and the feature selection are performed simultaneously.

The remainder of the paper is organized as follows: Section II discusses some of the previous work in this area, Section III presents the design of the system, Section IV discusses the experimental setup, Section V presents the results and gives some discussion and Section VI presents the conclusions and discusses the future directions for this work.

II. Previous Work

Traditional feature selection consists of two interconnected parts. A feature set performance measure is used to estimate the fitness of a feature set and a feature set selection process is used to select and improve the feature set being tested, based on the information provided by the performance measure.

Feature set performance measures can be grouped into two major categories: wrapper and filter [3]. Wrapper techniques evaluate the fitness of a feature set by evaluating the set directly on the intended classifier. Filter techniques estimate the fitness of a feature set using a calculated measure that does not require testing with the classifier. In general, wrapper methods tend to give better results [4], but can be computationally expensive, especially for trained classifiers.

A. Feature set selection techniques

Two of the best performing feature set selection techniques are sequential search and stochastic search [5]. Sequential search techniques create and evaluate feature sets by adding and removing promising features. The most common sequential algorithms are the sequential forward floating search (SFFS) and its counterpart the sequential backwards floating search (SBFS) [6].

A number of papers have tested the SFFS and SBFS algorithms against various other sequential search algorithms.

Pudil, Novovičovaá and Kittler [6] and Zongker and Jain [7] both recommend SFFS/SBFS.

Nakariyakul and Casasent [8] propose an extension to SFFS, called improved forward floating search (IFFS), that allows weak features to be conditionally replaced. The new algorithm achieves better results at the expense of a longer run time.

Kudo and Sklansky [5] compare genetic algorithms to a number of sequential and simple search techniques. The results indicate that the genetic algorithm is competitive in terms of run-time and generates feature sets that perform very well. They recommend either a genetic algorithm or SFFS/SBFS.

Other optimization techniques, such as simulated annealing have also been found to work well for feature selection [9].

B. Feature set evaluation measures

A number of different wrapper and filter-based measures have been proposed in literature. Two common filter-based feature set evaluation measures are Fisher's interclass separability criterion [10] and RELIEF [11]:

1) Fisher's interclass separability criterion : This measure is high when the average distance between points in the same class is small, and the average distance between points in different classes is large. This gives a set of widely spaced clusters that each contain points from a single class. It is computed as follows [10]:

$$J = trace[(Q_b + Q_w)^{-1} Q_b] \tag{1}$$

where J is the Fisher's interclass separability criterion, Q_w is the within-class scatter, and Q_b is the between class scatter. Q_w and Q_b are defined as [10]:

$$Q_w = \frac{1}{N} \sum_{c=1}^{C} \sum_{p=1}^{N_c} (x_{pc} - M_c)(x_{pc} - M_c)^T \tag{2}$$

$$Q_b = \frac{1}{C} \sum_{c=1}^{C} (M_c - M_{\tilde{c}})(M_c - M_{\tilde{c}})^T \tag{3}$$

where N is the number of samples, C is the number of classes, N_c is the number of samples in class c, x_{pc} is point p from class c, M_c is the mean of class c, and $M_{\tilde{c}}$ is the mean of all samples not in class c.

2) RELIEF : For each point, the RELIEF algorithm measures the difference in distance between the k nearest points in the same class (nearest hits) and the k nearest points in a different class (nearest misses). The nearest hit is defined as [12]:

$$H_{pc} = \min_{i \neq p} d(x_{pc}, x_{ic}) \tag{4}$$

where points x_{pc} and x_{ic} are both from class c, x_{pc} is the point being evaluated, and $d(x_p, x_i)$ is the distance between points x_{pc} and x_{ic}. The nearest miss is defined as:

$$M_{pc} = \min_{i,j} d(x_{pc}, x_{ij}) \tag{5}$$

where the x_{ij} is in a different class than x_{pc}. The measure can then calculated for each individual class as:

$$r_c = \frac{1}{N_c K} \sum_{p=1}^{N_c} \left(\sum_{k=1}^{K} M_{pc}^k - \sum_{k=1}^{K} H_{pc}^k \right) \tag{6}$$

where H_{pc}^k and M_{pc}^k are the k^{th} nearest hit and miss for x_{pc}.

C. Hierarchical classifiers

Class set division is most often encountered when using binary classifiers to solve multi-class problems by reformulating them as multiple binary problems. This is commonly done using a one-vs.-rest approach, where C classifiers are used and each classifier determines if a point is part of a specific class. This approach can also be used for multi-class classifiers, and can improve the overall accuracy [13].

Another option proposed by Cheong, Oh and Lee [14] is the binary tree architecture (BTA), which they test using support vector machines (SVM). Each SVM divides the classes into two groups, which are then subdivided further until one class is selected. The decision about how to subdivide the classes is made using a self-organizing map. Overall, the authors found that SVM-BTA is able to outperform both the one-vs-one and one-vs-many modes of multi-class SVM classification. However, using a SOM to select the divisions is subjective because a SOM does not give a definite division between different groups. The SVM-BTA technique would be easier to apply and would produce more consistent results if the division procedure was less subjective.

III. Hierarchical Classifier Design

This work extends the ideas presented by Cheong, Oh and Lee [14] by providing a more robust method for designing the tree structure and extending the tree structure to work with multi-class classifiers. It also performs feature selection for the individual base classifiers in the tree.

A genetic algorithm is used to simultaneously optimize the features and tree structure. Genetic algorithms [15] are a type of optimization technique that is based on the process of natural selection. They work by randomly generating a large set of potential solutions to a problem (the "population"), and then iteratively identifying good solutions and combining these to create new populations. A single solution to a problem is specified by a genome. Hence a genetic algorithm requires a genome that can be used to construct an individual solution, a method for scoring each individual and an evolutionary operator that can be used to create new solutions.

Genetic algorithms are selected as a basis for this work due to their proven effectiveness for feature selection [5], and due to the ease of specifying both the feature selection and the tree structure in the genome. Other optimization techniques such as simulated annealing could also be used to solve this problem, and may be explored as future work.

A hierarchical classifier uses a number of base classifiers that separate the data into progressively smaller groups of classes. Consider each base classifier as a black box with up to C outputs for a multi-class classifier. The job of each base

classifier is to place points from each class into its assigned output. This is depicted in Fig. 1. Each output can either represent a single class or a set of classes. If the base classifier has more than one class assigned to one output, another base classifier is added below to further separate the classes. Hence the output of the higher level base classifiers determines if there are lower level base classifiers. The tree structure is specified in the genome by specifying the outputs for each class at each level of the tree. The genome also specifies the feature set for each base classifier.

The remainder of this section outlines the details of the genetic algorithm implementation, including the specification of the genome, the construction of the hierarchical classifier and the selection of the base classifier feature sets from the genome, the scoring function for each individual in the population, and the evolutionary operators that are used to generate new individuals in successive generations.

A. Genome Representation

The genome consists of two major parts. The first part of the genome is used to determine which features are used in each base classifier. The second portion of the genome is used to determine the classifier structure by specifying the class outputs at each level of the classifier.

In a multi-class classifier, the smallest number of base classifiers required is one, where each of the C classes in the first classifier maps to a single output. The largest number of base classifiers required is $C-1$, which occurs if each base classifier is binary (see Fig. 2). The largest number of classifier levels is $C-1$, which occurs when each classifier separates a single class. This is illustrated in Fig. 2b.

The first portion of the gene gives a true or false (one or zero) value for each feature for each possible base classifier. The largest number of base classifiers required in a system is $C-1$. Hence, the first portion of the genome consists of $F(C-1)$ genes, where F is the number of features.

The second portion of the genome gives the output number of each class in each layer of the genome. The largest possible number of layers required in a hierarchical classifier is $C-1$. Hence to specify the output for each class in each layer, the second portion of the classifier is $C(C-1)$ genes each of which can take a value between zero and $C-1$.

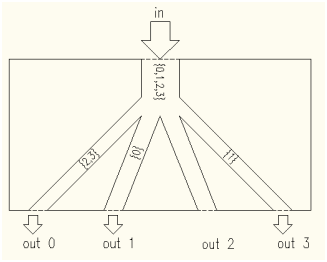
The full genome is illustrated in Fig. 3.

B. Hierarchical Classifier Construction

The hierarchical classifier is built from the second part of the genome, starting from the root.

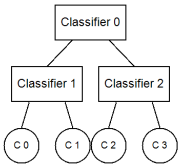
Genes for classes that are fully separated in higher layers are maintained, but once a class is separated, no further base classifiers are constructed for that class. Hence these gene values do not affect the tree. In Fig. 2, these genes marked as 'X'.

The construction of the classifier is accomplished by checking each class in numerical order starting from class zero and building the base classifiers it requires. Later classes are built off of the existing structure.



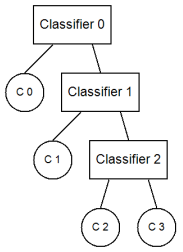
...	Classifier sec.				
...	level 0				...
...	3	1	0	0	...

Fig. 1. Classes split to different outputs. The left side of the figure illustrates a base classifier dividing the classes into different outputs and the right side of the figure shows the portion of the gene that describes this base classifier. Points that are in class 0 are sent to output 3, points that are in class 1 are sent to output 1 and points that are either in class 2 or class 3 are sent to output 0. Another base classifier would further separate the points from output 0.



...	Second portion of gene											
...	level 0				level 1				level 2			
...	0	0	1	1	0	1	0	1	X	X	X	X

(a) balanced binary tree and gene. In level 0, classes 0 and 1 are sent to one output (output 0) and classes 2 and three are sent to another (output 1). Hence, in level 1, classes 0 and 1 are separated by one base classifier, added under output 0 of the top level classifier, and classes 2 and 3 are separated by another base classifier, added under output 1 of the top level classifier. All classes are fully separated in level 1, so the level 2 portion of the gene does not affect the tree structure.



...	Second portion of gene											
...	level 0				level 1				level 2			
...	0	1	1	1	X	0	1	1	X	X	0	1

(b) long binary tree and gene. In level 0, class 0 is sent to output 0 and all the other classes are sent to output 1. Hence class 0 is fully separated in the first layer. The gene continues separating one class each level by placing a single class in a different output. Genes marked as X are genes for classes that have been separated in higher layers. These are maintained, but do no affect the tree structure.

Fig. 2. Genes for two four-class hierarchical classifiers. The right side of the figure shows the portion of the genome describing the hierarchical classifier structure and the left side shows the corresponding hierarchical classifier.

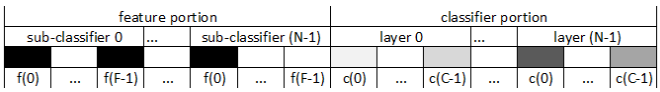


Fig. 3. Full genome used for feature selection and classifier construction. The first portion is used for feature selection and has one gene for feature for each base classifier in the tree. The second portion describes the tree by giving the output number for each class in each level of the tree.

```
for c = each class {
   initialize array stillInGroup to all 1
   numInGroup = numClasses - 1;
   for lvl = each layer {
      outc = output of class c in level lvl
      for i = each class {
         outi = output of class i in level lvl
         if((outi != outc) && stillInGroupi) {
            stillInGroupi = 0;
            numInGroup --;
         }
      }
      if(numInGroup == 0) {
         set class c as fully separated in level lvl
      }
   }
}
numLayersTotal = max(separated)+1;
```

Fig. 4. Pseudocode for calculating class separation layers

```
add root (level 0) base classifier
for c = each class {
   currClassifier = 0 // layer 0 classifier
   sepLvl = level where c is fully separated
   numShared = # levels where c shares base classifiers
               with lower numbered classes
   for lvl = 0 to (numShared-1) {
      outc = output # for class c in level lvl
      add outc as output for class c in currClassifier
      lastClassifier = currClassifier
      currClassifier = child classifier for output outc
   }
   for lvl = numShared to sepLvl {
      add a new classifier and initialize
      set parent of new classifier as lastClassifier
      set child of output outc in lastClassifier as
          new classifier
      outc = output # for class c in level lvl
      add outc as output for class c in new classifier
      set features of new classifier
      lastClassifier = new classifier
   }
}
```

Fig. 5. Pseudocode for adding base classifiers

Each base classifier knows its immediate parent and its level. Each base classifier also maintains a list of the children base classifiers for each of its outputs. If an output is not being used, this is indicated in the child list. If a single class is assigned to an output there are no base classifiers and this is also indicated in the child list. Each base classifier also has a list that indicates the output to which each class is assigned. If a class is not being classified by a particular base classifier, this is indicated in the class list. Lastly, each base classifier maintains a list of the features that it is using.

The hierarchical classifier is constructed in stages. First, the tree is examined to determine in which layer each class is separated. Next, the tree is built by building the base classifiers for each class. Last, degenerate base classifiers, where no classes are actually separated, are removed by adjusting the parents and children. See Fig. 4 and 5 for pseudocode.

C. Feature selection for base classifiers

The features used by each base classifier are selected in the first portion of the genome. There is one section for each base classifier, with F integers that can be one or zero, where F is the number of features. If the value is one, the feature is used by the base classifier, if it is zero, the feature is not used. This structure can also be extended to perform feature weighting by using non-binary values. In this work, however, only binary values are used. The use of feature weighting may be explored in future works.

D. Scoring

The score for each classifier is based on an estimate of the accuracy, which can be estimated in one of two ways. The first method estimates the accuracy of the tree as a whole, which requires trained classifiers and a wrapper-based measure. The entire tree is built, and the accuracy is estimated by testing directly on the hierarchical classifier. As with other wrapper methods, this method is intuitive and should provide a good measure of the performance [13].

The second method combines the accuracy estimates of each individual base classifier to estimate the accuracy of the tree. Because wrapper methods can be computationally expensive, filter measures can be beneficial if the classifier requires training or the dimensionality of the data set is high [16]. There are a number of common filter-based measures that can be used to assess accuracy. However, these measures are not commonly used for multiple-classifier systems and cannot be directly applied to a tree structure with multiple base classifiers each using different feature sets. Instead, the second method evaluates the accuracy of each base classifier individually and then combines the estimates into an estimate for the entire tree. This allows the use of simpler filter-based measures that can be calculated only on a single base classifier using a single feature set.

When using the second method for estimating accuracy, the number of correctly classified points in a class is estimated by multiplying the total number of points in that class in the training set by the estimated accuracy of each base classifier that is used to classify that class. These numbers are then used to estimate the overall accuracy, as shown in equation 7.

$$A_{all} = \frac{1}{N} \sum_{c=1}^{C} N_c \prod_{i \in B_c} A_{ic} \tag{7}$$

where A_{all} is the estimated accuracy of the entire tree, B_c is the set of base classifiers that have class c as an output, and A_{ic} is the estimated accuracy of base classifier i on class c.

The accuracy estimates for the scores are calculated using various measures including a validation set on the entire tree, cross validation on the individual base classifiers, Fisher's interclass separability criterion (see Section II-B1), RELIEF (see Section II-B2) and a new metric termed count-based RELIEF, described below. A scaled version of Fisher's interclass separability criterion is also tested, where the criterion is divided by the square root of the number of features.

Count-based RELIEF is a newly proposed measure that is related to the RELIEF measure. Instead of measuring the distance between the nearest hits and misses, count-based RELIEF counts the number of hits before the nearest miss. While RELIEF attempts to identify features that maximize the average distance between points from different classes and minimize the average distance between points in the same

class, the count-based RELIEF attempts to find features that cluster points such that points are close to a large number of points in their own class, regardless of the absolute distance. The new measure has the advantage of being bounded and normalized, even for a non-normalized data set. This is important since not all base classifiers classify the entire data set, which can result in base classifiers with non-normalized features.

The new measure is calculated as:

$$r_c = \frac{1}{N_c(N_c-1)} \sum_{p=1}^{N_c} \sum_{i=1}^{N_c} t(x_{pc}, x_{ic}) \tag{8}$$

and

$$t(x_{pc}, x_{ic}) = \begin{cases} 1, & \text{if } d(x_p, x_i) < M_p, i \neq p \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

The score also includes penalty terms for the number of layers in the classifier, the number of features used by the tree and the sum total number of features used by the base classifiers. The score is given as:

$$S = A_{all} - \alpha_t \frac{F_t}{F} - \alpha_b \frac{F_b}{F(C-1)} - \alpha_l \frac{N_l}{C-1} \tag{10}$$

where S is the score, F_t is the total number of features used by the entire tree, F_b is the total number of features used by the base classifiers, N_l is the number of layers being used, $C-1$ is the maximum number of base classifiers and layers in the tree, and the terms α_t, α_b and α_l are the penalty terms.

Here, a distinction is made between the number of features used by the entire tree, and the total number of features used by the base classifiers. The term F_t counts the number of features used by the base classifiers, but if more than one classifier uses the same feature, it is counted only once. This gives the total number of features that will need to be calculated to use the classifier. The term F_b totals the features used by the base classifiers and this penalizes against classifiers with a large number of features.

E. Genetic algorithm evolutionary operators

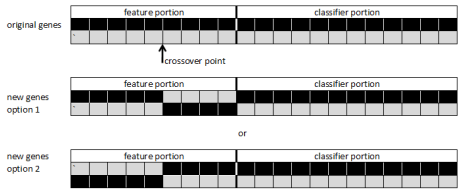
Because a small change to the tree portion of the gene can result in a large change to how the tree is built, a simple one-point crossover function is not the ideal way to breed genes.

The function randomly selects to either change the feature or the gene portion of the tree in each breeding. If the feature portion is selected, the algorithm performs a one point crossover in the feature portion, and randomly attaches each new feature portion to the second portion of each gene. If the classifier portion is selected, the algorithm performs a one point swap, as illustrated in Fig. 6b. Mutations can affect a point either part of the genome.

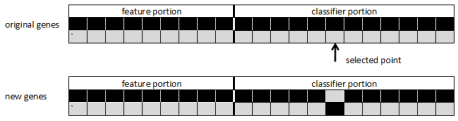
IV. Experiments

The genetic algorithm is coded in C++, and the performance measures are evaluated in MATLAB by calling the MATLAB engine from the C++ program.

The tree structures generated by this algorithm are all compared against a flat classifier, which is single base classifier



(a) crossover in the feature portion on the genome



(b) crossover in the classifier portion on the genome

Fig. 6. Crossover functions used in this work

TABLE I
TESTED PERFORMANCE MEASURES

Performance measure	flat or tree?	estimated on tree (method 1) or base classifiers (method 2)
all features	flat	N/A
validation set	flat	N/A
validation set	tree	entire tree
cross-validation	tree	base classifiers
Fisher's	tree	base classifiers
Fisher's (scaled)	tree	base classifiers
RELIEF	tree	base classifiers
count-based RELIEF	tree	base classifiers

TABLE II
DESCRIPTION OF DATA SET 1, WHERE x IS THE% OF NOISE (0% OR 5%)

Class	Feature 0	Feature 1	Feature 2	Feature 3	Feature 4
0	0 + x%	0.05 + x%	1 - x%	noise	noise
1	0.05 + x%	0 + x%	1 - x%	noise	noise
2	noise	noise	0 + x%	0 + x%	0.05 + x%
3	noise	noise	0 + x%	0.05 + x%	0 + x%

that separates all of the classes. A number of different filter and wrapper based measures are tested and compared with feature selected flat classifiers and flat classifiers using the entire set of features. Feature selection for the flat classifiers is also performed using genetic algorithms. A summary of all the tests is given in Table I.

A. Data Sets

The hierarchical classifier is tested on two artificial data sets. The data sets are divided randomly into test and training sets and all results are reported on the test sets. When validation sets are used for scoring, the validation portion is 20% of the data set, with 60% of the data set used for training, and 20% used for testing. In all other cases, 80% is used for training and 20% is used for testing.

1) Data set 1: The first data set is a four class data set, with five features. Feature two separates classes zero and one from classes two and three. Features zero and one separate classes zero and one and are noise otherwise. Features three and four separate classes two and three, and are noise otherwise. A second version of the data set includes 5% uniform, random noise. The data set is described in Table II.

2) Data set 2: The second data set has four features and four classes. Feature zero separates the classes fully and the

TABLE III
DESCRIPTION OF DATA SET 2 WITH NOISE, WHERE x IS THE AMOUNT OF NOISE (0%, 5% OR 20%)

	Feature 0	Feature 1	Feature 2	Feature 3
Class 0	$0 \pm x\%$	noise	noise	noise
Class 1	$0.25 \pm x\%$	noise	noise	noise
Class 2	$0.5 \pm x\%$	noise	noise	noise
Class 3	$0.75 \pm x\%$	noise	noise	noise

TABLE IV
RESULTS FOR DATA SET 1 WITH 0% AND 5% NOISE. THE COLUMN "# FEATURES (NO REPEATS)" GIVES THE TOTAL NUMBER OF FEATURES USED IN THE HIERARCHICAL CLASSIFIER (F_t). IF THE SAME FEATURE IS USED BY MORE THAN ONE BASE CLASSIFIER, IT IS COUNTED ONLY ONCE IN THIS COLUMN. THE COLUMN "# FEATURES (REPEATS)" SUMS THE TOTAL NUMBER OF FEATURES USED BY ALL THE BASE CLASSIFIERS (F_b). IF A FEATURE IS USED BY MORE THAN ONE BASE CLASSIFIER, IT IS COUNTED EACH TIME IT IS USED.

	Accuracy (in %)		# features (no repeats)		# features (repeats)		# base classifiers	
Test	0%	5%	0%	5%	0%	5%	0%	5%
All features	67.7	68.3	5	5	5	5	1	1
Features only	96.2	86.8	2	2	2	2	1	1
Validation set	100	100	2	3	2	3	2	2
Cross-validation	98.7	100	2	3	2	3	1	2
RELIEF	100	85.0	5	5	9	7	3	2
Fisher's	100	81.2	5	5	9	7	3	2
Fisher's (scaled)	100	57.5	5	1	5	1	3	1
Count RELIEF	100	100	3	5	3	5	2	2

remaining features are noise. This data set is also tested with 5% and 20% noise on feature zero. With 20% noise, the data set is not fully separable even when using only feature zero. The data set is illustrated in Table III.

The first data set tests the ability of the system to find a tree-based structure. The second data set tests the ability of the system to detect that a single level classifier is sufficient, and also tests how it performs in the presence of noise. Neither data set is fully separable by a single level (flat) KNN classifier. Data set one contains five partially informative features, and data set two contains three irrelevant features and one relevant feature that can be used to separate the classes.

B. Genetic algorithm parameters

An initial population of 500 is used, with 50 individuals being replaced in every generation. The algorithm is allowed to run for 70 generations.

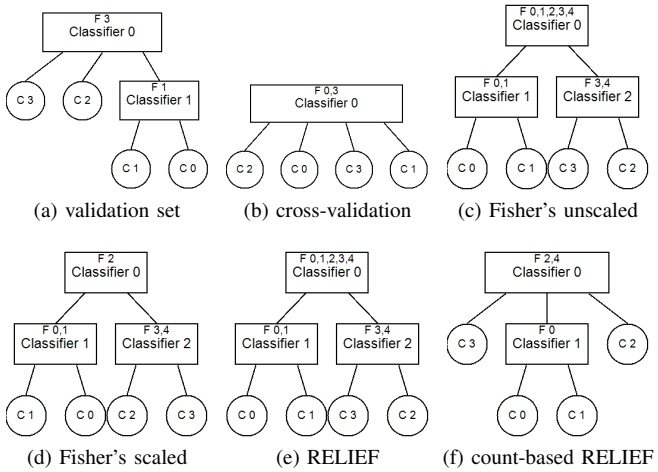
V. RESULTS AND DISCUSSION

A. Results from data set 1

The results for data set one are presented in Table IV. The hierarchical structures are shown in Fig. 7 and 8.

The accuracy of a flat classifier with no feature selection is fairly low for this data set. The distance separating different classes is relatively low compared to the noise, which is difficult for a KNN classifier. Feature selection alone raises the accuracy significantly for the data set without noise. For the data set with noise, the gain of feature selection is not as significant.

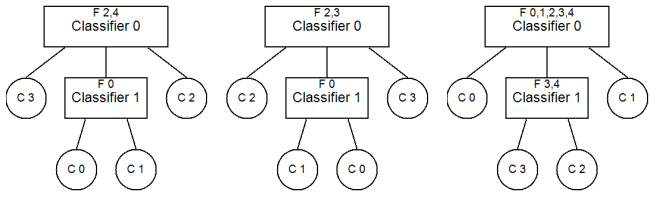
The hierarchical classifiers are able to give the best accuracy. However, the different accuracy estimate methods produce quite different results. The validation set appears to produce good results. In the data set with no noise, the classifier generated using cross-validation collapses down to the same single-level classifier generated with feature selection alone.



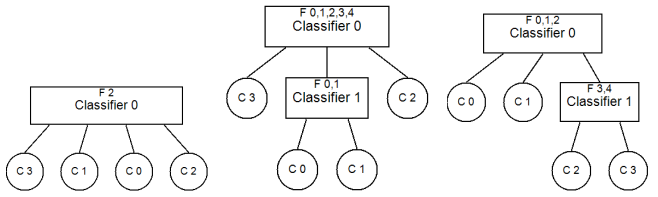
(a) validation set (b) cross-validation (c) Fisher's unscaled

(d) Fisher's scaled (e) RELIEF (f) count-based RELIEF

Fig. 7. Hierarchical classifiers generated for data set 1 with no noise



(a) validation set (b) cross-validation (c) Fisher's unscaled



(d) Fisher's scaled (e) RELIEF (f) count-based RELIEF

Fig. 8. Hierarchical classifiers generated for data set 1 with 5% uniform random noise

For the data set with no noise, scaled Fisher performs quite well, and actually selects the same tree as was originally envisioned (see Fig. 7d). Unfortunately, it does not perform nearly as well in the data set with additive noise, selecting only a single feature and a flat classifier. It is likely that the scaling is disproportionately large and the accuracy gain from adding additional features is outweighed by the scaling factor. Scaling Fisher's to the maximum possible size is clearly not a good way to discourage the use of additional features. It is likely better to simply re-tune the scoring parameters. However, this does mean that the algorithm will need to be re-tuned for different data sets to give good results.

The unscaled version of Fisher's criterion selects a similar tree to the scaled version for the noise-free data set, but selects all five features for the top level of the tree (see Fig. 7c). Although the tree is able to attain 100% accuracy on the test set, the selected tree is not ideal, as it uses a large number of features and also has an increased chance of a mis-classified

TABLE V

RESULTS FOR DATA SET 2 WITH 0%, 5% AND 20% NOISE ON FEATURE 0. THE COLUMN "# FEATURES (NO REPEATS)" GIVES THE TOTAL NUMBER OF FEATURES USED IN THE HIERARCHICAL CLASSIFIER (F_t). IF THE SAME FEATURE IS USED BY MORE THAN ONE BASE CLASSIFIER, IT IS COUNTED ONLY ONCE IN THIS COLUMN. THE COLUMN "# FEATURES (REPEATS)" SUMS THE TOTAL NUMBER OF FEATURES USED BY ALL THE BASE CLASSIFIERS (F_b). IF A FEATURE IS USED BY MORE THAN ONE BASE CLASSIFIER, IT IS COUNTED EACH TIME IT IS USED.

	Accuracy (in %)			# features (no repeats)			# features (repeats)			# base classifiers			correct feature?		
Test	0%	5%	20%	0%	5%	20%	0%	5%	20%	0%	5%	20%	0%	5%	20%
All features	92.1	93.4	62.3	4	4	4	4	4	4	1	1	1	N/A		
Features only	100	100	74.0	1	1	1	1	1	1	1	1	1	Y	Y	Y
Validation set	100	100	58.2	1	1	4	1	1	5	1	1	2	Y	Y	N
Cross-validation	100	100	66.3	1	1	2	1	1	3	1	1	2	Y	Y	N
RELIEF	100	100	74.0	1	1	2	3	2	3	3	2	2	Y	Y	N
Fisher's	100	100	74.0	1	1	1	3	1	1	3	1	1	Y	Y	Y
Fisher's (scaled)	100	100	70.0	1	1	1	3	1	1	3	1	1	Y	Y	Y
Count RELIEF	100	100	78.0	1	1	1	1	1	2	1	1	2	Y	Y	Y

point in the top level of the tree. In the data set with noise, the tree selected is similar to the tree selected for by RELIEF. In both cases, the tree is fairly reasonable, separating two classes in the top level and the other two classes in a lower level (see Fig. 8c and 8e). However, in both cases more features than necessary are selected in the top level of the tree.

The count-based RELIEF measure performs well on both versions of this data set, creating the same tree as the validation set in the non-noisy data set, and adding two additional features in the noisy data set that increase the robustness of the classifier. (see Fig. 7f and 8f). In fact, using a validation set or count-based RELIEF, the algorithm was able to obtain a more compact classifier than was originally envisioned (see Fig. 7d). These classifiers separate two classes completely in the first layer and use only two base classifiers total, as pictured in Fig. 7a and 7f. This tree uses both fewer base classifiers and fewer features in the noise-free case, and would be faster on average.

B. Results from data set 2

The results for data set two are presented in Table V.

Fisher's criterion is able to find the correct single-level tree for the data sets with 5% and 20% noise. However, for the noiseless data set, Fisher's separates the outputs into three base classifiers that all use the same feature. For the 5% and noiseless data sets, RELIEF also splits the classifier into two or three base classifiers, all using the same features. It is unable to find the correct feature for the 20% data set. The count-based RELIEF measure is able to find the appropriate one level classifier for the separable data sets, but splits the 20% noise set into a two level tree.

Since the only feature used in all these trees is feature 0, the division is unnecessary for the 1-NN classifier. However, for a classifier with a finite capacity, splitting the data set in this manner may actually provide a benefit, by allocating more resources to classes that are more difficult to classify.

In all the tests, the choice of performance measure affects the tree and feature set selected. This indicates the importance of selecting an appropriate performance measure.

VI. CONCLUSIONS AND FUTURE WORK

The proposed algorithm is capable of finding a feature-selected hierarchical classifier that can outperform a single level classifier. It can also find an appropriate single level classifier, although it has some difficulty with inseparable sets.

The performance measure is an important consideration and affects the results. The newly proposed count-based RELIEF works well, creating similar trees to the validation set in most cases and outperforming it on the inseparable set.

The next stage of this work will test this algorithm on more varied and real-world data sets, and compare this method with more existing feature selection methods. Future work will also investigate the effect of the performance measure and will focus on finding a classifier-specific performance measure that can accurately estimate the performance of a classifier using a specific feature set. This will allow this work to be extended to use more complex classifiers where it is unrealistic to use a wrapper-based performance measure.

REFERENCES

[1] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.

[2] J. Mao, K. Mohiuddin, and A. Jain, "Parsimonious network design and feature selection through node pruning," in *Proc. 12th IAPR Intl. Conf. on Pattern Recognition*, vol. 2, oct. 1994, pp. 622 –624.

[3] H. Liu, L. Liu, and H. Zhang, "Boosting feature selection using information metric for classification," *Neurocomputing*, vol. 73, no. 1-3, pp. 295 – 303, 2009.

[4] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1-2, pp. 273 – 324, 1997.

[5] M. Kudo and J. Sklansky, "Comparison of algorithms that select features for pattern classifiers," *Pattern Recogn.*, vol. 33, no. 1, pp. 25 – 41, 2000.

[6] P. Pudil, J. Novovicova, and J. Kittler, "Floating search methods in feature selection," *Pattern Recogn. Lett.*, vol. 15, no. 11, pp. 1119 – 1125, 1994.

[7] D. Zongker and A. Jain, "Algorithms for feature selection: An evaluation," *Proc. 13th Intl. Conf. on Pattern Recogn.*, vol. 2, pp. 18 – 22, 1996.

[8] S. Nakariyakul and D. P. Casasent, "An improvement on floating search algorithms for feature subset selection," *Pattern Recognition*, vol. 42, pp. 1932 – 1940, 2009.

[9] I. A. Gheyas and L. S. Smith, "Feature subset selection in large dimensionality domains," *Pattern Recognition*, vol. 43, no. 1, pp. 5–13, 2010.

[10] A. Webb, *Statistical pattern recognition*. Arnold, 1999, pp. 104–105.

[11] K. Kira and L. Rendell, "The feature selection problem: traditional methods and a new algorithm," in *AAAI-92. Proceedings Tenth National Conference on Artificial Intelligence*, Menlo Park, CA, USA, 1992, pp. 129 – 34.

[12] I. Kononenko, "Estimating attributes. analysis and extensions of relief," in *Proc. European Conf. Mach. Learn.*, vol. 784, Catania, Italy, 1994, pp. 171 – 171.

[13] N. Sanchez-Marono, A. Alonso-Betanzos, and R. Calvo-Estevez, "A wrapper method for feature selection in multiple classes datasets," *Proc. 10th Intl. Work-Conf. on Artificial Neural Netw.*, vol. 1, pp. 456–463, 2009.

[14] S. Cheong, S. H. Oh, and S.-Y. Lee, "Support vector machines with binary tree architecture for multi-class classification," *Neural Information Processing*, vol. 2, no. 3, pp. 47–51, 2004.

[15] F. O. Karray and C. DeSilva, *Soft Computing and Intelligent Systems Design*. Pearson Education, 2004.

[16] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1289–1305, 2003.