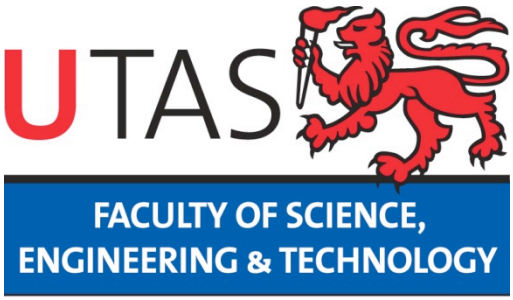


School of Computing
& Information Systems

DynamicWEB: A Conceptual Clustering Algorithm for a Changing World

Joel David Scanlan, BComp. (Hons)

A dissertation submitted to the Faculty of Science, Engineering and Technology, University of Tasmania in fulfilment of the requirements for the Degree of Doctor of Philosophy.

June, 2011

I STATEMENT OF ORIGINALITY

This thesis contains no material which has been accepted for a degree or diploma by the University or any other institution, except by way of background information and duly acknowledged in the thesis, and to the best of my knowledge and belief no material previously published or written by another person except where due acknowledgement is made in the text of the thesis, nor does the thesis contain any material that infringes copyright.

Date:

II STATEMENT OF AUTHORITY OF ACCESS

This thesis may be made available for loan and limited copying in accordance with the *Copyright Act 1968*.

Date:

III STATEMENT OF CO-AUTHORSHIP

The publications of the work undertaken in the course of this research are the following:

Scanlan, JD and Hartnett, JS and Williams, RN (2005). Identifying Reconnaissance Activity: A Strategy for Network Defence. Proceedings of 6th Australian Information Warfare & Security Conference, 24-25 November 2005, Geelong, pp. 210-215

- Mr. Joel Scanlan (80%) is the primary author. He proposed the initial research question, conducted the research and prepared the material for publication.
- Jacky Hartnett (10%) and Dr Raymond Williams (10%) of the School of Computing and Information Systems, University of Tasmania, both provided general guidance and editing advice as supervisors.

Scanlan, JD and Hartnett, JS and Williams, RN (2006) Dynamic WEB: Profile Correlation Using COBWEB. Proceedings of the 19th Australian Joint Conference on Artificial Intelligence, 4-8 December, 2006, Hobart, Tasmania, pp. 1059-1063

- Mr. Joel Scanlan (80%) is the primary author. He proposed the initial research question, conducted the research and prepared the material for publication.
- Jacky Hartnett (10%) and Dr Raymond Williams (10%) of the School of Computing and Information Systems, University of Tasmania, both provided general guidance and editing advice as supervisors.

Scanlan, JD and Hartnett, JS (2008). A context aware scan detection system, International Journal of Computer Science and Network Security, 8 (No1, January 2008)

- Mr. Joel Scanlan (80%) is the primary author. He proposed the initial research question, conducted the research and prepared the material for publication.

- Jacky Hartnett (10%) and Dr Raymond Williams (10%) of the School of Computing and Information Systems, University of Tasmania, both provided general guidance and editing advice as supervisors.

Scanlan, JD and Hartnett, JS and Williams, RN (2008). DynamicWEB: A method for reconnaissance activity profiling. Proceedings 2008 International Symposium on Parallel and Distributed Processing with Applications (ISPA-08), 10-12 December 2008, Sydney Australia

- Mr. Joel Scanlan (80%) is the primary author. He proposed the initial research question, conducted the research and prepared the material for publication.

- Jacky Hartnett (10%) and Dr Raymond Williams (10%) of the School of Computing and Information Systems, University of Tasmania, both provided general guidance and editing advice as supervisors.

Scanlan, JD and Hartnett, JS and Williams, RN (2008). DynamicWEB: Adapting to concept drift and object drift in COBWEB. Proceedings 21st Australasian Joint Conference on Artificial Intelligence , 1-5 December 2008, Auckland, New Zealand, pp. 454-460.

- Mr. Joel Scanlan (80%) is the primary author. He proposed the initial research question, conducted the research and prepared the material for publication.

- Jacky Hartnett (10%) and Dr Raymond Williams (10%) of the School of Computing and Information Systems, University of Tasmania, both provided general guidance and editing advice as supervisors.

Scanlan, JD and Hartnett, JS and Williams, RN (2009). Machine Learning in a changing world. Proceedings of 3rd International Workshop on Artificial Intelligence and Technology (ASISAT), 23-24 November 2009, Hobart.

- Mr. Joel Scanlan (80%) is the primary author. He proposed the initial research question, conducted the research and prepared the material for publication.

- Jacky Hartnett (10%) and Dr Raymond Williams (10%) of the School of Computing and Information Systems, University of Tasmania, both provided general guidance and editing advice as supervisors.

We the undersigned agree with the above stated proportion of work undertaken for each of the above published manuscripts contributing to this thesis.

Signed: ……..

Date: ……..

Mrs Jacky Hartnett

Supervisor

School of Computing and Information Systems

University of Tasmania

Signed: ……..

Date: ……..

Dr Raymond Williams

Supervisor

School of Computing and Information Systems

University of Tasmania

IV ABSTRACT

This research was motivated by problems in network security, where an attacker often deliberately changes their identifying information and behaviour in order to camouflage their malicious behaviour. Addressing this problem has resulted in a new adaption to the unsupervised machine learning technique COBWEB.

In machine learning and data mining the aim is to extract patterns from data in order to discover a meaning underlying the processes that are taking place. In most cases, each object is observed once, and then the patterns that have been extracted can be used to classify newly-observed objects. Conceptual clustering aims to do this in such a way that the patterns that are learned are human readable. Concept drift algorithms allow concepts to change over time, although most undertake this in a supervised manner, which presents a challenge when looking for novel classes.

This research focuses on the classification of objects that change over time across multiple observations. The objects may change their own characteristics (labelled as object drift in this research) or maintain the same characteristics, but change their identifier. In addition to this, it is also possible for the concept that describes a group of objects to itself change (known as concept drift). In addition to the possible application within the security domain, the method was generalised and tested across a range of machine learning and data mining domains. In the process it was shown that the method was robust in the presence of concept drift, which occurs when a group of objects that define a given concept change their characteristics, resulting in the definition of that concept having changed over time.

The ideas of concept drift and object drift are not only relevant within the computer security field, but can be of significance in any knowledge domain. Therefore, any method presented to address this learning problem should be generalised enough to be applicable in many application areas.

The new method, entitled DynamicWEB, extends the existing conceptual clustering method COBWEB to allow for profiles to be added and removed from the concept hierarchy. An index structure was implemented using an AVL tree to facilitate fast scalable searching of the knowledge structure. As the target objects change over time the profiles of each target are updated within the structure, maintaining an up-to-date

representation of the domain. The profiles contain derived attributes, which are formed across multiple observations of each object, with the aim of retaining knowledge of how the object has changed over time. As well as preserving context over time, Dynamic Web uses multiple trees and so, transforms the learner into an ensemble classifier.

In addition to testing the method on the security and network based datasets, a number of other datasets are also examined. A new dataset (a modified version of Quinlan's weather dataset) is presented in order to illustrate how Dynamic Web operates in the presence of object drift. The method is also tested on several well-known machine learning datasets, some of which exhibit concept drift. Along with these artificial datasets, a group of real-world datasets, including several sourced from the Australian Bureau of Statistics, were also examined, illustrating DynamicWEB's ability to adapt to change.

This thesis describes the work done to enable DynamicWEB to adapt to both concept drift and object drift, both of which are characteristic of many application domains. DynamicWEB is also capable of profiling an object across multiple observations to allow for accurate prediction and inter-object relationship discovery.

V ACKNOWLEDGEMENTS

My supervisors, Jacky and Ray, for their continued support and advice over the years of my PhD candidature. Their assistance in undertaking the work, this thesis and the research papers that were produced, was invaluable. Without them I would not have been able to undertake this research.

I would also like to thank Mike Cameron-Jones for his advice at several points during the research and also for proof reading a sizable portion of this thesis.

I would also like to thank Dr Rob Edmondson and Kevin Manderson for allowing me access to several network based audit logs which were used in my research.

My wonderful wife Meg, without whose love and support I doubt I would ever have been able to undertake this research. Thanks also to both of our families for being a support network to both of us while we undertook our doctoral studies.

Thanks to my two office mates over the last 4 years or so: Phil and Matt. Thanks for putting up with my awesome sense of humour and for many fun times late at night.

The Cows and FGI honours/post-grad/lecturer cricket teams for the outlet away from research each week, or the LAN parties over the years. The same goes for my Virtual World crew for weekly lunches and movie nights.

My Lord and Saviour, Jesus Christ, who is with me always.

VI TABLE OF CONTENTS

VII LIST OF TABLES

VIII LIST OF FIGURES

Chapter 1

Introduction

“Nothing in the world is permanent, and we're foolish when we ask anything to last, but surely we're still more foolish not to take delight in it while we have it. If change is of the essence of existence one would have thought it only sensible to make it the premise of our philosophy.”

William Somerset Maugham (1874 - 1965)
The Razor's Edge, 1943

1.1 INTRODUCTION

The world around us is changing. In effectively every domain of knowledge, in the realms of science and industry and in people's personal lives, change occurs in some manner, over time. Weather is monitored nightly on the news; thousands of people are employed monitoring various parts of a nation's economy; millions world-wide earn a living from changes within the stock market; and engineers monitor infrastructure. Some change is sudden, such as the recent Global Financial Crisis, or events in our lives such as the birth of a child. Others happen more slowly, such as Climate Change or the process of aging within the human body. Observing and trying to understand change is the basis for many professions and areas of investigation.

From a computational perspective, observing and recording change, in an effort to understand it, has been carried out since the development of computers. The first stage in this process is to record data for later analysis. Machine learning is a field of computing that aims to develop methods for discovering patterns within data. Data mining is the process of applying these methods to large repositories of stored data with the aim of extracting knowledge. In the fields of machine learning and data mining, many researchers have examined datasets that contain change in various forms.

1.2 MOTIVATION

The work reported in this thesis was inspired by a learning problem within the field of computer security. The knowledge domain is one in which many observations of a given user's activity are recorded over a significant time period. This information, when viewed in context, presents a profile of activity that can then be used to determine whether the user is carrying out a specific type of behaviour. The security sub-field that inspired this research was related to a specific type of port scanning reconnaissance.

Port scanning reconnaissance involves multiple observations of users who scan a network, over the course of a time period. The users' behaviour changes over the time period within which these observations are taken. Activity which is defined as malicious may change over the observed time (concept drift), while also a user's

activity may go from being considered benign to that of a threat (object drift). For an effective comparison to be made between these activity profiles there is a need for the context of these multiple recorded activities to be preserved.

Within the joint fields of machine learning and data mining there is an apparent lack of learning methods which observe the same objects, multiple times, over a time window. Across these observations, it is possible for the objects that are being observed to change in some way, and it is important for this change to be incorporated into the learning model.

1.3 RESEARCH AIMS

The research presented in this thesis aims to operate within the learning scenario outlined above. To enable this, the aims for the learner are as follows:

1. The learner needs to be able to profile object activity over an extended time period.
2. The learner needs to be able to establish relationships between these profiles.
3. The learner needs to be able to adapt to concept drift.
4. The learner needs to be able to adapt to object drift.
5. The leaner needs to be able to preserve context across multiple observations.
6. The learner needs to be able to track a large number of target objects simultaneously in real-time.

These six aims outline a method that is highly adaptive with the ability to profile objects over time. The fundamental element of the learner is the production of a profile that is based on the data obtained from multiple observations of target objects over time (1). This data is a recorded history of behaviour exhibited by the target object.

As each of these profiles is built upon data relating to individual objects within a group, it is highly beneficial to be able to relate these objects to one another (2). By discovering relationships between the different objects under examination, patterns

can then be extracted from the dataset, and these patterns can then be used to model the dataset. This dataset may not have defined classes (as with the application which inspired this research), and because of this, the learner needs to be an unsupervised technique.

If the target objects[1] are changing over time, then there are two forms of drift that the learner needs to be able to adjust to: concept drift and object drift (3 and 4)[1]. Concept drift occurs when the class description of a group of objects changes over time. For example, the definition of what is considered fashionable among a group of people may change over time as trends come and go. Object drift occurs when a target object migrates from one resultant concept to another, for example when a given person changes from being in one fashion clique of people to another.

As many observations and updates occur to the profiles relating to each target object, there is an over-arching context that needs to be preserved (5). This context is a historical one representing the past behaviour of the given object. Being able to preserve the fact that four of the numerical attributes of a particular object have been decreasing in value, over time, while a fifth attribute has been increasing in value, provides a significant benefit when it comes to classifying an object as opposed to just storing the most recent observed value of each attribute. Such preservation of contextual information is vital if behaviour is to be profiled over time.

The final aim listed above is for the learner to be able to profile a very large number of objects at once. While in many application domains the targets of interest will be few in number, the application that drove the need for this learner examines thousands of target objects simultaneously and so scalability is very important. Further, this knowledge domain, operates on live data and needs to operate in real-time to respond to any threats upon the network.

Because the machine learning method developed as part of this research will be operating in a learning environment that is quite different to that in which most methods operate it is important for it to be designed with more than just one

[1] Definitions of these terms used throughout the thesis can be found in the Appendix A

application in mind. One of the main aims of this research is to produce a learning method that is generalised and applicable to a wide range of domains.

1.4 THESIS OUTLINE

The following is an overview of the chapters within this thesis

The next chapter is a brief literature review of clustering techniques within the machine-learning field, to provide some background to the way in which learning techniques extract knowledge from a dataset. This chapter will introduce the machine learning area of *Conceptual Clustering*.

The third chapter is an examination of clustering methods that have been developed to examine datasets in which change occurs during the learning process. These methods aim to adapt to this change and learn from it. The chapter will introduce the topic of *Concept Drift*, which is one of the most active areas of research within machine learning for methods that adapt to change, and the most relevant to the research described in this thesis.

The fourth chapter will describe two probabilistic conceptual clustering techniques. One of these, entitled COBWEB, has been modified in order to carry out some of the research described in this thesis. Research by other authors, also building upon this method, will be briefly examined in this chapter.

The fifth chapter describes, in detail, the new method, entitled DynamicWEB. This method has been developed to carry out the research described in the thesis. It initially outlines the motivations and goals for the method and then explains the extensions that were made to the COBWEB algorithm in order to meet these goals.

The sixth chapter examines the performance of the COBWEB implementation used within DynamicWEB and then examines DynamicWEB as applied to other machine learning datasets, one of which was created specifically for this work, while the others have been used by other machine learning researchers examining similar problems. This chapter focuses on small easily understood datasets. The remainder of the chapters examine larger datasets that are not real world problems.

The seventh chapter displays DynamicWEB's performance upon several real-world datasets, including two provided by the Australian Bureau of Statistics and one derived from a well-known data mining contest held several years ago.

The eighth chapter examines DynamicWEB's performance on two network-based datasets. One is the scan correlation dataset that originally inspired the research and the other is a dataset detailing network performance on a network spread across the states and territories of Australia.

Finally the conclusion chapter draws together the results that were described within the previous three chapters and also discusses directions for further work.

Chapter 2

Clustering Techniques

"If you leave things alone you leave them as they are. But you do not. If you leave a thing alone you leave it to a torrent of change."

G. K. Chesterton (29 May 1874 – 14 June 1936)
Orthodoxy, 1908

INTRODUCTION

In the first chapter of this thesis several goals were outlined with respect to a classification scenario involving behaviour profiling. This chapter will look at data and conceptual clustering. Data clustering is the more traditional form of clustering, and is the more frequently used technique in data mining. Conceptual clustering is however more suited to behaviour profiling, and it will be examined and contrasted with its more traditional counterpart. This is done as a precursor to the next chapter, which will examine conceptual clustering methods that adapt to change over time, and the chapters that follow, which introduce the new method being presented. This method is built upon COBWEB, an Incremental Hierarchical Conceptual Clustering Algorithm, which will be discussed in detail in Chapter 4.

2.1 DATA CLUSTERING

Clustering methods aim to discover the natural groupings of instances (items with multiple data attributes) within a given dataset. Clustering is a data mining approach in which the eventual clusters are a simplification of the data into a model. These clusters are effectively subsets within the population, with the grouping being determined by shared characteristics between instances. This model can then be utilised for classification or visualisation of the dataset. The clusters that are located are usually previously unknown, and it is through the use of these techniques that the relationships between instances (ie. patterns) are discovered.

A diverse range of clustering techniques has been developed since the late 1960s. An

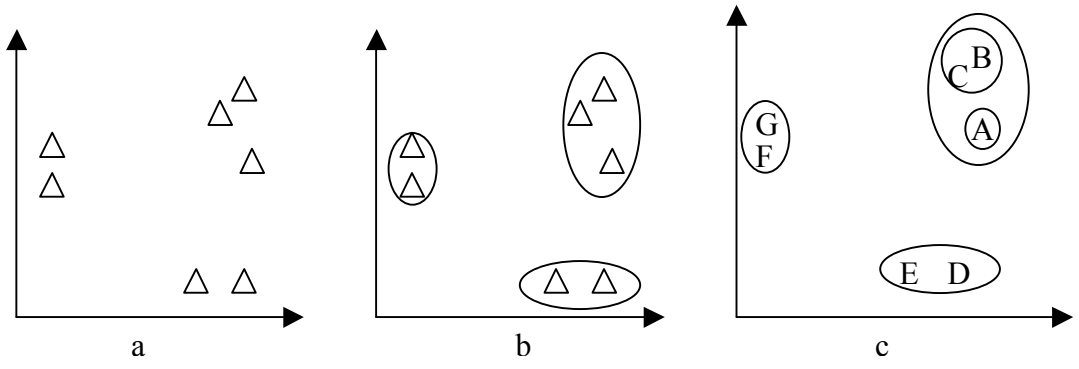


Figure 1. a) A data series. b) The resulting clusters from a partition technique. c) The resulting tree from a hierarchical technique.

obvious and relevant division between the methods occurs between partitional and hierarchical methods. Hierarchical methods create a series of partitions, nested one within the other, in a tree structure. Conversely, partitional methods create a single partition between the resulting clusters. Figure 1 is a basic illustration of the difference between the two methods based on the dataset shown in (a). The two methods will now be examined separately.

2.2 PARTITIONAL CLUSTERING

Partitional clustering creates only a single partition within the data (as shown in Figure 1). The result of this is a simple structure compared to the nested partitions produced by a hierarchical system, thus requiring less computation per instance. This means that it is well suited to applications with large datasets, and is often used in engineering applications. This approach will now be outlined.

Given a dataset of n instances, or objects[2], the aim is to discover a partition that results in the creation of K clusters, or subsets. Each instance should be most similar to the other instances within its assigned cluster and least similar to those in the other clusters. The size of K is often fixed, although this is not true in all methods. The choice of K is very important as it governs the output produced; too high a value of K results in output that is too fine or over fitted to the problem, too low a value and the output can be too coarse and with multiple actual clusters within a single cluster found by the technique. Methods for finding the ideal value of K have been the subject of extensive research (Dubes 1987; Tibshirani, Walther et al. 2000; Salvador and Chan 2004).

The initial starting values of the cluster centres, or seed points, are associated with the size of K. These starting values also have a large impact on the outcome of the algorithm (Figure 2), and, as a result, multiple runs with different starting values are often trialled until the best values are discovered (ie. those seed points that most effectively cover the area). A common approach is to use a selection of K objects at random from within the dataset to be the seed points. Future runs then select other K

[2] Also referred to commonly in the literature as patterns. This term is not used within this thesis to avoid confusion with patterns extracted from multiple observations in methods discussed in other chapters.

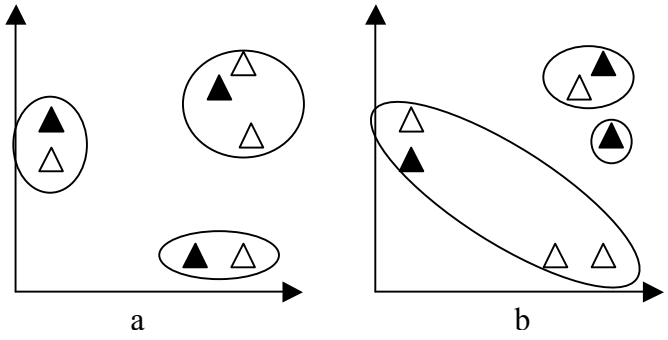


Figure 2. The seed points are the coloured triangles; (a) illustrates the clustering that results from the ideal seed points. While (b) is the clustering that results from poorly chosen seed points.

values, and runs are repeated until the best are found. An alternative option to this is a method in which points within the range of the object parameters are selected as the seed points.

Once the initial values of the seed points have been defined the instances within the dataset can all be allocated to the one with which they are most similar. When all the instances have been assigned, a criterion function to calculate the "goodness" of the partition is generated. The centroid value of each cluster is calculated based upon the instances within each cluster (this occurs at different times within the various approaches). This value then replaces the seed point for each cluster. After the centres have been calculated, merges and cluster reassignments can occur, followed by a re-calculation of the criterion function and the centroid. This is repeated in a loop until convergence of the partition occurs. A highly important element of this process is the criterion function, used to measure the quality of the existing partition. The most frequently used criterion within partitional clustering techniques is the Squared Error. Squared Error will be discussed here, but many other criteria have been used (Milligan 1981).

Squared Error is a cumulative measure of error across all n instances in K clusters and is expressed as the following:

$$e_K = \sum_{j=1}^{K} \sum_{i=1}^{n_j} \left\| x_i^{(j)} - c_j \right\|^2$$

where $x_i^{(j)}$ is the i^{th} instance (currently assigned to the j^{th} cluster) and c_j is the centroid of the j^{th} cluster. Therefore for each instance the difference between it and the centroid of its host cluster is calculated; with the total value of e_K being the total difference for all instances in all clusters. Various partitional methods treat this value differently: some wait for the value to converge before ending while others have a threshold value of change between rounds before ending. A very common partitional method that makes use of the Squared Error criterion is the K-Means Clustering Algorithm, which will now be discussed in detail.

2.2.1 THE K-MEANS CLUSTERING ALGORITHM

The k-Means approach was proposed by MacQueen (1967) and has since been a very active area of research (AnderBerg 1973; Hartigan 1975). It is still frequently used not only in research (with endless variants) but also in benchmarking other methods (Ordonez, 2003). K-Means is a very simple method that largely follows the basic partitional clustering structure outlined above in 2.2. It has a time complexity of O(n) which makes it appealing, and is not difficult to implement. MacQueen (1967) outlined multiple k-Means variants within his landmark paper: *Some Methods of Classification and analysis of multivariate observations*, Table 1-Table 3 detail these methods.

1)	Select *K* cluster centres; using the first instances in the dataset, or randomly selected instances or locations within the range of possible instances.
2)	Assign each instance to the nearest cluster and then re-calculate the centroid of the cluster. Repeat for all *n* instances.
3)	After all *n* instances have been added take the current centroids and fix them as new seed points. Pass back through the dataset assigning all *n* instances to the nearest seed point.

Table 1. Simple k-Means Algorithm

The simplest method outlined by MacQueen is the simple k-Means method in Table 1. It is very similar to the summary method detailed in 2.2 and, other methods proposed by Forgy (1965) and Jancey (1966). A key difference between MacQueen's method and those of Fogy and Jancy is that the centroid calculation occurs after each instance is added, rather than after all the instances have been added. Furthermore simple k-Means by MacQueen does not continue until convergence, but for just the

1)	Select *K* cluster centres; using the first instances in the dataset, or randomly selected instances.
2)	Assign each instance to the nearest centroid and then re-calculate the centroid of the cluster. Repeat for all *n* instances
3)	Take each instance and assign it to its nearest cluster. If that cluster is not the one it was placed in during Stage 2 then place it in the new cluster and update the centroids of the new and old clusters.
4)	Repeat Stage 3 until convergence is reached, or until all *n* instances are cycled through without a change occurring.

Table 2. Convergent k-Means Algorithm

one cycle and one re-allocation, unlike that of Fogy and Jancy. However, MacQueen's second version of k-Means is one that does implement a convergence process and is detailed in Table 2.

Convergence in Forgy, Jancey and MacQueen, and indeed many other methods, can mean several things. The first is the simplest measure where a reallocation phase is completed without a single instance changing clusters. This means that all instances are in their nearest cluster, and further cycles through the dataset will not make any more changes to the partition. The second option for judging convergence is with the use of the Squared Error measure. If, after another reallocation phase, there has been a minimal change to the value of e_K, or it is low enough to be under a threshold, then convergence is judged to have occurred. Both methods give an assurance that the best partition, using those seed points, has been achieved.

The third method outlined by MacQueen (Table 3) is similar to the first method in that this variant did not continue to convergence (in MacQueen's version, other people have implemented this using the same methods as mentioned above). However, it instead introduced the possibility of a *K* value that changes during splitting and merging of clusters through the introduction of two distance thresholds *C* and *R*.

The first threshold is the coarsening parameter (represented by a *C*). This value is the minimum distance that two clusters can be apart. If the distance between two centroids is smaller than this value then they are merged to make a single cluster. The provision of merging of clusters avoids two virtually identical clusters forming

1)	Define values for *K*, *C* and *R*
2)	Select *K* cluster centres; using the first instances in the dataset, or randomly selected instances.
3)	Calculate the distances between the *K* seed points; if the distance between two seeds is less than the coarsening parameter *C* then merge the two seeds. Continue with this step until all seed points are separated by more than a distance of *C*.
4)	Add each of the *n* instances, as each instances's nearest centroid is located, if the distance is greater than the refining parameter *R*, then create a new cluster with the instance as its centroid. If the distance is less than *R* then add to the nearest cluster and re-calculate the centroid. Calculate the distance between this centroid and all other clusters. If the distance is less than *C* then merge the two clusters.
5)	After all *n* instances have been added take the current centroids and fix them as new seed points. Pass back through the dataset assigning all *n* instances to the nearest seed point.

Table 3. k-Means with Coarsening and Refining Algorithm

directly beside each other due to poor initial selection of seed point. The second threshold is known as the refining parameter (represented by *R*). The refining parameter creates a new cluster for instances that are of a distance greater than R, the closest existing centroid. The aim of this is to remove the effect of an outlier on the centroid of a cluster.

MacQueen was not the first to introduce merging and splitting in partitional clustering: the ISODATA method by Ball and Hall (1965) predates it. Their method of splitting and merging is not performed as an instance is added, but rather is based upon the variance just prior to the instance re-allocation. However, it was similar in that it also used a parameter-based threshold approach to heuristically choose which clusters needed merging or splitting.

This section has described the process of partitional clustering methods with a focus on the k-Means method. This was discussed not only because k-Means is a very commonly used algorithm, but also because it is referred to in 1.6.1 where a partitional method and a hierarchical method are hybridised.

2.3 HIERARCHICAL CLUSTERING

Hierarchical clustering, as already mentioned briefly, is a clustering technique in which more than a single partition is constructed, but further partitions are nested within each other forming a tree with branches and leaves at the furthermost points. Each branch is in effect a cluster which is partitioned from the rest of the data. Figure 1 (see page 8) uses a graph to illustrate a tree that was created using a hierarchical clustering process. However, as it may not be immediately obvious which instances relate to which parts of the tree on Figure 1, a modified version is now presented below as Figure 3.

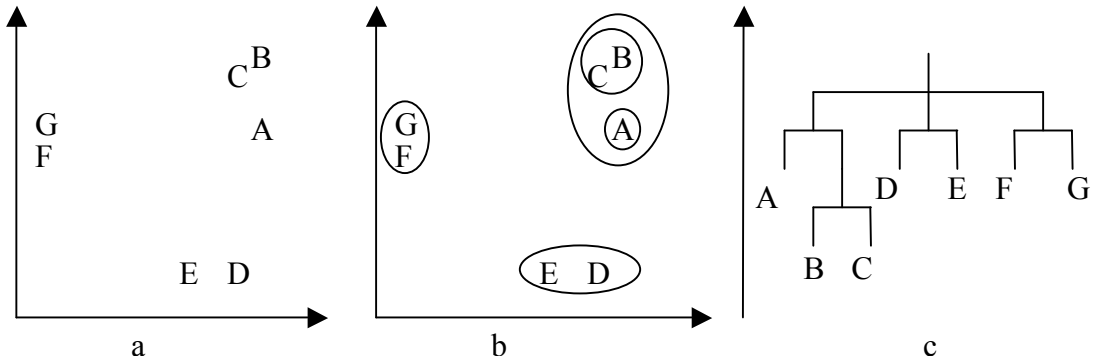


Figure 3. a) A data series; now expressed with identifier labels for each instance. b) The graph representation of the nested partitions produced by the hierarchical method c) A dendrogram displaying the identifiers for each instance in their resulting location.

The dendrogram in Figure 3c shows the tree that has been created, and illustrates the relationships between different instances. The pairs of D and E and F and G are very similar to each other, and are therefore clustered together in sibling leaves. Conversely, the cluster that contains A, B and C has a wider variation in the represented values. As B and C are more similar to each other than they are to A, a child node, or cluster, is created to capture this. It can be seen that the resulting output from a hierarchical algorithm is quite human readable, and possibly more so than its partitional counterpart.

Hierarchical clustering, like partitional, has several main components which comprise the approaches that are modified to produce the different variants. These are the mode of construction, and the clustering method. Two modes of construction are commonly used: Agglomerative and Divisive methods. Clustering methods

generally fall into one of the following types: single linkage, complete linkage, average linkage or sum of squares (Jain, Murty et al. 1999).

Of the two main hierarchical clustering modes of construction types agglomerative is the more common. Table 4 outlines the way this method operates (AnderBerg 1973). It starts with all the *n* instances being grouped in their own individual cluster. Then, step by step, the clusters are merged until all instances are in a single node. This is known as the root node.

1)	Begin with *n* clusters; each containing one instance. Propagate the proximity matrix with the distance, or similarity, measures.
2)	Using one of the clustering methods (in conjunction with the proximity matrix) locate the two clusters that have the greatest similarity (labelled c1 and c2). Merge the two Clusters.
3)	Reduce the number of clusters by one; update the proximity matrix for c1 and delete c2 from the matrix.
4)	Repeat steps 2 and 3 until all instances are merged together into a single root cluster.

Table 4. Agglomerative Clustering Method

At each step the two clusters that are merged are the two determined to be most similar according to a similarity matrix developed in conjunction with a relevant clustering method (as discussed below). The divisive method operates in the reverse order (Jain, Murty et al. 1999). Instead of merging disjointed clusters together, all of the instances start in the root node and are split apart until all the instances are in their own cluster. The choices used to perform the split, as with the merges in the agglomerative method, are based upon the clustering method.

2.3.1 CLUSTERING METHODS

The most important part of a hierarchical clustering technique is the specific clustering method that is used. As mentioned above there are four main types used: single linkage, complete linkage, average linkage or sum of squares. Other methods are usually variations of these four. The clustering method expresses the relationship strength, or similarity, of clusters to one another. Within partitional clustering, instances are assigned to the closest cluster as they were added. In hierarchical methods the instances are already present, and the clustering choice is purely about splitting and merging the clusters.

The Single Linkage method is the simplest of the common methods of hierarchical clustering. The measure uses the distance, or correlation, between the closest two instances in two neighbouring clusters. In distance-based forms of the method it is the closest two instances within the cluster (minimum distance), while in correlation forms of the method it is the two instances that have the most in common (maximum similarity). We can generalise both forms and reduce it to the two instances that have the shortest distance or strongest link across the two clusters. Figure 4a illustrates two different pairs of comparisons in dimension space between three clusters. This method has one downfall: it clusters purely in relation to the closest member of the cluster, and so can at times result in long chain like structures when examined in dimension space. This means that instances which are clustered together can actually be quite dissimilar (AnderBerg 1973).

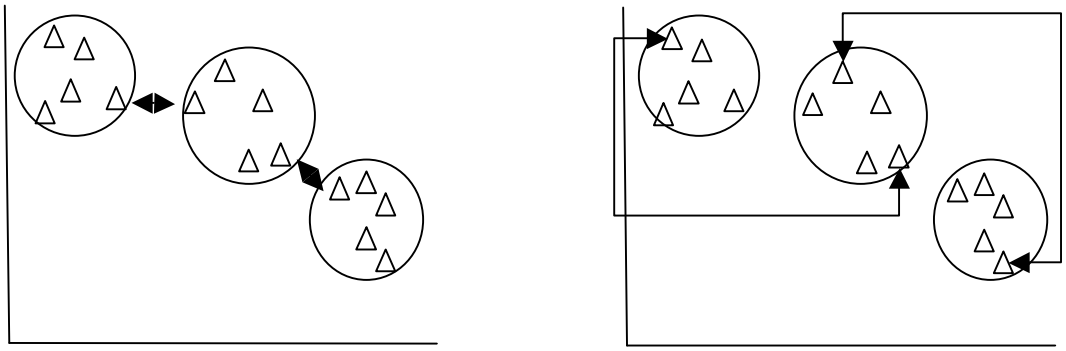


Figure 4. a) Single Linkage Clustering Method: the closest two instances within each cluster are compared. b) Complete Linkage Clustering Method: the furthest two instances within each cluster are compared.

The Complete Linkage Method is similar to the Single Linkage Method and is illustrated by comparing two single instances in neighbouring clusters in Figure 4b. However, unlike the Single Linkage Method where the most similar instances are compared, in the Complete Linkage Method the most dissimilar instances within the two clusters are compared. In effect, the comparison shows the full span of the possible merged cluster. The smaller the value, the closer the two clusters are to each other, and therefore the more suited to merging they are. While this overcomes the chaining problem present in the Single Linkage Method, it tends to be too conservative and results in poorly separated clusters (Hansen and Delattre 1978).

Both the Single Linkage and Complete Linkage methods are very simple, but each has drawbacks as well as benefits. The limitation with both methods is their reliance upon a single instance within a cluster, and that the instance used is an edge instance on the cluster. This causes some bias in the estimation of how close the cluster is to other neighbouring clusters. It is this problem which the Average Linkage Method aims to overcome. Instead of calculating the distance or similarity between two clusters by examining the distance between two instances, the Average Linkage method calculates the average of all pair wise distances between the two clusters. Using the data series in Figure 4 as an example, all the distances for each of the 5 instances in each cluster to the 5 instances in the neighbouring cluster are calculated. These are all then averaged (5 measurements per instance, across 5 instances in this case). This value is then used to judge the distance or similarity of the two clusters.

2.4 CONCEPTUAL CLUSTERING

In previous sections data clustering has been discussed in relation to the partitional and hierarchical approaches. Both of these techniques, and indeed the vast bulk of other clustering approaches, focus on some form of numerical distance measure between the instances presented. The learning that occurs is based upon the use of this distance measure between instances. As such, they can be described as "learning by example" (Fisher (1987). These systems also tend to give equal weight to all attributes, and do not take into account the relevance or irrelevance of some attributes in a clustering outcome (Michalski 1980). Conceptual clustering differs quite markedly from data clustering in that each cluster has a description based upon the instances it is assigned. As such, the cluster has an identity based upon the commonality that is present within the instances at that node. As it is through these observations that the clustering occurs, conceptual clustering can be described as "learning by observation" (Fisher 1987).

Conceptual Clustering was first outlined by Michalski (1980) and was further expanded upon in multiple joint publications with Stepp (1981; Michalski, Stepp et al. 1981; Michalski and Stepp 1983). Conceptual clustering aims to produce concept descriptions for each class. This then allows for clusters to have a simple conceptual interpretation based upon these descriptions. Data clustering methods, while often useful for many things such as classification, are not as simple to interpret or fully

understand. The goal of conceptual clustering extends beyond that of data clustering to not only discover the relationships within the data, but also to discover human readable clusters. Furthermore, the aim is for these classes to fit descriptions which illustrate a true "is-a" (subclass-of) relationship. To aid in achieving this, conceptual clustering techniques often make use of a hierarchical structure. As each of the descriptions, or concepts, are formed they are placed in the tree. Within this structure the broader concepts are located towards the root, with more specific concepts nested within those higher parent concepts, as children. Figure 5 is an illustration of a concept hierarchy showing the possible clusters that could be discovered from a fictitious dataset about motorised vehicles. The *Car* and *Truck* concepts are each a child of the broader concepts of *Road Vehicle* and obviously the root, *Vehicle*.

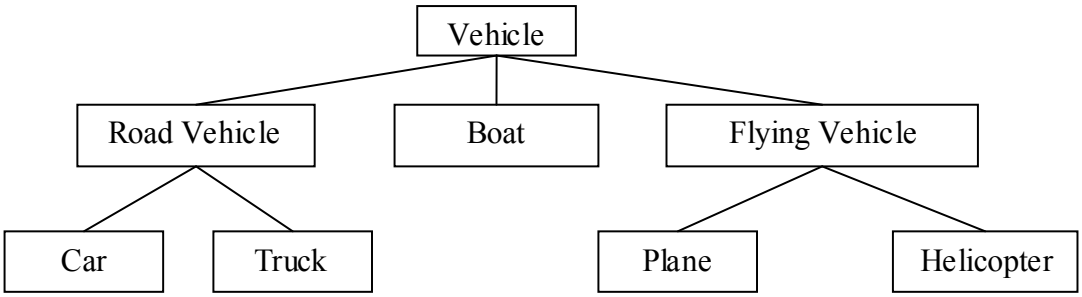


Figure 5. A illustration of a concept hierarchy motor vehicles

A significant amount of the work in conceptual clustering has been undertaken by AI researchers together with researchers from the cognitive psychology field (Gluck and Corter 1985; Medin, Wattenmaker et al. 1987). There has been a great deal of work done on the way humans learn, both in supervised and unsupervised environments, and so there is a natural relationship between the two fields. In psychology, clustering is referred to as "sorting". Observations by psychologists have found that human sorting techniques differ greatly from the data clustering methods already outlined in this chapter. Instead of sorting based upon the differences between a range of attributes, humans use only a few. This results in a few attributes deciding the class of an instance and the remainder being largely ignored (Medin, Wattenmaker et al. 1987). It is these simple, and yet strangely effective, properties that conceptual clustering aims to model. A classic example of extracting a simple concept is shown in Figure 6 (Michalski 1980).

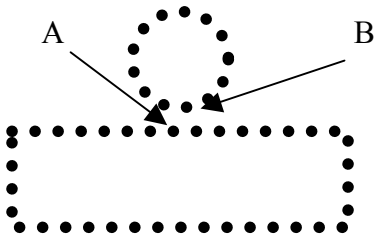


Figure 6. Clustering in Concepts, not pairwise distance (Michalski 1980).

In a data clustering paradigm the cluster membership of points A and B would be decided based upon their pairwise distance. However, when a human views the scenario they immediately see the concepts of a circle and a square. These two shapes could be one inside the other, or overlapping, and yet a human will simply see the shapes, and not be overly concerned about their colour, size or other attributes. This is an example of a simple concept description that still effectively joins multiple instances together as a single entity.

Conceptual clustering is therefore able to undertake two different modes of learning: clustering and characterisation. In clustering, the goal is to produce groups of instances which are similar, while characterisation aims to determine useful concepts among the objects present, that are associated by meaning; essentially a process of concept formation.

In the area of concept discovery there are both supervised and unsupervised methods. Some of the models produced have been the result of a direct collaboration between cognitive psychologists and AI researchers, as discussed above. Among the many models produced there are two main types of conceptual clustering methods: Conjunctive and Probabilistic.

2.4.1 CONJUNCTIVE CONCEPTUAL CLUSTERING

Conjunctive methods aim to produce a simple logic expression to serve as the cluster description that fits a collection of objects. This conjunctive statement is similar to those produced within decision trees; however, the goals and methods for producing these are dissimilar and they have markedly different computational complexity (Fisher 1987). It can be noted that ID3 (Quinlan 1986) itself also had links to the psychology field, having being developed from the Concept Learning System

(CLS) system proposed by Hunt, Marin and Stone (1966).

The foundational work done in the conceptual clustering area by Michalski and Stepp (1981) made use of a conjunctive method called PAF[3]. This method was the first to join together the two subtasks of clustering and characterisation in a fully automated fashion. This is achieved by the way in which the concept descriptions are made, and the way they are represented. The main components of PAF are the representation scheme, representation and allocation functions, and evaluation criterion. A discussion of each of these components and the algorithm description follows.

2.4.1.1 THE REPRESENTATION SCHEME

The purpose of the Representation Scheme is to characterise the objects that are within a cluster. Within PAF there are two representation schemes: a preliminary scheme based upon the seed of the cluster; and a conjunctive statement that describes the objects within the cluster. This conjunctive statement, referred to as a logic complex (called VL_1), is an expression derived from the earlier work by Michalski (1974) in the variable value logic system.

Within a dataset of n objects each object has a set of variables $x_1, x_2, .., x_n$. Each of the variables has a domain, $d(x_i)$, which details the range of possible values for that variable. The number of those values is given as d_i. If the domain of a variable states that it is numerical, it also details the range of valid numbers. Likewise, a nominal variable also details a list of valid values. For example a domain depicting colour would be represented in this way: $d(x_i)$ = {blue, red, green, orange}. Using this representation of a variable, x_i, and its related domain information, a conjunctive statement can be formed. The statement, or complex, is comprised of one or more logic units called a selector. A selector can be represented as

$$[x_i \# R_i]$$

where xi is the variable of interest, R_i is a reference to one or more values from within the variables domain and # represents a relational operator. For example,

[3] Polish-American-French

1)	Select *K* initial seeds. They can be randomly chosen or based upon some criterion.
2)	For each seed determine the *star* of *m* complexes that are maximally general, but do not cover any of the other seeds. If the number of complexes generated exceeds *m* then an evaluation criterion (LEF discussed on page 23) selects the best ones to remain.
3)	For each *star* remove all unnecessary values from each complex. i.e those such that when removed the complex still covers the same observed objects[4].
4)	From each *star* a single complex is selected such that the resulting set of complexes cover the entire data range and are mutually disjoint.
5)	The clustering is evaluated using LEF across all *n* objects. On the first iteration the clustering is stored, every iteration after that the clustering is only stored if it is superior. The algorithm terminates after a specified number of iterations occur without improvement.
6)	From each complex a new seed is selected and the algorithm iterates again from step 2.

Table 5. PAF Algorithm

the selector [colour = blue, red] is satisfied whenever colour has the value blue or red. Likewise [width < 20] is satisfied whenever the value of width is less than 20. Individually the selectors are quite simple, but when joined across all of x_n a conjunctive statement can illustrate concepts quite effectively.

[colour = blue, red] [width < 20] [weight = 2 .. 10] [length ≠ long]

Within the complex the selectors are merged together with an implicit "and" between them. The above complex describes a concept that is either blue or red, with a size less than 20, weight that is between 2 and 10 and with a length that is not long.

2.4.1.2 THE REPRESENTATION FUNCTION

The representation function (Steps 1-3, 6) determines a set of *K* disjoint complexes, referred to as $\alpha_1, \alpha_2, .. \alpha_K$, for a set of *K* seeds, $e_1, e_2, .. e_K$ from the complete data set of *E*.

[4] Michalski and Stepp refer to the objects as "events" within their work. The term "objects" has been used here for consistency with the other algorithm descriptions earlier in the chapter.

1. Complex α_i covers seed e_i and none of the other seeds,

2. The union of all α_K covers all of the dataset to be clustered E,

3. All of $\alpha_1, \alpha_2, .. \alpha_K$, maximise the evaluation criterion.

The complexes are generated using the calculation of what is termed a *star*. A *star* is a set of complexes that cover a single seed to the exclusion of all the other seeds. The star function is represented as

$$G\left(e_i \mid F, m\right)$$

where G is the set of complexes, e_i is the seed, F is the set of all other e_k seeds discounting e_i, and m is a integer threshold. The star is a set of no more than m complexes, sorted based upon the evaluation criterion. If the number of complexes in G exceeds m then the worst-rated complexes based upon the evaluation criterion are removed. Once the star has been created the highest rated complex is chosen as the complex to represent the seed.

This function is a major component of the algorithm, and is itself rather computationally expensive. Initially the seed selection is done randomly from within E. However, after that, there are two seed selection methods used. Initially central objects that fit the maximum number of properties within α_i are chosen. However, when cluster improvement does not occur, border objects that match only a minimum number of properties in a complex are selected. This occurs until the algorithm terminates, which takes place after a specified number of iterations occur with no improvement.

2.4.1.3 THE ALLOCATION FUNCTION

The allocation function (Step 4) is far simpler than its representation counterpart. Where the latter creates the complexes, $\alpha_1, \alpha_2, .. \alpha_K$, the allocation function uses these complexes to form the clustering $C_K = \{ c_1, c_2, .. c_K\}$. Where c_i contains all observed instances from E that fit α_i.

2.4.1.4 THE EVALUATION CRITERION

The evaluation criterion specifies, and aims to guarantee, certain qualities within the clusters and thus within the concepts produced. It is used throughout PAF in most steps, but most notably in steps 2 and 5. The method, as implemented by Michalski and Stepp (1980, 1981), allows the user to maximise one or more of the following four measures: fitness between data and clusters, inter-cluster differences, essential dimensionality and simplicity of representations.

- The fitness between the clusters and the data is a measure of the sparseness of the clusters with the minimum sparseness as the preferred fitness.

- The inter-cluster difference is measured by the sum of degrees of disjointedness between every pair of complexes in the clustering. This measure is a count of the number of selectors within the complexes (after selectors which intersect have been removed). Maximising this criterion promotes long descriptions covering non-intersecting variable values.

- The essential dimensionality is defined as the numbers of variables which independently divide the set of complexes. i.e they are present in complexes, but contain different values in each selector. Such differences are enough to differentiate between multiple clusters.

- The simplicity of cluster representations describes a count of the number of sectors that are within all complexes.

The above criteria are combined together to form a single measure called the Lexicographical Evaluation Functional with tolerances (LEF) (Michalski 1980). The LEF is represented as follows:

$$\left(c_1, \Delta_1\right), \left(c_2, \Delta_2\right) .. \left(c_i, \Delta_i\right)$$

where c_i is one of the four criterion (as already described) and Δ_i is the tolerance threshold (0 to 100%). All clusters are first evaluated on c_1, and those that score higher than Δ_1 are retained, and then evaluated on each of the other criteria in turn. At each evaluation, those clusters that meet the threshold value are retained. This process continues until there is only a single cluster remaining (i.e the best one) or

there are no more criteria remaining. In this later case all remaining clusters are of acceptable quality and can be chosen. The final choice for the resultant cluster can be made based upon the ordering of the remaining clusters using quality measures.

2.4.2 PROBABILISTIC CONCEPTUAL CLUSTERING

The ground work for the area of conceptual clustering was largely carried out by Michaski and Stepp (1980, 1981). Other researchers have since developed a number of conceptual clustering methods that are probabilistic in design, and not conjunctional. The probabilistic method developed by Hanson and Bauer (1989), WITT, will be discussed here, and COBWEB and two other methods will be discussed in chapter four. In total four probabilistic methods are discussed in this thesis.

Hanson and Bauer (1989) suggest that there are four disadvantages to using concept description based on logic statements.

The first problem with using logic statement methods is that membership to a concept, or category, is strictly based upon meeting the given conditions. In other words, a value is either necessary (equality or inequality) or sufficient (within a given range). Hanson and Bauer argue that this creates an "Aristotelian" view of categories, meaning that they are characterised solely by their shared properties and not by their actual likeness, thus possibly failing to reflect their true similarity. Within human categorisation objects may be considered related without specific values being necessary or sufficient; this has been referred to as the concept of polymorphy (Wittgenstein 1953).

Secondly, concepts that are illustrated through logic expressions have firm boundaries and do not contain a gradient or level of membership. However, categories contain members some of which are more tightly fitted to a representation than others. As such some objects are more suited to membership of a particular category than others. However, the less suited object is still a member of the category.

A third problem is that a key feature of a concept is the interrelationship between the features within the contained objects. Logical-statement-based methods, while ignoring the relationships between features, can be overly focused on the

commonality of features belonging to each object within the concept. The cohesion within a concept, that is the interrelation between features, can provide for structuring within a concept.

Finally, the absoluteness of a logical expression used to express a concept does not cater for comparisons between categories, or relative properties. Within human categorisation, categories arise from direct comparison with other objects and categories within context. As such, each category is defined relative to the others by comparison.

All four of these points expressed by (Hanson and Bauer 1989) represent a common thread: logic expressions are not sufficiently flexible to express categories. Logic expressions, by definition, are finite rules, and they provide a model which does not completely express categorisation from the human perspective. It is this ability that probabilistic concept formation aims to provide.

2.4.2.1 WITT

WITT[5] (Hanson and Bauer 1989) is a conceptual clustering system which builds upon the work done by Michalski and Stepp. The method is similar to PAF in that it generates a concept description for disjoint clusters, created utilising the attribute-value pairs of a group of instances. However, the focus of WITT's concept creation and clustering is that of the interrelatedness of features and not just the attribute value pairs on their own. As such the concepts are represented as co-occurrences between features across attribute-value pairs. WITT realises these co-occurrences through the use of contingency tables. A given contingency table for a group of instances represents the attributes within these instances in a matrix. The matrix counts the number of times that different attributes with certain values appear in conjunction with each other. WITT, unlike PAF, is probabilistic in nature and utilises these contingency tables to calculate how likely certain features are to be found together based upon how many times different attribute-value pairs have occurred together, in unison. WITT measures the inter-instance correlation using a metric called cohesion. It acts in a similar way to a distance measure in a data clustering

[5] Hanson and Bauer don't appear to describe what WITT stands for or is named after, although it's possible it is named after the philosopher Wittgenstein who is discussed in their work.

technique, but is used to illustrate conceptual likeness and is far more computationally complex. It is a measure of the distance in terms of relations between features, calculated from the contingency tables. The following section details how this cohesion metric is calculated.

2.4.2.2 COHESION

Hanson and Bauer (1989) defined cohesion, C_c, of a concept c as:

$$C_c = \frac{W_c}{O_c}$$

where W_c is the within-concept cohesion of the concept c, and O_c is the average cohesiveness between c and all other concepts. Categories, or concepts, are not usually formed in isolation from outside input or comparison to existing concepts. Concepts are formed utilising both knowledge within the cluster, and outside of it. A person will form a concept in their mind that an eagle and a hawk are both birds, while at the same time acknowledging that they are not fish. The concept is formed by maximising the closeness within the concept of birds, while also minimising the similarity across categories.

The within-concept cohesion, W_c, is a measure the average variance across the co-occurrence of attribute-pairs within c. It is defined as:

$$W_c = \frac{\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} D_{ij}}{\frac{N(N-1)}{2}}$$

where N is the total number of attributes, and D_{ij} is the co-occurrence distribution within the contingency table for attributes i and j. Dij is further defined as:

$$D_{ij} = \frac{\sum_{u=1}^{i} \sum_{v=1}^{j} f_{uv} \log(f_{uv})}{(\sum_{u=1}^{i} \sum_{v=1}^{j} f_{uv})(\log(\sum_{u=1}^{i} \sum_{v=1}^{j} f_{uv}))}$$

where f_{uv} is the frequency with which value u of attribute i and value v of attribute j co-occur. Each contingency table is a matrix comprised of n and m values, while u and v refer to the number of times that attribute i and j each occurred. As such D_{ij} involves summing all u x and v values over the whole contingency table. Using this equation, if there was perfect co-occurrence within a given table, having the attributes always occurring together, D_{ij} would equal 1.0. If, instead, co-occurrence occurred equally across all combinations then the resultant D_{ij} would be zero. All other combinations fall between these two extremes, serving as a metric of distribution of co-occurrence within the table. W_c can be calculated using the value of D_{ij} for each contingency table within concept c,. The summed values of each Dij within c are divided by a function of N to produce the variance within the c, thus demonstrating its cohesion.

The second component required to calculate the cohesion within a concept, C_c, is O_c which is defined as:

$$O_c = \frac{\sum_{i \neq k=1}^{K} B_{ck}}{K-1}$$

where K is the total number of concepts, and B_{ck} is the measure of relative cohesion between the concepts c and k. B_{ck} is defined as:

$$B_{ck} = \frac{1}{W_c + W_k - 2W_{c \cup k}}$$

where W_c is the measure of within-concept cohesion of c, W_k is the measure of within-concept cohesion of k, and $W_{c \cup k}$ is the measure of cohesion within a union of the two concepts c and k. O_c is thus the sum of cohesion measures between c and all other L concepts, and then divided by L-1 to calculate the average cohesion across of the whole set of concepts.

2.4.2.3 THE WITT ALGORITHM

The WITT algorithm is largely controlled by the cohesion metric and, given a set of N instances, the algorithm could consider all possible clusters. However, as N

increased so would the number of resultant concepts. The algorithm is bound by two thresholds to create "good" concepts, while operating as efficiently as possible. The first phase of the algorithm won't be discussed at length here, but is discussed in (Hanson and Bauer 1989). Basically the initial phase of the algorithm creates some starting clusters, utilising a simple distance metric and a strict threshold (T_1) to verify quality. This phase is largely a data clustering technique, and is referred to as the pre-clustering algorithm. However, once completed, the cohesion measure is then utilised to create a concept hierarchy. Again, this phase utilises thresholds (T_2 and T_3) to ensure quality. The algorithm continues to iterate as long as the cohesion factor between the two most similar clusters is greater than T_3 enabling these clusters to be merged. Once these prospective merges score less than the threshold, the complete clustering has been achieved. This phase is detailed in Table 6:

1)	Compute the cohesion score C for all unclustered instances and existing concepts.
2)	Select the highest instance-cluster pair with score S
3)	If S is greater than T_2 then add the instance to the cluster and go back to Step 1
4)	If not, then use the pre-clustering algorithm again to generate more initial clusters.
	1) For each new cluster c, if $W_{i \cup c}$ is less than T_3 for all k then add c.
	2) If any new clusters are added then go to Step 1.
5)	Else calculate the within-cluster cohesion factor $W_{c \cup j}$ for all clusters and select the pair with the highest score. If the score is higher than T_3 then merge clusters and go to Step 1, else stop.

Table 6. The WITT Algorithm

2.5 SUMMARY

This chapter has given a brief overview of the topic of clustering, focusing on the two main forms: data clustering and conceptual clustering. Within data clustering partitional and hierarchical methods were explained. Conceptual clustering was then explored, highlighting both conceptual and probabilistic methods with in-depth explanations of the PAF and WITT algorithms.

This research examined several conceptual clustering methods, with special attention paid to methods that handle change over time. The next chapter will examine several more conceptual clustering techniques which aim to adapt to change over time.

Chapter 3

Knowledge Acquisition Over Time

"We cannot adopt the way of living that was satisfactory a hundred years ago. The world in which we live has changed, and we must change with it."

Felix Adler (August 13, 1851 – April 24, 1933)
The Religion of Duty,1906

INTRODUCTION

Multiple clustering techniques were examined in the previous chapter and discussing them provided the basic concepts of clustering techniques. This chapter will discuss some further clustering techniques but will then focus on a specific problem within the field; that of learning over time. As the last chapter was a clustering overview, this chapter aims to be an overview of machine learning over time, with special attention paid to clustering methods. Within this area, one of the main topics that will be discussed is that of Concept Drift.

3.1 TIME SERIES ANALYSIS

Data that has been recorded within a time series is one of the more popular types of data that data mining research has examined over the years. This is, in part, a result of the sheer number of domains which are data rich and operate in a sequential or time-based environment. Intrusion detection and meteorology are classic examples of areas that operate in time-based contexts. Other domains include health science, sales and customer data, web and email filtering, engineering and economics (Fuller 1996).

With such a broad range of data sources it is unsurprising that this has been an active area of research. Within the field there is a clear division into two categories: time series clustering and online learning. The area that is of particular interest in this research is online learning. Online learning operates on a non-fixed-size dataset, clustering data as it is generated from a source, in an "always on" fashion. In contrast time series data mining methods operate on a given fixed-size dataset.

3.2 TIME SERIES CLUSTERING

Time series clustering techniques are similar to traditional clustering techniques in that they aim to cluster a dataset of size n into partitions based upon some form of similarity measure. Unlike traditional clustering however, the points are not attribute pairs but values within a time series. The methods that have been developed can be loosely grouped into two categories: whole sequence and subsequence clustering (Lin, Keogh et al. 2003). Both types of technique aim to cluster multiple time series in relation to each other and are briefly explained here.

Whole sequence clustering compares multiple different time sequences with one another. It is similar to conventional clustering methods examining discrete objects, but instead of an object being comprised of a range of attributes it comprises values within a given time series. This method is used with datasets that contain multiple sets of short time series, such as a patient's heartbeat measured over a minute.

Subsequence clustering features only a single time series. However, this one time series is split up into multiple sequential time series that are then clustered. Clustering these sequential time series reveals differences or similarities within different time windows in a single time series. Whereas whole sequence clustering is used to compare shorter time series, subsequence clustering is useful for analysing much larger single time sequences. For example subsequence clustering may be used to analyse a year of weather data at one location, whereas whole clustering might analyse several days across multiple years.

These two basic techniques are at the core of most time series clustering methods. Both, when used in a hierarchical system, produce dendrograms similar to Figure 7. The systems are predominantly implemented using methods that are derivations of those discussed in the Chapter 1. In a survey paper of time series methods Liao (2005) describes the popularity of k-Means variants and agglomerative hierarchical methods within this area. The methods used are obviously modified to suit the application area; but they are strongly grounded within general clustering theory, and as such they will not be discussed in any more depth here.

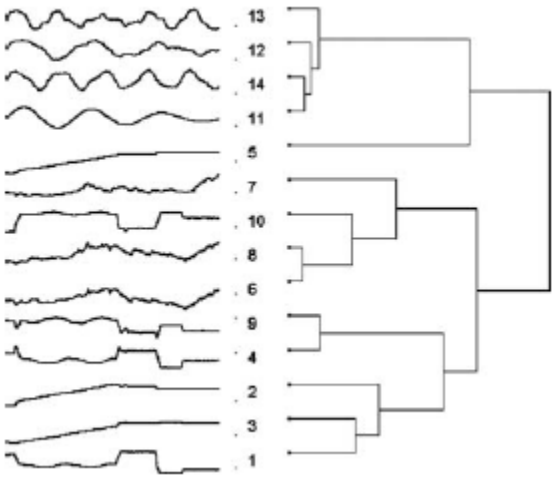


Figure 7. Dendrogram of the "reality check" dataset (Wang, Smith et al. 2006)

3.3 ONLINE LEARNING

Online learning methods operate in a persistent fashion incorporating data on the fly as it is generated or received. They are intended for datasets which are dynamic in nature and can be of unknown size. The core feature of these methods is that the datasets are present within the context of time. The time series methods discussed in the previous section cluster the different series as the objects of interest, comparing one series to another. Online methods however, aim to compare objects within the context of time, based upon a series of attributes associated with each object. They operate in a way similar to the fixed size data mining algorithms discussed in the previous chapter.

Most data mining techniques operate independently of the ordering of the dataset. It is common for a dataset to be randomised and analysed multiple times with different random orderings of the data. This is usually undertaken to prevent the order of the data adversely affecting the resulting data structure. For example when a hierarchical tree is being constructed several of the early braches within the tree are created as a result of the similarity or dissimilarity between the first few presented instances. If these instances are skewed in some way, compared to the remainder of the dataset, then the data structure created may have some bias within it. Such a scenario can usually be avoided through undertaking multiple runs and utilising randomisation of the ordering of the instances between runs. However with online learning techniques the ordering is part of the underlying context in which the instances occur, and is relevant to the data mining being undertaken upon the dataset. For example if the dataset is related to the stock prices within a given market over the course of a week, the order of the instances, and thus the order of the price fluctuations, within the dataset is of great importance. If the dataset is randomised then a great deal of information would be removed, and the knowledge that could be extracted is minimal.

Further, online learning methods aim to utilise this context of time to detect what is termed as “concept drift”. Concept drift is the process of a class definition changing over time. In online learning systems, where knowledge acquisition and classification is occurring continually, being able to detect and respond, or merely adjust, to changes within the data is crucial to the accuracy of the system. For

example it is common for supermarket chains to monitor customer shopping habits based on loyalty card usage. The goods that customers buy vary greatly, but across the population of a town or city there are trends that can be extracted that respond to external variables that have a time relationship such as to pay days, inflation, holidays or natural disasters. All of these processes affect what goods customers are likely to buy, but would largely act as a hidden context within the dataset. An online learning system aims to learn or adjust to such events or processes. Traditional methods which randomise the dataset would miss this knowledge or treat it as noise within the dataset.

Online learning methods aim not only to discover concept drift, but also to adjust with it to ensure that the current model is optimal. Concept drift occurs in two main forms: sudden and gradual (Stanley 2003). Within the customer shopping example a natural disaster would cause a sudden drift within the purchase profiles of a group of people. However, steep increasing inflation over the course of a year may slowly adversely affect the number or types of items the group of customers purchase and this is gradual concept drift. Spam email message filtering is a knowledge domain that has been the subject of a great deal of research over the last decade. The notion of what a spam email message is has not changed during all of the years of research. It still remains simply an email which is unwanted, and usually unsolicited. However over that time the content of spam messages has changed to avoid being filtered by various detection systems. Even beyond the content itself, the origins and the methods of sending the spam have also changed. For this reason online learning methods have been used in spam filtering research (Wang, Guan et al. 2006). A form of a spam message is the "email worm" which propagates through infected users' address books. Outbreaks of this nature have gained attention, within the mainstream media, when virtually overnight, email systems have been overwhelmed with spam messages containing the virus. Unlike most spam filtering which adapts slowly, with time these incidents require rapid reaction by filtering systems.

In the above two scenarios (customer tracking and email filtering) both forms of concept drift is apparent. If a data mining engine being employed in either case, did not respond to the changes that were occurring within the data, the classifications made would be incorrect. As such, methods that are used to detect concept drift need to incorporate knowledge quickly in an effort to react as quickly as possible with the

most accurate response possible. Therefore most online learning algorithms are incremental learners as opposed to batch learners.

3.3.1 INCREMENTAL VS BATCH LEARNING

At a fundamental level, any data mining method is a process of examining a set of data items, and generalising the data into a model. Incremental methods differ from batch methods in the way the model is updated when each instance within the dataset is examined. Whereas batch methods require the entire dataset to be present before the model can be created, model creation by incremental methods occurs as each portion of the data set is examined.

Several of the clustering techniques discussed in Chapter 2 require all of the data series to be present at the start of the clustering process. These are batch learners: k-means clustering, agglomerative hierarchical clustering and PAF clustering are all of this type. By having the entire dataset present the data can produce a model that is an accurate representation of the whole dataset. Further, it can be run multiple times to confirm the model produced. However, when the data is arriving in a stream, and all is not present at the beginning of the learning process, then an alternate method is required. Klinkenberg (2004) discusses the implementation of online algorithms using a batch technique. While it is online as far as data arrives over time and it is incorporated, it does, however, arrive in batches that are examined as they arrive and then integrated into the model. However, this is the exception, and even Klinkenberg suggested methods for converting to an incremental technique. Within incremental approaches, only a single data item is required at a time, with the model being built up with each added instance. The model that is produced is fluid and adapts to the data as it arrives, being incorporated into the model in an optimal way, via the merging and splitting of various concepts as they are formed, or by the reinforcement of the existing concepts.

Within the clustering field incremental approaches are well suited to hierarchical methods, and particularly those that are divisive in design. However, there are incremental versions of many different techniques. Partitional approaches also have incremental models, and indeed Pham and Dimov et al (2004) describes an incremental k-Means method. The original publication of the WITT technique (Hanson and Bauer 1989) included an incremental version which only involved a

few minor adjustments to the basic WITT algorithm. Incremental methods are present within non-online learning contexts, and are often treated in the same way as batch methods, being run multiple times on non-fixed order problems. Incremental methods can suffer bias from poor ordering of data in the same way that a batch method can. The randomisation of the dataset across multiple runs, can avoid this bias, if there is no context to be taken into account.

3.3.1.1 DATA DRIFT

Another consideration when examining batch methods which may operate on a data set that has a time element is the issue of Data Drift (Quionero-Candela, Sugiyama et al. 2009). Data Drift is fundamentally a similar problem to that of concept drift, in that it involves the change of objects or the characteristics that define their resulting concepts, over time, and the process of adjusting to this change. However, the time that is being examined in data drift is the time gap between collecting the training set and the testing sets, whereas in online learning there is no gap between training and testing. The newly observed data is incorporated into the model by the learner and concept drift is then witnessed to take place. However, with data drift, the classes or objects have changed between the training phase and the learning phase. Thus the learner is built on data that does not represent the concepts currently in the observed stream. Data Drift has been an active area of research in recent years and Quionero-Candela, Sugiyama et al (2009) have collected a range of approaches countering six different forms of Data Drift.

3.4 ONLINE LEARNING METHODS

Online learning methods examine data that arrives continually, as it is created or observed in the real world. However, as this can occur over significant periods of time, quite large data sets can be examined. To limit the computation time when analysing a large dataset not all of the data is stored within the algorithm at any one time. A key component of the work that has been done relating to these techniques has been the development of methods to choose which data to retain within the system, and which to remove or 'forget'. Of the various methods that have been produced there are three main paradigms used to counter this problem while at the same time achieving the primary goal of adapting to concept drift. These are instance selection, instance weighting and ensemble learning (Tsymbal 2004). Each of these

three is an intermediary step between the flow of data into the system, and the evaluation criterion within the algorithm that is being used to extract knowledge from the data. Two online learning systems that are examples of these methods will be examined for the remainder of this chapter: STAGGER and FLORA. STAGGER is an example of an ensemble learner using concepts with weights and FLORA uses instance selection (along with some weighting of concept descriptions, similar to instance weighting). An example of instance weighting is the use of Support Vector Machines (SVM) by (Klinkenberg 2004) where instances have a weight based on their age.

3.5 STAGGER

STAGGER[6] is a supervised learner created by Schlimmer and Granger (1986) and was one of the pioneering data mining techniques designed to react to concept drift. STAGGER was not only designed to adjust to concept drift, but to do so within environments where there was noise present. Real world datasets, generally, contain noise. Being able to examine datasets in a fashion that caters for this noise, while also allowing for concept drift to take place and to be accounted for, is of great use in dealing with such datasets.

1.	**Initialisation:** Create starting concepts within the graph based upon single attribute value pairs, each with starting weights.
2.	**Projection:** Matches a concept description to a new instance being examined.
3.	**Evaluation:** Determines how a concept is functioning and adjusts weights accordingly.
4.	**Refinement:** Creates or removes concept descriptions to improve the fitness of the method.

Table 7. The Four main components of STAGGER

STAGGER is a probabilistic data mining approach that utilises a weighted node clustering graph. STAGGER, being a supervised method, requires that each instance that is read into the system has a class identifier attached to it. This knowledge of the class of a given instance is used to adjust the weights of Boolean concepts as the

[6] STAGGER doesn't appear to stand for anything, but is possibly a reference to the stepped nature of the graph that is produced when plotting its performance over time.

algorithm functions. The method of creating, searching and adding of concepts within the graph are founded within Bayesian statistics. Schlimmer and Granger outlined that the STAGGER learning process as being comprised of seven components, four of which are within the algorithm itself and detailed below in Table 7. Each of these will be discussed in full.

3.5.1 INITIALISATION

STAGGER operates using a graph comprised of nodes, each containing a concept description in the form of a Boolean statement. When the graph is created, before any instances are examined, the initialisation step creates a set of nodes to populate the graph. Each of these nodes contains a simple Boolean statement about a single attribute and a value. For example, using an attribute describing size, with the possible values small, medium and large, each of these would be a starting concept at a node. As such, when the algorithm starts, all the nodes are very simplistic, but as it progresses through the other steps more concepts are added with multiple attribute pairs within the concepts, increasing the complexity of the Boolean statements.

When the initial nodes are created each has a pair of weights assigned to it. These weights influence the predictions during the projection stage of the algorithm. Each of the weights for each node initially is set to 1. This then gives each a starting value with no bias to any particular class or node. The two weights function as a predictive measure of an element, one weight is for cases that match and the other for unmatched cases. For example if the attribute is *size* and the value *small* one weight would measure how predictive of the class of the node this has been in the past, based upon the instances it has seen with this attribute. While the other weight stores how predictive it is if the attribute value pair does not match this node in respect to the class. These simple characterisation elements are combined by later stages to expand the graph by the creation of maximally general or maximally Boolean complexes through the use of conjunctive or disjunctive operators.

3.5.2 PROJECTION

Projection is the component within STAGGER that aims to match an instance to a concept description. The comparison between the instance and the concept description is completed on an element by element basis. For each element within the characterisation that appears in the instance the weights are reinforced to build

knowledge within the system. If the instance is of the positive class then the matched weight is incremented; if the class is that of negative then the unmatched class is similarly incremented. The weights represent the expectation that a given element is likely to be a member within the resultant class. These weights are used to compute a collected expectation of whether a given instance is of a class.

The collected expectation is calculated utilising Bayesian formulae derived from the work completed by Duba, Gasching and Hart (1979) in mineral exploration. The formula represented below multiplies together the odds of a positive instance from those previously seen and two measures called: Logical Sufficiency (LS) and Logical Necessity (LN). The resulting value is that of the expectation of instance *x* being of the positive class.

$$Expectation(+ \mid x) = Odds(+) \times \prod_{\forall matched} LS \times \prod_{\forall unmatched} LN$$

LS and LN are both calculated for sets of elements within the instance that are either matched or unmatched by the characterisation. The product of each of these sets of values is then utilised to produce the expectation. LS and LN are briefly described here, but a full derivation is in the appendix to Schlimmer and Granger (1986). LS is estimates the likelihood of an outcome (O) for the feature (F). The value of LS ranges from zero to infinity. Values of 1 indicate independence, while values less than 1 indicate a negative correlation between the feature and the class. All other values above one are evidence for positive correlation.

$$LS = \frac{p(F \mid O)}{p(F \mid \neg O)}$$

LN is very similar to LS except that it examines what effect the absence of a feature has on the likelihood of the positive class appearing. Values greater than 1 mean that there is negative correlation, values less than 1 mean that there is a positive correlation, and values of one are irrelevant to the expected outcome.

$$LN = \frac{p(\neg F \mid O)}{p(\neg F \mid \neg O)}$$

Together, LN and LS, within the cumulative expectation equation, show the odds in favour of the class of the given instance being positive according to this concept description. The value produced is again very similar to the two produced from the individual LS and LN parts, in that if the value is less than 1 then it's unlikely to be the case. Values of 1 illustrate uncertainty in prediction. STAGGER completes this process for all existing concepts in an attempt to match the instance *x* to as many nodes as possible in order to discover the highest expectation, while also incrementing the weights of the other nodes appropriately, to increase knowledge within the system.

3.5.3 EVALUATION

Within STAGGER the concept descriptions that are present are fluid, with new descriptions being created and ineffective ones being removed. The weights upon the characterisations are modified with each instance examined by the system, depending upon predictiveness of the concept description on each instance. The effectiveness of a given characterisation is calculated using the Evaluation method utilising these weights. The evaluation method is based upon research conducted in the field of psychology examining rats(much like the basis of COBWEB discussed in Chapter 4), and the way they react to viewing novel stimulus and unpleasant stimulus, and the variations between the two (Rescorla 1968). Similar to the weights being adjusted within STAGGER, the research by Rescorla tallied the rat's expectation of an unpleasant stimulus in the presence or absence of a novel stimulus. Across multiple scenarios the rat learned to associate the unpleasant stimulus (US) in the presence of a novel stimulus (NS) while the following was true: $p(US \mid NS) > p(US \mid \neg NS)$. This is what is referred to as *contingency* within classical conditioning and has been reinforced in related techniques with other animals (Gamzu and Williams 1971) and humans (Wasserman, Chatlosh et al. 1983). Across these various subject types, contingency learning theory states that if there is positive and negative evidence present for a class then the learning is impaired. It is this impairment that data mining systems aim to overcome.

This research in conditioning was used by Schlimmer and Granger in the context of data mining to present Table 8 as a representation of the weights used in STAGGER. Comparisons between a characterisation and an instance can be seen to have one of

two outcomes: positive or negative. Using the terminology from the work of Bruner, Goodnow and Austin (1956) these two options in turn represent two options to the weight in question: confirming or infirming. A positive match confirms the positive predictiveness (C_P) of a characterisation, while that same instance compared to a negative characterisation confirms (I_P). The reverse is true in the case of a negative instance.

Instance	Characterisation	
	Matched	Unmatched
Positive	Confirming (C_P)	Infirming (I_P)
Negative	Infirming (I_N)	Confirming (C_N)

Table 8. Listing of the characterisation weights

By keeping tallies of the situation configurations illustrated within Table 8 it is possible to calculate the values for LS and LN simply. The Bayesian formula to calculate the two measures based upon these counts is:

$$LS = \frac{C_P(I_N + C_N)}{I_N(C_P + I_P)}$$

$$LN = \frac{I_P(I_N + C_N)}{C_N(C_P + I_P)}$$

Full derivations of these are found in the appendix of Schlimmer and Granger (1986) along with the logical variant discussed earlier under projection. The values that are returned from the evaluation process are used within the refinement method by being compared to threshold values set within the system.

3.5.4 REFINEMENT

The process of refinement aims to increase the effectiveness of the learned concepts as measured by the evaluation method. There are four basic outcomes of the refinement method: pruning, generalisation, specialisation or inversion. The latter three are expressed within Boolean logic as OR, AND and NOT. When STAGGER commences learning, after the initialisation stage, there are only single element characterisations (the middle of Figure 8). It is through the process of refinement that

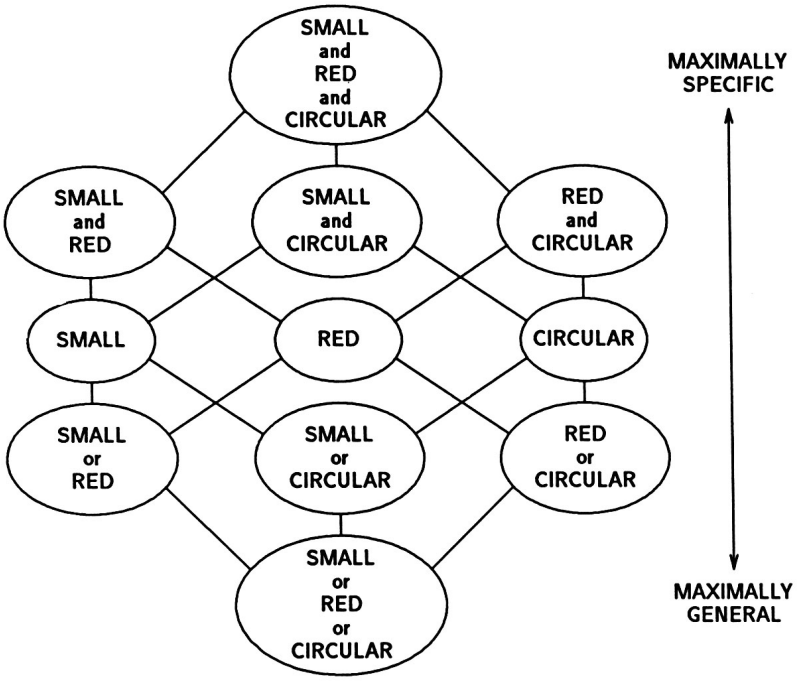


Figure 8. Graph of characterisations produced by STAGGER spanning from maximally specific to the maximally general (Schlimmer and Granger 1986).

the rest of the characterisations are grown, based upon the learning and evaluation that occurs in the previous two stages. Figure 8 shows a section of a graph created by STAGGER.

The selection within the refinement method as to whether a change in the graph needs to be made is based upon whether an error has been made during the projection process. If an error has occurred then a new characterisation is formed. The type of characterisation that is added depends upon the error, and whether or not it was a case of the existing characterisations being too general or too specific (which may depend on the threshold value in the system that then results in a prune). If the error is one of omission, a false negative, then a generalising characterisation is created. If it is an error of commission, a false positive, then a specialisation characterisation is created. In both situations a negation characterisation is also considered. The elements that are chosen to form the new characterisation are also decided based upon the error that has occurred. Related existing characterisations are drawn upon when deciding which elements to include. Scores from the evaluation method are utilised in conjunction with threshold values (to assure quality) to choose the optimal characterisation to be added to the system.

3.5.5 THE STAGGER ALGORITHM

STAGGER has been described above in the various components. Table 9 illustrates the way in which each of these different components function together as a whole from an algorithmic viewpoint. The initial characterisations are formed, and these are then built upon as more instances are observed. The method, in a supervised fashion, reinforces the weights. The evaluation method then corrects the graph when an instance is placed incorrectly. The optimal characterisations from the refinement are then chosen using the threshold values. This process of stepping through the components of the method, adjusting the graph based upon the correctness of the placement, continues as each new instance is observed. The method builds a graph, improving as is observes more instances. But if concept drift occurs, the method adjusts the graph that is produced.

1.	Generate initial characterisations, initialising weights
2.	Compare instance to all of the existent nodes within the graph. If positive classification results then reinforce knowledge through weights and move on to the next instance. If negative classification results then reinforce weights and go to 3.
3.	Evaluate nodes within the graph. If error is an error of commission then propose a conjunction and a negation. If it was an error of omission then propose a disjunction and a negation. Consider a prune.
4.	Choose the optimal solution that is above the defined thresholds.
5.	Move on to the next instance at 2 until the end of the dataset.

Table 9. The STAGGER algorithm.

3.5.6 STAGGER CONCEPTS DATASET

Schlimmer and Granger were pioneers in the area of concept drift, and along with the contribution of the STAGGER learner, they also contributed a dataset. The STAGGER concepts dataset is an elegant dataset that contains two sudden drifts, forming a dataset with three distinct goal concepts to be learnt. It has been used by other authors and will be examined later in this thesis. The dataset of 90 instances contains 3 attributes and a class label as shown in Table 10. The 3 attributes are all nominal in type with 3 possible alternate values; while the class is a simple Boolean.

#	Name	Values
1	Size	Small, Medium or Large
2	Colour	Red, Blue or Green
3	Shape	Square, Circular or Triangular
4	Result	True or False

Table 10. Attribute listing of the STAGGER concepts dataset.

The dataset commences with the positive resulting class defined as *size == small and colour == red* before the sudden drift after 30 instances into *colour == green or shape == circular*. At which point all of the positive instances that have been observed to date are now incorrect. So any future observations are going to be classified incorrectly, and require a method to adapt and relearn across more observations. This is the effect desired for a dataset that is designed to model sudden drift. A second sudden drift after another 30 instances into a third resulting concept defined as *size == medium or large*. Again requiring the method to adjust to the sudden drift over the final 30 instances.

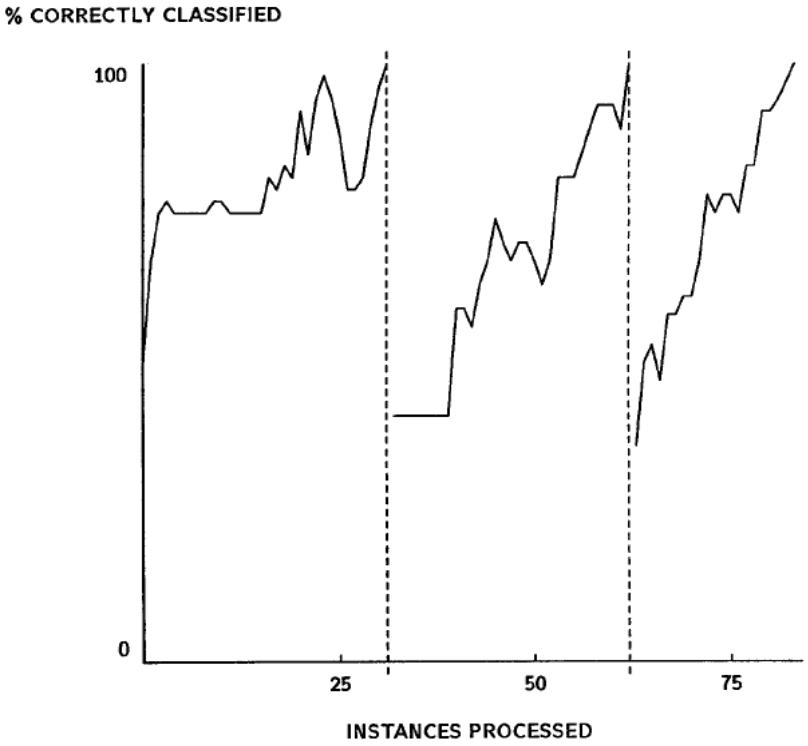


Figure 9. The learning response of STAGGER to concept drift within the STAGGER Concepts dataset (Schlimmer and Granger 1986).

Within Figure 9 the performance of STAGGER upon the concepts dataset is shown. All three concepts were learnt by STAGGER, nearing 100% predictive accuracy by the end of each concept within the dataset. As can be seen at 30 and 60 instances within the dataset, when the sudden drift occurs, the learner then struggled to predict the correct class, falling below 50% predictive accuracy each time. This is expected as the concept that was learnt by STAGGER is now incorrect and the new class descriptions need to be learnt.

3.6 FLORA

STAGGER (Schlimmer and Granger 1986) and other conceptual clustering methods were developed in the 1980's. In the mid-1990s two other conceptual clustering algorithms that attempt to deal with concept drift were introduced. These were FLORA (Kubat and Pavlickova 1991; Widmer and Kubat 1996) and COBBIT (Kilander and Jansson 1993). These two methods both operate in a time sensitive environment using a time window of instances upon which to base the learner's knowledge. FLORA will be described here, but COBBIT will be examined in the Chapter 4.

The FLORA learning method was described by Widmer and Kubat (1996) and is a conceptual online learning system. FLORA is similar to STAGGER in that it also aims to compensate for concept drift, and to improve the model produced by supervising the adaption to the drift that is taking place within the data. FLORA aims to produce a hypothesis that is a set of concepts that describe a problem space. The idea that is central to FLORA and COBBIT's ability to adapt to concept drift is that of having the learner function on a subset of the observed instances, comprising only those that have been recently observed. To do this they use a sliding window of the total observed instances within a dataset to work with, as shown within Figure 10. The motivation for the window based approach is outlined by Midmer and Kubat (1996) stating "only the latest examples are relevant and should be kept in the window, and that only the description items consistent with the examples in the window are retained". As only the most recent observations are retained, any past variations within the dataset are forgotten, therefore, allowing for concept to drift over the course of a dataset.

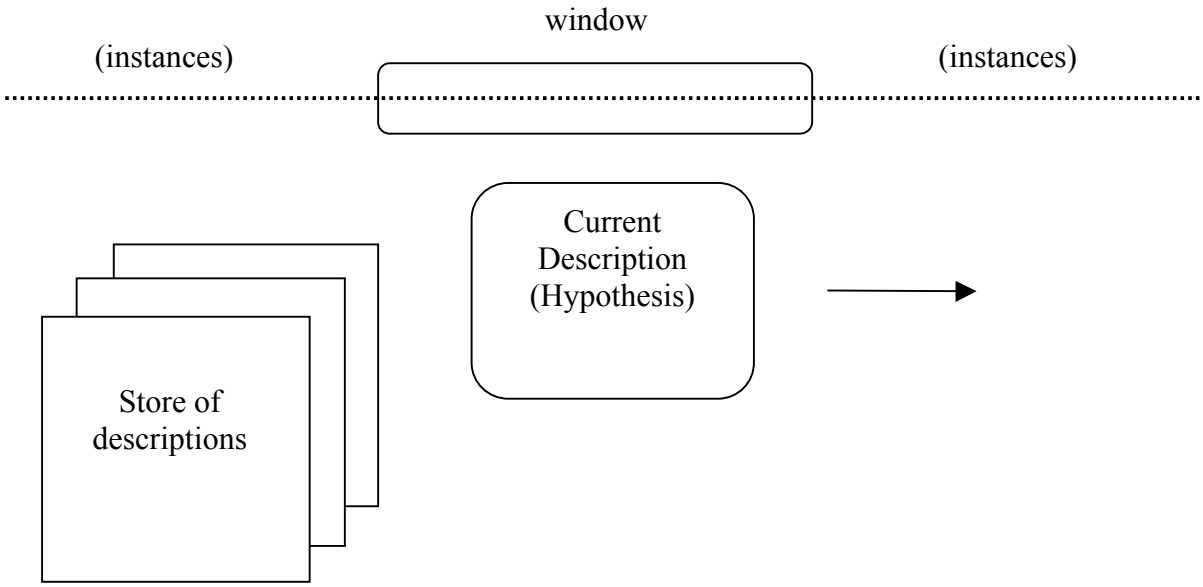


Figure 10. The window examining a stream of instances, utilising only the instances within the window to form the current hypothesis (Widmer and Kubat 1996).

FLORA uses this window in conjunction with three sets of concept descriptions. The first set is termed the ADES (accepted descriptors) and includes those descriptions that match positive examples within the window. In addition to a second set, called the NDES (Negative descriptors) includes those descriptors that summarise the negative examples in the window. The final set is the PDES (potential descriptors) which includes those descriptions that are too general for the present examples in the window. Descriptors migrate between these three sets (or are totally removed) based on the how well they describe the instances currently within the window. This mogration process is partly guided using a measure of fitness that is tied to each of the descriptors within these sets as shown in Table 11. Each descriptor in each set has a counter that tracks how many of the current instancess within the window match this descriptor. Within the PDES set, a counter is recorded for how many instances match it and are positive and how many match it and are negative, while the ADES and NDES sets only maintain a single count each. If a counter is at 0 then no instances match it, and it can be removed. If it has a high number then it matches many instances within the current window.

ADES = {$ADes_1$ | AP_1, $ADes_2$ | AP_2 , ..}	// Accepted Descriptors
PDES = {$PDes_1$ | PP_1 | PN_1, $PDes_2$ | PP_1 | PN_1, ..}	// Potential Descriptors
NDES = {$NDes_1$ | NN_1, $NDes_2$ | NN_2 , ..}	// Negative Descriptors

Table 11. Descriptor sets and the counters associated with each set

A portion of FLORA learning method is outlined within Table 12. The *Learn* function, as shown in Table 12, shows how an observed instance is matched to an existing descriptor within the ADES set. If this is done the AP is incremented. If this match does not occur then generalising an existing descriptor from within ADES, NDES and PDES is tried. If a descriptor can be generalised, while not excluding any existing matched instances in the window, then it is adopted. If a descriptor cannot be found then a new descriptor is formed and added to ADES. Matched descriptors in the PDES set are then incremented, and any conflicting descriptors within NDES are promoted to ADES. Other functions, such as *generalise* and *forget*, are not examined here, but are covered by Widmer and Kubat (1996). However we will focus on their various extensions to this approach and the problems they examined within the concept drift domain for each of the extentions. This examination of *Learn* does however highlight how FLORA examines each instance one at a time, comparing it to the three sets of descriptors in turn, forming knowledge about the domain. The *forget* method operates in a similar fashion to the *Learn* function, examining which descriptors in each set are relevant to the instance being forgotten. However, instead of incrementing the counter on the descriptors, they are decremented. If a descriptor is found to have a count of 0, meaning that none of the instances in the window match the descriptor any longer, it is removed from the set.

Learn(I) // Instance I which has just been observed.
Let ADES, NDES and PDES be the three sets of descriptors.
Let M be a Boolean with the default false meaning the Instance hasn't been matched.
For each J^{th} descriptor D in the set ADES
    If I matches D
        M becomes true
        Increment AP_J
If M is false
    Let G be a the result of attempting to generalising a descriptor in ADES, NDES or PDES
If M is false and G is NULL
    Create a new descriptor for I in ADES with a AP of 1
For each K^{th} descriptor D in the set PDES
    If I matches D
        Increment PP_K
For each L^{th} descriptor D in the set NDES
    If I matches D
        Delete D from NDES
        Add D to ADES

Table 12. The Learn function within FLORA

In addition to the original version of FLORA (Kubat and Pavlickova 1991) three additional versions were also produced to deal with three relevant problems within the domain: dynamic window size, recurring contexts and noise. These three methods were simply termed FLORA2, FLORA3 and FLORA4 (Widmer and Kubat 1996) and each will be briefly described here. After this the performance of these methods on the STAGGER concepts dataset is shown in Figure 12.

3.6.1 FLORA 2

An obvious difficulty with a concept drift method that operates using a window method is knowing how big the window should be. If there is sudden drift, such as that within the STAGGER concept dataset, then a small window enables this sudden drift to be adapted to quickly, as the old data is constantly being removed. However, if the window is small, then perhaps the concepts to be learned are not fully described by the learner. Widmer and Kubat (1996) argue that a solution to this is to have a window that is dynamic in size, and as such handles both portions of this problem, being big enough to describe the dataset when the concepts are stable, while also being able to shrink if the dataset has experienced a significant degree of drift and causing some of the older observed instances in the window to reduce the predictive accuracy.

The approach to creating a dynamically sized window in FLORA2 uses three threshold values: *lc*, *hc* and *p*. The first two, *lc* and *hc*, are for measuring coverage of the ADES over the domain. The third, p, is a threshold of acceptable predictiveness. These two coverage thresholds are based on a ratio of the total number of descriptions to the number of instances within the window. This measures the complexity of the descriptions that are contained within the hypothesis. If the ratio is above *hc* and the current accuracy is above *p* then the set in ADES is stable, but possibly over-complex. So the window is then shrunk slightly. Alternately if the ratio is below *lc*, or *p*, then drift is expected and the window is shrunk by 20% in an effort to accommodate the drift. If the ratio is below *hc* then the system slowly grows the window. With these various changes to the state of the window, it is possible to shrink the window when drift is apparent, or grow it again after the drift has occurred to fully describe a domain. As an example of threshold values used, Widmer and Kubat (1996) stated that on the STAGGER concepts dataset values of p=70%,

lc=1.4 and *hc*=4.0 were used. In the context of the sudden drift within the dataset, it is easy to see that a *p* below 70% is easily attained when the sudden drift occurs (Figure 12).

3.6.2 FLORA3

When examining data that changes over time it is logical to conclude that a concept definition, or indeed set of concept definitions, may reappear later in the dataset. Widmer and Kubat (Widmer and Kubat 1996), realising this, chose to add a memory of past stable hypotheses into FLORA3, the idea being that if the concept drifted back to a concept that had already been witnessed then when it reappeared, instead of the letting the descriptions slowly revert to the earlier version, it would be more efficient to adopt the earlier version as it had been previously learned.

This process within FLORA3 is comprised of 3 main steps. Firstly, once a stable concept is learnt then it is stored away by the learner. Then if drift is suggested, through FLORA2's detection methods, then past stored hypotheses are trialled in comparison to the current hypotheses. This is completed by comparing the stored hypotheses to the instances within the current window and scoring each of the hypotheses based on the ratio of correct to incorrect descriptions of the instances in the current window. If a past hypothesis is shown to outscore the existing hypothesis then it is adopted as the new current hypothesis.

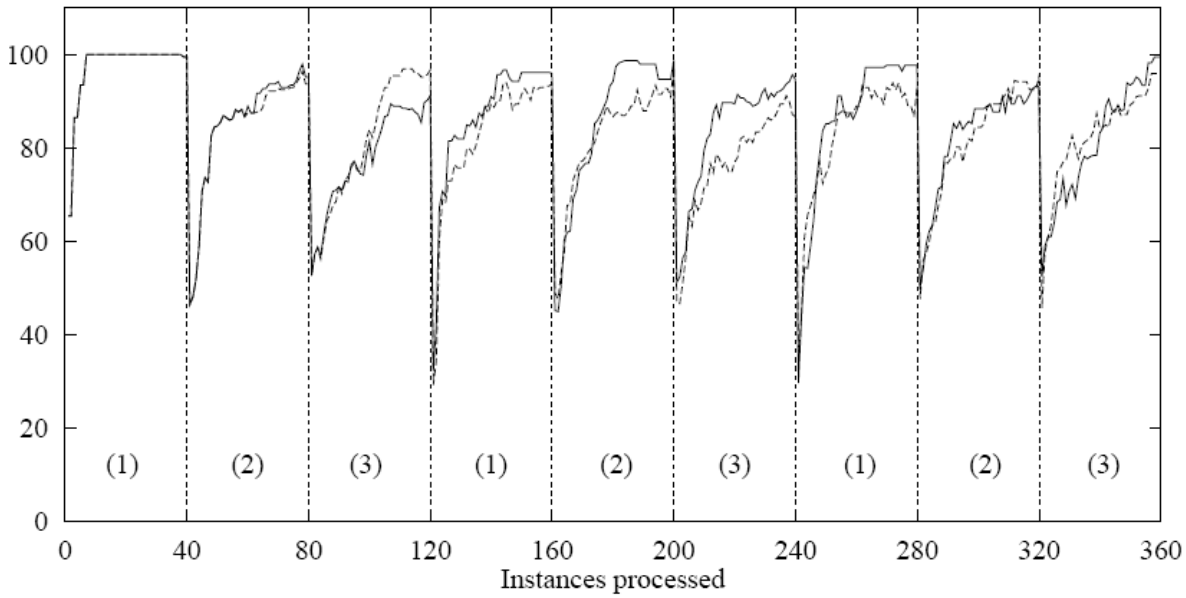


Figure 11. FLORA3 (solid) compared to FLORA2 (dashed) at relearning the same three STAGGER Concepts three times (Widmer and Kubat 1996).

The hypothesis is then generalised to the instances that are presently within the window, which may still include some members of the old concept learned prior to the drift in STAGGER. This is why in Figure 11 the predictive accuracy is still rather low directly after a sudden drift. However, as can be seen in Figure 11, FLORA3 does perform better than FLORA2 and FLORA3 on first encountering the new concepts.

3.6.3 FLORA 4

Noise is change that is not true change but variance within the data and it can appear similar to concept drift in a dataset. Noise can come from various sources, whether it is natural variation, or inconsistencies in the way that the data within a domain is measured over time. The key point about noise in relation to drift, is that it is a false change that should not be adapted to. If a concept drift method mistakes noise for drift then it may take actions to adapt to a drift that is not actually present, reducing the effectiveness of the learner. Therefore, if a method is tolerant to noise, it is better equipped to handle actual drift occurring within noisy environments.

Within FLORA each descriptor had a counter, stored along with it, that measures its utility within the current window. In FLORA4 this is replaced with a confidence measure detailing how well the given measure predicts in the current window. If a confidence measure falls below a set threshold level (set as 80% within the STAGGER Concepts trial in Figure 12) then it is removed. The main change that this introduces, is to allow ADES and NDES to have descriptors that in some cases cover positive and negative values. This results in a learner that is more tolerant of specific incorrect descriptors, when they are of greater value across the remainder of the instances in the window. PDES now stores those descriptors which have too low a confidence level to be contained within ADES and NDES.

Figure 12 illustrates how FLORA2, FLORA3 and FLORA4 perform upon the STAGGER Concepts dataset. The dataset is slightly larger than the version used to illustrate STAGGER's (Schlimmer and Granger 1986) performance above in Figure 9. This version contains 120 instances in total, with each concept lasting 40 instances. FLORA4 finishes each concept with the highest accuracy, but it also takes the longest to adjust to each concept after the drift has taken place. Both FLORA2 and FLORA3 perform very similarly until FLORA3 learns a few past stable

hypotheses which it can then use to replace the existing current hypothesis. At this point it tends to branch away from FLORA2 as the slightly different learning method produces enough variation in behaviour between the two to be notable.

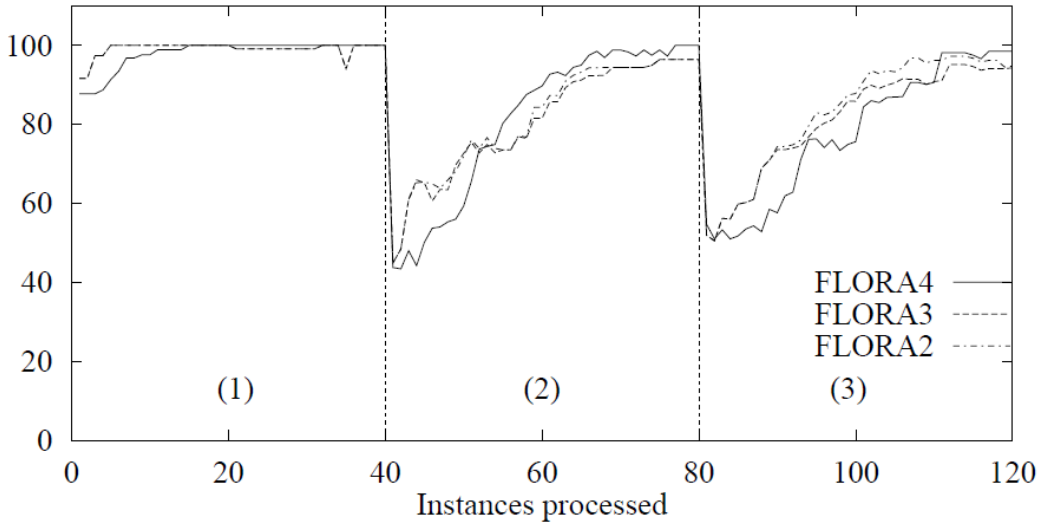


Figure 12. The learning response to concept drift of FLORA upon the STAGGER Concepts dataset (Widmer and Kubat 1996).

3.7 SUMMARY

This chapter has examined the problem of data that changes over time. The main focus was on concept drift, where the definition of the target classes within a dataset change over time. Several related fields to this were briefly touched on before two concept drift methods were examined. Both of the learners that were examined, STAGGER and FLORA, were supervised conceptual machine learning methods. In addition to these two learners, the STAGGER Concepts dataset was also introduced, with some results on both of these methods shown. Chapter 6 will describe how the DynamicWEB method presented within this thesis performs when testes on this dataset.

Concept drift is of primary importance to this research because it is this phenomenon that the research is attempting to account for in a machine learning context. However, while both methods discussed in this chapter are foundational to the field, they are both supervised approaches, making them unsuitable for meeting the aims outlined in the introduction. Furthermore, they are not designed with multiple observations of the target objects in mind. However, they have provided valuable insight into how to handle concept drift.

Chapter 4

COBWEB

"And it ought to be remembered that there is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things. ... [This stems partly from] fear of the opponents, who have the laws on their side, and partly from the incredulity of men, who do not readily believe in new things until they have had a long experience of them."

Niccolò Machiavelli (3 May 1469 – 21 June 1527)
The Prince, 1532

INTRODUCTION

The preceding chapters have described various clustering methods progressing from simple k-Means through to conceptual clustering and finally concept drift algorithms. This thesis presents a new unsupervised learner that is able to operate in the presence of concept drift and is built upon an existing conceptual clustering algorithm. This chapter describes the existing algorithm known as COBWEB. It is this method that has been modified to produce the technique that is presented within this thesis.

4.1 INTRODUCTION TO COBWEB

The COBWEB algorithm was published by Fisher in (1987). It was completed during an active period in conceptual clustering research during which many different techniques were presented. Along with the methods already discussed in this thesis, the other techniques drawn upon by Fisher were: CYRUS (Kolodner 1983), EPAM (Feigenbaum and Simon 1984) and UNIMEM (Lebowitz 1986). UNIMEM will be briefly examined here as it is the method most similar to COBWEB. After this, COBWEB will be examined, followed by a review of work in which it has been extended by other authors to further its capabilities.

4.2 UNIMEM

UNIMEM, created by Lebowitz (1986; 1987), (Lebowitz 1986; Lebowitz 1987) is a hierarchical conceptual clustering method which is very similar to the methods presented in Chapter 2. Lebowitz named UNIMEM for the phrase: UNIversal MEMory model, with the aim of creating a system that would be applicable across a wide range of domains. It was trialled on nine different datasets, and this showed that this goal was realised. UNIMEM is built upon the Generalisation-Based Memory framework, which also relates to its naming; however, it is the concept formation portion which is of relevance for the work described in this thesis. Lebowitz outlined four key features that characterise UNIMEM:

- Learning is achieved by observation in an unsupervised manner;
- Knowledge is learnt incrementally allowing for the model to be accessed before the entire dataset has been examined;

- It handles large numbers of examples;
- The generalisations are pragmatic and do not need to exactly cover all instances within the given group.

UNIMEM aims to learn by observing instances and grouping similar instances together, building up a model representing the natural groups present in the dataset. This is not a supervised process, but is completed by the algorithm, basing its decisions purely upon the characteristics of the given instances. As it is implemented within a hierarchical structure, the model is able to represent the groups in such a way that the most general are towards the root, with increasing specificity towards the leaves.

UNIMEM is an incremental method which takes a single instance at a time and searches for the best place, or places, to fit it in the existing concept hierarchy. Within UNIMEM (and also CYRUS, but not COBWEB) instances are able to be stored in more than one location, allowing for overlap. Once a location has been found for the instance, the concept hierarchy is updated and the next item is examined.

The clusters formed within UNIMEM are based upon the presence of observed features within a group of instances. For each of these groups a class description is created that depicts the instances that are resident within the node, based upon the attribute-value pairs (referred to as features) which are common to those instances. These descriptions are located within all nodes in the structure, both at leaf and intermediary nodes. Each attribute-value pair also carries a numerical value conveying the predictiveness of the feature. This predictiveness score is adjusted upon the observation of features within instances, and is used in conjunction with several knowledge threshold parameters set when the tree is created. Table 13 details the main portion of the UNIMEM algorithm. Table 14 presents two methods, called *Generalise* and *Evaluate*, which the algorithm uses.

When a new instance, *I,* with its set of features, *F,* is presented to UNIMEM it is examined at the root node. Subsequent examinations then occur of the children of the root recursively until all ideal locations are found. Upon examination, the features which match the current node, *N*, are placed into the set *H*; those features that do not

```
UNIMEM(N, I, F) // Current Node, Current Instance, Unassigned Feature Set of I
Let S be an empty list of nodes
Let K be the features in F that don't match the features at N
Let H be the features in F that match with features at N
If N is not the root node
    Evaluate(N, H, K), if it returns true then return the empty list S
For each child, C, of node N
    If C is indexed by a feature in K
        S becomes the union of S and UNIMEM(C, I, K)
If S is still an empty list
    For each instance, J, of Node N
        S becomes the union of S and Generalise(N, J, I, F)
If S is still an empty list store I in node N with features K
    For each feature, J, in H set of features
        Increment the predictiveness score R of J by 1.
        If R crosses threshold
            Remove J as an index to N
Return N
```

Table 13. The UNIMEM Algorithm

match are placed in *K*. If the current node is not the root then the Evaluate function (Table 14a) is called using the current node and the sets of matched and unmatched features.

The Evaluate function adjusts the predictiveness score of non-permanent features upon the current node. These are features that have not yet made it to the class description of the node. For each feature, *J*, which is found within *H* to be shared with the feature set at *N*, the predictiveness score *R* is incremented. Likewise, if feature *J* is not found within *K*, the predictiveness score is decremented. After both of these are examined, upper and lower thresholds are checked for the retention or promotion of the feature. If the feature is removed, another threshold is examined in order to determine the minimum number of features at the node. If there are too few then the whole node is removed, and the function returns TRUE back to the UNIMEM algorithm. In all other cases FALSE is returned.

If the result from the Evaluate function was TRUE then UNIMEM returns an empty list as the current node has been removed, and the parent is the final location for the

```
Evaluate(N, H, K) // Current Node, Matched Features, Unmatched Features
For each non-permanent feature, J, in H
    Increment predictiveness score R for J on N
    If R is crosses threshold make J a permanent feature of N
For each non-permanent feature, J, in K
    Decrement the predictiveness score R for J in N
    If R is low enough then remove J from N
        If N has too few Features remove N from the list of children in parent node
            Remove all indices to N
            Return TRUE
Return FALSE
```

```
Generalise(N, J, I, F) // Current Node, Instance at Node, Current Instance,
Unassigned Features
Let H be the features that match in the instances J and I
If H contains enough features
    Create a new child C of node N
    Index and describe C by the features H
    For each feature K serving as an index to C
        Increment the predictiveness score R of K by 1
        If R crosses threshold
            Remove K as an index to C
     Remove J as an instance of N
     Store J and I as instances of C
```

Table 14. The Evaluate (a) and Generalise (b) functions used by the UNIMEM algorithm.

instance. When the value returned is FALSE, UNIMEM then proceeds by calling itself recursively upon child nodes which match with features within *K*. This is how the UNIMEM tree is searched to locate positions where the instance, *I*, can be placed. Locations that are found are then stored within the list *S*. If, after the examination of the children, the list S is empty then the Generalise (Table 14b) function is called upon for each instance *J* at *N*. The Generalise function compares the current instance to all the instances stored at the present node. If the current

instance and the J^{th} instance in the node share enough similarities then a new child node is created, growing the tree. The predictiveness scores for the features within the new node are incremented.

If the set *S* still remains empty then the current instance is stored within the current node. Each feature at the node which matches the current instance has its predictiveness scores incremented. If any of the predictiveness scores cross the knowledge threshold then the feature is removed from the class description.

In this way the hierarchy is grown, instance by instance, integrating new knowledge as it is observed. UNIMEM can examine datasets comprised of both numeric and nominal valued attributes. The match or non-match based method described above obviously can function well with nominal data, but when it comes to numeric data, where two values are almost identical in comparison to the remainder of the dataset, this poses a problem for a strict Boolean feature decision. To overcome this, UNIMEM allows the quality measure to produce values between 0 and 1 (where 0 is a total mismatch and 1 is a perfect match). Then, when comparing a set of features, UNIMEM uses another threshold parameter to determine the maximum distance allowed between two features in order for them to be considered the same. In addition to this, Lebowitz (1986) described a method used to calculate effective *gaps* within numerical data to serve as partitions. This improves UNIMEM's ability to consider two numerical values to be the same feature value.

UNIMEM creates useful concept hierarchies through the observation of shared features between instances. The process of knowledge acquisition is largely controlled by the use of multiple parameters that govern predictiveness thresholds and the minimum and maximum number of shared features required in order to create or delete a node within the hierarchy.

4.3 COBWEB: THE METHOD

COBWEB (Fisher 1987), similar to UNIMEM, aims to discover natural groups within datasets by establishing the presence of relationships between the different instances. It also learns incrementally as an unsupervised learner. COBWEB's structure is hierarchical, similarly to that of UNIMEM. The COBWEB tree is grown instance by instance, often re-structuring sections of the tree as more data is added.

Each instance is only stored at a single location within the tree, and unlike UNIMEM, there are no parameters with which to tune the knowledge acquisition.

The key component of the COBWEB algorithm is the measure of similarity which is used to establish relationships between instances. Both the addition function and the mechanism used to search for instances within the tree, employ a heuristic measure called the *category utility*. Category utility was described by Gluck and Corter (1985) (and also in Corter and Gluck (1992)) as a method for the creation of basic categories in a similar manner to those created by the human brain. Basic level categorisation, as drawn upon by Gluck and Corter, was described by Mervis and Rosch (1981). A basic level category is defined as one which is preferred to a more generalised or specific category during object recognition. For example, "a dog" in preference to other categories' labels such as "animal" (more general) or "Labrador" (more specific). Gluck and Corter, using the category utility, were able to produce the same categories as those produced in human psychological testing. Fisher's goal in using the category utility within COBWEB was to produce categories which are not only predictive, but also are easily human readable.

Category utility is a measure of similarity between instances, and therefore acts as a measure of the quality of a given cluster. The category utility is represented by the result of a calculation which takes account of each attribute in an instance, comparing it to the attribute values of the other instances within a category, and returning the utility as a measure of how much information they all have in common. This research does not modify the category utility; therefore the one presented here as used in COBWEB is also the same as that used within the new algorithm known as DynamicWEB. The calculation of the category utility will now be explained further, before moving on to a discussion of the COBWEB algorithm.

4.3.1 CATEGORY UTILITY

The category utility measure can be most easily understood as a type of distance measure. The output from the calculation determines whether or not an instance is enough 'a-like' the other instances within (MacQueen 1967) a cluster to be made a member of that category itself. It is much more complex than the simple k-Means measure described in Chapter 2, but it serves a similar function. For a closer comparison within conceptual clustering the most similar measure that has been

examined in this thesis so far is the cohesion measure within WITT (Hanson and Bauer 1989) (described in Chapter 2).

The category utility calculation takes account of each attribute in an instance, comparing it to the attributes of the instances already within a given cluster, and returning the utility as a measure of how much information they all have in common. The function aims to maximise the similarity between objects of the same class, while also maximising the dissimilarity between it and those in instances in other classes. Within the same class, the probability that class members share a particular attribute value is $P(A_i = V_{ij} \mid C_k)$ where $A_i = V_{ij}$ is the attribute value pair and C_k is the class. The greater this value, the more frequently class members share the given attribute-value pair. Similarly, the greater the value of $P(C_k \mid A_i = V_{ij})$ the less common this value is in other classes and the more predictive it is of this class membership. These two probability measures for individual attributes can then be combined to produce a measure of overall cluster quality trading off the two values:

$$\sum_k \sum_i \sum_j P\left(A_i = V_{ij}\right) P\left(C_k \mid A_i = V_{ij}\right) P\left(A_i = V_{ij} \mid C_k\right) \quad (1)$$

where *k* varies over classes, *i* over attributes and *j* over values. The products of the two probabilities are summed across all classes, attributes and values weighted by a third probability measure $P(A_i = V_{ij})$. This weighs the importance of individual values, resulting in greater worth being given to those that appear many times than to those that are rare. This can then be transformed using Bayes' rule to be shown as:

$$\sum_k P(C_k) \sum_i \sum_j P\left(A_i = V_{ij} \mid C_k\right)^2 \quad (2)$$

The expected number of attribute values that will be guessed correctly (knowing the class) is $\sum_i \sum_j P(A_i = V_{ij} \mid C_k)^2$ assuming that the guessing strategy is probability modelling (Gluck and Corter 1985). In probability modelling each attribute value is guessed with a probability equal to the probability of it occurring.

Gluck and Corter (1985) define the category utility as the increase in the expected number of attribute values that can be correctly guessed given a set of K categories compared to the expected correct number of guesses achieved with no knowledge. The first component is equation 2, and the no-knowledge component is $\sum_i \sum_j P(A_i = V_{ij})^2$. The difference between the two measures is then divided by the number of categories, K. This process calculates the average increase and allows comparisons to be made between different sized categories.

$$\frac{\sum_{k=1}^{K} P(C_k) \sum_{i}^{0} \sum_{j}^{0} P(A_i = V_{ij} \mid C_k)^2 - \sum_{i}^{0} \sum_{j}^{0} P(A_i = V_{ij})^2}{K} \quad (3)$$

COBWEB(N, I) // Current Node, Current Instance
If N is a leaf
    return leaf as final location
Else
    Calculate the category utility X for creating a new leaf Q
    For each Child, C, of node N
        Calculate category utility for placing I in C
    Let P be the node C with the highest category utility W
    Let R be the second highest category utility
    Calculate the category utility Y for merging P and R
    Calculate the category utility Z for splitting P
    If W is the highest score
        Place I in P with **COBWEB(P, I)**
    Else If X is the highest score
        COBWEB(Q, I)
    Else If Y is the highest score
        Let M be the result of Merge(P, R, N)
        COBWEB(M, I)
    Else If Z is the highest score
        Elevate the children on P with Split(P, N)
        COBWEB(N, I)

Table 15. The COBWEB Algorithm

The category utility measure shown here is the version used by Fisher (1987), which is a slight simplification of that described by Gluck and Corter. Further, this version was designed with nominal values in mind; a numerical version will be discussed in Section 4.4. The COBWEB algorithm depends upon the category utility, and, as this has now been described, the algorithm itself will be discussed.

4.3.2 THE COBWEB ALGORITHM

COBWEB is an incremental conceptual clustering method built upon the category utility described in Section 4.3.1. Knowledge is grown one instance at a time as concepts are formed based upon the discovery of natural groups within the dataset. In COBWEB the concept descriptions are not a conjunctive statement such as in CLUSTER or UNIMEM. Instead concepts are represented by a probabilistic representation. This differs from the attribute-value counting that occurred within UNIMEM.

When a new instance, *I*, is added to COBWEB, its resulting location is found by searching through the current tree (Table 15), and trialling four different options at each potential location. At each node the best option is chosen based upon the highest category utility, and so the search through the tree is greedy, rapidly ignoring the branches which did not 'win' comparisons.

The first of these possible solutions is to simply create a new leaf category *Q*. This is always the case with the first item added to the tree, but in subsequent instances it evaluates whether or not another class should be created as a child of the node. Once the leaf has been created, its category utility, *X*, is calculated for comparison with the other possible solutions.

The second option tested, is the incorporation of the instance into an existing child class of the current node. Initially this is at the root, but COBWEB then recursively traverses down the tree to find the ideal category. During this process, at each node the two most suitable children (*P* and *R*) for the current instance are tracked. The most suitable child, *P*, has the score *W*.

The third and fourth options are inverse operations of each other: merge and split. It is these two operations that optimise the tree for greater knowledge retention. The

Outlook	Overcast (0.60), Sunny (0.20), Rainy (0.20)
Windy	True (0.60), False (0.40)

Table 16. Probabilistic Concept Descriptions for the two nominal attributes within the Weather Dataset

merge function considers merging the two children of the current node with the highest category utility as calculated during the incorporation stage. Conversely, the split function considers removing the best child and elevating its children to being children of the current node. After both of these computationally complex options have been completed, the category utility of the possible resulting placements of *I* is calculated.

Once each trial is completed the resulting category utilities are compared (*W*, *X*, *Y* and *Z*) and the most favourable option is then adopted. As each instance is observed by COBWEB this process occurs, building the tree with each item placed in the categories that are most similar to it.

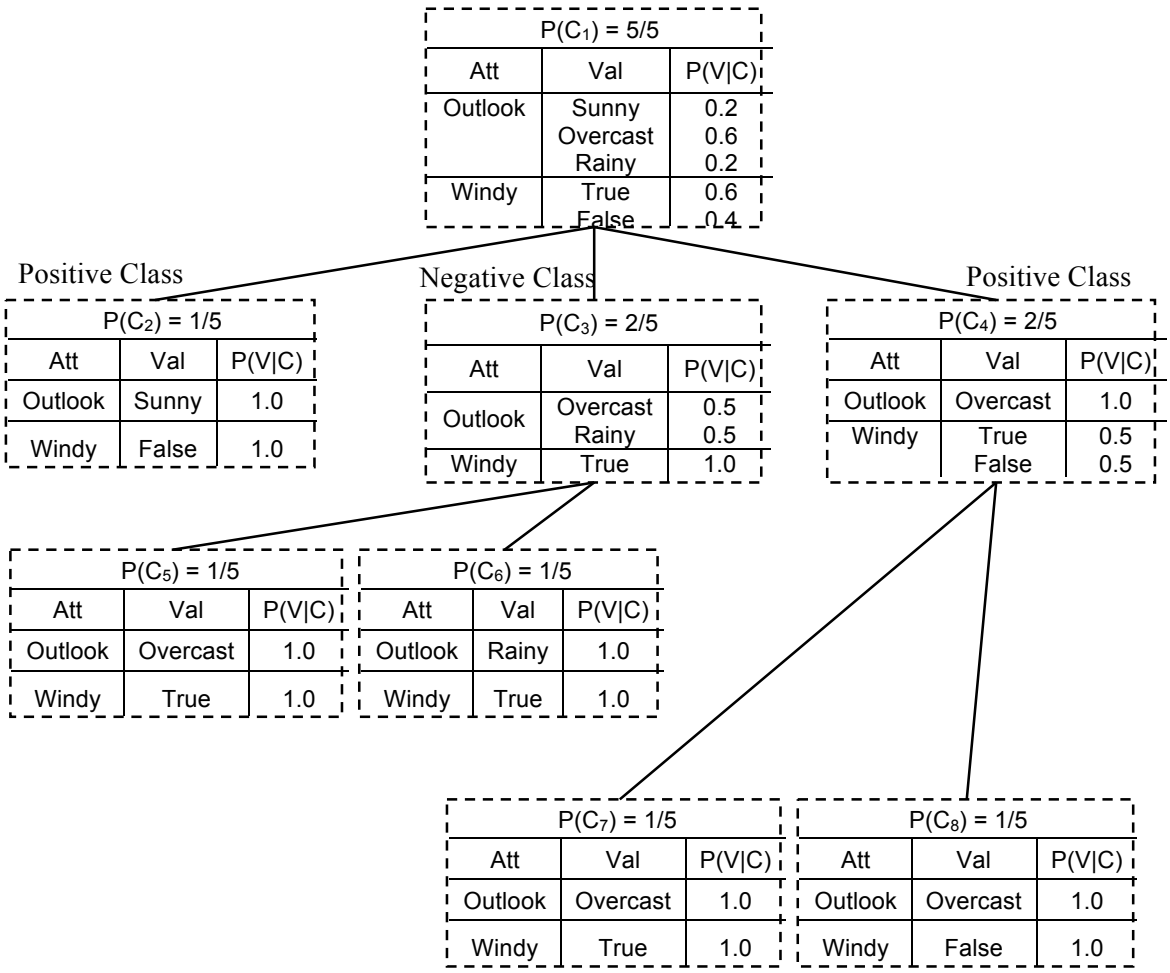


Figure 13. A COBWEB hierarchy that has been created from 5 instances of the Dynamic Weather dataset (Appendix A).

Figure 13 details a COBWEB tree that has been created from the first five instances of the Dynamic Weather dataset (Appendix A) which is discussed in Chapter 6. This figure illustrates probabilities generated by each of the nodes for the two nominal attributes within the dataset: Outlook and Windy. Outlook has three possible values: Sunny, Overcast and Rainy; while Windy is a Boolean attribute. The probability listed is calculated from two integer scores stored at each node. One is the count of the instances which are present at that node, while the other is a count of occurrences of an attribute value at that node. If all instances at that node possess a certain attribute value pair then the probability is 1.0; if half possess the value the probability is 0.5. It is these values that are input into the category utility calculations.

4.4 CLASSIT: EXTENDING COBWEB

A short time after COBWEB was described another method was published called CLASSIT (Gennari, Langley et al. 1989). CLASSIT has the ability to work with numerical values. CLASSIT is primarily the work of Gennari, but it is worth noting the co-authorship of the method by Fisher. CLASSIT is an extension to COBWEB with the core of the technique still remaining as it was in COBWEB. The modification required changes to be made to the evaluation method used to calculate the category utility, a minor change to the concept representation and the introduction of two system parameters.

The ability to use numerical attributes is an obvious extension to be made to COBWEB, as real world problems are commonly described with such values. Gennari, when deciding how to get CLASSIT to function effectively on numerical data, did consider the way in which UNIMEM utilises the creation of artificial gaps within the continuous data. However, he chose to retain the actual values within the instances, but to introduce the use of the average and standard deviation into the concept description at the node level, where numerical values were already involved.

Another addition to CLASSIT is that of a system parameter called *cutoff*. This is used as a threshold to judge whether further progression down the tree is required, or whether the current location is specific enough for accurate categorisation and classification. The cutoff is a knowledge threshold which has some similarity to other thresholds in methods such as UNIMEM. Within CLASSIT the cutoff value is

examined when making the final decision as to which option to take. If the option with the highest category utility does not pass the value of the cutoff, then the option is not undertaken, and the current node is the best location for the instance. The cutoff value in effect can control the height of the tree, and the tightness of the clusters that are represented within it. Such an addition is argued for by Quinlan (1986), in order to improve the performance of decision trees for classification by avoiding over fitting the structure to the data observed. This occurs as a result of the creation of an exhaustive structure tightly based upon all values in the dataset whereas a tree with decreased height may more adequately describe the domain in general.

To enable CLASSIT to be able to examine numeric valued attributes, the main change that needed to occur was to the evaluation function. The category utility that is used for numeric attributes is different from that which is used for nominal attributes. The two terms used in 1.3 (repeated below) need to be generalised for real valued attributes:

$$\sum_{j}^{values} P(A_i = V_{ij} \mid C_k)^2 \text{ and } \sum_{j}^{values} P(A_i = V_{ij})^2 \quad (4)$$

These two terms both involve a sum of squares of the probabilities of all *j* values for the attribute *i*. The first term, as within 1.3, has knowledge of the class C_k while the second expression does not. This means that the difference between the two represents the increase in the number of correct guesses of the values present within the category *k*. However, as this is now within a continuous domain, the summation needs to be changed to integration. In doing so, an assumption needs to be made about the possible distribution of the values present. Gennari argues that the best assumption, considering the range of possible datasets that could be examined, is that it is a normal distribution. For the summation of the first item the distribution is that of the category, while for the second item it is of the parent, and in both cases the integral is:

$$\sum_{j}^{values} P(A_i = V_{ij})^2 \Leftrightarrow \int \frac{1}{\sigma^2 2\pi} \exp\left(\frac{x-\mu}{\sigma} \right)^2 dx = \frac{1}{\sigma} \frac{1}{2\sqrt{\pi}} \quad (5)$$

where μ is the mean and σ is the standard deviation. As this is only used for comparisons, the $\frac{1}{2\sqrt{\pi}}$ term can be dropped, allowing it to be revised down into the new category utility function:

$$\frac{\sum_{k}^{K} P(C_k) \sum_{i}^{I} \frac{1}{\sigma_{ik}} - \sum_{i}^{I} \frac{1}{\sigma_{ip}}}{K} \qquad (6)$$

where I is the total number of attributes, K is the total number of classes in the partition, and σ_{ik} is the standard deviation for an attribute i in class k, and σ_{ip} for the attribute i in the parent node p. This resulting function is similar to the category utility for nominal attributes within COBWEB.

A problem with this function is that when there is only a single instance within a class the standard deviation is zero. The result of 1/σ then adversely affects the tree. To combat this, Gennari introduced a constant called *acuity* to give some value of difference between these largely empty nodes. The value is a minimum value for σ. This introduction of a second system parameter does have the negative effect of artificially broadening the tree structure, especially in the early stages, at the leaves. The higher levels within the structure itself are not as affected by this, as many of the nodes contain more than a single instance in their branch.

Figure 14 extends upon the categories which were illustrated within Figure 13, now showing the σ and *mean* for the concept descriptions. As can be seen by the value of σ at the leaves, the acuity results in the lowest σ value being 1. The σ and *mean* shown above are calculated using an incremental method using a sum of squares (Gennari, Langley et al. 1989).

The CLASSIT extension has become almost synonymous with COBWEB itself in some of the literature. This is probably due to it being of such use within datasets that contain numerical values, but also due to it being, in part, the work of the original authors. It is in this way that they are connected and seen to be the same method.

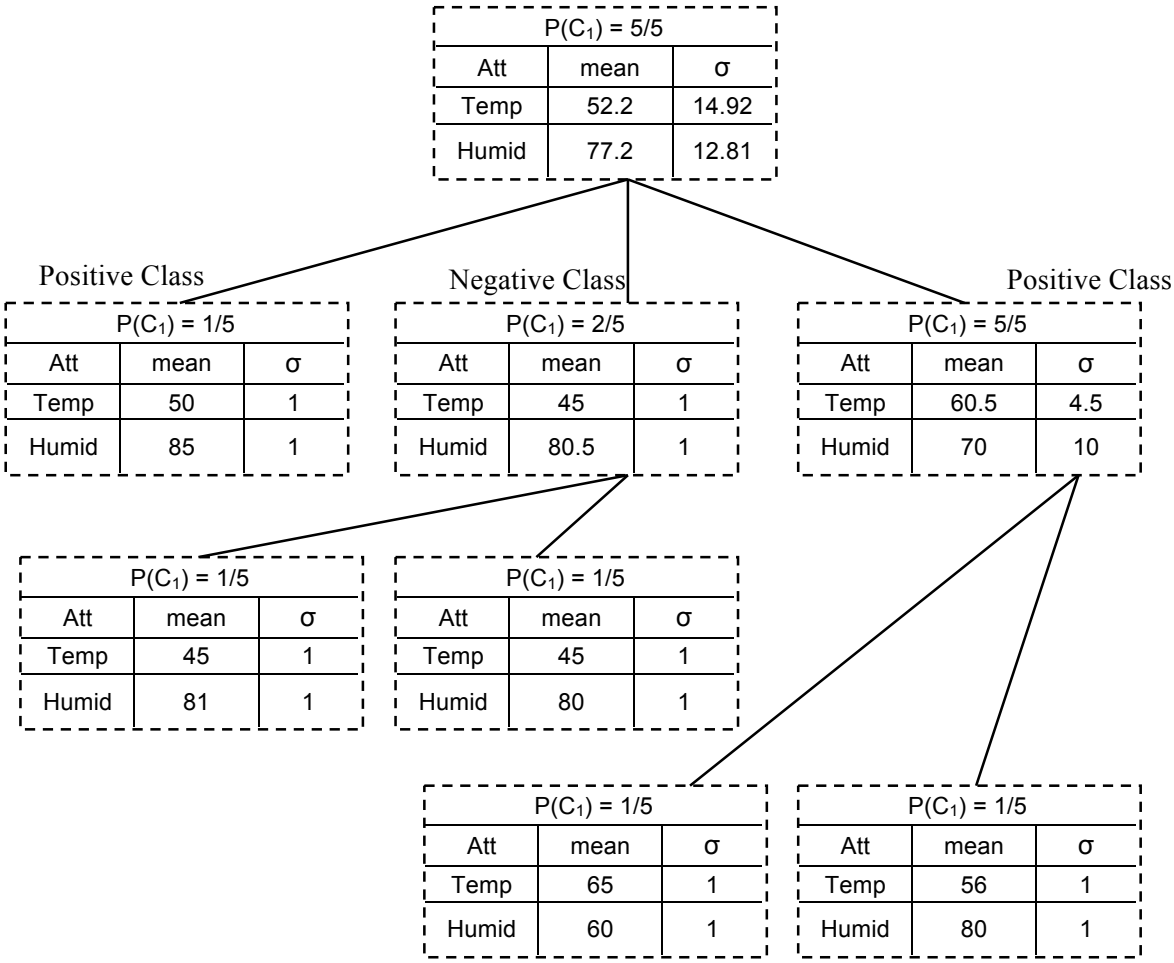


Figure 14. A CLASSIT hierarchy that has been created from 5 instances of the Dynamic Weather dataset.

4.5 OTHER WORK USING COBWEB

Since its publication, COBWEB has been, and continues to be, one of the more recognised conceptual clustering algorithms and has over 1200 citations. Six years after Fisher's initial publication, Fisher and Xu et al published details (1993) of COBWEB being implemented in a range of real world applications such as: fault diagnosis, bridge design, and human gait analysis. The method was shown to be of use, not only in machine learning research but also in real applications. This has resulted in other researchers extending and modifying COBWEB in various ways to increase its suitability for different domains and problems. Two of these research projects are of sufficient importance to this research to explore here. These are titled: ARACHNE and COBBIT. However, three other methods will also be briefly mentioned first, as examples of other modifications researchers have completed.

Of the various extensions and modifications that researchers have made to COBWEB there is a clear divide into two groups according to the type of modifications made. The first is to modify the COBWEB method itself. An example of this can be seen in the work completed by (Sahoo, Callan et al. 2006) where CLASSIT's numerical category utility was modified from being a normal distribution to being a Katz's distribution. Katz-CLASSIT and CLASSIT were then compared on the analysis of textual documents with Katz-CLASSIT being the superior method.

The second type of modification made to COBWEB involved the integration of COBWEB with existing techniques in order to produce an ensemble learner. These systems are created by researchers who want to leverage the strengths of COBWEB, but found components in COBWEB to be detrimental within their domain of interest. Instead of modifying the technique itself, as in Katz-CLASSIT mentioned above, or COBBIT described below, they have used it in conjunction with another method largely as it was designed. Li, Holmes and Pfahringer (2004) combined COBWEB with k-means to produce a method that can scale to large datasets yet uses a smaller memory footprint, while still possessing a knowledge hierarchy. Within this method instances are first observed and categorised within COBWEB, but once within a sub category, they are then clustered using k-means to produce a finer grained grouping. The knowledge hierarchy still persists for the over arching concepts, but the smaller k-means built clusters are built within these.

In a similar fashion Jungsoon et al (1996) describes an example where COBWEB is used after another method has examined the instances. Jungsoon et al's method is designed to develop a hierarchy of knowledge, but, to reduce the effects of the order dependency of COBWEB, the data is pre-processed using a Genetic Algorithm (GA). The hierarchical GA used by Jungsoon chose which of the initial instances to observe first. The initial instances have a major impact upon the shape of the COBWEB tree produced, and the GA was able to decide which were the most suitable instances to be examined first. Once this stage was completed COBWEB would then continue to integrate new knowledge as it arrived, and produced an improved tree compared with that produced without the use of the GA.

These three methods illustrate how the COBWEB method has been an active area of research. These methods are not of great relevance to this thesis; however, two that

are will now be examined in more detail. The first of these, called ARACHNE, aims to improve COBWEB's control mechanism. The second, called COBBIT, equips COBWEB with the ability to adapt to concept drift.

4.5.1 ARACHNE

Following on from the work that was completed in COBWEB and CLASSIT, one of the collaborators on those methods produced another work called ARACHNE (McKusick and Langley 1991) (Langley collaborated with both Fisher (Fisher and Langley 1985; Fisher and Langley 1990) and Gennari (Gennari, Langley et al. 1989) in publications in this area discussed previously).

The goal of ARACHNE, as described in McKusick and Langley (1991) and Iba and Langley (2001), was to modify COBWEB's learning mechanisms to better handle two of the respects in which it was shown to perform poorly: initial ordering bias and noisy datasets. The first of these was caused by its heavy order dependence. The hierarchy produced from COBWEB can reflect the ordering in which the observed instances were incorporated by the learner. If a poor ordering is presented, then the tree may produce a less than optimal structure, and does not correct itself particularly well. The second is that COBWEB, while performing well as a predictor, could at times produce clusters that are not as clearly identifiable as concepts as would be desired. McKusick and Langley argue that this is particularly apparent within datasets that contain a sizeable amount of noise. ARACHANE aims to modify the COBWEB mechanisms to allow the hierarchy to have better self-maintenance abilities to improve its performance in the presence of a bad initial ordering or significant amounts of noise. This is undertaken by adding additional restructuring mechanisms to the learner. Similar hierarchical re-organisational work has also been undertaken with other learning methods by Reich and Fenves (1991) and Nevins (1995) in finding misplaced clusters.

The main addition to the control mechanism in ARACHNE involves two constraints that verify that nodes in the hierarchy are in the correct place. The constraints examine the correctness of the vertical and horizontal placement of nodes in the knowledge hierarchy. When a new instance is placed at a node in the hierarchy it modifies the probabilistic description of its parent node. The constraints then ensure that the neighbouring nodes are still in the correct location within the structure. Once

an instance is added, each of the sibling nodes of the parent concept are examined to see if they meet the vertical constraint. Each node is compared to both its parent and grandparent nodes to ensure that it is still more similar to its own parent than to its grandparent, even though another instance has been added to the concept. If a node is now more similar to its grandparent node than its parent node, it is lifted up in the hierarchy. As this again changes the concept description of the parent, each of its child nodes is then again compared to enforce the vertical constraint. This process has enables a misplaced node to be lifted up in the hierarchy, allowing it to find a more correct location.

After the vertical constraint has been satisfied for all the child nodes of the concept in which the newly added instance resides, the second constraint is then examined. The second constraint examines the sibling nodes to ensure that they are horizontally well placed. Each of the nodes is compared to see if any two nodes are more alike to each other then they are to the parent node. If this is found to be the case, then they are merged to produce a new child node of the parent concept. In this way differentiation within an existing concept is encouraged, refining the cluster quality.

Over the course of a dataset these corrective processes allow for misplaced nodes to be lifted up and then lowered (through merges) to produce a tree that is more resilient in the scenarios mentioned above. These mechanisms extend the COBWEB merge and split mechanisms by focusing on the sibling nodes that were not affected by the addition of the new instance. McKusick and Langley (McKusick and Langley 1991) found that ARACHNE performed quite similarly to COBWEB on two natural domains (Congressional voting and Soybean), with ARACHNE reaching convergence a little faster. However, they did find that in datasets that were noisy there was a more substantial difference between the two methods and that this did increase as the noise level increased. Further Iba and Langley (2001) showed that ARACHNE could recover from a poor ordering much more rapidly then COBWEB could, showing that ARACHNE could perform more effectively under the two target scenarios.

4.5.2 COBBIT

COBBIT by (Kilander and Jansson 1993) is a variation of COBWEB that allows it to adapt to concept drift. When Kilander and Jannsson approached the problem of

concept drift in the context of COBWEB they found three shortfalls within Fishers method. These are:

1. COBWEB does not differentiate between old and new instances, meaning that if drift occurs then more of the "new" instances are required to shift the concept.

2. COBWEB retains all observed examples which results in a scalability problem in large datasets.

3. COBWEB is order dependent, meaning that the hierarchy created can change depending on the order of observation of a set of instances.

When Kilander and Jannsson approached these three problems they decided that there needed to be a way of removing knowledge from the hierarchy once it was no longer needed, or once it was out of date. If change occurs within the domain over time, then the knowledge that predates this change should be removed. Further, a method of deciding when this should occur would be needed. They discussed utility and predictiveness measures and instead opted for the less computationally expensive method of having the method function within a time window of the whole dataset, but only examining a portion at any given moment. This approach is very similar to the one undertaken by the supervised learner FLORA (Widmer and Kubat 1996) discussed in Chapter 3.

The window used within COBBIT is a first in first out (FIFO) approach. A least-recently used approach was contemplated, but the FIFO approach was adopted. Figure 15 illustrates the way a FIFO method operates on a stream of instances being observed by the system. Within COBBIT this is implemented using a list that was

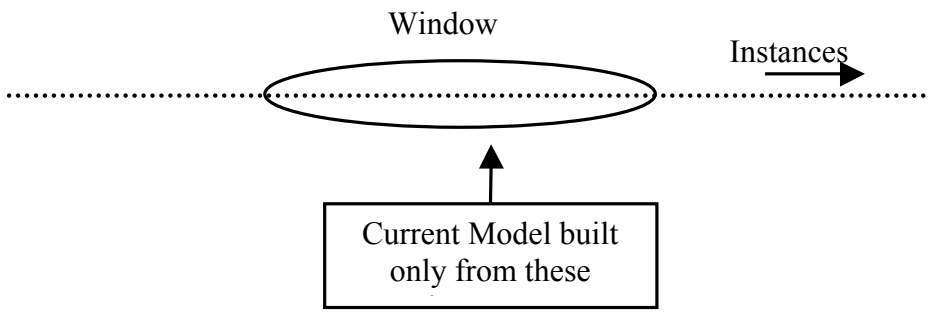


Figure 15. The Time Window used within COBBIT

separate to the COBWEB structure, but when an instance was removed from the list, a delete operator was called upon the COBWEB tree to subtract the knowledge pertaining to the instance. This was done recursively to update the knowledge, at each parent node, of the resulting category in which the instance was stored. Other than the simple subtraction of the instance from the tree, the COBWEB method was largely unchanged from that presented by Fisher. This was possible because the COBBIT control structure was separate from COBWEB, treating COBWEB purely as a component within COBBIT.

Testing completed by Kilander and Jannsson showed that COBBIT easily out performed COBWEB, in terms of accuracy, on concept drift problems. It also used fewer computer resources and ran faster on large datasets. As COBWEB retains all of the data, by removing the older instances it means that new additions are less computationally complex, resulting in an increase in scalability. In Chapter 6 COBBIT will be compared with DynamicWEB on standard concept drift data.

4.6 SUMMARY

This chapter has examined the COBWEB conceptual clustering algorithm in the context of the work prior to it, and then also other work that has built upon it. Most notable of these post-COBWEB developments is the extension CLASSIT enabling the learner to operate upon numeric data. Also briefly examined was the method ARACHNE that aims to counter some of the suggested shortfalls of COBWEB. Then finally the concept drift tracking method COBBIT was introduced. This will be used in the work described in Chapter 6.

The next chapter will look at a new extension to COBWEB that aims to allow it to profile objects over multiple observations. This method will build upon the strengths of COBWEB while equipping it to operate within a different learning scenario to that for which it was originally designed.

Chapter 5

DynamicWEB: Learning over Multiple Observations

πάντα χωρεῖ καὶ οὐδὲν μένει

"Everything changes and nothing remains still."

Heraclitus of Ephesus (535–c. 475 BC)
Panta rhei

INTRODUCTION

The previous chapters have reviewed the different conceptual clustering and concept drift methods presented in the literature. In particular the COBWEB conceptual clustering algorithm was thoroughly examined. This chapter builds upon this knowledge by presenting a new method entitled DynamicWEB. DynamicWEB is a modification of COBWEB that enables it to operate upon target subjects that are re-sampled many times.

5.1 INTRODUCTION TO DYNAMICWEB

DynamicWEB is an unsupervised conceptual clustering approach which aims to adjust to concept drift as well as drift of a target object from one resultant class to another, over time. This second form of drift, which is labelled *Object Drift*, is fundamentally different from the problem focussed on by previous work within the area of concept drift, although it is an obvious progression within machine learning. This chapter will first discuss this problem of object drift and the goals which need to be met to deal with this problem, before then detailing the algorithms used in DynamicWEB and the implementation of these.

5.2 THE PROBLEM

Concept drift, as already discussed within Chapter 3, is the process where a class definition changes over the course of time within a dataset. Methods which are able to handle concept drift adapt to this shift within the fundamental knowledge that describes a given resulting class. The concept model produced to these methods is updated as drift occurs. This facilitates a more robust classification technique well suited to time-related knowledge domains. Within the datasets examined by existing concept drift research, and indeed a large portion of data mining research, each object being examined is only described once (Figure 16). It is quite common for identifiers for the individual objects not to be present within the dataset as it is accepted that they are all separate entities. These datasets can be referred to as being latitudinal, having a singular observation of many different objects, each referred to as an instance. As there is only a single observation of each object, methods of adapting to concept drift use techniques such as time windows, or utility measures. As each object only appears once, these instances are all of equal worth initially, and

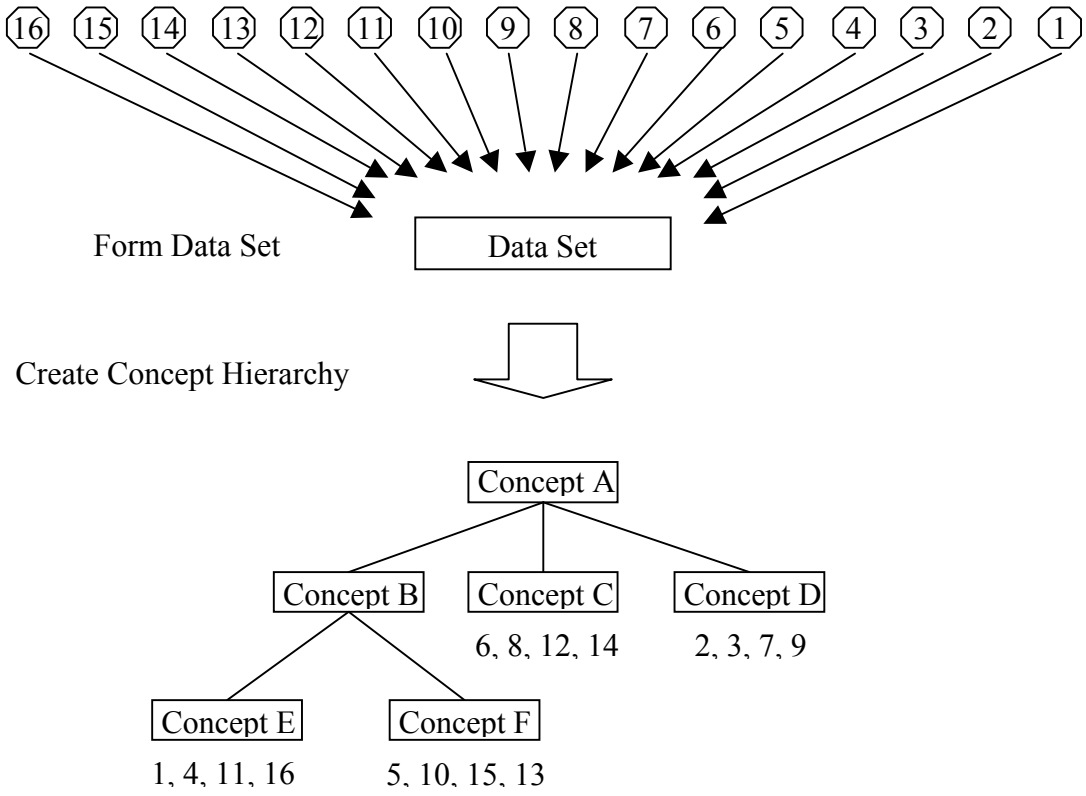


Figure 16. In Concept Drift each object is only observed from one instance in the dataset. During the creation of the hierarchy the concept descriptions are able to adapt to variation between instances of the same class. (Arrow heads indicate the number of times an object has been sampled).

so the quality (in terms of predictiveness) of a given instance can be determined via age or utility quite effectively. Figure 16 illustrates how a concept hierarchy is formed from single observations of objects. As each instance is integrated into the structure the concept descriptions are able to change and adapt to the information being received, to reflect the shared characteristics of the instances resident in each concept.

The problem being examined in this thesis is related to concept drift. However, for this work, the drift is that of an object of interest moving from one resultant class to another upon the observation of more data. Within concept drift each object is usually only observed once within a dataset, and as such has a single instance relating to it within the dataset. However, there are learning problems where a given object is sampled multiple times, resulting in many instances within the dataset relating to an individual object. Across these observations it is likely for an object to change characteristics and possibly the classification. If these changes are simply recorded as another instance being examined, and the change is not linked back to

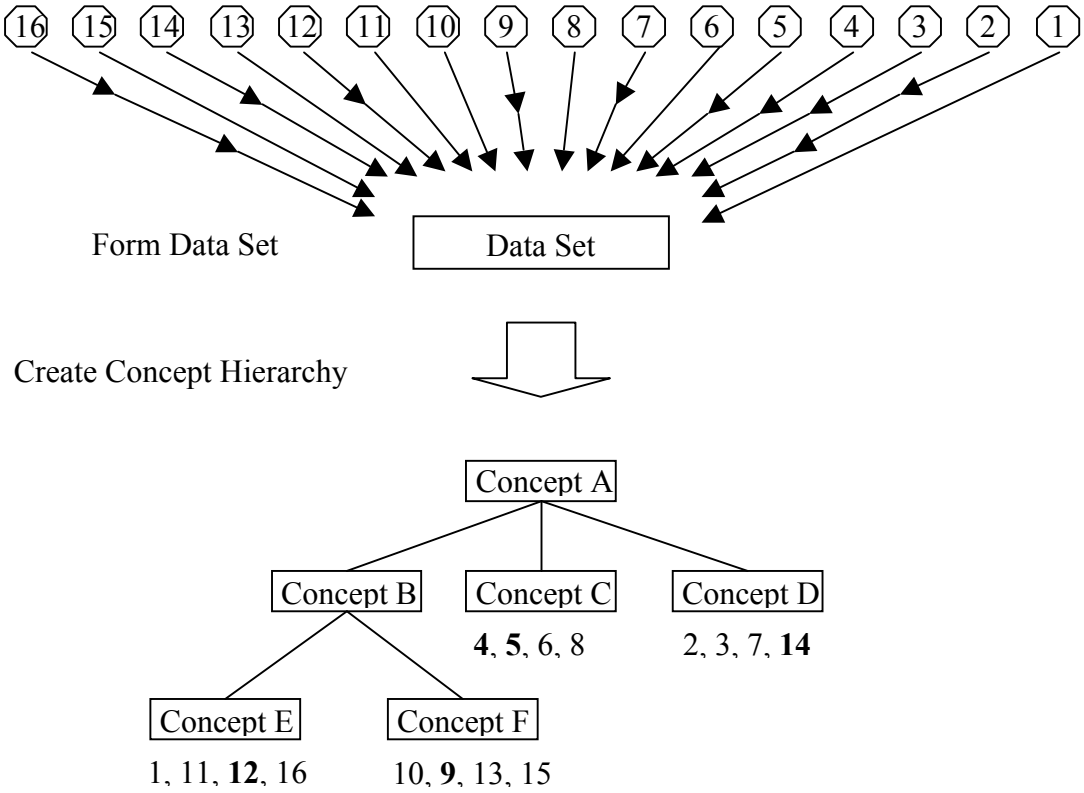


Figure 17. Object drift is the change of an object across multiple observations as recorded within instances in the dataset. Each object is able to be sampled more than once.

the original observed instance, then the knowledge that could be gained from examining the change is lost. For example, if the objects being examined are cars which were all sold around the same time and then tracked together as a group over their various services that occurred during the years following, the linking of these various observations is of more use than if each observation has been treated as a separate instance. Figure 17 illustrates a data set being formed from differing numbers of observations of disparate objects. Each object is represented within the dataset by several instances from different observations. A concept hierarchy has been formed from the dataset, and the instance-profiles listed in the tree in bold are those which have drifted from one resultant concept group to another when compared with Figure 16. These instance-profiles are the combined result of multiple observations of a single object.

5.3 LEARNING GOALS

The previous section outlined a problem type called object drift which is related to, although not addressed by, concept drift. A method is presented in this chapter that is

able to learn in the presence of both concept and object drift. This method was designed with the following goals:

1. To maintain a profile of an object built from multiple observations
2. To establish relationships between profiles of multiple different objects
3. To adapt to concept drift within the domain of the target objects
4. To allow a profile to migrate from one resultant concept to another when new data relating to the object is observed
5. To operate in a noise tolerant fashion
6. To enable searching for a given profile to scale to large datasets

These goals will now be expanded upon below:

To be able to adapt to object drift, the item which is being examined by the algorithm needs to be associated with all the available relevant knowledge about the target object. This knowledge is stored within a profile (1) and is updated as new knowledge is observed. This allows for drift to be adapted to the context of past states of the objects' attributes. Depending on the domain in question and the individual attribute value pairs, this may result in the need for derived attributes to be used to detect any trends taking place that may affect classification. These derived attributes are measures produced across multiple observations of each target object.

Once instance-profiles have been created by the observation of multiple instances of each target object, they can then be compared to one another to establish what relationships may be present within the dataset (2). This enables the creation of a hierarchy that represents these relationships. As this problem space involves drift, it is crucial to have a method that is able to adapt to concept drift (3) and object drift (4).

An important factor that needs to be considered within concept drift is noise tolerance (5). If noise is mistaken for drift then the model can be adversely affected which will then negatively affect classification accuracy. Producing a method that can adapt to concept drift without performing badly under noise is challenging and is

discussed at length in the literature, along with most authors' descriptions of their methods for dealing with this (STAGGER and FLORA as discussed in Chapter 3). Any new method developer needs to keep this in mind. Along with noise tolerance another factor to plan for is scalability. Within most machine learning methods the structure produced has very little to do with any identifier that is attached to a given instance. As such, searching the produced data structure to find a previously observed instance is somewhat difficult to do efficiently in large datasets, as it basically requires each item in the structure to be examined. As the problem space relies upon profiles that are built across multiple observations, requiring a method for instance look-up and updating, the search phase needs to be efficient enough to allow the method to scale to real world datasets of real applicability (6). Otherwise the method would be of limited application.

5.4 DYNAMICWEB

DynamicWEB is the learning method presented within this thesis that aims to fulfil the goals outlined above. Most of the goals pertain to the ability to adapt to object drift occurring within a dataset, in the context of the dataset covering a period of time. As this change occurs, the aim is to adapt the resulting concepts, and to allow for an object to be updated and re-assigned to a concept, allowing it to drift from one class to another. These processes require a concept formation method to be used, but do not require a new one to be designed for the purpose. As this thesis focuses upon the problem of drift, an existing concept formation method will be incorporated into the method, with modifications made to it to meet the other goals discussed above. The following section outlines what changes need to be made to an existing method to allow it to achieve these goals.

5.4.1 COBWEB: THE CHANGES REQUIRED

COBWEB (Fisher 1987) was chosen as the existing conceptual clustering method to be used as the foundation of the new technique. COBWEB, as explored within Chapter 4, is a well respected and expanded upon method. It has also been put to use within real world applications, and has been previously utilised within the domain of concept drift (Kilander and Jansson 1993). By using an implementation of COBWEB that includes the CLASSIT extension (Gennari, Langley et al. 1989), the method is able to operate upon both numeric and nominal datasets, allowing it to handle a

wider range of target datasets. The manner in which numeric data is handled in COBWEB gives it a distinct advantage over other conceptual techniques, due to its probabilistic nature compared with the conjunctive conceptual techniques used in other methods. However, COBWEB in its original form was not intended to have any changes occur to the instances that were present within the tree structure. This is obviously a key component within a method where profiles are required to be updated frequently, and presents the first hurdle to overcome in allowing for COBWEB to function as an object drift adaptive method.

Within Fisher's COBWEB there is no provision for modifying or removing an instance once it is within the structure. Furthermore, a modification of an instance within the tree itself would change the category utility of the node containing it, and any parent nodes, thus adversely affecting the integrity of the tree. Any operators that are going to modify or remove an instance would need to update the utility score of the node and all its parents.

In order to add operators to COBWEB that facilitate removal or modification of an instance, a search of the structure needs to be implemented. However, the COBWEB tree is sorted based on the similarity of the instances resident within it and so, to locate a given instance, each instance would have to be examined in turn, resulting in a O(n) search time (where n is the total number of observed instances). Such a search time severely hinders the scalability of this search mechanism to large datasets, and so is insufficient for the needs of this proposed learning method.

If COBWEB is the existing technique to be used within the learning method the following list of shortfalls of COBWEB for this purpose would need to be addressed:

1. COBWEB does not contain an update mechanism for an instance already present within the structure.

2. COBWEB does not allow for deletion of instances already present within the structure.

3. The search time for COBWEB is too high to scale effectively.

5.4.2 SEARCH IN DYNAMICWEB

To achieve these goals, DynamicWEB needs to enable multiple examinations of the same object over time. These examinations require that the representation of the given object within the COBWEB tree be kept up to date. To differentiate between the objects being examined, the datasets must contain an identifier for each object. To facilitate efficient modification and deletion operations, DynamicWEB needs a search method to be able to locate instances within the COBWEB concept hierarchy based upon this identifier. The hierarchy itself is sorted based upon the similarity of the instances to one another, so a traversal of the tree would result in a O(*n*) search time, and is independent of the identifier. To utilise this identifier, and overcome the fact that the search is concept based, while not drastically modifying the COBWEB structure itself, an index structure was created to act in tandem with the concept hierarchy. When an operation upon an instance already present within the concept hierarchy needs to occur, this second structure uses the identifier to locate the node, within the COBWEB structure, where this instance is currently located.

In deciding the type of indexing structure to use for this function two methods were trialled: a Hash table and an AVL tree. These two methods are both well suited to the role of an index, with the hash table boasting an expected search time of O(*1*) (under suitable distribution assumptions) and the AVL tree (Adelson-Velskii and Landis 1962) a worst case search time of O(log *n*). With a hash table, however, to maintain a O(*1*) the ideal table size needs to be large enough relative to the dataset. As such an automatically sized table also with a worst case of O(log *n*) search time was used here. With each method it is possible to store data within the index, and in this implementation a reference to a node within the COBWEB hierarchy is stored.

	Hash Table	AVL Tree
Insert		
66009	6250	2359
30000	1609	984
10000	359	328
1000	47	47
Insert and Lookup		
66009	11109	4265
30000	1641	985
10000	375	328
1000	47	47

Table 17. A comparison of performance a Hash table and an AVL tree. The metric is measured in millionths of a second

Table 17 shows the result of using these two different methods over a range of dataset sizes. The identifiers used in this table were taken from a security audit log of traffic probing a firewall. The 66,000 total identifiers are all unique IP addresses which probed the gateway. The dataset that was used to obtain these identifiers is examined within Chapter 8.

The AVL tree was found to outperform a Hash table over large numbers although the Hash table matched it for speed while the dataset was quite small. The top half of the table shows the time taken, in millionths of a second, to create the index from the list of identifiers, while the times in the lower half of the table also include using the structure to lookup the value to retrieve a pointer to another data structure. For the vast bulk of datasets that would be likely to be examined, this table shows it would not matter whether a hash table or AVL tree was used due to their limited number of items to track, however, this research is focussed on datasets with very large numbers of unique identifiable objects and the AVL tree's scalability makes it the obvious choice.

With each identifier the AVL tree stores a reference to the location of the cluster where the instance is found in the COBWEB hierarchy. This 'tying together' of the

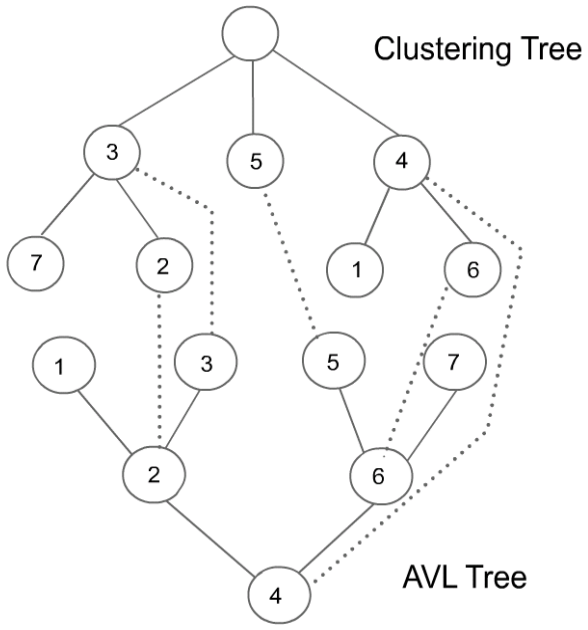


Figure 18. The AVL Tree acting as an index to the COBWEB concept hierarchy (links for profiles 1 and 7 are not shown). Each instance stored in the concept hierarchy has a counter part in the AVL with a pointer to its location.

two tree structures is illustrated in Figure 18. The AVL structure is sorted on the identifiers 1 through 7, which places 4, as the middle point, at the root, while the COBWEB concept hierarchy is sorted on the contents of the instances, and appears haphazard to a human observer (or search mechanism) of the identifiers present.

5.4.3 UPDATE MECHANISM

The major advance on the COBWEB algorithm, provided by DynamicWEB, is the ability to track an object over time by building a profile from the observed instances relating to the object. The contents of this profile will be discussed further shortly, so we will just discuss it as an instance for the moment. In the COBWEB algorithm by Fisher there is no mechanism to modify an instance once it has been assimilated into the hierarchy. The probabilities which are acting as the concept descriptions obviously contain the relevant knowledge about all the instances below a node. There are scenarios where simply updating the instance in place and then flowing those effects through would be acceptable, and indeed, in these cases, would be also more efficient, but this is not true of all cases. In some instances where attributes are to be replaced, the variation could cause the instance to move to a new location a significant distance away. The migration process would then be cumbersome, and computationally complex. Therefore DynamicWEB adopted the approach of removing the profile from the tree, adjusting the surrounding area, and then re-adding the updated profile. We will now examine this process.

DynamicWEB(I, D) *// Identifier of target object, new instance data of target object*
Let R be the root of the concept hierarchy tree
Let V be the AVL index tree
If I is found within V let its location in the hierarchy be L
    Call **remove(I, L)** returning the profile in its current form as P
    Let the result of calling **updateProfile(P, D)** be P
    Let L be the resulting location of adding an updated P to the concept hierarchy calling **COBWEB(R, P)**
Else
    Add I to the index V, creating profile P
    Let L be the resulting location of adding P to the concept hierarchy calling **COBWEB(R, P)**
Set the reference stored at I in V to being L

Table 18. The DynamicWEB Update Mechanism

Table 18 above shows the highest level of the DynamicWEB update algorithm (there is a complexity discussion is Appendix F). This is the portion of the algorithm that handles interaction between the concept hierarchy formed by the COBWEB algorithm and the AVL tree acting as an index. When a new instance is presented its identifier is searched for within the index to see if it already has a profile within the concept hierarchy. If the object has not yet been observed the new instance is added directly to the hierarchy. If the object has already been observed at least once, then the profile is extracted from the hierarchy and all knowledge of it is subtracted from the node in which it was placed, as well as from all of its parents. Once the profile is

remove(I, N) *// Identifier of target object, Node where existing profile of target object is stored, or its parent when called recursively*

For each profile, J, at Node N

    If the identifier of the J^{th} profile matches I

        Let P be the J^{th} profile

Remove P from N

Recalculate the probabilistic concept description for the node N

If current node is not the root

    If N is a leaf then call **remove(I, the parent of N)** else do the following

    For each Child, C, of node N

        Calculate category utility for placing P in C

        Let A be the node C with the highest category utility

        Let B be the second highest category utility

    Consider merging the nodes A and B and calculate the category utility for the merger

    Consider splitting the node A and calculate the category utility for the split

    If the highest category utility (of merging, splitting or current state of N) is above the cut off

        Set the state of N according to the relevant scenario, updating the concept description probabilities accordingly

        Call **remove(I, the parent of N)**

    Else

        Let the current state of N stand and do not consider merging or splitting in the future

        Call **updateTree(I, the parent of N)**

return P

Table 19. The remove function of DynamicWEB

removed the statistics that comprise the probabilistic concept description need to be updated for each parent node back to the root from the node in which the instance was clustered. If this does not occur the probabilities representing the concept are no longer correct once the profile has been removed.

The remove function (Table 19) starts at the node which is returned by the lookup ofthe index. It extracts the profile that matches the current object's identifier, which is returned once the concept hierarchy has had the profile removed. The concept description for the current node is then recalculated. If the current node is not the root, then the remove method is recursively called with the target node to be examined being the parent of the original. This occurs instead of considering a merge or split of its children because the children were not affected by the removal of the profile. All subsequent calls of the remove method up the tree repeat the extraction of the profile from the node, and the updating of the concept description. Merging or splitting the children of the node most similar to the removed profile is then considered. This refinement of the tree allows the clusters to be in their optimal fitting form after the knowledge removal has occurred. However these operations are not considered at every node back to the root. The operations are only considered until at one level the resulting category utility is less than the cut-off threshold utilised by the insert operation within the CLASSIT extension to COBWEB. This enables the removal operation to be more efficient by only considering a merge or a split when one is likely to actually be of consequence. Once this threshold has been met, the remove operation is no longer called recursively back to the root as it is extremely unlikely it would pass that threshold again. Instead the updateTree operation is called. The updateTree function (Table 20) removes the profile from

---

updateTree(T, N) *// Identifier of target object, a node which has knowledge of the node to be removed*

For each profile, J, at Node N

    If the identifier of the J^{th} profile matches T

        Let P be the J^{th} profile and then exit loop

Remove P from N. Recalculate the probabilistic concept description for the node N

Recalculate the category utility for N

If N is not the root node then call **updateTree(T, the Parent of N)**

Table 20. The updateTree function of DynamicWEB

each node from the point it is called to the root node, at each stage updating the concept description. Once the updateTree and remove methods have fully removed the profile from the concept hierarchy, it is returned back to the DynamicWEB function in Table 18. This profile is then modified to reflect the new information present within the most recent observation. This process, and more detail about the profiles will now be examined.

5.4.4 PROFILE

For DynamicWEB to be able to adapt to object drift within a dataset over time the observed instances need to be examined in such a way as to establish what change has occurred and if there is knowledge to be acquired in these changes. To do this DynamicWEB uses profiles of activity built across the multiple observations to adapt to object drift as it occurs within a dataset. The profiles need to be robust and customisable to allow them to function across a wide range of domains containing a diverse range of attributes.

The profiles within DynamicWEB are comprised of two main types of attributes: most-recent attribute value pairs and derived attributes. The most-recent value allows for the current state of an object to be used within the learning process, while the derived attributes allow the history of the object to be used within the learning process. The former of these two is largely the same as an instance within COBWEB and other learning methods, being the current state of an attribute on the target object. However, it is kept up to date when new information about the object is observed within the dataset. By changing the value within a profile to the most recently observed value a profile is able to drift from one resultant class to another, or to update the concept description for a given class. However, this is only a representation of the object at a given point in time and does not offer any insights into what trends or history a given target object has been through to aid the knowledge acquisition within the tree.

The derived attributes are used to capture the history of an object and are a combination of all the previously observed values per instance within a data set for a given target object. The derived attributes (Table 21) include: mode, mean, standard deviation, maximum, and minimum. Such attributes have a strong grounding within statistics and establish the historical context of the attribute value pair within the

dataset. These derived attributes allow DynamicWEB to create groups of profiles that are not only similar now, but also historically. This enables a more accurate representation to be generated through the preservation of contextual information about the objects and the dataset. The result is that there is less negative influence, from any noise present within the dataset, upon the resultant conceptual hierarchy, due to the normalising effects of the mean and standard deviation.

Derived Attribute	Attribute Type	Description
Mode	Nominal	Most frequently observed value of this attribute value pair
Count	Both	Count of instances seen relating to the target object
Mean	Numeric	The average value for this attribute value pair across the set of observed instances.
Standard Deviation	Numeric	The standard deviation within the set of observations.
Maximum	Numeric	The highest value that has been observed of this attribute value pair.
Minimum	Numeric	The lowest value that has been observed of this attribute value pair.
Trend	Numeric	An indicator of the trend of values of this attributes value pair.
Gap	Numeric	The difference between the two most recently observed values.

Table 21. Listing of the derived attributes that are used within the Profiles in DynamicWEB

Most of the derived attributes are based upon the observed values of the numeric attribute-value pairs. The derived attribute Count is the only one which operates on both the nominal and numerical types. It is a measure of the number of instances observed relating to a target object. The Mode, the most commonly seen value, is the only other derived attribute used with nominal values. The Mean, Standard Deviation and the other derived attributes are ill suited to describe discrete attributes. However, the mostly commonly observed discrete value (the Mode) does preserve some historical context of the target object. The remainder of the derived attributes are used with the numeric values observed about a target object. Mean and Standard Deviation are of obvious use in describing the historical context of a target's attribute values. Minimum and Maximum further illustrate the extent of the range of observed values.

In addition to these measures, there is a trend value for numeric attributes. If the most recent observation is higher in value than the previous observed value it increments by 1; if it is lower, then it decrements by 1. If there are many increases over a long period of time the trend value is high, likewise if the values are repeatedly decreasing the trend value becomes increasingly negative. This accounts for cyclical patterns of an attribute within a dataset, where a value may increase and contract repeatedly. It is a measure of the context of the part of the cycle a target object is in, in comparison with other objects. Another measure which compares the two most recently observed values is the Gap. The Gap is the deviation between the two most recently observed values for an attribute. This allows a short term deviation to be observed to increase the profile's ability to track change within fast moving data, or within data with a wide variation of deviation which may be smoothed by the standard deviation.

All of these derived attributes are stored within the profile, and are updated whenever a new observation occurs. The updateProfile function (Table 22) updates all the individual attribute-value pairs to the value that was just observed along with their associated derived attributes. The updateProfile method is called, as shown in Table 18, directly after the profile has been removed from the hierarchy and before it is re-added. The change that has occurred within the profile then allows for it to be placed within the hierarchy accounting for the change that has occurred, but also with the context preserved within the derived attributes.

```
updateProfile(O, P) // Old version of the profile, Newly observed object data
For each attribute, A, in O
        If the A^th attribute is derived then
            Recalculate the derived value including new data from P
        Else
            If the A^th value is different in O then in P
                Update the version in O with the value from P
Return O
```

Table 22. The updateProfile function of DynamicWEB

5.4.5 MULTIPLE-DIMENSION TREE

One of the key goals of concept drift algorithms is for them to be resistant to noise that is present within a dataset (Schlimmer and Granger 1986; Widmer and Kubat 1996). If noise is interpreted as drift, then the drift that will occur will be false and hinder the growth of knowledge within the tree. DynamicWEB reduces the impact of noise within a dataset by using derived attributes within each target object's profile. These tend to have a generalising affect over multiple observations, reducing the impact of a noisy observation. However, as the category utility calculation results in a combined value across all attribute value pairs it is possible that two independent variables may act as noise to one another hindering knowledge growth, or over-complicating the concept hierarchy. This is not actual noise, but change that is occurring within unrelated attributes, impacting on performance both computationally and in the resulting hierarchy.

To combat this, and traditional noise, it is proposed to be able to compartmentalise the clustering into parallel trees. This acts to remove any unwanted interference. Furthermore, this can offer a performance boost in situations where new data is observed and only requires some of the attributes to be updated. These smaller trees work faster than the whole, and, if not all of them need to be updated for a given piece of new information, then computation time is saved. Each tree produces its own class definitions and classifications. The classification results can then be combined to form a final result. This approach is not without precedent, with ensemble-clustering being used in concept drift learning methods involving a group of decision trees (Stanley 2003) and another using the combination of an incremental decision tree and naïve Bayes (Kolter and Maloof 2003). Both systems achieved increased classification accuracy by using multiple clusterings when reacting to concept drift. However, these methods did not operate by splitting the instance into multiple subgroups of attribute value pairs, but were effectively time windows of various lengths. The most accurate tree that resulted was the structure that was adopted.

While within DynamicWEB there is no promotion or removal of separate tree structures, the grouping of certain attribute-value pairs into different trees allows for similar performance gains to be made. If a group of attributes is updated frequently,

while others rarely change, the computation required to update a smaller tree comprised of only the frequently changing values is less than for the tree of all attributes. Obviously this does not suit all datasets, however if the context (the interdependence of the attributes from one another) tolerates this by producing accurate results when the clustering results of the trees are combined, then the performance gain is to be welcomed. Examples of multiple trees are examined within the results chapters while the implementation will be discussed now.

5.4.5.1 IMPLEMENTING DYNAMICWEB'S MULTIPLE DIMENSIONS

DynamicWEB operates upon multiple observations which are combined into a single profile within a clustering structure. The update mechanism for this involves the use of an index which is based on an AVL tree. To extend DynamicWEB to operate across multiple tree structures, each of these need to be indexed to allow for the updates to each tree to occur independently of each other. This allows for one tree to be updated, while another remains in its current form. However, it is not necessary to duplicate the index structure itself, but merely to extend the existing one to map out not just a single concept hierarchy, but several. Each entry within the index not only links to the existing profile within one tree, but to each of the different portions of the profile stored in each tree currently in use. Figure 19 illustrates an index in use across three different concept hierarchies, labelled C1, C2 and C3. The target object A's data is stored in three part-profiles, one in each tree. Each profile can be quickly

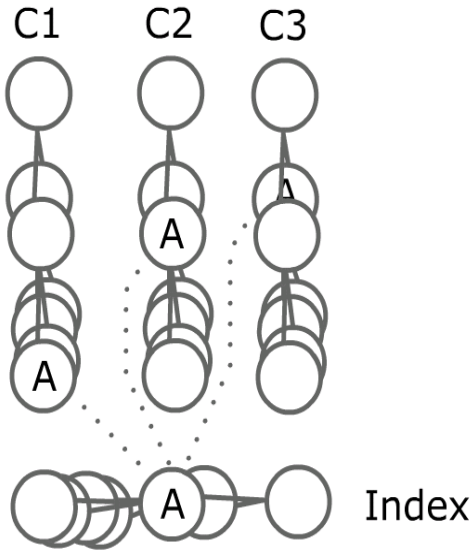


Figure 19. The index mapping out the location of the profile 'A' within three separate concept hierarchies.

accessed via the index and updated as new data relating to the attributes stored within each portion is observed.

To facilitate this more segmented update process, the DynamicWEB algorithm listed in Table 18 is modified and shown as Table 23. As each new observation is observed by DynamicWEB, it searches the index for an object with the same identifier. If the identifier is not found then the profile is split up into the separate profiles and added to each tree, with the resulting location within each tree recorded within the index. If the identifier is already present within the index then the individual portions of the profile are updated with the newly presented information. If a profile portion is

---

DynamicWEB(I, D) *// Identifier of target object, new instance data of target object*

Let H be the set of root nodes for the concept hierarchies

Let V be the index tree

Let S be the result of splitting D into a set of part profiles for each H

If I is found within V let its list of locations be L

  For each hierarchy, J, in H

    Let N be the J^{th} node in L, that for the hierarchy J

    Let K be the part profile within N with the identifier I

    Let T be the J^{th} value in S, that for the hierarchy J

    Let T now be the result of combining the existing data in K with the newly observed data within T

    If T not equal to K then

      Call **remove(I, N)**

      Let M be the resulting location of adding the updated profile T to the concept hierarchy J with **COBWEB(J, T)**

      Set the J^{th} value in L to be M

Else

  Let L be a new node in V for the identifier I

  For each hierarchy, J, in H

    Let T be the J^{th} value in S

    Let M be the resulting location of adding T to the concept hierarchy J calling **COBWEB(J, T)**

    Set the J^{th} value in L to be M

Table 23. The DynamicWEB Update Mechanism using multiple trees

unchanged then it is not updated, meaning that it is not removed and re-integrated into the concept hierarchy; the existing profile remains as it was. This saves on the computational cost of re-clustering an item which has not been changed in the update process. Such an occurrence results when a profile portion is comprised of attributes that are not derived, but are merely showing the most recent value observed. If, however, the profile does contain derived attributes, or the observed value from the profile has changed, then the profile is removed from its hierarchy and re-inserted into the tree from the root location. Each portion of the profile is examined, one after the other, with some resulting in an update while others may not.

Once the multiple trees are established, they can then be examined and compared for structural differences between the concept hierarchies. By examining the dataset in separated portions, different patterns within the data may be discovered. Furthermore, the classification results produced by the different structures may also be different and these can be examined, or combined to form an overall classification across the forest. The simplest way to handle this output is to give each tree equal weight, and produce an overall classification value, based upon a majority of resultant classes as voted by each tree. This is not a new idea but is related to Bagging (Breiman 1996) and Boosting (Valiant 1984; Schapire 1990; Schapire 2002). However, unlike Bagging, also known as bootstrap-aggregating, the models are not separate models with each formed from a slightly different set of training instances, but rather different parts of the one stream of instances that are observed. Likewise, it is not the same as Boosting, which allows for different weights to being applied to the different models, based on their predictive performance, in that DynamicWEB is a completely unsupervised method. However introducing some weights in a boosting approach based on performance is an extension that could be made.

5.5 SUMMARY

This chapter has outlined the DynamicWEB learning method that was built upon the existing machine learning technique known as COBWEB. The fundamental difference between the two is that DynamicWEB was built with the goal of being able to profile objects across multiple observations of the same objects over time. Allowing for both concept and object drift to take place within the learner.

The DynamicWEB update mechanism uses removal and profile update mechanisms that employ an index structure. As DynamicWEB aims to compare profiles of activity within a time-based context, the update mechanism involves the use of derived attributes that aim to preserve contextual information about the previously observed values of the attributes within an observation of a target object. By preserving this context, DynamicWEB aims to establish and track relationships between target objects over many observations.

In addition to context preservation this chapter has described the use of DynamicWEB as an ensemble learner, utilising multiple concept hierarchies in parallel with each other. The aim of this is to split the learning task into smaller sub tasks for comparison with each other or as components of an ensemble classifier.

DynamicWEB has been described here as an elegant modification and extension of the COBWEB method to allow it to adapt in learning environments which contain change. The chapters that follow examine DynamicWEB's performance across a range of problem domains which contain change.

Chapter 6

Method Verification and Demonstration

"Change alone is eternal, perpetual, immortal."

Arthur Schopenhauer (22 February 1788 – 21 September 1860)
German philosopher

INTRODUCTION

The previous chapter has outlined DynamicWEB, the method being presented within this thesis. This chapter and the following two present results of this method when trialled across a range of datasets. This chapter aims to verify that DynamicWEB functions correctly upon several small well studied datasets. Several results are compared to published results of COBWEB to illustrate that the additions that have been made have been undertaken correctly.

6.1 INTRODUCTION

When a new method is proposed it is important to verify that the method functions as it should upon small understandable datasets and not just demonstrate it upon large datasets that may contain unknown patterns. The method proposed within this thesis, as detailed within the preceding chapters, is quite specialized compared to many techniques that already exist within the neighbouring areas. Further the datasets which inspired this work within security contain a large number of unknowns (in relation to the true classification results), meaning that they are not practical for fully verifying a method.

A key component of verifying that DynamicWEB functions as it was designed is to confirm that the underlying implementation of COBWEB (Fisher 1987), and the CLASSIT (Gennari, Langley et al. 1989) extension, is also correct. This chapter will first examine the performance of DynamicWEB on a few small datasets which were examined within the previous work on COBWEB. Then a dataset that was created for testing DynamicWEB more fully will be described and examined. This dataset was based upon a popular dataset in machine learning, but was extended to allow for multiple observations of the target objects. Thirdly a dataset that has been commonly used within the concept drift literature is examined (Schlimmer and Granger 1986). This leads to a comparison between the two concept drift methods built upon COBWEB: DynamicWEB and COBBIT (Kilander and Jansson 1993).

6.2 VERIFYING THE COBWEB IMPLEMENTATION

The COBWEB implementation that is contained within DynamicWEB was implemented based upon the pseudo code and accompanying explanation that was

included within the 1987 and 1989 papers (Fisher 1987; Gennari, Langley et al. 1989). In these papers several results were presented and these will be compared to the results achieved by the DynamicWEB implementation. Along with these results, the values for the category utility were also provided in the papers, and were demonstrated within a sample problem as multiple incorporation operations were undertaken. These results were matched to the DynamicWEB implementation in order to ensure that this part of the algorithm was functioning correctly.

In the first publication of COBWEB, Fisher (1987) focused upon the Small Soybean dataset (Michalski 1980). His testing methodology, drawing upon Quinlan's (1986) work, operated using training and testing sets. These were not separated at the commencement of a trial, and the same dataset was used to test and to train. However the training was tested at intervals during the knowledge acquisition. As the knowledge hierarchy was grown after every 5 instances, the structure was tested upon the entire remaining unseen items within the dataset (the dataset contains 47 instances in total). There are four classes within the dataset with three of the classes having 10 examples, and the fourth having 17. When this testing occurred, each item was classified by the hierarchy without any knowledge of this item being retained by the structure in the process. Figure 20 illustrates the learning performance of the DynamicWEB implementation compared with the published performance of COBWEB.

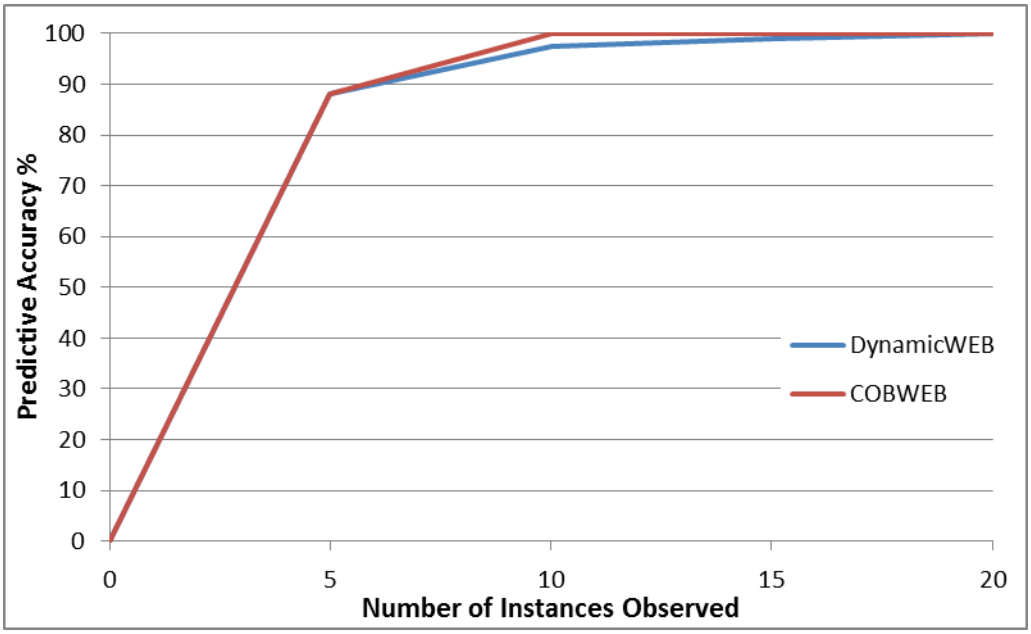


Figure 20. Fisher's implementation of COBWEB vs the DynamicWEB implementation.

The implementations operate almost identically with only a minor difference between the two. Fisher's results show that it reached 100 percent predictive accuracy after witnessing 10 instances from within the dataset across all runs that were undertaken, while the implementation used in DynamicWEB operated at 97.3% accuracy after 10 instances. This disparity may be accounted for by the difference in testing methodology and variation introduced by the randomness of the instance ordering. Fisher did not state in his 1987 paper how many runs his results were averaged across. The results shown for DynamicWEB were averaged over 100 runs, with the dataset being randomised between each run. Within those 100 runs over half achieved 100 percent predictive accuracy after 10 instances were witnessed, with another 25% only failing to correctly classify one of the remaining unseen instances. The Soybean dataset is a four class problem and as such it is not surprising, over 100 runs, that all of these runs do not correctly describe all four classes perfectly within 10 randomly chosen training instances. Further, within the Soybean dataset there is a bias within the classes. There is not an even distribution of classes as one class is represented by 70% more items than the other three classes. Thus, the initial instances which are used to build the hierarchy have a marked impact on how rapidly the method will learn the domain completely. As each trial is based upon a new randomisation of the dataset, some variation between one run and the next should be expected.

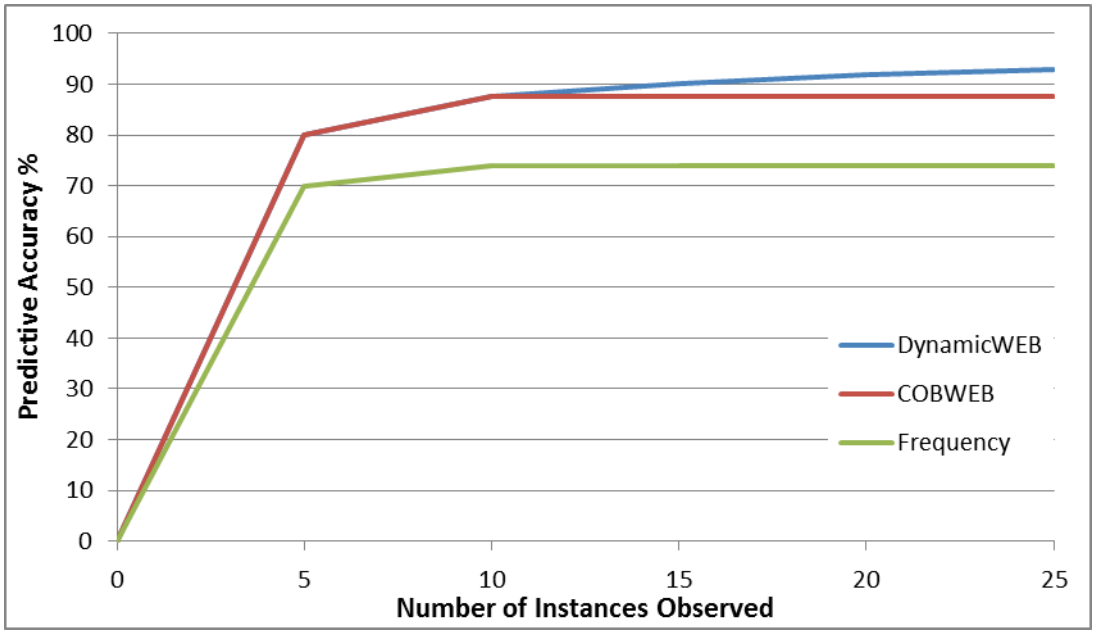


Figure 21. Fishers implementation of COBWEB vs. the DynamicWEB implementation in predicting a missing attribute

Fisher, in demonstrating the effectiveness of COBWEB at learning a domain, extended his trials upon the Soybean dataset beyond that of predicting the resultant class of an instance to also predicting what an attribute value might be when in the presence of the other 35 attributes. Figure 21 shows the performance comparison between COBWEB and DynamicWEB, along with the Frequency-based approach also shown by Fisher. This experiment tests each of the 35 attributes in turn, where they have been removed and then predicted based upon the other attribute value present within the instance. Each of the 35 attributes was run 25 times and were then averaged together to produce the DynamicWEB performance graph shown. Again we do see some variation between the two implementations. The main difference between the two is that the published version of COBWEB converges at 87% predictive accuracy, while the implementation in DynamicWEB reaches 93% after 25 instances while still improving. This could again be due to the reasons outlined above; a result of different numbers of trials being completed and then averaged. The attributes are all nominal with the most discrete values being seven, and with several attributes having only four values. In the context of this, it is quite surprising that the published COBWEB results so rapidly converge after 10 instances. While it's not an impossible scenario, it may be the result of only running the different attributes a single time each, or perhaps only a few times each.

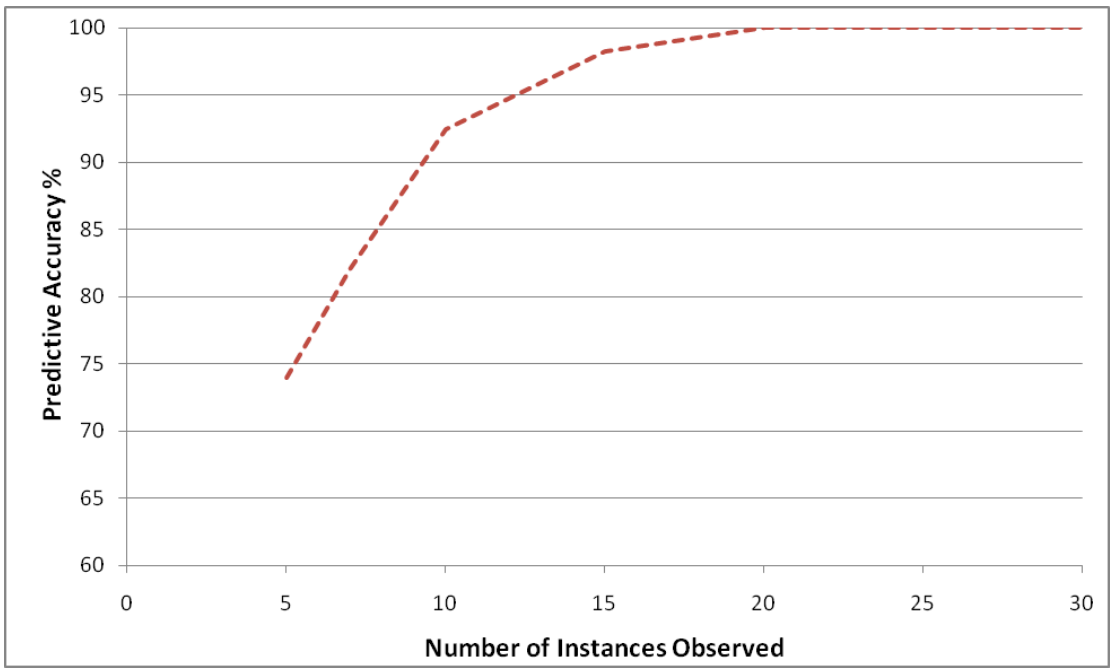


Figure 22. DynamicWEB's implementation of COBWEB operating upon the quadruped animal's dataset.

The version of COBWEB included within DynamicWEB also includes the CLASSIT extension which enables it to operate upon numerical data. Figure 22 shows the DynamicWEB implementation of COBWEB (with CLASSIT) learning a purely numerical domain, the quadruped animals dataset (Gennari, Langley et al. 1989). This dataset was described by Gennari as an example dataset for CLASSIT, although it was not presented in a form illustrating its ability to learn the domain, but was discussed in relation to some other learning techniques not examined here. However the result shown above in Figure 22 does match the expected performance described within the text of the paper. This dataset will be examined further when DynamicWEB's capabilities as an ensemble learner are illustrated in Section 6.4.

In summary, the implementation of COBWEB within DynamicWEB functions effectively upon the datasets shown in Figure 20 and Figure 21. There is some minor variation in the performance when compared with the published results of COBWEB. However these can be accounted for if we consider the effects of ordering upon the structure produced and several unknowns in the testing methodology of Fisher, and more specifically the number of runs over which his results were averaged. The implementation of COBWEB used within DynamicWEB does effectively learn the knowledge domains upon which it is tested, and, furthermore, does this in a comparable way to that published by Fisher (1987). This leads us to conclude that the implementations are suitably equivalent to one another.

6.3 DYNAMIC WEATHER

DynamicWEB differs from COBWEB, and most other machine learning techniques, in that each instance is not the only instance that relates to a target object. The two datasets used above on the COBWEB algorithm, Soybean and Quadrupled Animals, are both examples of datasets in which each instance is the only piece of data that relates to a object. As the bulk of methods operate on problems like this, there is a lack of datasets that contain multiple observations of each object to track over time. As such a new dataset was created to demonstrate and verify DynamicWEB.

The dataset, entitled Dynamic Weather, which was created for this purpose, is based upon the well known Weather dataset by Quinlan (1986). Quinlan's Weather dataset is comprised of fourteen instances each with four attributes and a class value. The

Name	Type	Values
Outlook	Nominal	Sunny, Overcast, Rainy
Temperature	Numeric	Numeric
Humidity	Numeric	Numeric
Windy	Nominal	True, False
Class	Class	Yes, No

Table 24. Listing of the attributes within the Weather dataset

dataset describes weather conditions in which to play golf[7]. E.g. if it is raining and windy then the player does not play, whereas if it is overcast and warm then the player plays. The dataset contains both numeric and nominal values within simple concepts such as temperature, wind and humidity. It is a simple dataset which is easily understood and has been used widely as a research dataset and in machine learning textbooks (Witten and Frank 2000).

For a dataset to be able to demonstrate DynamicWEB it needs to include multiple observations of a target object, monitoring change over time, allowing for an object to drift from one resultant class to another. Extending the Weather dataset to include multiple weather recordings at a range of locations is a simple and elegant concept

Name	1st Measurement	2nd Measurement	3rd Measurement
St Andrews	No	**Yes**	Yes
Gleneagles	Yes	Yes	**No**
Carnoustie	Yes	Yes	Yes
Dornoch	Yes	Yes	Yes
Porthrawl	No	No	No
St Davids	No	No	**Yes**
Nefyn	Yes	Yes	Yes
Pennard	No	No	No
County Down	Yes	**No**	No
Portrush	No	No	No
Bally Bunion	Yes	Yes	Yes
Lahinch	No	**Yes**	**No**

Table 25. The classes at the different points of measurement over the day; object drifts are shown in bold.

[7] The paper does not state that the task being decided upon is golf however Quinlan released files containing the dataset at the time were labeled golf.data and golf.names.

for a sample dataset. Dynamic Weather includes three recordings over the course of the day (suggested to be 9 am, noon and 3 pm) at twelve different golf courses. The twelve courses, listed in Table 25, are named for actual golf courses in Scotland, Wales and Ireland and experience different weather over the course of the day. Four of the golf courses allow for golf to be played all day long, three have no golf played all day; the remaining five migrate from one class to the other over the course of the day. Two of these migrate from being able to play to then not being able to play, another two do the reverse of this, while one object starts off not playing, then playing during the middle of the day, before again not being able to play at the end of the day. The names of the golf course act as unique identifiers within the dataset for each location. When an instance within the dataset is observed by DynamicWEB it is this name that is looked up within the index. If found, then the profile for that identifier is updated with the new information from the newly observed instance.

The class labels were applied to the instances, based upon a modified decision tree (Figure 23) created by running Quinlan's C4.5 (1993) decision tree learner[8] on the original Weather dataset. The tree that was produced by C4.5 only contained the hierarchy with the three outlook nodes, along with the wind and humidity branches. The top level in the tree (shown in bold in Figure 23) was added to ensure that all 4 attributes played an active role over the three measurements within Dynamic Weather, as there are three times as many instances within that dataset as in the original Weather dataset. The value of 50 degrees Fahrenheit was chosen as it was

```
temperature <= 50: no
temperature > 50
|       outlook = sunny
|       |       humidity <= 75: yes
|       |       humidity > 75: no
|       outlook = overcast: yes
|       outlook = rainy
|       |       windy = TRUE: no
|       |       windy = FALSE: yes
```

Figure 23. Modified C4.5 Decision Tree of the Weather Dataset.

[8] The implementation used is contained within the WEKA Data Mining Tool Kit (Hall, Frank et al. 2009) and is actually an implementation of J4.8.

below any value within the original weather dataset and wouldn't therefore produce a different class for any instances within that dataset, while also being a value which represented a fairly realistic scenario.

Dynamic Weather is a complete dataset which allows for DynamicWEB to be tested and verified in several ways. The full dataset can be examined in Appendix A. Initially DynamicWEB will be examined to verify that it updates the current weather over the day at each location. After this, it shall be examined in a more profile building scenario using derived attributes, and then finally the same problem is looked at in the context of multiple DynamicWEB trees.

6.3.1 PERFORMING PROFILE UPDATES WITH DYNAMIC WEATHER

The simplest way to track an object as it changes over time is to update the profile of that object with the most recent observed attributes values for that object. This obviously does not retain the context of the behaviour of the given objects in the past. This will be examined later. Even without this contextual information, it is still worthwhile to have profiles that illustrate the current state of the objects of interest, and in the case of Dynamic Weather the current conditions at 12 golf courses.

Over the course of the fictitious day that is represented within the Dynamic Weather dataset there are three instances for each target object. The first instance will establish the profile within the concept hierarchy, while the other two will each update the attributes values within the profile to the current values as the day progresses. Before we look at the concept hierarchy produced by having all twelve objects added and updated over the day, a smaller group of just four objects will be examined. This will verify that the changes are occurring as they should, and

Location	Outlook	Temperature	Humidity	Wind	Play
Porthrawl	rainy	45	80	TRUE	No
Bally Bunion	overcast	65	62	TRUE	Yes
County Down	sunny	55	80	FALSE	Yes
Nefyn	sunny	72	62	FALSE	yes
County Down	overcast	50	80	TRUE	No

Table 26. Five instances from the Dynamic Weather dataset. The 5th instance is an instance used to update the 3rd instance.

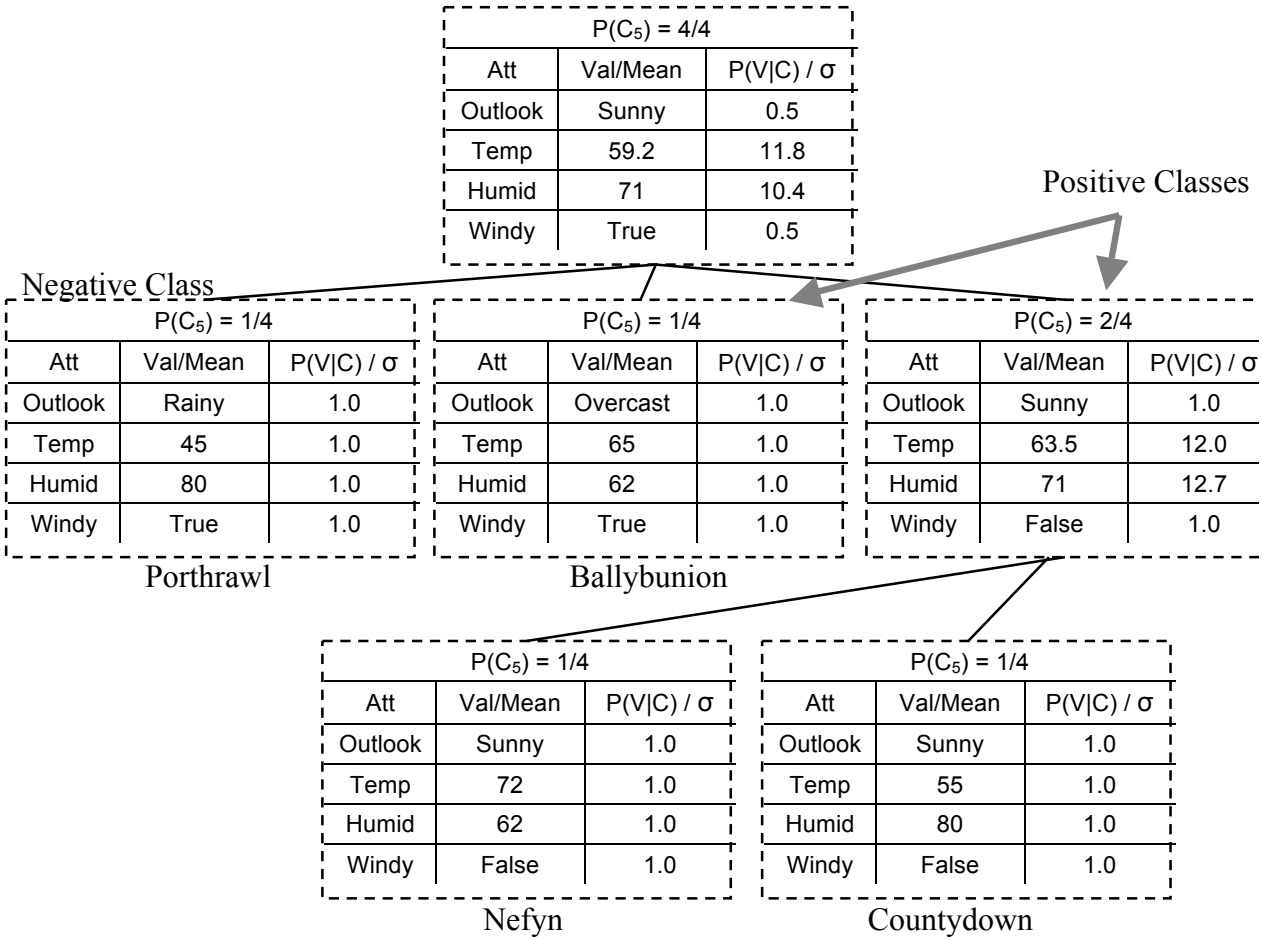


Figure 24. The concept hierarchy produced from the first 4 instances shown within Table 26.

examine how they are represented within the concept hierarchy. The four objects being considered are Nefyn, Porthrawl, Bally Bunion and County Down. For each of these objects there is an instance shown in Table 26. There is a single instance for each of the four objects, and then a fifth instance for the Countydown object which drifts from the positive class to the negative. Each of the other three objects are stable, being resident within a single class across the three observations within the full dataset. Having three stable objects with only a single one drifting to a different class was chosen for ease of understanding, and we will not examine the other instances at length here. The five instances in Table 26 are transformed by DynamicWEB into the two hierarchies shown in Figure 24 and Figure 26. The first of these illustrates how the hierarchy appears after the four objects have been observed once. County Down is shown within the positive class, within a node with Nefyn. Bally Bunion is not clustered within the same node as these two, seemingly because of the difference in wind and outlook. At this point the tree is exactly the same as that which would be created using COBWEB.

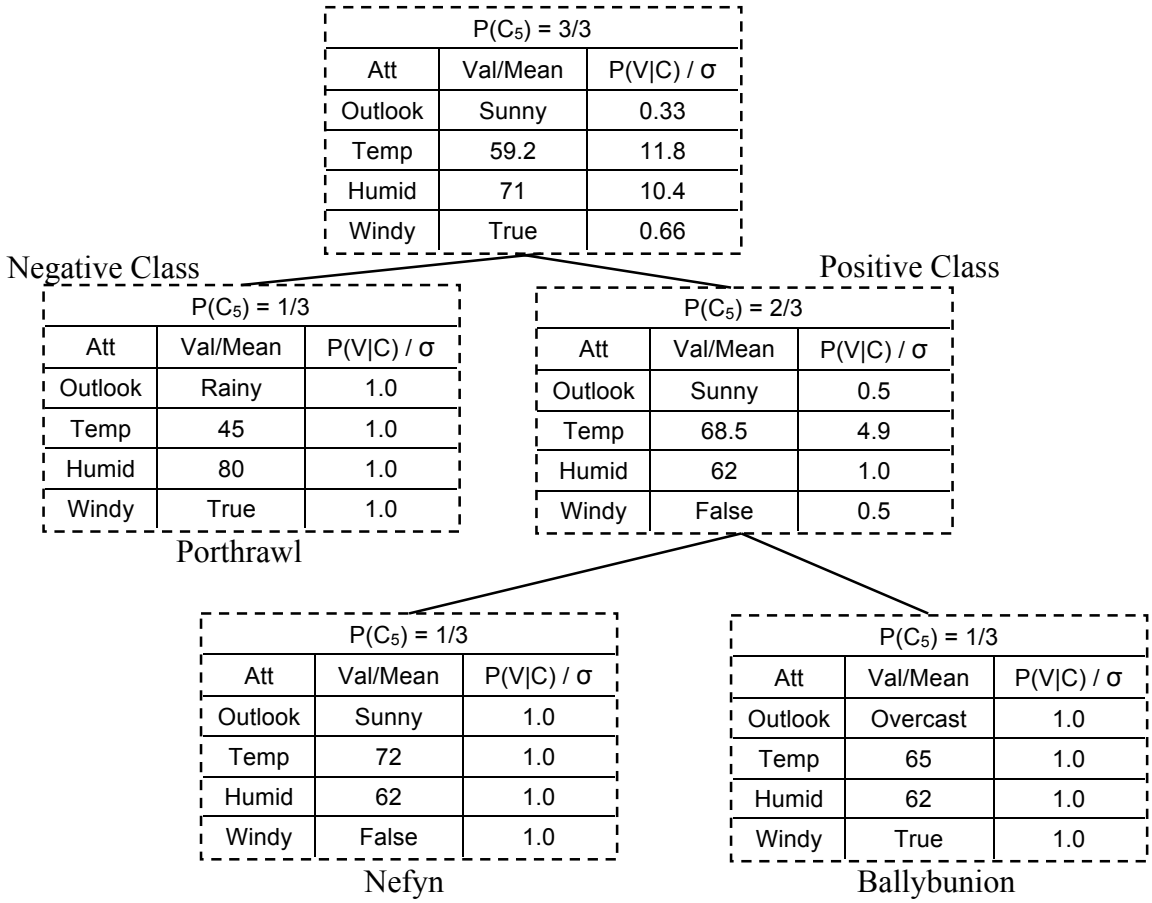


Figure 25. The hierarchy which occurs between the removal of County Down profile and its readmission after it has been updated.

After the initial four instances are incorporated, the fifth instance is then observed and it is at this point that DynamicWEB takes over from COBWEB. The fifth instance is the second observation that relates to the County Down object. The result of this is that the County Down profile needs to be updated with the new information. The first stage of this process is to remove the existing profile from the hierarchy, adjusting the tree as needed to ensure that the best possible hierarchy remains. Figure 25 is the hierarchy that results from removing the County Down profile. The main difference between the hierarchies shown in Figure 24 and Figure 25 is the merging of the Nefyn and Bally Bunion nodes. They are still represented by their own leaves within the node, and not by the same leaf. Thus the removal of County Down did not result in three nodes each with one instance at the level below the root, but in a binary structure which learnt the class boundaries. This merge operation produced a superior tree to that which would have resulted if the concepts were left as they were after the removal of the County Down profile. DynamicWEB aims to always produce the best possible tree it can at any given moment, even part

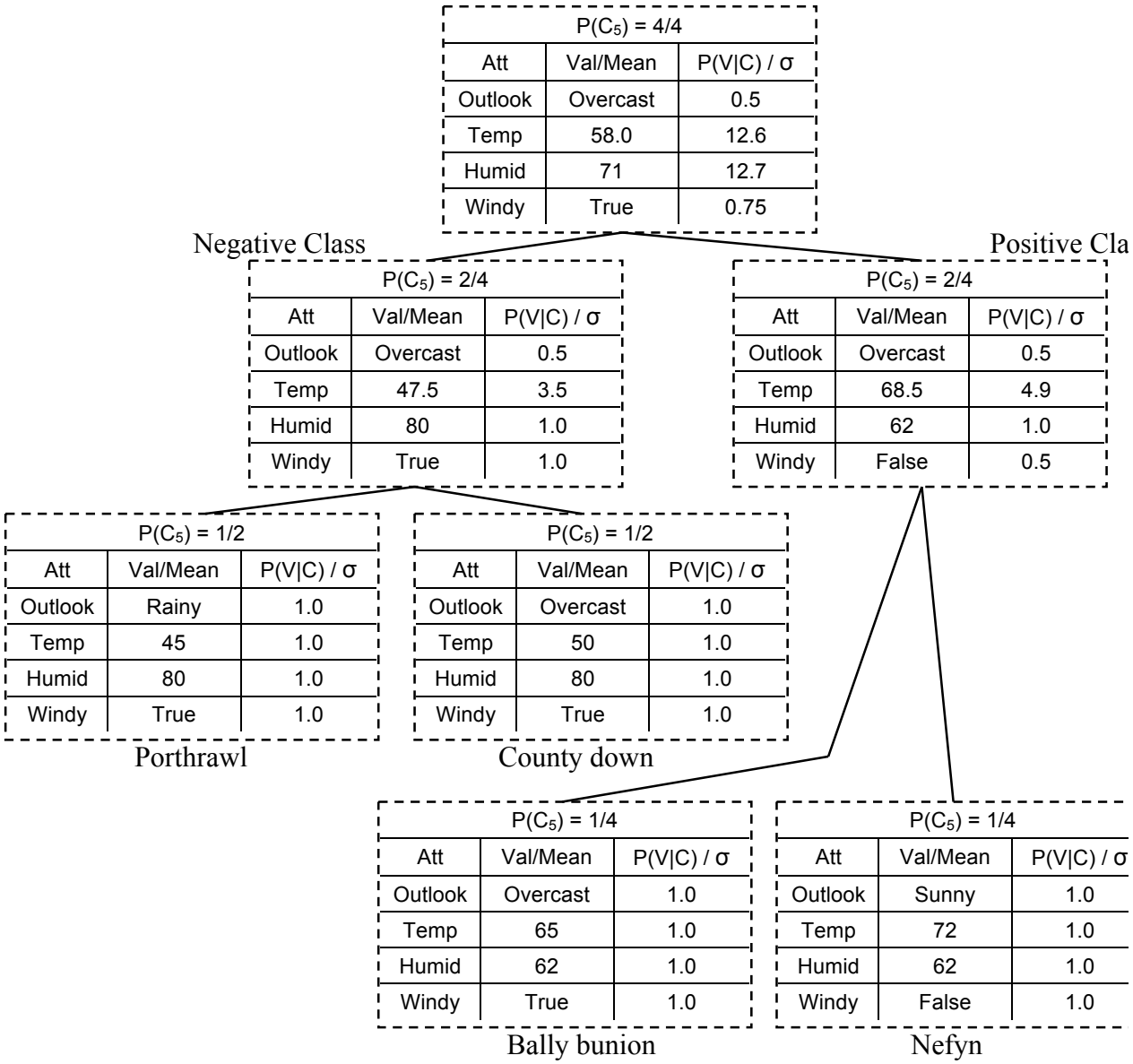


Figure 26. The concept hierarchy after the 3rd instance has been updated with the new data contained within the 5th instance.

way through an update operation.

Ensuring that this is the case during the update operation allows the reintegration of the profile to occur within a scenario that is not too biased towards the former existence of the profile within the concept hierarchy prior to the update operation. Figure 26 shows the resultant categories for the dataset once the County Down profile has been updated with the data from the 5th instance. Its class value has now migrated to the negative due to the change in its weather conditions (which can be viewed in detail within Table 26). The tree structure within Figure 26 is very similar to that within Figure 25 with the only change being the integration of the County Down profile into a cluster that contains Porthrawl, instead of Porthrawl being a

child of the root. The overall structure of the concept hierarchy remains the same with a binary division into the two classes directly from the root, but with each node now having two leaves in each; one for each object. If the remainder of the instances within the Dynamic Weather dataset for these 4 objects are run upon this tree, the structure that is produced, after all three updates to all four objects, is the same structure as that shown within Figure 26. This represents correct behaviour as there are no further changes to the classes of these four objects.

6.3.2 TESTING THE COMPLETE DYNAMIC WEATHER DATASET

The demonstration in the previous section demonstrated, in detail, what occurs within the DynamicWEB hierarchy when an update to a profile occurs. In this section we will extend this to monitoring what occurs at a higher level during an entire run of the Dynamic Weather dataset. Instead of only four objects being examined, this time the full set of twelve, shown in Table 25, will be observed. Across the three updates of each of the twelve instances there are six object drifts that produce a change in the resultant class of the object. Within all of the trees shown in the figures below, each leaf node is pure in class; the highest cutoff value at which this occurs is 0.09.

The first hierarchy, Figure 27, illustrates the structure that is produced after each of the twelve objects have been observed once. The structure is quite broadly spread, largely due to the order of the instances. This portion of the process is identical to

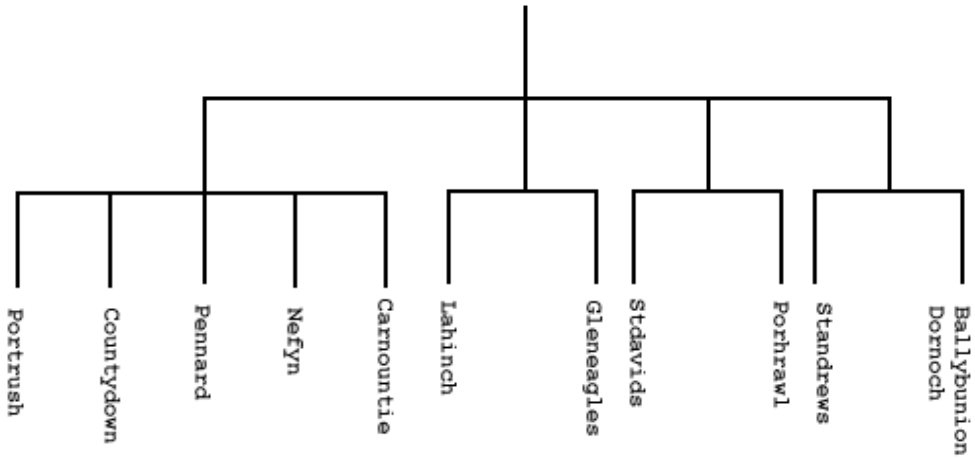


Figure 27. The concept hierarchy after one instance of each object has been observed.

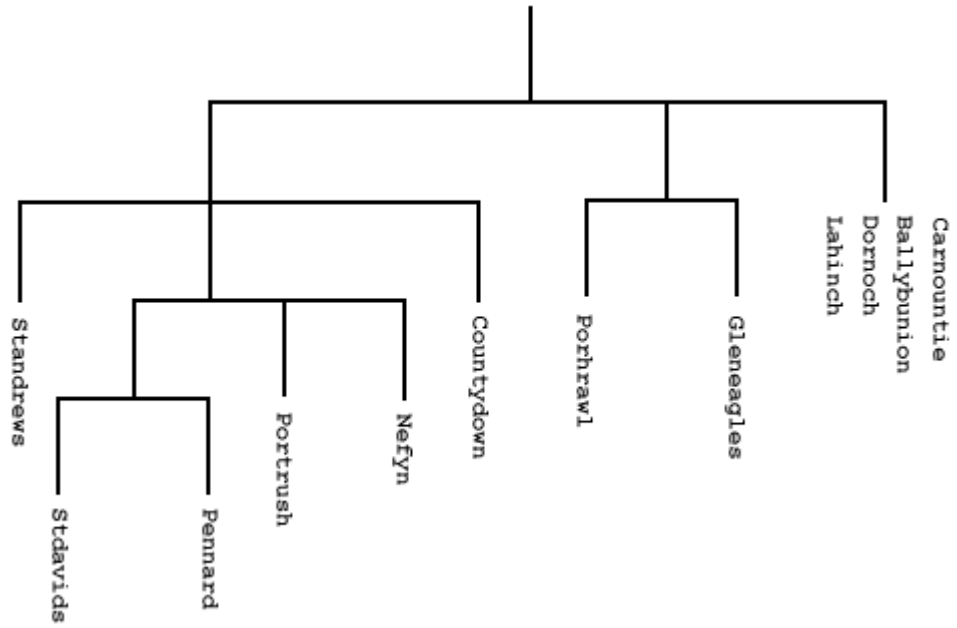


Figure 28. The concept hierarchy after each of the objects have had another observation and have all been updated once.

COBWEB and as such can suffer from its order dependency. DynamicWEB, through its updating, is able to largely nullify this as it examines order dependent datasets through multiple updates of each profile. As each profile gets updated, and in that process removed and re-added to the hierarchy, the structure optimises itself by always choosing the best option. This allows the re-addition of a profile to a hierarchy to build upon the knowledge that has been added to the tree since it was previously inserted into the tree. The result of the first twelve updates to the hierarchy is shown in Figure 28. The structure is now not as flat as in Figure 27 with much more depth present, and with some closer matching of sister leaves of the same class. Also four profiles of the positive class are present within the right furthermost leaf node.

Within the updates that occurred between the points in time represented by the two structures above, three objects have drifted from one resultant class to another. The objects for *Lahinch* and *St Andrews* have both gone from the negative class to the positive, while *County Down* drifted from the positive class to the negative. The locations where these profiles are now stored within the tree are with profiles of the same resultant class. This shows DynamicWEB successfully allowing for the object drift and clustering the updated profiles in the correct location. The second round of updates from the third set of observations produced the hierarchy shown in Figure 29. This hierarchy is even simpler in structure than that in Figure 28, and now has

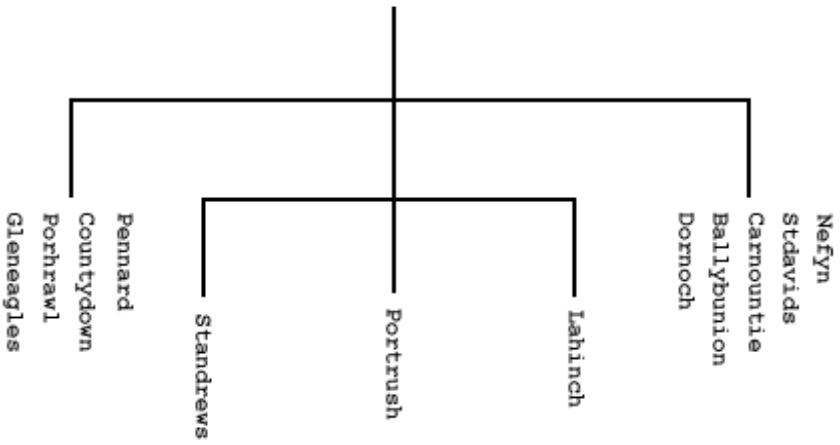


Figure 29. The concept hierarchy after the second round of updates have been completed.

two leaf nodes, with four or more profiles resident within them. Again, during this round of observations there were three objects which drifted from one concept to another, with *Lahinch* again drifting, to the negative class along with *Gleneagles*; while *St Davids* drifted to the positive class.

The tree structures that were shown within the figures above are quite useful in terms of visualising the knowledge structure produced. They are easily human readable, and do perfectly describe the structure at an exact moment within the timeframe being examined in the dataset. However, their main shortfall is a lack of being able to visually show what drifting has occurred over time. Through comparing two trees it is possible to garner that *St Davids* was clustered very near to *Dornoch* in the update between trees two and three. But knowing how much they changed across all three tree structures or viewing those two objects in comparison to the other objects is not readily apparent. In an effort to illustrate the drifting of objects in relation to one another, Figure 30 displays a category measure for each of the objects within the dataset across the three measurements. These measures allow for direct comparison between the different objects across the time period of the dataset, illustrating the changing similarity of the objects. The measure used is a modified category utility of a singleton node that only contains two object profiles. These profiles are the same as those used within the DynamicWEB knowledge hierarchy and as such the comparison that takes place has a direct link to the current state of the hierarchy. The measure taken compares each object within the dataset to a single object within the dataset, producing a similarity measure between the two

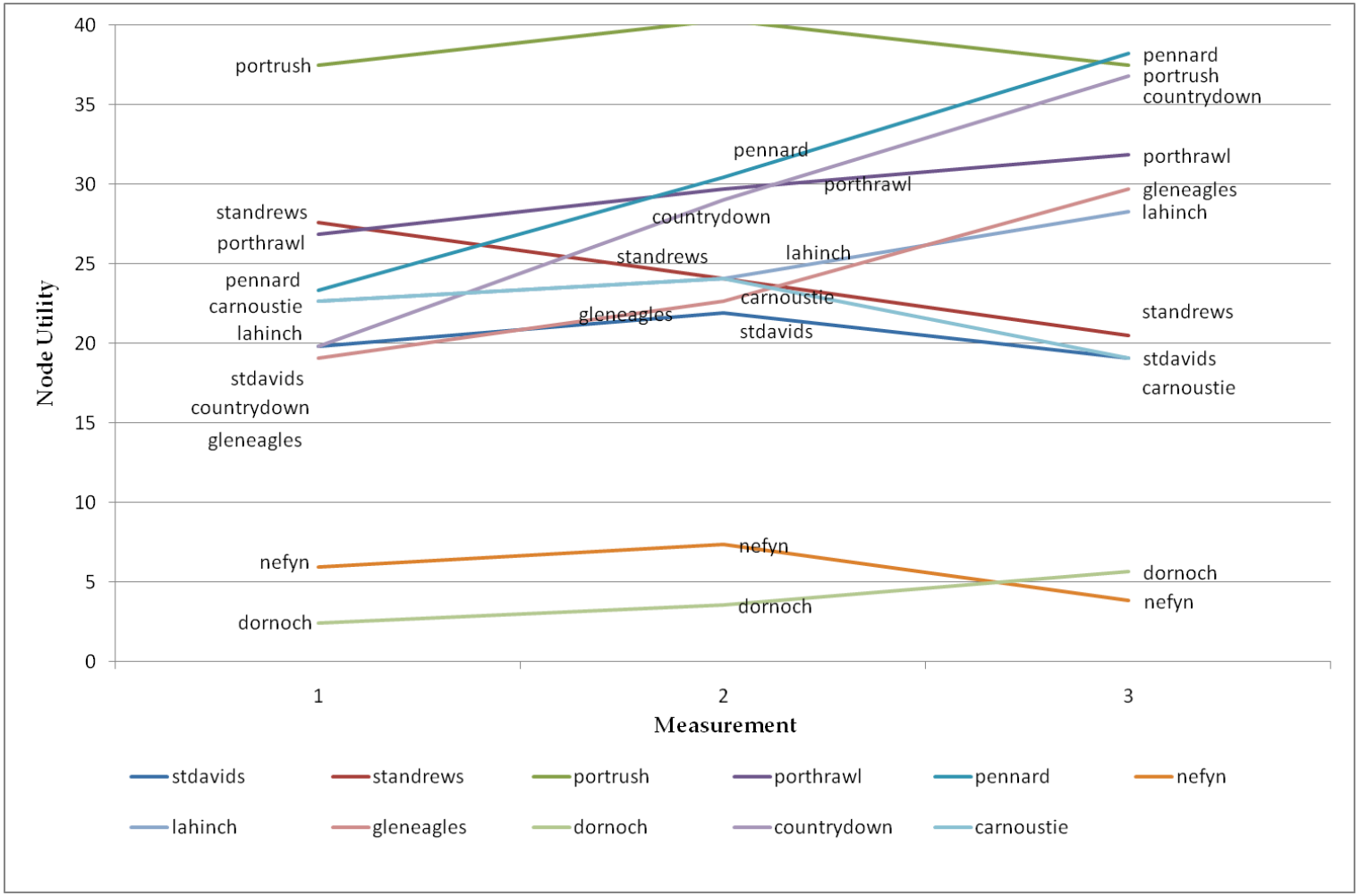


Figure 30. Tracking the objects overtime in relation to the other objects within the dataset

objects. Figure 30 is built upon comparing all of the object profiles to the *Bally Bunion* profile. The value of the Y-axis is an un-normalised category utility calculated based upon the two profiles.

In computing this measure as a direct comparison between two objects, we areignoring the existing knowledge within the hierarchy, as it would bias this measure, meaning that it was not a pure direct comparison between the profiles as they currently stand. As we are not using the statistics within the hierarchy, it is only numeric attributes that are used within comparison. If we were to include the nominal attributes within the context of a one to one comparison, they would carry too much extra weight within the similarity measure to produce a useful metric here. This is a short-coming. As such it is shown here purely for a visualisation aid to indicate what is taking place within the Dynamic Weather dataset.

6.3.3 PERFORMING UPDATES WITH DERIVED ATTRIBUTES

In addition to the ability to update the profiles within the DynamicWEB hierarchy to contain the most recently observed value, there is also the ability for DynamicWEB to make use of derived attributes. Derived attributes are statistical representations of the current and previously observed attribute-value pairs of an object. These derived attributes allow the profiles to store the context of past observations of an object, and then to contribute this to the clustering process. The derived attributes were described in more depth within Chapter 5 (5.4.4). Most of the derived attributes are derived from attributes with numerical values. There are two numerical attributes within the Dynamic Weather dataset: Humidity and Temperature. Of the derived attributes, the two most generally recognised and easily understood statistical measures are mean and standard deviation, so these are used here to demonstrate the effect of derived attributes.

A DynamicWEB tree was created that only observed the average and standard deviation of two actual attributes Temperature and Humidity. The resulting tree, consisting of four profiles with four attributes in each, is shown in Figure 31. The structure shown here is the same as that shown in Figure 26, although to achieve this a category utility of half (0.045) that which was previously used (0.09) was needed. This drop in the knowledge threshold was required to produce the correct hierarchy as the other attributes were ignored in the learning process, therefore leaving the

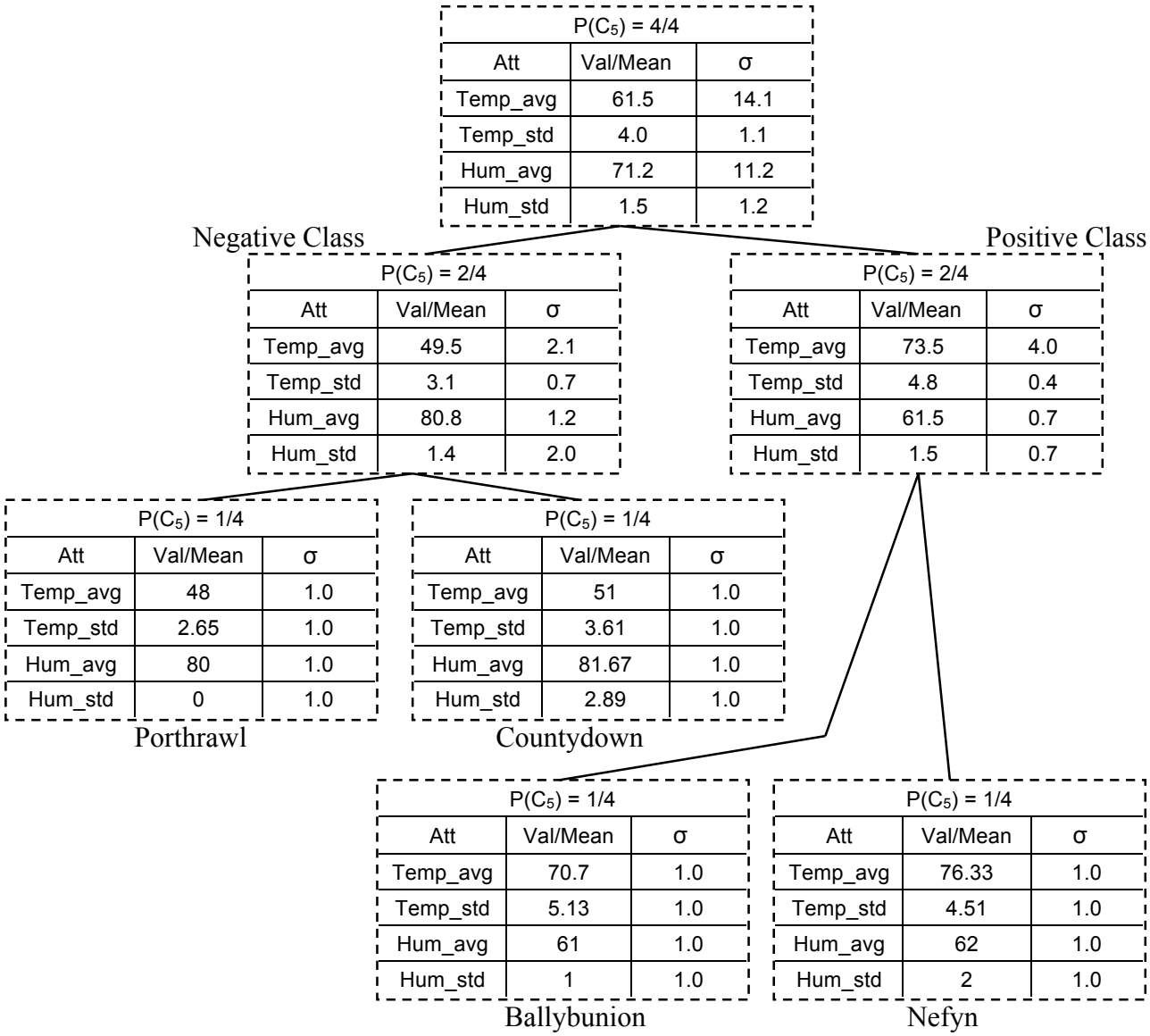


Figure 31. The concept hierarchy after all 12 instances of the 4 object subset of Dynamic Weather have been incorporated.

domain less well defined. This change in threshold increases the computational overhead of the method to run upon the dataset. To take a single object as an example, the temperature and humidity values recorded for Nefyn were 72, 81, 76 and 62, 60, 64 respectively. The derived result for average and standard deviation for the two attributes within the profile were then 76.33, 4.51 and 62, 2 respectively.

Obviously within a dataset only containing three samples of each object, such measures have limited value; however this does clearly demonstrate the way in which DynamicWEB works with derived attributes. Using the full Dynamic Weather dataset with all attributes and instances, in conjunction with these four derived values, a tree of the same level of correctness to that shown within Figure 29 can be

achieved with a slightly increased knowledge threshold (0.09 to 0.1). While again of little impact on a dataset of this size, improvements like this are of more benefit in larger datasets, as will be seen in the two chapters that follow this one.

6.4 MULTIPLE DYNAMICWEB TREES

In addition to the derived attributes, another extension which has been used with the DynamicWEB learner described in this thesis is the usage of multiple concept hierarchies simultaneously upon a single dataset. The goal of this is to split the learning problem into multiple portions and for each to be clustered in a hierarchy separately. When an instance of unknown class is then to be classified, it is classified by each hierarchy in the forest, producing a set of possible classifications. This is undertaken for two main reasons. The first is to reduce the complexity of the problem that is being examined within a single hierarchy, allowing for more subtle patterns to be discovered. The second is that it allows for concept hierarchies to be developed from a dataset based on different attributes, and then compare the impact of the various attributes upon the results or structure of the concept hierarchies. In addition

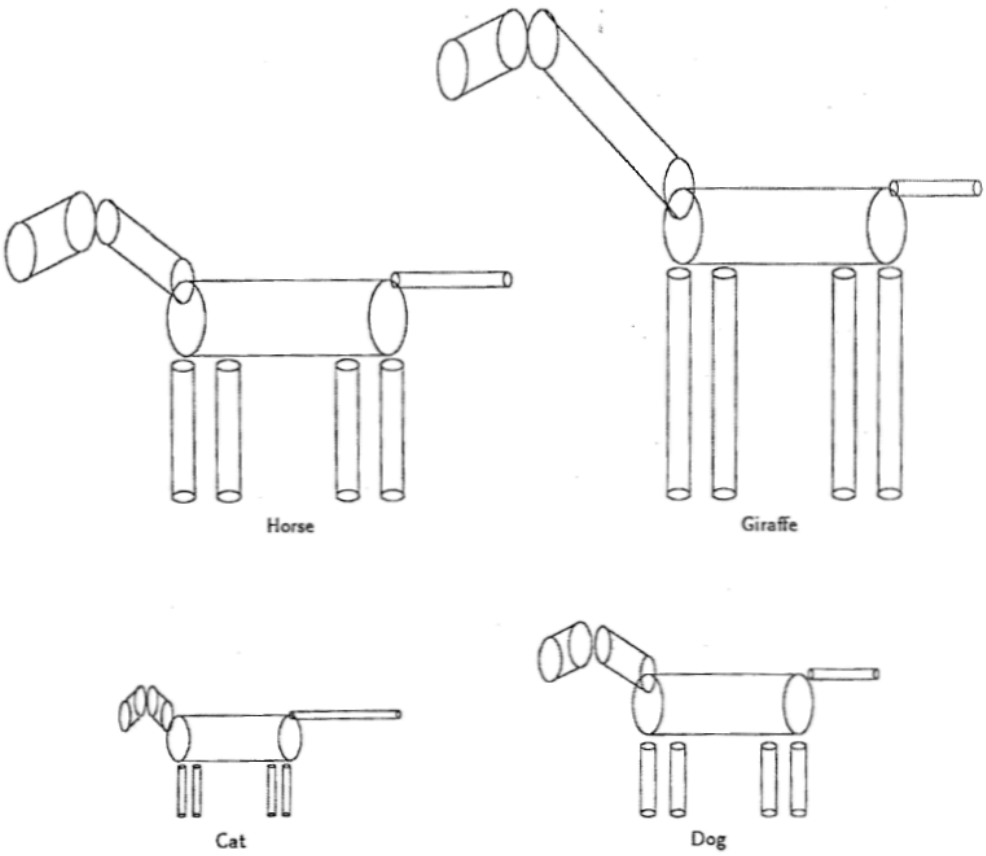


Figure 32. The cylinder representations of the Quadruped Animals present within the dataset.

to this, attributes which conflict can be utilised by one tree, but not another, to minimise those negatives effects. Overall, from a classification standpoint, it allows for datasets with many attributes, that may interact in subgroups, to be treated appropriately, then allowing for the knowledge of the forest to be utilised instead of that of a single tree. This is similar to previous techniques, such as Bagging (Breiman 1996) and Boosting (Schapire 2002), mentioned in the previous chapter.

Gennari et al (1989) described the Quadruped Animals dataset when demonstrating the CLASSIT extension to COBWEB. It contains over 500 instances, each describing one of the four animals shown in Figure 32. These animals are described within the dataset by 72 attributes. These attributes can be broken down into 8 groups, each containing 9 attributes. The eight groups all describe the following different parts of the animals: Neck, Head, Torso, Tail and the 4 legs. The attributes define the part of the animal based on its height, width, location, radius, and texture.

Using the DynamicWEB implementation of COBWEB (with a cut-off threshold of 0.02) to examine this dataset, the predictive accuracy, averaged over 100 runs, of the dataset is shown in Figure 33. This testing was completed by randomising the dataset, incorporating several instances, and then testing the hierarchy produced on the unseen instances from the dataset. This was completed 100 times with the dataset

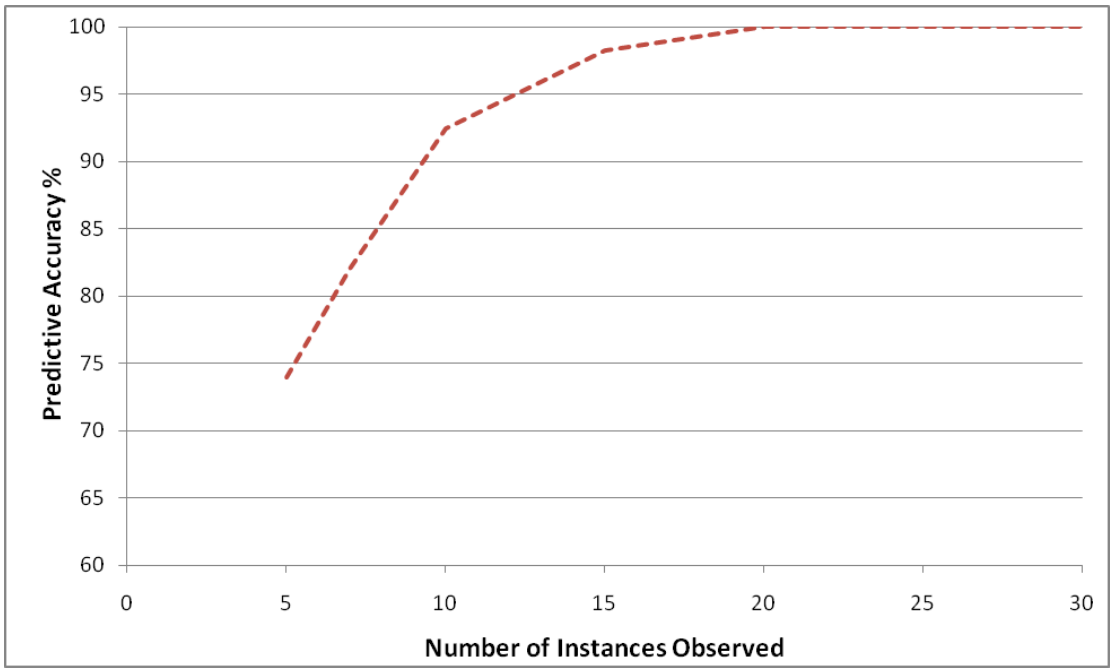


Figure 33. Predictive accuracy of the Quadruped Animals dataset in a single concept hierarchy.

ordering being randomised before each trial and the result for all the trials being averaged across all runs. This establishes the baseline performance of the Quadruped Animals dataset within a single hierarchy.

The Quadruped Animals dataset was then run using a forest of 8 hierarchies, one for each body part, the results of which are shown in Figure 34. Within this graph there is a very tight grouping in the performance of 5 of the hierarchies: the four legs and the head. These 5 all perform better as separate structures than if they are combined into a single tree structure. However, three of the other hierarchies (Neck, Tail and Torso) perform markedly worse than their counterparts or the single tree structure. As the data for these hierarchies is used within the single structure, it is likely that they are having a negative effect on the predictiveness of the structure produced. The hierarchy based on the Neck has a dip in accuracy at 25 instances observed due to it frequently restructuring the tree at that time.

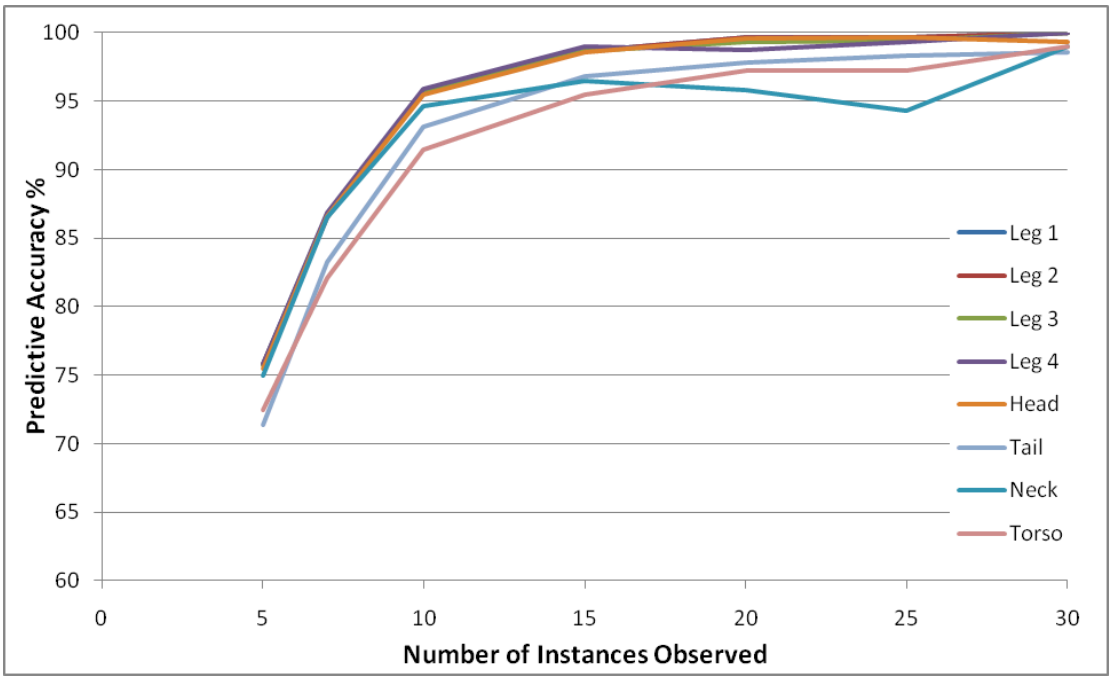


Figure 34. The predictive performance of the hierarchies that were each formed on the data of a different body part.

Each of the trees within the forest is built using the instances in the dataset that relate to the body part they are tracking. As each structure that is produced is different they all produce their own classification for each animal instance. Using these predicted values, an overall classification, for a given instance, can be made. Each tree is treated equally in deciding the classification of an instance (although weighting some

trees more highly if they are shown to be more predictive is an obvious extension to this). The result is then decided by a majority vote of the trees (greater than 4 votes in agreement in this case). If no majority is established then an incorrect prediction is recorded. The result is shown in Figure 35 and compared with the prediction performance of the single tree as well as the best of the trees within the group of 5 that were shown to perform well in Figure 34. There is a significant difference between the voted forest and the single hierarchy, with the former exhibiting a 2-4% increase in predictive accuracy over the first 20 observations. The performance of the voted forest is lower than some of the individual trees in the forest (such as the group of five) prior to 20 because of a lack of sufficient agreement between the trees to gain a majority. However, when guided by this simple rule, the performance was still greater than that of a single tree.

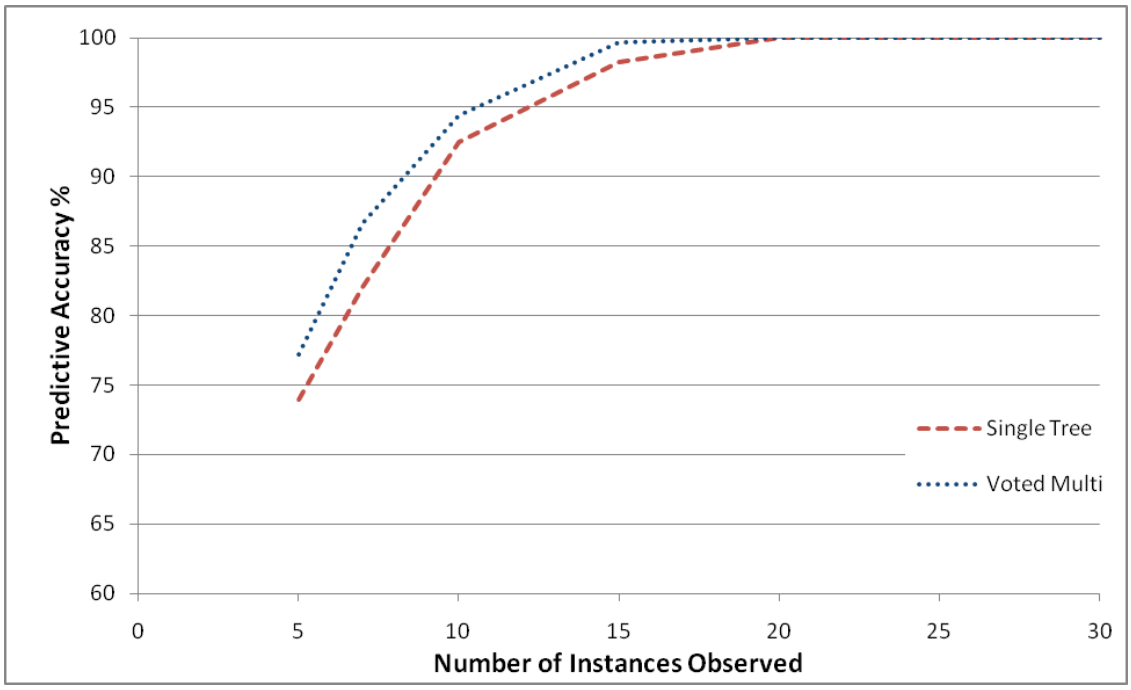


Figure 35. Performance comparison between the single COBWEB tree and the voted multiple tree configuration.

When examining the results discussed above one can see that a large portion of the improvement in performance of the forest over the single tree is related to the fact that there are 4 legs which all perform very well, while all obviously being very similar to each other and thus usually voting together. Whichever class the 4 legs agree on, only one other part of the animal with the same resultant class was required for a majority. While this does act to simplify the dataset, this correlation within the dataset is likely to be discovered within other learners (such as the single hierarchy),

and only highlights that certain datasets are well suited to such learning methods.

6.5 STAGGER CONCEPTS DATASET

In Chapter 3 a supervised learner called STAGGER was examined. This learner was one of the first learners cap ble of adapting to concept drift, when publishing this method (Schlimmer and Granger 1986) also described a dataset that contained two sudden concept changes, resulting in three distinct concept descriptions over the course of the dataset. The instances within the dataset were made up of three attributes: size (small, medium, large), colour (red, green, blue) and shape (square, circular, triangular). The dataset posed a two class classification problem, with the positive class being that which changed across the three concepts. The three goal concept descriptions that exist within the dataset are shown within Table 27. Concept one is the positive class at the start of the dataset and then changes drastically to become concept two after a period, and then concept three takes over for the last portion. All instances which don't match the current given goal concept are members of the negative class. The STAGGER Concepts dataset aims to test how fast a learner can adjust to a sudden drift in the target concepts.

Concept One	size = small and colour = red
Concept Two	colour = green or shape = circular
Concept Three	size = large or medium

Table 27. The three concepts which are present in the STAGGER concepts dataset.

Figure 36 illustrates how the STAGGER method operated upon this dataset. The classification accuracy of the method dropped to half each time the concept changed, but it then recovered and learned the new goal state over the following 30 instances. Schlimmer and Granger did not fully describe their testing methodology within the paper. However, it is assumed that they tested the learning structure upon a different set of instances from those used to build the structure, and tested after each new instance from the dataset was incorporated into the learner.

Discussed shortly after STAGGER in Chapter 3 was another method called FLORA (Widmer and Kubat 1996). This method is capable of adapting to concept drift, and, like COBBIT (Kilander and Jansson 1993), operates upon a time-window-based approach. However, similar to STAGGER but not the COBWEB-based approaches,

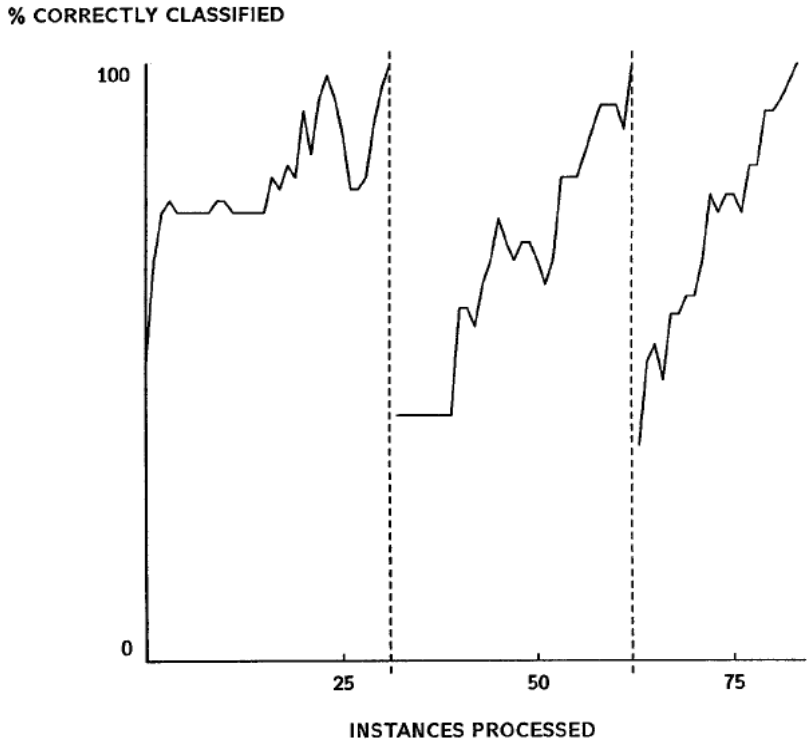


Figure 36. The learning response to concept drift of STAGGER upon the STAGGER Concepts dataset (Schlimmer and Granger 1986).

it is a supervised approach. Both of these supervised approaches use the class label information to reinforce the knowledge growth within the tree, allowing them to adapt to the change faster than they would have been able to without it. Figure 12 illustrates how the three most advanced versions of the FLORA algorithm adapt to the STAGGER concepts dataset. Unlike the results shown in

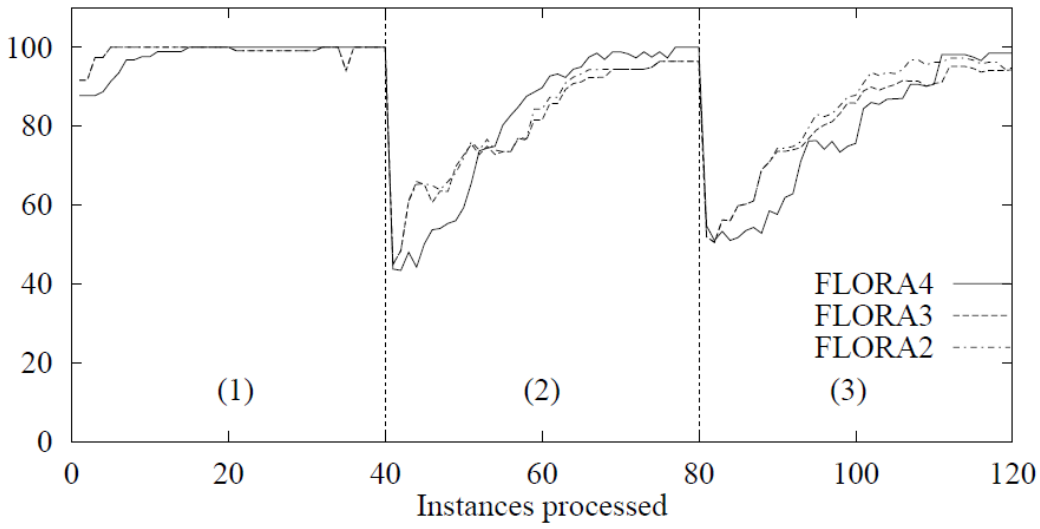


Figure 37. The learning response to concept drift of FLORA upon the STAGGER Concepts dataset (Widmer and Kubat 1996).

Figure 9 the dataset is slightly larger this time while still being generated upon the same concepts. Each concept lasts 40 instances, with 120 instances being observed in total. Separate from these 120 instances, there are 100 test instances of each of the three concepts. After each of the 120 instances is incorporated, 100 instances of the current class are then trialled against the method. As expected, similar to situation with STAGGER, the predictive accuracy of the method drops dramatically after the concept drifts suddenly. The three variations of the method then relearn the problem space achieving almost perfect prediction towards the ends of each of the sets of 40.

DynamicWEB is quite a different learning method to the two that are shown above, and also markedly different from COBBIT, which will be discussed below. DynamicWEB is an unsupervised approach which differentiates it from the two listed above. It also does not operate using a time window as COBBIT and FLORA do. Further, it operates upon profiles, updating items that are being re-sampled. As such it doesn't truly fit the mould for the type of problem demonstrated in the STAGGER concepts dataset, for the 120 instances are meant to stand independently of each other. To overcome this, and to allow the reaction time of DynamicWEB to be compared against the others, an identifier was added to the dataset. The identifier is a number

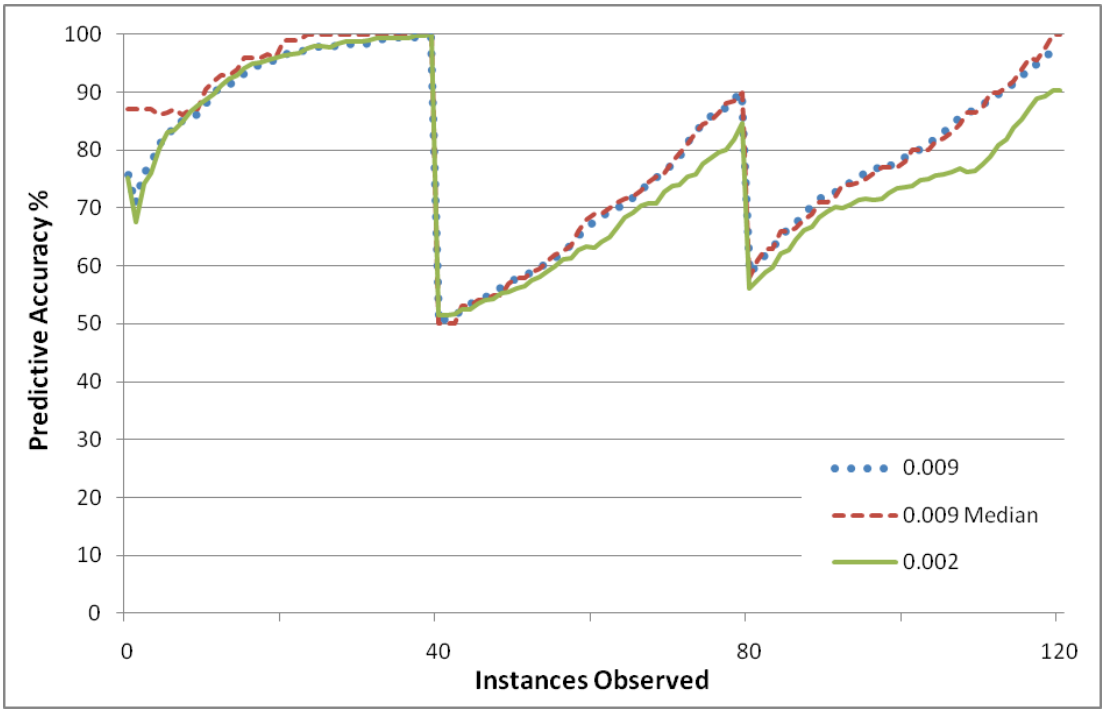


Figure 38. The learning response to concept drift of DynamicWEB upon the STAGGER Concepts dataset averaged across 100 runs

between 1 and 40, which was repeated once in each of the three concept groups. The orders were randomised so that the identifiers are not sequential or in the same order in each segment. After each instance within the dataset had been assimilated into the concept hierarchy, a set of 100 test instances for the current goal concept was run. The predictive accuracies, averaged over 100 runs of the test datasets, are shown in Figure 38. The learning result shown in Figure 38 is comparable to those shown above in relation to STAGGER and FLORA. The speed of recovery from the sudden concept drift is slower in DynamicWEB than shown within FLORA; however this is largely due to the unsupervised nature of DynamicWEB. DynamicWEB's adjustment to the concept drift as a result of the updating of knowledge, not from a supervised reaction that forces the removal of knowledge. DynamicWEB performs least favourably on the second concept. However STAGGER also failed to perform in this case, along with all but one of the FLORA models. The one FLORA model that did achieve this, did so near the end of the concept, with this concept being the slowest of the three concepts that it learned. As we will be examining the performance of DynamicWEB on this dataset in a single run shortly, also included in Figure 38 is median of the trial, to compare with the average. DynamicWEB (using a 0.009 category utility threshold) performed quite well on most random orders presented on this problem. Further these two measures show that the single ordering used below is fairly typical.

The fourth method that we will examine here in relation to the STAGGER concepts dataset is the other COBWEB-based method, COBBIT. The same ordering of data used with DynamicWEB (Appendix B) was also used with COBBIT. Using a single order, instead of multiple runs, allows a direct comparison between the two similar methods' performance on the same ordering of data and allowing that order to play a contextual role. The version of COBBIT being used here (built upon the same version of COBWEB implemented for DynamicWEB) is a purely windowed, unsupervised version. As discussed earlier in Chapter 4 COBBIT did have a supervised addition that dynamically shortened the window size but that was not used here. Instead a purely unsupervised comparison was chosen. The datasets which were examined within the Kilander and Jansson (1993) publication of COBBIT examined single and multiple concept drifts, with window sizes from 16% of the size of the dataset up to the complete dataset. Within the single drift scenario, in a dataset

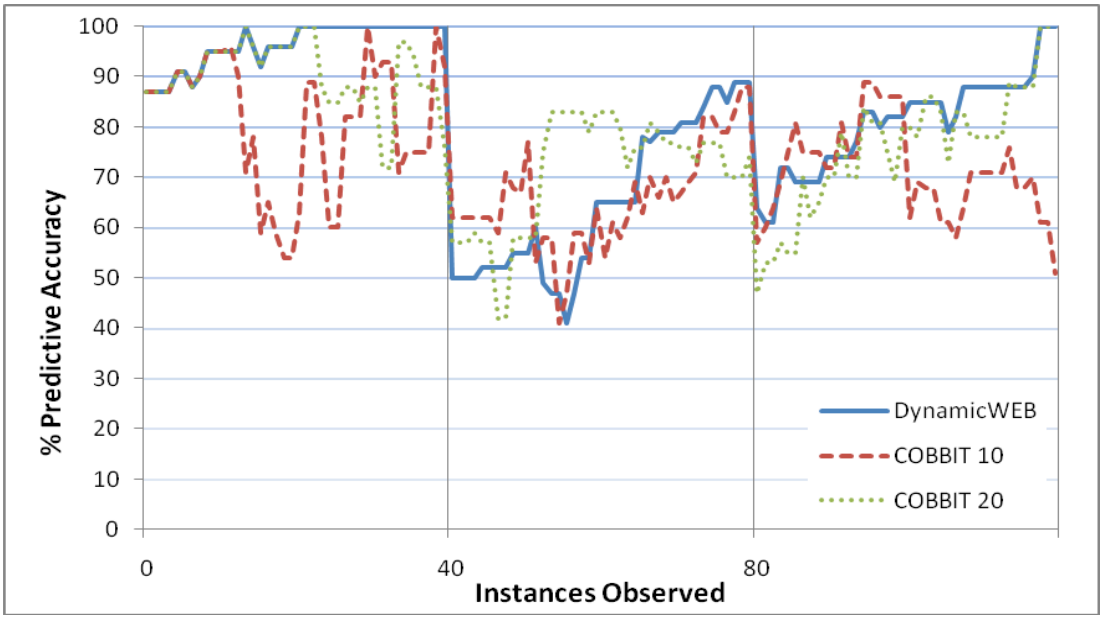


Figure 39. The learning response of DynamicWEB and COBBIT with the two window sizes of 10 and 20 upon the STAGGER Concepts dataset.

of 30 instances, the best performance was found to be with window sizes of 10 and 15 instances. To translate this to window sizes to trial on the STAGGER concepts dataset, window sizes of 10 and 20 were tested and are shown in within Figure 39. These are similar sizes to those trialled by Kilander and Jansson, especially the quite small value of 10. This value, as seen in Figure 39, reacts quite quickly when drift occurs, being the fastest to improve the predictiveness of the model to above 75% on both occasions. However, 10 instances is not enough to completely describe the STAGGER concepts dataset and even before drift occurs it suffers from an inability to fully describe the problem in such a small window. This sized window never predicted at 100% accuracy on the second or third concept. The window size of 20 overall performs much better than the size of 10, eventually reaching 100% accuracy on the 3rd concept after both drifts have taken place. It, like DynamicWEB, fails to fully master the second concept, although the window size of 20 was predicting at 20-40% more accuracy for about a quarter of the second concept. Upon examination, this was seen to be a peculiarity of the ordering which happened suit a window size of 20. However as can be noted, this did not remain the case, with DynamicWEB overtaking in a steady learning curve. Within the third concept however the window size of 20 was close behind in mirroring the predictive accuracy of DynamicWEB.

Both of the window sizes tested above were smaller than the size of each of the concepts, while in (Kilander and Jansson 1993) tested sizes that were larger than the

concept, even up to the size of the whole dataset. In Figure 40 window sizes of 40 and 60 were trialled upon the STAGGER concepts dataset. The window size of 40 effectively matches what DynamicWEB is doing by having 40 profiles that are updated, effectively removing the old knowledge irrespective of the point in the first concept at which it was added (compared to the first-in first-out approach used in COBBIT). The variation shown in the graph is purely the variation due to this ordering. However, the item of note within this graph is that, when you increase the window to a size larger than the concept size, the method's predictiveness suffers greatly. This is not a surprising result, but when combined with the poor performance of the small window above, it does highlight the importance of having the correct window size for the dataset being examined. This parameter tuning is not required in DynamicWEB, and it is in that regard closer to the parameter free model that was in

the original COBWEB. Within a scenario where objects are sampled multiple times and there are identifiers for the individual objects being sampled, being able to leverage these identifiers alleviates the need to tunes parameters to achieve the best performance within a changing stream of data.

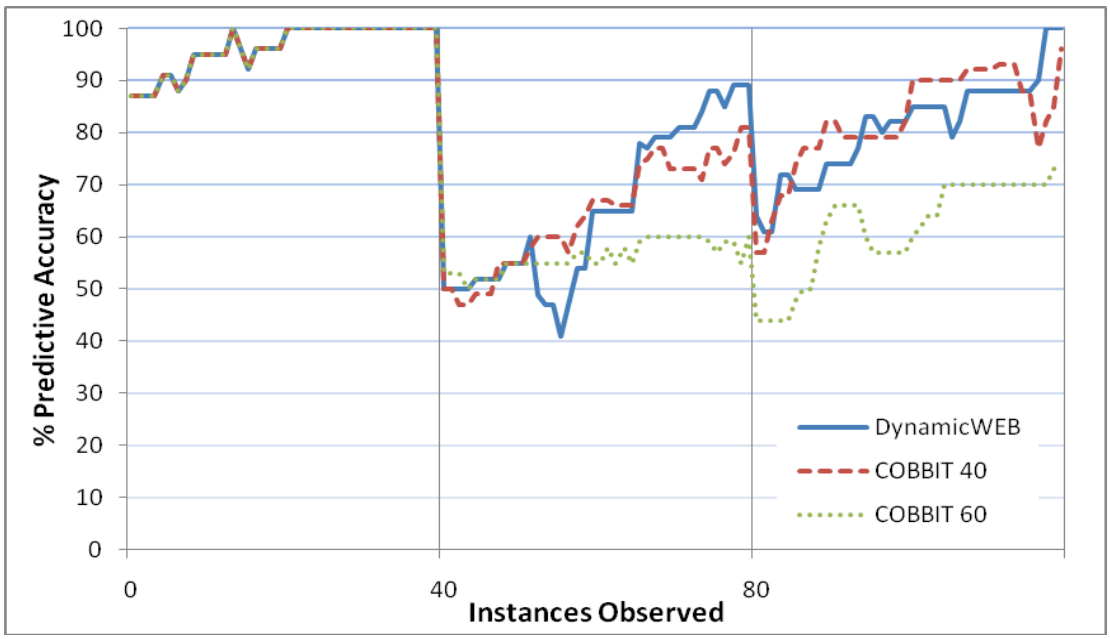


Figure 40. The learning response of DynamicWEB and COBBIT with the two window sizes of 40 and 60 upon the STAGGER Concepts dataset.

6.6 SUMMARY

This chapter has focused on verifying two different facets of the work reported in this thesis. Firstly the COBWEB implementation used for this work was examined to ensure that it was a correct duplication of Fisher's original implementation. This was done by comparing the predictive performance achieved by our implementation to that reported in Fisher's publications. As this work builds upon the work previously carried out in COBWEB and CLASSIT, it is obviously important to make sure the implementation is as accurate as possible.

The second half of the chapter examined the various additions that comprise DynamicWEB. The chapter explored the update mechanism at an individual-profile level and at a whole-dataset level within the Dynamic Weather dataset. In addition to this, a multiple tree implementation was examined on a dataset contained within Gennari's paper detailing CLASSIT.

After this, the performance of DynamicWEB was compared with other learners, when used on the STAGGER Concepts dataset. A comparison was made with the published results of two unsupervised learners (STAGGER and FLORA), This comparison showed that DynamicWEB was able to perform reasonably effectively. After this, a direct comparison was then made to the related learner, COBBIT, upon a single ordering of the dataset, illustrating that it could perform slightly better than the window-based approach, without the need for tuning the window size.

This chapter has established that DynamicWEB is a capable learner capable of adapting to change that may occur within a data stream. This was shown through using it on various simple learning problems that are easily understood. The remainder of this thesis uses datasets that are real world problems and not toy datasets.

Chapter 7

Examining Real World Datasets

"In times of change, learners inherit the Earth, while the learned find themselves beautifully equipped to deal with a world that no longer exists."

Eric Hoffer (July 25, 1902 – May 21, 1983)
American social writer and Philosopher

INTRODUCTION

In the preceding chapter, several artificial datasets were examined using the DynamicWEB conceptual clustering technique. The small size of these datasets and prior use by other authors mean that the content of these datasets is largely known. In this chapter several real-world datasets will be examined to see whether or not DynamicWEB can extract from them any structure or meaning that is not immediately obvious. The predictive ability of Dynamic Web will also be demonstrated on a large dataset.

7.1 INTRODUCING THE DATA

This chapter examines the performance of DynamicWEB on three datasets of a non-artificial origin. The first two of these were suggested by and sourced from the Australian Bureau of Statistics (ABS). They were selected after discussing the style of problem DynamicWEB was aiming to address with an employee of the ABS (Edmondson 2009). These two datasets will be discussed in more depth in their respective sections, but in overview, they detail various economic measures relating to the States and Territories of Australia. The first examines economic measures of the national product, by state, while the second describes the labour force, by industry, using measures relating to both part-time and full-time employment. However, it is worth noting that these datasets were examined in this thesis, not by a demographer but by a data mining researcher, and so any structure that is found has been interpreted by a non-expert in the field unless otherwise stated. The datasets are being examined here as an example of datasets on which DynamicWEB could be used, and the discovery of structure within the data is goal of this chapter. The third dataset examined within this chapter is the Physiological Modelling dataset published by BodyMedia (2004). This dataset contains the measurements recorded by a wearable computer monitor worn by a group of human test subjects as they undertook a range of activities. This sizable dataset will be examined here to present DynamicWEB with a prediction problem for comparison with other authors' performance.

7.2 AUSTRALIAN NATIONAL ACCOUNTS: STATE ACCOUNTS

The National Accounts dataset (ABS 2009) describes various economic measures for the eight Australian states or territories. The measures were taken each year for 17 years between 1990 and 2006. The measures, listed in detail in the table below, are measures of gross state product and income on a year-by-year basis for each state. All of the attributes, with the exception of the identifier and a date stamp, are numeric in nature. There is a total of 16 attributes for each state or territory. With 17 measurements of each of the 8 target objects the dataset contains 128 total instances.

This dataset does not contain any class labels. However it does present a scenario for illustrating any structure that may be present within the dataset. Beyond discovering whether DynamicWEB can represent the structure of the dataset at any given point of

#	Type	Description
1	ID	State or Territory identifier
2	Date	Date of data being recorded
3	Numeric	Gross state product: Chain volume measures
4	Numeric	Gross state product: Chain volume measures - Percentage changes
5	Numeric	Gross state product per capita: Chain volume measures
6	Numeric	Gross state product per capita: - Percentage changes
7	Numeric	Real gross state income: Chain volume measures
8	Numeric	Real gross state income: Chain volume measures - Percentage changes
9	Numeric	Real gross state income per capita: Chain volume measures
10	Numeric	Real gross state income per capita: Chain volume measures - Percentage changes
11	Numeric	Gross state product: Current prices
12	Numeric	Gross state product: Current prices - Percentage Changes
13	Numeric	Gross state product: Ratio
14	Numeric	Gross state product per capita: Current prices
15	Numeric	Gross state product per capita: Current prices - Percentage Changes
16	Numeric	Gross state product per capita: Ratio

Table 28. A description of the attributes that are within the National Accounts dataset.

time within the length of the period it covers, there is also the question of whether building a profile over time can discover more structure within the dataset by utilising the overall context of the dataset. This is undertaken through the use of derived attributes. In addition to this preservation, multiple trees are also trialled, splitting the dataset up into portions. Each of these portions is then examined simultaneously in parallel trees.

Several trials were completed using various combinations of the attributes listed in Table 28, together with derived attributes. The initial trials did not discover much structure within the dataset and are not shown here, but can be found, with accompanying discussion, in Appendix C. The most structure that was found in those trials was found in hierarchies based on the attribute groups that were weighted slightly more too scaled metrics. The scaled metrics are largely per-capita-based measures. As the task being examined involves profiling the eight state and territories in relation to each other, it is logical that attributes that largely reduce the effect of the population size of the states compared to each other would allow for a more meaningful comparison. With this in mind, another trial was run, grouping similar numerical attributes together. The first tree is comprised of the six attributes (Numbers 4, 6, 8, 10, 12, and 15 in Table 28) that record the percentage change within the measures. The second tree contains the attributes that relate the measures on a per capita basis (Numbers 5, 6, 9, 10, 14, 15, and 16). There are two attributes here, number 6 and 10, which overlap and appear in both of these structures. The same derived attributes, mean and standard deviation, were still included in both of there trees.

The structures produced by these two trees across the whole 17 years are shown in Figure 41. In the tree that tracked the profiles containing the percentage change data, the structures produced have revealed stronger relationships than were present within the context-less structures in Appendix C. On several occasions, multiple profiles are deemed similar enough to be present at the one node, while more objects are also present within child nodes. The hierarchy that contained the per-capita data is also more structured than that obtained from the other context trials and the context-less benchmark, but does not exhibit as many relationships as the percentage change tree, which may indicate some independence between the

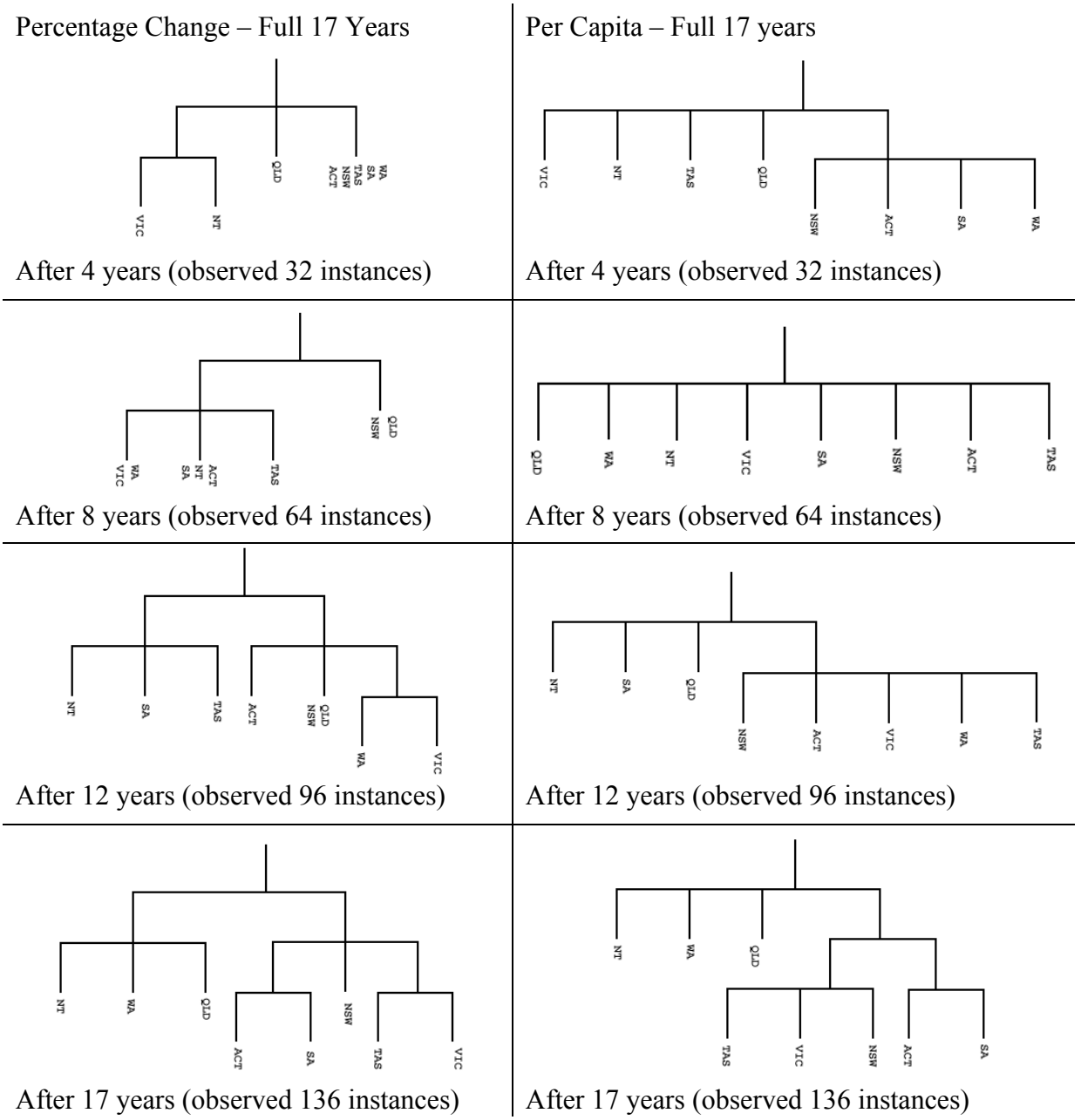


Figure 41. The Percentage Change and Per Capita trees with the profiles being based on the full set of 17 years worth of data.

attributes being examined in the two different trees. Overall the two structures do not have a great deal in common, based upon their structural characteristics, beyond the similar overall divisions in the first and last tree. A further trial was then completed to examine the impact of using a 5-year trial window on the derived attributes within the profiles and the results of this are shown in Figure 42. These two sets of structures are quite different from those produced using the complete 17 years' worth of data. This illustrates that the change occurring within the dataset is not uniform and that short term and long term results differ from one another. Therefore there is

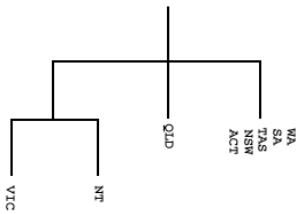
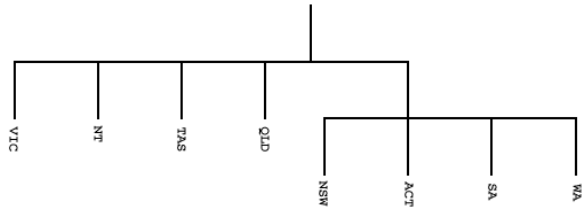
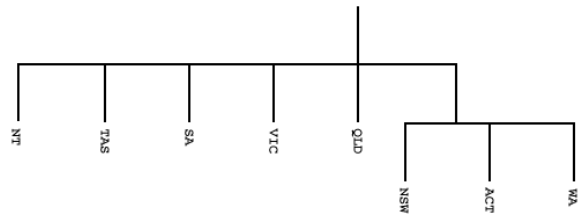
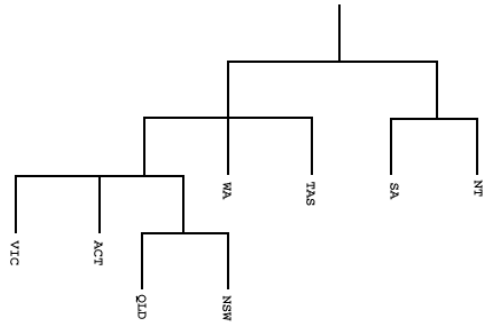
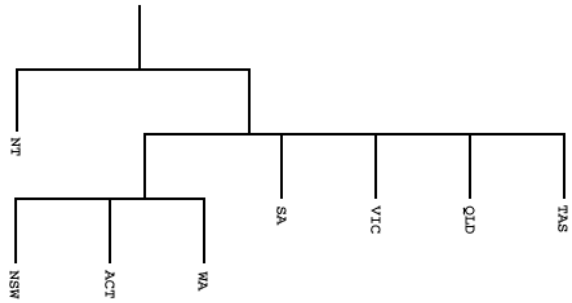
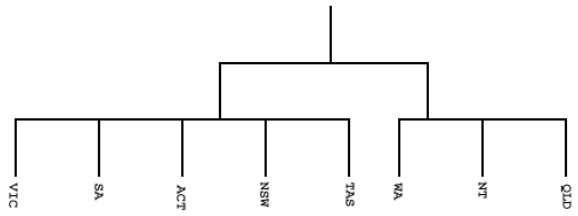
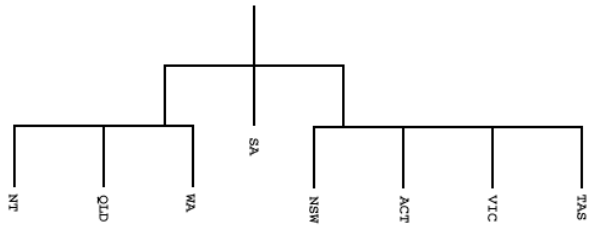
Percentage Change –5 Years	Per Capita –5 years
	
After 4 years (observed 32 instances)	After 4 years (observed 32 instances)
	
After 8 years (observed 64 instances)	After 8 years (observed 64 instances)
	
After 12 years (observed 96 instances)	After 12 years (observed 96 instances)
	
After 17 years (observed 136 instances)	After 17 years (observed 136 instances)

Figure 42. The Percentage change and Per Capita trees with the profiles being based on a 5 year window.

value in being able to adjust how much context is being retained by DynamicWEB, depending on the domain that is being examined and the requirements of the analysis being undertaken. On comparing the two structures, there are fewer tight clusters found within the second hierarchy, based upon the percentage change, although the profiles are still located in the same neighbourhoods of the overall structure. The per-capita hierarchy possibly yields slightly more structure using the time window, with the structure at 8 years having grouped several profiles together and with the

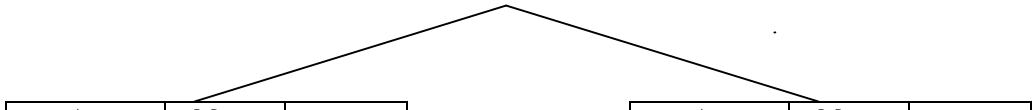
structure that is shown after 17 years having WA, NT and QLD now having been grouped together under a parent node instead of all being within their own child of the root node.

The two hierarchy series illustrated in Figure 41 and Figure 42 were shown to Dr Edmondson of the ABS (2009), who has had previous experience with this dataset. After examining the trees, he said that DynamicWEB had found a structure that is comparable to those produced by other, similar learning methods. There is a clear division present towards the end of the dataset. As expected, QLD, WA and NT were clustered together, presumably due to the recent mining boom in these states, while the other, larger east coast economic centres, were together. He found the occasional grouping of SA with other states to be of interest. It is somewhat of a wildcard state as, due to various historical factors, it is quite different to the others in an economic sense. Its similarity to other states varies over time across the different attributes, not remaining that much 'alike' another state for very long. Its higher mobility within the structure compared to other states illustrated that it was harder to cluster. Overall, Dr Edmondson was quite pleased with DynamicWEB's performance using the dataset.

The following diagrams of the knowledge hierarchies produced by DynamicWEB allow for relationships between the different profiles to be compared visually. The concept descriptions within the knowledge hierarchies can also be extracted and examined.

Within Figure 43 the concept descriptions are detailed for the two nodes that are the children of the root note within the final Percentage Change structure illustrated in Figure 41. The Nation Accounts dataset is a purely numeric dataset and so the concept description is comprised of a set of mean and standard deviation values of the 18 attributes within each profile.

Within these two descriptions there are a number of σ values equal to 1.0. This is the acuity value which is the minimum allowed value of deviation and 1.0 is the value used throughout the thesis in line with (Gennari, Langley et al. 1989) as discussed within Chapter 4 (CLASSIT). On this occasion it infers that this top division is fairly



Att	Mean	σ
#4	5.55	1.0
#6	3.70	1.13
#8	9.41	1.0
#10	7.44	1.0
#12	13.62	1.0
#15	11.61	1.004
STD #4	2.63	1.0
STD #6	2.41	1.0
STD #8	3.79	1.109
STD #10	3.32	1.282
STD #12	4.30	1.193
STD #15	4.02	1.231
Trend #4	1.22	1.301
Trend #6	6.78	1.0
Trend #8	5.22	2.048
Trend #10	-13.67	1.0
Trend #12	-9.67	1.0
Trend #15	-2.56	6.637

Branch contains: QLD, NT and WA

Att	Mean	σ
#4	2.11	1.0
#6	1.36	1.0
#8	2.6	1.0
#10	1.84	1.0
#12	5.06	1.0
#15	4.31	1.0
STD #4	2.31	1.0
STD #6	2.38	1.0
STD #8	2.11	1.0
STD #10	1.96	1.0
STD #12	2.41	1.0
STD #15	2.32	1.0
Trend #4	0.07	2.49
Trend #6	3.36	2.34
Trend #8	0.64	1.08
Trend #10	-13.36	1.0
Trend #12	-9.36	1.0
Trend #15	3.38	11.79

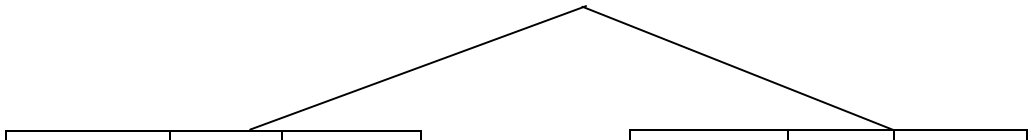
Branch contains: NSW, VIC, TAS, ACT, SA

Figure 43. The Concept Description of the two child nodes of the final structure shown within Figure 41.

stable, and that there is not a great deal of diversity in attribute values within each of the two branches.

These descriptions allow for the clusters created to be examined and understood. In this example it can be seen that the profiles with the greatest percentage increase, across the period of the dataset, have been separated from those with lesser percentage increases. Most of the description values within the concept that contains Queensland, the Northern Territory and Western Australia are larger than those in the other side of the tree. The largest differences between the two concept descriptions are the derived trending values for attributes 6 and 8 (Gross state product per capita and Real gross state income: Chain volume measures).

In addition to the concept descriptions outlined, there are the two concept



Att	Mean	σ
#4	5.77	1.5
#6	3.70	1.48
#8	9.33	1
#10	7.73	1
#12	14.57	1.35
#15	12.33	1.50
STD #4	3.22	1
STD #6	2.84	1
STD #8	4.5	1
STD #10	3.63	1
STD #12	6.4	1.15
STD #15	5.80	1.16
Trend #4	0.67	2.31
Trend #6	7.33	1.15
Trend #8	4.67	3.05
Trend #10	-14	1
Trend #12	-10	1
Trend #15	1.33	11.01

Branch contains: QLD, NT and WA

Att	Mean	σ
#4	1.86	1
#6	1.04	1
#8	2.71	1
#10	1.9	1
#12	4.91	1
#15	4.11	1
STD #4	2.04	1
STD #6	2.11	1.02
STD #8	1.46	1
STD #10	1.09	1
STD #12	2.45	1
STD #15	2.27	1
Trend #4	-0.78	1.93
Trend #6	2.92	2.16
Trend #8	1.071	1.49
Trend #10	-13.36	1
Trend #12	-9.36	1
Trend #15	5.93	9.23

Branch contains: NSW, VIC, TAS, ACT, SA

Figure 44. The Concept Description of the two child nodes of the final structure shown within Figure 42.

descriptions shown in Figure 44, which are those for the same two child nodes of the root (in Figure 43) but from the windowed data shown in Figure 42. These two nodes contain the same profiles of states as the other concept description, but this time the derived attributes are only built upon the 5 most recently observed years. These two different sets are similar, which is to be expected since they are profiling the same target objects, but they also show some difference which is to be expected because they been built using a different number of observations of the target objects. In analysing these two concept descriptions to discover whether this difference was significant, a paired t-Test (Paired Two Sample for Means) was undertaken. The difference between the two concept descriptions for the branch containing QLD, NT

and WA profiles was statistically significant ($t(35) = -2.97$, $p<.001$ α 0.05). However, the alternate branch was not considered significant with a α of 0.05. While this does split the results of the significance built upon these different structures, the fact that there is enough difference for one of the top level divisions within the tree to be considered notably different means that the concept hierarchy produced is also significantly different when constructed with a varied amount of the context being retained.

The examination of this dataset has shown DynamicWEB was able to create and maintain simple profiles using a moderate number of observations of a few target objects. Structure could also be discovered within the National Accounts dataset using derived attributes preserving the context of change occurring within the dataset.

7.3 LABOUR FORCE DATASET

The second dataset to be examined here is another sourced from the Australian Bureau of Statistics. It is called the "Labour Force" dataset ((ABS) 2009). The Labour force dataset describes the numbers of persons employed in the 8 States and Territories of Australia based upon the industry in which they work. The data is also divided into the number of people who are working full time and the number working part time, with a total also included. A list of the industries that are recorded within the dataset is given in Table 29. The dataset covers a period of 24 years with 4 measurements recorded within each year between November 1984 and August 2008. There is a total of 95 recordings per state within the dataset (with partial years recorded also counted).

This dataset was examined briefly in its current form and the results of this are found in Appendix D. These results show that the dataset is very regular and also presents a similar learning scenario to the National Accounts dataset. As such, after this initial trial, the dataset was transformed so that the entities being tracked were the individual industries themselves. This presents a dataset with twice as many target objects, but still with many attributes describing each profile, as each state's data items relate to an industry at one point in time. As a result there is a total of 1710 observations of 18 target objects, each with 24 attributes (three for each state).

#	ID	Attribute
1	AFF	Agriculture, Forestry and Fishing
2	Mining	Mining
3	Manuf	Manufacturing
4	EGWS	Electricity, Gas and Water
5	Const	Construction
6	WholeT	Wholesale Trade
7	RetailT	Retail Trade
8	ACS	Accommodation
9	Tran	Transport and Storage
10	CommS	Communication and Services
11	FinIn	Finance and Insurance
12	PropBus	Property and Business Services
13	GAD	Government Administration and Defence
14	Edu	Education
15	HealCS	Health and Community Services
16	CultRS	Cultural and Recreational Services
17	PerSer	Personal and Other Services
18	Total	Total (All Industries)

Table 29. Industries that are described by State or Territory within the Labour Force dataset.

Within this dataset, which details each industry individually, the concept that is of most interest, apart from the comparison between each of the industries, is the comparison between the part time and full time jobs in each industry. Therefore, in examining this dataset with DynamicWEB, a two-tree forest was created with one tree tracking the full time employment data, while the other tracked the part time data. As with the previous dataset, the first trial examined is a simple profile, based on the mean and standard deviation of the observed attributes. The result was a set of profiles in each tree, with each profile containing 24 attributes: with the most recently observed value, the average and the standard deviation recorded for each attribute. Eight structures, produced by DynamicWEB over the period of a single year in 1995 (after having incorporated 11 years worth of observations) are shown in Figure 45. Four of the structures are for the full time employees and four for the part time employees. The structures were for the most part fairly stable with only a small degree of migration taking place over the course of the year. Changes produced within the tree rarely altered the nearest neighbours of a profile and were generally

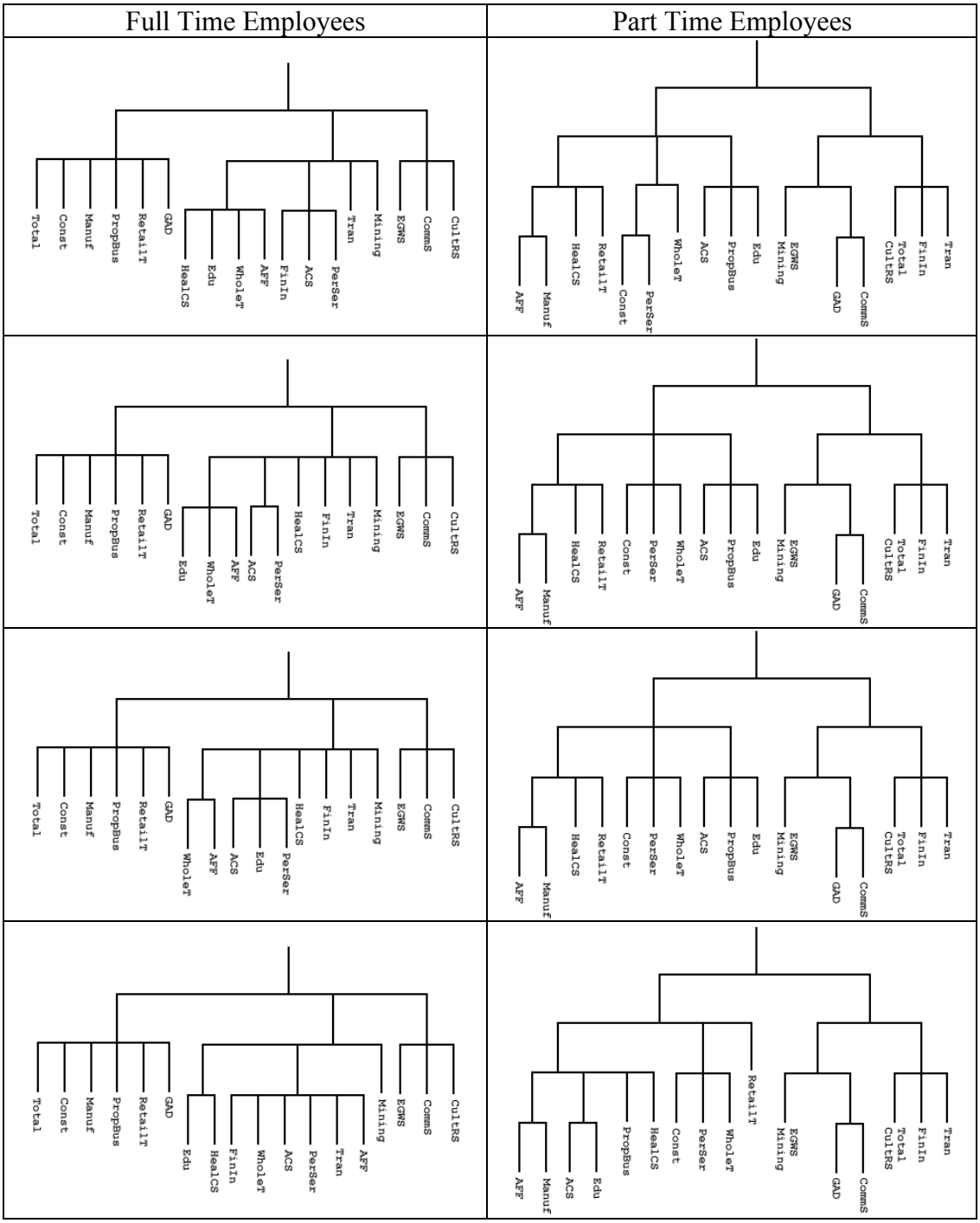


Figure 45. Eight structures produced covering the four observations that occurred in 1995. The left set of structures represent the full time employees by industry, and the right set represent the part time employees.

just the result of a split or merge operation taking place during the update process.

The knowledge hierarchies produced (Figure 45) model the industries within the Labour Force dataset from an overall size perspective. Industries that are large are grouped together, and those that are small are grouped together. While this is useful for monitoring industry trends and growth, also of interest would be the changes that

are taking place that are independent of industry size. Another trial was conducted, this time using a forest of two trees, one for full time employees and one for part time employees, but where all the attributes are derived attributes. The attributes used are the trend, the standard deviation (as a percentage of the most recent observed value) and the difference between the most recent observed value and the second most recent value (also as a percentage of the most recent value). All three of these focus on the change that is taking place, with the first two tracking the change

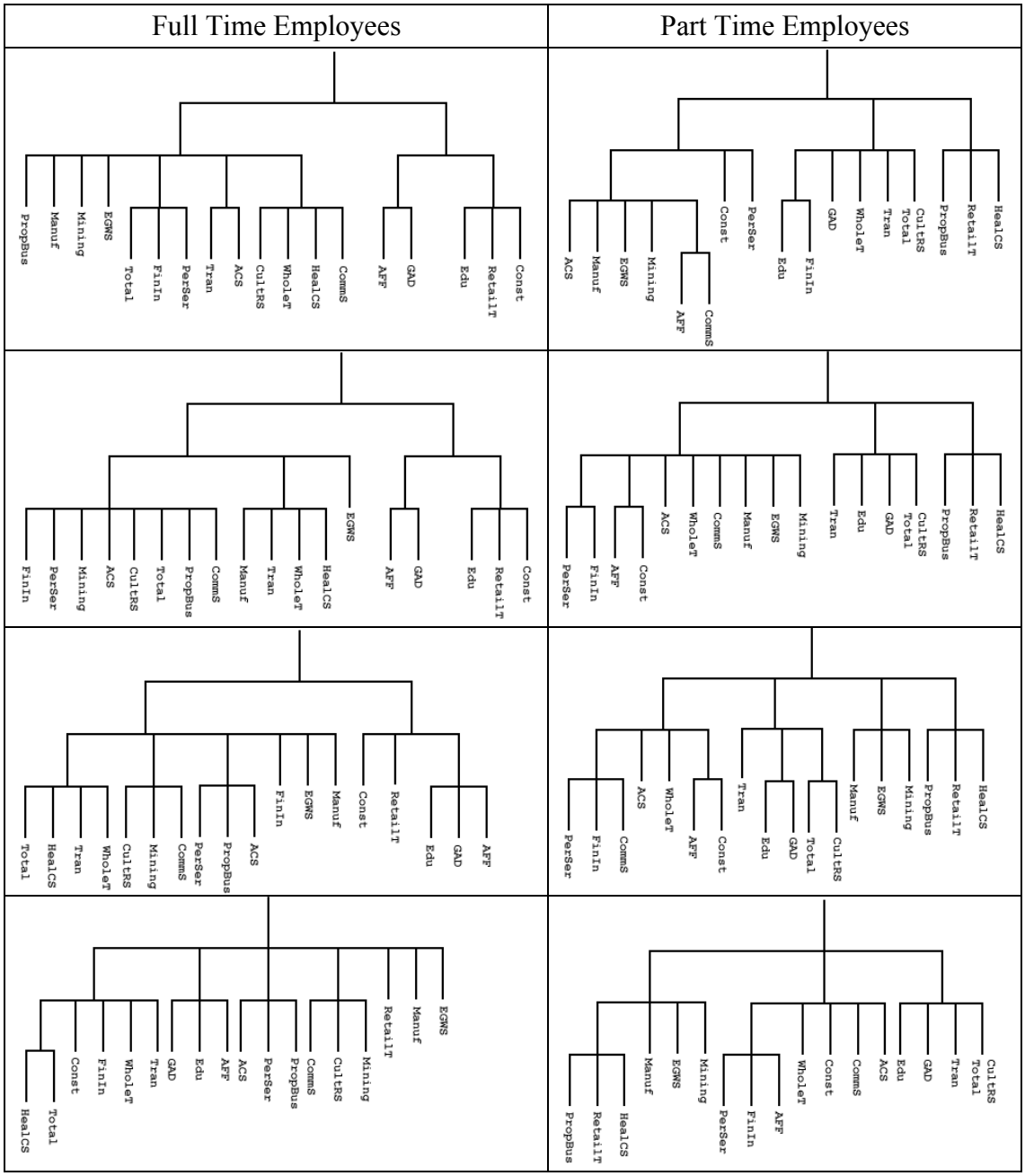


Figure 46. Eight structures formed over the year 1995. The left is formed upon the data relating to the Full Time Employees, while the right is the Part Time employees using trend instead of mean.

over time, while the third focuses on the scale of the most recent change. The structure that has been produced with this derived attribute concept hierarchy is less stable than the structure produced using the mean- and standard-deviation-based attributes. It is still reasonably stable, but several large restructures occur (for example in the 4th *Full Time* tree and in the 3rd and 4th *Part Time* trees). However the nearest neighbours of profiles are still largely constant across these changes. This may imply that some of the industries have undergone similar changes to each other, over the course of the year, resulting in them migrating, as a group, away from other industries. This could be due to the influence of seasonal effects on some industries. Retail is an obvious example of an industry that would be affected by seasonal factors, and an examination of the major changes that take place within the part time hierarchies reveals that the *Retail* profile does indeed move from its previous position in the first three tree structures to a position located further down the tree in the fourth tree structure. The seasonal nature of the Retail industry is not necessarily the underlying cause of this change in its position within the fourth tree structure, but it is a possibility.

The results from this dataset were also shown to Dr Edmondson of the ABS (2009) and he was similarly (as with the National Accounts dataset) pleased with DynamicWEB's performance. He agreed with comments outlined above in relation to the Retail sector's profile within the concept hierarchies. However, he indicated that in its present form this dataset was harder to evaluate and extract value from, due to the number of profiles present compared to that of the National Accounts dataset. He suggested that the hierarchies would be of substantial interest if they were viewable in a visualisation package. This would facilitate examination of the concept descriptions 'on the fly' as the concepts are formed to enable more knowledge to be extracted from the hierarchy. This idea is discussed further in Chapter 9.

7.4 PHYSIOLOGICAL DATA MODELLING CONTEST

The two Australian Bureau of Statistics datasets discussed in the previous section were examined in an effort to discover structures within these datasets, and to establish whether the use of profiling could improve our ability to produce knowledge hierarchies through the preservation of context. This section examines the

predictive accuracy of DynamicWEB within a dataset which contains thousands of updates occurring to each profile.

The Physiological Data Modelling Contest (PDMC) dataset is a collection of recordings that were gathered using a wearable body monitor produced by a company called BodyMedia (2004). The dataset was supplied by BodyMedia for a contest that was held at one of the workshops at the 2004 International Conference on Machine Learning (ICML). The dataset consists of the combined outputs recorded by wearable body monitors worn by the members of a sample group of twenty one people. These participants each pursued their ordinary daily routines, whilst wearing the monitor, and simultaneously noted down the activities that they were undertaking. Some of the activities recorded included sleeping, watching television, working at a desk, riding a bike, driving a car and using a computer. Periods of the day that were not recorded still remained within the dataset but are noted as being unlabelled. In total there are about 200,000 measurements contained within the dataset.

Along with the information about the wearer's activity during the day (listed in the

#	**Attribute**	**Description**
1	User ID	Numerical
2	Session ID	Numerical
3	Session Time	Time in seconds measured once a minute.
4	Age	Age in years
5	Smoker	Whether the person smokes or not (1, 0)
6	Handedness	Left or Right handed (0,1)
7	Gender	Male or Female (0, 1)
8	Annotation	Numeric based code
9	Sensor 1	Galvanic Skin Response (the electrical conductivity of the wearers skin)
10	Sensor 2	Heat Flux (the amount of heat that is being dissipated through the skin)
11	Sensor 3	Near Body Temperature (air temperature near the skin)
12	Sensor 4	Pedometer
13	Sensor 5	Skin temperature
14-17	Sensor 6-9	Accelerometer (longitudinal and transverse)

Table 30. A description of the attributes present in the Physiological Data Modelling Contest (PDMC) dataset.

dataset as the annotation), other attributes were recorded within the dataset (Table 30). Each individual wearing a monitor is represented by a userID, and each continuous period, during which they wore the monitor, is listed as a session. The *session time* attribute preserves the ordering of the data in its sequential state within the session. The remaining attributes describe the individual's behaviour as recorded by the device's sensors or their details such as age, gender, handedness and whether or not they smoke.

In the contest that was held using this data there were several learning goals to be achieved. Two of these involved prediction of the annotation class within a test set that did not contain the annotation values. The prediction scenarios were both two-class problems, in which a single annotated activity class is to be predicted as the positive class, while the remainder of the activity classes are grouped together to form the negative class. Several of the annotations are to be ignored in each scenario due to the likelihood that that the positive class may have been undertaken inside those time periods also. For example unlabelled time periods are one of these to be ignored within both scenarios. The two target classes in the contest were the annotated activities of watching TV and of sleeping, represented by the annotation values of 3004 and 5102 in the dataset respectively.

These two learning scenarios are analysed here using DynamicWEB, with the aim of finding out how quickly DynamicWEB is able to learn the domain and reach convergence. The performance of DynamicWEB was measured multiple times and ten-fold cross validation used to determine the average learning capability of DynamicWEB within this domain. The dataset was split into multiple datasets of 10,000 observations, roughly spanning between 50 and 120 sessions. These smaller portions were each then used to build a concept hierarchy, and this hierarchy tested by measuring the accuracy of its predictions when applied to another portion of the data. Not all of the splits within the dataset were used, as some of them contained no examples of the positive class (specifically the television watching class as it only makes up 2.5% of the total data); however, ten trials were completed for each learning scenario. Note that, while this is a sequential dataset, using ten-fold validation is a valid testing methodology as the sessions are largely complete within the sequence folds, and in their comparisons to other segments of the dataset are merely comparing disjoint sessions completed by different users. The overall

structure of the dataset is not that of a single sequential data stream, but rather a collection of complete data streams, each one is sequential in itself. As such, each fold contains a collection of complete sequences, each of which forms a complete profile. The folds therefore all contain complete profiles enabling them to be used in comparison to one another.

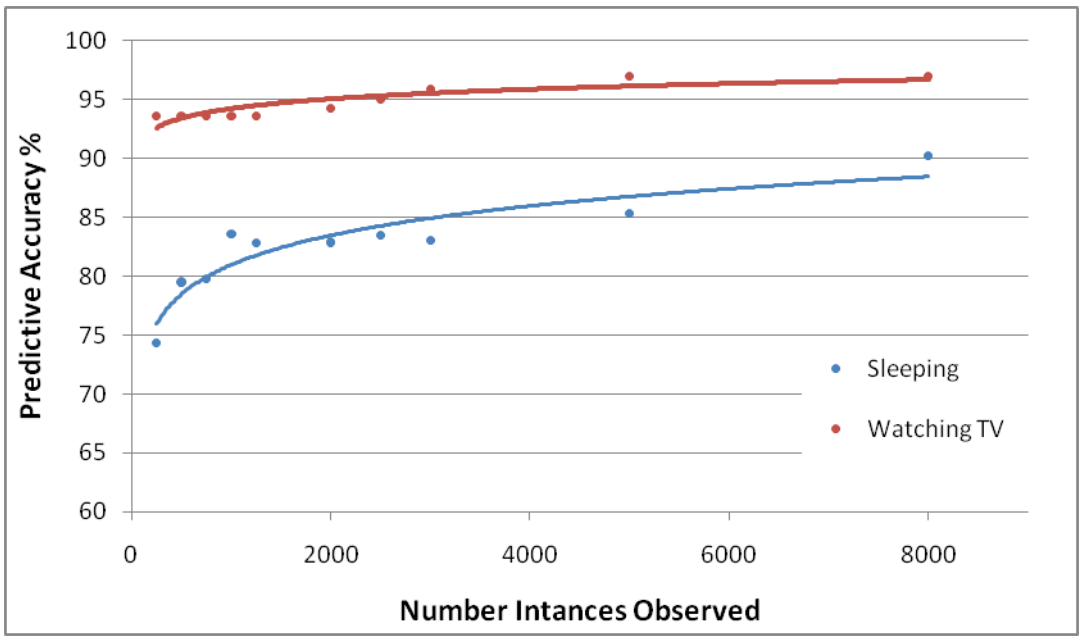


Figure 47. Learning performance of DynamicWEB upon the Sleeping and Watching TV scenarios using the most recently observed value.

The first complete run of the two learning scenarios is presented in Figure 47. In this trial, DynamicWEB only stored the most recently observed value of each of the attributes that describe each target object. This also includes the class value where it may have changed over time. However, this value was not used within the learning process because it represents the target class value that is being predicted here. One of the results of this is that each profile stored within the hierarchy can only be used in the prediction of the last class that was observed from this target object. Therefore, if over the observed time period a profile is a member of multiple classes, then, once the observations cease to occur, the only knowledge that is retained from which to make predictions is the last known class. This is an inherent shortfall of DynamicWEB in relation to the storage of past contextual information. As the goal of DynamicWEB is to track each object and develop a profile of its activity, if we were to add a mechanism that divided a profile once such an event occurred we would then have multiple profiles for a single given object and this conflicts with the

main principles of DynamicWEB. However, it is acknowledged that such a mechanism could be useful in some knowledge domains.

Examining the various submissions to the ICML workshop reveals a wide range of results. The best predictive accuracy for "sleeping" was just over 85% and for "watching tv", just over 91%[9]. The median of the 16 submissions for the first scenario was 70% and the second was 82%. DynamicWEB's performance compares very favourably with this, exceeding that achieved by the best performers from the contest held at the workshop. It should be noted, however, that the performance comparison shown in Figure 47 does mask a shortfall within the DynamicWEB results on the *Watching TV* scenario. This shortfall will be discussed further shortly.

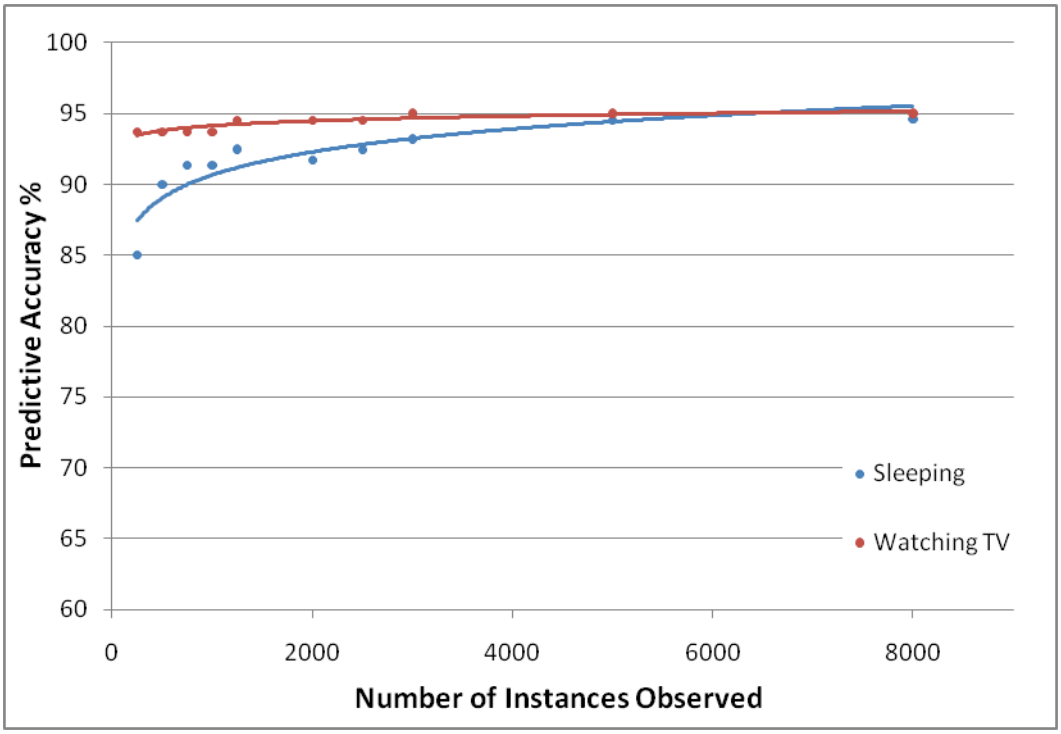


Figure 48. Learning performance of DynamicWEB on the *Sleeping* and *Watching TV* scenarios using 2 derived attributes to store contextual data.

The previous trial of DynamicWEB's predictive ability only stored the most recently observed value relating to the wearer of the device. Within the next trial, two derived attributes were added to the profiles to preserve some of the contextual information relating to the session being undertaken. The attributes that have been added are the

[9] A lower value than this was shown in the final summary presentation for the workshop without explanation as to why. However these higher values are reported in the presentation and the actual paper submissions on the PDMC website.

mean and the *standard deviation* of each of the seven sensors. These two derived attributes are operating within a time window of 5 observations, representing 5 minutes of recorded time. As some sessions recorded within the dataset extend over several hours and cover multiple different activities, using the complete set of the data for each session is counter-productive. A 10-minute window and a 20-minute window were also trialled. The same results were found for the 10-minute window as were found for the 5-minute window, but the accuracy for the 20-minute window was significantly lower. Other methods (Gama and Rodrigues 2004) presented within the workshop were also based on a window of recently viewed observations. The results of the DynamicWEB windowed trial is shown in Figure 48 and illustrate a notable improvement in the speed of learning and final accuracy when using the *Sleeping* activity as the target class. The hierarchy was able to achieve a 90% predictive accuracy using 6000 fewer observations when using the most recently observed attribute values than when not using the derived attributes. The predictive accuracy for the *Watching TV* class appears to be slightly negatively affected by the usage of conserving the context within the derived attributes with a predictive accuracy that is 2.5% lower after observing 8000 instances.

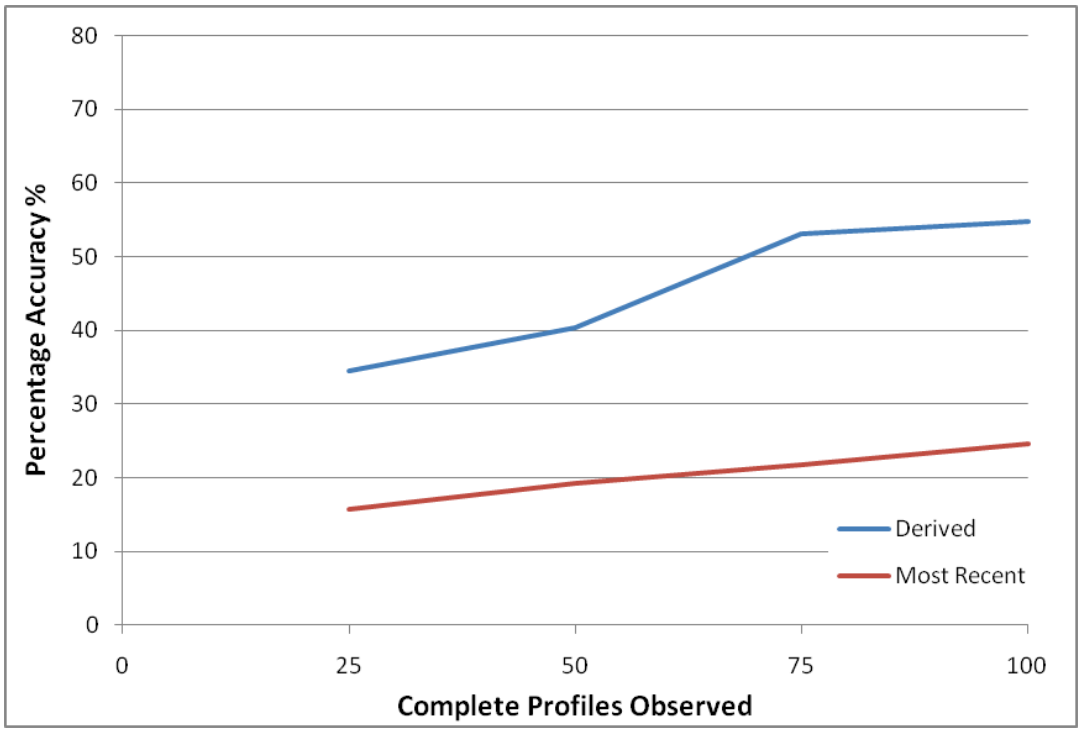


Figure 49. Comparison between the classification accuracies for just the positive class of the *Watching TV* scenario in the two trials.

Both of the figures above show DynamicWEB in a positive light, with the learner performing well when predicting the *Watching TV* class in both trials, and the *Sleeping* class improving markedly in the second. However, in both of these trials the *Watching TV* class predictive performance using DynamicWEB's is being overstated due to the nature of the data which is being examined. The over 90% accuracy illustrated is the accuracy of predicting both the positive and negative class within the problem; however, the positive class is quite rare within the dataset, making up only about 2.5% of all the activity values. With such an unevenly distributed dataset, if all instances were simply classified as the negative class then a 97.5% accuracy would be attained. Obviously this would not be an acceptable model, but it does highlight the need to examine what the classification accuracy of the positive class is by itself. Within Figure 49 the predictive accuracy results of just the positive class for the *Watching TV* scenario are shown. The x-axis in this graph represents completely formed profiles averaging the results from 10 separate datasets that each had at least 100 different profiles. These vary in length from 8,000 to 12,000 total observations each. Even though only 2.5% of all the instances recorded in the dataset are for the class to be predicted here, most of these datasets actually contained 3 or 4 objects of the goal class in their final produced hierarchy. This slight over-representation is possibly a quirk of the dataset with the participants switching off their sensor while watching some television for some reason. The results for the two sets of attributes shown in Figure 49 illustrate a much greater difference between the two than that visible within Figure 47 and Figure 48. The performance for the most recently observed values is quite disappointing, with less than 30% accuracy achieved after 100 profiles have been examined. When derived attributes are introduced, preserving some of the context of the observations creating the profiles, the performance more than doubles. While this is obviously not as impressive as the results shown in the earlier figures, where accuracies exceeded 95%, it does accurately represent the true performance of DynamicWEB on the dataset. Further profiles were added in an attempt to improve DynamicWEB's performance, but they didn't succeed in doing so, with just over 55% being the final average accuracy achieved. On several runs within the 10-fold cross validation process an accuracy of up to 68% was attained, but again on several other runs lower accuracies were achieved.

This aspect of the dataset was noted by the contestants Wei-Hao Lin and Alexander Hauptmann (Lin and Hauptmann 2004), who stated that "the model suffers greatly from the scarcity of positive sequences and performs poorly" although they did not detail what predictive accuracy they had achieved on just the positive class. However, another entrant to the contest (Azé, Lucas et al. 2004) that didn't highlight this issue but did detail their performance upon just the positive class by itself, reporting a result of 67.5%. This is better than that achieved by DynamicWEB by about 10%, which does highlight that the initial results for DynamicWEB shown in Figure 47 were covering a shortfall in predictive performance. The *Sleeping* activity class does not suffer from this same difficulty as it represents almost 50% of all of the observations within the dataset. Although, generally each person would only record one session a day of sleep, based on the percentage makeup of the activity, most were diligent in wearing the sensor. Overall it can be said DynamicWEB has performed fairly well in comparison with the contestants in the ICML contest.

7.5 SUMMARY

Within this chapter three non-synthetic datasets have been examined. The first dataset (the National Accounts dataset) was sourced from the Australian Bureau of Statistics. It was examined to see if a knowledge hierarchy displaying structure within the datasets, between the states and territories, could be produced. It was found that using derived attributes within the learning process enabled significant structure to be discovered.

This was then followed by a second dataset (Labour Force) that was also sourced from the ABS. Labour Force initially presented a similar learning scenario with more recordings of the states and territories, but in relation to employment in 17 industries. This dataset was transformed to profile the industries themselves across the states. These were then compared with each other in the part time and full time contexts. Unlike the National Accounts dataset, which only contained one annual measurement, Labour Force contains 4 measurements per year. A single year about half way through the dataset was examined quite closely, showing how the structure drifted over the year, with some objects, or groups of objects migrating from one branch to another within the hierarchy.

The third dataset examined was one that presented a classification scenario in the presence of very dynamic and highly dense data containing thousands of recordings of multiple target objects, across many sessions. DynamicWEB showed here that it was able to operate within an environment of great change and to still produce accurate predictions.

Within this chapter the ability of DynamicWEB to profile activity across multiple observations was extended beyond that which was shown in the previous chapter. DynamicWEB's ability to profile activity across multiple observations, preserving the context of time, makes it quite a unique unsupervised learner.

Chapter 8

Profiling Network Activity and Performance

"He that will not apply new remedies must expect new evils; for time is the greatest innovator."

Sir Francis Bacon (22 January 1561 – 9 April 1626)
"On Innovation," Essays, 1597

INTRODUCTION

The research described in this thesis was originally inspired by a detection problem within the network security domain. In the previous two chapters we have examined a combination of small synthetic machine learning datasets and several non-synthetic data mining datasets, all of which were un-related to network security. This chapter will describe the use of DynamicWEB upon network-based data, including data related to the original security problem.

8.1 INTRODUCTION TO THE DATA

DynamicWEB was produced in response to a detection problem that was discovered while research into scan correlation was being under taken. This research will be described briefly to provide some context for the detection problem. The fundamentals of this problem, along with other security detection scenarios, drove the philosophical design principles that led to the development of DynamicWEB. Within security research, in the fields of intrusion detection, scan correlation, and other detection domains, behaviour profiling is a topic which has been subject to much investigation (Amoroso 1998). Developing profiles of certain activities is the foundation of anomaly detection. DynamicWEB, as has been outlined in previous chapters, develops profiles across multiple observations of the same target objects, retaining some of the context across these observations. This chapter will describe the construction of profiles, initially based on port scan activity from source IP addresses, with the objective of inferring relationships between different port scan profiles. After this, a dataset detailing the network performance of a large number of computers upon a national private computer network will also be examined. Both of these activities require profiles to be built, based upon events relating to networked computers that are sampled multiple times. These profiles operate within a time context, as the events are sequential through time, but are tracking many different individual computers which have a wide geographical spread.

8.2 SCAN CORRELATION

The use of network technologies, especially the internet, has become extremely widespread in the last few decades. Most people now use computer networks not only within their working lives, but also recreationally. Ecommerce is a massive industry, with an online presence now seen as a necessity for businesses. However, in addition to these productive uses there are also networked users who wish to use these widespread networks for malicious reasons. Due to the sheer size of the networks that spread around the globe, those who wish to do harm to others have a barrier which they first must cross: locating targets. Over the last decade the malicious users who have had the most impact on a global scale through network misuse are those who have used vast numbers of compromised computers to accomplish some task. Worms had a large impact during the early 2000's while bot-nets are currently an area of concern among security experts (Bailey, Cooke et al. 2009). A key component in both of these forms of attack, and indeed all network attacks, is knowledge of the address of possible target computers. This knowledge is usually obtained by the use *of port scans*. A *port scan* is basically the reconnaissance portion of an attack upon a given computer. It is carried out via the use of automated (or partially automated) port scans by which an intruder is able to discover populated IP addresses, together with key information about a host computer, such as the operating system being used, what ports are open and whether certain applications are being run. This information can then be used to launch an attack against the computer, attempting to make use of some existing vulnerability for that specific platform. The research described in this section aims to develop a profiling method that links multiple source IP addresses together as one entity. The intention of this technique is to show that two attacking profiles originate from the same user, who has changed his IP address to avoid detection.

8.2.1 PORT SCANS

Throughout much of the research and development of intrusion detection systems, scans of hosts and ports have been described as the reconnaissance portion of an intrusion (Li, Song et al. 2004), and only shown to be of use once further malicious activity has occurred. Due to the sheer number of scans that occur, it has been far too

computationally expensive to attempt to correlate scan data. As only a small percentage of scans ever translate into full blown hacking attempts this has not been seen as a serious concern.

However, recent research is tackling the problem of correlating this data in an efficient manner (Jung, Paxson et al. 2004). If the data can be analysed and correlated, then further attacks may be thwarted, or scan profiles from repeat offenders constructed. The precursor nature of reconnaissance activity can then be used to develop defensive mechanisms for later use, if resultant attacks do occur.

There has been a great deal of research conducted on the analysis of audit logs and network activity within the context of Intrusion Detection (ID) and much of the knowledge learned can be translated to scan correlation. Many challenges have been overcome in the past 20 years, enabling ID systems to scale to large networks while at the same time remaining effective. Scan correlation research now hopes to achieve these same twin goals. Before we examine existing scan correlation systems, it is appropriate to take a closer look at why scans occur.

Every IP address gets scanned. There are a finite number of IP addresses; it is the way that network addressing was designed. To re-visit the often-used analogy of a hacker being like a burglar breaking into a computer instead of a home, imagine a burglar who has access to an address book of every house in the world, and can find out a few details, such as whether anyone lives there and what the alarm system is, without leaving the relative safety of his own home. This is why every computer gets scanned. Hackers have an address book of possible locations, and, with a handful of scans that last only a few seconds, they can discover whether a computer is at the address and can identify some of the services that it is running. Figure 50 illustrates recorded activity of a scanner probing 250 IP addresses in less than 1 minute.

There are many readily available tools which allow for various automated scans to be completed at the click of a button. The most commonly used probing utility, used both by system administrators and malicious users alike, is called nmap (fyodor 2005). Nmap allows for a wide range of different scans to be completed over various IP ranges or

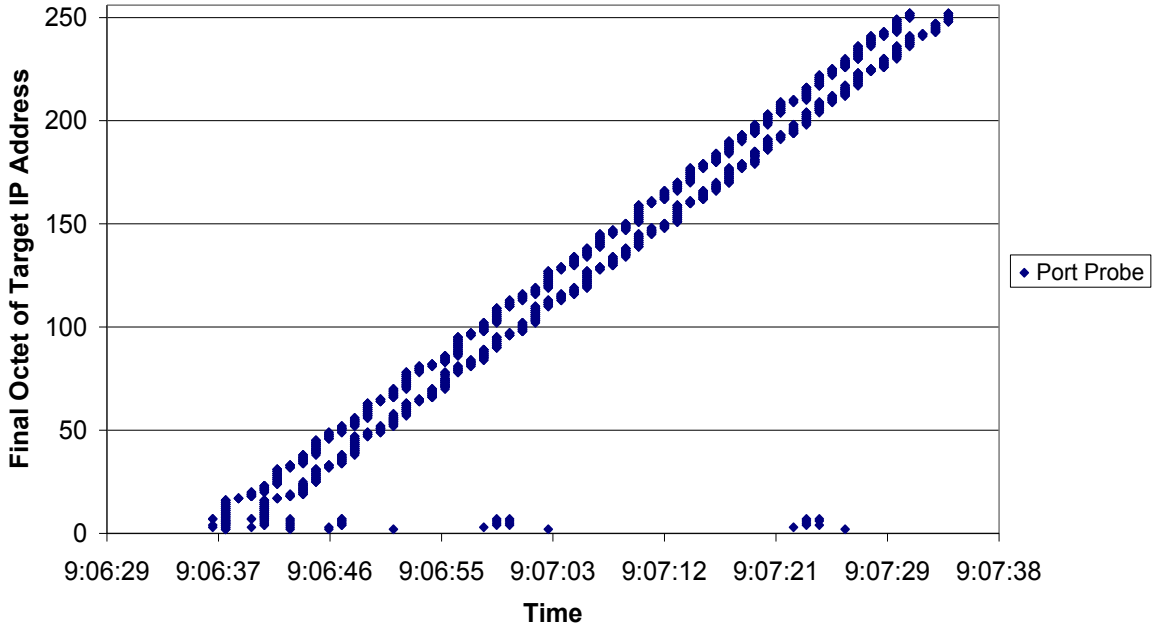


Figure 50. A source IP recorded scanning across an entire class C IP address range.

lengths of time. It comes equipped with over 700 different recognisable operating system finger prints to match to scan results. Furthermore, it also allows for various scans, such as idle scans, that guarantee anonymity. An idle scan involves using a second computer as an intermediary to hide the identity of the scanner (Zalewski 2005). The attacking PC probes the target, spoofing the source address as being from an idle computer, while at the same time constantly probing the idle PC. If the idle computer is only responding to the hacker's activities, the ID in the header of the response packets coming back to the hacker will increment when the target PC responds to the spoofed packet. This allows the malicious user to know that the target IP and port are indeed present and open.

Reconnaissance activity occurs whenever an attack is about to take place. Malicious computer users also probe systems well ahead of a more direct attack, looking for targets and then weaknesses. In this way, scans are undertaken across vast ranges of IP space, first mapping the locations of gateways, networks and hosts, before then probing these computers looking for vulnerabilities to attack. However there is also scan activity which is benign in nature, originating from sources such as web crawlers and proxies. Often these types of services appear as a scanner, but are totally benign. One of the

challenges facing scan correlation systems is to classify benign scan activities as well as malicious ones.

8.2.2 SCAN CORRELATION SYSTEMS

This section will briefly examine two scan correlation systems, the first of which is similar in some respects to previous work completed by the authors. This work was detailed in section 4. Both techniques operate using statistical anomaly-based methods involving thresholds. Existing IDS systems such as Bro (Paxson 1999) and Snort (SourceFire 2004) both make use of thresholds in the context of scans. These both use fixed values, one of 20 and one of 100, with no apparent justification for the values chosen. The second scan correlation system discussed in this section shares some of the same goals as the DynamicWEB system introduced in this thesis.

8.2.2.1 THRESHOLD RANDOM WALK: SEQUENTIAL HYPOTHESIS TESTING

At the same time as the work discussed in section 8.2 was undertaken, similar work was being completed and published by Jung et al (2004). Their system focused primarily on distinguishing between the benign scans that take place and the malicious scans. The question that Jung et al were answering was how to detect when a scanner is malicious and when it is benign. Both Jung's and the work in 8.2, at a fundamental level, involve simple profiling of scan behaviour, to extract meaning from otherwise noise-laden scan activity.

Jung et al (2004) proposed a detection algorithm called Threshold Random Walk (TRW). The algorithm is based on the mathematical technique called Sequential Hypotheses Testing described by Wald (1947). The basis for the algorithm is that scans or failed connections to unpopulated IP space are much more likely to come from a malicious user than an authorized user. As a possible scan takes place for each host that is probed, the TRW algorithm notes whether the source IP was successful or not, with special attention given to whether the destination IP was in use or not. For the source IP, a tally is kept of the results and, once that value reaches a threshold level, a decision is made as to whether the source IP was an unauthorised scanner or not. The method has

given good results, outperforming both the Bro and Snort intrusion detection systems when detecting scanners. In addition, web crawlers and proxies were distinguishable from regular scanners, largely because they rarely probed unpopulated space.

While the system gave good results on the networks on which it was tested, it would be interesting to see how it would perform on a more heavily populated IP space. The best results occurred on a network which was only 4.47% populated, with the second network tested being only 42% populated. The system seems to have been built without considering how many companies, government departments and countries are running out of IP space, or are at least avoiding attaining more, using the bulk of the IP space they own. This problem could however be of less relevance once IPv6 is in widespread use.

8.2.2.2 SPADE AND SPICE: SIMULATED ANNEALING

One of the most ambitious projects that has been proposed to date in scan correlation is the Spice and Spade System by Stainford et al (2002). The system was being built with Defense Advanced Research Projects Agency (DARPA) funding in the U.S. However, they lost their funding because of departmental spending cuts before it was completed. The authors have now moved on to other projects.

The work that had been proposed, and partially implemented, by Stainford et al (2002) was intended to correlate scan activity and would not only classify the user as a scanner, but would also be able to link them to past scan activity. The goal of such a connection would be to link users who operate over a long time, under multiple source IP addresses, to avoid detection. This is precisely what is required in a system that identifies and tracks reconnaissance activity. The system they proposed operates with two components, a sensor (Spade) and a correlation engine (Spice).

The proposed system involved Spade feeding events into Spice, along with an anomaly score it had generated based on the source IP's activity (the negative log of the probability of the event occurring). Spice then places the event in a graph, noting the various properties of the event, such as source IP, target IP, target and source port and time. The location at which the item is placed into the graph is decided by using a search

algorithm called Simulated Annealing. Prior to each event being added, the graph is searched to locate the best location for it, placing it near events of a similar nature. It is here that past scans and events can then be correlated to determine whether a source IP has been changed. It could also be used to decide on the correct response to the user's activity. However for the reasons mentioned above, Spice was never implemented.

The approach appears to have significant potential and its results would have been very interesting if it had been implemented. The paper which detailed the proposal mentioned how it could be of great interest to implement it in a distributed fashion, allowing for a more robust system that would scale to very large networks. While this work was not completed it demonstrated a possible advance in the methods for scan correlation, and is a relevant pre-cursor to the work reported in this thesis. It is also an example of a related work that was examining profile behaviour in a similar way to that which is under taken in this research in relation to object drift. While that is not a term that is used by these authors, what they describe does fit within the bounds of the definition of object drift laid out in Chapter 5.

8.2.3 PREVIOUS RESEARCH

As mentioned above, DynamicWEB was proposed in response to some previous work described in Scanlan et al (2004). This research investigated whether it was possible to detect malicious IP addresses scanning multiple gateways upon the same network. If

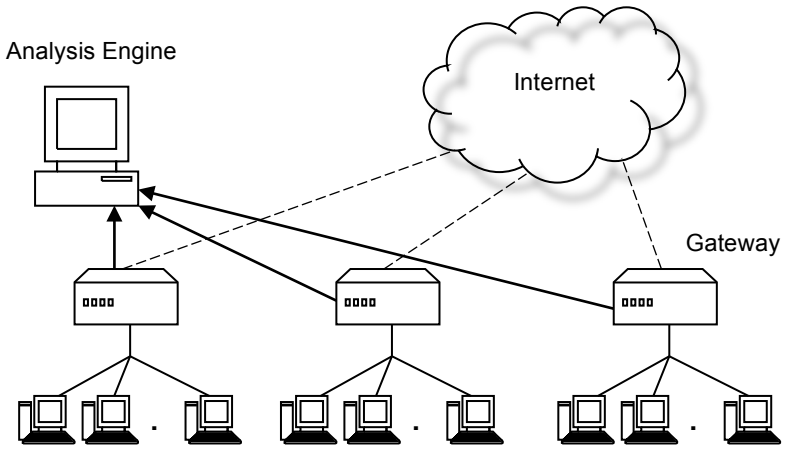


Figure 51. The network which was examined in the research contained multiple gateways. The analysis log covered this range of gateways

detected, the source IP address would then be blocked across all the other gateways on the network before it actively probed each of them. By detecting the activity and pre-emptively blocking the IP addresses, the malicious scanners were denied the opportunity to probe the whole network. The source IP addresses were then unable to map out populated IP addresses within the network, and to learn any more information, or attack them. Three audit logs, of varying lengths up to a month long, were analysed in this work. Each audit log was a centralised log covering a range of gateways, as illustrated in Figure 51. More details about each of the logs are shown in Table 31.

It is a trivial process to detect whether a source IP address probes more than a single gateway within a network once all gateways logs are amalgamated. In Table 31 the number of unique IP addresses that probed the network is shown. Across the three different audit logs, between 11.5 and 12 percent of source IP addresses probed more than a single gateway within the time span of the log. The port scanners who are interested in multiple gateways are in the minority. Therefore the key to being able to detect and react in a timely fashion is to detect source IP addresses probing multiple gateways in a scalable fashion.

	Single Gateway	Multiple Gateway	Length
Log 1			10 Days
Source IP	5990	776	
% of Total	88.5	11.5	
Log 2			20 Days
Source IP	67029	8948	
% of Total	88.2	11.8	
Log 3			30 Days
Source IP	77431	10467	
% of Total	88	12	

Table 31. The three logs covered periods of 10, 20 and 30 days. Each log recorded between 11.5 and 12 percent of source IP addresses probing multiple gateways within the network.

To enable this to occur, it is necessary to locate a maximum threshold period of time or activity, in which to monitor a source IP address. If, in this time, the source IP was judged to be a threat across multiple gateways then it could be blocked from the whole network. If it wasn't classified as such a threat in the time period then it would be

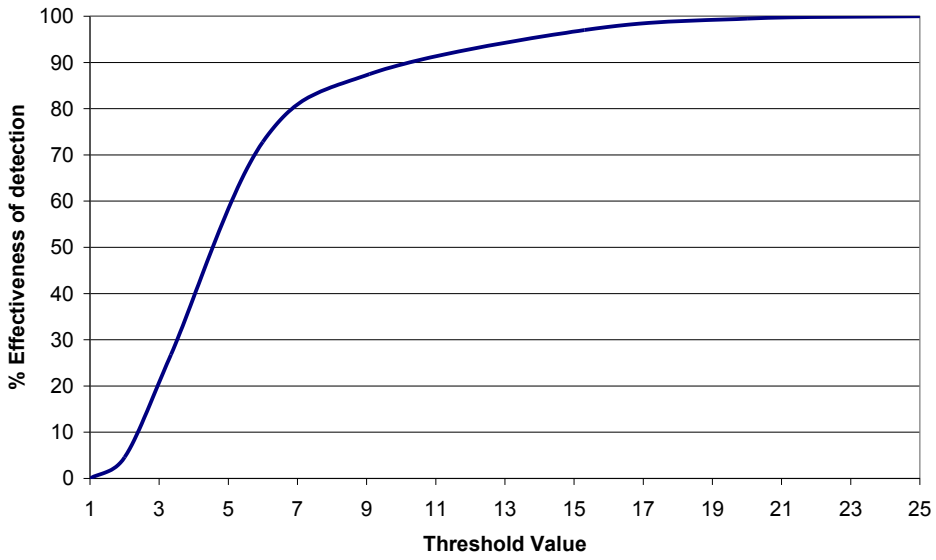


Figure 52. Illustrates how long multiple gateway scanning source IP's scan a single gateway before moving to the next.

assumed they the source IP was only interested in a single gateway (on which it had already been blocked).

Figure 52 demonstrates how many probes were sent before a given IP address would then probe a second gateway within the network. As can be seen from the graph (based upon data from Log 1) the vast bulk of scanners interested in multiple gateways would only probe a single location 6 or 7 times before moving on. Ninety percent of multiple gateway probing sources in Log 1 had moved on by their 10th scan. Within Log 2 and 3 the probing sources attempted one or two probes less, possibly due to the increased length of the datasets allowing for more time for sources to probe another gateway. A threshold value of 11 was chosen as an effective value to be used. After 11 probes to a single gateway, the analysis engine stopped following data relating to the given source address. Up until that point, if any sources probed multiple gateways then the action module was notified and the IP was blocked from accessing the whole network. Once an IP was blocked across the gateways it was shown that it would quickly cease its activities, as shown in Figure 53. This work is discussed at more length in Scanlan et al (2004).

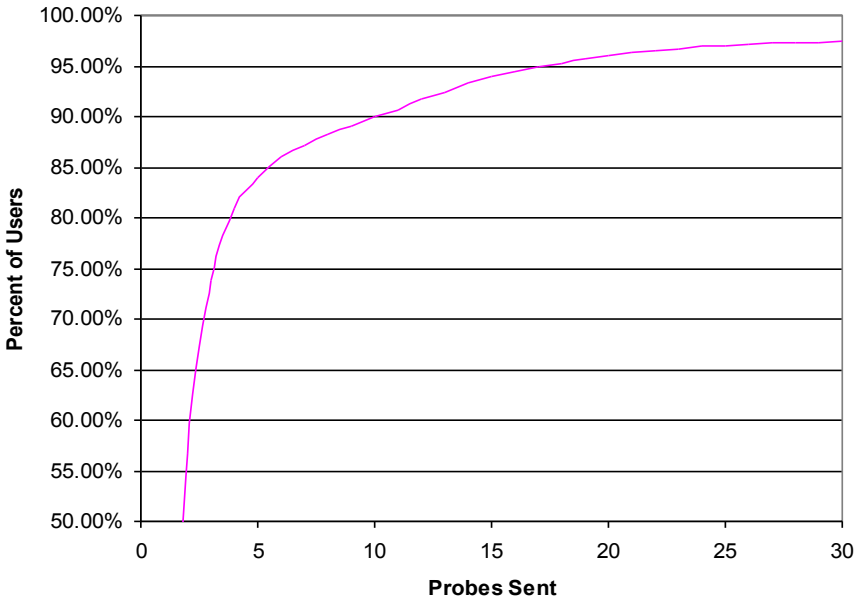


Figure 53. Shows how long a scanner continues to scan after being blocked across the whole range of gateways.

8.3 THE PROBLEM

Most of the port probes present within the audit logs are either benign in nature, or are single target host probes. The latter are simply blocked and ignored, and the source never probes a different target IP within the log. Source IP addresses that target multiple gateways on the network were examined and blocked from accessing the whole network, including via gateways they did not probe.

However, the authors unexpectedly found, also within the audit logs, an unusual scenario, which is illustrated in part in Table 32. The table shows four different sets of port scans, conducted by four different source IP addresses. Each address targeted multiple different gateways with the timed gaps between probes shown above. The gaps are markedly different from the rest of the scans within the audit logs, which represent 60 days of traffic overall. Furthermore, there are quite a few more examples which are also quite different from the remainder of the scans within the log. The source IP's in each of these examples are all from the same Class-C address range, as are most of the other examples found, with only a few coming from a neighbouring Class B range.

Probe #	IP1	IP2	IP3	IP4	Mean (sec)	STD (sec)	%
1	---	---	---	---			
2	3	3	3	3			
3	85757	85757	85789	85742	85761	19.81	0.02%
4	3	3	3	3			
5	21527	21519	21484	21536	21517	22.75	0.11%
6	3	3	4	4			
7	21637	21641	21682	21621	21645	25.98	0.12%
8	3	3	3	3			
9	21564	21587	21521	21579	21563	29.42	0.14%
10	3	3	3	4			
11	21587	21592	21586	21588	21588	2.63	0.01%
12	3	4	3	3			
13	21619	21618	21637	21616	21623	9.75	0.05%
14	4	3	3	3			
15	21569	21594	21596	21592	21588	12.61	0.06%
16	3	3	3	3			
17	21591	21593	21580	21596	21590	6.98	0.03%
18	3	4	3	4			

Table 32. Four different sets of 18 probes each originating from a different IP address. The gap between the probes is shown above in seconds. In addition to these gap times, the mean and standard deviation of these gaps are also listed.

It is a distinct possibility that these source IP addresses all represent the same user, and that the user in question is systematically mapping the network from different source IP addresses in an attempt to avoid detection. The goal of DynamicWEB here is to discover any relationships between the port scan profiles of different source IP addresses. As each source IP conducts another scan, its profile is updated and compared with other source IP profiles. This problem is one where relationships between many target objects need to be extracted in the confines of a dataset that contains object drift and concept drift. The dataset is ordered over time, with many observations of each target object, requiring that their observation histories need to be preserved.

8.4 PROFILING PORT SCANS

When profiling the activity of port scanners, there is only a limited amount of data known about the user who is scanning a given computer. Their IP address is known, but this is quite possibly fraudulent, making it unreliable as a data source. Here it is used as a unique identifier for the profile. The port that they are probing is known, although,

again, most probes are of the same ports, Previous work described in this thesis examined whether or not a malicious user probed more than a single gateway on the wider network, and so we also have that information. However, the main knowledge that can be ascertained about an attacker, in this scenario, is that detailed in Table 32: the timings of their port scans. This information can be used to build a profile of each scanning IP within Log 3 shown in Table 31. The content of the profile is shown in Table 33.

1	Source IP	The Source IP address which sent the port scan.
2	Target IP	The Target IP address for the port scan.
3	Port	The Target Port.
4	Gap	The gap in seconds since the last port scan from the source IP.
5	Mean (of gap)	The mean of the gap between all scans from the source IP
6	Standard Deviation (of gap)	The standard deviation of the gap between port scans.
7	Multiple Gateways	A Boolean value indicating whether or not this host probed multiple gateways.

Table 33. The Attribute value pairs within the dataset examined. Several are derived and are created by DynamicWEB. These are updated as new data for the profile is observed.

The timing of the scans is represented within the profile as the gap (Attribute 4). These represent the amount of time from the previous scan to the most recent. The context of all of the scans is then stored, including the values of the derived attributes, *Mean* and *Standard Deviation* (Attributes 5 and 6). In addition to these, a Boolean derived attribute, indicating whether or not the scanner has previously scanner other gateways, is also stored. This value is updated whenever a new observation is incorporated into the profile. The aim here is to profile those users who behave in a strange manner, conducting scans over a long period of time in an attempt to hide their network mapping activities.

Shown in Figure 54, is a cluster representation generated by DynamicWEB[10] from audit log 3. The audit log covered over 10,000 unique IP addresses, with 1200 source addresses targeting multiple gateways. Of those multiple gateway probing IP addresses, there appeared to be at least 15-20 groups of IP's addresses that stood out as fitting a pattern quite different from the rest of the scans present. These were extracted from the database created by the previous threshold-based system and stored for manual comparison with the output from DynamicWEB. The individual IP addresses are not show here for privacy reasons.

The tree, shown in Figure 54, displays a large cluster of 1180 unique IP addresses as a child of the root. The remainder of the tree contains the 57 other IP addresses in the dataset. The tree contains only those IP addresses that probed multiple gateways, with the singe target hosts having been removed heuristically. The IP addresses not present within the large cluster are those which differ sufficiently to be recognised as not being an ordinary port scan. A large proportion of the nodes in this half of the tree contain just two IP addresses. These IP's are quite similar in behaviour to each other. It is these IP addresses that we are suggesting represent the same end user. Sibling leaf nodes are also usually very similar to each other, and are prime candidates to be linked together after more activity has been witnessed. This structure was visible within the dataset after 2000 observations; with the smaller branch only containing 9 profiles at that point.

This figure shows that progress has been made towards developing relationships between the source IP addresses which are much more meticulous in their examination of the network. The vast majority of the scanners who probe each of the gateways on the network probably carry this out in a fashion similar to that shown in Figure 50 where they scan vast amounts of IP space. These scanners make up the bulk of the leaf node with 1180 IP addresses inside it and so have been successfully separated from the IP's of interest.

[10] Manually examining 1200 IP addresses for relationships between them is a much smaller task then the full 10,000 multiple gateway scanning IP addresses.

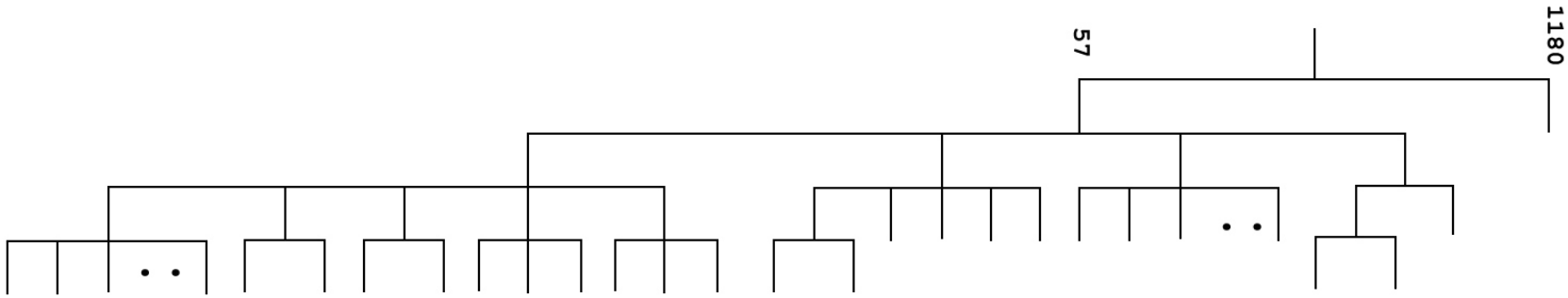


Figure 54. The clustering tree created by DynamicWEB from the audit log. The top level division separates the behaviour profiles that are all very similar (1180 profiles) from those with greater difference in behaviour (57 profiles).

If DynamicWEB were to be run upon a live system now it would be possible to ascertain, when an update has occurred, whether the given IP has been clustered with another scanner IP. The administrator could easily be notified of the relationship, and a closer monitoring of the IP's activities could occur.

However, the 57 IP addresses identified by this process do not include all of the profiles of interest that we discovered upon manual inspection. Furthermore, some IP's included in the 57 were not flagged by the manual inspection and could possibly be linked. This result implies that we have not extracted all of the possible linked profiles from within the dataset. Examining those that were discovered, we estimate that the method is detecting about 60-70% of the profiles. While this number is quite low, the process is detecting this group of 57 from within a total group of IP addresses numbering over 10,000, which is a little over half a percent of the dataset. As such this result is seen as a very positive one and a step forward in the area of profiling scan activity. It may be that this method requires further derived attributes or more input data from other network based sensors to increase its detection level.

8.4 AUSTRALIAN BUREAU OF STATISTICS NETWORK PERFORMANCE

Network monitoring can be undertaken for reasons other than just the provision of security from an intrusion detection or prevention perspective. For example network monitoring can be used to monitor the quality of a service. In addition to the dataset examined in the first half of this chapter, a private network performance log was sourced from the Australian Bureau of Statistics (ABS). This dataset includes several performance measures and other details for a large number of computers that reside on the ABS's private network. This audit log of performance activity is not a publically available log, as are the two datasets presented in Chapter 7, but is a private log which was provided by the ABS and used with permission (Edmondson 2009).

The audit log covers a large number of computers located across several floors of buildings used by the ABS, and also across multiple locations around Australia. This audit data is used by the ABS IT department to identify bottlenecks and other performance issues on their network. If a single computer is experiencing issues,

then that may indicate that it has been compromised or has mis-configured applications or services that require attention. If a group of computers at a given location have displayed a change of behaviour then perhaps a router or switch that they share has a problem which is affecting all of them. The audit log contains an entry for each time that a user has logged onto a computer across the ABS network over a 22 day period in late 2008. There are over 36,700 recorded entries in the audit log, covering 3,000 computers. The attributes that are covered within the dataset are outlined in Table 34.

#	**Name**	**Description**
1	Record Identifier	Number
2	Global Timestamp	Timestamp at the central recording location
3	Local Timestamp	Timestamp at the local computer
4	User Identifier	ID of the user logging onto the computer
5	Workstation Identifier	ID of the computer being logged into
6	Location	Current location of the computer. There are a total of 23 locations in the dataset.
7	Operating System	The Operating system in use on each computer. Although they are all using Windows XP.
8	Desktop/Laptop	The kind of computer: Desktop or Laptop
9	Brand of Computer	Dell, HP, Acer, IBM, Toshiba, VMware
10	CPU (Mhz)	Ranges from 500 to over 4000.
11	RAM (mb)	Ranges from 512 to 4096.
12	Time to Load Shell (sec)	Time taken for the computer to load
13	Time to Map Resources (sec)	Time taken to map network resources
14	Total Login Time (sec)	Time taken for the user to login on the computer

Table 34. The attributes which are present within the ABS Network Performance dataset.

Most of the attributes in the dataset describe either the computer that the entry is being recorded for, or the point in time that the recording took place. However the final three attributes within Table 34 specify the length of time required for three specific tasks to be completed. These measure the actual performance of the target computer. The poorer that network performance on a given computer, the larger these values are. The user and computer identifiers are matched to the same entity, but are randomised, so there is no bias within the dataset to assigning certain identifiers to given locations.

This dataset presents several learning goals to test DynamicWEB's ability to extract structure. Within the dataset there are both laptops and desktop computers. Is DynamicWEB able to produce a model that can correctly predict between these two classes? There are also several brands of computers on the network. Is DynamicWEB able to differentiate between them? Lastly, the dataset covers computers spread across a range of geographically separate locations. Does this affect the results and is DynamicWEB able to extract a structure that can predict the location of a computer? These problems will be discussed below.

The first learning problem to be examined here is whether DynamicWEB is able to learn a structure that is able to predict whether a piece of computer hardware is a desktop or laptop. This is a two-class prediction problem with an approximate 3:1 split between the two classes, with desktops being the more common. In examining this problem the dataset was split into three, allowing for cross validation to occur to estimate the average performance across 6 different combinations of training and testing data. The division was carried out so that all records relating to a given computer were found in a single dataset. Furthermore, the datasets were made up of recorded values spread through the entire 22 day period of the audit log, to minimise any effects of network topology changes by ensuring that they affect all datasets equally. Each dataset covers just over 1000 computers, and about 12,000 records.

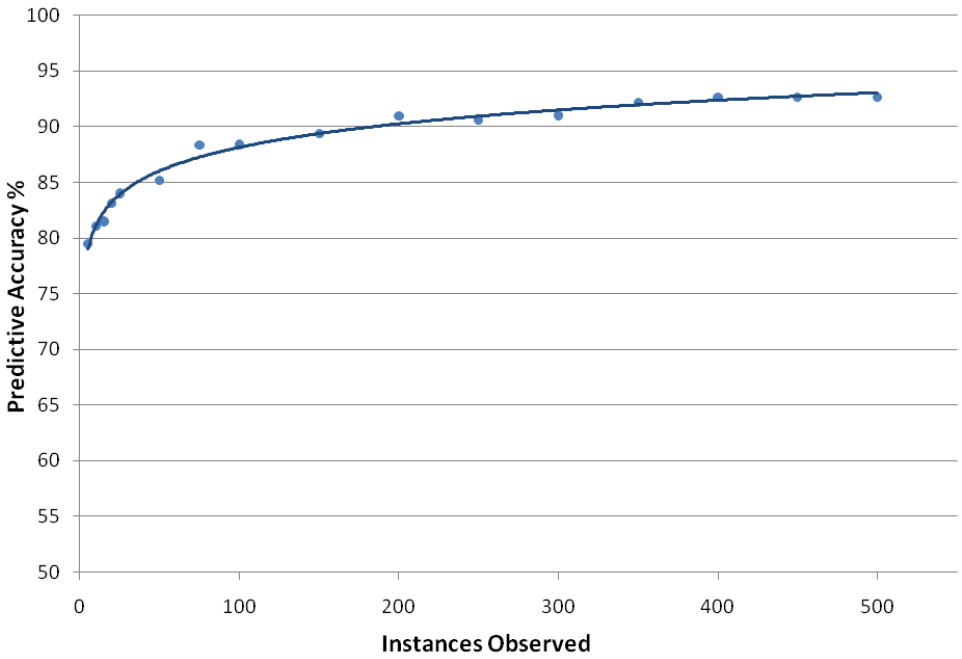


Figure 55. The performance of DynamicWEB at predicting whether or not a computer is a Laptop or a Desktop.

The results by DynamicWEB, when learning this two-class problem, are shown in Figure 55. DynamicWEB learns the problem moderately well after about 50 observations, achieving an 85% predictive accuracy across the two 12,000 strong testing sets, with an accuracy of over 90% being achieved after 200 observations. The method continues to improve slowly after that, achieving an accuracy of 93% after 500 observations. The is a good result considering that the individual computers are spread out around a large network, connecting different locations, with different network paths to the central location, where the recordings are being made. However, as it is a two-class problem with a notable class imbalance it is important to also examine the performance of the two classes independently. In Figure 56 the

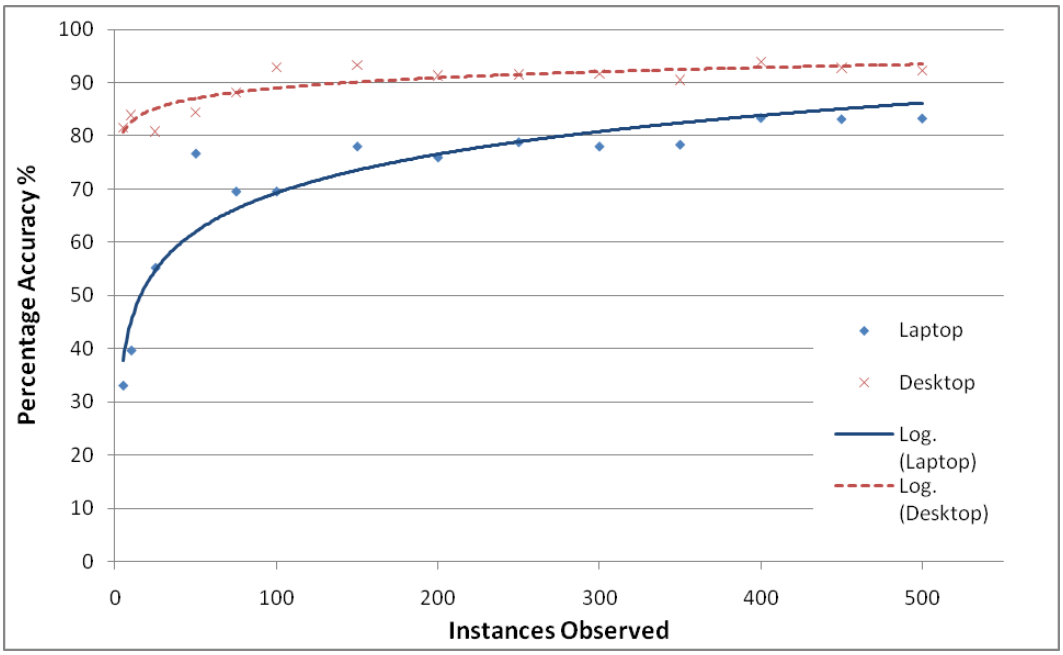


Figure 56. Comparison of the predictive performance of DynamicWEB on the Laptop and Desktop classes.

predictive accuracies of the two classes are compared. As one would expect, the method is more effective in predicting the more populous Desktop class, with it being correctly predicted in over 80% of cases over the entire experiment. However, while the Laptop class is predicted less accurately, initially, due to the fact that there are few of the training observations of that class, its predicted accuracy reaches 70% after 100 observations, and then reaches approximately 85% by the 500th training observation.

In addition to examining whether DynamicWEB was able to predict the type of computer, its ability to predict the brand of computer was also examined. The dataset contains 7 different brands of computer (or device): Dell, Acer, IBM, Toshiba, VMWare, Hewlett Packard, and IPX. Figure 57 shows the accuracy achieved by DynamicWEB when predicting the brand of computer, based on the timings and computer specifications. DynamicWEB achieved an accuracy of over 90% after 10 observations and was able to reach 95% after 250 observations. However, this performance, especially towards the start with both very high predictive accuracy and marked variability, highlights the uneven distribution of the dataset. The vast bulk of the dataset is actually comprised of two brands of computer: Acer and Dell. The other brands make up less than 1% (0.7%) of the total dataset.

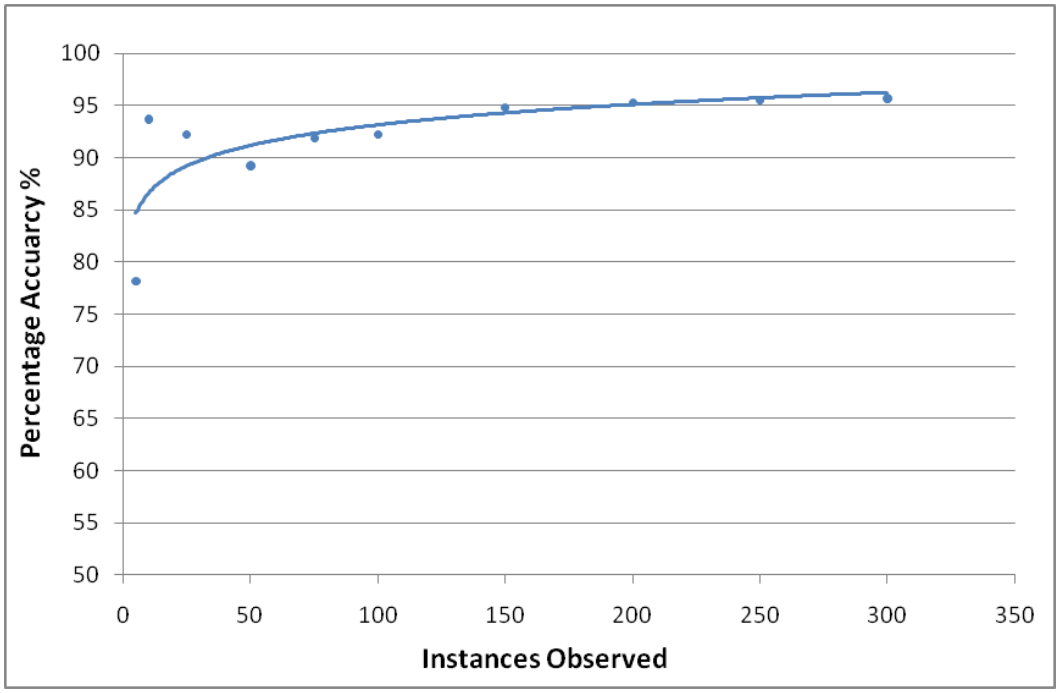


Figure 57. The predictive accuracy of DynamicWEB in predicting the brand of computer.

The distribution ratio of the two main classes is 3:1, with Dell being the more common of the two classes. In the results shown in Figure 57 the other 5 classes that make up less than 1% of the data are all predicted incorrectly. This is due to their rarity and the fact that each brand is not represented in all of the folds. However, the predictive accuracies for the main two classes are shown in Figure 58. A comparison between the accuracies achieved by DynamicWEB, when predicting the two target classes, shows that it achieves a higher accuracy with the Dell class than with the less frequent Acer class. The Dell class is predicted with over 95% accuracy after only 5

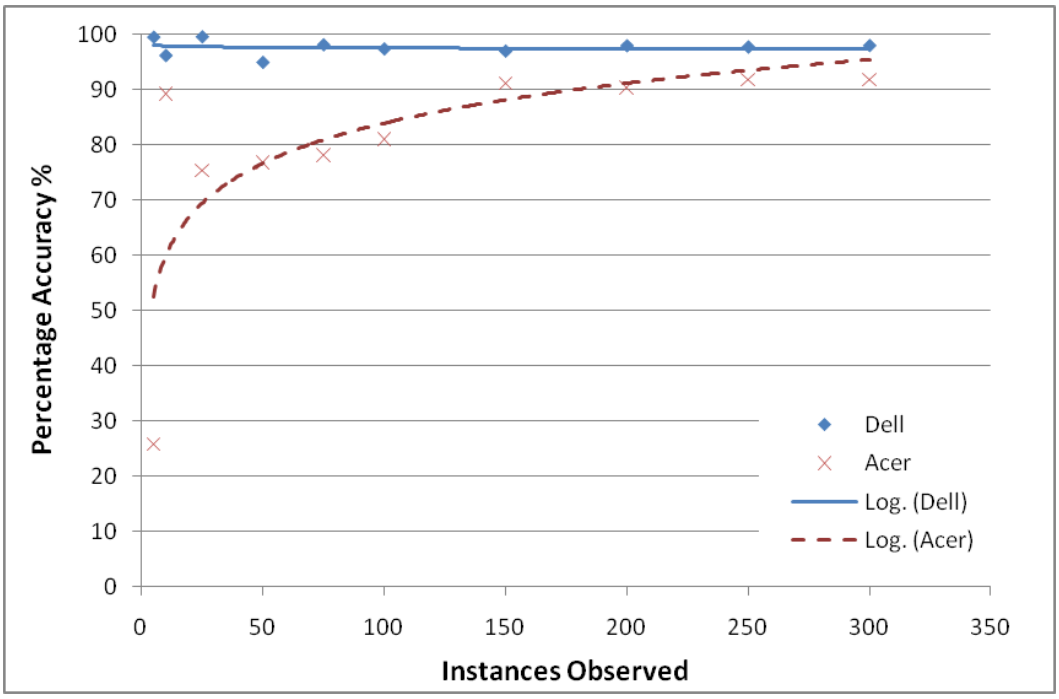


Figure 58. Comparison of predictive accuracy of DynamicWEB on the Dell and Acer classes.

observations, while the Acer class is predicted with 90% accuracy after 200 observations.

The third attribute to examine and then aim to predict with DynamicWEB is that of the location of the computers. The dataset covers a range of computers spread across each of the states or territories of Australia, along with 2 large multi-story buildings located in Canberra. When examining this dataset, however, it was found that DynamicWEB struggled to predict computer location to an acceptable accuracy. Upon finding this and then consulting with Dr Edmonson (2009) it was found that the time measures are standardised in a way that would allow for structure, with these measures occurring both interstate and locally, depending on the resources being requested at the given time. In addition to this, two different states, both communicating with a 3rd state, from which they are each a similar distance, could appear to be the same to each other. Due to these complications, this class can't be predicted, but it is worth noting that these time measures add a significant noise factor to the dataset which DynamicWEB was able to handle effectively when predicting other classes

Both of the examples, relating to this dataset, initially used only the most recently observed value within the profile. Derived attributes were trialled with near identical

results returned. This indicates that the knowledge domain is very regular for the two problems examined, even with the presence of a noisy location attribute. The next step with this domain would be to use DynamicWEB as a monitoring system aiming to detect performance anomalies on the network. The dataset that was used in this research did not contain any known anomalies to detect, so the next step would be to synthesise and examine these. To do this, more knowledge is needed to determine what form anomalies normally take, in order to be able to synthesise them in a meaningful way. While this extra research has not been completed, the research that has been undertaken showed that DynamicWEB did perform acceptably on the dataset, and that such a system would be able to return some interesting results.

8.6 SUMMARY

DynamicWEB has been applied to two datasets derived from computer network systems. The first related to the problem which inspired DynamicWEB in the first place: port scanning. DynamicWEB was shown to be capable of profiling scan activity, enabling relationships to be drawn between different port scanners. This was achieved by using derived attributes to preserve the history of past scans undertaken by each scanner. This problem presented a true profiling scenario, and DynamicWEB was able to separate out less than 1% of the dataset by building profiles over time. While it was estimated that only 60-70% target scanners were found, this was still seen as a positive result for the unsupervised learner to be able to glean from the 30 days and near eighty thousand observations in the dataset.

Secondly, DynamicWEB examined the performance data from a large, widespread, computer network. In examining this dataset, DynamicWEB was able to predict the brand and type of computer, based on its specifications and the time it required to complete certain specified tasks. This showed that it was able to learn the knowledge domain to a point where is possible that, if it was fully applied to the knowledge domain in the form of a network monitor, it would be able to detect anomalies.

In this chapter, two datasets were examined covering vast amounts of data and long time periods. DynamicWEB was able to examine this data and extract patterns from the observations. DynamicWEB has shown itself to be a capable online learner that is able to function on large datasets. It would however need more research on either

of these domains before it could be fully applied in a successful application, but it has shown that it does contain the potential to fulfil a role in a network security or network monitoring domain.

In the context of the learning goals outlined in the introduction, this chapter has demonstrated that profiling of the behaviour of a target object was able to take place across multiple observations. In these datasets there is a sizable variation by some of the target objects, which could be considered to be object and concept drift. Both datasets also dealt with a sizable number of target objects and was able to establish relationships between them.

Chapter 9

Conclusions and Further Work

"Change is the law of life. And those who look only to the past or present are certain to miss the future."

John F. Kennedy (May 29, 1917 – November 22, 1963)
Address given in Frankfurt, June 25, 1963

INTRODUCTION

The primary aim of this research was to develop an unsupervised machine learning method capable of profiling the activity of a target object across multiple observations over long time periods on large amounts of data in real time. This is a problem which is relevant across many knowledge domains and has a high applicability in applied machine learning. The work carried out to undertake this research, which resulted in the development of a new learning method, will now be summarised and results from tests that were conducted using this method will be reported. After this, the further directions that this work could take will be explored.

9.1 RESEARCH CONTRIBUTION

The main contribution that has been made by the research presented in this thesis is a new machine leaning method entitled DynamicWEB. This method was required in order to meet the six needs that were outlined in the introduction and provides the following six capabilities:

1. The learner is able to profile object activity over an extended time period.
2. The learner is able to establish relationships between the profiles.
3. The learner is able to adapt to concept drift.
4. The learner is able to adapt to object drift.
5. The leaner is able to preserve context across multiple observations.
6. The learner is able to be able to track a large number of target objects simultaneously in real-time.

Existing methods in this area of research are largely supervised learners or are batch-based approaches and aren't able to build a profile over time of a single target object. DynamicWEB is presented as a method that builds profiles across multiple observations of a set of target objects and is an unsupervised learner. This online approach allows DynamicWEB to operate on a stream of data within time sensitive contexts. The method is a hierarchical probabilistic conceptual clustering learner built upon COBWEB (Fisher 1987; Gennari, Langley et al. 1989). COBWEB is a

respected method in the machine learning field and, as such, is seen as a solid foundation to build upon. An index structure was added to COBWEB to enable fast search that facilitate the addition of new operations. This index structure is an elegant solution to enable the learner to have a scalable search able to meet the size requirements of large datasets containing many objects of interest (6).

Two new operations were added in DynamicWEB to those already present in COBWEB: Remove and Update. These operations were implemented with care to maintain the integrity of the existing concept hierarchy. As change takes place within the concepts over many observations, it is vitally important to maintain the quality of the concepts that are being learned. This allows DynamicWEB to tackle learning domains which COBWEB was incapable of examining, while still remaining true to the theoretical base on which it was founded.

Once the hierarchy was modified to enable profiles to be added and removed, DynamicWEB had the capacity to update profiles, combining data from multiple observations. This allowed for profiles, stored within the hierarchy, to contain the most recently observed data, and by enabling the re-addition of profiles to the hierarchy, allowed the learner to adapt to any changes in any of the profile. By allowing for changes to take place within the concept hierarchy, concept drift or object drift can be accommodated by the learning process (3 and 4). It is DynamicWEB's ability to operate in the presence of these twin forms of drift that sets it apart from other machine learning methods.

Once DynamicWEB was provided with the ability to update profiles when new observations occur, it was possible to preserve past observations by using derived attributes to store historical context (1 and 5). This allowed the learner to become aware of trends occurring in relation to each observed object. DynamicWEB's ability to preserve context allows it to profile the behaviour of an object over a number of observations. This is a simple idea for a learner to aim to undertake, however, it is an ability that the bulk of machine learning methods are unable to carry out. In addition to preserving context, the derived attributes also improve the ability of the learner to handle noise in the dataset because they enable the profile to contain some attributes that have a smoothing effect over past observations.

DynamicWEB, with the ability to profile objects over multiple observations, is able to establish relationships between the profiles created (2). The hierarchy that is produced relies upon the previous work carried out in COBWEB and CLASSIT, but builds upon this foundation to learn in a totally different environment than that in which these methods were intended. Using these relationships, which are constantly being updated, DynamicWEB's concept hierarchy is able to be used in several many different learning tasks. It can be used to discover patterns between different activity profiles, or it can be used to make classifications in real-time. A summary of the results that were produced using DynamicWEB in this research will now be discussed.

9.2 RESULTS SUMMARY

After DynamicWEB was introduced in Chapter 5, the remainder of the thesis described the results of applying the method to a number of different knowledge domains. An important aim of the research was to ensure that DynamicWEB would be suited to a range of application domains. While its original innovation derived from a single application (port scanning reconnaissance profiling), for it to be truly useful it needs to be more broadly applicable.

After initially confirming that the COBWEB implementation used within DynamicWEB produced comparable results to those produced by COBWEB itself, several small machine learning datasets were examined. DynamicWEB was then demonstrated on the Dynamic Weather dataset, which was created specifically for that purpose. This dataset illustrates several examples of object drift in an environment where the concepts are themselves not strictly defined.

The Quadruped Animals dataset was examined to illustrate DynamicWEB's ability to function as an ensemble learner. Here, using the "wisdom of the crowd", DynamicWEB was able to produce improved classification accuracy by splitting the attributes in the dataset across eight trees, derived in parallel. The final machine learning dataset examined was the STAGGER Concepts dataset. This dataset demonstrates concept drift, with three distinct concepts being present within the dataset. The dataset has been used by multiple authors and was used when comparing the performance of DynamicWEB with that of COBBIT. DynamicWEB

demonstrated an improved performance compared with COBBIT on a single ordering of the data. DynamicWEB also performed well, in comparison with two supervised learners, when tested over 100 trials.

After the validation was completed, several real world data mining datasets were examined (Chapters 7 and 8). These spanned several application domains, and included a combination of datasets from the Australian Bureau of Statistics (ABS) as well as a data mining workshop dataset (Chapter 7). All of these datasets are real world datasets and most have not been examined using machine learning techniques prior to this research.

Several of the datasets did not have existing class labels so DynamicWEB was used to examine them to see if any structure could be extracted from within the dataset. In some cases this was not possible and the results relating to those are recorded in the appendices. In cases where DynamicWEB was able to discover structure, it had been assisted by the use of derived attributes which, preserving the context over time, thus allowed for trends within the data to be discovered.

Real world datasets were also used to evaluate DynamicWEB's predictive ability. DynamicWEB demonstrated an acceptable level of success when predicting the class distribution in the "watching TV" class of the BodyMedia dataset (2004). On the ABS network performance dataset, attribute values for some nominal classes (type and brand) were predicted reasonably accurately. It was concluded that DynamicWEB could potentially be applied within the domain in order to locate computers that were behaving abnormally.

The original inspiration for this research was the port scanning problem, in which users change IP address whilst undertaking scan activities, in order to avoid detection. Examination of the port scanning dataset (Chapter 8) showed that groups of similar port scan profiles, with scans stretching across large time periods, could be revealed by the unsupervised learner and then compared with each other to establish relationships between them. A key complication of this problem which aided in focuses the research that was undertaken was that these scan profiles change over time; with profiles being built and being considered benign to then being of interest (object drift) or the possibility of behaviours changing over time (concept drift). The

method described here, as demonstrated first on the smaller datasets earlier is able to adapt under these learning difficulties. The scan dataset was also shown to be able to extract relationships between profiles further illustrating DynamicWEB's ability to meet these challenges.

9.3 FUTURE DIRECTIONS

This thesis described DynaimicWEB and discussed the needs of the application for which it was originally designed. However, this thesis does not detail a full testing on that knowledge domain, instead opting to confirm that DynamicWEB is a useful approach across a broad range of domains. Therefore, more work could be completed evaluating the method not only for port scanning, but also for dealing with other security problems. The eventual goal is for it to be used in a live system.

Indeed DynamicWEB could be applied across a range of different domains, as it has shown itself as a very capable learner. In discussions about the performance of DynamicWEB, Dr Edmondson of the Australian Bureau of Statistics (Edmondson 2009) stated that it would be highly useful if DynamicWEB was integrated with a visualisation package that allowed for the concept descriptions to be easily read and monitored over time. Such an extension would be of significant interest because it would enable people to use DynamicWEB across a range of learning domains, simply and easily.

In addition to these application-specific advances, from a theoretical machine learning direction, it could also be worthwhile to examine further improvements to DynamicWEB. For example it could be interesting to incorporate the ARCHANE (McKusick and Langley 1991) control structure advances to see if any learning gains are achieved. In addition to this, some work was carried out, although not completed, which aimed to use multiple DynamicWEB hierarchies in a layered manner. Different knowledge thresholds in each tree then produce increasingly more specific relationships between profiles, acting as a multiple stage filter. Similar work to this has been carried out by other authors (Bala, DeJong et al. 1995; Li, Holmes et al. 2004) previously.

This research has highlighted a gap in the machine learning field, and DynamicWEB is the first learner to attempt to address this gap. There is scope for other methods to

be produced to attempt to profile objects across multiple observations, allowing for the target objects to drift from one resultant class to another. There is also scope for other applications to be examined in the context of object drift,. This thesis has attempted to describe the problem broadly, to act as a platform not only for DynamicWEB to be used in multiple applications, but also to inspire other researchers to develop machine learning approaches that adapt in real-time to changes in input data. .

10

References

(ABS), Australian Bureau of Statistics. (2009). Labour Force, Australia. **http://www.abs.gov.au/ausstats/abs@.nsf/mf/6202.0**.

ABS, Australian Bureau of Statistics. (2009). Australian National Accounts: National Income, Expenditure and Product. **http://www.abs.gov.au/ausstats/abs@.nsf/mf/5206.0/**.

Adelson-Velskii, G. and E. Landis (1962). "An algorithm for the organization of information." Doklady Akademii Nauk SSSR(146): 263-266. (Russian). English translation by Myron J. Ricci in Soviet Math. Doklady, 3:1259–1263, 1962.

Amoroso, E. G. (1998). Intrusion detection : an introduction to Internet surveillance, correlation, traps, trace back, and response. Sparta, N.J., Intrusion.Net Books.

AnderBerg, M. R. (1973). Cluster Analysis for Applications. New York, NY, Academic Press, Inc.

Azé, J., N. Lucas and M. Sebag (2004). A Genetic ROC-based Classifier. Twenty-First International Conference on Machine Learning. Banff, Alberta, Canada.

Bailey, M., E. Cooke, F. Jahanian, Y. Xu and M. Karir (2009). A Survey of Botnet Technology and Defenses. Cybersecurity Applications & Technology Conference For Homeland Security(CATCH), Washington, DC, USA, IEEE Computer Society.

Bala, J., K. DeJong, J. Huang, H. Wechsler and H. Vafaie (1995). Hybrid learning using genetic algorithms and decision trees for pattern classification. . International Joint Conference on AI (IJCAI-95). Montreal, Quebec**:** 719-724.

Ball, G. H. and D. J. Hall (1965). ISODATA: A Novel Method of Data Analysis and Pattern Classification. Technical Report. Menlo Park, CA, Stanford Research Institute.

BodyMedia (2004). Physiological Data Modeling Contest, ICML-2004 Workshop**:** http://www.cs.utexas.edu/~sherstov/pdmc/.

Breiman, L. (1996). "Bagging Predictors." Machine Learning **24**(2): 123-140.

Bruner, J. S., J. J. Goodnow and G. A. Austin (1956). A study of thinking. New York, John Wiley & Sons.

Corter, J. and M. Gluck (1992). "Explaining basic categories: Feature predictability and information." Psychological Bulletin **111**(2): 291-303.

Dubes, R. C. (1987). "How many clusters are best?-an experiment." Pattern Recognition **20**(6): 645-663.

Edmondson, R. (2009). Personal Communication, Head, Technology Services (Tas), Australian Bureau of Statistics, Hobart.

Feigenbaum, E. A. and H. A. Simon (1984). "EPAM-like models of recognition and learning." Cognitive Science **8**(4): 305-336.

Fisher, D. and P. Langley (1985). Approaches to conceptual clustering. Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, Morgan Kaufmann.

Fisher, D. and P. Langley (1990). "The structure and formation of natural categories." In G. H. Bower (Ed.), The psychology of learning and motivation: Advances in Research and Theory (Vol. 26).

Fisher, D., L. Xu, J. R. Carnes, Y. Reich, S. J. Fenves, J. Chen, R. Shiavi, G. Biswas and J. Weinberg (1993). "Applying AI Clustering to Engineering Tasks." IEEE Expert: Intelligent Systems and Their Applications **8**(6): 51-60.

Fisher, D. H. (1987). "Knowledge Acquisition Via Incremental Conceptual Clustering " Machine Learning **2** (2): 139-172

Fisher, D. H. (1987). "Knowledge Acquisition Via Incremental Conceptual Clustering." Machine Learning **2**(2): 139-172.

Forgy, E. W. (1965). "Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of classification." Biometrics **21**: 768-9.

Fuller, W. A. (1996). Introduction to statistical time series. New York, Wiley.

fyodor (2005). Nmap, http://nmap.org/.

Gama, J. and P. Rodrigues (2004). Physiological Data Modeling Contest. Twenty-First International Conference on Machine Learning. Banff, Alberta, Canada.

Gamzu, E. and D. R. Williams (1971). "Classical conditioning of a complex skeletal response." Science **171**: 923-925.

Gennari, J. H., P. Langley and D. Fisher (1989). "Models of incremental concept formation " Artif. Intell. **40** (1-3): 11-61

Gluck, M. and J. Corter (1985). Information, uncertainty and the utility of categories. Seventh Annual Conference of Cognitive Science Society, Irvine, CA.

Gluck, M. and J. Corter (1985). Information,uncertainty,and the utility of categories. Proceedings of the Seventh Annual Conference of the Cognitive Science Society. Irvine,CA, Lawrence Erlbaum Associates**:** 283-287.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten (2009). "The WEKA Data Mining Software: An Update." SIGKDD Explorations **11**(1).

Hansen, P. and M. Delattre (1978). "Complete-link cluster analysis by graph coloring " Journal of the American Statistical Association **73**: 397-403.

Hanson, S. J. and M. Bauer (1989). "Conceptual Clustering, Categorization, and Polymorphy." Machine Learning **3**(4): 343-372.

Hartigan, J. A. (1975). Clustering Algorithms, John Wiley & Sons, Inc.

Hunt, E. B., J. Martin and P. Stone (1966). Experiments in Induction. New York., Academic Press.

Iba, W. and P. Langley (2001). Unsupervised learning of probabilistic concept hierarchies. Machine learning and its applications. V. K. G. Paliouras, & C. D. Spyropoulos. Berlin, Springer.

Jain, A. K., M. N. Murty and P. J. Flynn (1999). "Data clustering: a review." ACM Comput. Surv. **31**(3): 264-323.

Jancey, R. C. (1966). "Multidimensional group analysis." Australian Journal of Botany **14**(1): 127-130.

Jung, J., V. Paxson, A. W. Berger and H. Balakrishnan (2004). Fast Portscan Detection Using Sequential Hypothesis Testing. IEEE Symposium on Security and Privacy.

Kilander, F. and C. G. Jansson (1993). COBBIT - A control procdure for COBWEB in the presence of concept drift. Proceedings of the European Conference on Machine Learning. P. B. Bradzil, Springer Verlag.

Klinkenberg, R. (2004). "Learning drifting concepts: example selection vs. example weighting." Intelligent Data Analysis, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift **8**(3).

Kolodner, J. L. (1983). "Reconstructive memory: A computer model." Cognitive Science **7**(4): 281-328.

Kolter, J. Z. and M. A. Maloof (2003). Dynamic Weighted Majority: A New Ensemble Method for Tracking Concept Drift, IEEE Computer Society.

Kubat, M. and J. Pavlickova (1991). The System FLORA: Learning from Type-Varying Training Sets. Proceedings of the European Working Session on Machine Learning, Springer-Verlag.

Lebowitz, M. (1986). "Concept learning in a rich input domain:Generalization-based memory." Machine learning:An artificial intelligence approach **2**.

Lebowitz, M. (1987). "Experiments with Incremental Concept Formation: UNIMEM." Machine Learning **2**(2): 103-138.

Li, C., Q. Song and C. Zhang (2004). MA-IDS Architecture for Distributed Intrusion Detection using Mobile Agents. 2nd International Conference on Information Technology for Application, Harbin, China.

Li, M., G. Holmes and B. Pfahringer (2004). Clustering Large Datasets Using Cobweb and K-Means in Tandem 17th Australian joint conference on artificial intelligence (AI-04). T. D. Gedeon and L. C. C. Fung. Cairns, Australia, Springer-Verlag. **1:** 368-379.

Liao, T. W. (2005). "Clustering of time series data--a survey." **38**(11): 1857-1874.

Lin, J., E. Keogh and W. Truppel (2003). Clustering of streaming time series is meaningless. Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. San Diego, California, ACM.

Lin, W.-H. and A. Hauptmann (2004). Informedia at PDMC. Twenty-First International Conference on Machine Learning. Banff, Alberta, Canada.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations Proceedings of the 5th Berkeley Symposium Mathematical Statistical Probability. **1:** 281-297.

McKusick, K. B. and P. K. Langley (1991). Constraints on tree structure in concept formation. Proceedings of the Twelfth International Joint Conference on Artificial Intelligence. Sydney, Australia, Morgan Kaufmann**:** 810-816.

Medin, D. L., W. D. Wattenmaker and S. E. Hampson (1987). "Family resemblance, conceptual cohesiveness, and category construction." Cognitive Psychology **19**(2): 242-279.

Mervis, C. and E. Rosch (1981). "Categorization of natural objects." Annual Review of Psychology **32**.

Michalski, R. S. (1974). Variable-valued logic: System VL1. Proceedings of the 1974 International Symposium on Multiple-Valued Logic. Morgantown, West Virginia. **:** 323-346.

Michalski, R. S. (1980). "Knowledge acquisition through conceptual clustering: A theoretical framework and an algorithm for partitioning data into conjunctive concepts." International Journal of Policy Analysis and Information Systems **4**(3): 219-244.

Michalski, R. S. (1980). "Learning by being told and learning from examples: an experimental comparison of the two methodes of knowledge acquisition in the context of developing an expert system for soybean desease diagnoiss." International Journal of Policy Analysis and Information Systems **4**(2): 25-161.

Michalski, R. S. and R. E. Stepp (1981). An application of AI techniques to structuring objects into an optimal conceptual hierarchy. Vancouver, BC, Canada, Morgan Kaufmann Publishers Inc.

Michalski, R. S. and R. E. Stepp (1983). "Learning from observation: Conceptual clustering." Machine Learning: an artificial intelligence approach **1**: 331–363.

Michalski, R. S., R. E. Stepp and E. Diday (1981). "A recent advance in data analysis: Clustering objects into classes characterized by conjunctive concepts." Invited chapter in the book Progress in Pattern Recognition, L. Kanal and A. Rosenfeld (Eds.) **1**: 33-55.

Milligan, G. W. (1981). "A Monte Carlo study of thirty internal criterion measures for cluster analysis." Psychometrika **46**(2): 187-199.

Nevins, A. J. (1995). "A branch and bound incremental conceptual clusterer." Machine Learning **18**: 5-22.

Paxson, V. (1999). "Bro: a system for detecting network intruders in real-time." Computer Networks **31**(23--24): 2435--2463.

Pham, D. T., S. S. Dimov and C. D. Nguyen (2004). "An Incremental K-means algorithm." Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science **218**(C7): 783-795.

Quinlan, J. R. (1986). "Induction of Decision Trees." Machine Learning **1**(1): 81-106.

Quinlan, J. R. (1993). C4.5: programs for machine learning, Morgan Kaufmann Publishers Inc.

Quionero-Candela, J., M. Sugiyama, A. Schwaighofer and N. D. Lawrence (2009). Dataset Shift in Machine Learning, The MIT Press.

Reich, Y. and S. Fenves (1991). The formation and use of abstract concepts in design. In Fisher, D., Pazzani, M., & Langley, P., Concept Formation: Knowledge and Experience in Unsupervised Learning. San Mateo, CA, Morgan Kaufmann.

Rescorla, R. A. (1968). "Probability of shock in the presence and absence of CS in fear conditioning." Journal of Comparative and Physiological Psychology **66**: 1-5.

Sahoo, N., J. Callan, R. Krishnan, G. Duncan and R. Padman (2006). Incremental hierarchical clustering of text documents. Proceedings of the 15th ACM international conference on Information and knowledge management. Arlington, Virginia, USA, ACM.

Salvador, S. and P. Chan (2004). Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms, IEEE Computer Society.

Scanlan, J., S. Lorimer, J. Hartnett and K. Manderson (2004). Intrusion Detection by Intelligent analysis of data across multiple gateways in real-time. Australian Telecommunication Networks and Applications Conference (ATNAC), Bondi Beach.

Scanlan, J., Hartnett, J., Williams, R., (2008). DynamicWEB: Adapting to concept drift and object drift in COBWEB. Proceedings 21st Australasian Joint Conference on Artificial Intelligence , 1-5 December 2008, Auckland, New Zealand, pp. 454-460.

Schapire, R. E. (1990). "The strenght of weak learnablity." Machine Learning **5**(2): 197-227.

Schapire, R. E. (2002). The boosting approach to machine learning: An overview
Workshop on Nonlinear Estimation and Classification, MSRI.

Schlimmer, J. C. and R. H. Granger (1986). "Incremental Learning from Noisy Data." Mach. Learning **1**(3): 317-354.

SourceFire (2004). Snort 2.0: Detection Revistited. Columbia, SourceFire Inc.**:** 7.

Staniford, S., J. A. Hoagland and J. M. McAlerney (2002). "Practical automated detection of stealthy portscans " J. Comput. Secur. **10** (1-2): 105-136

Stanley, K. (2003). Learning Concept Drift with a Committee of Decision Trees. Austin, TX, Dept. Computer Sciences, University of Texas**:** 14.

Stanley, K. O. (2003). Learning Concept Drift with a Committee of Decision Trees. Austin, TX, USA, Department of Computer Sciences, University of Texas at Austin.

Tibshirani, R., G. Walther and T. Hastie (2000). Estimating the number of clusters in a dataset via the Gap statistic. Technical Report, Stanford University, Department of Biostatistics.

Tsymbal, A. (2004). The problem of concept drift: Definitions and related work. Dublin, Trinity College

Valiant, L. G. (1984). "A theory of the learnable." Communications of the ACM **27**(11): 1134-1142.

Wald, A. (1947). Sequential Analysis. New York, John Wiley and Sons.

Wang, Q., Y. Guan and X. Wang (2006). SVM-Based Spam Filter with Active and Online Learning. Proceedings of the Fifteenth Text Retrieval Conference (TREC 2006), Gaithersburg.

Wang, X., K. Smith and R. Hyndman (2006). "Characteristic-Based Clustering for Time Series Data." Data Min. Knowl. Discov. **13**(3): 335-364.

Wasserman, E. A., D. L. Chatlosh and D. J. Neunaber (1983). "Perception of causal relations in humans: Factors affecting judgments of response-outcome contingencies under free-operant procedures." Learning and Motivation **14**: 406-432.

Widmer, G. and M. Kubat (1996). "Learning in the Presence of Concept Drift and Hidden Contexts." Machine Learning **23**(1): 69-101.

Witten, I. and E. Frank (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java implementations`. San Francisco, Morgan Kaufmann Publishers.

Wittgenstein, L. (1953). Philosophical investigations. Oxford, Basil Blackwell.

Yoo, J. P., C. C. Pettey and S. Yoo (1996). A hybrid conceptual clustering system. Proceedings of the 1996 ACM 24th annual conference on Computer science. Philadelphia, Pennsylvania, United States, ACM.

Zalewski, M. (2005). Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks, No Starch Press.

11

Appendix A – Glossary of Terms

APPENDIX A – GLOSSARY OF TERMS

The following glossary of terms defines several of the key words which are used throughout this thesis. Most of these words are known in the field of data mining, while a few of them are either new, or perhaps taken on added meaning in the problem space being investigated in this thesis.

Concept

A concept is basically the resultant class trying to be predicted by a machine learning method in a given dataset. However, as described in Chapter 2 (Section 4), it is comprised of a description based on the shared values that are present in the instances that are in that concept. It was first described by Michalski (1980) and his work has been built on by many authors whose work has been described in Chapters 2, 3 and 4.

Instance (or item)

An instance is a collection of data about a given object (a description of the state of that object); i.e a group of attribute-value pairs. Each of the methods described in the early chapters of this thesis examine datasets that are comprised of instances that are all independent of each other. Each instance stands alone as the only description of the target object in question. As DynamicWEB is introduced, the concept of having multiple instances (multiple recordings of attribute value pairs) of a single target within a dataset is also introduced. This is fundamentally different from most other methods described in the thesis. These multiple instances, or observations of a target object, are combined into a profile. This profile is actually then treated as a single instance when it is added to the DynamicWEB concept hierarchy.

Target Object

The problem that motivated the work in this thesis was in the area of network security, and involved developing behaviour profiles based on recorded audit log data of unknown IP addresses. As such the target objects are malicious users on a computer network. DynamicWEB was produced responding to this problem, and is a method that allows the tracking of target objects in a given

dataset across multiple observations. In this thesis the objects that are spoken of are the entities that are described within the multiple datasets examined. In each of these datasets there are multiple recorded behaviours for each object.

Concept Drift

As described within the thesis several authors (Schlimmer and Granger, 1986; Widmer and Kubat, 1996) have completed research in the area of concept drift and have defined the concept quite well. It is fundamentally the idea that the basic values defining a given resultant class in a dataset change over time. The concept drifts because the definition of what defines that concept changes over time within the period covered by a dataset.

Object Drift

As defined on the previous page, an object is an entity, which is being represented within a dataset by instances that describe its state. In this thesis several datasets are examined where there are multiple instances describing the state of an object over a period of time. The change of that state is what we refer to as object drift. Over multiple observations of that target object it may be that it changes its resultant class. In the example of network security it may be the case that a given IP goes from being considered benign to being considered malicious over multiple observations describing its behaviour. Object drift is a new concept that was first described in this way in this research and its associated publications (Scanlan, 2008).

12

Appendix B – Dynamic Weather Dataset

APPENDIX B – DYNAMIC WEATHER DATASET

The following is a dataset that was created as part of this research to demonstrate object drift occurring across multiple observations. It is discussed in Chapter 6.

```
% Top 4 in Scotland, Wales, Ireland
% Stay Y: 4
% Stay N: 3
% N -> Y: 2
% Y -> N: 2
% N -> Y -> N: 1
% temperature <= 50: no (4.0)
% temperature > 50
% |   outlook = sunny
% |   |   humidity <= 75: yes (2.0)
% |   |   humidity > 75: no (3.0)
% |   outlook = overcast: yes (4.0)
% |   outlook = rainy
% |   |   windy = TRUE: no (2.0)
% |   |   windy = FALSE: yes (3.0)
@relation weather
@attribute location {standrews, gleneagles, carnoustie,
dornoch, porthrawl, stdavids, nefyn, pennard, countydown,
portrush, ballybunion, lahinch}
@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}
@data
standrews,overcast,45,81,TRUE,no
gleneagles,overcast,56,80,FALSE,yes
carnoustie,sunny,85,50,FALSE,yes
dornoch,overcast,65,60,TRUE,yes
porthrawl,rainy,45,80,TRUE,no
stdavids,rainy,50,75,TRUE,no
nefyn,sunny,72,62,FALSE,yes
pennard,sunny,45,75,FALSE,no
countydown,sunny,55,80,FALSE,yes
portrush,sunny,90,90,FALSE,no
ballybunion,overcast,65,62,TRUE,yes
lahinch,overcast,49,78,FALSE,no
standrews,overcast,55,78,TRUE,yes
gleneagles,rainy,58,79,FALSE,yes
carnoustie,sunny,90,45,FALSE,yes
dornoch,sunny,70,58,FALSE,yes
porthrawl,rainy,49,80,FALSE,no
stdavids,rainy,55,75,TRUE,no
nefyn,sunny,81,60,TRUE,yes
pennard,rainy,49,81,TRUE,no
```

```
countydown,overcast,50,80,TRUE,no
portrush,sunny,100,90,TRUE,no
ballybunion,sunny,72,61,FALSE,yes
lahinch,overcast,52,75,FALSE,yes
standrews,sunny,60,74,TRUE,yes
gleneagles,rainy,54,81,TRUE,no
carnoustie,sunny,87,45,FALSE,yes
dornoch,sunny,72,55,FALSE,yes
porthrawl,rainy,50,80,TRUE,no
stdavids,overcast,58,70,FALSE,yes
nefyn,rainy,76,64,FALSE,yes
pennard,rainy,46,85,TRUE,no
countydown,rainy,48,85,TRUE,no
portrush,sunny,98,90,TRUE,no
ballybunion,sunny,75,60,FALSE,yes
lahinch,overcast,50,75,TRUE,no
```

13

Appendix C – STAGGER Concepts Dataset Listing

APPENDIX C – STAGGER CONCEPTS DATASET LISTING

The following is the listing of the data that was used in Chapter 6 where a direct comparison was made between DynamicWEB and COBBIT. The impact of the order upon the two different methods of adjusting to concept drift was the primary reason for that examination and so the order is shown here.

```
@relation Stagger
@attribute id real
@attribute size {small, medium, large}
@attribute colour {red, blue, green}
@attribute shape {square, circular, triangular}
@attribute result {true, false}
@data
1,large,blue,circular,false
2,medium,green,trangular,false
3,small,green,square,false
4,large,red,square,false
5,small,red,circular,true
6,large,blue,square,false
7,small,green,square,false
8,medium,red,circular,false
9,small,green,circular,false
10,small,green,trangular,false
11,medium,green,trangular,false
12,small,red,circular,true
13,medium,red,square,false
14,small,red,trangular,true
15,large,green,trangular,false
16,small,red,trangular,true
17,medium,blue,circular,false
18,small,blue,circular,false
19,small,red,trangular,true
20,medium,green,trangular,false
21,large,red,circular,false
22,small,blue,trangular,false
23,small,blue,square,false
24,medium,green,circular,false
25,small,green,circular,false
26,small,red,square,true
27,medium,red,trangular,false
28,small,red,circular,true
29,large,blue,circular,false
30,medium,green,square,false
31,large,blue,square,false
32,large,blue,circular,false
```

```
33,large,blue,circular,false
34,small,green,circular,false
35,small,red,square,true
36,large,red,trangular,false
37,large,blue,square,false
38,small,blue,trangular,false
39,large,green,square,false
40,small,green,trangular,true
1,medium,green,trangular,true
11,small,blue,square,false
39,medium,blue,trangular,false
32,small,green,square,true
28,small,red,trangular,false
20,large,red,trangular,false
37,small,red,circular,true
21,large,blue,circular,true
33,small,red,square,false
24,small,red,circular,true
26,small,blue,square,false
10,medium,red,circular,true
9,small,green,trangular,true
38,small,red,circular,true
27,medium,red,circular,true
15,large,red,square,false
34,medium,green,square,true
40,large,green,square,true
17,small,green,square,true
22,medium,red,trangular,false
6,small,blue,trangular,false
18,medium,blue,trangular,false
13,medium,red,trangular,false
16,medium,blue,square,false
7,small,green,square,true
5,large,green,circular,true
29,large,green,square,true
4,large,red,circular,true
23,small,red,square,false
8,large,green,trangular,true
3,large,blue,circular,true
35,large,red,square,false
30,medium,blue,square,false
12,small,blue,circular,true
25,small,green,circular,true
36,small,blue,circular,true
31,small,red,square,false
14,medium,green,circular,true
2,medium,green,trangular,true
19,large,green,square,true
12,large,blue,trangular,true
32,large,blue,square,true
38,large,green,trangular,true
```

```
18,small,green,trangular,false
24,small,blue,trangular,false
29,medium,green,square,true
8,medium,red,trangular,true
36,large,green,trangular,true
13,medium,red,circular,true
28,small,red,circular,false
23,large,red,trangular,true
17,small,green,trangular,false
1,large,green,square,true
34,large,green,trangular,true
2,small,blue,circular,false
9,large,green,trangular,true
14,large,green,trangular,true
20,medium,blue,square,true
6,small,red,circular,false
21,small,green,square,false
15,medium,red,square,true
31,large,red,circular,true
10,large,blue,circular,true
26,large,green,square,true
19,large,blue,trangular,true
40,large,green,circular,true
3,medium,green,square,true
39,medium,red,trangular,true
4,large,red,circular,true
27,small,green,square,false
5,medium,red,trangular,true
11,medium,red,trangular,true
16,large,green,square,true
35,small,red,circular,false
33,small,blue,square,false
37,large,green,square,true
22,small,blue,circular,false
30,small,green,trangular,false
7,medium,green,trangular,true
25,small,red,circular,false
```

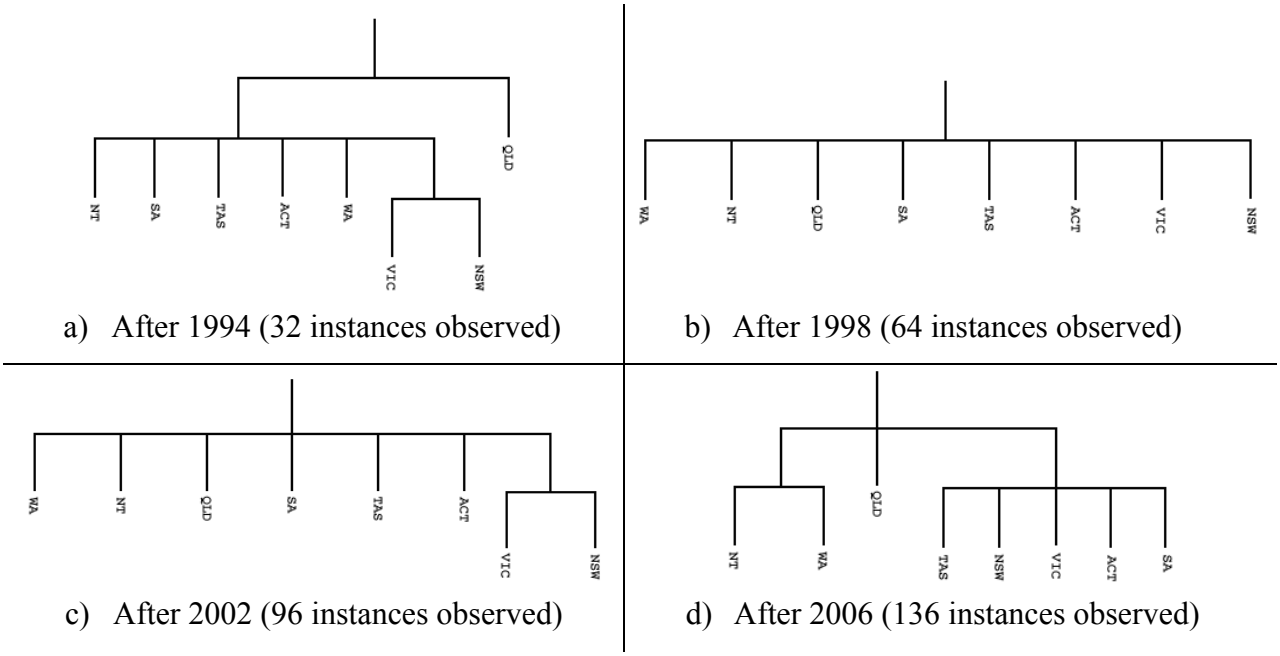
14

Appendix D – Additional Results for National Accounts Dataset

APPENDIX D – NATIONAL ACCOUNTS

The following is a group of trials conducted on the National Accounts dataset. The results of most interest are included within the body of the text in Chapter 7.

The simplest way to demonstrate changing structure within a dataset over time is to have DynamicWEB simply store the most recently observed value for each attribute. In Figure 59 four trees are shown which illustrate the structure produced by DynamicWEB at 4, 8, 12 and 17 years. The structure produced by only storing the most recent value was not stable, and frequently took the form shown for years 8 and 12, with some leaves merging to form a combined node of disparate leaves occasionally (as in year 12). These trees indicate that there is insufficient regularity or structure within the data to enables objects to be grouped together and for relationships to be displayed. These trees provide a useful comparison with the structures that are formed when some of the contextual data for previous observations relating to each target object are retained.



a) After 1994 (32 instances observed)

b) After 1998 (64 instances observed)

c) After 2002 (96 instances observed)

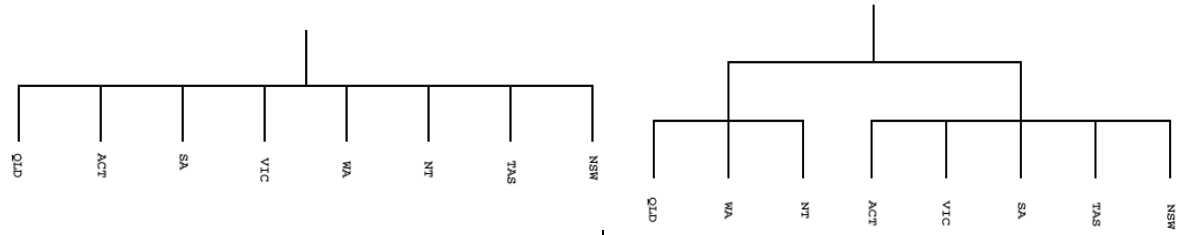
d) After 2006 (136 instances observed)

Figure 59. These four structures are from four different points in time during DynamicWEB's analysis of the National Accounts dataset.

The simplest tree configuration that retains any context from the previous activity, across the 17 measurements, is obtained by simply taking the mean of each attribute, as a derived attribute, along with the most recently observed value. The tree produced in this way is shown in Figure 60a. This illustrates how using a simple

profile such as this may teach us very little about the structure of the dataset we are examining, and may even be less helpful than not using contextual information at all. The tree represents each target object as its own leaf, linked directly to the root node; showing that DynamicWEB was unable to discover any separable clusters, containing more than a single profile, within the dataset.

The derived attribute was calculated using the data from each observed measurement of each target object, resulting in 14 derived values each built upon 17 measurements in this dataset. This derived attribute was the *mean*, and in this case it did not have any generalising effect. This resulted in a structure where none of the profiles were considered similar to each other. However, as the data covers a 17-year time window, the derived attribute was tested under the assumption that this would retain only a portion of the past data to see if this would aid in establishing relationships. However, a time window of 5 years, or one third of the dataset, was tested and found to have no impact on the resulting structure produced.



a) The structure found when using the mean as a derived attribute.

b) The final structure from the third tree in the second trial.

Figure 60. The structures that were found in the first two trials to retain context within DynamicWEB.

Within the description of the National Accounts dataset in Table 28 (Chapter 7) there are obvious groups of attributes that relate to the same measure. One of the strengths of DynamicWEB is its ability to split a problem into several portions and then examine each portion in separate trees, in parallel with each other. Within classification problems, the classification results can then be used in conjunction with one another, and although in this circumstance there are no resultant classes to predict, the resulting data structures can be viewed and compared with one another. The most obvious groupings of attributes to trial first are those formed by grouping the three main measures within the dataset: Gross state product- Chain volume measures, Real gross state income and Gross state product- Current prices. Using

DynamicWEB, the dataset was analysed with the attributes being split between the trees so that the four first attributes (#3-6) were in one tree, the next four were in the second tree (#7-10) and the remaining six (#11-16) within the third tree. In addition to these, two derived attributes were added per attribute to each profile, lifting the total number to 12 in the first two trees and 18 in the third. The two derived attributes used for each of the original observed values were the *standard deviation* and the *trend*. The *trend* is incremented with each observation of a numerical field that is greater than the previously observed value, or decremented if the next observed value is smaller. Over time, consistently increasing values will result in a high number, fluctuating results will remain near zero, and consistently shrinking values will be negative.

The trees produced by DynamicWEB in this trial exhibited more structure than those produced in the first trial. However, the structure was only apparent for one or two years of the observations (once the derived attributes started to form), and then the 8-separate-leaf node structure re-appeared for several years. These results were found when the full dataset was used for the derived attributes, or if a 5-year time window was used for the derived attributes. While some structures (such as those shown within Figure 60b) were discovered within both the 5-year window and the full dataset, they only split the dataset into two groups of leaves and didn't draw any stronger relationships then that. However, the tree illustrated in Figure 60 used data that was weighted slightly more to metrics that were per-capita measures. As the task being examined involves profiling the eight state and territories in relation to each other, it is logical that attributes that largely reduce the effect of the population size of the states compared to each other would allow for a more meaningful comparison. Comparisons of this nature are examined within Chapter 7.

15

Appendix E – Additional Results for the Labour Force Dataset

APPENDIX E – LABOUR FORCE

The following is an examination of the Labour Force dataset in its original configuration, with the states being the objects of interest.

Figure 61 shows two sets of structures formed by DynamicWEB when examining the Labour Force dataset. The structures on the left represent the knowledge hierarchy that was formed by examining all of the full time employees by industry per state, while those on the right represent those that are part time. The profiles used in these hierarchies are based upon the most recently observed value, the *standard deviation* of that value, and the *trend* of the value. The structures shown in Figure 61 are much more stable in nature than those shown within the

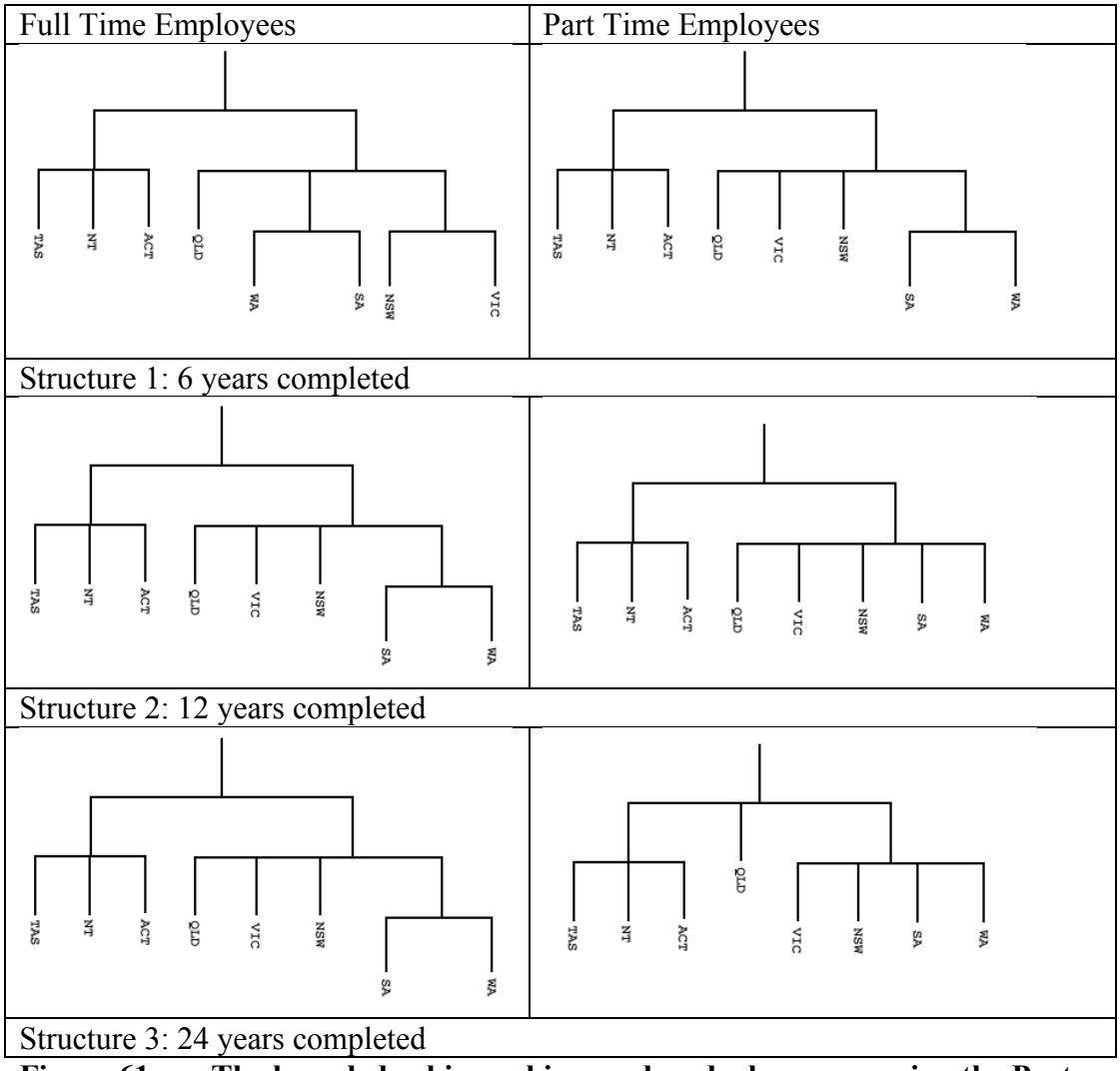


Figure 61. The knowledge hierarchies produced when comparing the Part Time and Full Time.

National Accounts dataset, which also tracked these target objects (as discussed in the Chapter 7). The reason for this could be that, even with industry shifts within this time period, when comparing all of the industries within a state to other states and territories, these changes do not cause sufficient difference to noticeably modify the hierarchy produced. However, it is not completely stable because Queensland moves from the grouping of five states and territories to which it initially belonged, to becoming a leaf off the root node in the part time structure. Due to the similarity between this dataset and the National Accounts dataset, further examination of this dataset in this from was not undertaken. Instead the dataset was transformed to examine the industries as the objects of interest and the results of this are discussed in Chapter 7.

16

Appendix F – DynamicWEB Complexity

APPENDIX F – DYNAMICWEB COMPLEXITY

This thesis outlines the learning method, DynamicWEB, which modifies the existing method COBWEB. These key modifications to the learner allow for a concept hierarchy constructed using the COBWEB algorithm to be changed by updating an instance in the hierarchy, or removing an instance. Both of these functions are still heavily based on the key calculation (which has not been changed in this work) used by COBWEB: the category utility. This has not been changed in this work. The new algorithm DynamicWEB, contains three main operations using the category utility measure: *insert* (unchanged from COBWEB), *remove* and *update*. Below we will examine each of these operations in terms of their complexity.

The insert operation which is responsible for building the hierarchy is unchanged from COBWEB. It searches the existing hierarchy for the ideal place with insert the most recently presented instance. Each node in the tree is compared with the instance using the category utility, to measure how well suited it is to be stored in that branch of the hierarchy. This process is repeated until it is stored at a new leaf, or a node that it is well suited it is to be discovered. It is this search process, with many category utility calculations taking place, that is the core of the computational complexity of both COBWEB and DynamicWEB. The complexity of the *insert* operation can be expressed as follows (Fisher, 1987):

$$O(B^2 \log_B n \, AV)$$

where B is the average branching factor of the hierarchy, n is the number of instances already stored, and A and V are the number of attributes and the average number of values per attribute respectively. Further explained, for each child sibling at a node compared to a presented instance, there is a complexity of O(BAV), and therefore $O(B^2AV)$ for the set. The height of the hierarchy can be approximated by $\log_B n$ which combines for the above total expression of the cost to incorporate a single instance into a COBWEB hierarchy. A brief discussion of the branching factor can be found in the earliest COBWEB paper (Fisher, 1987).

The two main operations that were added to *insert* in order to create DynamicWEB are *remove* and *update*. The *update* operation is fundamentally a *remove* followed by an *insert* (with the re-added instance being modified with new information). The

remove operation can also be used to remove knowledge from the tree once it has expired should a time window be being applied to the learner. The first step of the remove operation is to locate the instance within the knowledge hierarchy to be removed this could result in a worst case of searching the entire tree (i.e a O(n)). This is not an efficient search due to the hierarchy being sorted based on the similarity of the stored instances and not on the identifier of the instances. Instead, however, DynamicWEB uses an AVL (Adelson-Velskii and Landis, 1962) tree as an index of all the current locations within the tree with a search complexity of log n. Now incorporating this search to aid in locating instances in the hierarchy thus adds a log n cost to the insert operation in order to maintain the AVL tree as opposed to O(n).

Once the instance of interest has been located, its removal process involves the inverse operation of the insert to remove the knowledge from the tree. Each parent node, from the direct parent, all the way to the root of the tree, has the knowledge subtracted from it, resulting in a cost of

$$\mathbf{O(\log_B n\, AV)}$$

where $\log_B n$ is the approximate height of the hierarchy, and A is the number of attributes that influence the probabilistic concept descriptions within the nodes; and V is the number of possible values, for each of these attributes, that are searched in order to locate the one which the instance being removed exhibits.

The *update* operation, from a complexity point of view, is a combination of the *remove* and *insert* operations. The knowledge relating to the instance is removed from the hierarchy, before a newly updated instance with freshly incorporated observed data is inserted. The AVL tree is searched once to locate the instance in the hierarchy, and this is then updated, to store the new location for the instance, without searching for it again.

DynamicWEB, in comparison to COBWEB, does derive a benefit from the ability to update which has an effect on complexity. COBWEB, as discussed in Chapter 4 Section 5, can suffer from order dependence; the result being that a less than ideal hierarchy can be produced. The work carried out by McKusick and Langley (1991) in ARACHNE aim to counter this. The update mechanism in DynamicWEB would have similar effects to the work on ARACHNE, although these are not empirically

examined in this thesis. The further work mentioned in Chapter 9 highlights that investigating this and possibly adding the added ARCHNE operators to DynamicWEB could produce some interesting results.