Machine Learning manuscript No.
(will be inserted by the editor)

Automated Hierarchical Construction of Multiple Classifier Systems

Matthew Leigh Fudge · **Massimiliano Pontil** ·
Giorgio Valentini

Received: date / Accepted: date

Abstract A new approach to Ensemble creation, the Clustered Tree Ensemble (CTE), and a number of variations of it are proposed. This technique combines classifiers in a tree structure driven by cluster analysis of the classification accuracy of each of the candidate classifiers. The tree structure reflects the similarity of classes in the training data. This technique is compared empirically with a number of recognised Ensemble models including Bagging, Boosting, and Stacking using a variety of constituent classifier types on the data sets described. The results of this comparison indicate that the CTE model is competitive with the recognised Ensemble models.

Keywords classification · regression tasks · Classifier Ensembles · Ensemble techniques · improving learners · new ensemble

1 Introduction

All instances of classifiers have their own individual strengths and weaknesses: some are too general (they are not precise enough), some are too specialized (they lack the ability to generalise well to unseen data), while others for a number of reasons are not able to adequately map the problem domain. These properties of classifiers are affected by the classifiers' own parameters, for example the training termination criteria, the dimensional scaling used etc, whose relationships to the behaviour of the classifier are not, in general, well understood. Research in the field (Clemen 1989; Drucker *et al.* 1993; Freund and Schapire 1996; Merz 1999; Quinlan 1993; Xu *et al.* 1992; Yu *et al.* 1997) has been shown, through experimentation and theory, that by combining the choices made by multiple classifier instances into

Matthew Leigh Fudge
UCL Department of Computer Science, London
Massimiliano Pontil
UCL Department of Computer Science, London and
Istituto Italiano di Tecnologia, Genova
E-mail: m.pontil@cs.ucl.ac.uk
Giorgio Valentini
DI, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano
Tel.: +39-02-503-16225
Fax: +39-02-503-16373
E-mail: {valentini}@di.unimi.it

one decision, the reliability, accuracy and a number of other properties of the solution can be improved over the results obtained from using a single classifier. There are, however, few methodical approaches to the construction and training of such classifier combinations which can construct an appropriate structure for a given problem automatically. In other words these classifier ensembles are constructed without regard for the properties, such as class relationships, that are embedded in the problem domain.

This thesis proposes a new approach to ensemble construction, which is called the Clustered Tree Ensemble (CTE). The research examines a number of methods that utilise inter-class relationships during the construction of an ensemble learner. The tree-like structure that is generated exposes these relationships, allowing easier human interpretation of the problem domain and the limitations and abilities of the constituent classifiers used.

Breiman (Breiman 1996) wrote with reference to Bagging Trees: "what one loses, with the [bagged] trees, is a simple and interpretable structure. What one gains is increased accuracy".

In addition to the interpretability that CTE's structure brings, its tendency to place similar classes in close proximity to one another within the tree structure shows the potential for its use as a technique to generate hierarchical indexes for easy human searches of large media databases.

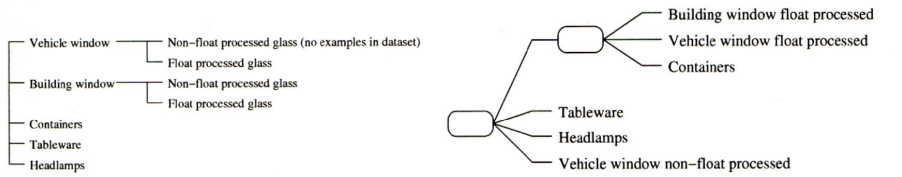
The Clustered Tree Ensemble superficially shares a number of similarities with other tree algorithms developed in the past few years, such as Option Trees (Buntine 1992; Kohavi and Kunz 1997). In some ways the CTE could be seen as a generalisation of Option Trees that allows complex classification algorithms to be used as decision nodes. However unlike the Option Tree approaches its prime goal is to build a tree based on clusters found in classification results from decision nodes within the tree. This allows the tree structure to be adapted to the abilities of its constituent classifiers.

The CTE and a number of its variations are compared to a variety of known ensemble methods with respect their classification accuracy on a range of data sets. The data sets used for these comparisons cover a wide spectrum of complexity, from simple domains with few classes and small feature vectors to problems with many tens of classes and hundreds of features.

1.1 Outline of research

This is an investigation of a set of tools developed recently for classification and regression tasks, namely Classifier Ensembles. The aim is to place a selection of these Ensemble techniques in context with the hope of improving our understanding of their approaches, advantages and disadvantages. The thesis first considers a number of common classifier ensemble systems to provide a baseline from which to compare the proposed Clustered Tree Ensemble (CTE). This involved:

- Firstly examining a set of learners taken from the Machine learning, Statistical and Connectionist fields of classification which is used as the constituent classifiers in Ensemble creation.
- Followed by examining some standard Ensemble learner models with emphasis on the choices made in the design of each and the functional blocks they use.
- Then describing the data sets that are used in the comparison which includes two created specifically for this thesis. An optimised version of the Spatial grey level dependence texture analysis method that developed from the creation of one of these data sets is proposed and examined.



(a) Explicit associative classification

(b) Implicit associative classification example. Non-float processed glass from buildings is removed as there are no examples of it in the data set.

Fig. 1: *Approaches to associative classification.*

– The results show a reduction in the memory requirements and a significant improvement in execution speed for images with a large number of grey levels and for third or higher order texture measure computations.

2 The Clustered Tree Ensemble

In a number of situations, such as the creation and querying of large digital libraries, it is advantageous to organise the data entries into semantic categories. These categories allow a user to browse the data base with relative ease by providing a concept of similarity, or localization to entries even if they are not of the same class (Huang *et al.* 1998a; Lienhart and Hartmann 2002; Wang *et al.* 1999).

There are two kinds of associative classification: explicit and implicit. Explicit associations are those specified for a category from an external source. The original Glass database shown in figure 6.1 (a) can be described as an explicitly associative classification problem. It contains a number of class labels that are associated in some way to create, in this case, a tree-like structure. There is, however, not necessarily a relationship between the structure of the branch groupings in this hierarchy and any detectable patterns in the feature vectors associated with the class labels. In other words the associative nature of the class designators is defined explicitly as data that are external to the problem domain and which cannot in general be derived from the problem data-set. In the case of the Glass data-set the explicit information used to create the hierarchy consists of the function for which the glass is intended (for decorative use or for use in building or automotive windows) and the method of production (float or non-float) both of which are logical to a human observer but are not necessarily derivable from the data-set which consists of the chemical composition of the glass. It is possible for example that Automotive float processed-glass and Building float-processed glass are chemically identical while their functional uses are substantially different.

Unlike explicit associations, implicit associations, often called semantic associations, are derived from patterns occurring in the data set of a problem domain. Figure 6.1 (b) shows a fictional associative tree generated from the Glass data set.

This example shows an implicit association between float-processed glass from buildings and vehicles. This association may have been predictable from the class labels for the problem but did not show up in the explicit association provided in the class labels (Figure 6.1 (a)). Note that glass fragments from “Containers” have also been grouped with float-processed glass due to their similar chemical composition.

The Clustered Tree Ensemble Classifier (CTE) presented here is a supervised learning method that finds implicit associations between the user defined class labels of a problem domain. It represents these relationships in a hierarchical structure that is derived from analysis of the performance of constituent learners. A hierarchical structure was chosen to represent the implicit associations for three reasons:

- It simplifies human interpretation of the semantic associations when compared to other methods (Bradshaw 2000; Huang 1998) such as general graphs.
- It facilitates human navigation and browsing of the resultant database (Chang *et al.* 1997).
- It allows the ensemble's constituent classifiers to be "focused" on subsets of the class labels taken from the problem domain, which can improve the individual learners' performance.
- It can be implemented in parallel with resonable efficiency. It is relatively trivial to train each sub-classifier independently under the constraints of the tree structure (a sub classifier cannot be trained until its parent has been). This allows an ideal parallel impementation to construct the CTE in time proportional to the depth of the tree. Bagging by comparison can be fully parallelised as there is are no dependencies between the constituent classifiers. Adaboost is not (at the level of granularity under consideration) parallelisable at all, as every constituent classifier trained is dependent on the results of the previous one.

The CTE scheme attempts to utilise a number of concepts taken from studies of classifier combinations and ensemble learners (chapter 3.2) to enhance its performance.

Once a CTE is trained and built it allows unlabelled feature vectors to be assigned to their correct semantic category (branch of the tree) with the knowledge that incorrect classifications will be "near" the correct classification in the hierarchical structure. This concept of locality is important for human browsing of the newly classified features, as it allows incorrectly classified entries to be located in close proximity to the correct category.

2.1 Algorithm Description

The process of building/training a Clustered Tree Ensemble (CTE) starts by splitting the data-set into two subsets called the Training and Verification data-sets. The verification set is withheld from the training process completely and is used only to assess the performance of the finished CTE. The Training data-set is passed to the CTE algorithm where it is again split into two sub-sets that are given the names Train and Test. A base-classifier is then trained using the Train data-set and its performance assessed using the Test set. This assessment produces an accuracy for the classifier as would normally be obtained when checking the result of a Test run. It also produces a confusion matrix by determining how many samples of each class were correctly classified and how many were mistakenly classified as each of the other classes. The accuracy is assigned to the classifier node and can later be used to weight the contribution of this node when combining the outputs from all classifier nodes in the CTE.

Line 5 in algorithm 4 uses the confusion matrix to create a complete undirected weighted graph (confusion graph) by examination of pairwise confusion between classes. Construction of the confusion graph is not strictly necessary, but it does provide a common data format in which confusion data can be passed to the clustering algorithms.

CTE algorithm

1. Create a new Learner.
2. Split data set into two (Train and Test).
3. Train the Learner on the training set.
4. Create confusion matrix using testing set.
5. Build confusion graph from the confusion matrix.
6. Perform cluster analysis on confusion graph (produces clusters of classes that are strongly confused).
7. For each cluster that contains more than one class create new sub-classifier (node in the tree) to classify each class in that cluster as well as a place holder class called "*Other*" that represents all classes that this learner does not categorise.
8. Build new output mapping for this classifier.
9. Build new input-to-output mappings for each sup classifier.
10. If hierarchy contains untrained classifiers go to 3.

Fig. 2: *High-level schme of the CTE algorithm.*

For any two classes A and B, $offDiagAB$ and $offDiagBA$ are counts of the number of examples of A incorrectly classified as class B and the number of examples of B incorrectly classified as class A respectively this is normalised by the total number of samples of class A and class B that were processed, $totalAB$. The weight assigned to the edge connecting vertices A and B in the new graph is then:

$$\frac{offDiagAB + offDiagBA}{totalAB} \tag{1}$$

The next step in the CTE construction process is to analyse the confusion graph to find clusters of classes that are confused and therefore similar in some way. The decision to build a undirected graph of confusions allows any of large number of graph theory clustering techniques to be used. Using a different form to represent the inter-class confusions would make the use of statistical or other kinds of clustering techniques more appropriate. The graph theory clustering modules are discussed in more detail in section 2.1.1. The clustering module returns a set of graphs, each of which corresponds to a group of classes that the current base-classifier can not distinguish clearly between. Each cluster that contains a single class implies that the class is not highly confused with any other and needs no further work. For each of these singleton clusters a new *terminal node* is added to the CTE as a child of the current *classification node* and labelled with the name of the class it contains. These terminal nodes are not used when the CTE is classifying unseen examples but do provide a visual representation of the classes that a particular classifying node handles well.

For each cluster that contains more than one class vertex a new classification node is added as a child of the current classification node in the CTE. For each of these new classification nodes a new version of the Training data-set is created by random re-sampling of the original Training data-set.

2.1.1 Clustering Modules

Clustering can be loosely defined as the process of organizing objects into groups whose elements are in some way similar. There are two common approaches: partitioning in which each object is assigned to only one group, and hierarchical clustering in which each group (size ¿ 1) consists of smaller groups. Within each of these approaches lie tens if not hundreds

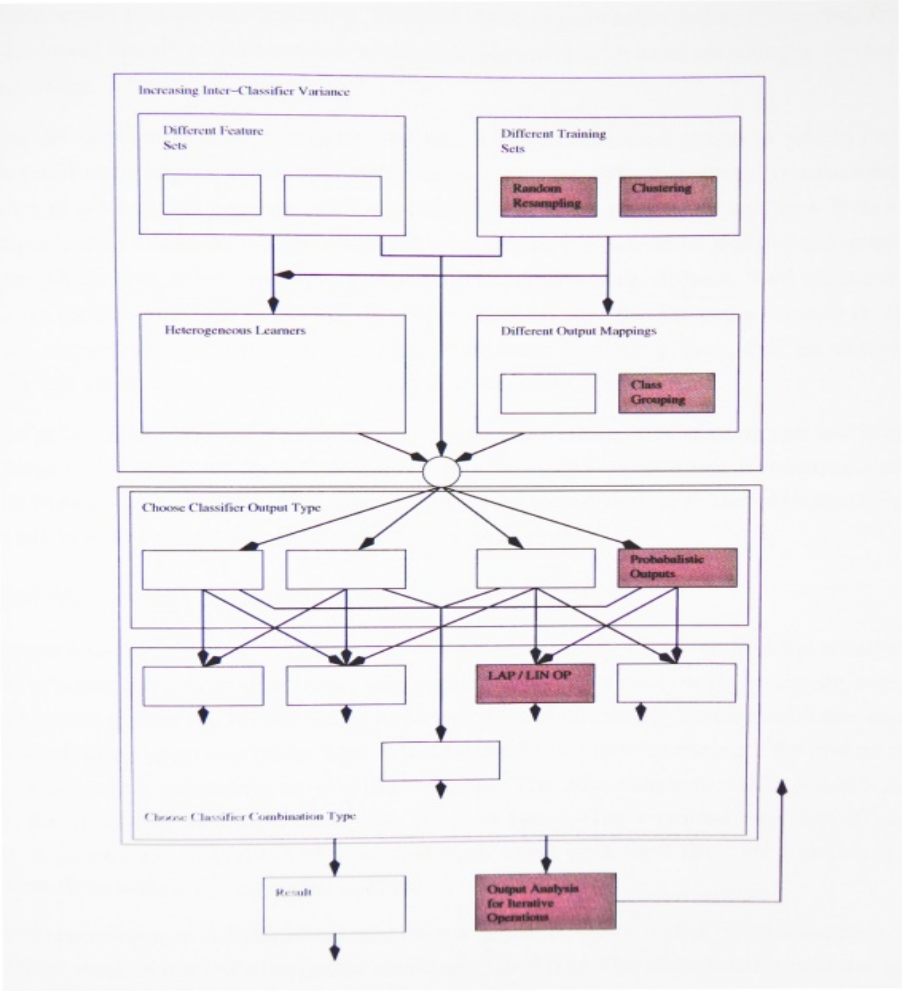


Fig. 3: *Overview of the Clustered Tree Ensemble design choices. The diagram shows that random re-sampling and clustering are used to create diversity in the training data set, while class grouping based on the clusters is used to create variations in the output mappings of individual classifiers.*

of techniques from a wide variety of subject areas. A useful starting point for deciding which clustering algorithm to use is Jain *et al.* (1999) who present a survey of classical clustering algorithms. For a more recent review, see e.g. Xu and Tian (2015).

The data from the confusion matrix is extracted into a fully connected graph in which each vertex represents a class and each edge a confusion between two classes. The computed confusions are assumed to be symmetrical during this process, such that Class A is mis-catagorised as Class B as often as Class B is mis-catagorised as Class A, which while not true allows the use of an undirected graph to represent the confusions which simplifies implementation of the clustering algorithms. Two clustering techniques explored in this section that are easily applicable to data in this format are a variant of the Minimum Span-

ning Tree algorithm called the Thresholded Maximum Spanning Tree, and an extension to Minimum Capacity cut algorithms called the Recursive Minimum Capacity Cut.

The goal is to split the graph into a set of graphs were the members of each graph are highly confused in that they have high values on the edges connecting them. To achieve this a minimum capacity cut is required. The Maximum Spanning Tree algorithm is used in preference to the Minimum Spanning Tree algorithm to achieve this result as explained in the next section.

Thresholded Maximum Spanning Tree A simple approach to finding approximations to minimum capacity cuts is to find the minimum spanning tree and post process the graph to remove edges according to some criteria to create separate graphs. This however would ignore the higher value links that would potentially have joined the seperate graphs together. An alternative approach taken here is to find the Maximum Spanning Tree and as the algorithm progresses discard edges according to a similar criteria. The advantages to this approach are that it can be built into the spanning tree algorithm with no post processing required and that if vertices in the resulting set of graphs are linked via at least one high value path then they are transitively part of the same confusion from which the classifier suffers.

The minimum spanning tree (MST) of an undirected graph is the tree that, with least cost, connects all vertices together. An example of a weighted undirected graph and its associated minimum spanning tree is shown in Figure 4 . In this figure the underlying graph is shown in grey with the minimum spanning tree overlayed in heavy black.

A number of approaches exist that will efficiently find the MSP of a graph such as Prim's (Prim 1957) and Kruskal's (Kruskal 1956) algorithms. This implementation uses Kruskal's algorithm which is summarised in Algorithm 5. Kruskal's algorithm is a greedy technique, meaning that it takes the best choice at each step. The algorithm commences by creating a set of trees each rooted at a vertex in the the graph (7 trees in this example). Each of the edges is examined to find the one that has the lowest associated cost, with the additional criterion that its end points are in separate trees, which ensures that no loops are created. The edge found by this process is selected (edge value of 103 in Figure 4) and the two trees containing its end point vertices are merged together. This process is repeated by finding the next lowest cost edge (labelled 112) and so on (labelled 125, 146, 152, 201 in order).

Algorithm 5 Kruskal' Minimum spanning tree algorithm. TO BE REMOVED The edge with cost 168 to the top right of the figure is not selected as doing so would connect vertices A and E together when they are already linked through vertices B, C, and D (In other words vertices A and E are already in the same tree and therefore no further edge is needed to connect them). At each step the number of trees remaining in the forest (set of trees) must be reduced by one as a single tree is made by combining two unconnected ones. Termination of this process occurs when only one tree remains in the forest.

Kruskal's algorithm (Kruskal 1956) provides a useful starting point for a clustering algorithm. It is however limited in that it always terminates when only one tree remains rather than when an appropriate but arbitrary k trees remain. To allow the algorithm to be used to cluster the inter-class confusions a number of modifications were made. These modifications are as follows:

1. The first modification processes the data prior to performing Kruskal's algorithm and involves removing any edge the associated value of which falls below a specified threshold. Removing edges in this way means that multiple unconnected graphs are given to

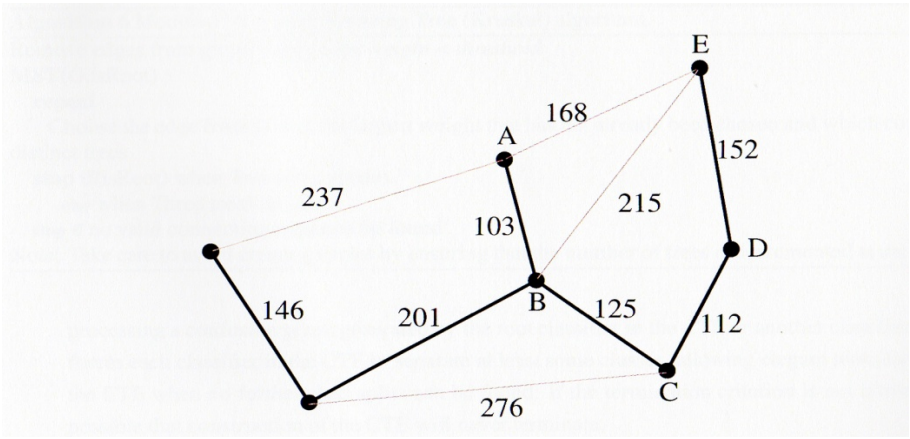


Fig. 4: *Minimum spanning tree (black) of a weighted undirected graph (grey)*

the MST algorithm causing it to terminate early as it cannot find edges that connect some of the sub-graphs (as there are none). The termination criteria of the algorithm must therefore be modified to terminate when one tree remains (as in the Kruskal implementation) or when all edges have been examined and no suitable one can be found. A second effect of this preprocessing is to speed up the computation of the MST as it can reduce the number of edges that need to be considered when merging trees in the main section of the algorithm. The threshold parameter is always in the range 0-1 and represents the level of inter-class confusion allowed. Higher values allow more inter-class confusion, lower values less.

2. The second modification is to the termination criterion used by Kruskal and forces the MST algorithm to terminate when 2 or 3 trees remain rather then one, depending on whether the algorithm is processing a confusion graph generated by the root classifier in the CTE or another classifier.This forces each classifier in the CTE to separate at least some classes, allowing elegant termination of the CTE when no further class splits can be found. If the termination criterion is not altered it is possible that construction of the CTE will never terminate.
3. The third modification is to the edge selection process and changes it such that the edge with the highest cost is selected rather than the edge with the lowest cost. This effectively changes the algorithm so that it finds the Maximum spanning rather then the minimum spanning tree. The Maximum spanning tree can only be found if we assume that loops are not allowed in the resulting trees. The modified algorithm therefore finds spanning trees that link together classes that are highly confused.

With these changes in place the modified algorithm (Algorithm 6) takes an undirected positive weighted graph that is not necessarily connected, along with a flag determining whether the algorithm is being executed on behalf of the root classifier node in the CTE or any other classifier node.

It returns a forest of trees in which each tree represents a cluster of classes that have been found to be confused beyond the threshold parameter. However while the algorithm is simple and fast it depends on linking clusters together based on the weights of single edges its output can therefore be quite sensitive to small variations in the input graph. It assumes that the highest value edge linking to graphs is representative of all the other edges that could

Modified Minimum Spanning Tree (Kruskal) algorithm.
Remove edges from graph where edge *weight* < *threshold* MST(G,isRoot) **repeat** Choose the edge from G with the largest weight that has not already been chosen and which connects distinct trees **stop if**(is Root) when Two trees remain, else when Three trees remain **stop if** no valid connecting edge can be found Note: Take care to avoid creating cycles by ensuring that the number of trees is decremented at each step.

Fig. 5: *High level scheme of the modified Minimum Spanning Tree algorithm*

have linked the same two graphs. If one edge gets an unusually high confusion weighting then two graphs can be joined together even though there is very little confusion between any other nodes in the two graphs.

Minimum Capacity Cut Clustering based on Kruskal's MSP algorithm is fast, however the results it produces are prone to large variation for small changes to the input graph. For this reason a second approach that provides more stability, at the cost of increased execution time, is also examined.

The cut of a graph is the set of edges that when removed splits it into two sub-graphs. The cut is minimum capacity if the edges removed to create it have the lowest totalled cost (capacity) of all the possible cuts that could be made. Computing the minimum capacity cut of a graph is one of the most intensively used basic tools in optimisation applications such as automatic graph drawing (Mutzel 1995), the travelling salesman problem (Junger *et al.* 1995), network reliability (Grotschel *et al.* 1995), and sequential ordering (Ascheuer *et al.* 2000).

A large number of techniques have been found for computing the minimum capacity cut of an undirected weighted graph (Gomory and Hu 1961; Goldberg and Tarjan 1988; Padberg and Rinaldi 1990; Hiroshi and Toshihide 1992; Hao and Orlin 1992; Nagamochi *et al.* 1994; Junger *et al.* 2000). The major difference between these approaches is the variation in execution time (computational complexity). The method used here was developed by Stoer and Wagner (1997) as a simplification of earlier work by Hiroshi and Toshihide (1992) and, if implemented using Fibonacci heaps, yields an expected complexity of $O(mn + n^2.logn)$ where n is the number of vertices and m the number of edges in the graph.

The Stoer Wagner approach to finding a minimum cut (see Algorithm 7) creates a set S of graph nodes that exist on one side of the proposed cut. This set is initially populated by a single randomly selected node from graph G. The nodes not in S are then ordered by the sum value of the edges linking them directly to nodes in S. The node with the maximum cut with respect to S is then added to set S. If two nodes exist in G the minimum cut has been found and can be returned, otherwise a new graph G' is created by shrinking the two vertices, v_n and v_{n-1}, that have the lowest cut with respect to S. This new graph is passed recursively to the Mincut algorithm. The algorithm terminates by comparing the cut it produced and the cut produced by calling Mincut onG' and returning the smallest.

The process of shrinking nodes, in a four vertex non-complete undirected graph, to computeG' from G is illustrated in Figure 6.4 (courtesy of Goemans and Papaefthymiou

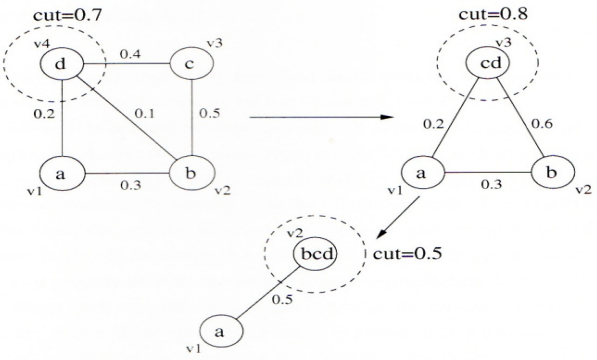


Fig. 6: *Illustration of the Mincut algorithm*

(1991)). The graph shown top-left is the original that is converted to the top-right representation by shrinking nodes *c* and *d*. This would occur if d was the initial vertex randomly selected by the Mincut algorithm. Vertex *a* is connected only to vertex *d*, therefore its edge weight remains the same when connected it to the composite (shrunk) node *cd*. Node *b* however is connected to vertices *c* and *d* and therefore the weights of both of these edges (0.1 and 0.5) need to be summed to produce the edge weight linking vertex *b* to the new composite vertex *cd*. When the Mincut algorithm is called on the new graph G' it randomly selects the new composite node *cd* as its starting node. This implies a minimum cut of 0.8 but as there are more than two nodes remaining in G' the node with the largest edge weight with respect to vertex cd is chosen to create a new graph G'' prime\prime}prime\prime that can be used to call the Mincut algorithm again. The lower graph is the final call to Mincut as the graph now only contains the minimum required two nodes. Therefore it returns the current cut (value 0.5) to its caller. The caller then compares the value of the cut that was returned to it (0.5) with the value of the cut that it produced (0.8) and returns the lowest of these. At the next level up corresponding to the top-right graph in the illustration the comparison of cuts is again performed (0.5 (middle) versus 0.7 (top left)) and the algorithm exits, returning the minimum of these. The result is the Minimum capacity cut of graph *G*.

Recursive Minimum Capacity Cut The Minimum capacity cut algorithm described (Stoer and Wagner 1997) above finds the optimal cut that splits a graph into two sub-graphs. However the requirement for the Clustered Tree Ensemble is to subdivide the graph into *k* subgraphs, where *k* is optimal in some manner. To achieve this automated splitting a recursive algorithm is used that takes the cut produced by applying the Mincut algorithm to graph *G*, and uses it to partition *G* into two new Graphs *G*-left and *G*-right (*l* and*r* in algorithm 8) by removing the edges in the cut. The value of the cut produced by passing *G* to the Mincut algorithm is normalised by division by the number of edges that cross the cut. As all edges in *G* have weights in the range 0-1 this normalisation produces a figure that is also in the range 0-1. A similar scheme for normalisation of the cut value is proposed by Shi and Malik (2000) as a modification to the earlier image segmentation work of Wu. and Leahy (1993) who suggest that without such normalisation of the cut value the mincut algorithm is prone to pruning small clusters from the graph. If the normalised cut, t, is greater than a pre-set threshold value then the current cut is returned as the final cut otherwise the Recursive min-

Recursive Mincut algorithm.

```
Recursive Mincut (G, threshold,isFirst)
M = {}   // An empty set of graphs
N = [V(G)]  //number of vertices in G

if(n<2)
  return M = M ∪ {G}   // not enough vertices to subdivide
endif

c=Mincut(G)      // call Minimum capacity cut algorithm

t=sum(c)       // find the value of the cut

t = t/|E(c)|        // normalise the cut value by the number of edges in the cut list

if((t≤threshold) or (isFirst))
l = createGraph(G,c,"LEFT")     // create graph from vertices and edges in G to left of cut c
r = createGraph(G,c,"RIGHT")     // create graph from vertices and edges in G to right of cut c
M = M  ∪ {RecurseMincut(l,threshold,FALSE))
M = M  ∪ {RecurseMincut(r,threshold,FALSE))
else
M = M ∪  {G}
endif

return M
```

Fig. 7: *Pseudo code of the Recursive Mincut algorithm*

imum cut algorithm is called on the *G*-left and *G*-right graphs. The*isFirst* parameter is set to TRUE only on the first call to RecurseMinCut, guaranteeing that at least two sub-graphs will be produced. The algorithm returns a set M of k graphs, where k is determined by a combination of the threshold value and the weights of the edges in G. The threshold parameter, which is always in the range [0,1), allows cuts above the specified confusion level to be discarded as not of interest, causing less constituent classifiers to be created.

Algorithm 8 Both clustering methods produce a set of sub-graphs that are used to build the class list for nodes at lower levels in the CTE. Sub-graphs containing a single vertex result in the creation of a terminal leaf. Those sub-graphs with more than one vertex are used to create a new input-to-output mappings of the data-set that will be passed to new lower level classifier instances. Using an example presented earlier in this section, table 6.2 is produced from table 6.1 by application of one of the described clustering modules. A set of three sub-graphs has been produced by the clustering algorithm corresponding to classes *AC*,*B* and *D*. Two of these sub-graphs, *B* and *D*, contain a single vertex each and therefore result in terminal nodes in the Clustered Tree Ensemble, meaning that the current learner classifies them satisfactorily. The final sub-graph contains two vertices, *A* and *C*, and will be used to create a new data-set mapping that can be passed to a new lower level classifier in the CTE. This new data-set will map all instances of classes *B* and *D* to a new class named *Other* while data-set instances of classes *A* and *C* will be left unchanged.

Performance of Clustering Modules The primary constraint when deciding which clustering model to use is the time that they take to run on a given size graph. This becomes

Fig. 8: *Execution times of Maximum Spanning Tree and Minimum Cut clustering algorithms. Both algorithms have their threshold set to 0.0. Abscissa shows the number of vertices in the graph being processed.*

especially important as the number of classes in the problem domain, and hence the number of vertices in the graphs, increases. To show the difference in performance between the two clustering method implementations the following experiment was performed.

An algorithm was used that could create a complete undirected weighted graph with a given number of vertices, v. For values of v between 3 and 45 inclusive 10 graphs were created. Each of the clustering algorithms, the Maximum Spanning Tree and the Minimum Capacity Cut, was run on all ten graphs at each value of v. For these experiments the threshold value passed to the clustering algorithms was set to zero to produce the worst case performance. The graph in figure 8 shows the average results of the runs for each value of v.

The graph shows that the Maximum Spanning Tree (MST) cluster is significantly faster than the Minimum Capacity Cut (MCC) for all values of$v$ above 12. Below $v = 12$ the results are mixed, suggesting that the MCC algorithm should be used for its increased stability in clustering.

2.1.2 Re-mapping the Data Set

During construction of the CTE new classification nodes are added when confusion above a specified threshold is found to exist between classes. The new classification node is required to try to improve the classification performance of only those confused classes and not any others. To this end a new version of the Training data-set is created by modifying its input-to-output mappings according to the decisions made by its parent node. This however causes very large amounts of memory to be consumed, especially as the CTE becomes large. To reduce the memory and processing requirements when building the CTE the training data-set is never directly modified: instead it is accessed via a mapping. The mapping provides a bi-directional relationship between the classes in the original data-set and the classes that the classification node requires, as shown in Figure 9.

For performance reasons, the mapping consists of two Hash tables, one of which contains the classes from the training data-set and the second the classes required. To build a new input-to-output mapping for a classification node, the cluster of confused classes is examined and a new mapper created that contains all classes from the data set in its left hand hash table and all classes from the cluster in the right hand hash table. A one-to-one rela-

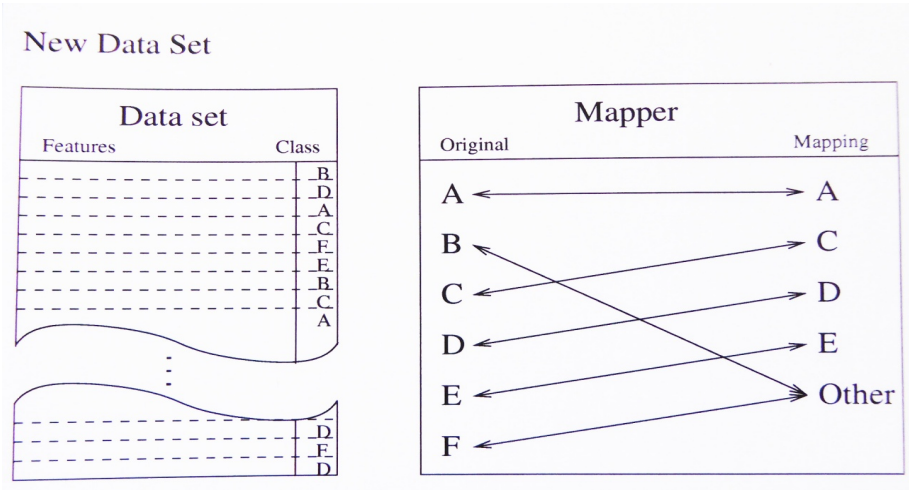


Fig. 9: *Re-mapping the Data-set*

tionship is created between the classes on the right hand side to their corresponding class on the left. A new class called *Other* is then added to the right hand side and related to all classes in the left hand table that are not represented in the right hand hash table. The *Other* label represents all classes that this classification node is not interested in distinguishing between. To handle the many to one mapping that is implied by the *Other* meta-class, the values inserted into the right hand hashtable are linked lists.

To get a re-mapped data sample the class assigned to the sample is looked up in the left hand table and the relation followed to the right hand table to discover its new identity. This operation takes an expected constant time but can degenerate to time proportional to the number of classes in the worst case.

2.1.3 Un-biased Data Set

Set During training, new sub-classifiers in the tree structure are provided with new class encodings or mappings of the data-set. This allows the sub-classifiers to focus on learning to distinguish between a set of highly confused classes rather than between all classes in the data-set. The*Other* meta-class that is introduced to the problem space of each sub-classifier contains all classes that the sub-classifier will not try to distinguish between. This introduces an imbalance in the number of instances of each class that the sub-classifier uses for training. For example, consider a problem domain consisting of nine classes with an equal number of examples of each. If a sub-classifier is told to specialize on distinguishing between two of these then the other seven classes are grouped into the *Other* meta-class. The *Other* meta-class therefore contains seven times more examples then either of the two classes that the classifier is supposed to focus on. This disparity, or bias, in the number of examples of each class can cause learning problems for some types of base-classifier, such as the Artificial Neural Network, and gets more extreme the deeper the CTE becomes. A variety of methods for coping with this situation exist, such as creating a Cost matrix for each learner in the tree or applying a weighting to the examples of each class. However the varying types of base classifier examined in this document are not consistent in which of these approaches, if any,

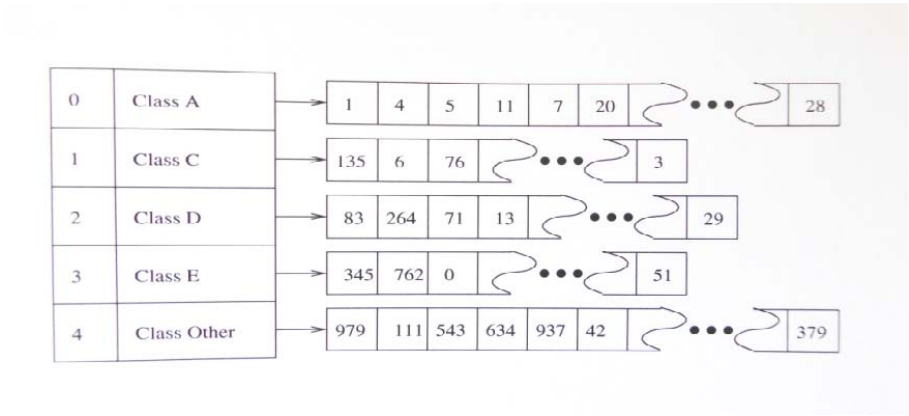


Fig. 10: *Remove bias from the Data-set*

they can handle. Therefore a technique of re-sampling the sub-classifier specific data-set was used, as this operates independently of the type of base-classifier chosen.

At each non-leaf node in the tree a version of the data-set is created that contains the unique output class mapping for that sub-classifier. This new re-mapped data-set contains the bias described above and as such is generally not suitable for training the classifier at this node. The re-mapped data set is therefore passed through a further filtering process to remove the bias in the number of examples of each class prior to its use in training the classifier node. This process involves re-sampling the data-set to ensure that an equal probability exists of randomly drawing any of the classes from it. The data-set produced by this re-sampling process contains the same number of examples as the original data-set. However, it contains some duplicates of the instances of classes that are rare in the input. To maintain the data-set lengths some randomly selected instances of classes that are common in the input data-set have to be removed from the output. Removing examples of classes that are common generally means removing examples of classes that are members of meta-classes such as *Other*.

Re-sampling such as that described here has two benefits. First, it removes the bias that can affect the learning process of some classifiers, and secondly it can promote diversity among the classifiers that constitute the ensemble, as each learns from a data-set that contains different examples.

The disadvantage of the simple system used here is that it does not retain the prior class probability of the original data-set which may in certain circumstances, such as the Zero-R learner, be relevant. A more complex method of removing the bias could be implemented that maintained the prior probability.

As with the process of re-mapping the data-set, performing the previously described algorithm for bias removal would result in additional data-sets being created for the training process, significantly increasing the memory used by the system. To avoid this problem a look-up system shown in Figure 10 is used, which consists of an array of arrays. In the first array shown on the left each index corresponds to a class. Each of the entries in this array point to a secondary array that contains the indices of all of the examples of that class in the underlying data-set. To produce a random example that comes with an equal probability from any of the classes a two stage process is used. First a class is chosen using a uniform random number generator. This selects a class with uniform probability. Next a

Expected Outcome						Error per class		
A	*B*	*C*	*D*			*A*	=	5
20	1	**3**	0	*A*	Produced Output	*B*	=	3
2	22	1	0	*B*		*C*	=	6
3	0	**19**	0	*C*		*D*	=	0
0	2	2	25	*D*		Total	=	14 = 14%
						Average error Per class	=	14/4 = 3.5 examples

Table 1: *Example confusion matrix and table of the number of sample inputs incorrectly classified per class. The confusion matrix shows the labelled class values horizontally and the class values produced by the classifier vertically.*

Expected Outcome					Error per class		
AC	*B*	*D*			*A*	=	3
45	1	0	*AC*	Produced Output	*B*	=	3
3	22	0	*B*		*D*	=	0
0	2	25	*D*		Total	=	6 = 6%
					Average error Per class	=	6/3= 2.0 examples

Table 2: *Confusion matrix produced by merging the classes A and C from Table 1 into meta-class AC. The confusion matrix show the labelled class values horizontally and the class values produced by the classifier vertically. Note the lower error per class, total error and average error per class.*

second independent random number is used to select an entry from the array associated with the chosen class. The index thus found is used to access the feature vector in the underlying data-set.

This system can produce slightly different results from those described in the previous approach, in that it does not explicitly remove feature vectors from classes that have a large number of examples and does not explicitly duplicate vectors from classes that consist of a small sample set. Duplication can however occur for base classifiers such as artificial neural networks that continually extract an indeterminate number of randomly selected samples.

2.1.4 Effects of Re-mapping Based on Clusters

One method of assessing the performance of a classifier is to examine the confusion matrix it produces when presented with a previously unseen data-set. The confusion matrix shows the number of examples of each class that were correctly classified. Of more interest in the current context is that the matrix also contains a count of the number of examples of each class that were incorrectly classified and the class to which they were erroneously assigned.

A classifier that performs well will therefore produce a matrix in which the majority of the entries are along the diagonal, showing a high level of correct classifications. Examination of these matrices produced by a number of classifiers on a variety of data sets shows that there is often a greater level of confusion between certain classes than others. The algorithm proposed here tries to find those groups of classes that have a relatively large inter-class

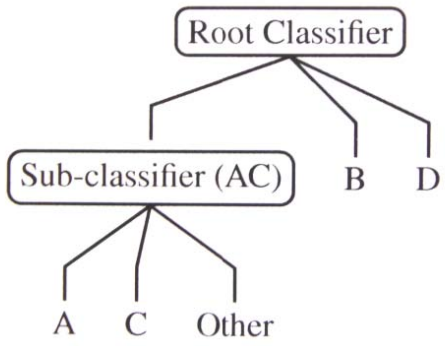


Fig. 11: *Representation of the tree created from the matrices in Tables 1 and 2.*

confusion, so that they can be merged together, thereby reducing the error of the classifier that produced the matrix.

The artificial example in Table 1 consists of 25 examples of each of four classes A, B, C and D, resulting in a data set of 100 samples. In this demonstration matrix the strong diagonal represents those examples that are classified correctly, such as class D. Reading down column D it can be seen that D type samples are only categorised as being of class D. This does not imply that class D is not confused with other classes, as in the example groups B and C have samples mis-classified as class D. Examination of row D shows that columns B and C have entries in them and are therefore confused with D.

This is an example of the lack of symmetry in the confusion between classes, an issue which is not addressed by the CTE algorithm, which assumes all inter-class confusions are symmetrical such that if class$B$ is confused with class D then class D is equally confused with class B. Classes A and C have the largest inter-class confusion in this example, with A being categorised as C 3 times and C as A 3 times, resulting in a total inter-class error of 6 examples. To a lesser extent B and A, B and C etc. are also mis-classified. If classes A and C are merged into a meta-class labelled AC the confusion between them is removed, allowing the confusion matrix to be modified to that shown in Table 2.

From the results in Table 2 it can be seen that, in a crude sense, the overall classification error has been reduced from 14 errors (an average of 3.5 examples per class), in the original matrix to 6 errors (an average of 2 examples per class) at the expense of loosing the ability to discriminate between classes A and C. During this merging process the classifier from which the confusion matrix was produced is not retrained or altered. Its four outputs A to D are just conceptually re-mapped to the three outputs AC, B and D.

During evaluation of the ensemble the original four outputs of this classifier are used. An interesting line of future research is whether or completely retraining it using the new mappings, can provide improved performance over the conceptual re-mapping currently implemented. A second option, for classifiers that support it such as instance based clssifiers, is to modify the classifier internal data structures so that it only produces the merged results.

This perceived error reduction is a minor benefit obtained by the algorithm. A second, and more significant, benefit comes from now having the ability to separately classify the AC meta class, produced by analysis of the confusion matrix from the first classifier, to obtain a more focused classification of the$A$ and C classes. This allows the complexity of the

Output Vector	0.2	0.3	0.5	→	Output Vector	0.2	0.25	0.3	0.25
Class Labels	*A*	*C*	*Other*		Class Labels	*A*	*B*	*C*	*D*

Table 3: *Example of using replication to produce an output vector with length that matches the number of classes in the problem space. On the left is an example result vector with three entries from Subclassifier AC in Figure 11 that is converted to an output vector with four entries on the right hand side.*

problem to be reduced towards the representational ability of the base classifier, which can in theory allow a more accurate classification. The "Focusing" of the classification problem is achieved using a sub-classifier that is trained using a new input-to-output coding of the problem space. This encoding is derived from the meta-classes found by examination of the parent learners performance. Instead of having output classes$A$, B, C and D, as the original classifier and the data set had, the new classifier is taught using output classes A, C, and "*Other*" where *Other* is a meta class consisting of all classes that this learner is not trying to separate (in this case classes B and D). The "*Other*" meta-class contains those classes that the first learner could confidently discriminate between or that have been allocated to a different sub-classifier. Introducing the "*Other*"meta-class is essential as it allows the sub classifier to handle the entire problem space, and hence the tree structure produced can be used as a classifier ensemble rather that a tree structured classifier. The overall effect of creating sub-classifiers that handle the classes that the primary classifier could not separate well is to implement, in effect, a weak form of error correcting output coding, while allowing the complexity of the problem domain to be reduced towards the representational ability of the constituent classifier.

We now have a structure consisting of two classifiers in a tree structure (Figure 11) each of which has a distinct output coding: the root classifier with four outputs and the sub-classifier with 3 outputs. If this process is continued iteratively until no further groupings can be found for non-leaf nodes in the tree (as is the case with this example) an ensemble learner is created that, at each stage, attempts to minimise inter-class confusion while utilising a variation of error correcting output coding in a tree structure that keeps classes that are similar in close proximity to one another. As each learner randomly creates its own training data-set and confusion matrix creation data-set from the training set passed to the algorithm, many of the properties associated with bagging are also exploited. This is due to random resampling of the overall test set. It must be noted that the evaluation of the algorithm uses an entirely different validation data-set, which is separated from the problem data before training commences.

In practice the analysis of the confusion matrix is undertaken using a clustering algorithm (see section 6.2.1) that finds sets of groupings automatically rather than the ad-hoc manual classification used in this example.

The number of non-leaf nodes in the tree equals the number of learners that have been trained to act as part of the ensembleand is dependent on a variety of factors. The most factor is the number of unique classes in the training data. Secondary factors include the amount of inter-class confusion in the problem space and the threshold parameters passed to the clustering algorithm, which in turn determine when the tree growing (learning) process is terminated.

To classify a sample the ensemble tree is traversed and each constituent learner in it is asked to classify the input vector. Every classifier is capable of doing this, as each has been trained using data that represents all classes in the problem space, albeit different encodings of them. The resulting vectors from all the classifiers are passed to a classifier combination scheme to produce the ensemble's answer.

2.1.5 Combining the Base Classifier Results.

A large variety of approaches exist that solve the classifier combination problem (Alkoot and Kittler 1999). i.e. combining the results from a number of classifiers to produce a more accurate response. However, unlike many ensemble learners, each classifier in the CTE algorithm produces an output vector with a different length (a different number of outputs per classifier). To simplify the combination of these varing length result vectors they are first made into vectors of equal length. The method chosen to handle the varying length output vectors is to replicate the output for each *Other* meta-class to all examples the meta-class contains after normalising by the number of entries in the meta-class. An example of this is the *AC* classification node in Table 2 which has three outputs while the problem domain shown in Table 1 requires four. The values in the output vector corresponding to the probabilities of *A* and *C* can therefore just be copied into the actual result vector, however the value associated with the "*Other*" meta-class is replicated into the two remaining positions corresponding to classes *B*and *D*. Hence this classifier contributes a "vote" to all classes clustered in the "*Other*" meta-class along with those classes it directly classifies. This process is shown using an example result vector in Table 3. After a vector of uniform length is created it is normalised such that it has a unity sum of its entries.

This process is applied to the output from each constituent classifier in the tree structure to produce a set of output vectors of the correct length. A subtle problem occurs at this point as many common classifier combination techniques, such as border count voting, are not normally required to handle duplicate values. They are a very rare occurrence in normal use. From the previous description of how uniform length output vectors are created from the output of each base classifier it can be seen that quite a large number of duplicate values are produced. Therefore due consideration must be given to how duplicate values are handled by the CTE algorithm. For the confidence and average voting approaches this problem is not relevant.

The border count, as described in Black (1987), allocates for a output vector of length k a rank in the range 1 to k for each value in the vector with the largest value k, assigned to the most confident output and the smallest to the least confident. This process is performed on all vectors produced by the ensemble's constituent classifiers. The rankings produced for each class from each classifier are then summed to produce a final ranking which, when normalised, is used as the probabilistic output from the ensemble. The duplication problem described previously occurs when we try to find the rank value of each element in a classifier's output vector when there are multiple elements with the same value. The most common solution is to randomly allocate ranks to those duplicate elements. However, due to the large number of duplicate elements produced by the algorithm, this is not a suitable solution. Therefore a selection of three other approaches to the solution of this problem have been evaluated, along with the confidence voting and the averaged voting classifier combination schemes.

Borda Count In the table below the traditional method of handling vectors with equal values is shown. A demonstration vector with eight entries, of which three have the same

value, is shown. This vector is not normalised to maintain clarity. The most common method of handling this situation is to rank the unique values, taking into consideration the number of duplications, and then to randomly assign the remaining ranks to the duplicate entries (ranks 3, 4 and 5 are randomly assigned to the entries containing question marks in the table). In most classifier combination systems output vectors with duplicate values are rare, so the averaged performance of an ensemble, using this technique, over a number of outputs is not significantly affected by the random allocation of importance to the duplicate values.

The three variations of the Borda count that have been investigated are very similar to the normal approach, with slight variations in the manner in which they handle duplicate entries in the vector. All three approaches result in entries with duplicate values having the same rank and therefore the same contribution to the final result. This is unlike the method above, in which the duplicate entries end up contributing varying amounts that are allocated in a random manner.

Maximum Borda Count The Maximum Borda count, shown below, works by assigning each duplicate entry the maximum rank value from the range of ranks available to the set of duplicates. In the table below the range of ranks available to the duplicates is the same as in the previous example (i.e. 3, 4, and 5) therefore every entry in the constructed vector with a value of 0.3 is assigned the rank of 5.

Unlike the previous approach this assignment allows the duplicate entries an equal contribution to the final result. The ranks assigned to entries with lower values than the duplicates (0.1 and 0.2) are, however, assigned a disproportionately low contribution due to the missing rank values 3 and 4.

Minimum Borda Count The Minimum Borda count is the exact opposite of the previous (Maximum Borda Count) approach in that the duplicate entries are assigned the lowest rank from the range available to them. In this example the lowest rank available, 3, is assigned to the duplicates. This results in the duplicate entries contributing equally to the final result. The drawback with this method is that it emphasizes the contribution of entries with the highest ranks.

Collapsed Borda Count The Collapsed Borda count is a compromise between the two previously mentioned approaches. It assigns an equal rank to duplicate elements without reducing the contribution of any of the elements. This is done by giving the duplicate entries the maximum rank value available to them as in the Maximum Borda count. Unlike the maximum Borda count however, the ranks assigned to elements with a lower value than the duplicate elements are shifted to ensure that the rank values are contiguous. This means that in the example below rank values 1 and 2 no longer exist.

Weighted Base Classifiers In addition to the classifier combination techniques described here the CTE allows an optional bias to be applied to the outputs of each constituent classifier. This bias is derived from the accuracy of each classifier on its Test data-set and is used to scale the values of its outputs prior to their use in the combination process.

Applying these weights is possible when using any of the classifier combination techniques described in this section.

Data Set	Combination Technique				
	Average	Combination	Minimum	Maximum	Collapsed Borda
Balance	72.48	75.84	73.12	68.16	69.44
Glass	69.18	64.51	66.82	64.01	63.54
Iris	92.00	93.33	93.33	93.33	92.67
Letter	88.79	91.79	91.48	87.45	92.05
Segment	96.58	96.54	96.58	96.28	96.84
Vehicle	71.15	68.91	68.44	68.20	66.78
Waveform	73.80	73.98	72.62	72.84	72.80
Average	80.57	80.69	80.34	78.61	79.16

Table 4: *Combination approach results.*

2.1.6 Examination of the Combination approaches

To provide an idea of the relative performances of the combination methods described in the previous section a simple experiment was performed. This involved the Clustered Tree Ensemble algorithm using the Maximum Spanning Tree clustering module without classifier weighting (CTE-MST, see section 2.2 for descriptions of the abbreviations used) and the C4.5 decision tree as the base classifier. The data sets used in the experiment excluding only the Gaussian and Tea problems.

The CTE-MST ensemble was run nine times using stratified cross validation on each of these data sets for each combination technique. A summary of the results is shown in table 4.

The averages shown in table 4 illustrate that the performance of the CTE-MST algorithm is not highly dependent on the type of classifier combination used as most of the combination approaches produce very similar results. Overall, the technique of combining base classifier results based on confidence produces, by a slim margin, the best overall accuracy for the Ensemble configuration under consideration. Using combination by confidence, however, requires that the Ensemble be provided with the confidence threshold as an extra user-configurable parameter. The effect of this parameter is related in some manner to the number of classes in the problem domain and is not easily determined. If the threshold is set to an inappropriately small or large value for a particular data set it can result in the ensemble producing no effective classification for some input feature vectors. For these reasons confidence-based combination will not be used when comparing the CTE to other ensemble

algorithms. Average combination performs almost as well as confidence based combination and is used in all ensemble comparison experiments.

2.1.7 Pruning the Tree

Once the Hierarchy is built, or during its construction, pruning techniques can be employed to reduce the size and complexity of the final tree. One approach to pruning (Huang *et al.* 1998b) is to test the classification accuracy of the entire hierarchy after each new sub-classifier is added. If the new sub-classifier reduces the accuracy of the ensemble then the newly added sub-classifier is removed. This approach has similar characteristics to the early stopping algorithm used to terminate the Back Propagation algorithm on Feed- Forward Neural Networks, in that it does not guarantee to achieve an optimal classifier due to the local nature of the pruning.

A second but more computationally expensive approach is to build a complete tree and to then remove or merge branches that are deemed unnecessary to the performance of the Clustered Tree Ensemble. This allows global optimisations to be utilised rather than the local ones used in the previous method.

Neither of these methods is implemented, due the to fairly small nature of the Trees created using the data-sets under study. As the trees are small there is little advantage to optimising the structure. Instead, a more basic parameter based approach is used that, to some extent, allows the size of the tree to be controlled. This parameter is the Threshold argument passed to the clustering modules described in section 2.1.1.

2.2 Abbreviations

For the rest of this document the Clustered Tree Ensemble will be known, in general terms, as the CTE. In many circumstances a number of abbreviations may be appended to this title to distinguish between specific CTE implementations. The term CTE will be followed by a three letter abbreviation of the clustering method used in its implementation such as:

- MST:- The Maximum Spanning Tree (See section 6.2.1) clustering algorithm is used.
- MCC:- The Minimum Capacity Cut (See section 6.2.1) clustering algorithm is used.

In addition to these three letter clustering descriptors an additional extension, W, to the title may be used to indicate that the CTE utilises classifier weighting.

These rules result in four acronyms for the current set of options that can be applied to the CTE.

- CTE-MST :- Clustered Tree Ensemble using Maximum spanning tree clusterer.
- CTE-MST-W :- Clustered Tree Ensemble using Maximum spanning tree clusterer and applying weights to the classification results obtained from each base classifier.
- CTE-MCC :- Clustered Tree Ensemble using Minimum Capacity Cut clusterer.
- CTE-MCC-W :- Clustered Tree Ensemble using Minimum Capacity Cut clusterer and applying weights to the classification results obtained from each base classifier.

2.3 Results on Illustrative Data Set

One method of assessing whether an ensemble learner produces any performance increase is to compare its accuracy with that of a classifier configured in the same manner as the

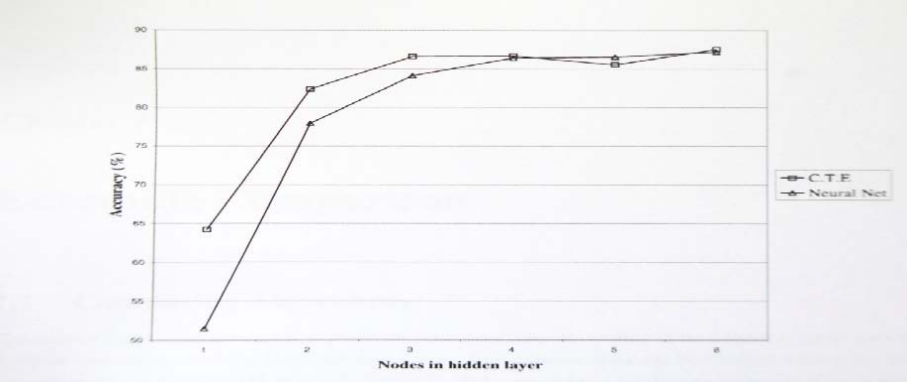


Fig. 12: *Classification accuracy of the CTE-MST and Neural Network learners versus the number of nodes in the hidden layer. The classification accuracy reported is the average of five trials against the verification data set using cross validation.*

ensemble's constituent classifiers. A performance increase should be noticeable, whether the base classifiers are strong or weak learners, as long as they perform better than random selection (outperform the Zero-R classifier). The ensemble should show the most significant improvement when using weak learners, as strong learners may be able to find satisfactory solutions to the classification problem on their own. In the following experiment the CTE algorithm, with a neural network as the base classifier, is compared with the performance of a neural network alone. The neural network classifier was chosen for this comparison as its performance can be altered by the simple method of changing the number of nodes in its hidden layer. This enables a series of experiments to be performed by changing the "strength" of the base classifier.

To perform the following experiments the data set was split into five equal sized blocks that were combined in various permutations to produce five different versions of three data sets: the training, test and verification sets. The training set contains three of these blocks, while the test and verification sets consist of one block each. The five disjoint permutations are used for cross validation.

Due to the simple nature of the example problem the layout of the Neural network, when used on its own and as part of the CTE algorithm, was adjusted to have a only single hidden layer rather than the two often used for more complex problems.

Figure 12 shows how the performance of the Neural Network and CTE-MST change with the number of nodes in the single hidden layer. Both algorithms tend towards the maximum accuracy (87.87%), as the number of network nodes increases. The CTE-MST algorithm reaches this expected "upper bound" on classification accuracy with only three nodes in the single hidden layer used by its base classifier, again as expected for a network that is characterising the problem well. The single neural network however requires four nodes to achieve this same accuracy. This suggests that the methods the CTE uses to create diversity amongst its constituent networks are succeeding for this problem and that the CTE's approach to combining the results of those base classifiers can produce an improved classification performance.

2.4 Conclusions

This Chapter has introduced, described and investigated the Clustered Tree Ensemble, a novel approach to the creation of ensemble learners based around the principle of class similarity. The classifier created forms a tree structure, with classes that are similar to each other remaining in close proximity within the tree. This clustering or grouping of similar classes has potential uses that allow human querying of large digital image libraries.

The simple experiment in section 2.3 shows that the methods used to create diversity among the constituent learners and classifier combination techniques chosen have the potential to improve classification accuracy over a solitary learner. Chapter 7 will examine and discuss the performance of the CTE and its variants with relation to a number of commonly used ensemble techniques.

3 Ensemble Comparison

3.1 Comparing Algorithms

The aim of these comparisons is to place the Clustered Tree Ensemble in its correct context with respect to other ensemble machine learning algorithms. The comparisons are performed within the Waikato Environment for Knowledge Analysis (Weka) [1]. Weka provides a consistent framework for testing and comparing learning algorithms, although its most valuable asset is perhaps its library of implemented learning algorithms.

Performing an unbiased comparison of machine learning algorithms requires the careful planning and consideration of a large number of issues in both the experimental setup and the analysis of the results produced. The methodologies used for comparison and evaluation of learning algorithms have until recent years received little attention, with the notable exceptions of Langley (1988) and Jensen and Cohen (2000). However over the last few years approaches to experimental evaluation have become the focus of a number of articles in computer science journals. Cohen (2017) described general methods for the empirical analysis of Machine learning algorithms and provides a good overview of this field. Salzberg published a critique (Salzberg 2000) of data mining publications which also provides a tutorial on the ideas of statistical validity. Prechelt (1996) discussed some of the problems of statistically comparing algorithms, specifically in the field of neural networks research, and concluded that many of the current articles published in neural network related journals were not acceptable in that they failed a simple test that he described as follows:

> "An algorithm evaluation is called acceptable if it uses a minimum of two real or realistic problems and compares the results to those of at least one alternative algorithm."

Tichy *et al.* (1995) found that in general articles from computer science journals also had similar failings. Their results show that up to 40% of articles drawn from a selection of computer science journals that probably should have an empirical evaluation have none at all.

In an attempt to avoid falling victim to the criticisms levelled by these authors (Cohen 2017; Tichy *et al.* 1995; Prechelt 1996; Jensen and Cohen 2000; Salzberg 2000) a number of the issues they have raised are discussed here, starting with some of the problems surrounding the choice of data-sets.

[1] University of Waikato, New Zealand. http:\\www.cs.waikato.ac.nz\ml\weka\

3.1.1 Data Sets

When comparing learners the data-sets chosen can fail in three main ways:

- The problem domain is too simple: Holte [93] found, on experimentation using a large number of commonly used and popular data sets taken from the UC Irvine (UCI) Machine Learning Repository [12], that using an extremely naive classifier yields results comparable to those produced by much more sophisticated learners such as CART. Cohen (2017) calls this the *ceiling effec*t and suggests that it is a product of the data-set having an underlying function that is too simple to approximate and that the utility of comparisons on such data-sets is reduced as they produce little useful information. It must be noted that many data-sets from the UCI repository are artificial. A data set from UCI that is derived from real world data and suffers from this trend to a certain extent is the Iris data-set which, in Table 5, exhibits results that are strongly bimodal across various base classifiers. They either classify it accurately (88% to 95% accuracy) or not at all (33% accuracy) with the exception of the Decision Stump Classifier (66% accuracy). It has been included in this comparison as it is the oldest and possibly most widely used learning problem available. This ceiling effect can also be attributed to the over-capacity of the learning algorithms.
- The problem domain is too complex: Cohen (2017) describes this as the *floor effect*. The input-output function underlying the data is difficult to approximate, therefore all learners produce very poor results. In this situation it is difficult to assess the benefits of individual algorithms due to the very high base-line errors. An example of such a data-set is the DARPA Switchboard speech recognition evaluations in which word error rates of around 40% are considered normal. Of the data sets used in this comparison the "Tea" set seems to fall victim to this complaint, although the difficulty that many classifiers have in finding a mapping for it may be related to the number of features it contains and the fact that a number of individual classes in the data-set are, to all intents and purposes, identical to each other. The floor effect is also affected by the representational capacity of the learning algorithms.
- The number of training samples is too small: Raudys and Jain (1991) describe some of the problems caused by having a small number of samples in the problem space and how these problems are linked to the number of features, the complexity of the problem, and the type of classifier in use. The suggestions given towards alleviating these problems, however, are biased towards those who are creating data sets rather than using pre-existing sets, as is the case in this comparison. Of the two data sets that were created for this thesis the "Tea" set is most likely to suffer from this problem due to the very large number of parameters in its feature vectors when compared to the number of samples per class. However,as described previously, it may be that the problem is just too complex .

Data Repositories Communal data set repositories such as the UCI Machine Learning repository (Blake and Merz 2017) create a peculiar, although perhaps minor, problem when using their data-sets to compare learning algorithms. The easy availability of the data-sets from such repositories and the ability to compare algorithms that this availability provides can encourage the development of learning algorithms which are refined during development using such data-sets. In these circumstances one of the criteria for the algorithm under development can become that it should outperform or match the performance of other known algorithms on a number of data sets from such a repository. At a minimum researchers tend

	Zero-R		VFI		Naïve Bayes		C4.5		Decision Stump	
DataSet	Acc	Var	Acc	Var	Acc	Var	Acc	Var	Acc	Var
Balance scale	46.08	0.11	60.75	19.31	89.76	1.24	79.36	1.86	61.76	0.98
Iris	32.00	0.00	95.11	2.47	95.56	2.19	92.67	3.61	66.00	0.00
Letter	4.07	0.00	61.29	0.63	64.12	0.58	86.31	0.39	7.00	0.08
Vehicle	25.61	0.16	52.76	1.13	45.55	3.05	73.05	1.70	40.46	0.57
Waveform	33.84	0.01	57.24	1.31	79.86	0.70	74.69	1.44	56.55	0.66
Gaussian	11.10	0.00	72.27	1.71	88.16	0.66	86.92	0.60	22.19	0.01
Segment	14.29	0.00	77.37	0.79	80.00	0.77	96.29	0.95	28.54	0.06
Tea	2.75	0.00	32.42	1.01	41.07	1.64	35.31	1.43	5.58	0.09
Glass	35. 52	0.74	55.30	4.07	45.93	6.78	69.17	6.27	44.39	2.31

	Hyper-pipes		SMO		Neural Net		IB1		Decision Table	
DataSet	Acc	Var	Acc	Var	Acc	Var	Acc	Var	Acc	Var
Balance scale	46.08	0.11	87.36	1.37	50.16	12.45	82.72	1.64	78.77	3.31
Iris	91.56	3.57	83.56	3.84	32.67	1.00	94.67	2.83	92.67	3.74
Letter	23.04	0.78	58.81	0.84	4.29	0.45	95.2	0.16	69.04	0.84
Vehicle	33.69	1.06	68.12	1.94	25.73	0.19	68.79	1.85	64.14	2.99
Waveform	47.19	0.92	86.34	0.81	86.17	1.15	72.81	1.16	73.25	1.45
Gaussian	58.37	1.32	56.99	7.06	66.17	5.27	82.96	0.66	86.89	0.61
Segment	75.71	2.23	88.76	1.12	14.29	0.00	96.39	0.43	90.98	1.00
Tea	31.02	1.58	40.41	1.05	2.79	0.05	31.47	1.31	31.41	2.18
Glass	53.26	5.89	54.65	6.88	35.52	0.74	68.70	4.80	63.55	3.98

Table 5: *Accuracy (%) and variance of base classifiers on all data sets.*

to repeatedly run experiments using a small number of such data sets to fine-tune their algorithm. Each of these experiments should be considered a separate experiment. For example if ten different combinations of parameters are tested then significance levels (p-values) of 0.005 would be needed to be comparable with a p-value of 0.05 for a single experiment (Salzberg 2000). This can become a problem when analysing the performance of a well known classifier, as it is highly likely that the parameters it uses will, to some extent, have been optimised over the problem domains in the repository and the number of fine tuning experiments is unlikely to be known.

To alleviate these problems a number of precautions have been taken:

- The CTE algorithm has been developed and tested exclusively using four artificially generated inhouse data-sets (containing 4,5,6 and 7 classes respectively) of various complexity. The exception to this rule has been in the selection of base classifier method (section 6.2.6) which used data sets drawn from the UCI repository as the original authors have extensively tested the base classifiers against data sets from this repository. Minor adjustments of the parameters of some of the base classifiers was undertaken. After these adjustments the "Tea" and "Gaussian" data sets were added for comparison but at this stage but no parameter tuning was undertaken.
- While comparisons on known data sets from the UCI repository are useful in placing a new algorithm in context the results are not necessarily valid. Any comparison therefore needs to be made on previously unknown data sets to remove experimental bias. The "Tea" and "Gaussian" data sets perform this function for the comparison of the ensemble approaches.

3.2 Algorithm Comparison

To assess the effectiveness of the CTE algorithm a comparison with a selection of well known Ensemble learning techniques was undertaken. The algorithms chosen for this comparison, Bagging (Breiman 1996), Boosting (Schapire and Freund 2014) and Stacking (Wolpert 1992), are representative of the variety of approaches to classifier combination and ensemble learning.

3.2.1 Experimental Setup

A set of six experiments were performed to assess the relative performance of each of the ensemble learning systems when used with varying base classifiers. The base classifiers selected for this purpose were Zero-R (Hansen and Salamon 1990), Hyper-pipes (Frank *et al.* 2010), Naive Bayes (Domingos and Pazzani 1997), VFI (Demiröz and Güvenir 1997), C4.5 and Decision Stump (Quinlan 1993). Each experiment involved the input data being randomly sorted three times to produce three versions of the training and verification data sets. For each of these three randomised data sets a three-fold stratified cross validation was used to train and evaluate each of the ensemble learners. This resulted in each ensemble learner being evaluated 9 times for each base classifier type on each original data set. A total of 3402 ensemble learners were trained and evaluated considering the 9 evaluations of the 7 ensemble learner types using 6 base classifier types on the 9 data sets. Due to the large number of runs per base classifier type (567) only classifiers that had low training and evaluation times were selected as the base classifiers for the ensembles. This eliminated the use of learners such as Artificial Neural Networks, which have a large training time, and *k*-Nearest Neighbours, which have a large evaluation time. The result of this decision is that the base classifiers are often fairly poor or weak classifiers (Schapire *et al.* 1997).

The parameters applied to the Ensemble learners and the base classifiers are shown in Appendixes B and A respectively. The parameters for the base classifiers that were used in the ensemble comparison were taken directly from the earlier comparison of the base classifiers. The parameters for the ensemble classifiers were adjusted using the four artificial data sets mentioned earlier and then applied to the ensemble comparison.

3.2.2 Comparison Results

The averaged results for these experiments grouped by base classifier are shown in Table 6 and 7. A comprehensive comparison of these results containing the average, variance and t-test analysis are available in (Fudge 2010).

Unlike the other ensemble learners investigated there is an upper limit on the number of base classifiers that can be created by the CTE algorithms. This limit is related to the number of classes in the problem domain. It is therefore expected that the CTE learner will exhibit a poor performance on problems that contain a low number of classes, with increasing performance on problems with large numbers of classes. To maximise the number of base classifiers created by the CTE algorithms, the threshold parameter (tolerance) for inter-class confusion is set to zero for the all variants of both the CTE-MST and CTE-MCC algorithms. Using this setting for the threshold parameter will, in theory, maximise the performance of the CTE algorithms. By comparison the Bagging and Boosting ensembles have the number of constituent learners they can utilise set to a maximum of five.

In the case of Boosting this is a reasonable figure, as the algorithm used usually terminates before constructing even this number of classifiers, especially when using data sets

such as the Gaussian, Iris, Glass, Balance and Letter. On the Tea data set a slightly improved performance can be obtained by allowing a larger number of base classifiers to be used. The Bagging ensemble uses exactly this number of constituent classifiers and in all circumstances its classification performance can, at least in theory, benefit from using an increased number of learners. Figure 13 shows how the performance of Bagging can improve with an arbitrarily large number of base classifiers.

Zero-R As expected using Zero-R as the base classifier produces poor results as it produces an accuracy equal to randomly selecting an answer. None of the ensembles based on this classifier is able to produce an significantly improved performance over that of the base classifier its self. All of the ensembles produce an accuracy of 46.08% on the balance scale data set, an insignificant improvement over the 45.76% produced by the base classifier. In many situations the ensembles have a worse performance than the Zero-R classifier, as is the case with the Iris data set for which the base classifier is 33.33% accurate. The CTE algorithms match this performance. However Bagging, Boosting and Stacking all have a slightly lower accuracy (32.67%, 32.00%, 32.00% respectively with a confidence value of 0.031). In another example using the Letter data set, CTE and Bagging perform worse than the Zero-R (3.67%, 4.06% respectively) while the Boosting and Stacking ensembles equal the Zero-R's performance. The overall poor performance of the ensemble learners was expected, due to the Zero-R base classifier having a low variance (Table 5) and rarely performing better than random chance, both of which are detrimental to the performance of ensemble learners (Brodley and Lane 1996; Hansen and Salamon 1990; Krogh and Vedelsby 1994).

Table 8 (left) contains ranked accuracies for the ensemble learners using the Zero-R base classifier averaged over all experiments on all data sets. It shows that the CTE-MST, Bagging, AdaBoost, and Stacking ensembles have accuracies that are approximately equal to the accuracy of the base classifier 22.81% (Table 5). Interestingly the MST-W, MCC and MCC-W versions of the CTE algorithm produce significantly worse average performances then a single Zero-R classifier. These poor results can be traced back to high classification errors associated with the Balance and Glass data sets.

Hyper-pipes The Boosting and Stacking ensembles perform poorly in comparison with the CTE and Bagging algorithms when using Hyper-pipes as the base classifier. This is shown in Table 9, where Bagging and the CTE variants have significantly higher performances than AdaBoost and Stacking. Boosting and Stacking have a performance that is comparable to or worse than the Hyper-pipes base classifier, while the performance of the CTE variants and Bagging ensembles is slightly better. None of the ensembles is able to produce large improvements on the performance of the base classifier. This is thought to be caused by the relatively low variance of the base classifier on the data sets (Table 5). Boosting on the Iris data set using Hyper-pipes as a base classifier could not be performed and therefore no result has been entered in Table 6. All ensembles had an equal performance on the Balance dataset, however, for all other data-sets, the Stacking ensemble produced significantly worse results then the other ensembles under consideration.

Naive Bayes When using the Naive Bayes learner as a base classifier the CTE ensembles are outperformed by almost all other ensembles, with the exception of Stacking on the Tea and Glass data sets. In the case of these two data-sets the fact that the CTE variants outperform the Stacking algorithm is not of significance, as the performance of the CTE variants and Stacking are both well below the accuracy obtained using the Naive Bayes classifier on its own. For example the best CTE variant and Stacking score 43.43% and 34.12% (average)

Zero-R base learner	Ensembles						
Data Set	CTE-MST	CTE-MST-W	CTE-MCC	CTE-MCC-W	Bagging	Boosting	Stacking
balance (3)	46.08	46.08	7.84	7.84	46.08	46.08	46.08
iris (3)	33.33	33.33	33.33	33.33	32.67	32.00	32.00
letter (26)	3.67	3.67	3.81	3.67	4.02	4.07	4.07
vehicle (4)	25.65	25.65	25.65	25.65	25.73	25.61	25.61
waveform (3)	33.84	33.84	33.84	33.84	33.42	33.84	33.84
gaussian (9)	11.11	11.11	11.11	11.11	11.11	11.10	11.10
Segment (7)	14.29	14.29	14.29	14.29	14.29	14.29	14.29
tea (40)	2.39	2.39	2.82	2.82	2.71	2.75	2.75
glass (6)	35.52	4.21	7.94	7.94	35.52	35.52	35.52

VFI base learner	Ensembles						
Data Set	CTE-MST	CTE-MST-W	CTE-MCC	CTE-MCC-W	Bagging	Boosting	Stacking
balance (3)	51.73	53.17	44.64	55.85	56.27	63.26	55.85
iris (3)	94.67	94.22	94.89	94.89	95.11	93.11	90.89
letter (26)	41.44	50.00	53.98	53.79	61.99	61.29	63.53
vehicle (4)	52.17	54.06	53.15	52.32	53.11	52.80	49.45
waveform (3)	60.29	59.49	59.47	59.87	59.72	57.24	33.59
gaussian (9)	73.12	73.87	78.61	78.56	73.89	72.27	70.78
Segment (7)	83.26	80.85	78.53	78.70	77.75	77.37	83.52
tea (40)	28.29	22.05	22.07	20.58	29.40	31.30	27.41
glass (6)	54.19	53.76	55.14	56.39	58.10	56.23	51.86

Hyper-pipes b.l.	Ensembles						
Data Set	CTE-MST	CTE-MST-W	CTE-MCC	CTE-MCC-W	Bagging	Boosting	Stacking
balance (3)	46.08	46.08	46.08	46.08	46.08	46.08	46.08
iris (3)	91.78	92.44	92.22	91,56	92.56	N/A	86.89
letter (26)	30.28	29.35	30.42	30.27	28.89	32.88	19.46
vehicle (4)	37.00	36.09	37.39	37.43	39.64	34.55	30.26
waveform (3)	51.80	52.59	52,26	51,07	53.69	48.31	39.96
gaussian (9)	63.81	63.90	63.80	63.82	65.10	59.75	53.15
Segment (7)	80.51	79.05	79.55	80.29	80.20	75.92	61.11
tea (40)	32.95	32.91	33.56	22.57	33.71	32.92	20.33
glass (6)	51.06	52.00	51.55	50.60	54.67	54.33	42.66

Table 6: *Average Ensemble Learner accuracy (%) on each data set grouped by base classifier. The number of classes in each data set is appended to the data set name. The column averages for the Zero-R, Hyper-pipes and VFI base classifiers are shown in Table 8, 9 and 11 respectively.*

C4.5 base learner	Ensembles						
Data Set	CTE-MST	CTE-MST-W	CTE-MCC	CTE-MCC-W	Bagging	Boosting	Stacking
balance (3)	70.40	74.45	74.56	74.88	83.20	80.53	80.34
iris (3)	93.11	94.00	93.11	92,44	93.78	93.33	92.44
letter (26)	91.20	87.91	89.59	89.31	89.66	92.17	85.78
vehicle (4)	69.74	69.66	72.22	70.72	71.71	73.80	71.08
waveform (3)	72.71	73.94	72.89	73.61	79.60	79.67	74.62
gaussian (9)	86.24	85.80	86.22	86.36	87.36	85.56	85.96
Segment (7)	95.86	95.30	96.09	95.96	96.13	97.20	95.97
tea (40)	42.38	37.51	41.33	39.45	40.75	41.54	32.52
glass (6)	64.79	62.60	64.17	66.67	70.41	70.25	63.41
Naive Bayes base l.	Ensembles						
Data Set	CTE-MST	CTE-MST-W	CTE-MCC	CTE-MCC-W	Bagging	Boosting	Stacking
balance (3)	69.60	72.59	70.24	70.24	89.71	89.50	90.99
iris (3)	94.89	94.67	94.67	94.89	94.67	95.11	95.33
letter (26)	50.47	57.87	59.28	59.34	64.35	64.12	56.55
vehicle (4)	44.68	44.92	44.05	44.76	45.23	45.55	45.07
waveform (3)	80.43	79.87	79.81	79.79	79.93	79.86	82.28
gaussian (9)	87.49	88.19	87.79	87.76	88.19	88.16	87.07
Segment (7)	73.49	80.74	79.58	78.60	80.19	80.00	82.66
tea (40)	30.12	31.26	32.38	32.56	41.15	41.07	24.67
glass (6)	43.43	44.98	45.33	48.41	51.23	46.55	34.12
Decision Stump b.l.	Ensembles						
Data Set	CTE-MST	CTE-MST-W	CTE-MCC	CTE-MCC-W	Bagging	Boosting	Stacking
balance (3)	62.82	54.94	57.23	55.63	68.00	74.08	61.76
iris (3)	69.56	69.56	65.78	69.56	75.11	94.00	66.00
letter (26)	13.41	12.78	10.88	10.34	8.10	7.00	7.00
vehicle (4)	40.86	40.50	40.82	40.31	41.88	40.46	40.46
waveform (3)	57.25	57.91	57.15	57.23	57.71	61.96	56.55
gaussian (9)	36.94	43.57	24.63	23.45	22.20	22.19	22,30
Segment (7)	51.13	58.18	48.57	48.59	47.37	28.54	23.46
tea (40)	7.98	9.11	7.52	6.88	5.50	5.58	5.59
glass (6)	16.68	17.45	25.39	23.67	47.50	44.39	44.39

Table 7: *Average Ensemble Learner accuracy (%) on each data set grouped by base classifier. The number of classes in each data set is appended to the data set name. The column averages for the C4.5, Nave Bayes and Decision Stump base classifiers are shown in Tables 12, 10 and 13 respectively.*

Ensemble	Accuracy (%)
CTE-MST	22.88
Bagging	22.84
AdaBoost	22.80
Stacking	22.80
CTE-MST-W	19.40
CTE-MCC	16.63
CTE-MCC-W	15.61

Ensemble	Accuracy (%)
Bagging	54.95
CTE-MCC	54.09
CTE-MST	53.92
CTE-MCC-W	53.85
CTE-MST-W	53.83
AdaBoost	46.97
Stacking	44.32

Table 8: *Ranked average accuracies. Left: Zero-R; Right: Hyper-pipes.*

Ensemble	Accuracy (%)
Bagging	70.52
AdaBoost	69.99
Stacking	66.53
CTE-MCC-W	66.26
CTE-MST-W	66.12
CTE-MCC	65.90
CTE-MST	63.84

Ensemble	Accuracy (%)
Bagging	62.82
AdaBoost	62.76
Stacking	61.22
CTE-MCC-W	60.16
CTE-MST-W	60.05
CTE-MCC	59.91
CTE-MST	58.54

Table 9: *Ranked average accuracies. Left: Naive Bayes; Right: VFI.*

accuracy respectively on the Glass data-set while the Naive Bayes classifier produces an accuracy of 45.93%. None of the Ensembles produces a consistent performance improvement over the Naive Bayes base learner, with the exception of the Bagging algorithm run on the Glass data set, where a performance improvement of 5.3% is obtained (51.23% for Bagging versus 45.93% for standard Naive Bayes). Table 10 shows the average accuracy of the ensembles on all data-sets. Bagging and AdaBoost perform significantly better than Stacking, CTE-MCC-W, CTE-MST-W and CTE-MCC which in turn show a noticeable, though statistically insignificant, improvement over the CTE-MST ensemble.

Voted Feature Intervals When using Voting Feature Intervals as the base classifier the averaged results (Table 6) across datasets show a fairly balanced performance for all algorithms. Stacking stands out as having a significantly lower average accuracy on most problems except Image segmentation and Tea Recognition. Its worst performance is obtained on the Waveform problem, where the average accuracy it obtains is approximately 18.5% lower than its nearest rival. On the Weight balance data-set no significant performance difference is noticeable (Appendix C1) although this is mainly due to the very large variances (10%-20%) associated with the results of all ensemble techniques. This suggests that the VFI base classifier is not suitable for this problem. Possibly this is due to the complete overlap of the problem's features versus its class designations. The Gaussian problem shows the CTE-MCC and CTE-MCC-W algorithms outperforming all other ensembles by approximately 5%, with confidences in the range 0.0-0.003. However the more complex Tea and Letter recognition data-sets show all CTE variants being outperformed by both Bagging and Boosting.

C4.5 The C4.5 algorithm (Table 10, left), like VFI, produces very similar results for all ensemble techniques on all problem domains. However, unlike VFI there are statistically

Ensemble	Accuracy (%)
AdaBoost	79.34
Bagging	79.18
CTE-MCC	76.69
CTE-MCC-W	76.60
CTE-MST	76.27
Stacking	75.80
CTE-MST-W	75.69

Ensemble	Accuracy (%)
AdaBoost	42.02
Bagging	41.48
CTE-MST-W	40.45
CTE-MST	39.62
CTE-MCC	37.55
CTE-MCC-W	37.29
Stacking	36.93

Table 10: *Ranked average accuracies. Left: C4.5; Right: Decision Stump.*

significant differences in performance on many of the data sets but, due to the low variances the C4.5 ensembles seem to produce, these differences tend to result in very small performance gains. A number of notable exceptions to this observation exist. Using the Balance data-set the CTE variants have a lower performance, of between 5.55% and 12.8%, then Stacking, Bagging and Boosting. The Waveform data-set shows Bagging and Boosting producing performance improvements of between 4.98% and 6.97% over all other ensembles.

In general, ensembles using the C4.5 constituent classifier produce the best overall performance for all problems when compared to ensembles using the other base classifiers under consideration. The only classifier that can compete with C4.5 and occasionally outperform it is the Naive Bayes learner on the Gaussian, Weight-Balance, Iris, Tea and Waveform data sets.

Decision Stump With the exceptions of the Vehicle and Waveform data-sets, on which it performs slightly better than Hyperpipes, the Ensembles that use Decision Stump as a base classifier produce the worst overall clas- sification accuracies on all problem domains (Table 10, right). This excludes the Zero-R learner as it is not actually capable of classification. Within the set of Ensembles based on the Decision Stump classifier, the patterns follow those predicted at the beginning of this Section. Problems containing a large num- ber of classes such as the Tea and Letter recognition data-sets allow the CTE algorithms to outperform Bagging, Boosting and Stacking. Problems with a small to medium number of classes including the Iris, Waveform, Balance and Glass data sets cause Bagging, Boosting and, to a smaller degree, Stacking to produce lower classification errors then the CTE learner variants.

3.2.3 Weighted versus Unweighted CTE

It does not appear that applying confidence weights to individual base learners produces any reliable improvement in performance. These confidence weights are derived from the accuracy obtained by the constituent classifier on its training data set and are used to scale its outputs prior to their injection into the chosen combination approach. All of the experiments carried out in this Chapter use the Averaging Combiner. It is possible, although unlikely, that the use of a different combiner would produce significantly different results.

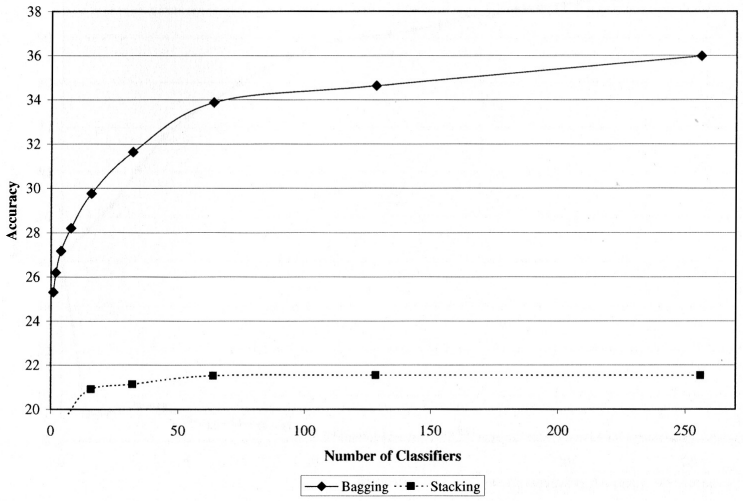


Fig. 13: *Performance change of Bagging versus the number of constituent classifiers used when trained on the Letter Recognition data set.*

3.3 Algorithm Characterisation

This section examines some of the characteristics of the CTE algorithm when compared to the Bagging Boosting and Stacking Ensembles. The aim of this characterisation is to assist in selection of the parameters for the CTE algorithms.

3.3.1 Number of base classifier vs performance

Figure 10 shows how the classification accuracy of the Bagging and Stacking ensembles increase with the number of base classifiers used. The graph is created from the averaged results of five experimental runs of the Bagging and Stacking algorithm, using the Hyperpipes learner as the constituent classifiers on the Letter recognition database. In theory the performance of Bagging will continue to increase towards the optimum classification accuracy as the set of constituent classifiers is enlarged. However, beyond a certain point the cost in terms of computational time and memory requirements becomes too high to justify further enlarging the set of constituent learners. The classification accuracy obtainable with the Stacking ensemble also increases as the effective number of base classifiers is enlarged (in this case by increasing the number of cross validation folds performed). However, its performance plateaus at around 64 folds. The AdaBoost algorithm, like the CTE algorithm, does not allow such an analysis to be directly performed, due to the dynamic nature in which they enlarge their set of base classifiers. AdaBoost allows a limit to be set on the maximum number of constituent classifiers to be used, although in general it will not create this many. In all of the experiments performed the maximum number of constituent classifiers that the AdaBoost.M1 algorithm created was 11, on the Tea data set (40 classes), with a mean below 3 for all data sets. The number of base-classifiers that the CTE algorithms can create is a function of the threshold parameter, the number of classes and the input vectors. On this

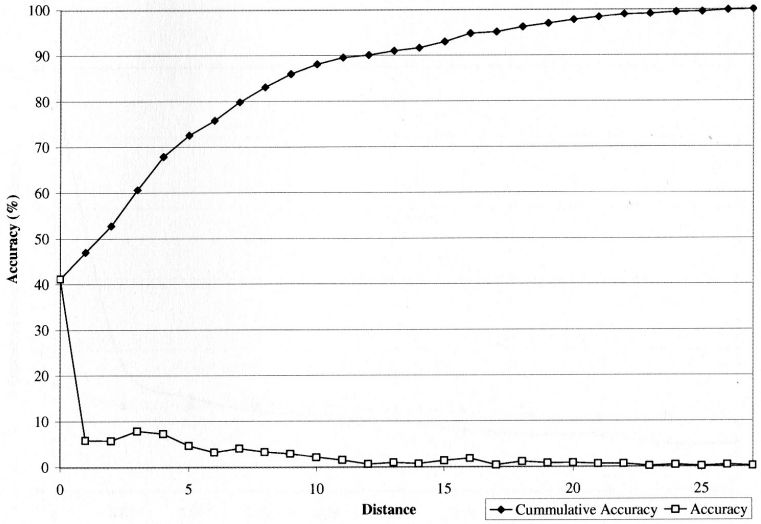


Fig. 14: *Accuracy versus distance for the CTE-MCC learner. The base classifier is Hyperpipes and the threshold parameter is set to zero. Tea Data set.*

problem the CTE-MCC algorithm produced a maximum of 29 (mean 27) base classifiers when the threshold (tolerance for inter-class confusion) was set to zero.

Figure 14 shows the distance in the tree from the correct classification to the answer selected by the CTE instantiation. The data used to create the graph are the results derived from 3 3-fold cross validated runs. The algorithm used is the CTE-MCC with Hyper-pipes constituent classifiers. Distance is measured as the number of constituent classifiers separating the leaves that represent the selected and correct class labels in the tree. For example, in Figure 15 Tableware and Headlamps have a separation distance of 1, as they are both explicitly classified by the same constituent classifier. Tableware and Containers however have a separation distance of 2, as two classification nodes (constituent classifiers) have to be traversed to linkthe two. The graph shows that on average 27 constituent learners were created by the CTE-MCC algorithm when analysing the Tea data set. It is of interest to note that the accuracy plot in general drops as the distance increases (discounting the first point) which suggests that when the CTE produces incorrect classifications they are likely to be close (in terms of the tree structure) to the correct answer. Appendix D shows a number of example tree structures produced using the CTE algorithm and describes some of their properties.

Figure 16 shows the number of base classifiers created by the CTE-MCC algorithm on the Tea and Gaussian data sets. Each data point is the average of 3 tests, each using 3-fold cross validation, on both data sets. The threshold values are incremented by 0.02 between data points. The graph shows that as the threshold value is increased the number of constituent classifiers reduces, but that the required value of threshold for a particular reduction is dependent in part on the number of classes in the problem domain. Figure 16 shows that the number of classifiers in the CTE's generated from the Tea data set drop rapidly within the threshold range of 0.00 to 0.02. After this point the number of classifiers generated drops away slowly with increasing threshold value. The number of classifiers in

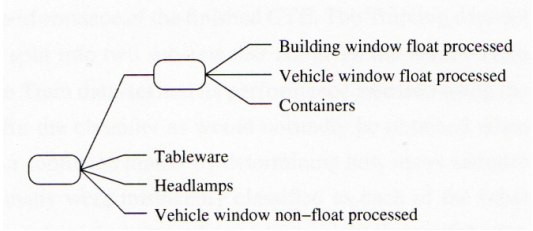


Fig. 15: *Demonstration CTE tree structure for the Glass data-set.*

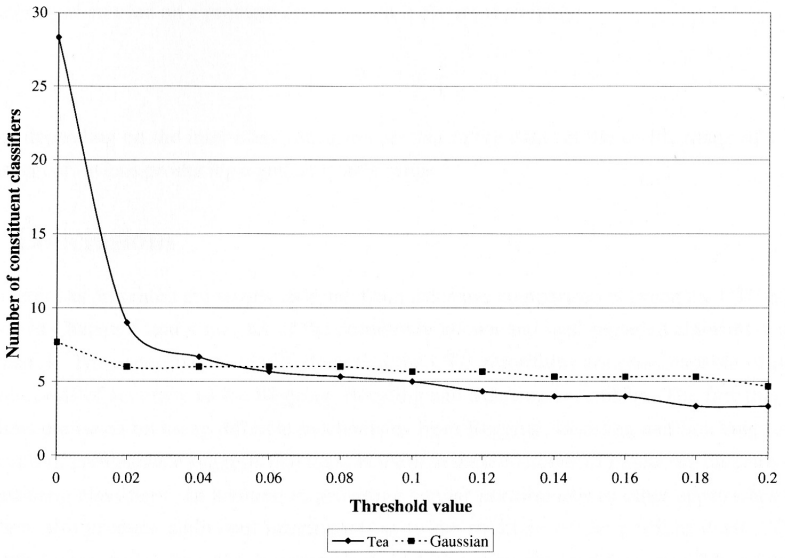


Fig. 16: *Number of constituent classifiers versus Threshold value. Results are shown for C4.5 constituent classifiers in the CTE-MCC algorithm.*

CTE's generated from the Gaussian data set reduces very slowly over the same range by comparison to the results from the Tea problem.

Figure 14 shows an expanded view ofthe results from the Tea data set over the range 0.000 to 0.002. Results such as these show that a good guideline for the practicable range of the threshold parameter, t, when used on a problem domain with k classes is roughly:

$0 \leq t \leq \frac{2}{k}$

however depending on the inter-class confusion present in the data set the usable range of t can vary, with larger confusions producing a greater usable range.

3.3.2 Results

This section has presented the results obtained from extensive comparison between the CTE algorithms proposed in Chapter 6 and a number of the commonly known and well regarded algorithms presented in Chapter 3. The experimental results show that the CTE algorithms

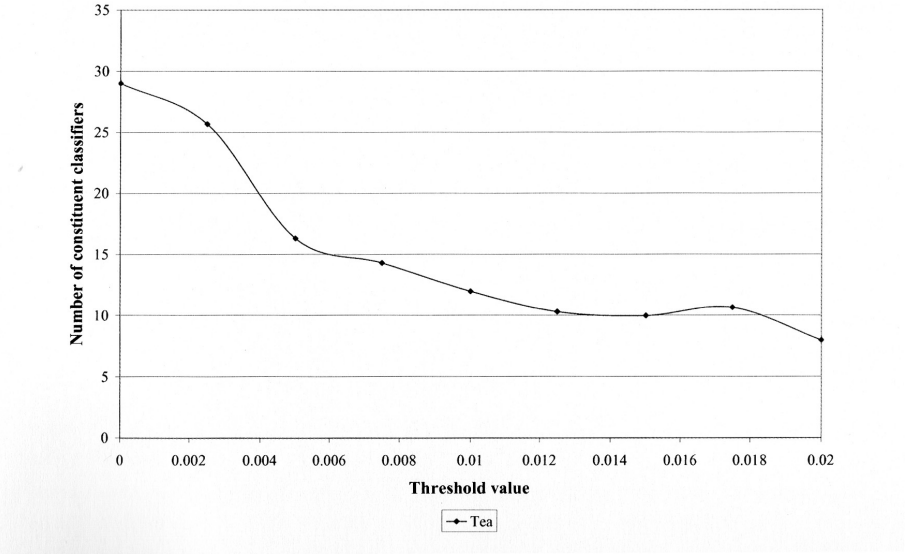


Fig. 17: *Number of constituent classifiers versus Threshold value. This is an expanded view of Figure 16.*

are often capable of producing the same levels of accuracy as the Bagging, Boosting and Stacking ensembles. The fact that the CTE algorithms are based on using different mechanisms from Bagging, Boosting and Stacking to produce these levels of performance suggests that there is merit in these methods of producing diversity amongst the constituent classifiers. In addition to providing similar performance to other approaches the CTE algorithms also produce additional information as to the structure of the problem itself. This extra information about the relationships between classes which can be obtained from the CTE ensembles can be, in and of itself, enough reason to choose a variant of the CTE learner over one of the other ensembles for specific problem domains. Appendix D contains a number of examples which show the information contained in the tree structures.

4 Conclusions

4.1 Summary

A novel approach to Ensemble creation has been proposed, that utilises a number of techniques to promote variation and diversity amongst the constituent classifiers. These techniques are different from those used by the well-known Bagging, Boosting, Stacking and Error Correcting Output Coding ensembles. The methods used by Bagging, Boosting, Stacking and ECOC are listed below:

- Bagging uses sampled variations of the data-set used to train each constituent classifier.
- Boosting varies the data-set used to train each constituent classifier by emphasising the importance of feature vectors that were previously classified poorly. This is performed using re-sampling or sample weighting.

- Stacking uses sampled variations of the training set in a similar manner to Bagging. These are used to train the first layer, whose output is use to create a new data-set that is sampled to train the second layer.
- Error Correcting Output Coding (ECOC) changes the output encoding used to train each of the constituent classifiers. The methods used to determine the output encodings are not, in general, dependent on the problem domain. They are, instead, designed to allow optimal recombination in the presence of noise (errors). These methods however tend to be based on the assumption that errors are evenly distributed amongst the classes being classified. In most problem domains this is not the case.

It should be noted that the constituent classifiers tend to perturb their input data in some random manner in addition to the techniques employed by the ensemble of which they are a part. The term ”sampled variations” covers a number of re-sampling methods including random re-sampling, and cross-validation.

The methods utilised by the Clustered Tree Ensemble (CTE), to create diversity amongst its constituent classifiers, include a number of those used by the Ensembles listed above. These methods are the use of sampled variation, which all ensembles employ in some manner, and of modifying the output encoding. The difference between the CTEs output encoding strategy and that used by ECOC lies in the method by which the class (output) encodings are selected. ECOC uses information theoretic techniques to statically analyse the required outputs. It decomposes them in such a manner that optimal recombination can be undertaken in the presence of randomly distributed errors. However, in the domain of machine learning, the errors are mis-classifications and tend to be far from uniformly distributed. By comparison the CTE encodes its outputs in a manner that is dependent on the actual ”noise” present between specific classes. The noise is measured as the inter class confusion. This approach could provide advantages over the static methods used by ECOC.

A number of experiments were executed to assess the viability of the CTE’s approach to creating a diverse set of constituent classifiers. These experiments compared the performance of four versions of the CTE with three other Ensemble learning techniques; Bagging, Boosting and Stacking. To reduce the effect that the choice of constituent classifier would have on the result, seven sets of classifier ensembles were created and tested each based on a particular type of constituent classifier. The performances of the forty-nine resulting ensembles were compared using seven commonly available data sets and two problem sets that were created for this thesis.

The results of these ensemble comparisons varied depending on the data set and constituent classifiers used. The best overall accuracies, by a small margin, were obtained using Boosting. However for many combinations of data-set and constituent classifier the CTE methods produced comparable or better results. This was generally the case when data sets with larger numbers of classes were examined. This trend was predicted and is due to the nature of the CTE construction process which tends to produce more base classifiers for data sets with larger numbers of classes. Overall the experiments showed that the Clustered Tree Ensemble approach has the ability to produce a level of inter-classifier diversity that is essential for classifier combination techniques to work.

While not always producing the best performance, the CTE provides the additional advantage of associating related classes within a tree structure. It in effect produces extra information about how similar the classes are to one another and presents this in a hierarchical manner. Some examples and explanations of these tree structures are shown in Appendix G.

4.2 Future Research Directions

The research and experiments undertaken for the creation of this document have highlighted a number of areas in which further investigation is warranted. This section discusses and refines these areas and suggests a number of paths, approaches and experiments that future research may follow and perform.

4.2.1 Combining ensemble techniques

The methods used by the CTE to create diversity amongst its constituent classifiers are different from those used by the classifier ensemble methods with which it was compared. This suggests that a greater degree of diversity could be obtained if the techniques from the CTE, Bagging, Boosting and Stacking could be combined in some manner. An obvious approach to investigating this possibility is to combine a number of ensemble classifiers.

All of the ensemble learners investigated are capable of being used as the constituent classifiers within any of the other ensembles. This should allow the benefits of a number of approaches to be combined. For example, Boosting provided the best results on the majority of data sets using the specified base learners, but it does not produce the additional information contained in the tree structure constructed by the CTE. It would therefore be appropriate to use Boosting as a constituent classifier for the Clustered Tree Ensemble. This allows the tree of implicit associations that the CTE creates to still be constructed while also employing the benefits of the Boosting ensemble. The major drawback with this approach is the additional computational and memory requirements imposed by the additional classifiers which would result.

A less resource-hungry method of integrating the diversity creation methods from the two ensemble algorithms would be to combine the techniques each of them uses. One such approach that can be retrofitted to the CTE is to modify the algorithm it uses to remove bias from the data sets it uses to train each constituent classifier. This code currently removes the a priori bias in class numbers present in the data-set constructed for each classifier. Modification could allow the algorithm, in a similar manner to the approach used by boosting, to add a bias weight to all examples in the new data set that were mis-classified by the parent classifier.

4.2.2 Measuring confusion

The technique presented in this document for measuring the inter-class confusion is quite simple. It is based on finding a scalar value for each pair of classes. This value is normalised such that it falls within well specified bounds, as this aids in the user's choice of threshold value. A number of formulas could be used in place of this approach that may provide more information or information of greater reliability.

An undirected graph is built from these measures. Each edge in the graph is assigned a weight proportional to the confusion between the two classes (vertices) that are its end points.

It must be noted that the confusion table from which the inter-class confusion is calculated contains more information than the current measure uses. The current measures assume that the confusion of class A with class B is symmetrical, such that A is as confused with B to the same extent that B is confused with A. However this is not the case. In many circumstances the confusion between two classes is not symmetrical. The information describing the asymmetry is contained in the confusion matrix but is discarded using

the current method. Therefore an obvious enhancement would be to create a directed graph from the confusion matrix. This directed graph would contain two directed edges for each pair of vertices (classes). Each of the directed edges would be associated with one component of the inter-class confusion. In this manner the mis-classifications of class A as class B would be kept separate from the mis-classifications of class B with class A.

This approach would require that the clustering modules currently used to analyse the inter-class confusion be changed to algorithms that can handle directed graphs. A significant drawback to this approach is the fact that it doubles the number of edges that need to be considered when trying to partition the confusion graph into clusters. Part of the investigation would need to consider the trade off between the computational complexity of this approach and the performance (accuracy) benefits, if any, that it achieves.

4.2.3 Clustering methods

The CTE algorithm allows a variety of graph based clustering techniques to be used inter changeably. Two such methods, the Maximum Spanning Tree and Minimum Capacity Cut, are implemented. These two methods produce similar overall classification accuracies for the CTE on a variety of data-sets. The major differences between them are in the resulting structure of the CTE and in the spatial and computational complexity of the two. A number of other approaches to k-clustering exist that could reduce the computational requirements of the CTE algorithm. Such approaches include stochastic solutions to clustering which, while not necessarily producing the same result as the deterministic methods currently employed, can provide approximate solutions in a smaller time period. This could be especially useful if the recommendation in section 8.2.2 of researching the use of directed confusion graphs is undertaken. These graphs will have significantly more edges to process than the current undirected graphs and would therefore benefit more form these kinds of optimisation. It is also possible to simply modify the CTE source code such that clustering algorithms that are not graph based can be used. Future work could include an investigation into the utility of the CTE structures produced by a variety of clustering techniques in a similar manner to the approaches suggested in section 4.2.6.

4.2.4 Re-mapping Based On Clusters

Constituent classifiers in the CTE have their output classifications conceptually re-mapped such that they only produce explicit votes for the classes that they are delegated to be responsible for. These classes are selected by cluster analysis of the performance of the constituent classifier. This conceptual re-mapping is currently implemented by averaging the results for classes for which this constituent is not responsible into a single class called ”Other”. An interesting line of research is to investigate the effect of different methods of performing this re-mapping.

There are a number of methods that could be investigated:

– Retraining the constituent classifier such that it learns the relationship between the feature vectors and the new output mapping. The new mapping is based on the clustering analysis of the performance of the original constituent classifier. A consequence of this approach is that retraining the classifier may remove the basis upon which the new input-output mapping was chosen. The newly trained classifier may therefore not be able to represent this mapping in an optimal manner. This approach could be easily integrated with the trial-and-error iterative pruning techniques described in section 4.2.5.

– Different methods for performing the conceptual re-mapping of the classifier outputs to the expected outputs. The current method averages outputs for classes that this constituent is not responsible for. Other techniques could include a weighted combination of the combined classes.

The investigation should include an examination of the effects of re-mapping techniques on the structure of the tree and the consequences in terms of the classification performance and time taken to train the CTE.

4.2.5 Tree Pruning

The current implementations of the CTE build a tree structure whose size and form results from the interaction between the Confusion Threshold parameter and the performance of the constituent classifiers on the problem data-set. Because of this interaction it is, in general, difficult to determine an appropriate value for the threshold. This often results in trees that are overly expressive or that have too few branches. The Confusion Threshold approach can be considered as a trivial form of iterative tree pruning in that it determines whether branches may or may not be constructed. However it does not examine how the change effects the performance of the CTE. For this reason an investigation into more comprehensive methods of pruning the tree is warranted. If these tree pruning techniques are used it is possible to discard the unpredictable Confusion Threshold parameter and rely on the parameters used by the pruning method which, depending on the approach used, may be of greater relevance to a user. Such pruning techniques could improve the performance of the CTE in terms of both execution speed and classification accuracy by discarding constituent classifiers that do not contribute to, or reduce, the accuracy.

Pruning techniques can be split into two are two general approaches: Global and Iterative.

Global techniques Global pruning methods are applied to the fully constructed tree. They aim to merge unnecessary child branches of the tree into their parents. This is usually performed within the context of achieving an optimal trade off between classification accuracy and the execution speed obtainable when classifying unlabelled examples. A smaller tree should be able to produce a classification in less time than a larger tree. The merges are considered in terms of the overall effect they will have on the tree structure. These approaches can provide better optimisations than those produced by the iterative approaches but, in general, take significantly longer to perform.

Iterative techniques Iterative techniques are generally applied while the tree is being constructed. They are used to decide if a particular branch should be added to the tree. Two approaches exist towards the way in which this decision is made:

– The decision can be made based on an heuristic rule such as that used by the Confusion threshold, which assumes that a lower inter-class confusion will improve the classification accuracy but also increase the number of nodes in the tree. These heuristic rules tend to be fast to evaluate and often do not require retraining of all or part of the CTE.
– The decision can be made based on trial and error. In this case the pruning algorithm assesses the consequences of a particular decision by trying them out. The candidate branch would be added to the CTE and the performance of the CTE with this branch would be compared with the performance before the branch was added. Performance improvements in terms of accuracy, information content or complexity are then used as the basis of the decision.

This results in a greedy algorithm that will not, in general, achieve the best overall result. The advantage when compared to the Global techniques discussed above is the, often significantly, lower time required to make the decision.

4.2.6 Utility of the Tree Structure

An area of this research that warrants further investigation is the the utility of the tree structured categorisations that are produced by the CTE algorithms. The tree structures appear to be accurate representations of the relationships inherent in the data and, as such, provide a number of useful attributes when catagorising text or images for human interaction perusal. These applications of the CTE algorithm were beyond the original scope of the research but emerged as a potential application domain for the CTE methods as an unforeseen result of the approach taken.

Future research should include comparing the tree structures produced by the CTE with the structures produced by techniques that are specifically aimed at providing a solution to this problem. The aim of such comparisons would be to highlight performance deficiencies in the CTE approach.

Acknowledgments

I would like to thank:

- My family, Pam, Peter and Ruth for their support without which I could not have written this thesis.
- Dr. Mike Forshaw and all of the members of the Image Processing Group at UCL for their many useful thoughts and suggestions.
- Dr John Filby and Dr Mark Hodgetts and the staff of Sira Technology Centre for their hard work and dedication.
- The staff of the Advanced Instrumentation Centre UCL
- Unilever and British Steel and the Engineering and Physical Science Research Council.

Sponsored by: Engineering and Physical Science Research Council, Sira Technology Centre.

.

Appendices

A : Base Classifier Configuration

In this appendix the Weka options used for each of the base classifiers are described. For more information about Weka programming environment, please see Witten *et al.* (2017).

A.1 Zero-R

The Zero-R classifier has no configuration options to set.

Weka options: none

A.2 Hyper-pipes

The Hyper-pipes classifier has no configuration options to set.
Weka options: none

A.3 Naive Bayes

The Naive Bayes classifier is trained without using kernel estimators.
Weka options: none

A.4 Voted Feature Intervals

The voted Feature Intervals (VFI) classifier has two options in the implementation used

- Weight by confidence: Enabled - This options enables a simple weighting scheme to be applied to the voting.
- Bias: 0.6 - The exponential bias used by the weighting system.

Weka options: ' -B 0.6'

A.5 C4.5 Decision Tree

The C4.5 classifier has numerous options many of which are not relevant when used in the configuration described below.

- Unpruned: False - Whether or not the tree will be pruned on completion of its training. In this case it will be pruned.
- Reduced Error Pruning: False - Whether Reduced error pruning will be used rather than the stan- dard C4.5 pruning. Standard C4.5 pruning will be used.
- Confidence Factor: 0.25 - The confidence factor used for pruning the tree.
- Minimum Number of Objects: 2 - The minimum number of instances required at each leaf.
- Sub-tree Raising: True - When pruning sub-trees can be raised.

Weka options: '-C 0.25 -M2'

A.6 Decision Stump

The Decision Stump classifier has no options to set. The Decision Stump learner does not produce a probabalistic output.
Weka Options: none

A.7 Neural Network

The Neural network used in the experiments is trained using a simple Back-propagation algorithm. It however has a large number of options that can be set, which can dramatically change its performance on a variety of data sets.

- Layout: 4, 4 - The layout of the nodes in the network. The network is fully connected with two hidden layers both containing 4 nodes. The input layer matches the number of features in the input vector while the output layer contains a node for each possible class.
- Validation Threshold: 20 - The number of times that the error on the validation set can get worse before the network is terminated.
- Normalise Attributes: True - The attributes in the input vector are normalised before presentation to the network. This can result in lower training times.
- Momentum: 0.4 - The momentum term used by the back-propagation algorithm.
- Learning Rate: 0.7 - The amount of the error propagated back though the network while training.
- Decay: True - Allows the learning rate to be reduced as the epoch number increases. This can improve the general performance of the network and helps prevent the network diverging from its output targets.
- Reset: True - Allows the learning rate to be reset to a lower value if the network starts to diverge.
- Validation size: 10 - The size in percent of the test set that will be used for validation. See Validation Threshold.
- Training Time: 1000 - The maximum number of epochs used to train the network. The validation set can be used to terminate the training process before this value is reached.

  Weka Options: `'-L 0.3 -M 0.4 -N 1000 -V 10 -S 0 -E 20 -H "4, 4" -D'`

A.8 Sequential Minimal Optimisation (SMO)

The Sequential Minimal Optimisation (SMO) Support Vector Machine (SVM) classifier is only capable of handling two class problems. To overcome this issue a Multi-class classifier wrapper was used, making the SMO algorithm effectively a output coding ensemble learner. This output coding involves creating an instance of the SMO classifier per output class.

A.8.1 Multi-class wrapper

- Error correction mode: No correction - The type of error correcting output coding to use. No error correction in this case.
- Random width factor: 2.0 - A multiplier used in conjunction with randomised error correction coding. Not applicable in this case as no coding is used.
- Base classifier: SMO - the base classifier to use.

A.8.2 SMO classifier

The SMO classifier has numerous arguments most of which have been left at their default values.

- Cache size: 1000003 - The size of the cache. This should be a prime number.
- Normalise data: True - Normalise the data before use.
- Complexity component (c): 1.0
- Tolerance: 0.001 - The tolerance of the results accuracy.
- Epsilon: 1×10^{12} - The value of epsilon for the round off error.
- Lower Terms: False - Use lower order terms.

Weka Options: '-E 0 -R 2.0 -W SMO - -C 1.0 -E 1.0 -A 1000003 -T 0.0010 -P 1.0E-12'

A.9 Instance Based Classifier (IB1)

The IB1 classifier has no arguments. Unlike most of the other classifiers used it does not produce a probabilistic output it returns just a vote for the most likely class.

Weka Options: none

A.10 Decision Table

- Max Stale: 5 - The number of fully expanded non-improving subsets to consider before ending a best first search.
- Use IBk: False - Use global table majority rather than nearest neighbour.
- Cross Validation: 1 - The cross validation method to use. 1 implies leave one out cross validation.

Weka Options: '-X 1 -S 5'

B : Ensemble Learner Options

The options required to configure the ensemble learners used in the comparison in Chapter 7 are described in this Appendix. The Weka Machine Learning software environment was used as the basis for these experiments and therefore along with general descriptions of the options used the specific configuration strings used by Weka are also given.

B.1 Clustered Tree Ensemble (CTE)

The Clustered Tree Ensemble (CTE) algorithm has a number of options. Those options shown here are the default configuration used as a basis for all experiments unless otherwise stated. This is configuration is labelled CTE-MST in the ensemble performance comparisions.

- Classifier: The base classifier type to use
- Combiner: Averaging - The type of classifier combiner that is used.
- Threshold: 0.0 - The threshold value for discounting interclass confusions from the merge process.
- Use Weights: False - Do not use classifier weighting
- Merge Type: Maximum Spanning Tree Merge - The method used to find clusters at all nodes in the tree.
- Latex Output: - - Do not produce a latex file describing the tree produced.

Weka options: '-S 0 -M 0.0 -O 1 -L - -B false -K 0 -W <base classifier>'

B.1.1 CTE-MST-W

Clustered Tree Ensemble using Maximum Spanning Tree clustering with classifier weighting enabled.

- Use Weights: true - Weight the outputs from the individual base classifiers according to thier performance on the internal test-set.

Weka options: '-S 0 -M 0.0 -O 1 -L - -B true -K 0 -W <base classifier>'

B.1.2 CTE-MCC

Clustered Tree Ensemble using recursive Minimum Capacity Cut clustering

- Merge Type: Minimum Capacity Cut Merge - The method used to find clusters at all nodes in the tree.

Weka options: '-S 0 -M 0.0 -O 1 -L - -B false -K 1 -W <base classifier>'

B.1.3 CTE-MCC-W

Clustered Tree Ensemble using recursive Minimum Capacity Cut clustering with classifier weighting enabled.

- Merge Type: Minimum Capacity Cut Merge - The method used to find clusters at all nodes in the tree.
- Use Weights: true - Weight the outputs from the individual base classifiers according to thier performance on the internal test-set.

Weka options: '-S 0 -M 0.0 -O 1 -L - -B true -K 1 -W <base classifier>'

B.2 Bagging

The Bagging Ensemble Learner has three options:

- Number of Iterations: 5 - The number of bagged classifiers to train.
- Bag size in percent: 100 - The size of each bag as a percentage of the number of entries in the training set. Each bag is created by randomly sampling this number of entries from the training set.

Weka options: '-S 1 -I 5 -P 100 -W <base classifier>'

B.3 Boosting - AdaBoost.M1

The AdaBoost algorithm has three options of relevance to the experiments:

- Resample: false - Boosting is performed using weights on data set entries rather than dataset resampling.
- Maximum number of Iterations: 5 - The maximum number of iterative training cycles that the boosting algorithm will perform.
- Weight Threshold: 100 - The percentage of the weight mass to use for training each classifier.

Weka options: '-P 100 -I 5 -S 1 -W <base classifier>'

B.4 Stacking

The Stacking ensemble classifier is slightly different from the other ensemble classifiers described in this Appendix in that it allows multiple different types of base classifier to be used together along. These fall into two catagories: the meta or second stage classifier and a selection of primary classifiers.

- Meta Classifier: The classifier to be used as the meta classifier. For all experiments this was the same as the classifier selected for the stage one classifier.
- Number of folds: 5 - The number of times to rtain each stage one classifier type.
- Stage one Classifier: A selection of classifiers can be entered here. In the experiments a sinlge classifier of the same type as the Meta classifier is entred here.

Weka options: '-B "base classifier" -X 5 -S 1 -M <base classifier>'

The '-S' parameter used by all ensemble learners sets the seed to the random number generated used by each.

C : Ensemble Accuracy

The figures in this appendix show box plots of the accuracy obtained for each Ensemble type using a variety of base classifiers. A set of graphs is drawn for each data-set. To ease assessment of the relative performances of the ensembles the results are grouped first by the type of base classifier used such that all ensembles using a specific base classifier type on a chosen Data-set are side by side. The results are next grouped by the Data-set used to produce them.

- All graphs for a specified data-set use the same ordinate range.
- The results for ensembles based around the Zero-R classifier have been removed from these dia- grams as the comparatively low accuracies produced using this learner would skew the range that the plots represent reducing their intelligibility.
- Circles are used to represent values outside 150% of the interquartile range of the box plot.
- Alongside each box plot is a plot of mean and variance.
- For each data-set there are a number of tables which contain the confidence values produced by computing the pairwise t-test with Bonferroni adjustment on Ensembles that use the same base classifier. A value of 1.0 indicates a high confidence of two result sets being indistinguishable. A value approaching zero indicates low similarity between the result sets.
- Complete pairwise t-tests that include all combinations of ensemble learner and constituent classifier type are not shown due to their extremely large size.

Examination of this data allows a preliminary assessment of the similarity of result sets to be made.

For detailed results please see Appendix F of Fudge's the PhD thesis Fudge (2010).

D : Tree Structures

This appendix contains some examples of the tree structures created by running the Clustered Tree Ensemble on a number of data sets. It is not possible to show all of the tree

structures produced when performing the experiments described in Section 3 as many thousands were created. The trees were created from two data sets chosen to allow easy human interpretation of the results. The first is the Gaussian data set and the second is the Letter recognition data set.

D.1 Gaussian Data Set

The image of the data set shown in Figure G.1 shows four obvious groupings of the classes:

- Group A consists of Class 1.
- Group B consists of two classes: Class 2 and Class 3.
- Group C consists of three classes: Class 4, Class 5 and Class 6.
- Group D also consists of three classes: Class 7, Class 8 and Class 9.

Because of the artificial nature of this data set it is known that the number of data sets that are in the incorrect group is minimal.

The first two example trees are shown in Figures 19 and 20. These diagrams show the results of running the the CTE algorithm using the Minimum capacity cut and Maximum spanning tree clustering modules on the 9 point Gaussian data set. The base classifier used is Naive Bayes, which is well suited to this problem's domain in that its underlying modeling of the data matches the model used to create the data set.

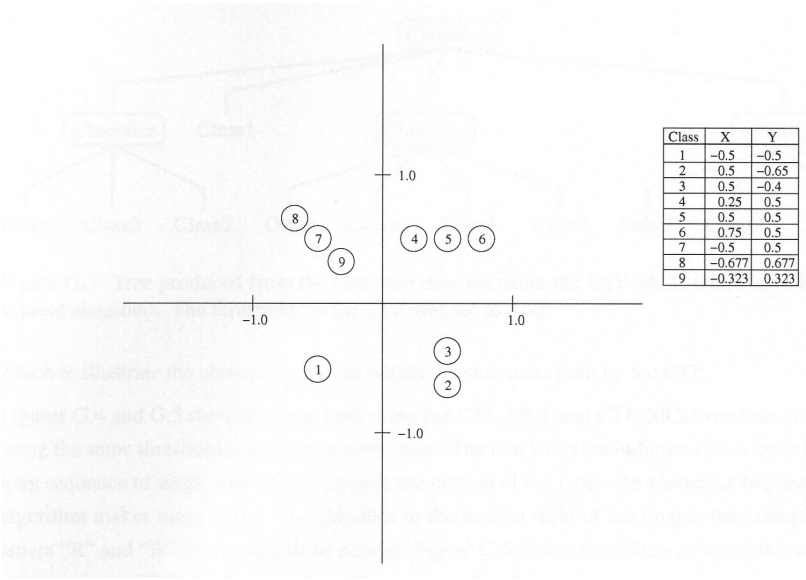
Both figures show that the class groupings visible in the data set are reflected in the tree structures produced. Figure 19 shows the results from using the Maximum Spanning Tree clustering module. This tree is is overly large due to the threshold parameter being set to a value that is too low. Setting the threshold parameter to low values decreases the tolerance for inter-class confusion, resulting in a large tree. Setting the threshold parameter to a larger value will reduce the number of classifiers in the resulting tree. Although, in this case the threshold value is poorly chosen, the class groupings are still visible in the tree structure. Each of the groups described above are represented by branches from the root classifier.

Figure 20 shows the tree produced using the Minimum Capacity Cut clustering module. In this structure the class groupings are more obvious than in the tree in figure 19. Each group is represented by a classification node rooted at the classifier node that represents the problem domain.

This tree was produced using the same threshold parameter as was used for the CTE-MST example above however, due to the differences in the algorithms, this threshold produces an optimal tree for this problem. It should also be noted that the MCC module is less sensitive to changes in the threshold parameter than the MST module. The MCC module, for this reason, tends to produce trees that are more consistent in structure over multiple experiments.

D.2 Letter Recognition Data Set

The features in the Letter recognition data set are derived simple image processing operations performed on noisy images of print written letters in a variety of fonts. Because of the problem's basis on such a familiar subject (humans being very good as distinguishing between letters), it is a useful data-set with which to illustrate the class relationships within the structures built by the CTE.



Class	X	Y
1	−0.5	−0.5
2	0.5	−0.65
3	0.5	−0.4
4	0.25	0.5
5	0.5	0.5
6	0.75	0.5
7	−0.5	0.5
8	−0.677	0.677
9	−0.323	0.323

Fig. 18: *Centroids of the 9 Point Gaussian Data Set. The circles show the standard deviations of the class groupings.*

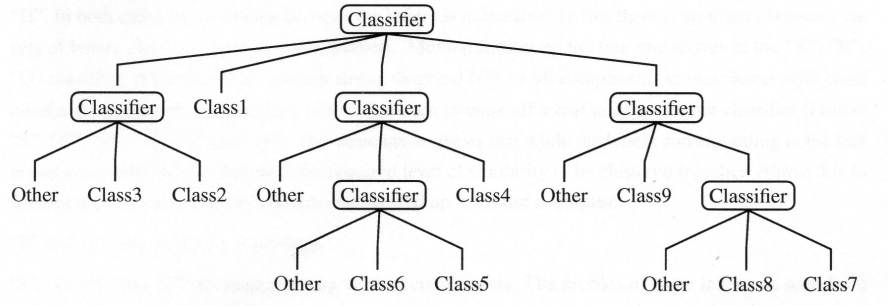


Fig. 19: *Tree produced from the Gaussian data-set using the CTE-MST with Naive Bayesian constituent classifiers. The threshold for the CTE was set to 0.02.*

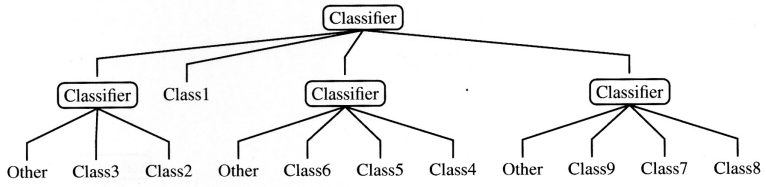


Fig. 20: *Tree produced from the Gaussian data-set using the CTE-MCC with Naive Bayesian constituent classifiers. The threshold for the CTE was set to 0.02.*

Figures 21 and 22 show the trees built using the CTE-MST and CTE-MCC versions of the algorithm using the same threshold parameter in both cases. The tree structure in figure 21 is quite sparse, with a long sequence of single classifiers. Towards the bottom of the figure the clustering imposed by the CTE algorithm makes more sense. The classifier to the bottom right of the image, for example, shows that letters ”R” and ”B” are thought to be similar. Figure 22 shows the effects of using the same threshold value with the CTE-MCC algorithm. The tree produced by this experiment has a very high ratio of leaves to classifiers and does not produce any clear clusters. It does show a tendency for its bottom-most classifier to handle letters with enclosed areas, such as ”O” and ”D”, while the root classifier handles those letters with strong central diagonal components and no closed spaces, such as ”M”, ”W” and ”Y”.

Decreasing the threshold value used by the CTE-MCC algorithm to 0.003 doubles the number of constituent classifiers created for this problem. Figure 23 shows the tree structure created using this threshold value. It can clearly be seen that similar shaped letters are being grouped together. For an example look at the third and fourth classifiers down from the top. Between the two of them they classify letters ”L”, ”J”, ”I”, ”F”, ”E”, ”B”, ”P” and ”D”, which share many similar characteristics. This is especially true when one considers the features extracted from the letter images which contain poor descriptors of the letters. One such measure is to draw a line across a part of the letter and count how many times the letter and line ”touch”.

Figure 24 shows the tree structure produced using the CTE-MCC algorithm on the Letter recognition data set. It this instance the threshold was set to 0.002. This leads to a tree that is almost as sparse as that shown in figure 21. However this figure, perhaps more than any of the others, demonstrates the clustering undertaken by the CTE algorithm. Starting at the bottom of the figure and working our way up we have have two classifiers, one for the letters ”G” and ”E” and one for the letters ”O”, ”D”, ”B” and ”H”. In both cases the similarity between the letters is noticeable. In fact there is similarity between the sets of letters classified by both these learners. Moving further up the tree one arrives at the ”K”, ”R”, ”Q” classifier. All three letters contain strong diagonal NW to SE components in their lower right hand quadrants. A number of classifiers follow that each spawns off a leaf node and a sub classifier (Letters ”S”, ”C”, ”Z”, ”X”, ”L” and ”N”). This structure suggests that while the letters corresponding to the leaf nodes are similar they do not meet the required level of similarity to be clustered together. Above this in the tree there are a number of clusters that showed up in earlier examples:

- ”P” and ”F”: the similarity is obvious.
- ”I”, ”J”, ”T” and ”U” all contain strong vertical components. The profiles of these letters, as seen from the side, are also very similar.
- finally ”A”, ”M”, ”W”, ”Y” and ”V” all contain strong diagonal components.

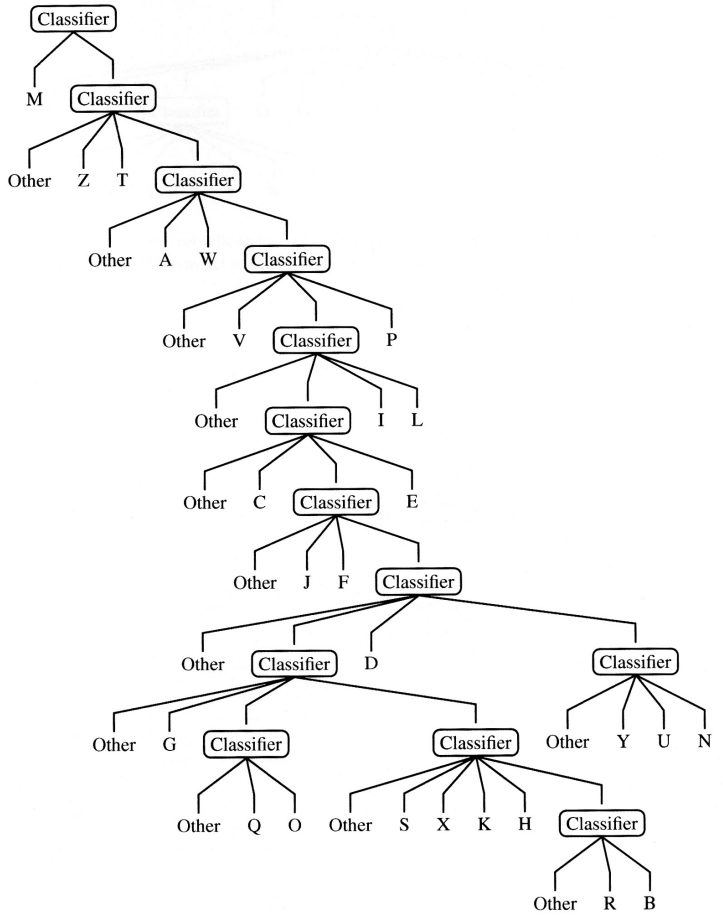


Fig. 21: *Tree produced from the Letter data-set using the CTE-MST with C4.5 constituent classifiers. The threshold for the CTE was set to 0.002.*

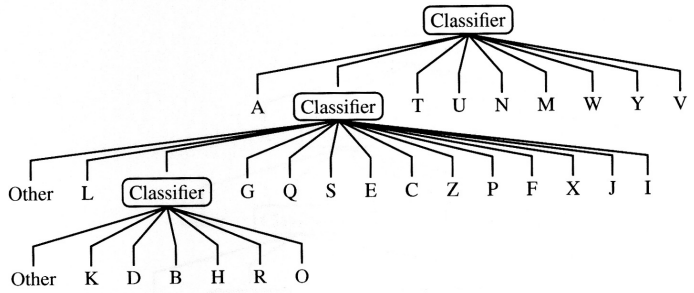


Fig. 22: *Tree produced from the Letter data-set using the CTE-MCC with C4.5 constituent classifiers. The threshold for the CTE was set to 0.005.*

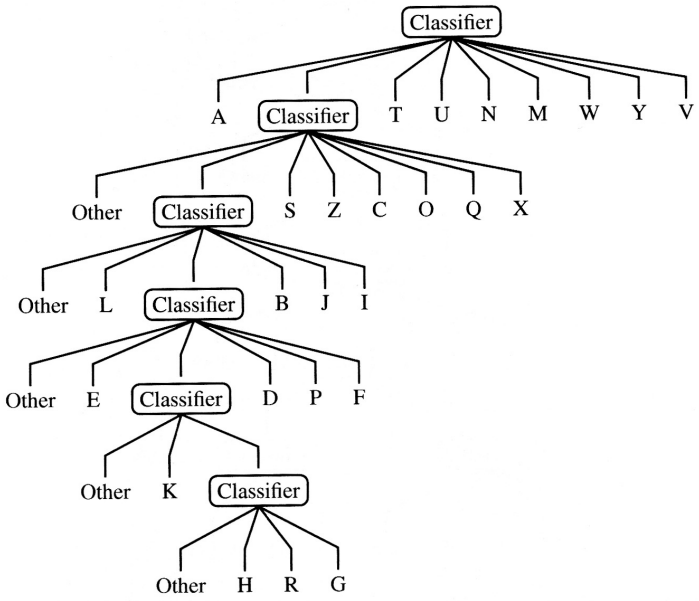


Fig. 23: *Tree produced from the Letter data-set using the CTE-MCC with C4.5 constituent classifiers. The threshold for the CTE was set to 0.003*

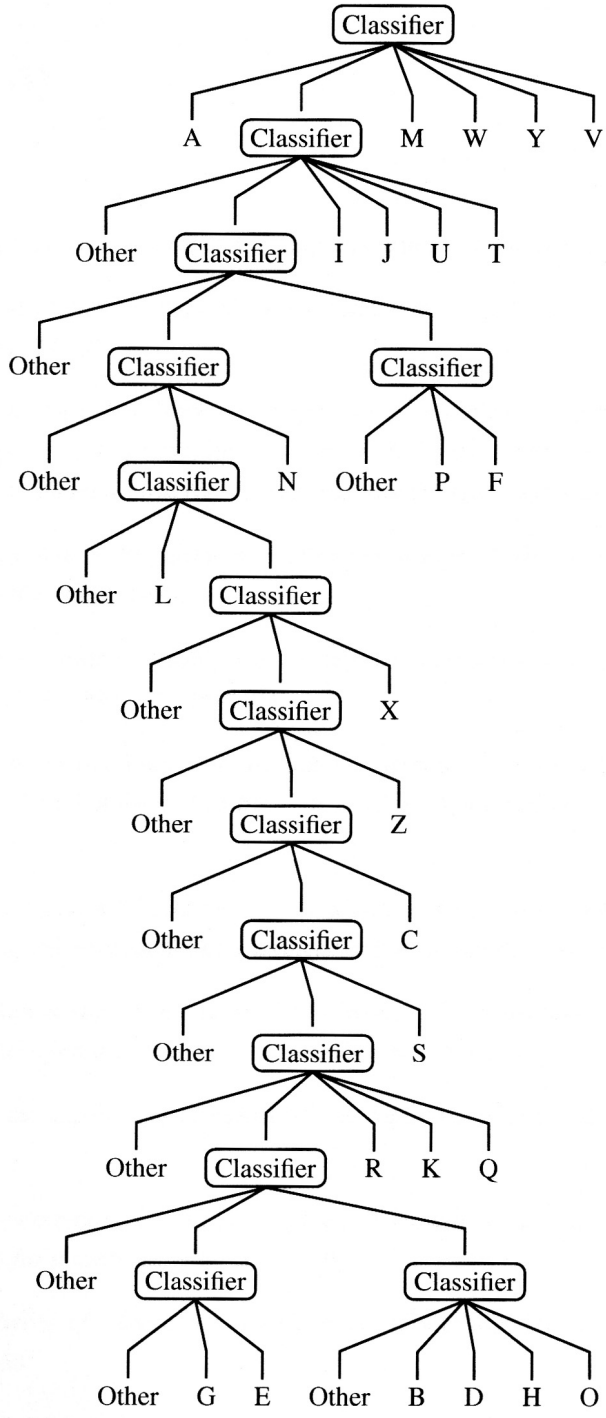


Fig. 24: *Tree produced from the Letter data-set using the CTE-MCC with C4.5 constituent classifiers. The threshold for the CTE was set to 0.002.*

References

Alkoot, F. and Kittler, J. (1999). Experimental evaluation of expert fusion strategies. *Pattern Recogn. Lett.*, **20**(11-13), 1361–1369.

Ascheuer, N., Jünger, M., and Reinelt, G. (2000). A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Computational Optimization and Applications*, **17**(1), 61–84.

Black, D. (1987). *The theory of committees and elections*. Springer.

Blake, C. and Merz, C. (2017). Uci repository of machine learning databases.

Bradshaw, B. (2000). Semantic based image retrieval: A probabilistic approach. In *Proceedings of the Eighth ACM International Conference on Multimedia*, MULTIMEDIA '00, pages 167–176, New York, NY, USA. ACM.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Brodley, C. and Lane, T. (1996). Creating and exploiting coverage and diversity. In *In Work. Notes AAAI-96 Workshop Integrating Multiple Learned Models*, pages 8–14.

Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, **2**(2), 63–73.

Chang, S.-F., Smith, J., Beigi, M., and Benitez, A. (1997). Visual information retrieval from large distributed online repositories. *Commun. ACM*, **40**(12), 63–71.

Clemen, R. (1989). Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, **5**, 559–583.

Cohen, P. (2017). *Empirical Methods for Artificial Intelligence*. MIT Press.

Demiröz, G. and Güvenir, H. (1997). *Classification by Voting Feature Intervals*, pages 85–92. Springer Berlin Heidelberg, Berlin, Heidelberg.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, **29**, 103–130.

Drucker, H., Schapire, R., and Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 42–49. Morgan-Kaufmann.

Frank, E., Hall, M., Holmes, G., Kirkby, R., Pfahringer, B., Witten, I. H., and Trigg, L. (2010). *Weka-A Machine Learning Workbench for Data Mining*, pages 1269–1277. Springer US, Boston, MA.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, pages 148–156, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Fudge, M. (2010). *Automated Hierarchical Construction of Multiple Classifier Systems*. Ph.D. thesis, University College London.

Goemans, M. and Papaefthymiou, M. (1991). *Advanced Algorithms: Lecture Notes For: 18.415/6.854*. Research seminar series. Laboratory for Computer Science, Massachusetts Institute of Technology.

Goldberg, A. and Tarjan, R. (1988). A new approach to the maximum-flow problem. *J. ACM*, **35**(4), 921–940.

Gomory, R. and Hu, T. (1961). Multi-terminal network flows. *SIAM Journal on Applied Mathematics*, **9**(4), 551–570.

Grotschel, M., Monma, C., and Stoer, M. (1995). Design of survivable networks. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 617 – 672. Elsevier.

Hansen, L. and Salamon, P. (1990). Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, **12**(10), 993–1001.

Hao, J. and Orlin, J. (1992). A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '92, pages 165–174, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Hiroshi, N. and Toshihide, I. (1992). Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, **5**(1), 54–66.

Huang, J. (1998). *Color-spatial Image Indexing and Applications*. Ph.D. thesis, Ithaca, NY, USA. AAI9838769.

Huang, J., Kumar, S., and Zabih, R. (1998a). An automatic hierarchical image classification scheme. In *Proceedings of the Sixth ACM International Conference on Multimedia*, MULTIMEDIA '98, pages 219–228, New York, NY, USA. ACM.

Huang, J., Kumar, S., and Zabih, R. (1998b). An automatic hierarchical image classification scheme. In *Proceedings of the Sixth ACM International Conference on Multimedia*, MULTIMEDIA '98, pages 219–228, New York, NY, USA. ACM.

Jain, A., Murty, M., and Flynn, P. (1999). Data clustering: A review. *ACM Comput. Surv.*, **31**(3), 264–323.

Jensen, D. and Cohen, P. (2000). Multiple comparisons in induction algorithms. *Mach. Learn.*, **38**(3), 309–338.

Junger, M., Reinelt, G., and Rinaldi, G. (1995). The traveling salesman problem. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Handbooks on Operations Research and Management Science: Network Models*, pages 225–330. North Holland, Amsterdam, The Netherlands.

Junger, M., Rinaldi, G., and Thienel, S. (2000). Practical performance of efficient minimum cut algorithms. *Algorithmica*, **26**, 172–195.

Kohavi, R. and Kunz, C. (1997). Option decision trees with majority votes. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 161–169, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Krogh, A. and Vedelsby, J. (1994). Neural network ensembles, cross validation and active learning. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, pages 231–238, Cambridge, MA,

USA. MIT Press.

Kruskal, J. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, **7**(1), 48–50.

Langley, P. (1988). Machine learning as an experimental science. *Machine Learning*, **3**(1), 5–8.

Lienhart, R. and Hartmann, A. (2002). Classifying images on the web automatically. *Journal of Electronic Imaging*, **11**, 11 – 11 – 10.

Merz, C. (1999). Using correspondence analysis to combine classifiers. *Machine Learning*, **36**(1), 33–58.

Mutzel, P. (1995). A polyhedral approach to planar augmentation and related problems. In *Proceedings of the Third Annual European Symposium on Algorithms*, ESA '95, pages 494–507, London, UK, UK. Springer-Verlag.

Nagamochi, H., Ono, T., and Ibaraki, T. (1994). Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming*, **67**(1), 325–341.

Padberg, M. and Rinaldi, G. (1990). An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, **47**(1), 19–36.

Prechelt, L. (1996). A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice. *Neural Networks*, **9**(3), 457 – 462.

Prim, R. (1957). The shortest connecting network and some generalisations. Technical Report J36, Bell. Syst. Tech.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Raudys, S. and Jain, A. (1991). Small sample size effects in statistical pattern recognition: recommendations for practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**(3), 252–264.

Salzberg, S. (2000). On comparing classifiers: A critique of current research and methods. *Data Mining and Knowledge Discovery*, **1**.

Schapire, R. and Freund, Y. (2014). *Boosting Foundations and Algorithms*. MIT Press.

Schapire, R., Freund, Y.and Barlett, P., and Lee, W. (1997). Boosting the margin: A new explanation for the effectiveness of voting methods. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 322–330, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(8), 888–905.

Stoer, M. and Wagner, F. (1997). A simple min-cut algorithm. *J. ACM*, **44**(4), 585–591.

Tichy, W., Lukowicz, P., Prechelt, L., and Heinz, E. (1995). Experimental evaluation in computer science: A quantitative study. *J. Syst. Softw.*, **28**(1), 9–18.

Wang, K., Zhou, S., and Liew, S. (1999). Building hierarchical classifiers using class proximity. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 363–374, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Witten, I., Frank, E., M.A., H., and Pal, C. (2017). *Data Mining: Practical Machine Learning Tools and Techniques*. Elsevier.

Wolpert, D. (1992). Original contribution: Stacked generalization. *Neural Netw.*, **5**(2), 241–259.

Wu. and Leahy, R. (1993). An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **15**(11), 1101–1113.

Xu, D. and Tian, Y. (2015). A comprehensive survey of clustering algorithms. *Annals of Data Science*, **2**(2), 165–193.

Xu, L., Krzyzak, A., and Suen, C. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, **22**(3), 418–435.

Yu, K., Jiang, X., and Bunke, H. (1997). Lipreading: A classifier combination approach. *Pattern Recognition Letters*, **18**(11-13), 1421–1426.