DynamicWEB: Adapting to Concept Drift in COBWEB

Joel Scanlan, Jacky Hartnett and Raymond Williams

School of Computing and Information Systems, University of Tasmania,
Tasmania, Australia
{joel.scanlan, j.hartnett, r.williams}@utas.edu.au

Abstract. Examining concepts that change over time has been an active area of research within data mining. This paper presents a new method that functions in contexts where concept drift is present, while also allowing for modification of the instances themselves as they change over time. This method is well suited to domains where subjects of interest are sampled multiple times, and where they may migrate from one resultant concept to another. The method presented here is an extensive modification to the conceptual clustering algorithm COBWEB, and is titled DynamicWEB.

Keywords: Data Mining, Contextual Clustering, Concept Drift

1 Introduction

Everything changes with time. Ideas change, humans change, concepts change. This is evident in many areas and data mining is no different. The field of concept drift aims to notice changes within a given dataset, and then adapt to these changes. Learning algorithms that allow concept drift are able to be more robust over time. Within online learning applications being able to adjust a class definition as a result of changes within a domain allows a learning method to produce a model which is accurate up to the current moment.

Concept drift, as examined within the context of data mining, has been studied since the 1980s. Various techniques for detecting and reacting to change within a dataset have been developed, some of which will be discussed within this paper. However, this paper discusses a new method that is closely related to concept drift, but is focused upon the changes of individual objects which are examined multiple times over a given time period, where they might drift from one resultant class into that of another. As these objects change with time, drift within the concepts which they form can also take place. The resulting two different forms of drift being accounted for within the one algorithm described in this paper.

This paper will first examine previous work within the area of concept drift and conceptual clustering algorithms before presenting the algorithm completed by the authors known as DynamicWEB. Some preliminary results will then be presented and discussed.

2 Concept Drift

A large portion of the data mining field focuses on datasets which operate independently of time and ordering within the data. It is extremely common for methods to be run many times upon the same datasets with the ordering being randomised between executions. This strategy removes any bias in the ordering of the data, and achieves a true measure of the technique's quality by determining the average of these executions [1]. However, some datasets and domains are present within the context of time, and, therefore, the ordering of the data is a vital part of the dataset, inherently providing more detail than the instances themselves. Randomisation of data within these domains results in knowledge loss. Often these methods operate in an ongoing fashion, incorporating new data as it is produced or recorded by a system. These kinds of techniques are often called online-learning methods.

Within some domains that operate in a time context, the algorithms used need to incorporate new knowledge with previously observed items. The data might be entering the system in real time as output from a device or sensor. This situation cannot be handled by a large number of data mining techniques which are batch-based, requiring all of the dataset's instances to be present at the outset. Online learning methods function incrementally, increasing the knowledge within the system over time as more data is observed, but without requiring all of the data to be present at the outset. If the whole dataset is present at the beginning then an online learner can still utilise the ordered context of the dataset, and the entire dataset is not actually required. This allows continuous datasets to be examined.

The value in being able to examine a continuous dataset is being able to detect events, or changes, as they occur. For example, if the dataset is about the stock prices within a given market over the course of a week, the changes that occur in the prices are very tightly linked to the ordering of the data. Their examination needs to occur in their existing order, allowing the overarching trends of the week to be extracted. If there is a dramatic change within the dataset, again, having them sequenced is important in order to grasp the surrounding events leading to this change. This simple example quickly illustrates the usefulness of mining data within a time series. Concept drift occurs when the resultant class definitions from data mining a time series change during the dataset's examination. This can be due to a factor or cause that is recorded within the dataset itself, or it could be a symptom of some external variable not recorded within the dataset.

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 trends can be extracted that respond to time-related external variables such as 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 and adjust to such events or processes. Traditional methods, through randomisation could miss this knowledge or treat it as noise. Online learning methods aim not only to discover concept drift, but also to adapt with it to ensure that the current model is optimal. Concept drift occurs in two main forms: sudden and gradual [2]. Within the

customer shopping example, a natural disaster would cause a sudden drift, while inflation over the course of a year could cause gradual drift.

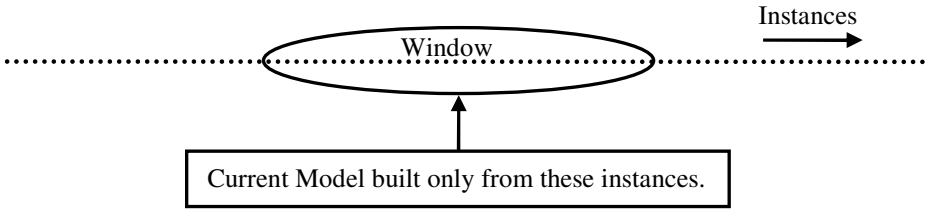
Concept Drift Methods

Two methods which are well regarded within the field, and which were somewhat foundational, are STAGGER [3] and FLORA [4]. They will be described here. The two methods, at their core, are each classic examples of two modes of adapting over time; knowledge utility (STAGGER), and time windowing (FLORA). These techniques aim to differentiate noise from meaningful data and then incorporate concept drift into the model. Both methods are supervised learners which does set them apart from our own which is an unsupervised approach.

STAGGER is a probabilistic data mining approach that uses a weighted node clustering graph. STAGGER, being a supervised method, requires each instance read into the system to have a class identifier attached to it. This knowledge of the class is used to adjust the weights of Boolean concepts as the algorithm functions. The creation, searching and adding of concepts within the graph are founded within Bayesian statistics. If the resultant classification that occurs from an instance was being placed in the graph, then the graph would be corrected with node creation or removal occurring where required. A full description of the STAGGER approach can be found [3].

FLORA was developed several years after STAGGER. Unlike STAGGER, where the worth of knowledge is judged based upon its utilisation, within FLORA, knowledge is retained for a given time period. After this it is removed from the system, regardless of its utility. Figure 1 illustrates the way in which FLORA's window progresses along a dataset of many instances. The instances in front of the window are the unobserved events which will used to update the model in the future, while those behind the window have already been observed and have now been discarded. The size of the window is obviously of great importance to the effectiveness of such a system. Within the different variations of FLORA presented by Widmer and Kubat [4] are included several that utilised a dynamically sized window quite effectively. The use of a window in this manner is an original concept, and, indeed a system which we will compare our own with later in the paper predates FLORA in using a window; however, the FLORA paper, along with STAGGER, are keystone papers within the concept drift area. Both explain simple concepts clearly, and, in FLORA's example, with multiple variations of the method.

Figure 1. FLORA: Visual representation of the window moving through a dataset.



3 Conceptual Clustering

Conceptual Clustering was first outlined by Michalski [5] and was further expanded with Stepp [6] introducing their PAF method. 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 classification, are not as simple to interpret or fully understand from the human perspective. 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, these classes fit descriptions which illustrate a true "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 concepts that are broad are located towards the root, with more specific concepts nested within those higher parent concepts as children.

This paper presents a method, entitled DynamicWEB, which at its core is a substantial modification to the COBWEB unsupervised conceptual clustering algorithm. This is not the first time that the COBWEB algorithm has been modified by researchers, and indeed other work has even been completed to allow it to adapt to concept drift. This other work is titled COBBIT and will also be briefly explained.

COBWEB

The COBWEB algorithm was published by Fisher [7] and builds upon the work completed by Michalski in PAF [5], and the UNIMEM [8] and CYRUS (1983) systems by other authors. While COBWEB draws from these methods, the most significant related work which is also incorporated into COBWEB is that of the Category Utility by Gluck and Corter [9, 10]. The COBWEB algorithm utilises a hierarchical tree to group the observed instances into concepts where traits are shared across the resident instances. The measure COBWEB uses to group the instances together is the category utility.

Gluck and Corter were able to show, using the category utility, similar basic level categorisation to that found within human psychological testing. Basic level categorisation, as used by Gluck and Corter, was described by Mervis and Rosch [11]. A basic level category is defined as one which is preferred to a more generalised or specific category. Fisher showed that by using the category utility in the data mining context a probabilistic conceptual clustering algorithm could be produced, that is highly effective.

The category utility calculation takes account of each attribute in an instance, comparing it to the attributes of the instances within a given cluster, returning the utility as a measure of how much information they have in common. This attribute-value pair comparison used within COBWEB is sufficiently resilient to produce a useful measure of likeness, and is also able to adapt to missing attributes within instances. Our research does not modify this calculation; for an in-depth explanation of its operation and derivation refer to Gluck and Corter [9, 10] or Fisher [7].

COBWEB is an incremental conceptual clustering algorithm and as such grows the knowledge stored within its structure one instance at a time. When a new instance is added to COBWEB, its resulting location is found by searching through the existing tree and trialling a range of options at each probable location (see Table 2). Each alternative is trialled, and a category utility is calculated for each result. The option with the best resulting category utility, above a cut-off threshold, is then identified as the best choice. If the cut-off threshold is reached then the current node is the best location for the new instance.

The first of these possible solutions is to check to see if it is necessary to create a new leaf cluster or class. 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 root. The second option trialled is to incorporate 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 class. The third and fourth options are inverse operations of each other: merge and split. These two operations optimise the tree for greater knowledge retention. The merge function considers merging the two children of the current node with the highest category utility as calculated during the incorporation stage. Conversely, when an instance is classified to a class, the split function considers removing the class and elevating the children. This, again, requires calculation of the category utility.

Once each trial is completed the resulting category utilities are compared, and the most favourable option is then adopted. With each instance this process occurs, building the tree by adding each item nearest to other items of similar attributes. Future merges and splits bring these groups into clusters that can then be used in classification.

Since the publication of COBWEB there have been multiple other authors who have modified the method to further the research. COBBIT by Kilander and Jannsson [12] is a variation of COBWEB that allows it to adapt to concept drift through the usage of a time window in a similar fashion to FLORA. COBBIT, as it is based upon COBWEB, is an unsupervised learner. The reasons of being unsupervised and related to COBWEB, and also able to function in the presence of concept drift, is why we use COBBIT to compare with DynamicWEB within this paper.

Table 2. The COBWEB insert mechanism

1.	Search children for the best match to the current instance (start at root for new instance)
2.	Create a new class; calculate category utility
3.	Incorporate into the best matching existing child at this node; calculate category utility
4.	Merge the two best matching children and incorporate forming a new class; calculate category utility.
5.	Split the best matching child; calculate category utility
6.	Select the option from above with the greatest category utility and then continue at 1. If the category utility is below the cut-off then move to 7.
7.	Move on to the next instance at step 1, continuing till the end of the dataset.

4 DynamicWEB

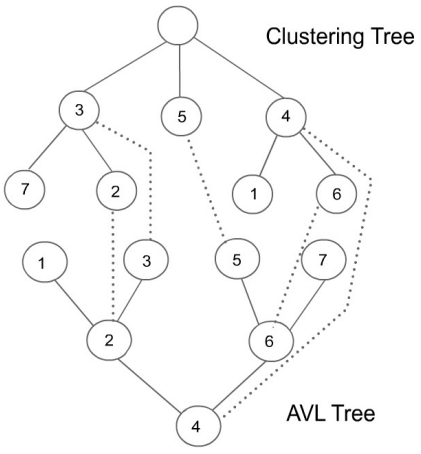
DynamicWEB was created to allow objects to be tracked overtime as they change within their domain of knowledge. Example domains include a person's behaviour being tracked within a security system, the weather at a given location as it changes or a person's health over a time period. It is common within data mining for methods to be able to examine latitudinal datasets, where many objects are sampled once. Patterns are extracted, and knowledge is gained. But there is a lack of methods for longitudinal studies where the same person, or object, is sampled many times requiring models to be updated with the new information. It is this problem space which DynamicWEB is investigating.

Concept drift is a related area to this but instead of just the concept changing with time; DynamicWEB is looking at objects that change with time also. As DynamicWEB aims to follow the individual objects as they change, the overarching concepts are able to drift. As such, DynamicWEB allows for both concept and object drift. As a given object changes with time the instance within the DynamicWEB tree is located, updated and re-clustered with respect to its neighbours reflecting the new information gained from its most recent change. To fully understand how this was undertaken an explanation of the modifications to COBWEB will now be discussed.

The Method

COBWEB in its original form was not intended to have updates occur to the instances that were present within the tree structure. The tree is sorted based on the likeness of the objects resident within in and to locate a given instance each instance would have to be examined in turn, resulting in a O(n) search time. Further, there was no provision for removing, or modifying, an instance within the tree once it was placed there. A modification of an instance within the tree itself would change the category

Figure 2. This represents the way that the AVL tree acts as an index for the clustering tree. The AVL tree is sorted by an identifier, while the clustering tree is sorted by attribute similarity.



utility of the node containing it, and any parent nodes, thus destroying integrity of the tree. Therefore a deletion and modification method was needed as well as a faster way of searching the tree in order to enable these operations to occur in a more efficient manner.

An index for the tree was implemented (Figure 2) using an AVL tree. For each instance that was added to the tree the identifier for the object was also stored within the AVL tree along with a pointer to its location within the clustering tree. The AVL tree could now be searched to locate the instances which have been clustered with a search time of O(*log n*).

When considering the update mechanism (Table 3), and the various scenarios' in which an item might require to be updated it was decided that the most appropriate action would be to remove the item from the tree, adjust the surrounding area, and then re-add the item. While there are scenarios where simply updating the instance in place and then flowing those effects through would be acceptable, and indeed in these cases it would be more efficient, this is not true of all cases. In some instances where attributes are replaced and the variation could cause the instance to move to a new location a significant distance away, the migration process would be cumbersome. To avoid this, the update mechanism removes the instance of interest, updating the tree recursively back to the root, and then re-adds the instance to the tree. The instance may contain attributes derived from multiple different observations of the object, thus creating a profile across the multiple observations. As the instance changes across multiple observations it may migrate from one classification to another, or the instances which are grouped together as a given class may migrate to a different position in the tree.

Table 3. The DynamicWEB update mechanism

1.	Search AVL tree for the instance to be updated, returning the node it is located at.
2.	Remove the instance from the node in the tree
3.	Update the relational statistics at that node
4.	Consider carrying out an operation on that node I. Is the node empty and needs to be removed II. Should the node be split? III. Should the node be merged? For options II and III calculate the category utility, comparing to the current value.
5.	If the update option produced a category utility that is better than the current, and greater than the cut-off, then perform operation. For all instances affected by an operation update their nodes within the AVL tree.
6.	If the current node is the root, or the cut off is greater than the suggested category utilities move onto step 7. Else move to the parent and perform step 2 onwards.
7.	Update the instance with the new information that has arrived. This may include I. Calculating derived attributes II. Replacing values
8.	Insert updated instance back into the clustering tree, updating the AVL tree location.

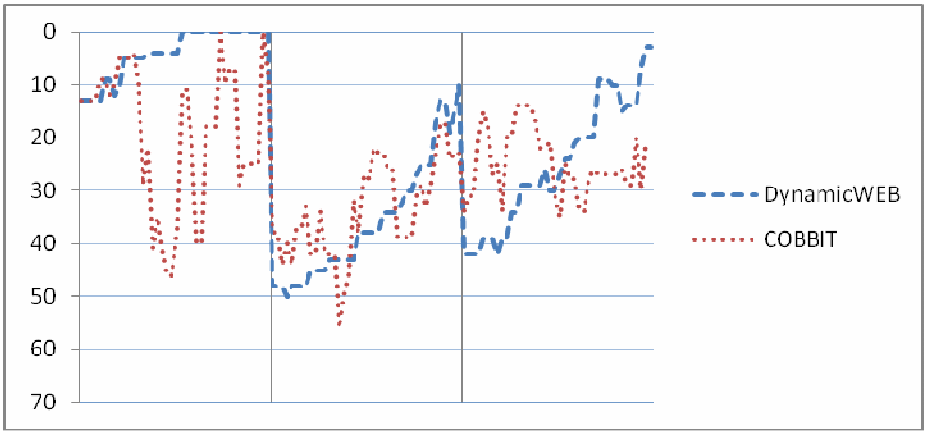
5 Results

DynamicWEB is quite a different method to those discussed previously. Its main strength is being able to update and keep track of individual objects over time as they change. Concept drift algorithms have previously focused on keeping track of a class definition, or measuring the effectiveness of algorithms at recovering a class definition that has been changed drastically. Figure 3 shows a comparison between DynamicWEB and COBBIT [12] using the STAGGER concept dataset [3]. The dataset is a series in which two sudden drifts occur within the resulting classes of the instances. The learning methods are then required to re-learn the resulting class definitions. Both STAGGER and FLORA used this dataset as a method of measuring how quickly a class could be relearned after a sudden drift. COBBIT is using a window size one quarter the size of each concept group (10). COBBIT starts to recover from the sudden drift faster than DynamicWEB, but is overtaken by the midpoint of each concept group.

DynamicWEB can be of use in traditional concept drift problems. However its primary goal is to track when individual objects change over time and drift from one class to another, independent of whether the classes are drifting or not. Figure 4 illustrates a small dataset that the authors created to show the way DynamicWEB functions on larger datasets. The dataset is based upon Quinlan's [13] weather dataset, but is expanded to include 3 measurements of 12 locations. The values of the instances were chosen after examination of the original dataset, and class labels were assigned to the instances using Quinlan's C4.5 [14]. Over these three measurements each location's temperature, humidity, rain and wind profiles are updated. Some locations start off within the positive class, and then migrate to the negative; some remain where they are; others migrate in the opposite direction.

To illustrate that DynamicWEB can scale to a much larger dataset a third test was completed on the Physiological Data Modeling Contest dataset [15]. DynamicWEB performed well monitoring the 35 objects, each of which were updated fifteen thousand times within the dataset.

Figure 3. A comparison of DynamicWEB vs COBBIT with the STAGGER concepts dataset.



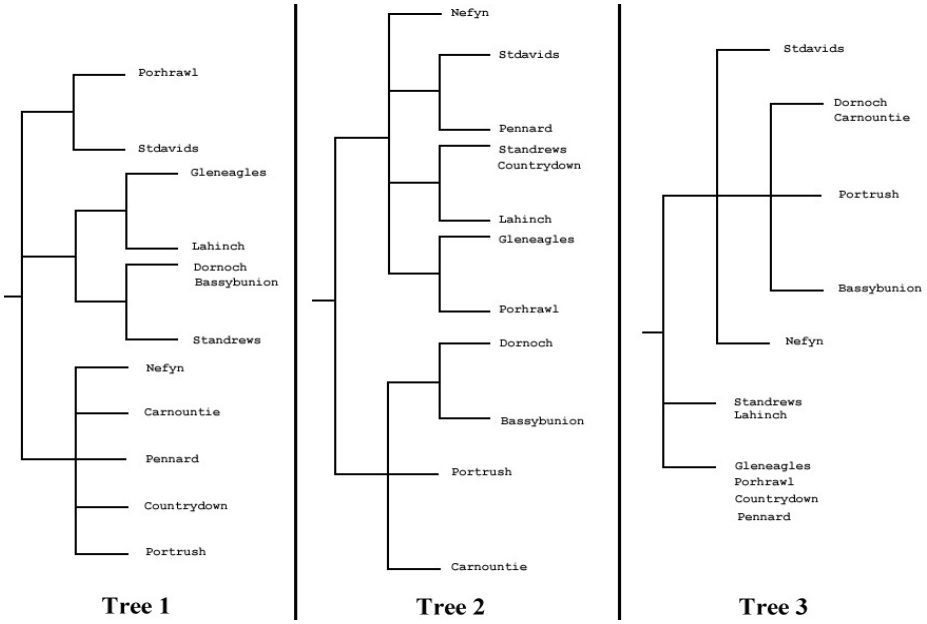


Figure 4. The progression of the DynamicWEB tree across three updates to a modified weather dataset. Each item within the tree is updated twice resulting in the modifed trees presented.

6 Further Work

DynamicWEB is currently being run on more datasets of various sizes, from multiple domains to illustrate its generality and usefulness across multiple knowledge domains.

DynamicWEB was also designed to operate using multiple clustering trees in parallel. The authors of both STAGGER and FLORA discussed the negative impact of noise upon concept drift, and the authors of this paper believe that it may be possible to reduce this impact by parallelising some of the clustering. Within datasets containing many attributes, some of the attributes operate independently of each other, and can act as noise to each other. This effect could be worsened by inherent noise within these attributes. By parallelising the clustering across multiple trees it may be possible to reduce this impact. This feature is currently being tested upon other datasets not shown here, and has previously been discussed elsewhere [16].

7 Conclusion

This paper discussed and presented a new method for not only adapting to concept drift, as well as adapting to change that is occurring within objects which are being observed multiple times.

The examination and monitoring of individual objects across multiple observations is a natural and useful extension to the area of concept drift. DynamicWEB has been presented as a novel approach to this problem space, while still being grounded within proven data mining theory by extending the respected method COBWEB [7]. The method is currently being tested upon different datasets from many domains in an effort to prove both its generality and usefulness in data mining universally.

A comparison between DynamicWEB and COBBIT using the STAGGER concept dataset was presented. Also shown was a diagram illustrating the changes that occurred after each objects' profile within the tree was updated twice.

In further work DynamicWEB enabled to perform conceptual clustering in parallel across multiple linked classification trees. This is being undertaken to reduce the impact of unwanted interactions between independent variables.

References

1. I. Witten and E. Frank, *Data mining: Practical machine learning tools and techniques with java implementations*`, Morgan Kaufmann Publishers, San Francisco, 2000.
2. K. Stanley, "Learning concept drift with a committee of decision trees," Dept. Computer Sciences, University of Texas, Austin, TX, 2003, p. 14.
3. J. C. Schlimmer and R. H. Granger, *Incremental learning from noisy data*, Mach. Learning **1** (1986), no. 3, 317-354.
4. G. Widmer and M. Kubat, *Learning in the presence of concept drift and hidden contexts*, Machine Learning **23** (1996), no. 1, 69-101.
5. R. S. Michalski, *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** (1980), no. 3, 219-244.
6. R. Michalski and R. Stepp, *Learning from observation: Conceptual clustering*, Machine Learning: An Artificial Intelligence Approach (1984), 331-363.
7. D. H. Fisher, *Knowledge acquisition via incremental conceptual clustering* Mach. Learn. **2** (1987), no. 2 139-172
8. M. Lebowitz, *Concept learning in a rich input domain:General-ization-based memory*, Machine learning:An artificial intelligence approach **2** (1986).
9. M. Gluck and J. Corter, "Information,uncertainty,and the utility of categories," *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Lawrence Erlbaum Associates, Irvine,CA, 1985, pp. 283-287.
10. J. Corter and M. Gluck, *Explaining basic categories: Feature predictability and information*, Psychological Bulletin **111** (1992), no. 2, 291-303.
11. C. Mervis and E. Rosch, *Categorization of natural objects*, Annual Review of Psychology **32** (1981).
12. F. Kilander and C. G. Jansson, "Cobbit - a control procdure for cobweb in the presence of concept drift," *Proceedings of the European Conference on Machine Learning*, P. B. Bradzil (Editor), Springer Verlag, 1993.
13. J. R. Quinlan, *Induction of decision trees*, Mach. Learn. **1** (1986), no. 1, 81-106.
14. ---, *C4.5: Programs for machine learning*, Morgan Kaufmann Publishers Inc., 1993.
15. "Physiological data modeling contest," http://www.cs.utexas.edu/users/sherstov/pdmc/, 2004.
16. J. Scanlan, J. Hartnett and R. Williams, "Dynamic web: Profile correlation using cobweb," *In: 19th Australian Joint Conference on Artificial Intelligence*, Springer, Hobart, Australia, 2006.