CONCEPT DRIFT LEARNING AND

ITS APPLICATION TO ADAPTIVE INFORMATION FILTERING

A Dissertation

by

DWI HENDRATMO WIDYANTORO

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2003

Major Subject: Computer Science

CONCEPT DRIFT LEARNING AND

ITS APPLICATION TO ADAPTIVE INFORMATION FILTERING

A Dissertation

by

DWI HENDRATMO WIDYANTORO

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

John Yen (Co-Chair of Committee)	Thomas R. Ioerger (Co-Chair of Committee)
Richard Furuta (Member)	Reza Langari (Member)
	Valerie E. Taylor (Head of Department)

December 2003

Major Subject: Computer Science

ABSTRACT

Concept Drift Learning and Its Application to Adaptive Information Filtering.

(December 2003)

Dwi HendratmoWidyantoro, B.S., Institut Teknologi Bandung;

M.S., Texas A&M University

Co-Chairs of Advisory Committee: Dr. John Yen
Dr. Thomas R. Ioerger

Tracking the evolution of user interests is a problem instance of concept drift learning. Keeping track of multiple interest categories is a natural phenomenon as well as an interesting tracking problem because interests can emerge and diminish at different time frames. The first part of this dissertation presents a **M**ultiple **T**hree-**D**escriptor **R**epresentation (MTDR) algorithm, a novel algorithm for learning concept drift especially built for tracking the dynamics of multiple target concepts in the information filtering domain. The learning process of the algorithm combines the long-term and short-term interest (concept) models in an attempt to benefit from the strength of both models. The MTDR algorithm improves over existing concept drift learning algorithms in the domain.

Being able to track multiple target concepts with a few examples poses an even more important and challenging problem because casual users tend to be reluctant to provide the examples needed, and learning from a few labeled data is

generally difficult. The second part presents a computational **F**ramework for **E**xtending **I**ncomplete **L**abeled **D**ata **S**tream (FEILDS). The system modularly extends the capability of an existing concept drift learner in dealing with incomplete labeled data stream. It expands the learner's original input stream with relevant unlabeled data; the process generates a new stream with improved learnability. FEILDS employs a concept formation system for organizing its input stream into a concept (cluster) hierarchy. The system uses the concept and cluster hierarchy to identify the instance's concept and unlabeled data relevant to a concept. It also adopts the *persistence assumption* in temporal reasoning for inferring the relevance of concepts. Empirical evaluation indicates that FEILDS is able to improve the performance of existing learners particularly when learning from a stream with a few labeled data.

Lastly, a new concept formation algorithm, one of the key components in the FEILDS architecture, is presented. The main idea is to discover intrinsic hierarchical structures regardless of the class distribution and the shape of the input stream. Experimental evaluation shows that the algorithm is relatively robust to input ordering, consistently producing a hierarchy structure of high quality.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to Dr. John Yen, my advisory committee chair, for all of his time and guidance in the ways of the research world. Yen has been very supportive of my research in every way. When he moved from Texas A&M University to Pennsylvania State University in 2001, he managed to leave me with a full one year, unrestricted research assistantship from which I was able to accelerate my research work to the fullest extent. His commitment to supervise my work from a long distance remained strong, despite his hectic schedule. Special thanks go to Dr. Thomas Ioerger, my advisory committee co-chair, for his constructive suggestions regarding my research. A great amount of time has been spent by him to help me revise the writing of my dissertation. I would also like to particularly thank my committee members, Dr. Richard Furuta and Dr. Reza Langari, for seeing me through this long journey. Kreshna Gopal, a close friend of mine, has presented me with a very thorough proofreading of the many chapters in the nearly-final draft. Kathy Flores also proofread the earlier version of Chapter V. I am very grateful for their generous help. There are several institutions that have contributed financially to my research. I would like to thank the U.S. Army Research Laboratory Contract DAAD17-00-P-0649 and Dell Foundation. I would also like to thank the Department of Computer Science at Texas A&M University for providing the financial support during my final year.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Being able to infer the most up-to-date user interests is of great importance because it can help select new relevant information and also can be used to filter out incoming irrelevant information. Despite the vast availability of information on the Internet and the ease in information seeking provided by current search engines, most newly available information that is potentially useful remains unexploited without active participation of users for searching it. Users, on the one hand, often do not know what kind of new interesting information that will become available and when. The information providers, on the other hand, do not have any knowledge about the information need of users. Information agents can fill in the gap between users and information providers so that relevant information can be delivered to users in a timely fashion. It is not questionable that the ability of such agents to automatically track the change of user interests over time plays a vital role.

Keeping track of multiple target concepts is a natural phenomenon. As an example, users can have several topics of interest in which articles (broadcasting news) they prefer to read (listen). The number and the variety of interest categories

---

The journal model is *Journal of Artificial Intelligence Research*.

can change dynamically over time. That is, each of the topics of interest can have different durations and time frames. This demonstrates the significance and raises issues that must be dealt with by tracking multiple target concepts. Nonetheless, tracking a single target concept at a time is an inherent assumption behind the technique developed in many existing algorithms (Klinkenberg & Joachims, 2000; Klinkenberg, 1999; Klinkenberg & Renz, 1998; Widmer 1997; Widmer & Kubat, 1996).

Tracking the evolution of user interests over time from a sequence of relevance feedback documents is a problem instance of concept drift learning. The majority of existing concept drift learning algorithms typically requires a large number of labeled data in order to achieve performances at satisfactory levels; and these algorithms generally assume the availability of such labeled data. Although unlabeled data are widely available in information filtering domain, acquiring labeled data is indeed still very prohibitive. For example, casual users tend to be unwilling to provide the relevance feedback needed to label the data (Jansen, Spink, & Saracevic, 2000). Thus, learning concept drift from a sequence of few labeled data poses an important problem. Addressing this problem could contribute significantly not only to the information filtering domain, but also to the more general field of concept drift learning.

This dissertation develops an algorithm for learning concept drift in information filtering domains capable of handling multiple target concepts. It also develops a computational framework that extends existing concept drift learning

algorithms in the absence of complete labeled data. The next two sections briefly describes the distinction between (*conventional /stable*) *concept* learning and *concept drift* learning, and then describes a common approach as well as problems faced by existing concept drift learning algorithms. Section 1.3 presents a general approach for learning with a few labeled data in conventional concept learning. New approaches for learning concept drift that overcome the limitation of existing algorithms will be outlined in Section 1.4, followed by a summary of key contributions of the work in Section 1.5.

1.1 Concept Learning versus Concept Drift Learning

Concept learning is a process of inferring a Boolean-valued function from a set of input and output examples (Mitchell, 1997), i.e., f: $X \rightarrow \{1,0\}$ where X is the space of input examples. In the information filtering domain, the input is a document d and the output is the document relevance (e.g., either *relevant* or *irrelevant*). Conventional concept learning assumes that the target function is static, i.e., the relevance values of all documents with the same topic category are the same. Hence, the input and output examples in conventional concept learning can be given to the learner in any order. The target function values are often referred to as the data (concept) labels.

Concept drift learning is a concept learning in which the target function changes over time (Bartlett, David, & Kulkarni, 1996; Helmbold & Long, 1994; Schlimmer & Granger, 1986). For example, the relevance of documents of the same topic category could change from time to time. Hence, the target functions in concept

drift learning, in contrast to conventional concept learning, is dependent on the ordering of the input and output examples. Given a stream of pairs of input and output examples, the task of concept drift learning is to output a sequence of target functions where each target function inferred at time t can only utilize the data given before t. The problem of concept drift learning essentially consists of two sub-problems: learning stable concepts such as in conventional concept learning and adapting to changing labels of concepts.

The *drift rate* in concept drift learning is an essential parameter, denoting the probability that two successive target concepts c_i and c_{i+1} disagree on a randomly drawn example (Helmbold & Long, 1994), e.g., $\Pr(c_i \neq c_{i+1})$. Intuitively, slower drift rates correspond to learning from data streams whose target concepts change less frequently with respect to the number of seen examples, and vice versa. Slower drift rates can also be associated with concept drift learning on easier learning problems because more labeled data are available to learn the same target function before it changes. Therefore, when the number of data for learning the same target function is reduced, the drift rate increases and the learning problem becomes more difficult, which is one of the main issues addressed by this dissertation.

1.2 Existing General Approaches to Concept Drift Learning

Despite the differences of existing concept drift learning algorithms, most of them stem from the same approach in that the algorithm's ability to adapt to concept drift is achieved by learning from a single window of most recent examples (Widmer &

Kubat, 1996; Widmer, 1997; Klinkenberg & Renz, 1998; Klinkenberg & Joachims, 2000). Figure 1.1 illustrates this approach. Obviously, the approach automatically excludes older examples that are no longer relevant.

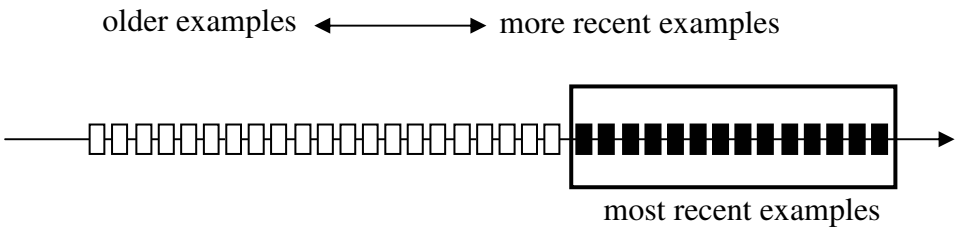


Figure 1.1: A typical approach to concept drift learning.

However, the single-window approach above suffers from the difficulty in determining the appropriate window size. The bottom line is that the drift rate is unknown *a priori* because it is impossible to predict when a concept change will actually happen, although its occurrence can be detected. Larger window sizes would enable learning with better performances on data stream with slower drift rates, and during which the target function is stable. However, it would take longer to get rid of non-relevant examples from the window when a concept change occurs, resulting in slower adaptation to the new target function. Fast changing target functions (i.e., more rapid drift rates) also could easily confuse the learner such as when target functions change two or more times within a window time frame. In contrast, smaller

window sizes allow quick adaptation to concept drift on either slower or faster drift rates. The disadvantage of having smaller window sizes is that it would never be able to learn stable concepts at the levels of accuracy desired even on the slow drift rate because of the smaller number of examples that are available for learning.

An adaptive window adjustment heuristic has been developed to address the window size determination problem (Widmer & Kubat, 1996; Klinkenberg & Renz, 1998), but most techniques employing the heuristic will work properly only on a learning setting with a slow drift rate. More specifically, an adaptive window adjustment heuristic determines the appropriate window sizes based on a trend in the system predictive performances, which are continuously monitored from the performance in predicting the last *m* seen labeled data. The window size is increased when the predictive performance is stable or improves, and is quickly decreased when a sudden performance drop is observed, indicating a concept drift. However, the system's predictive performance on which the heuristics depend cannot be reliably acquired on faster drift rates.

An algorithm for concept drift learning that learns from only a single window also could suffer from inability to track multiple target concepts simultaneously. Because the target concepts to be tracked could change at different time frames, not all of them can be optimally learned with a single-window approach even though all the target concepts change at slow drift rates. It is obvious that the expected performance will decrease as the number of target concepts to be tracked

simultaneously increases, not to mention if the target concepts also change at different rates.

To sum up, existing algorithms for learning concept drift are still inherently limited by to track a single target concept. Little effort, if any, has been devoted to deal with learning concept drift on a faster drift rate. The latter learning setting, as mentioned earlier, corresponds to learning concept drift with reduced number of labeled data.

1.3 Learning with Incomplete Labeled Data

Dealing with incomplete data is not a new problem. The Expectation Maximization (EM) algorithm (Dempster, Laird, & Rubin, 1977) is perhaps the first method to address this problem. The more recent algorithms include co-training (Blum & Mitchell, 1998; Nigam & Ghani, 2000), graph min-cut (Blum & Chawla, 2001), various techniques for query expansion in information retrieval (Mitra, Singhal, & Buckley, 1998; Buckley, Salton, Allan, & Singhal, 1995; Crouch, Crouch, Chen & Holtz, 2002; Xu & Croft, 1996; Iwayama, 2000), text classification (Nigam, McCallum, Thrun, & Mitchell, 2000), and various techniques developed for topic tracking in the Topic Detection and Tracking (TDT) evaluation (Allan, Papka, & Lavrenko, 1998; Yang, Pierce, & Carbonell, 1998; Yang *et al.*, 1999)

Basically, all these approaches are similar to one another in that artificially labeled data are incorporated to increase the number of labeled data used for learning. The additional data are selected automatically from unlabeled data, guided by some

kinds of similarity measures with the labeled data or objective functions. Provided that the unlabeled data that are truly relevant to the labeled data exist and can be correctly identified, this general approach should work very well without question. Despite the potential of identifying irrelevant unlabeled data that could have a detrimental effect, the existing methods are generally effective in their respective application domains.

However, the existing approaches for learning from labeled and unlabeled data assume the stability of concept being learned. These approaches are therefore not suitable for inducing concepts that change over time such as in concept drift learning.

1.4 New Approaches to Concept Drift Learning

The limitations of current approaches to concept (drift) learning, as described above, suggest two directions in which they can be improved. The first direction is to develop a method for tracking multiple target concepts simultaneously. The second direction is to devise a general method that addresses the problem of concept drift learning in the absence of completely labeled data.

1.4.1 Tracking the Changes of Multiple Target Concepts

The proposed method for tracking multiple target concepts is focused primarily on its application in information filtering domain. Conceptually, it extends a single-window approach by maintaining multiple window sets. Each window set is used for tracking a single target concept, and is dynamically created or deleted as necessary.

Furthermore, each set consists of two windows of large and small sizes. The former is intended to capture a stable concept such as general preference in a long-term interest, which allows capturing an accurate representation of the target concept. The latter window will be used for tracking the most recent tendency related to the target concept. The window with smaller size would facilitate a flexible adaptation to concept drift. The proposed method achieves a balance between the ability to learn stable concept and for quickly adapting to concept drift by learning each target concept with large and small window sizes simultaneously.

1.4.2 Concept Drift Learning in the Absence of Complete Labeled Data

Inspired by the success of techniques that combine labeled and unlabeled data in conventional concept learning, a similar technique is developed to learn concepts from a stream of labeled and unlabeled data. Assume that most data in the stream are unlabeled, and the labeled data are uniformly distributed in the stream. The sub-sequence of labeled data extracted from the stream is what is actually seen by the concept drift learning algorithm. It represents a stream of labeled data whose target functions change quickly one after another, i.e., fast drift rate. This dissertation proposes a general method for processing the sequence of labeled data, assumed to have a faster drift rate, into a longer sequence of labeled and artificially labeled data with a slower drift rate, which is easier to learn. The artificially labeled data are used to fill the gap in the labeled data, and are retrieved from relevant unlabeled data. A

concept drift learner can then be applied to learn the new stream. Figure 1.2 illustrates the idea in a simplified form.

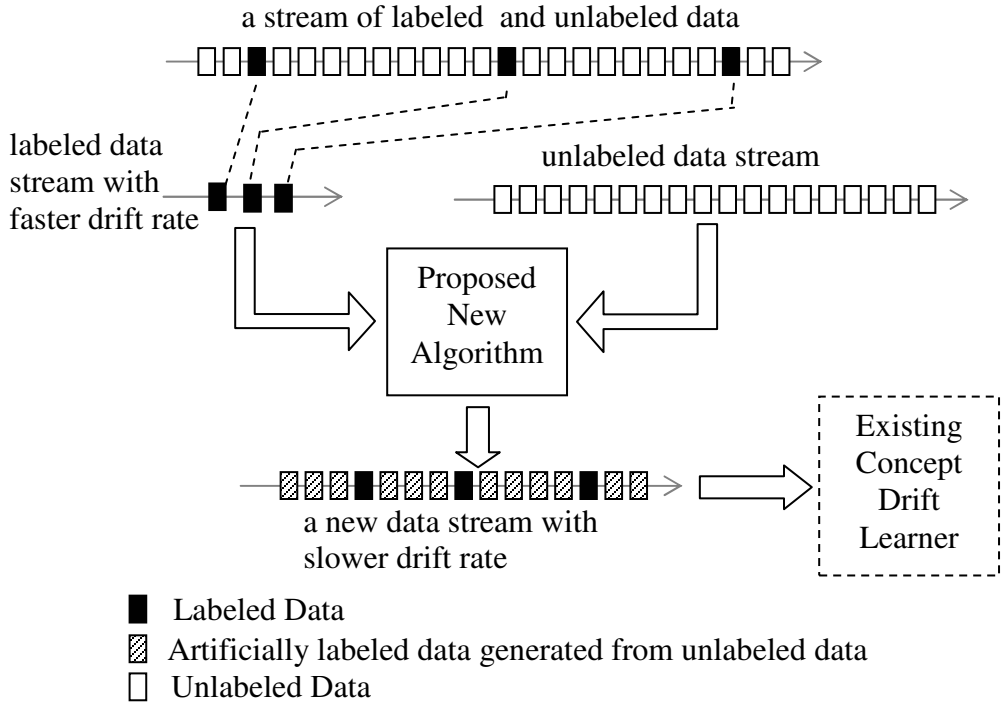


Figure 1.2: A new approach for incorporating unlabeled data in concept drift learning.

Unlike in stable concept learning, the subtlety in identifying relevant unlabeled data is more challenging in concept drift learning because the values of labels (target functions) in the labeled data can change over time. For example, a positively labeled instance A that appears earlier in the data stream will inherently change its label to negative when a new negative instance B with the same concept class as that of A is later presented. In this example, instances A and B are no longer

relevant and thus should be excluded from expanding the instances with relevant unlabeled data. Hence, incorporating unlabeled data in concept drift goes well beyond expanding labeled data with relevant unlabeled data because it also has to infer the changing labels of labeled data.

Furthermore, tracking the change of labels in a sequence involving a small set of labeled data is difficult, if not impossible, because the instance categories are typically unknown and cannot be induced reliably from the set. This dissertation addresses this problem by using labeled and unlabeled data observed from the input stream to predict the instance categories. In particular, a new concept formation system is employed to organize the data stream into a concept hierarchy in unsupervised mode. The concept hierarchy generated is basically a tree structure in which leaf nodes represent instances and internal nodes denote concepts that generalize their descendants. The concept category of an instance is then identified from one of its ancestors that best generalizes the instance.

1.5 Summary of Contributions

This dissertation investigates several aspects that have not been adequately addressed in concept drift learning, and develops a set of algorithms that directly or indirectly address them. In summary, this dissertation presents three contributions.

The first contribution is **a novel algorithm for tracking the evolution of user interests**. It provides a high-level approach for managing multiple windows in concept drift learning and a new strategy for striking a balance between long and

short window sizes. Its specific realization in information filtering domain is then presented. In this domain, the algorithm is able to learn flexibly the dynamics of user interests, including anticipation for long-term and short-term interests of the users (Widyantoro, Ioerger, & Yen, 2001).

The second contribution is a new computational framework for extending concept drift learning algorithms to deal with learning from a stream of sparsely labeled data. This dissertation describes a method for inferring the most up-to-date data labels and expanding the labeled data with relevant unlabeled data (Widyantoro, Ioerger, & Yen, 2003). In particular, it demonstrates how to incorporate a concept formation system, as well as the persistence assumption in temporal reasoning to do the task. The main role of the concept formation system is to build a concept hierarchy that will be used for identifying instance categories and retrieving relevant unlabeled data. The persistence assumption is adopted to infer the labels of instance categories. The method is general and can be viewed as the pre-processing step whose output can be used by virtually any existing concept drift learner.

The third contribution is **a new concept formation algorithm**. A new approach for concept formation is developed to provide a practical realization of the framework that pre-processes labeled data stream. The key idea in the algorithm is the exploitation of homogeneity and monotonicity properties of concept densities for guiding the incremental construction of a concept hierarchy from a data stream (Widyantoro, Ioerger, & Yen, 2002). The algorithm is relatively insensitive to some

degree to input ordering, and is capable of generating a quality hierarchy comparable to the quality of that of produced by typical non-incremental methods.

1.6 Roadmap

The rest of this dissertation is organized as follows. Chapter II presents a broad literature review regarding concept drift learning algorithms and systems, from practical machine learning approaches to theoretical results, to its application in intelligent agents and information filtering. Chapter III describes a novel concept drift learning algorithm for learning changing user interests, which is the first main contribution of this dissertation. It also describes other learning algorithms, and empirically evaluates their relative strengths and weaknesses. Chapter IV presents a computational framework so-called FEILDS that can extend the capability of an existing concept drift learning algorithm. One of the important components of FEILDS is a concept formation system. A new algorithm that realizes this concept formation system is described and fully evaluated in Chapter V. Finally, Chapter VI discusses the evaluation of FEILDS, followed by conclusions in Chapter VII.

CHAPTER II

LITERATURE REVIEW

This chapter reviews literatures related to concept drift learning and user interests modeling for adaptive information filtering. Section 2.1 surveys the underlying techniques of various practical systems for learning concept drift developed in the Machine Learning community. These systems include STAGGER (Schlimmer & Granger, 1986), FLORA (Widmer & Kubat, 1996), METAL (Widmer, 1997) and SPLICE (Harries, Sammut, & Horn, 1998), which have been designed and applied in non-information filtering domains. Section 2.2 describes methods and systems that have been developed for learning user interests in information filtering domains. The issues addressed by some of the works described in this section, particularly those that consider evolving user interests, represent a problem instance of concept drift learning in the domain. The last section provides overviews of existing theoretical results in concept drift learning.

2.1 Practical Concept Drift Learning Systems

Concept drift learning systems can differ from one another in (1) the representations of concept descriptions that affect the underlying concept learning algorithm, and in (2) the strategy in adapting to concept drift. Generally speaking, systems adapt to concept drift by deriving concept descriptions using a window of recent examples.

Alternatively, a form of meta-learning can be applied to explicitly detect a current context and then learn the concept descriptions from examples belonging to the current context. Table 2.1 summarizes the key features of four systems described in this section.

System Name	Concept Representation	Adaptation to Concept Drift
STAGGER (Schlimmer & Granger, 1986)	Weighted Boolean Functions	Thresholding the statistical counts
FLORA (Widmer & Kubat, 1996)	DNF without negation	Adaptive windowing
METAL (Widmer, 1997)	Probabilistic	Meta-learning and fixed-size windowing
SPLICE (Harries, Sammut, & Horn, 1998)	Decision tree	Meta-learning from batch process and/or windowing (optional)

Table 2.1: Key features of practical concept drift learning systems.

2.1.1 STAGGER

STAGGER is the first incremental learning system that addresses the concept drift problem (Schlimmer & Granger, 1986). Concept description in the system is a set of numerically weighted symbolic characterizations. Every characterization element is represented by Boolean functions of attribute-values, and is dually weighted using Bayesian weighting measures so-called logical sufficiency (*LS*), or positive likelihood ratio, and logical necessity (*LN*), or negative likelihood ratio. It determines the class

membership of a new instance from the *LS* weights of all matched characterizations and from the *LN* weights of all unmatched characterizations. The system also accumulates all the counts needed to calculate the Bayesian weighting measures as it moves forward over the data stream, allowing the Bayesian measure weights to be incrementally updated.

STAGGER seeks a succinct concept description that is generated from simple toward complicated descriptions. The concept description is refined only when the system fails to predict the class membership of a new instance. In such a case, the system applies a set of heuristics guided by the Bayesian evaluation measures to prune an established characterization that proves ineffective (i.e., its evaluation measure falls below a threshold) and/or to add a new generated characterization element whose weight surpasses the threshold. This process allows the system to respond quite effectively to concept drift.

Retaining the accumulation of all counts for the Bayesian measure update poses the strength as well as the weakness of the system. The history of counts has the effect of requiring about the same number of training instances to abandon a concept definition as that of instances to build it. This behavior, which is also empirically found in psychology of learning, allows STAGGER to model the resilience of concept learning appropriately. However, over-trained concept description also causes the system to slowly adapt to a new target concept when a concept drift does occur. Although not explicitly mentioned, STAGGER can be viewed as a concept drift learner with a single, very large window size and thus constitutes its weakness (The

window size in STAGGER actually increases linearly with the number of instance seen from the data stream, which can be considered as a window with infinite size).

2.1.2 FLORA Family Algorithms

Widmer and Kubat (1996) developed the FLORA family of learning algorithms. The system induces current target concept from a single window of recent examples by incrementally learning a new instance and forgetting the least recent one within its window. A concept description is represented by three description sets; one description covers both positive and negative examples within the window while the other two consistently cover only positive instances and only negative instances, respectively. Each description set is essentially a disjunctive normal form (DNF) formula without negation. The prediction of a new instance is based on its match with the description set covering only the positive instances.

FLORA-2 is the first realization of the FLORA algorithm that dynamically adjusts the window size during the learning process. The window size is quickly reduced when a concept drift is suspected, allowing the system to rapidly forget irrelevant older instances and focus only on examples relevant to a new concept. It then gradually increases the window size until a stable concept is reached in which case the window size is kept fixed. The adjustment of window size is based on two indicators: system's performance and the complexity of concept description. The first indicator is continuously monitored from the past prediction on a fixed number of recent instances. In the second indicator, the number of description items needed to

cover instances determines concept description complexity. A low system performance or a high number of description items is an indicator for the concept drift occurrence. This ability allows FLORA-2 to flexibly respond to concept drift and can avoid the problem of slow adaptation in an over-trained concept.

Another version, FLORA-3, is also able to store a new stable concept established from examples in the window and re-use them later when context change occurs and one of the stored concepts fits the current situation. When a concept drift is detected, the system will find the best candidate among the stored concepts based on their performance on classifying instances in the current window. The best candidate is then re-generalized using examples in the current window. If the updated best candidate is better than the current concept description, with respect to the concept complexity, then the generalized best candidate will replace the current concept description. The empirical experiments that had been conducted reveal that retrieval and modification of stored concepts increase the system's performance if old concepts do re-appear but it also could be erroneous by replacing current concept with a wrong stored concept.

The last version, FLORA-4, was developed to counter the brittleness of FLORA-2 and FLORA-3 in dealing with noise. As its predecessors strictly maintain the consistency of their concept descriptions with respect to the instances covered, the presence of noise in the instances causes unstable behavior that unnecessarily change the concept descriptions. FLORA-4 addresses this problem by applying a statistical confidence measure in maintaining the set of reliable generalizations.

Despite the flexibility in reacting to concept drift and in handling noise, the FLORA family of algorithms in general is designed under two assumptions. First, the rate of change in the target concept is rather low, which gives a chance for the system to see a sufficient number of instances for establishing a stable concept during the phases between periods of change. Clearly, the system will not work properly, at least not producing satisfactory performances, if the rate of change is high.

The second assumption is that only the latest examples, which are kept in the window, are relevant to current target concept. Although this assumption is reasonable as well as intuitive, particularly in dealing with concept drift, it is inherently limited to tracking only a single target concept at a time. In multiple-concept tracking, however, not all data representing the instances of current target concepts are recent because the relevance of some concepts introduced earlier may not have been denied, i.e., still being a part of target concepts. In contrast, some concepts introduced at later time may be no longer relevant. Therefore, the instance recency assumption does not hold in the case of multiple-concept tracking. Although FLORA-3 is able to store and retrieve old concepts, its sole purpose is to speed up the learning of recurrent concept while the underlying problem remains. Increasing the window size to the extent that will include the older target concepts does not help overcoming the problem because irrelevant instances may still lie between the most and the least recent examples in the window.

2.1.3 METAL Family Algorithms

In more recent work, Widmer (1997) exploits contextual clues, i.e., context-defining attributes, for tracking context changes. Borrowing his example, a person driving through a country border is likely to notice a systematic change in the distribution of vehicle license plates. In this example, license plate is the contextual attribute that indicates a change of the environment, suggesting one to adapt to the new rule. The contextual attributes, which are essentially not different from other attributes, are automatically detected by the learning system provided that such attributes exist. More specifically, an attribute is considered to be contextual if the distribution of its feature (i.e., attribute value) that co-occurs with a predictive feature is significantly different (measured by χ^2, i.e., the *chi-square* statistic) from the unconditioned distribution of the predictive feature. A predictive feature is an attribute value whose distribution in a class within a fixed window of recent instances is significant (also measured by χ^2).

Widmer proposed a two-level learning model consisting of a *meta-learner* and a *base level learner* that can detect contextual clues and react accordingly to a context change. Given a new instance, the meta-learner attempts to identify the contextual clues using the whole history of instances. The *base level learner* performs the classification task of the new instance; the contextual attributes identified by the meta-learner are used to focus the learning process on information in the window that is relevant to current context.

Two specific systems from this general model have been implemented in METAL-B and METAL-IB. The former version uses a naïve Bayesian classifier as the underlying learner. The *base level learner* in METAL-B performs the classification task based on learning from instances in the window whose contextual attribute values are the same (appears to belong to the same context) as that of the new instance to be classified. If no contextual attribute is found by the meta-learner, then all instances in the window are used for classification. The latter version, METAL-IB, employs instance-based classifiers as their underlying learning algorithms. The contextual information in METAL-IB is used for feature and exemplar weighting.

The METAL systems can be viewed as concept drift algorithms employing two windows. One window, which is used for the meta-learner, has a large size, increasing linearly with the number of instances. Another is the fixed size window that supplies the instances to the base-level learner for classification. Although the detection of concept drift is handled by the meta-learner, determining the right window size for the base-level learner is still a tricky issue. As discussed in (Widmer, 1997), the effectiveness of the meta-learner diminishes with the smaller window size and if the window is too narrow, the base-level learner lacks of needed data to learn the context. Too large a window, on the other hand, could introduce many conflicting instances that would prevent the system from finding predictive features and then could disallow the meta-learner from identifying contextual attributes. Moreover, since the actual concept is derived from the fixed size window of recent examples, the

METAL systems in general inherit the limitation of the single windowing approach, i.e., tracking only a single target concept.

2.1.4 SPLICE-based Family Algorithms

Harries *et al*. proposed a different approach for concept drift learning (Harries, Sammut, & Horn, 1998; Harries & Horn, 1998). Unlike typical concept drift learner in that the learning process is on-line and incremental, they take an off-line, batch learning approach in a supervised mode. During the batch-learning phase, the system attempts to identify a set of stable concepts through *contextual clustering* based on the regularities that emerge from a given training data sequence. It then uses the identified stable concepts as the basis for on-line prediction.

A family of SPLICE algorithms has been developed to perform contextual clustering from a training data sequence. Each instance is time-stamped based on its position in the sequence. The time stamps given to the training data form a continuous attribute that can indicate a change of context in the data series. A decision tree is then induced from the training data set using a batch learner (e.g., Quinlan's C4.5). Any test on attribute *time* in the induced decision tree is used to partition the data set into intervals and their partial concepts. A contextual cluster is identified from a set of intervals that have similar contexts, i.e., if the partial concept of one interval also covers the instances in another interval. The SPLICE-1 algorithm obtains the final stable concepts by applying C4.5 again on the resulting initial contextual clusters (Harries & Horn, 1998). The SPLICE-2 algorithm improves the

quality of partitions by iteratively refining the boundaries of contextual clusters (Harries, Sammut, & Horn, 1998) until a maximum number of iterations has been reached or no change happens in the last two iterations.

The system performs on-line prediction using a suitable stable concept that has been identified during the off-line learning phase. Two alternative methods have been suggested for selecting the most appropriate stable concept. The first method is to use a simple voting mechanism that selects a stable concept with highest classification accuracy on a window of recent instances. The second method is to apply a *meta-classifier* on a new instance for deciding which stable concept is appropriate for predicting the instance. After stable concepts have been identified during the off-line learning, all training data are copied and re-labeled by their corresponding stable concepts. The *meta-classifier* can then be constructed using C4.5 on the newly labeled training set.

The system's performance thus depends on the quality of stable concepts and the ability to correctly select a stable concept for classification. It adapts to concept drift by switching from one stable concept to another, similar to the FLORA-3 algorithm that retrieves a stored concept, during the prediction processes. The use of a window for selecting a stable concept, as described above, limits the system's ability to tracking only a single target concept while employing a meta-classifier for the selection process enables the system, at least theoretically, to learn multiple target concepts simultaneously. However, because the stable concepts are identified only

during the off-line learning, during the prediction phase the system is unable to predict if an instance belong to a new stable concept.

The method proposed in this research is similar to those of FLORA-3 and SPLICE in that stable concepts are stored and re-used whenever needed. These stable concepts are continuously maintained in the concepts hierarchy. It is also similar to METAL and SPLICE in that context is exploited in meta-level learning to detect the presence of concept drift. However, the existing methods assume the existence of a large number of labeled examples in order to work properly despite the similarities. In contrast, the proposed method is specifically designed to work when the number of labeled examples is much less.

2.2 Approaches to Adaptive Information Filtering

Information filtering is a task that classifies texts from a stream of text documents into either a relevant or an irrelevant category with regard to a user's interests (Hull, 1998). This section describes major methods for modeling user profiles, which provide the basis for the information filtering task. Changing interests of the user over time in such an environment is inevitable so that a system that performs the task must be able to continuously adapt to the new user interests, i.e., by learning from the user relevance feedback, in order to maintain the system's high performance. Thus, tracking a user's interests represents concept drift learning in information filtering/retrieval domain. Similar to the concept drift learning, most works in

adaptive information filtering suffer from requiring a large number of labeled examples.

2.2.1 Rocchio Algorithms

Rocchio's relevance feedback is an algorithm for learning user interests that has been well studied in information retrieval (Rocchio, 1971; Salton & McGill, 1983). Systems employing the Rocchio algorithm typically assume the stability of user interests and apply the algorithm as a batch process. The algorithm nevertheless can be straightforwardly modified to learn a sequence of feedback documents incrementally, and hence is able to adapt to changing user interests. The adaptability to react to the changing interests can be controlled from the weights assigned to a positive and a negative feedback document. However, the linearity in updating the user interest representation makes it difficult to quickly remove a long-standing interest, similar to the problem faced by the STAGGER algorithm. The single descriptor representation of the Rocchio algorithm also inherently reduces the algorithm's ability to learn multiple interest categories.

Allan (1996) explores the effectiveness of the Rocchio algorithm for information filtering by employing incremental feedback technique. Allan's experiments demonstrate that comparable results with the full judgments could be obtained using only a few incremental judgments (e.g., 10% of full judgments, corresponds roughly from 7 to 30 documents per query). He also empirically showed that the drift of user queries, i.e., queries whose notions of relevance change, could be

handled gracefully only when the greater proportion of feedback documents comes from the more relevant context. In spite of its importance in adapting to a new interest, determining the appropriate number of recent relevance judgments within a window remains an unsolved issue from this work.

An adaptive text-filtering task that performs on-line learning from an incoming stream of documents has been the research focus of the TREC-7 filtering track (Hull, 1998). Rocchio's relevance feedback algorithm is adopted in most systems participating in this track, and the best performance is achieved by systems that perform adaptive thresholding, little learning and minimal query expansion.

2.2.2 Window-based Approaches

Klinkenberg and Renz (1998) address the problem of tracking user interests using a window of recent document feedback. Unlike in typical on-line learning setting, the method assumes that the input of data stream arrives in batches, each batch containing an equal size of document set (e.g., 130 documents in this case). The window size, which is measured by the number of batches, is adaptively adjusted by monitoring the system's predictive performance. Specifically, the deviation of the system's predictive *accuracy*, *precision* or *recall* from learning documents in the window is an indicator of change in interests. Based on the extent to which the current system's predictive performance deviates from its average performance (over the last *m* batches), the window size is adjusted accordingly similar to the window adjustment of the FLORA-2 algorithm. The system adapts to the new interests by relearning

batches in the window. Klinkenberg and Renz experiment with various classifiers (e.g., Rocchio, Naïve Bayes, KNN, C4.5, etc.) and show that systems with adaptive window sizes consistently outperform those that employ fixed window sizes and those that learn only from documents in the last batch.

More recently, Klinkenberg and Joachims (2000) propose another window adjustment algorithm, also in a setting where the input stream is in a form of batch sequence. The window size is dynamically determined so that it maximizes the system's predictive performance on the last batch. More specifically, it trains the Support Vector Machine (SVM) classifier using various window sizes on previously seen batches, except the last batch, and selects the window size that minimizes the estimated generalization error on examples in the last batch seen. A further attempt has also been made to extend the work by employing Transductive SVM (TSVM) instead of the standard SVM classifier for solving a similar problem to that addressed in this dissertation (Klinkenberg, 2001). TSVM is an extension of SVM that takes into account unlabeled data on the test set (the next batch data) during the learning process so that the misclassification of data in that particular test set is minimized. However, this approach has never been evaluated using fewer labeled examples, making the effectiveness of this method unclear. Besides, there is still a controversy regarding the TSVM classifier itself. Specifically, using the test set for learning is *invalid* as a means of inductive inference in the first place. An analysis based on the standard Maximum Likelihood Estimate (MLE) / Fisher information also indicates

that TSVM in its current form is likely not to be helpful in general because it may mislead the classifier into maximizing wrong margins (Zhang & Oles, 2000).

2.2.3 Intelligent Agents for Information Filtering

Modeling user interests has also been an active research area in the Intelligent Agents community dealing with information filtering related problems. Although many agent systems with embedded user-profile learning modules have been developed, only a few of them address the problem of changing interests. Among of these agents are PVA (Chen, Chen, & Sun, 2002), ALIPES (Widyantoro, Ioerger, & Yen, 1999), FAB (Balabanović, 1997 & 1998), SIFTER (Lam, Mukhopadhay, Mostafa, & Palakal, 1996), AMALTHEA (Moukas & Zacharia, 1997), NEWT (Sheth, 1993). An interest category in the profile of these agents is represented by a descriptor (feature vectors), which is a list of feature and its weight pairs.

NEWT and AMALTHEA are multi-agent systems for personalized information filtering. Both systems employ evolutionary algorithms where populations are composed of individual agents each of which acts as a filter for an interest category. When the user interests change, the filter agents assigned to the old interests are eventually left out from the population by evolution and natural selection while new individual agents are created to filter the new interests. The fitness of each agent, which affects the agent survivability, is determined from the user's relevance feedback. AMALTHEA is essentially an extension of NEWT. While NEWT employed only a single type of agent (e.g., those for information filtering), AMALTHEA also

introduces information discovery agents whose relationship with the filtering agents is based on a simple economic model. Due to the nature of the algorithm, a great amount of effort from the user is required to rate information received.

SIFTER (**S**mart **I**nformation **F**iltering **T**echnology for **E**lectronic **R**esources) is a document filtering system developed by Lam *et al.* (1996). The system has been applied to filtering LISTSERV mails as well as research reports in computer science domain. Its algorithm for updating the user profile, which is designed to be able to detect and adapt to the shift in user interests, consists of two-level (meta-level like) learning approaches. The lower level employs a standard reinforcement-learning algorithm to learn the user interests. The upper level uses a Bayesian method to detect changes in the user model. The learning process in the lower level is reinitialized when the upper level detects the shift in the user interests.

FAB is a Web page recommendation service that combines the technique based on the Web page contents and the recommendations of other users, often called as collaborative filtering (Balabanović, 1997 & 1998). It uses the user feedback to update its user profile, which constitutes short-term learning. A user's interest that changes over time is modeled using a simple decay mechanism. For example, all weights in the profiles are multiplied by 0.97 at regular intervals.

ALIPES is a newsagent that regularly retrieves information from on-line newspapers and magazines on the Internet and presents a personalized news page to its users (Widyantoro, Yin, Seif El-Nasr, Yang, Zacchi, & Yen, 1999). A user's

interest category in this system is decomposed into long-term and short-term interest models, and the user profile maintains the representation of multiple interest categories. The system learns the user profile from explicit user feedback and adapts to changing user interests by exploiting negative examples and decaying the user profile's weights. The MTDR learning algorithm described in Chapter III in this dissertation is a significant improvement and refinement over the original learning algorithm of ALIPES.

Personal View Agent (PVA) is a software agent for tracking, learning and automatically organizing documents from the Internet (Chen, Chen, & Sun, 2002). A proxy (one of the system's components) logs every browsing request made by a user and the system uses this information to build the user's profile, assuming that a document visited longer that a threshold (e.g., 2 minutes) can serve as a positive feedback document. This allows a user profile to be learned automatically without requiring an explicit user feedback. A user profile in the system is represented by a category hierarchy called a *personal view*. The personal view is dynamically constructed based on the implicit feedback document received from the proxy whose classification in the personal view is guided by a pre-defined master category hierarchy called *world view*. PVA adapts to changing user interests by decaying the feedback document, which will eventually remove any interest category that has not been recently visited from the personal view. This method is essentially the same as learning from a window of recent interest categories seen.

System	Main Methods
SYSKILL & WEBERT (Pazzani & Billsus, 1997)	Naïve Bayes Classifier
NEWSDUDE (Billsus & Pazzani, 1999)	Hybrid Naïve Bayes and Nearest Neighbor
WEBMATE (Chen & Sycara, 1998)	Multiple TFIDF-based descriptor representations
NEWSWEEDER (Lang, 1995)	Minimum Description Length algorithm
WAIR (Seo & Zhang, 2000)	Implicit Feedback and Reinforcement Learning
PIN (Tan & Teo, 1998)	Fuzzy Adaptive Resonance Associative Map
INFOSCOPE (Fischer & Stevens, 1991)	Heuristic rules for automatic profile generation and direct profile update by user
SIFT (Yan & Garcia-Molina, 1999)	User-supplied keywords and relevance feedback

Table 2.2: A list of traditional information filtering systems.

There are also many other systems that have been developed for information filtering task where their learning algorithms cannot be used (or have not been designed) to handle the changes in user interests. Table 2.2 summarizes most of these systems and their major learning techniques or features. These systems either simply adopt the standard convergence-type machine-learning algorithm or employ a single-descriptor model for the representation of user profile. The typical machine learning

algorithms applied on some of these systems cannot be applied in an on-line fashion, which limits their utility.

Employing a single-descriptor representation, which is a single list of features and their weights, lacks the capability to adapt flexibly to a user's changes in interests. Given a sequence of negative feedback to a previously learned interest category, and/or a sequence of positive feedback representing a new interest to be learned, an algorithm built on this single-descriptor representation adapts to this new interest at a fixed, pre-determined pace. Systems that employ a single-descriptor representation as above make an implicit assumption that user interests change at a constant rate.

2.3 Theoretical Results on Concept Drift Learning

Concept drift learning has also been studied in the field of computational learning theory. Results from this field mainly establish theoretical bounds based on some assumptions regarding the number of examples to be tracked within a window and the kind of drift that can occur.

The first theoretical studies on tracking a concept as it evolves over time have been conducted by Kuh, Petsche, and Rivest (1991). They provide bounds on the number of examples needed for adapting to concept changes and the maximum rate of concept changes that can be tracked by a batch tracker (a tracking algorithm that maintains a sliding window of recent examples and learns from all examples in the window). The bounds are dependent only the complexity of target concepts,

theoretically measured by the VC-dimension of the concepts. The adaptation to a new concept is faster if the new concept is similar to the previous concept.

Helmbold and Long (1994) analyze a concept drift problem on domains whose target concepts change continuously but at a slow drift rate. They evaluate tracking algorithms that minimize the number of disagreements with the most recent examples based on the rate of target concept movement that can be tolerated between examples. More specifically, a general-purpose algorithm can tolerate concept drift rates up to $c_1\varepsilon^2/(d\ln\frac{1}{\varepsilon})$ where ε is the desired error rate, and d is the Vapnik-Chervonenkis dimension of the concept class (Blummer *et al.*, 1989). A more computationally efficient variant of this algorithm can tolerate target concept movements of at most $c_2\varepsilon^2/(d^2\ln\frac{1}{\varepsilon})$. They also provide results for the classes of half-spaces and axis-aligned hyper-rectangles showing that no algorithm can tolerate a concept drift greater than $c_3\varepsilon^2/n$.

The main result above is essentially a special case of a later work due to Barve and Long (1997), which constrains the allowable drift rate by ensuring that consecutive probability distributions have small total variation distance. The result was subsequently improved by Long (1998) to ε^3/d for agnostic learning and to ε^2/d for the realizable case.

Blum and Chalasani (1992) address the problem of learning switching concepts. Rather than slowly drifting through the concept spaces, their work allows to

switch between concepts in the class, representing a target concept that changes rapidly and abruptly. They restrict their framework on the number of concepts visited, or on the frequency of switching. The main results are mainly the computational complexity of predicting switching concepts on various switching concept models.

Bartlett, David and Kulkarni (1996) investigate the estimation of a target function sequence from a sequence of labeled, random examples. They provide the bounds on the sample complexity and the allowable drift rate of the target function estimation problem on three models. The first model allows infrequent but arbitrary changes of target concept, similar to Blum and Chalasani's work switching concepts. The second model allows target concept changes that correspond to slow walks on a graph whose nodes are functions. The last model limits the changes to small concept sizes, measured by the disagreement between consecutive target functions. They also studied the sample complexity and drift rate bounds for prediction of changing concepts.

WINNOW is an on-line algorithm for learning *k*-literal disjunctions that associates each disjunction with a weight and performs multiplicative update to its weights. Auer and Warmuth (1998) extend the WINNOW algorithm into SWIN (shifting WINNOW) to deal with target concepts that change over time. SWIN makes a stochastic prediction that returns one with a probability equal to the current weights. The weights of disjunction are updated only when SWIN makes a prediction mistake, and lower bound weights are added to guarantee a quick adaptation to the changes of

disjunction. They also provide worst-case bounds on the expected number of mistakes on any sequence of examples and any kind of target drift.

Herbster and Warmuth (1998) consider the problem of on-line prediction from a pool of experts in which the best expert might change as the patterns in the on-line sequence change. They extend the weighted majority algorithm (Littlestone & Warmuth, 1994) that maintains a single weight for each expert. The master algorithm combines the predictions of each expert according to their current weights. The experts' weights are then exponentially updated with respect to the past loss incurred by each expert. In order to be able to effectively track the sequence of best experts, they also redistribute a portion of an expert weight to the weights of other experts. Their theoretical results are mainly proofs for the guaranteed loss bounds of the master algorithm, relative to the loss of the best expert, for a variety of weight redistribution methods.

More recently, Bousquet and Warmuth (2002) propose a method for tracking a sequence of best experts in domains where the experts in the best partitions are from a small pool of m out a much larger set of n experts. Building on the methods developed by Herbster and Warmuth, they solve the problem by adding a *mixing update* that takes into account past posteriors to update the current weight of each expert. Loss bounds analysis on various coefficient-mixing schemes are also provided.

CHAPTER III

ALGORITHMS

FOR LEARNING CHANGING USER INTERESTS

This chapter presents a novel concept drift learning algorithm specifically developed for information filtering domains. The primary focus is to describe and evaluate a **M**ultiple **T**hree-**D**escriptor **R**epresentation (MTDR)-based approach for learning changes in user interests. Significantly refined from a master thesis work (Widyantoro, 1999), this chapter extends a previous work (Widyantoro, Ioerger, & Yen, 2001) by (1) evaluating the MTDR algorithm effectiveness on other aspects using a larger test collection, and (2) providing its performance comparison with other major algorithms. It demonstrates the advantage of MTDR algorithm for tracking multiple target concepts simultaneously, particularly when the tracking task involves long-live and short-live target concepts. This chapter also points out the limitation of MTDR and other existing concept drift learning algorithms for learning from a stream containing a few labeled examples.

The rest of this chapter is organized as follows. Section 3.1 describes basic representations and related techniques typically employed in information filtering domain. The next three following sections describe the Rocchio algorithm, the MTDR algorithm and two other generic concept drift learning algorithms based on the window of recent examples. The experiment procedures are presented in Section

3.5, and the experiment results on various concept drift learning tasks are then discussed in Section 3.6. This chapter concludes by summarizing the key contributions of this chapter in the last section.

3.1 Text Document Processing

The vector space model (Witten, Moffat, & Bell, 1994) is a commonly used representation for describing text documents. In this model, the content of a text document is represented by a feature vector in *n*-dimensional space where *n* is the number of unique terms contained in a document collection. Let *D* be a text document, then $\{(t_1, w_1), (t_2, w_2), \ldots, (t_n, w_n)\}$ is the feature vector of *D* where *t* is a term (word) and *w* is the weight of term *t*.

Weighting Document Terms. The text document representation as above requires a method to determine the weight of each term. A term's weight represents the degree of importance of the term in a document. A term that is more important is usually assigned a higher value than a less important one. **T**erm **F**requency-**I**nverse **D**ocument **F**requency (TF-IDF) is one of the major weighing schemes that has been well studied in the information retrieval literature. This weighing method assumes that terms that occur in fewer documents are better discriminators. If two terms occur with the same frequency in a document, the term occurring less frequently in other documents will be assigned a higher value. More specifically, the importance of a term is proportional to the occurrence frequency of the term in each document, and inversely proportional to the total number of documents to which the term occurs in a

given document collection (Salton & McGill, 1983). The importance of word *i*, denoted by w_i, in a document *D* is calculated as follows:

$$w_i = \frac{TF_i}{|D|} \log\left(\frac{N}{DF_i}\right) \tag{3.1}$$

where TF_i is the frequency of occurrence of term $\underline{t_i}$ in *D*, DF_i is the corresponding document frequency, and *N* is the number of documents in the collection. $|D| = \sum_j TF_j$ is the length of document, and is used to normalize term frequency in order to avoid favoring long documents over short documents.

Measuring Document Similarity. Given two document feature vectors, a similarity measure is needed to assess the degree to which a document matches a reference feature vector. This metric is usually used to evaluate documents in order to rank them and then filter those that are not relevant to the user interest. In the vector space model, the *cosine* coefficient is the most widely used similarity measure (Salton & McGill, 1983). The cosine coefficient calculates the difference in direction between two feature vectors, measuring the angle between these feature vectors, irrespective of their length. Given documents D_i and D_j, the similarity between the feature vectors of the two documents according to cosine formula is given by the following equation:

$$\text{sim}(D_i, D_j) = \cos\theta(D_i, D_j) = \frac{D_i \cdot D_j}{\sqrt{|D_i|} \times \sqrt{|D_j|}} \tag{3.2}$$

Text document representation is generally employed to represent a user's interest because the latter is often inferred from the former. Hence, the cosine similarity measure above is also an appropriate method for measuring the degree of interests. A set of documents is considered relevant to a user's interests if the cosine similarity of the two in vector space representation is high (e.g., closer to one).

3.2 Rocchio Algorithm

Rocchio relevance feedback is a query expansion mechanism for improving the quality of retrieval results based on relevance feedback in a static collection. It works by iteratively reformulating a new query from (1) the query of the preceding retrieval request, and (2) a set of relevant and irrelevant documents. Specifically, a query at a particular iteration t, denoted by Q_t, is of the form of vector $Q_t = \{(t_1, w_1), (t_2, w_2), \ldots, (t_n, w_n)\}$, containing a set of weighted words similar to documents retrieved so far. The relevance feedback process then generates the new query for the next retrieval iteration $Q_{t+1} = \{(t_1, w_1'), (t_2, w_2'), \ldots, (t_n, w_n')\}$ with altered weights w_i'.

The original Rocchio algorithm for query expansion during the relevance feedback process is as follows (Rocchio, 1971; Salton & Buckley, 1990):

$$Q_t = Q_{t-1} + \beta \frac{1}{n_{pos}} \sum_{pos} D_i - (1-\beta) \frac{1}{n_{neg}} \sum_{neg} D_j \qquad (3.3)$$

where β is positive constant between 0.0 and 1.0, n_{pos} is the number of relevant documents, and n_{neg} is the number of non-relevant documents. The parameter β

determines the amount of influence of relevant documents relative to irrelevant documents in query modification. The document retrieval process uses the cosine similarity measure to rank the retrieval results according to the similarity between the new query and documents in the collection.

Although originally designed to work in a batch process, the Rocchio algorithm can be easily adapted for learning changing user interests in incremental setting. The algorithm in this setting learns one document, either relevant (positive) document D_{pos} or non-relevant (negative) document D_{neg} at a time, practically setting $n_{pos} = n_{neg} = 1$ in the Rocchio algorithm. Naturally, the Rocchio representation is suitable for tracking a single target concept. As will be described shortly, this algorithm is also adopted for modeling the long-term interest of the MTDR algorithm.

3.3 MTDR Algorithm

MTDR algorithm is a concept drift learning algorithm that is crafted for tracking multiple target concepts in information filtering domain (Widyantoro, Ioerger, & Yen, 2001). The ability for tracking multiple target concepts is based on the observation that one can have several interest categories at the same time. The development of the algorithm is motivated by the need for capturing the long-term and short-term components of an interest category. The algorithm also inherently adopts the persistence assumption, which allows it to adapt to the change of short-term interests without disrupting the presence of long-term interests, and vice versa.

Long-term interests (e.g., interests in a research area) represent a user's general preferences (Billsus & Pazzani, 1999; Widyantoro, Ioerger, & Yen, 1999). These interests are formed gradually over the long run, and are fairly stable after they converge. In a concept drift algorithm that learns from a window of recent examples, the long-term interest model corresponds to the stable concept and can be acquired by applying large window sizes. Consequently, long-term interests tend to be inert, and the effort it takes to change the long-term interests could be proportional to the effort it takes to build them. On the other hand, short-term interests are very unstable by nature. For example, interests in current hot topics can change on a day-to-day basis. Such interests are inevitable and a common phenomenon in real life. Applying small window sizes can capture the short-term interests that correspond to unstable concepts in concept drift learning, enabling one to keep up with changes in the world quickly.

The MTDR algorithm attempts to learn the long-term and short-term interest models of an interest category and then to tradeoff the shortcomings and benefits between these two models. Conceptually, each interest category can be derived from a large and a small window of recent examples that are maintained simultaneously. Hence, it would require multiple window sets for tracking multiple interest categories. Instead of maintaining explicit windows, the MTDR algorithm creates explicit representations for each interest category model and applies an incremental update method that mimics the behaviors of having a large and small window in a window-based approach.

3.3.1 Interest Category Representation

The rationale is to represent long-term and short-term interest models of an interest category in separate descriptors and then to combine both models to get a more expressive representation; that is, a three-descriptor representation. The three-descriptor model is then extended to learn multiple user interest categories.

Modeling Long-term Interests. The motivation behind the modeling of long-term interests is to capture a user's general interests (i.e., stable concepts). The long-term interest model is built up gradually and the performance of the model is expected to improve consistently in-line with the increasing number of feedback examples learned. Consequently, the long-term interest model lacks the ability to respond promptly to recent feedback particularly when the model has learned from a large number of examples in the past.

The Rocchio algorithm satisfies the requirements to model long-term user interests over the long run since the effect of the Rocchio weight update rule is to cause a gradual change in interests. A long-term interest is modeled by a long-term descriptor *LTD*, which is updated using the following learning rule adopted from the Rocchio algorithm:

$$LTD_t = LTD_{t-1} + \beta D_{pos} - (1-\beta) D_{neg} \tag{3.4}$$

where $0 \le \beta \le 1$. The degree of interest in D with respect to *LTD*, denoted by $I_{LTD}(D)$, is simply the similarity value between D and *LTD*, i.e., $I_{LTD}(D) = \text{sim}(D, LTD)$.

Modeling Short-term Interests. The objective of short-term interest modeling is to adapt quickly to recent feedback. This ability is crucial particularly when the recent feedback reflects transient interests of the user. During the transition of change in an interest category (i.e., concept drift), the short-term interest models are also expected to help quickly eliminate the influence of the hard-to-forget long-term interests.

The short-term interest component is modeled by a pair of descriptors $STD = (PosD, NegD)$ where $PosD$ is a *positive descriptor* for representing the category of recent interest, and $NegD$ is a *negative descriptor* for representing specific subject not of interest. Given a positive feedback document D, the update of the positive descriptor is carried out as follows:

$$PosD_t = (1-\alpha) PosD_{t-1} + \alpha\, D \tag{3.5}$$

where $\alpha = (0,1)$ is the learning rate. It can be easily shown that for a sequence of positive documents D_i, the positive descriptor can be formulated by

$$PosD_t = \sum_{i=1}^{t} \alpha (1-\alpha)^{t-i} D_i \tag{3.6}$$

A similar computation is defined for learning from a negative feedback document by exchanging $PosD$ and $NegD$.

The cumulative discounted weight update rules applied for the short-term descriptors allow new interests to take over the representation space of the old interests as quickly as needed by setting the appropriate value of the learning rate.

The degree of interest in a document *D* according to the short-term interest model, denoted by $I_{STD}(D)$, is given by the difference between the similarities of *D* to the positive and negative descriptors.

$$I_{STD}(D) = \text{sim}(PosD, D) - \text{sim}(NegD, D) \tag{3.7}$$

Positive value of $I_{STD}(D)$ indicates that *D* is interesting, and vice versa.

An Interest Category Model. An interest category is represented by three descriptors combining the descriptors from the long-term and short-term interest models. Thus, an interest category is a three-descriptor model $TDR = (LTD, STD)$. Given a document *D*, the interest in *D* according to *TDR*, denoted by $I_{TDR}(D)$, is a mixture of the interests according to the long-term and short-term models, defined by

$$I_{TDR}(D) = \eta \ I_{LTD}(D) + (1-\eta) \ I_{STD}(D) \tag{3.8}$$

where η is a constant parameter between 0.0 and 1.0 that determines the impact of the long-term and short-term interest models in the three-descriptor model.

3.3.2 Learning Multiple Interest Categories

The three-descriptor model is designed to learn a single interest-category concept. This section describes an extension of this model for learning multiple interest categories using multiple three-descriptor representations. In principle, the algorithm maintains $MTDR = \{TDR_1, TDR_2, \cdots, TDR_m\}$ where each *TDR* is a distinct interest category concept. The interest in any document *D* given *MTDR*, denoted by

MTDR Algorithm ($\langle D, fb \rangle$: the relevance feedback document)

Let TDR_j be the j^{th} interest category model of $MTDR$,

M be the maximum number of *TDRs* maintained in *MTDR*, and

θ is the decision threshold constant (0,1).

Let $s = \text{sim}(D, TDR_i)$ such that

$\text{sim}(D, TDR_i) = \max_j \{\text{sim}(D, LTD_j),\ \text{sim}(D, PosD_j),\ \text{sim}(D, NegD_j)\}$

where LTD_j, $PosD_j$ and $NegD_j$ are the three descriptors of $TDR_j \in MTDR$.

If $(s < \theta)$

  If $(\|MTDR\| < M)$

    Create a new category TDR_k using $\langle D, fb \rangle$.

  Else

    Update the long-term and short-term interest models of TDR_i using $\langle D, fb \rangle$.

Else

  For $\forall m\ \text{sim}(D, TDR_m) \geq \theta$

    Update the long-term and short-term interest models of TDR_m using $\langle D, fb \rangle$.

Figure 3.1: MTDR algorithm.

$I_{MTDR}(D)$, is obtained from the maximum value of the interest in D for any *TDR* in *MTDR*. That is, $I_{MTDR}(D) = \max_i \{I_{TDR-i}(D)\}$.

Figure 3.1 describes the MTDR algorithm for learning and tracking the changes in multiple interest categories. A new interest category model will be created to store the category concept of a new document if the content of the document is

different enough from all existing models. A decision threshold θ is used to determine when the highest similarity to an existing interest category is low enough to justify creating a new interest category model.

The similarity of a document to an interest category model is defined as the maximum similarity between the document and either the long-term descriptor, the positive descriptor, or the negative descriptor of the model. When the similarity of a document to existing models exceeds a decision threshold θ for several interest category models, all these models are updated in order to maintain the consistency among similar target concepts (Widyantoro, Ioerger, & Yen, 2001). The parameter M is applied to limit the number of interest category models that can be generated in a multiple three-descriptor model.

3.4 Window-based Concept Drift Learning Algorithms

A window-based algorithm as shown in Figure 1.1 adapts to concept drift by sliding a window over recent examples and relearning a target concept from examples within the window. Variants of this method mainly differ from one another on the base learner employed for inducing the target concept and the method for adjusting the window size. Figure 3.2 provides the algorithm adapted for text document domain. It consists of two main components typically exist in an on-line learner. For each new relevance feedback document presented, the first component attempts to predict the document relevance by applying the *Prediction*() function and uses the relevance value that accompanies the document to check the accuracy of its prediction. It

Window-based Algorithm ($\langle D, fb \rangle$: the relevance feedback document)
Input:
D = Document.
fb = {1,0}, 1 for relevant document and 0 for irrelevant document.

Initialization:
$S = \varnothing$, a list of relevance feedback documents in order of arrival time.
C = *null*, a target concept.
$P = \varnothing$, a list of prediction results for performance monitoring.

On observing a feedback document D with relevance value fb:
Concatenate D at the end of S.
If ($C \neq null$) **then**
Let $p = 1$ if $Prediction(D) = fb$, or $p = 0$ otherwise.
Concatenate p at the end of P.
End-If

Target Concept Learning:
$n = GetWindowSize(P)$.
D_{LIST} = Get the most recent n documents from S.
$C = LearnTargetConcept(D_{LIST})$.

Figure 3.2: Window-based concept drift learning algorithm.

returns one (zero) for a correct (an incorrect) prediction. The prediction results are maintained in a list P for monitoring the system performance whenever needed.

The second component performs the actual learning. For simplicity, the new target concept is regenerated periodically after seeing k new documents by relearning n most recent feedback documents. n is the window size determined by the

GetWindowSize() function. If the algorithm employs a fixed window size, the *GetWindowSize*() function returns a pre-defined constant value. The window size *n* can also be determined by an adaptive window adjustment heuristic. The *LearnTargetConcept*() function induces the target concepts derived from the *n* recent documents using a selected concept learner.

3.4.1 Adaptive Window Adjustment Heuristic

Typical adaptive window adjustment heuristics (Widmer & Kubat, 1996; Klinkenberg & Renz, 1998) adjust the window sizes based on the changes in the system's predictive performance. The details of these heuristics are generally domain dependent. Figure 3.3 describes the heuristic implemented in the window-based concept drift algorithm used in this chapter, which has the same performance-based adaptation principle as those in the existing heuristics. The system's predictive performance is calculated from the outcomes of a fixed number of past predictions.

Let $Accuracy_{t-1}$ be the system's predictive accuracy measured when a concept $C_{t\text{-}1}$ is learned at time $(t-1)$ using a window of size $WindowSize_{t-1}$. After predicting the class of *k* new examples using the learned concept $C_{t\text{-}1}$, let $Accuracy_t$ be the new system's predictive accuracy that incorporates the prediction outcomes on the new examples. The heuristic will expand the window if it observes a performance increase (e.g., $Accurracy_t > Accuracy_{t-1}$) so that the new window size, e.g., $WindowSize_t$, will include both older examples for generating $C_{t\text{-}1}$ and the new *k*

Window-Adjustment-Heuristic Algorithm

Input:

$P = \{1|0\}^*$, a sequence of prediction results where 1 or 0 indicates a correct or an incorrect prediction.

Initialization:

$Accuracy_0 = 0$, previous predictive performance.

$\#PastPred = 10$, the number of past predictions for performance assessment.

$WindowSize = PastPred$.

Algorithm:

If $(\|P\| > \#PastPred)$ then

Let $Accurracy_t = \dfrac{\sum_{i=\|P\|-\#PastPred+1}^{\|P\|} P_i}{\#PastPred}$

If $(Accurracy_t > Accuracy_{t-1})$ /* predictive performance is increasing */

/* increase the window size to include unaccounted k new examples */

$WindowSize_t = WindowSize_{t-1} + k$

Else

If $(Accurracy_t < Accuracy_{t-1})$ /* predictive performance is decreasing */

/* reduce the window size proportionally to the current performance */

$WindowSize_t = \text{Max}\ \{2, Accuracy_t * WindowSize_{t-1}\}$

Else

/* predictive performance is stable */

If $(CrntAcc \geq 0.5)$ /* stable at a higher accuracy */

/* increase the window size by one */

$WindowSize_t = WindowSize_{t-1} + 1$

Else

/* reduce the window size when stable at a lower accuracy */

$WindowSize_t = \text{Max}\ \{2, Accuracy_t * WindowSize_{t-1}\}$

Figure 3.3: Window adjustment heuristic algorithm.

examples. The window size is proportionally reduced with respect to current performance if it either decreases from previous performance or is stable at a lower accuracy. If the predictive performance is stable at a higher accuracy, the window is slightly increased.

3.4.2 Base Learners

A base learner carries out the *LearnTargetConcept*() function in the window-based concept drift learning algorithm. For the evaluation of this algorithm, this dissertation considers two widely used learning methods as the base learner: Rocchio and *k* Nearest Neighbor (KNN). For reference convenience, **Window-Rocchio** algorithm will be used to denote the window-based learning algorithm that employs the Rocchio learner, and a version that uses the KNN base learner will be called **Window-KNN** algorithm.

In the Window-Rocchio algorithm, the positive and negative examples in D_{LIST}, which contains the n most recent relevance feedback documents, are equally weighed and the learning process in the Rocchio algorithm is performed using Equation 3.3. Let $D_{Rocchio}$ be the Rocchio descriptor, that is, the concept generated by the Rocchio algorithm. The prediction is performed by thresholding the similarity between a document D and the Rocchio descriptor $D_{Rocchio}$. Hence, the *Prediction*() function in Figure 3.2 is defined as follows:

$$Prediction_{Rocchio}(D) = \begin{cases} 1 \text{ if } sim(D, D_{Rocchio}) \geq \theta \\ 0 \text{ otherwise} \end{cases} \tag{3.9}$$

where θ is the decision threshold for the Rocchio classifier.

The Window-KNN method learns by simply storing all the given examples in D_{LIST}. Let D_{KNN} be the k documents in D_{LIST} that are most similar to a new document D. The class prediction of D, with respect to the stored documents, is based on the class of examples in D_{KNN} that maximizes their sum of similarities to D as follows:

$$Prediction_{KNN}(D) = \arg\max_{v \in \{0,1\}} \sum_{D_i \in D_{KNN}} \mathrm{Sim}(D, D_i) \cdot \delta(v, fb(D_i)) \qquad (3.10)$$

where $\delta(v, fb(D_i)) = 1$ if $v = fb(D_i)$ and where $\delta(v, fb(D_i)) = 0$ otherwise.

3.5 Experiment Setup

This section describes the setting of experiments for evaluating the four concept drift learning algorithms described earlier (e.g., MTDR, Rocchio, Window-Rocchio, and Window-KNN algorithms). Its primary purpose is to empirically validate the advantages and shortcomings of these algorithms on three aspects: (1) the ability for tracking multiple target concepts simultaneously, (2) the compliance with the persistence assumption about the change of target concepts, and (3) the effect of reducing the number of (labeled) examples. The following describes the data, experiment procedures and tracking problems needed to achieve this goal.

3.5.1 Document Collection

A subset of the Reuters-21578 1.0 test collection (Blake & Merz, 1998) was used in the experiments. The original collection contains 135 topics and 21,578 stories

obtained from the Reuters newswire in 1987. Of these stories, 12,902 had been assigned to one or more categories. The stories were divided into training and test documents according to *ModApte* split, which had 9,603 documents for the training set and 3,299 documents for the test set (Apté, Damerau, & Weiss, 1994).

The documents used in the experiments are selected among those in the *ModApte* split that have been assigned a single topic category. As a result, the test set contained 2581 documents consisting of 59 topics. The test documents were used to measure the model's accuracy. The rest of the training set, which contained 6452 documents, is used to generate a sequence of relevance feedback documents for modeling user interests incrementally. The documents are pre-processed by removing stop words, stemming the remaining words, identifying bigrams and extracting them as individual terms, and counting term frequencies. The document terms are then weighed according to the TF-IDF method (see Equation 3.1). These processes are common in the information retrieval literature (Witten, Moffat, & Bell, 1994).

3.5.2 Experiment Procedure

Following the standard in concept drift learning, the goal of experiments is to observe the system performance as target concepts (i.e., current user interests) change from time to time. Accordingly, the system is presented with a stream of feedback documents to learn sequentially, and its performances are measured on a fixed test set with respect to current target concepts at regular intervals after processing *m*

consecutive documents. A period of incremental learning on the *m*-document sequence and system performance measurement is then called a *tracking cycle*.

The test data was used to measure the model performance on each tracking cycle. The accuracy of a model measured at the end of a tracking cycle was calculated as follows. First, all documents in the test data were ranked using the learned model. The prediction accuracy of the model was then measured by calculating the percentage of target test documents ranked within the top *n* documents (where *n* is set to maximum number of documents in desired categories). Specifically, let *P* be the number of documents in positive topics that appear in the top *n* documents ranked by a model. The accuracy of the model at a tracking cycle *t* is calculated using the following equation:

$$Accuracy_t = \frac{P}{\sum_i TC_i} \times 100\% \qquad (3.11)$$

where TC_i are the numbers of documents in positive topic categories being considered in the current tracking cycle and $n = \sum_i TC_i$ is the total number of target test documents in the test data. This accuracy measure is essentially equivalent to the standard performance measure of recall-precision break-even point, a value at which precision is equal to recall in text categorization tasks (Lewis & Ringuette, 1994). The average accuracy value is calculated by averaging the system accuracy from the first tracking task to the end.

Tracking Cycle				
1 – 20	21 – 40	41 – 60	61 – 80	81 – 100
(*Trade*, +)	(*Trade*, −) (*Coffee*, +)	(Coffee, −) *(Crude, +)*	(*Crude*, −) (*Sugar*, +)	(*Sugar*, −) (*Acq*, +)
(*m*=1)	(*m*=2)	(*m*=2)	(*m*=2)	(*m*=2)

Table 3.1: Tracking task 1.

Tracking Cycle			
1 – 20	21 – 40	41 – 60	61 – 80
(*Trade*, +) (*Coffee*, +)	(*Trade*, −) (*Coffee*, +) (*Crude*, +)	(*Coffee*, −) (*Crude*, +) (*Sugar*, +)	(*Crude*, −) (*Sugar*, +) (*Acq*, +)
(*m*=2)	(*m*=3)	(*m*=3)	(*m*=3)

Table 3.2: Tracking task 2.

Tracking Cycle		
1 – 20	21 – 40	41 – 60
(*Trade*, +) (*Coffee*, +) (*Crude*, +)	(*Trade*, −) (*Coffee*, +) (*Crude*, +) (*Sugar*, +)	(*Coffee*, −) (Crude, +) (*Sugar*, +) (*Acq*, +)
(*m*=3)	(*m*=4)	(*m*=4)

Table 3.3: Tracking task 3.

3.5.3 Tracking Tasks

The data streams are generated according to a tracking task, a scenario that describes the evolution of topics of interest over time. The changes in topics of interest over time are simulated by alternating among interests in *Trade*, *Coffee*, *Crude*, *Sugar* and *Acq* topics. These five topics are called target topics (concepts) whose sizes in the test

Tracking Cycle	
1 – 20	21 – 40
(*Trade*, +) (*Coffee*, +)	(*Coffee*, –) (*Crude*, +)
(*m*=2)	(*m*=2)

Table 3.4: Tracking task 4.

Tracking Cycle	
1 – 20	21 – 40
(*Trade*, +) (*Coffee*, +) (*Crude*, +)	(*Coffee*, –) (Crude, +) (*Sugar*, +)
(*m*=3)	(*m*=3)

Table 3.5: Tracking task 5.

set are 75, 22, 121, 25 and 696, respectively. Tables 3.1 through 3.5 provide five tracking tasks used in the experiments. Each column in the tables describes the number and the topic of documents in the *m*-document sequence that is processed at each tracking cycle. In Table 3.1, for example, tracking cycles 21–40 process two-instance sequences; each contains one *Trade* document and one *Coffee* document ordered randomly in the sequence. Each tracking cycle uses a new set of documents from the training set that has not been seen. Information regarding the document topic category is not told to the system.

For simplicity, target concepts are made stable for periods of twenty tracking cycles. Feedback document set that marks the beginning of change in target concepts are given at the first tracking cycles during the twenty-tracking cycle periods, that is,

	Tracking Cycle				
	1 – 20	21 – 40	41 – 60	61 – 80	81 – 100
Tracking Task 1	*Trade*	*Coffee*	*Crude*	*Sugar*	*Acq*
Tracking Task 2	*Trade* *Coffee*	*Coffee* *Crude*	*Crude* *Sugar*	*Sugar* *Acq*	
Tracking Task 3	*Trade* *Coffee* *Crude*	*Coffee* *Crude* *Sugar*	*Crude* *Sugar* *Acq*		
Tracking Task 4	*Trade* *Coffee*	*Trade* *Crude*			
Tracking Task 5	*Trade* *Coffee* *Crude*	*Trade* *Crude* *Sugar*			

Table 3.6: Summary of target concept evolution over twenty-tracking cycle periods.

at tracking cycles 1, 21, 41 and so on. Topics with positive (+) labels indicate the desired target concepts at the respective tracking cycles. Documents with positive labels indicate relevant documents and are used to establish new (or emphasize the existing) target concepts. The negative labels are used to demote previously established target concepts. For example, a positive *Trade* document in Table 3.1 is given during the first tracking cycle to establish the new interest in *Trade* topic. The document set provided during the 21st tracking cycle contains one positive *Coffee* document and one negative *Trade* document, which changes the target concept from *Trade* to *Coffee*.

Amount of Labeled Data	Tracking Cycles
5 Percent	1, 21, 41, …
10 Percent	1, 11, 21, 31, 41, …

Table 3.7: Data streams with reduced number of examples.

Table 3.6 summarizes the evolution of target concepts implied by each tracking task over the twenty-tracking-cycle periods. The tracking tasks 1, 2 and 3 represent learning problems in order of increasing levels of difficulty, in terms of the number of target concepts that must be learned in each tracking cycle. These three tasks provide tracking problems satisfying the assumption in that only latest examples are relevant to current target concepts. The tracking tasks 4 and 5 involve long-live and short-live target concepts, which require *the persistence assumption* in order to properly track all the target concepts. Specifically, positive *Trade* documents are given during the first twenty tracking cycles but these tracking tasks never provide negative *Trade* documents afterwards implying that the *Trade* topic remains to be one of the target concepts for the rest of the tracking cycles. In these tracking tasks, *Trade* topic is the long-live target concept.

To observe the effect of labeled data reduction, the actual example sets are provided only at certain tracking cycles, and the same number of performance measurements is performed as the number of tracking cycles defined in the original tracking tasks. Therefore, the system performance at tracking cycles during which the examples sets are not given are expected to be the same. Table 3.7 provides the

details of tracking cycles at which the labeled example sets are made available for each data stream. For example, a data stream that contains only five percent as many examples as provided in the original tracking task can be obtained by providing the example sets on tracking cycles 1, 21, 41 and so on.

3.5.4 Parameter Settings

The algorithms described earlier introduce several parameters. The settings of parameter values employed in the MTDR algorithm are the same as those defined empirically in prior work (Widyantoro, Ioerger, & Yen, 2001). That is, the learning rate $\alpha = 0.3$ in short-term descriptor models, $\beta = 0.1$ in long-term descriptor models, $\eta = 0.5$ in interest category (three-descriptor) models, and $M = 8$ as well as $\theta = 0.175$ in MTDR models.

The *Prediction*() function in the Window-Rocchio algorithm, as described before, also relies on a pre-defined classification threshold. The experiments on tracking tasks 1-3 are conducted by varying the threshold values from 0.025 to 0.35 at 0.025 intervals. The performance achieved by this algorithm is selected from a threshold setting that produces the best outcomes. These thresholds are 0.15, 0.075 and 0.1 for tracking tasks 1, 2 and 3 respectively. The threshold for tracking task 4 is set to 0.075 simply because this task has the same number of target concepts as that of tracking task 2. Similarly, the threshold defined on tracking task 5 is 0.1.

Lastly, the performance of the Window-KNN algorithm is also selected from the k value in the KNN classifier that produces the best result in the respective

tracking task. The *k* values have been varied from 1 to 21 at intervals of two. The experiment results for tracking tasks 1 through 5 presented in the next section are a result from setting the *k* values to 9,9,7, 9 and 7, respectively.

3.6 Experiment Results

This section summarizes the results of numerous experiments that have been conducted using the four algorithms described earlier. The first two sections briefly review empirical results that explain the behaviors of the MTDR algorithm's components as well as the window-based learning algorithms. The capability of each algorithm on addressing the three aspects mentioned earlier will be discussed in the last three sections.

3.6.1 The Behavior of MTDR Algorithms

The interest category representation that underlies the MTDR algorithm is composed of long-term and short-term interest models. Figure 3.4 depicts empirical results that demonstrate the characteristics, strengths and weaknesses of these models. The figure also shows how their characteristics agree with the motivations behind the development of these models. The results presented in this figure used tracking task 1 as the learning problem, averaged over 10 runs. The more detail behaviors of these models on various parameter values were described in (Widyantoro, Ioerger, & Yen, 2001).

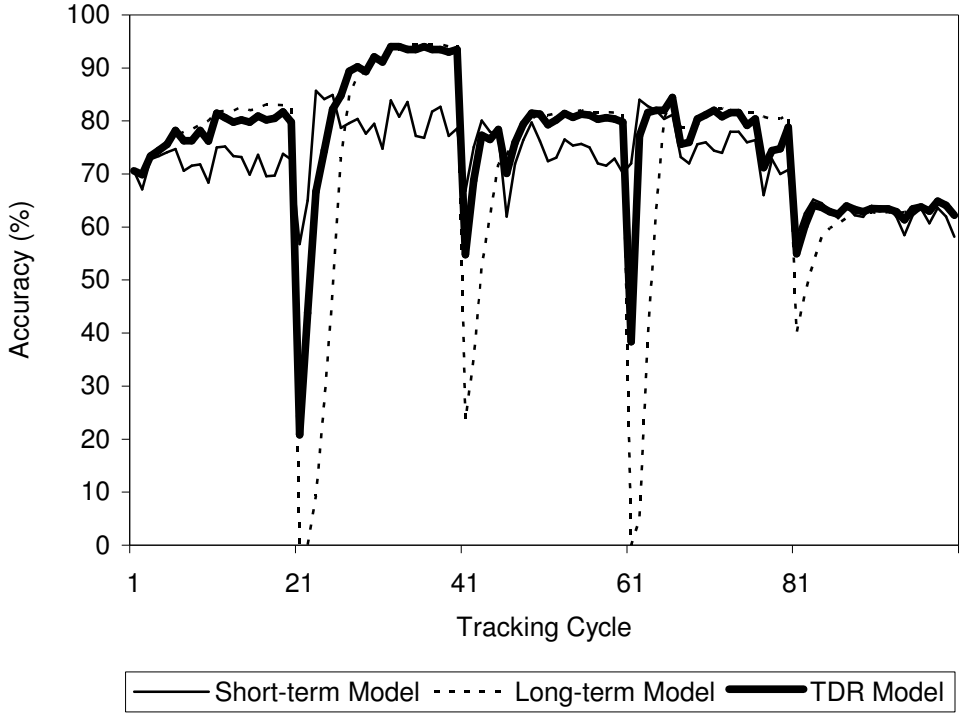


Figure 3.4: The characteristics of long-term, short-term and TDR models.

As shown in Figure 3.4, the long-term interest model improves its prediction accuracy consistently as it learns more relevance feedback documents. This property also represents the behavior of Rocchio algorithm because the long-term interest model is learned using this algorithm. The weight update rule of this model allows preserving common features of documents. Its weakness is that a long-term interest model by itself suffers from learning a dynamically changing interests at a slow, fixed rate. Because it learns and unlearns documents gradually, this model cannot remove

its old interests quickly enough when it has to learn a new interest. Therefore, the performance of this model usually drops drastically at each learning phase transition.

The short-term model at a higher learning rate tends to destabilize the model, causing the prediction accuracy of the short-term interest model to fluctuate erratically. However, the effect of changing interests on the prediction accuracy of the model is slight, if any. Lowering the learning rate could improve the stability and the average performance of the model (the figure of this is not shown) but could also cause the model to be more sensitive to changing interests, similar to the problem faced by the long-term interest model. Thus, there is a tradeoff between achieving higher performance and a more stable model, versus obtaining a more adaptive model for learning changing interests as a function of the learning rate. This tradeoff represents the strength and, at the same time, the weakness of the short-term interest model.

The three-descriptor model possesses a combination of the strengths and weaknesses of the long-term and short-term interest models. The expected strengths of this model are: achieving high prediction accuracy obtained from the long-term interest model, and being able to respond quickly on changing interests as in the short-term interest model. The weaknesses of both the long-term and short-term interest models hopefully can be reduced as much as possible.

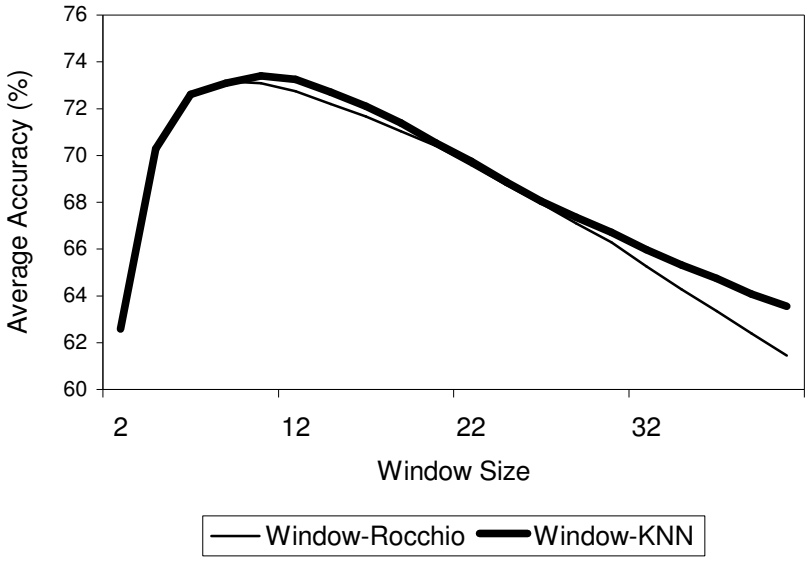


Figure 3.5: The effect of window size in the window-based learning algorithms with a fixed window size on the average accuracies.

3.6.2 The Behavior of Window-based Algorithms

Determining the appropriate window size is the main difficulty in the concept drift learning algorithm that relies on a fixed window of recent examples. Figure 3.5 depicts the average accuracies obtained from this algorithm as a function of window sizes. The figure confirms the expectation that both too small and too large of window sizes generate non-optimal performances. A smaller window size would retrieve examples with less or even no noise but the small number of examples retrieved is not enough to generalize the target concept. Conversely, a larger size of window draws more examples but the retrieved data would contain many more

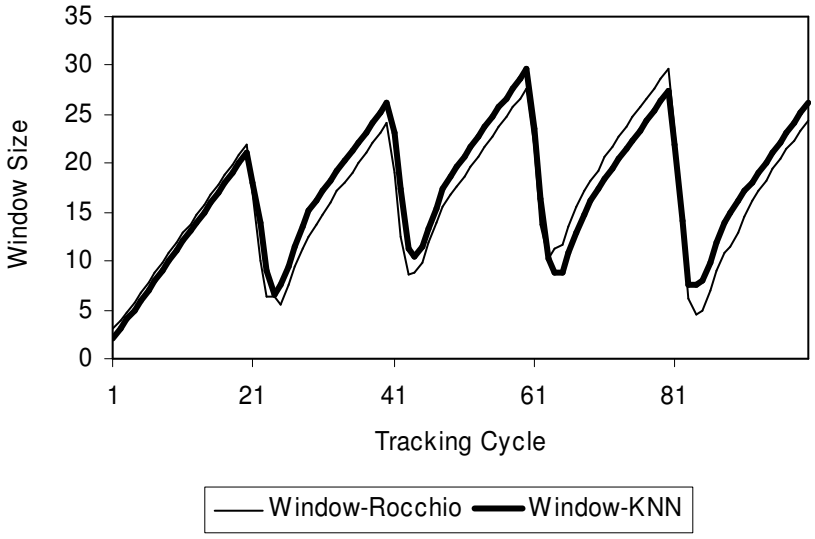
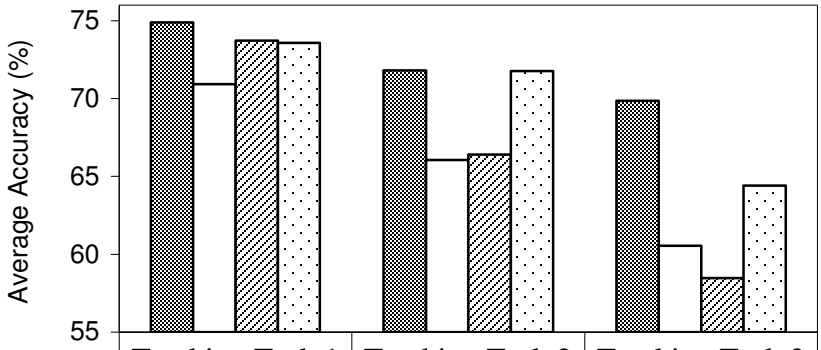


Figure 3.6: The adaptation of window size over time in window-based learning algorithms.

conflicting examples, hindering the learner to build an accurate target concept representation.

The window adjustment heuristic overcomes the window size determination problem by adaptively changing the window size in an attempt to include more examples or avoid incorporating noise. Figure 3.6 depicts the evolution of window sizes over time as a result from applying the window adjustment heuristic described by Figure 3.3. The window sizes expand as expected to include more relevant examples during stable periods, and shrink quickly during the transition of target concept changes (e.g., at tracking cycles 20, 40 and so on) that introduce potentially many conflicting examples. This indicates at least that the mechanism for adapting to concept drift works properly.



	Tracking Task 1 (Single Concept)	Tracking Task 2 (Two Concepts)	Tracking Task 3 (Three Concepts)
MTDR	74.90	71.80	69.85
Rocchio	70.92	66.06	60.54
Window-Rocchio	73.71	66.40	58.48
Window-KNN	73.58	71.77	64.41

Table 3.8: Performance comparison on tracking tasks 1, 2 and 3.

3.6.3 Performance Comparison

Now, the MTDR algorithm's components and the window adjustment heuristic of the window-based algorithms have been shown to behave as expected. This section will discuss the performance comparisons of these two classes of algorithms on three aspects: the ability for tracking multiple target concepts, the conformance to the persistence assumption, and the ability for handling few examples.

Tracking multiple target concepts. Recall that the numbers of target concepts to be tracked in tracking tasks 1, 2 and 3 are one, two and three, respectively. Table 3.8 depicts the average accuracies of the four algorithms on the three tracking tasks. In the task involving only a single target concept (e.g., tracking

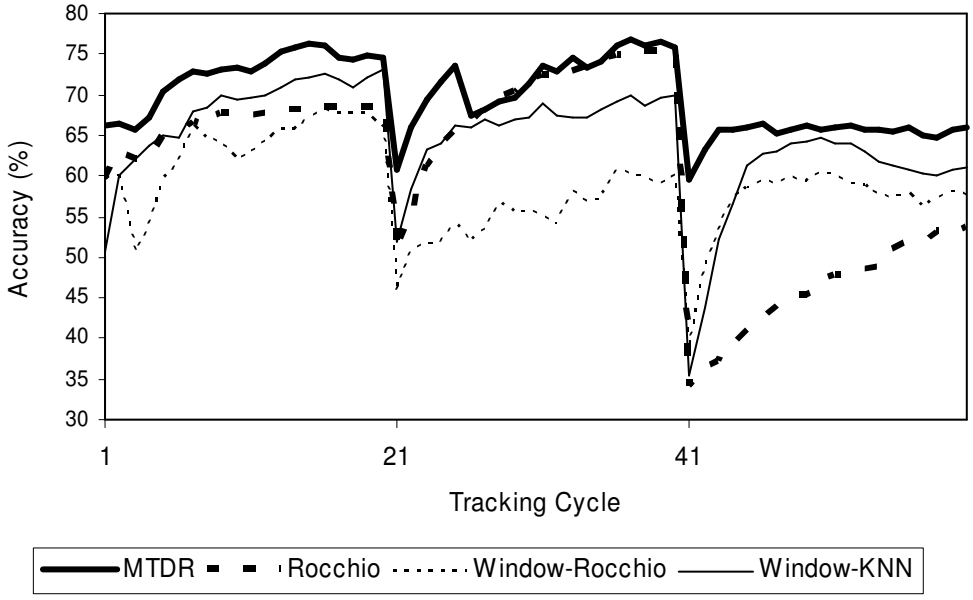
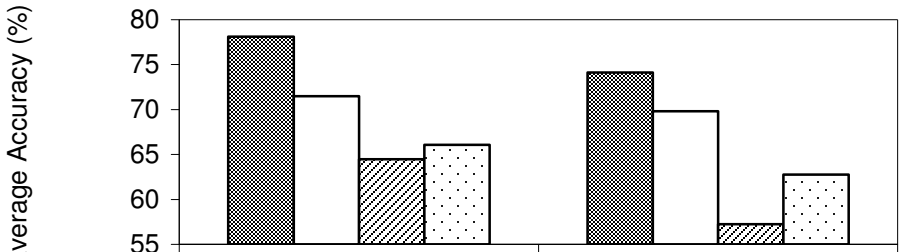


Figure 3.7: Performance over time on tracking task 3.

task 1), all algorithms perform comparably well except the Rocchio algorithm. The average accuracies of Window-Rocchio algorithm are worse than the others in tasks involving larger number of target concepts. Lastly, the MTDR algorithm outperforms the Window-KNN algorithm when tracking three target concepts simultaneously although their performances are still comparable in tracking task 2.

Figure 3.7 shows the performances over time of all algorithms on tracking task 3. The MTDR algorithm consistently performs relatively very well throughout the tracking cycles on this task. The observation that the performances of Rocchio and Window-Rocchio algorithms are relatively much worse are not surprising because the Rocchio algorithm is biased toward learning a single target concept and



	Tracking Task 4	Tracking Task 5
MTDR	78.12	74.13
Rocchio	71.50	69.81
Window-Rocchio	64.47	57.25
Window-KNN	66.06	62.78

Table 3.9: Performance comparison on tracking tasks 4 and 5.

thus lacks the representational power needed for learning multiple target concepts. At the opposite end, KNN algorithm is biased toward generating target concepts as many as example it acquires. This algorithm is naturally capable of learning multiple target concepts. Its classification accuracy in the Window-KNN algorithm, which is based on the *k* nearest examples, appears to be better than that of the Rocchio (Window-Rocchio) algorithm when involving multiple target concepts but is still not as good as the classification accuracy achieved by the MTDR algorithm. This observation provides a piece of empirical evidence that the MTDR algorithm, which has been designed to recognize and to track multiple target concepts, is better than the window-based algorithms applying the window adjustment heuristic.

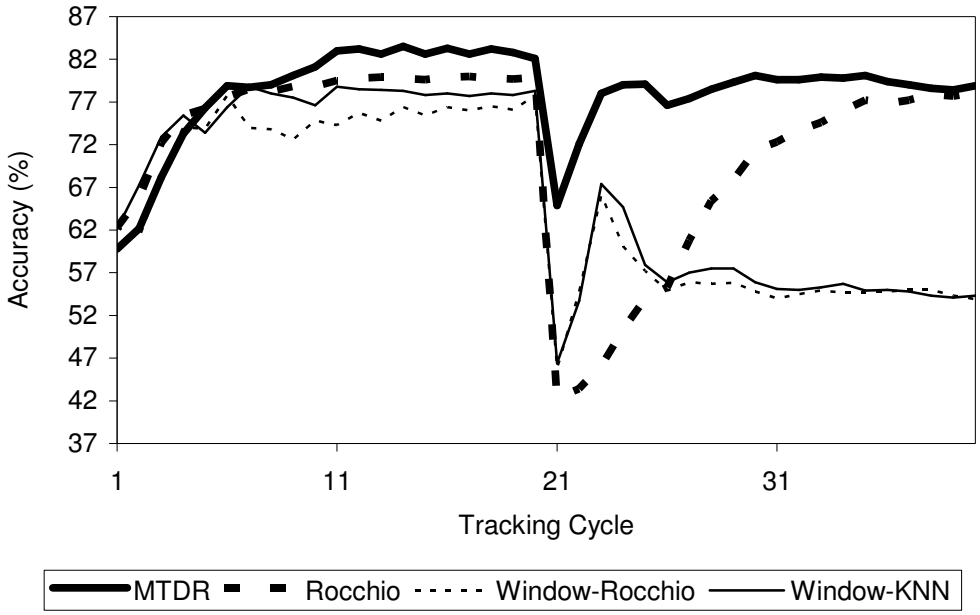


Figure 3.8. Performance over time on tracking task 4.

Conformance to the Persistence Assumptions. Technically, the recency example assumption that underlies the window-based learning algorithm will not properly work on tracking tasks 4 and 5 because these tasks require the conformance of the persistence assumption in order to track the long-live concept (e.g., *Trade* topic) defined in the tasks. Table 3.9 provides the average accuracies of all algorithms on these tracking tasks. While the performances of MTDR algorithm remain high, the performances of Window-Rocchio and Window-KNN are severely degraded.

Figures 3.8 and 3.9 depict the performances over time of the four algorithms on tracking tasks 4 and 5, respectively. As described earlier, the *Trade* target concept, which requires the persistence assumption to track on this task, was given only during

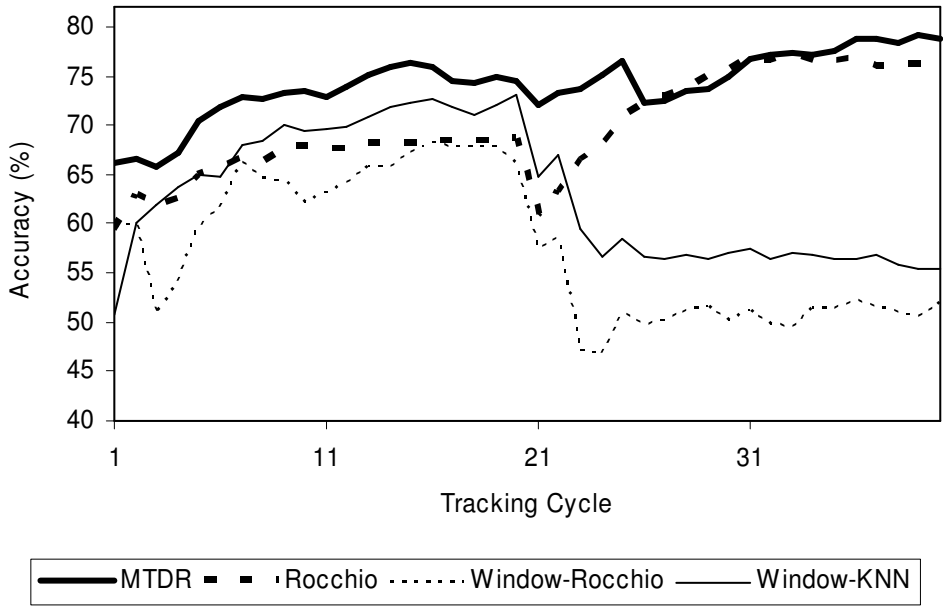


Figure 3.9. Performance over time on tracking task 5.

the first twenty tracking cycles. The change of non-*Trade* target concept that occurs during the 21st tracking cycles triggers the window-based algorithms to shrink their windows. Figure 3.8 clearly shows that the window-based algorithms start regaining their performances as they see more examples representing the new target concepts during the first few tracking cycles after the target concept change but it happens only shortly. As the windows move forward, they also quickly remove all examples needed for learning the *Trade* target concept, causing sudden drops in performance for failures in learning the old (long-live) target concept.

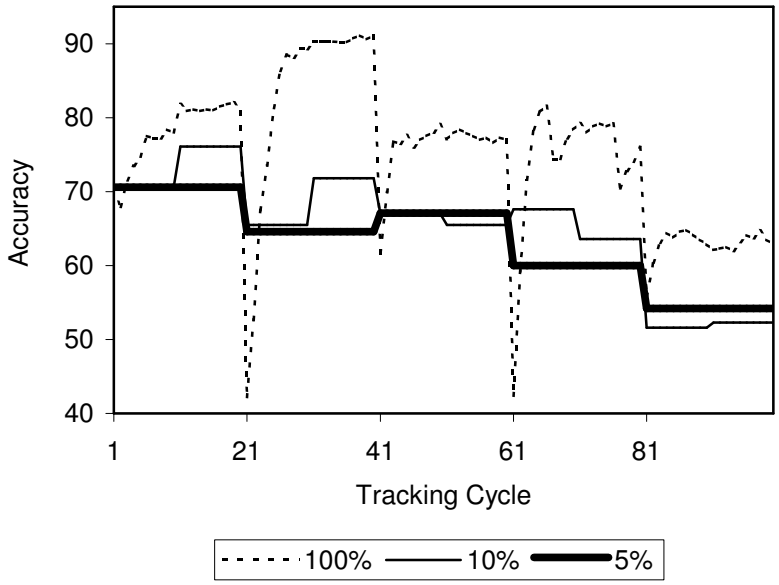


Figure 3.10: Performance over time with reduced number of examples on tracking task 1 and on MTDR algorithm.

The effects of labeled data reductions. Figure 3.10 illustrates a typical performance over time produced by learning with significantly reduced numbers of labeled examples. Starting from the first tracking cycles, the next example set in the 5% data stream is given at tracking cycles 21, 41 and so on (see Table 3.7 for a full description on this). The jagged lines on the curves with reduced number of labeled examples are due to the use of the same test set so that the performance will not change until the next labeled data are made available. In 5% and 10% cases, specifically, the performances are not expected to change until the next twenty and ten tracking cycles, respectively.

	Average Accuracy (%)			
	MTDR	Rocchio	Window-Rocchio	Window-KNN
Tracking Task 1				
100 Percent	74.90	70.92	73.71	73.58
10 Percent	65.16	53.26	66.19	59.19
5 Percent	63.29	46.35	60.26	57.09
Tracking Task 2				
100 Percent	71.80	66.06	66.40	71.77
10 Percent	62.77	53.27	54.52	49.69
5 Percent	60.12	49.25	39.49	46.80
Tracking Task 3				
100 Percent	69.85	60.54	58.48	64.41
10 Percent	63.73	55.53	48.26	51.18
5 Percent	59.53	53.00	42.93	46.66
Tracking Task 4				
100 Percent	78.04	71.57	64.35	66.22
10 Percent	68.94	59.05	57.20	59.92
5 Percent	65.45	52.35	57.20	56.29
Tracking Task 5				
100 Percent	74.13	69.81	57.25	62.78
10 Percent	68.22	59.72	45.79	51.24
5 Percent	64.69	56.90	41.50	39.55

Table 3.10: Summary of experiments with reduced number of examples on all tracking tasks and all algorithms.

Table 3.10 summarizes the average accuracies of all algorithms on all tracking tasks. The "100 Percent" rows are rewritten from Tables 3.8 and 3.9 for convenience in interpreting the results. Clearly, all algorithms suffer from being unable to maintain the high average accuracies at the reduced size of data streams. Although the window adjustment heuristic works pretty well with a sufficiently large number of examples (e.g., 100 Percent), its performance predictor in the window-based algorithm seems to be no longer accurate for properly adjusting the window size, resulting in even worse performance degradation than those of the MTDR algorithm.

3.7 Summary

This chapter has described four concept drift learning algorithms for tracking the evolution of user interests in the information filtering domain. The first one is the Rocchio relevance feedback algorithm. Originally developed as a batch process for improving the retrieval effectiveness in a static setting, this algorithm in this chapter is adapted through parameter tuning to work on a dynamic and incremental setting. This algorithm is selected mainly because it has been widely used and studied in the information retrieval community.

The second algorithm so-called MTDR represents a novel algorithm for learning the dynamics of tracking multiple interest categories. The algorithm adopts the persistence assumption regarding the user interests; that is, the user interests remain relevant until explicitly declared otherwise, and vice versa. Its main feature is combining the notions of long-term and short-term interest models in order to obtain

the strength of both models. The long-term and short-term interest models in the window-based concept drift learning algorithms correspond to models learned from large and small-size windows, respectively. These windows in the MTDR algorithm are implicitly modeled.

The last two algorithms are Window-Rocchio and Window-KNN; both are window-based algorithms that employ the Rocchio and KNN algorithms, respectively, as the base learners. Since a base learner is essentially a batch process in the main algorithm, the Rocchio algorithm in Window-Rocchio is applied as originally intended as a batch learner. The two algorithms employ an adaptive window adjustment heuristics for adapting to concept drift. The heuristics are derived from a general method based on the change in predictive performance. These two algorithms represent existing, commonly used algorithms for learning concept drift in the machine learning community.

This chapter provides empirical evidences that confirm the expected behaviors of the above four algorithms. The Rocchio algorithm adapted for learning concept drift (also the long-term interest model in the MTDR algorithm) is able to consistently improve its performance as it learns more examples, but is very susceptible to a change in target concept. The short-term interest model is relatively unstable but insensitive to concept drift. This chapter empirically shows the difficulty in determining the appropriate window size in a window-based learning algorithm; Small window results in an insufficient target concept generalization while large window precludes generating an accurate target concept. It also demonstrates that the

adaptive window adjustment heuristics employed in the Window-Rocchio and Window-KNN can alleviate the problem. In particular, the heuristics allow the window to expand during the period of stable concept learning and to quickly shrink when a concept drift does occur.

The superiority of the MTDR algorithm for tracking multiple target concepts has also been shown. Its main competitor is the Window-KNN whose performance is significantly worse than that of the MTDR algorithm only in tracking task 3 (tracking three target concepts). Furthermore, the performances of both window-based algorithms are significantly degraded when the persistence assumption is needed (i.e., when tracking long-live target concepts) in order to properly track the target concepts. The recency assumption that underlies the window-based algorithms represents the weakness addressed by the MTDR algorithm. Finally, this chapter empirically shows that all of these algorithms suffer from learning with significantly reduced number of examples.

CHAPTER IV

A COMPUTATIONAL FRAMEWORK FOR EXTENDING INCOMPLETE LABELED DATA STREAM IN CONCEPT DRIFT

Chapter III has shown that the performances of four concept drift learning algorithms consistently degrade when they learn from reduced numbers of labeled examples. This chapter presents FEILDS: a new computational **F**ramework for **E**xtending **I**ncomplete **L**abeled **D**ata **S**tream in concept drift learning, which extends the algorithms to deal with the issue. One of the system's inputs is the original labeled data stream that would normally be the input to the concept drift learners. FEILDS produces a new data stream that is fed to the (concept drift) learners. Hence, the system extends existing concept drift learning algorithms by modifying their inputs without modifying the algorithms.

The following section briefly reviews some practical and theoretical observations surrounding the problem that sheds light on a way to a solution. Section 4.2 provides the overview of the proposed solution. Section 4.3 describes the details of the system's components and methods. The advantages and shortcomings of the proposed method are then discussed in Section 4.4, followed by the chapter's summary in the last section.

4.1 Theoretical and Practical Observations

Poor performance as a result of learning from few examples is not only a problem in concept drift learning but also an issue in a less difficult, stable concept learning scenario. The requirement on the quantity of labeled data for learning stable concepts and adapting to concept drift is unfortunately inevitable without additional knowledge. As widely shown in Computational Learning Theory literature (Blummer *et al*., 1989; Mitchell 1997), reducing the sample size in stable concept learning would undercut the ability to approximate target concepts, which in turn would increase the classification error.

The problem is exacerbated in concept drift learning because the task also involves adaptation to possible concept drift, which is generally exploited from the given examples. More specifically, a few examples cannot provide reliable predictive performance needed by the heuristics of the window based algorithms for adapting to concept drift. A few negative examples in the MTDR and Rocchio algorithms are also insufficient for demoting old target concepts.

The empirical observations shown in Chapter III and the above practical observations are also well justified by existing theoretical findings. The *drift rate* in concept drift learning, as briefly explained in Chapter I, is an essential parameter, which denotes the probability that two successive target concepts c_i and c_{i+1} disagree on a randomly drawn example (Helmbold & Long, 1994), e.g., Pr ($c_i \neq c_{i+1}$). Hence, a slower drift rate corresponds to learning from a data stream whose target concepts

change less frequently, that is, having a longer sequence of data with the same target concept, and vice versa. Helmbold and Long (1994) provide theoretical bounds on the allowable drift rates that guarantee tractability with an error of at most ε as follows:

$$\Delta \leq \frac{c\varepsilon^2}{d \ln(1/\varepsilon)} \tag{4.1}$$

where $c > 0$ is positive constant, and d is the *Vapnik-Chervonenkis* dimension of a concept/hypothesis (Blummer *et al.*, 1989). Because c and d values are fixed, the bounds imply that the tracking problem is more difficult (i.e., producing higher error rates) on learning with fewer labeled data per target concept (i.e., higher drift rates). Hence, reducing the rate of drift according to the above equation is apparently the only option for improving the performance of a concept drift learner. FEILDS takes this general approach.

4.2 Overview of Approaches

Inspired by the success of techniques that combine labeled and unlabeled data in stable concept learning (Dempster, Laird, & Rubin, 1977; Blum & Mitchell, 1998; Blum & Chawla, 2001), a similar technique is developed for learning concept drift. FEILDS uses a set of relevant unlabeled data to compensate for the lack of labeled data, but for learning dynamically changing concepts. From the perspective of Computational Learning Theory, this general approach is guaranteed to improve performance. Provided that the relevant unlabeled data exist and can be correctly identified, incorporating these unlabeled data is equivalent to reducing the rate of

concept drift, which would increase the tracking accuracy. The setting of the input data is also well supported in the information filtering domain. Although labeled data in this domain are very expensive, the availability of unlabeled data is virtually unlimited and can be relatively easier to collect.

Without losing generality, the rest of this chapter assumes that the label value of a *labeled* instance is either 1 or 0. In the information filtering domain, this value corresponds to either a positive or a negative feedback document, respectively. In addition, an instance can be associated with a *concept category*. For example, *document topic* is the concept category of a text document. However, the information about the concept category of an instance is never told to the system.

The input of the system, as typical in concept drift learning setting, is a stream of instances. Unlike ordinary concept drift learning in which the labels of all instances in the stream are provided, only a very few of the instances' labels in the problem setting being addressed are made available to the learners. Furthermore, the majority of unlabeled data under a more realistic condition are irrelevant. Let $\mathbf{S} = \{x_1, \ldots, x_n\}$ be a set of instances taken from the stream (see Figure 4.1). The stream contains labeled data $\mathbf{L} = \{x_i \mid x_i \in \mathbf{S}\}$ and unlabeled data $\mathbf{U} = \{x_j \mid x_j \in \mathbf{S}\}$ such that $\mathbf{S} = \mathbf{L} \cup \mathbf{U}$ and $\mathbf{L} \cap \mathbf{U} = \varnothing$. Changing label values because of the change in target concepts is the main characteristic in the concept drift learning problem. When tracking the evolution of user interests, for example, a feedback document previously deemed relevant will become irrelevant when a later feedback document of the same

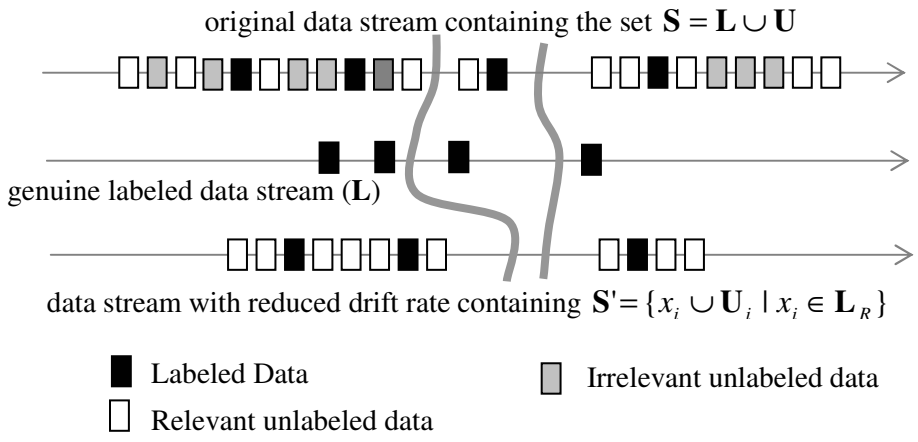


Figure 4.1: The illustration of approach for reducing the drift rate in a sparsely labeled data stream.

topic (i.e., the same underlying concept category) is judged irrelevant. The crux of the FEILDS's approach is to identify the set of labeled data $\mathbf{L}_R \subseteq \mathbf{L}$ whose label values have not changed. For each $x_i \in \mathrm{L}_R$, let $\mathbf{U}_i \subseteq \mathbf{U}$ be the corresponding subset of unlabeled data with the same underlying concept category as that of x_i. It then uses the set $\mathbf{S}' = \{x_i \cup \mathbf{U}_i \mid x_i \in \mathbf{L}_R\}$ to generate a new stream and assigns the label of each unlabeled instance $x_j \in \mathbf{U}_i$ with x_i's label.

Identifying the set $\mathbf{L}_R$ requires knowledge about the concept category of each instance in **L.** As in the above example, the change of feedback document relevance can only be accurately detected by knowing the topics of feedback document. However, the concept category of a given instance is unknown, and cannot be induced reliably from only a small set of labeled data. For example, identifying a set

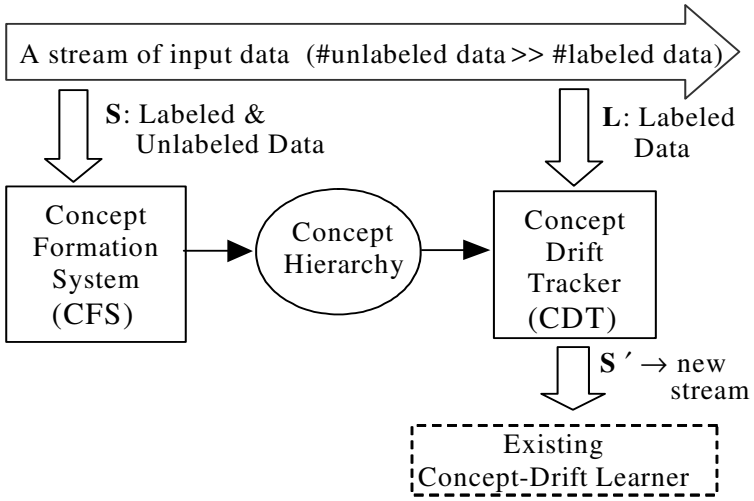


Figure 4.2: FEILDS architecture.

of terms that are representative to a topic category from a few document examples is difficult because a document typically contains many irrelevant terms. In order to provide a means for associating an instance with its concept category, FEILDS employs a concept hierarchy that is automatically constructed by clustering all incoming labeled and unlabeled data from the data stream. The next section describes the idea in greater details.

4.3 FEILDS Architecture

Figure 4.2 depicts the architecture of FEILDS that extends an existing concept drift learner to deal with incomplete labeled data stream (Widyantoro, Ioerger, & Yen, 2003). It consists of three main entities: (1) a concept formation system, (2) a concept hierarchy, and (3) a concept drift tracker. As shown in the figure, the concept

Input: a stream of documents ***Stream-*$_{\mathbf{S}}$**.

Initialization:

***Stream-*$_{\mathbf{L}}$** = ⟨∅⟩, the sequence of labeled instances.

$H = \emptyset$, the concept hierarchy.

Incremental Learning:

For each instance x observed from the stream ***Stream-*$_{\mathbf{S}}$**

Apply the CFS system to incorporate x into H incrementally.

If the label q of instance x is available

Concatenate $\langle (x, q) \rangle$ at the end of ***Stream-*$_{\mathbf{L}}$**.

Target Concept Induction (only when needed):

Apply the CDT component to identify a new expanded set $\mathbf{S}'$ based on the current values of ***Stream-*$_{\mathbf{L}}$** and H, and then generate a new stream ***Stream-*$_{\mathbf{S}}'$** arranged by the arrival time of data in $\mathbf{S}'$.

Apply a selected (conventional) concept drift learner to relearn ***Stream-*$_{\mathbf{S}}'$**.

Figure 4.3: The summary of FEILDS's approach.

hierarchy is at the heart of the FEILDS architecture. The concept formation system (CFS) incrementally constructs the concept hierarchy by organizing the input stream into a cluster hierarchy along with their corresponding concepts. The concept drift tracker (CDT) component is invoked only when needed by the concept drift learner. It takes as input a hierarchy of concepts and a sequence of labeled examples **L**, and infers the relevance of each concept category associated with a labeled example in **L**. This component outputs a new set $\mathbf{S}' = \{x_i \cup \mathbf{U}_i \mid x_i \in \mathbf{L}_\mathbf{R}\}$ that, as described above, contains the expanded relevant data $\mathbf{L}_R \subseteq \mathbf{L}$. Concept drift learning algorithms such

as those described in Chapter III can then be used to relearn ***Stream-*$_{\mathbf{S}}$′,** the new stream generated by rearranging all instances in $\mathbf{S}'$ according to the instance arrival times. Figure 4.3 summarizes the interactions among these components. ***Stream-*$_{\mathbf{S}}$** and ***Stream-*$_{\mathbf{L}}$** in the figure are the streams generated from the **S** and **L** sets, respectively.

4.3.1 Concept Formation System

The role of the concept formation system (CFS) component is to build a concept hierarchy incrementally from the input stream in an unsupervised mode. The construction of the concept hierarchy is essentially the same as building a hierarchy of clusters. During the course of learning, the concept hierarchy grows dynamically as the system receives more observations from the stream.

Because the concept hierarchy plays a central role in FEILDS, its quality is of great importance for success. Additionally, the requirement that it should also be constructed incrementally presents another challenge that would not be encountered had it been built in batch mode, which is not practical in the problem setting. FEILDS employs a new concept formation system that has been developed in this dissertation to address this challenge. Chapter V is devoted to describe and evaluate the concept formation algorithm.

4.3.2 Concept Hierarchy

The concept hierarchy is basically a tree structure with the following characteristics: (1) all leaf nodes represent document instances and thus are the most specific concept

nodes with respect to their ancestors, and (2) all internal nodes represent concepts that generalize their descendant concept nodes. Hence, the concept generality is increasing on any path from a leaf node to the root.

The concept hierarchy serves for the identification of (1) the concept category of an instance, (2) the set of instances belonging to a concept category and (3) the *least common subsumer* (*lcs*) concept. These processes are needed by the concept drift tracker (CDT) component. Let X be the instance space (e.g., leaf nodes) and C be the concept space (e.g., all nodes in the hierarchy). The following defines three general functions needed by the CDT for utilizing the concept hierarchy:

- $\delta : X \rightarrow C$ is an *instance generalization function* and is used for recognizing the concept category of an instance. For an instance x that is a leaf node, let $\mathbf{A}_X = x \cup \{c_1, \ldots, c_n\}$ be the set of x and x's ancestors where c_n is x's parent, c_{i-1} is c_i's parent and c_1 is the root node. Given x, the δ function returns a concept node $c \in \mathbf{A}_X$ that represents the concept category of x.
- $\varepsilon : C \rightarrow X^*$ is a *concept instantiation function*, which returns all leaf nodes that are descendants of a concept node $c \in C$. Since the node c represents a concept category, the ε function identifies all instances covered by the concept category.
- $\varphi : C^* \rightarrow C$ is a function that returns the *least common subsumer* (*lcs*) node of a given set of nodes. A node c_n subsumes a node c_m, denoted *subsume*(c_n,c_m), if c_m is a descendant of c_n or $c_m = c_n$ in the concept hierarchy.

The functions ε and φ above are straight forward given a concept hierarchy. The instance generalization function δ still requires a method that can accurately select the appropriate generalization of an instance from a sequence of concept nodes with increasing concept generality. This is a non trivial task even when provided with a perfect concept hierarchy. Because the concept hierarchy is automatically built in an unsupervised mode, no reliable information is available in order to determine a node that can best represent the concept category of an instance.

Best concept category representation implies that a node selected is neither *too general* (close to the root) nor *too specific* (close to the instance). Over generalization could mistakenly include unintended nodes of other concept categories, thereby adding noise to the concept category members. By contrast, too specific a node would lead to the problem of overfitting the instance that contributes little to providing new information. More detailed impact resulting from these two problems and how to address them will be discussed in the next section.

A node that distinctively partitions instances appears to be the one that appropriately generalizes an instance. The difficulty in identifying such a distinct node arises from the fact that the concept hierarchy alone does not have enough information for recognizing distinct node from a sequence of concept nodes with increasing generality (e.g., concept nodes in $\mathbf{A}_X$). The issue is addressed by using a validation set in order to characterize a distinct node. Section 4.3.4 elaborates this approach for implementing the instance generalization function above.

4.3.3 Concept Drift Tracker

The main CDT task is to infer a subset of labeled instances that are still truly relevant and then expand the subset with relevant unlabeled data. Given ***Stream-*$_{\mathbf{L}}$**, a stream of labeled data, the following six steps provide the detail processes performed by the CDT for generating $\mathbf{S}'$, the set of expanded L_R.

Step 1: Instance Sequence Generalization

Instance generalization is a process of identifying a concept category that can be associated with an instance, using the δ function described in Section 4.3.2. In this process, a sequence of labeled instances is transformed into a sequence of labeled concept nodes while preserving their ordering with respect to the ordering of the labeled instances. Let Q be the set of labels, and let ***Stream-*$_{\mathbf{L}}$** = $\langle(x_1,q_1),\ldots,(x_n,q_n)\rangle$ for each $x_i \in X$ and $q_i \in Q$ be a sequence of n labeled instances where an instance on the left side arrives earlier. Given a sequence of n labeled instances ***Stream-*$_{\mathbf{L}}$**, the instance sequence generalization process will output a sequence of n concept nodes ***Stream-*$_{\mathbf{C}}$** = $\langle (c_1, q_1),...,(c_n, q_n) \rangle$ such that $c_i = \delta(x_i)$ for each $c_i \in C$.

Step 2: Concept Node Sequence Partitioning

In this step, the problem of tracking multiple target concept categories is converted into multiple sub-problems of tracking a single concept. More specifically, this step partitions the concept node sequence according to a shared concept category, and rearranges concept nodes in each partition into a *concept node sequence partition* by maintaining their relative ordering in the original concept node sequence. Concept

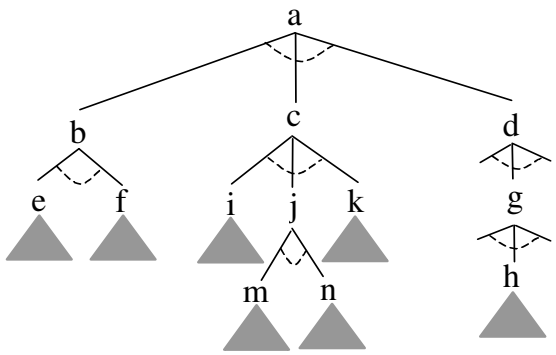


Figure 4.4: An illustration of concept hierarchy.

nodes in a set $\mathbf{C} = \{c_1,\ldots,c_n\}$ share the same concept category if there exists a *least common subsumer* (*lcs*) concept node in **C** that subsumes all other concept nodes, i.e., $c_i = \varphi(\mathbf{C})$ and $c_i \in \mathbf{C}$.

Example 1. This example refers to the concept hierarchy given by Figure 4.4. Let $Q=\{1,0\}$ be the set of labels. Let $\boldsymbol{Stream}\text{-}_{\mathbf{C}} = \langle(b,1),(h,0),(e,1),(g,0),(i,1),(d,0),(c,1),(d,1),(m,0),(f,0),(k,0),(b,0)\rangle$ be the sequence of concept nodes generated by the instance generalization process during the first step. According to Figure 4.4 and $\boldsymbol{Stream}\text{-}_{\mathbf{C}}$ above, $\mathbf{P}_1=\{b,e,f\}$, $\mathbf{P}_2=\{i,c,m,k\}$ and $\mathbf{P}_3=\{h,g,d\}$ are the concept node partitions since *b*, *c* and *d* are the *lcs* concept nodes for all concept nodes in partitions $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$, respectively. Three *concept node sequence partitions* are generated from the second step: $\boldsymbol{Stream}\text{-}_{\mathbf{CP1}} = \langle(b,1),(e,1),(f,0),(b,0)\rangle$, $\boldsymbol{Stream}\text{-}_{\mathbf{CP2}} = \langle(i,1),(c,1),(m,0),(k,0)\rangle$ and $\boldsymbol{Stream}\text{-}_{\mathbf{CP3}} = \langle(h,0),(g,0),(d,0),(d,1)\rangle$.

Step 3: Concept Node Sequence Contraction

The shared concept category in a *concept node sequence partition* (e.g., ***Stream-*$_{\mathbf{CP}}$**) is essentially the same so that two or more consecutive concept nodes in a ***Stream-*$_{\mathbf{CP}}$** with the same labels constitute a *redundant* fragment. Let a fragment be a sequence $\langle(c_m, q_m), (c_{m+1}, q_{m+1}),\ldots,(c_{m+n}, q_{m+n})\rangle$ satisfying $q_m=q_{m+1}=\ldots=q_{m+n}$ and $c_k=\varphi(\{c_m,c_{m+1},\ldots,c_{m+n}\})$ for some $c_k\in\{c_m,\ldots,c_{m+n}\}$. The current step eliminates this redundancy by iteratively searching for such a fragment and replacing it with it's *lcs* concept node until no further fragment is found. The final result is a *normalized concept node sequence partition*, or ***Stream-*$_{\mathbf{nCP}}$** for short. This stream describes the evolution of labels of a concept category and possibly its subcategory.

Example 2. ***Stream-*$_{\mathbf{CP1}}$** in Example 1 contains two fragments $\langle(b,1),(e,1)\rangle$ and $\langle(f,0),(b,0)\rangle$. The normalized ***Stream-*$_{\mathbf{CP1}}$** is ***Stream-*$_{\mathbf{nCP1}}$**= $\langle(b,1),(b,0)\rangle$ since the fragments' *lcs* is *b*. Similarly, ***Stream-*$_{\mathbf{nCP2}}$** = $\langle(c,1),(m,0),(k,0)\rangle$ and ***Stream-*$_{\mathbf{nCP3}}$** = $\langle(d,0),(d,1)\rangle$.

Step 4: Concept Node Label Identification

This step infers the label value of each concept node in the *normalized concept-node sequence partition* (e.g., ***Stream-*$_{\mathbf{nCP}}$**). The basis for inferring the label value is that of the *persistence assumption* in temporal reasoning, which states that once a fact is declared to be true, it remains true thenceforth until the fact is negated (Gabbay, Hogger, & Robinson, 1995). Consequently, the label of a sequence of identical concept nodes $\langle(c, q_m), \ldots, (c, q_{m+n})\rangle$ from a ***Stream-*$_{\mathbf{nCP}}$** can be represented by the

label assigned to the last concept node, i.e., q_{m+n}. A system in information filtering domain, as described earlier, typically uses two-value label: 1 and 0 for denoting relevant and irrelevant concept, respectively; and this system is interested only in relevant concepts. In such a system, a sequence of two identical concept nodes with conflicting labels whose label of the most recent concept node is 0 (i.e., $\langle(c,1), (c,0)\rangle$) can be dropped.

Example 3. Using ***Stream-*$_{\mathbf{nCP}}$** given by Example 2, the labels of *b* and *d* from ***Stream-*$_{\mathbf{nCP1}}$** and ***Stream-*$_{\mathbf{nCP3}}$** , respectively, are both 0. No further simplification can be made on ***Stream-*$_{\mathbf{nCP2}}$** . Hence, the simplified ***Stream-*$_{\mathbf{nCP}}$** , denoted ***Stream-*$_{\mathbf{snCP}}$**, are ***Stream-*$_{\mathbf{snCP1}}$** = $\langle(b,0)\rangle$, ***Stream-*$_{\mathbf{snCP2}}$**=$\langle(c,1),(m,0),(k,0)\rangle$ and ***Stream-*$_{\mathbf{snCP3}}$**= $\langle(d,1)\rangle$.

Step 5: Concept Node Decomposition

A set of concept nodes contains exceptions if the label of at least one descendant of the *lcs* concept node in the set disagrees with that of the *lcs* concept node. Concept exceptions can be directly identified from the *simplified normalized concept node sequence partition* that still contains two or more concept nodes, e.g., ***Stream-*$_{\mathbf{snCP}}$** in Example 3. This step decomposes the *lcs* concept node in such a sequence by enumerating its descendants that are indifferent to any of the conflicting nodes.

Let $ind(c_{m,}c_n) = \neg subsume(c_m, c_n) \wedge \neg subsume(c_n, c_m)$ be an *indifferent* relation in which no concept node subsumes the other. Let **E** be the set of concept-node exceptions, which are all descendants of an *lcs* concept node *c* whose labels are different from the label of *c*. Let $\psi(c, \mathbf{E})$ be a function that returns the decomposition

of *lcs* concept node c with regard to **E**. The decomposition function is recursively defined as follows:

$$\psi(c,\mathbf{E}) = \mathbf{E} \cup \{c_m \mid c_m \in c\text{'s child nodes} \wedge ind(c_m, e_n) \text{ for each } e_n \in \mathbf{E}\}$$
$$\cup \{\psi(c_m,\mathbf{E}) \mid c_m \in c\text{'s child nodes} \wedge \neg ind(c_m, e_n) \text{ for some } e_n \in \mathbf{E}\}$$

It returns the union of the exception set and the set of disjoint, most general of *lcs* node's descendants that are indifferent to all concept nodes in the exception set. The ψ function simply flattens the *lcs* concept node in order to separate its descendant concept nodes with contradictory labels from the ones that agree with its label.

Example 4. The decomposition of concept node c in $snCSP_2$ from Example 3 results in ***Stream-*$_{\mathbf{snCP2}}$** $=\langle(i,1),(n,1),(m,0),(k,0)\rangle$ since $\psi(c,\{m,k\}) = \{m,k\} \cup \{i\} \cup \psi(j,\{m,k\}) = \{i,n,m,k\}$. Note that $\{m,k\}$ is the set of concept-node exceptions (see Figure 4.4).

Note that the concept node decomposition above resolves any conflict in hierarchical concept nodes. In a special case where all concept nodes returned by the instance generalization function are disjoint (i.e., form flat partitions), such a conflict would never happen. Thus, the current step would be useful only if the method employed for realizing the instance generalization function might return concept nodes that form a hierarchy.

Step 6: Concept Instantiation

The last step extends concept nodes in all *simplified*, *normalized concept node sequence partitions* (e.g., ***Stream-*$_{\mathbf{snCP}}$**), using the concept instantiation function ε,

into a set of artificially labeled instances $\mathbf{S}'$. All instances are labeled with the label values of their associated concept nodes, and are arranged into a sequence of pairs of instance and its label according to instance arrival times (the new stream ***Stream-S′***).

Example 5. From Examples 3 and 4, the list of concepts and their labels are (b,0), (d,1), (i,1), (n,1), (m,0) and (k,0). Suppose $\varepsilon(b)=\{x_2, x_4\}$, $\varepsilon(d)=\{x_1, x_8\}$, $\varepsilon(i)=\{x_3,x_7\}$, $\varepsilon(n)=\{x_5,x_{10}\}$, $\varepsilon(m)=\{x_6, x_{11}\}$ and $\varepsilon(k)=\{x_9,x_{12}\}$. Then, it generates the new stream ***Stream-S′*** = $\{(x_1,1), (x_2,0), (x_3,1), (x_4,0), (x_5,1), (x_7,1), (x_8,1), (x_9,0), (x_{10},1), (x_{11},0), (x_{12},0)\}$ containing the expanded set of relevant labeled data.

The quality of ***Stream-S′*** generated by the CDT component depends heavily on the accuracy of instance generalization function δ employed. As defined in Section 4.3.2, the function δ returns a concept node in the set $\mathbf{A}_x$ = {$x \cup x$'s ancestors} that represents the concept category of x. The concept node returned by the function can be either too specific or too general. If the concept node selected is too specific, the CDT component may be unable to detect the occurrence of concept drift, introducing noise that contains conflicting examples in the set $\mathbf{S}'$. To illustrate this, suppose the concept hierarchy contains a concept node A, which has child nodes B and C, and A is the correct node for representing the concept category of all instances covered by nodes B and C. Suppose also that B or C is the node that will be selected by the instance generalization function instead of A. If B's instance is used to establish the target concept A, which is demoted later using C's instance, then the CDT algorithm will not be able to detect the fact that the target concept A is no longer relevant

because the concept categories of *B* and *C*'s instances are considered different by the instance generalization function. As a result, the expanded set $\mathbf{S}'$ generated by the CDT algorithm will contain A's instances with conflicting label values, which are supposed to be the same. In addition, too specific a concept node also reduces the coverage of correct examples retrieved from the concept hierarchy, which potentially decreases the ability of a concept drift (or stable) learner to accurately learn target concepts from the retrieved examples.

Furthermore, an instance generalization function that returns too general concept node can also cause the CDT component to generate false positive (or negative) examples. Similar to the above illustration, suppose that a concept node *A* has child nodes *B* and *C*, but now *B* and *C* are the correct nodes for representing the concept categories of all instances covered by nodes *B* and *C*, respectively. Suppose also that the instance generalization function returns node *A* for any input that is either *B* or *C*'s instance. False positive (or false negative) examples will be generated when either *B* or *C* (but not both) is declared as the target (or non-target) concept because the instance generalization function, which recognizes *A* instead of *B* or *C*, would consider *B* and *C*'s instances the same concept category. Unlike concept node that is too specific, the more general concept node could increase the coverage of correct examples.

Given a perfect concept hierarchy and a perfect instance generalization function, any *stable concept learner*, rather than a concept drift learner, can be applied for learning all (artificially) labeled instances identified as $\mathbf{S}''$ in the last step

without requiring instance ordering. However, the set $\mathbf{S}'$ can contain noise as described above. While a stable concept learner could not properly learn conflicting instances in **S'**, a concept drift learner could mitigate this issue by learning the ***Stream-***$_{\mathbf{S}}'$, especially for recovering the failure in detecting a concept drift due to the problem associated with selecting too specific concept nodes. Nonetheless, the latter learner still cannot resolve a noise that results from overly generalizing instances. This observation suggests avoiding selecting too general concept node if at all possible.

4.3.4 Instance Generalization Scheme

This section develops the technique for realizing the instance generalization function δ. FEILDS employs a validation set for providing the information needed to recognize distinct concept nodes based on their general characteristics in the concept hierarchy. As will be described in more details in Chapter V, one of the concept node properties in a concept hierarchy generated by the concept formation system employed (Widyantoro, Ioerger, & Yen, 2002) is the *concept* (or *cluster*) *density*, which is calculated from the average distance to the nearest neighbor among the child concept nodes. The concept density in the hierarchy tends to decrease at higher-level concept nodes. That is, the density of a concept node covering a smaller number of instances is higher that the density of the concept node's ancestors. Distinct concept nodes are identified by thresholding the concept density information whose cutoff point is empirically determined from the validation set. This method is essentially

similar to the process in *proximity dendogram* cutting that identifies clusters in the cluster hierarchy according to dissimilarity levels (Jain & Dubes, 1988). First, a concept hierarchy is incrementally built from a stream of data in the validation set. Because the instances' concept categories are known in the validation set, distinct concept nodes can be accurately recognized from the concept hierarchy. The threshold is then calculated from the densities of these distinct nodes.

More specifically, let H be the concept (cluster) hierarchy generated from the validation set containing a set of known concept categories T. Let $n_c \in H$ be a concept node in the hierarchy that corresponds to a concept category $c \in T$. Furthermore, let $\varepsilon(n)$ be a set of leaf nodes (document instances) in the hierarchy that are the descendants of concept node n. Let $\varepsilon(c)$ be a set of document instances that are the members of concept category c. The cluster n_c is identified from H by:

$$n_c = \arg\max_{n \in H} \left\{ \sum_{x \in \varepsilon(n)} m(x, c) - \sum_{c' \in T - \{c\}} \sum_{x \in \varepsilon(n)} m(x, c') \right\} \qquad (4.1)$$

where m: $X \times T \rightarrow \{1,0\}$ is a binary matching function such that $m(x, c) = 1$ if $x \in \varepsilon(c)$, or 0 otherwise. Hence, n_c maximizes the difference between the numbers of instances that are members of c and non-c concept categories. Now let μ_c be the average distance to the nearest neighbor among n_c's child nodes; μ represents the node density in the concept hierarchy. Thus, a higher μ value corresponds to a lower-density node and vice versa. Let $\mu_{c\text{'s parent}}$ be the density of n_c's parent. Taking μ_c as the threshold poses the risk of overfitting to a more specific concept node while

selecting $\mu_{\text{c's parent}}$ is completely inappropriate because it also covers the instances of other concept categories (overgeneralization). Therefore, the threshold is selected at a value between μ_c and $\mu_{\text{c's parent}}$, averaged over all concept categories in T:

$$\theta_k = \frac{1}{|T|} \sum_{c \in T} \max \left\{ \mu_c, \mu_c + k \cdot \left(\mu_{c\text{'s parent}} - \mu_c \right) \right\} \tag{4.2}$$

where $0 \le k \le 1$ is a non-negative constant. By default, $k = 0.5$, which maximizes the margins between overfitting and overgeneralization.

A concept c is a distinct concept node if it satisfies the following conditions:

1) $\mu_p < \theta_k$ for $\forall p \in c$'s descendants (the densities of all c's descendants are higher than the threshold), **and**
2) $\mu_c \le \theta_k \le \mu_{c\text{'s parent}}$ (c is the lowest-density node whose density is still higher or at least the same as the threshold).

These conditions virtually cut the concept hierarchy into non-overlapping distinct concept nodes each of which represents the concept category of its descendant leaf nodes. Hence, the function $\delta(x)$ returns a distinct concept node c, as defined above, that is either x or one of x's ancestors.

4.4 Advantages and Shortcomings

Theoretically speaking, there are at least three benefits of FEILDS that exploits unlabeled data as described above:

1. In the absence of additional labeled data, FEILDS can automatically improve the performance of a concept drift learner over time as more relevant unlabeled data become available.

2. Provided that a perfect concept hierarchy can be constructed and a correct generalization of each instance can be realized, the number of labeled data becomes less relevant for improving the system's performance. Nonetheless, a minimal number of labeled data would still be needed in order to establish or to negate target concepts. This advantage makes it possible to apply FEILDS in a more realistic setting particularly in an information filtering domain in which a real user tends to give only a few relevance judgments.

3. Although in this dissertation FEILDS is applied only in information filtering domain, the technique presented is relatively general, which allows it to be used in other application domains as well. In addition, the output produced by the concept drift tracker (CDT) component in the FEILDS's framework provides a more flexible architecture, enabling any concept drift learner suitable for a particular application is to be applied.

FEILDS also has a drawback. As in typical on-line learning in which a learning method is expected to be incremental, the proposed method is only partially incremental. Although the construction of concept hierarchy is incremental, the process of target concept induction is carried out in a batch mode because the CDT component must reprocess the entire sequence of labeled data, and the concept drift learner has to relearn the new stream generated by the CDT component. This

certainly adds an extra computational cost to compensate for the lack of labeled data. Nonetheless, this extra cost is not discouraging because the number of labeled data is assumed to be small, and the batch process is performed only when needed and when the concept hierarchy has changed.

4.5 Summary

Learning concept drift from an incomplete labeled data stream poses a serious problem, both theoretically and practically, to existing concept drift learners. The main contribution of this chapter is the description of the FEILDS architecture that modularly extends the capabilities of existing concept drift learning algorithms in dealing with the issue. The system analyzes and expands the learners' original input streams with unlabeled data into new data streams that would improve the learnability of the learners' inputs.

FEILDS architecture consists of three main entities: (1) a concept formation system, (2) a concept hierarchy, and (3) a concept drift tracker. The system assumes that the input is a stream of labeled and unlabeled data. The concept formation system (CFS) incrementally constructs a concept hierarchy from the input stream in an unsupervised mode. The detail of the CFS algorithm is described in Chapter V. Utilized mainly by the concept drift tracker (CDT) component, the concept hierarchy serves as the knowledge base for recognizing the concept category of an instance, and for identifying relevant unlabeled data associated with a labeled instance. The CDT

component analyzes the labeled data stream, resolves any conflicting examples and then expands relevant data identified.

The CDT component performs its task in several steps. First, it transforms a stream of labeled instances into a stream of labeled concept nodes. Next, it partitions the concept node streams into several, smaller concept node streams with respect to the categories of concept nodes. This process essentially converts the problem of tracking multiple target concepts into several sub-problems of tracking a single target concept. Each partition is then normalized, allowing the system to identify the relevance of concept nodes within the partition. Any exception of relevance within the concept node's subcategory in each partition is resolved using the concept decomposition technique. Finally, all instances belonging to the relevant concept nodes are retrieved from the concept hierarchy, and are arranged into a new stream in the order of instances' arrival times.

One of the critical processes is identifying a concept node in the hierarchy that best represents the concept category of an instance. FEILDS addresses this problem by thresholding the concept density (i.e., one of the concept properties in the concept hierarchy). The threshold value is determined empirically from a validation set.

Finally, the advantage and shortcomings of FEILDS are described. It can take benefits from unlabeled data, is suitable for situation that can only provide a little data, and is potentially transferable to other domains. However, it also introduces extra computational costs.

CHAPTER V

CONCEPT FORMATION SYSTEM

Concept hierarchy is the central entity that plays a significant role in the FEILDS architecture described in Chapter IV. Constructing the concept hierarchy manually is not practical and not scalable particularly in information filtering setting because the information that needs to be incorporated incrementally proliferates from the input stream. Hence, a more desirable method is to generate the concept hierarchy automatically, which is a form of process known as *concept formation* (Gennari, Langley, & Fisher, 1989; Fisher, Pazzani, & Langley, 1991). The process basically resembles the task of generating a cluster hierarchy in *numerical taxonomy* (Jardine & Sibson, 1971).

This chapter describes and evaluates a new concept formation algorithm so-called HOMOGEN. It has been developed in this dissertation for constructing a quality concept hierarchy incrementally. The following two sections describe the motivations behind the development of the algorithm and then outline the general approaches taken. Section 5.3 describes the foundations, the detail of the concept formation algorithm, and its time complexity analysis. Section 5.4 presents the evaluation of the algorithm, followed by the discussion of related works in Section 5.5. This chapter concludes by summarizing its contribution in Section 5.6.

5.1 Design Motivations

Constructing a quality concept hierarchy is the main issue in designing an incremental concept formation system, and particularly for supporting the success of the framework described in Chapter IV. A quality concept (cluster) hierarchy is the one that represents the intrinsic hierarchical structures of concepts (clusters) that exist in the input data. Thus, the hierarchy construction should be capable of capturing such intrinsic cluster (concept) structures. More importantly, the quality of the hierarchy contructed should be comparable to the quality of those generated by non-incremental methods. While no consensus yet exists on what constitutes intrinsic structures, it is likely that such structures cannot be assumed to have certain shapes or distributions.

Although many incremental concept formation systems have been developed in the past, most of these systems have not been designed to work in the text (information filtering) domain. In addition, the construction of these systems is mostly biased toward the shape and the class distribution of clusters, which could prevent discovering intrinsic structures inherent in the data. Although systems such as DBSCAN (Ester *et al.*, 1996), CURE (Guha, Rastogi, & Shim, 1998) and CHAMELEON (Karypis, Han, & Kumar, 1999) can handle clusters with complex shapes and/or different sizes, these systems employ non-incremental methods. In incremental systems, COBWEB and its family (Gennari, Langley, & Fisher, 1989; Biswas, Weinberg, & Fisher, 1998; Wagstaff & Cardie, 2000) prefer clusters with similar sizes. ARACHNE tends to build compact clusters (McKusick & Langley, 1991). Similar cluster shapes are also formed by the INC system (Hadzikadic & Yun, 1989)

whose underlying algorithm is based on the prototypical representations. HIERARCH's constraints (Nevins, 1995), which place child nodes around their parent, also appear to exhibit a bias toward certain cluster shapes.

The sensitivity to input orderings is a long-standing problem in incremental conceptual clustering (Fisher, Xu, & Zard, 1992), hindering a concept formation system from consistently building a quality concept hierarchy. Two major issues that can affect the sensitivity problem are *nodes misplacement* and *early commitment* on cluster membership. The former is mainly due to the changes of hierarchy structures while processing new observations so that nodes that are previously well placed become misplaced. The latter refers to the use of a fixed threshold value for deciding an observation's cluster membership, for example, those applied in INC (Hadzikadic & Yun, 1989) and UNIMEM (Lebowitz, 1987), which despite its practicality has its limitation in that it cannot adapt a cluster membership test to local properties of the cluster. Hence, early commitment on a cluster membership decision could prevent capturing an intrinsic hierarchical structure in the data set. The design of a concept formation system should minimize the nodes misplacement problem and avoid providing early commitment on the cluster membership.

5.2 Design Approaches

Motivated by the above problems, the conceptual clustering approach of HOMOGEN works on a metric space model that views an object (e.g., observation, cluster or node) as a point in a high-dimensional space. The density of points is used to define

the characteristic of a good cluster and as guidance to hierarchically organize a set of clusters. Informally, the *density* describes the spatial distribution of points, measured in terms of the average distance from a point to its nearest neighbor (this will be formally defined in Section 5.3). A hierarchy is represented as a tree structure in which a node in the tree denotes a cluster in the hierarchy. The approach to concept formation aims to construct a tree structure with two properties:

Property 1 (Homogeneity). A tree structure satisfies a homogeneity property if every node in the tree consists of child nodes with similar density locally, with respect to the distances to nearest sibling among the child nodes.

Property 2 (Monotonicity). A tree structure satisfies a monotonicity property if the density of a node is always at least as high as the density of its parent. That is, the density of nodes monotonically increases along any path in the tree structure from the root to a leaf node.

These two properties serve as guiding principles for minimizing the occurrence of misplaced nodes during the hierarchy construction. The homogeneity requirement is needed in order to form clusters with local density properties, that is, the densities of objects vary in intrinsic cluster structures. This property also does not bias toward the shape and the class distribution of clusters that makes it suitable for tracking evolving clusters in an on-line situation. In fact, the homogeneity property also relaxes the commitment in the cluster membership function by flexibly defining it based on the cluster density. Accordingly, a new object can be a member of a

cluster if the inclusion of the new object in the cluster will not violate the homogeneity property of the cluster.

Additionally, the monotonicity property requirement is based on the observation that higher-level hierarchies in most hierarchical systems are generally used to represent entities with broader contexts. This characteristic can be captured with the notion of monotonicity, also in terms of cluster density. Thus, the monotonicity property helps properly organize the hierarchical structures of clusters. The structure needs to be changed whenever the property is violated, and construction of the new structure aims to satisfy this property. Taken together, both properties are expected to construct a natural hierarchical structure such that nearby (*resp*. distant) clusters share a lower (*resp.* an upper)-level ancestor.

The clustering process of HOMOGEN can be viewed as the incremental version of hierarchical agglomerative clustering (HAC) methods (Everitt, Landau, & Leese, 2001; Miyamoto, 1990; Jain & Dubes, 1988) with two respects. First, it works in a bottom-up fashion, which is the same as to the manner HAC algorithms form cluster hierarchies in batch modes. The second similarity is that HAC also produces cluster hierarchies that tend to be monotonic. HOMOGEN's notion of monotonicity is basically a generalization of HAC's notion since the former produces a more general tree structure. In particular, the monotonicity in HAC is always determined from the distance between two child nodes (clusters) because of the *binary tree* structure it generates. HOMOGEN's monotonicity is based on the average distance to the nearest neighbor among the child nodes so that its notion of monotonicity will be the same as

that in HAC if a node has only two off springs. Unlike HAC that is biased toward generating tree structures with the fewest branching factors, HOMOGEN relaxes this restriction that allows it to construct a more comprehensible hierarchical structure.

5.3 Concept Hierarchy Construction

The first part of this section describes the hierarchy representation and provides the operational definitions of the homogeneity and monotonicity properties. This part lays the foundations for analyzing the problem complexity and for the development of HOMOGEN's concept formation algorithm. The time complexity analysis of the algorithm will be given in the last part.

5.3.1 Formal Foundations

A hierarchy $H = \{N_1, N_2, \cdots, N_n\}$ is a tree consisting of n nodes. Each node in the tree maintains two types of information: *concept* and *density*. The *concept* summarizes the descriptions of all observations covered by a node. The *density* describes the spatial distribution of the child nodes. An *internal node* has at least two child nodes. A node in the tree represents a cluster whose members are the set of child nodes. A leaf node is a singleton cluster covering a single observation whose concept description is the description of the observation itself.

Concept Representation. Let an observation $o_i = \{o_{i1}, o_{i2}, \ldots, o_{id}\}$ be a d-dimensional point where o_{ij} represents the value of the j^{th} dimension of the i^{th} observation. A concept $C = (c_1, c_2, \ldots, c_d)$ also has the same dimension as that of the observation.

Let $\mathcal{E}(N)$, the extension of *N*, denote the set of observations (leaf nodes) that are descendants of *N*.

Definition 1 (*Concept Description*). The concept description *C* of a node *N* is the center of *m* observations (leaf nodes) that are descendants of *N*, that is, $C = \{c_1, c_2, \ldots, c_d\}$ where $c_j = \frac{1}{m}\sum_{i=1}^{m} o_{ij}$ and $o_{ij} \in \mathcal{E}(N)$.

Definition 1 is basically the *cluster center* in a prototype-based clustering. The calculation of cluster centers for domains with continuous attribute values is straightforward. In domains with nominal attribute values, observations need to be represented as *binary feature vectors* in which the value of each dimension is either one or zero representing the presence or absence of an attribute value.

Density Representation. The density of a node is defined as the average distance to the closest neighbor among the child nodes. A natural way of obtaining the distances to the nearest neighbors is from the path given by the minimum spanning tree (MST) of the child nodes. The density representation of a node *N* is a triple $D = \langle NDP, \mu, \sigma \rangle$ where $NDP = \{d_i \mid d_i \in \Re\}$ is a population of nearest distance d_i, μ and σ are the average and the standard deviation of *NDP*. Each d_i in *NDP* is the length of an edge, measured by the distance from a child node to its nearest sibling, in the MST structure connecting the child nodes of *N*. Thus, the μ and σ values are locally defined over the distances among the child nodes. The distance between two nodes, with respect to

the concept descriptions of the two nodes, in general can be measured by using L_n distance functions as defined below:

$$L_n(N_i, N_j) = \left(\sum_{k=1}^{d} \left(c_{ik} - c_{jk} \right)^n \right)^{\frac{1}{n}} \tag{5.1}$$

where C_i and C_j are the concept descriptions (i.e., clusters centers) of nodes N_i and N_j, respectively. For example, the *Manhattan* (*Euclidean*) distance function is derived from n =1 (n = 2). The average value of *NDP*, μ, characterizes the density of a node (cluster) in which the density is higher with lower μ value. The average distance of a leaf node is defined to be zero (i.e., the distance between the leaf node and itself). Hence, a leaf node represents a cluster with infinitely large density.

Definition 2 (Monotonic Node). Let μ_N and μ_P be the average nearest distances with respect to the density representations of nodes *N* and its parent *P* respectively. *N* is a monotonic node if only if $\mu_N \le \mu_P$, that is, the density of *N* is higher than or equal to the density of its parent.

Definition 3 (Homogeneous Node). Let $D_N = \langle NDP, \mu, \sigma \rangle$ be a density representation of a node *N*. Given a lower limit $L_L = \mu - k\sigma$ and an upper limit $U_L = \mu + k\sigma$ where *k* is a positive constant, the node *N* is homogeneous, with respect to *k*, if and only if $L_L \le d_i \le U_L$ for $\forall d_i \in NDP$. The functions L_L and U_L define the lower and upper bounds based on the mean and the variance of the population.

Thus, a node is homogeneous if its distribution of the distances to the closest neighbors among the child nodes is within a bounded range around the mean. The variance factor k in L_L and U_L functions controls the tightness of the bounds. On smaller k values (i.e., tighter bounds), the notions of homogeneity are less tolerant to the variations of the child nodes, reducing the node sizes and increasing the depths of the tree generated. As k values are made closer to zero, the trees constructed would approximate the *binary trees* produced by HAC methods. Although very restrictive, *binary tree* representations are capable of reconstructing any distinct clusters inherent within the data. Higher k values (i.e., looser bounds) behave in the opposite directions. Sufficiently large k values would form single-level tree structures with very large branching factors, which are obviously very undesirable because these structures cannot separate distinct clusters. Hence, the bound functions should be sufficiently tight for preserving the representational power of binary trees and yet loose enough for capturing intrinsic cluster structures. The experiment results as reported in this chapter are obtained by setting k=1 (see Section 5.4.2 for more discussion on this).

Definition 4 interprets the effects of observing a new point that is not within the bounds of a node. See Figure 5.1 for the illustration of this interpretation.

Definition 4 (Low and High Density Regions Formation). Let N be a homogenous node with L_L and U_L as the node's lower and upper limits, respectively. Given a new point A, let B be an N's child node that is the nearest neighbor to A. Let d be the

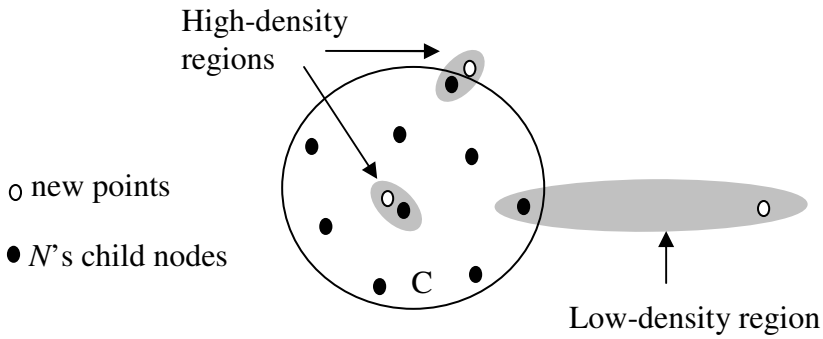


Figure 5.1: A set of new points that create regions of high and low density.

distance from *A* to *B*. If $d < L_L$, the region covering *A* and *B* represents the nearby (or within) *N* that has higher density than *N*. In this case, A (and B) is said to ***form a high-density region*** on N. If $d > U_L$, the region covering both the new point and its closest point represents a sparser region, in which case *A* (and *B*) is said to ***form a low-density region*** on *N*.

It is easy to show from Definitions 2 and 3 that a leaf node (or a singleton cluster) is by itself monotonic and homogeneous. An internal node with two child nodes is also a homogeneous node because the distance between the two child nodes is always within the node bounds. Accordingly, a hierarchy generated from the first two observations always satisfies the monotonicity and the homogeneity properties. Finally, any node forms a low-density region on a leaf node by Definition 4, except the leaf node's ancestors, due to the fact that the average nearest distance of a leaf node is zero.

5.3.2 A Preliminary Analysis of Problem Complexity

This section attempts to analyze the time complexity for producing a tree satisfying the monotonicity and homogeneity properties. First, it extends a batch clustering method for constructing a tree with the desired properties and then provides the time complexity for maintaining the tree properties in on-line situation. It also discusses an argument of why a tractable incremental algorithm that would perform similar task is an elusive problem.

Many variants of HAC algorithm can produce *binary tree* structures that meet the monotonicity and homogeneity criteria, with respect to Definitions 2 and 3. The tree structures generated by these algorithms, except the *Centroid*-based HAC, always satisfy the monotonicity property (Jain & Dubes, 1988) because a new higher-level cluster is formed in the order of increasing distance between two clusters. According to Definition 3, the *binary tree* structures generated by the agglomerative methods also satisfy the homogeneity property (i.e., due to the fact that a node with two child nodes is always homogeneous). The time complexity of these algorithms is at least $O(N^2)$ (Jain & Dubes, 1988).

Single-linkage method is a variant of HAC algorithms in which the distance between two clusters is determined by the distance of two closest data points in the different clusters. This variant can be extended using Definition 3 to generate more general tree structures that still meet the two criteria. Let's call this algorithm the **E**xtended **S**ingle **L**inkage HAC (or ESL-HAC for short). Briefly, the ESL-HAC

algorithm initially considers all points in the data set as singleton clusters similar to HAC algorithms. It selects a pair of clusters with the closest distance and then either (1) merges the two clusters if both are singleton clusters or if neither cluster can be inserted as the child node of the other, or (2) insert one of the clusters as a child node of another if doing so still maintains the homogeneity of the hosting cluster. The merging or insertion process is repeated for the next pair of clusters with the closest distance until there is only a single cluster. In the single-linkage method, the distance between nodes *A* and *B* is the same as the distance between *A* and a node *C* where *C* is the nearest *B*'s child node from *A*. If the distance between *A* and *B* is not smaller than the greatest distance among d_i where d_i is the distance between a child node of *B* and its nearest sibling, then inserting *A* as *B*'s child node will never decrease the average distance of d_i, and hence, will never violate the monotonicity properties of *B*'s child nodes. Since ESL-HAC algorithm processes pairs of clusters with increasing distances, the merging and the insertion operations described above also preserve the monotonicity property.

The main skeleton of ESL-HAC algorithm is the same as that of HAC algorithm so that the time complexity for generating a general tree satisfying the monotonicity and homogeneity criteria is also $O(N^2)$. In strictly on-line setting, these two properties can be preserved by rebuilding the tree each time encountering a new observation. The time complexity for continuously maintaining the hierarchy with the desired properties is therefore at least $O(N^3)$, which is clearly not interesting.

However, it is also not obvious whether there exists an algorithm with a time complexity of less than $O(N^2)$ that can incrementally incorporate a new point into an existing tree while still preserving the tree properties. The difficulty for inventing such an algorithm is based on the observation that an operation for maintaining the homogeneity of a node could destabilize the monotonicity of surrounding nodes (e.g., the child nodes and the ancestors), and vice versa. An algorithm that repeatedly repairs any node violating either property until the properties are satisfied would solve the problem but its termination cannot be guaranteed; it confounds the time complexity analysis of the algorithm.

Rather than pursuing both properties, the incremental algorithm of HOMOGEN takes a strategy that guarantees producing only a tree satisfying the homogeneity property. The algorithm relies only on heuristic rules for building a tree that tends to be monotonic. As will be discussed in Section 5.3.6, incorporating a new data point in HOMOGEN requires $O(\log N)$ time, making the time complexity of $O(N \log N)$ for incrementally processing all the data points.

5.3.3 The Algorithm Development

The approaches for generating a concept hierarchy incrementally can be divided into two stages, which are summarized by Figure 5.2. This two-stage algorithm is applied on observing the third and subsequent data points. The initial hierarchy is created by merging the first two points (the merging process will be described later). During the first stage, the algorithm locates a node in the hierarchy that can accept a new

Algorithm **Incremental Concept Formation** (*new observation*)

Stage I: Find and place the initial location for the *new observation*.
Stage II: Perform hierarchy restructuring on the affected nodes.

Figure 5.2: High-level description of HOMOGEN's concept formation algorithm.

observation in a bottom up fashion, and then inserts the new observation into the hosting node. The second stage performs hierarchy restructuring.

First Stage: Locating the Initial Placement in Concept Hierarchy

Locating the initial placement of a new observation is performed in the following sequence:

1. Find the best match concept over leaf nodes based on the closest distance to the new observation. To avoid exhaustive search by scanning the entire leaf nodes, the system performs a *beam search*, which maintains *k* best search paths, through the hierarchy in order to approximate the best match leaf node.
2. Starting from the parent of the closest leaf node, perform *upward search* to locate a cluster (or create a new cluster hierarchy) that can host the new observation. Heuristic rules are employed during this search.

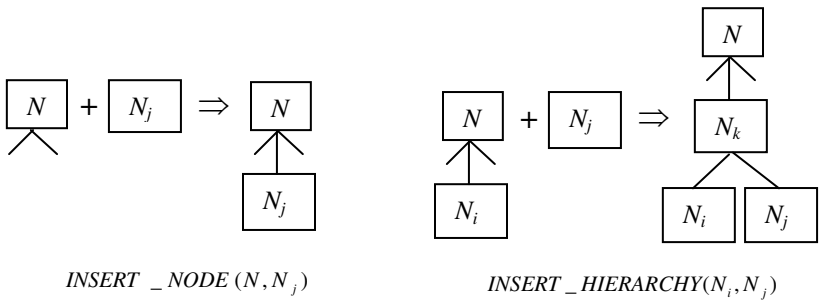


Figure 5.3: Node and hierarchy insertion operators.

Let's first define two basic operators that are needed to place a new observation in the hierarchy: node insertion operator and hierarchy insertion operator. For both operators, let N_j be the new observation.

Definition 5 (Node Insertion Operator) The node insertion operator, denoted by $INSERT\_NODE(N, N_j)$, inserts N_j as a new child of a node N (see Figure 5.3a).

Definition 6 (Hierarchy Insertion Operator) Let N_i be one of N's child nodes. The hierarchy insertion operator, denoted by $INSERT\_HIERARCHY(N_i, N_j)$, inserts a new node N_k in the hierarchy so that N_k becomes a parent of N_i and N_j, and is a child node of N (see Figure 5.3 b).

The *upward search* employs two heuristic rules to determine which insertion operator to apply. By utilizing the monotonicity property of cluster hierarchy, the

general idea of upward search is similar to the strategy of inserting a new element into a sorted list of bins.

Heuristic 1 (Node Insertion). Let *d* be the distance from a new observation N_j to the nearest child node of *N*, i.e., $d = \min\{L_n(N_j, N_i)\}$ where N_i is a child node of *N* (see Equation 5.1 for the definition of L_n). Let L_L and U_L be the lower and upper bounds of *N*, respectively, as in Definition 3. For *N* with two child nodes, these bounds are defined to be $L_L = k_L \cdot d_N$ and $U_L = k_U \cdot d_N$ where $0 < k_L < 1$ is a lower limit constant, $k_U > 1$ is an upper limit constant, and d_N is the distance between the two *N* child nodes. Perform $INSERT\_NODE(N, N_j)$ if and only if $L_L \le d \le U_L$.

In a node with two child nodes, the zero variance in the node's density representation would hardly allow the heuristic to insert a third child node. The heuristic addresses this problem by providing bounds derived only from the mean value. These special case bounds also play the role of determining the allowable variation in the distances to nearest neighbors. The bound constants are $k_L = 2/3$ and $k_U = 3/2$, which are determined empirically (see Section 5.4.2).

Heuristic 2 (Hierarchy Insertion). Let N_i be the child node of *N* closest to a new observation N_j. Perform $INSERT\_HIERARCHY(N_i, N_j)$ if and only N_j if forms *a high-density region* on *N*, and N_j *forms a low-density region* on at least one of *N*'s child nodes.

The node insertion operator, when applied on Heuristic Rule 1's conditions, attempts to preserve the cluster homogeneity. The applicability conditions of Heuristic Rule 2 are an indication that no cluster in the hierarchy can host the new observation without causing a significant density disturbance. Therefore, a new cluster hierarchy needs to be inserted in order to accommodate the new observation while minimizing the perturbation of the hierarchy monotonicity.

On each level in the hierarchy, the algorithm during the *upward search* examines the applicability conditions of each heuristic rule, applies the corresponding insertion operator whenever the conditions are satisfied and then stops. If none of the rules can be applied, the search proceeds to the next higher-level cluster (i.e., the parent of current cluster). If the search process reaches the top-level cluster (i.e., the root node), a new cluster hierarchy will be inserted at the top level using the hierarchy insertion operator, which replaces the root node with the new cluster.

A Walk Through Example

To clarify the idea during the first stage of algorithm, this section provides a walk through example explaining a step-by-step process as observing new data points (see Figure 5.4 for the illustration). Let's begin by observing the first two points, *A* and *B*, in which case the two points will be merged to generate an initial hierarchy (see Figure 5.4a). The algorithm starts executing the first stage when observing the third and subsequent points.

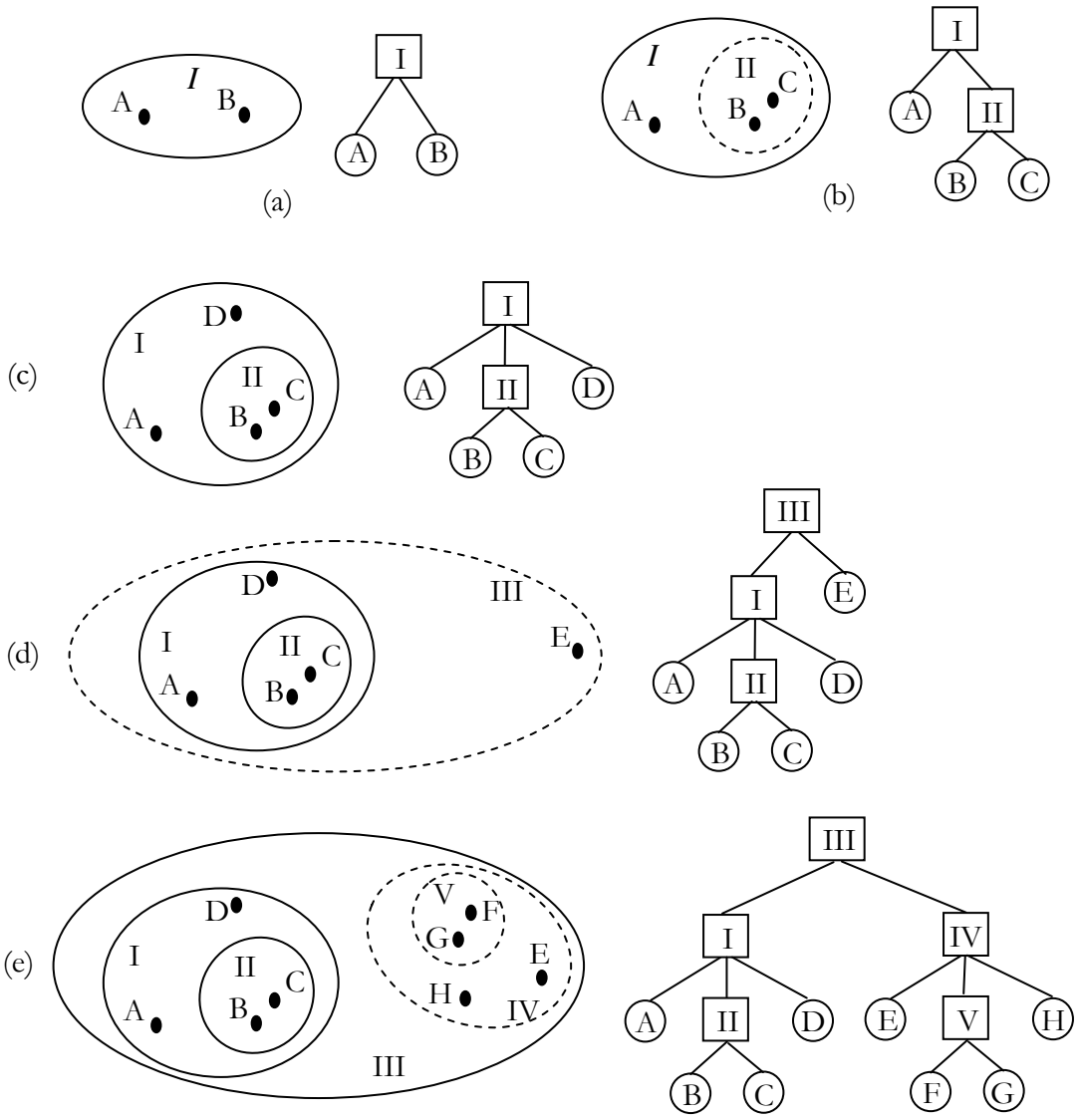


Figure 5.4: A walk through example.

Given a point *C* at the location as shown by Figure 5.4b, the closest point to *C* is *B*. Point *C* forms a low-density region on *B*, relative to the density of local cluster. Meanwhile, points *B* and *C* form a high-density region on cluster *I*. These two conditions satisfy the applicability of Heuristic Rule 2 that applies the hierarchy

insertion operator. As a result, it creates a new cluster *II* with *B* and *C* as its members and cluster *I* as the parent of cluster *II*.

Let's proceed by observing point *D* as illustrated in Figure 5.4c. The closest point to *D* is *C* in cluster *II*. Since *D* forms a low-density region on *C*, and points *D* (and *C*) form a low-density region on cluster *II*, the search then continues to the parent of cluster *II* (i.e., cluster *I*). The position of cluster *II* in cluster *I* is represented by the center of cluster *II*, which is in the middle of points *B* and *C*. Suppose the distance between *D* and the center of cluster *II* is still within the bounds of cluster *I* so that the applicability conditions of Heuristic Rule 1 are satisfied. As a result, point *D* is inserted as the member of cluster *I* using the node insertion operator.

Next, point *E* is observed (see Figure 5.4d). Point *C* in cluster *II* is the closest point to *E*. Point *E* forms a low-density region on *C* so that the search continues to the parent of *C*. It is obvious that *E* and *C* also form a lower dense region on cluster *II*, which directs the search to the parent of cluster *II* (i.e., cluster *I*). Cluster *II* is the member of cluster *I* closest to point *E*. Since *E* and the center of cluster *II* still form a lower dense region on cluster *I*, none of the heuristic rules is applicable on cluster *I*. Now the upward search has reached the top-level hierarchy. The first stage of algorithm then inserts a new cluster *III* on the top-level hierarchy so that the old root (cluster *I*) becomes a child of the new root (cluster *III*). Figure 5.4e illustrates the final hierarchy after observing points *F*, *G* and *H*.

Algorithm *Hierarchy Restructuring*

1. Let *crntNode* be the hosting node.
2. While (*crntNode* ≠ *null*)
3. Let *parentNode* ← Parent(*crntNode*).
4. Detect and recover the siblings of *crntNode* that are misplaced.
5. Perform homogeneity maintenance process on *crntNode*.
6. Let crntNode ← parentNode.

Figure 5.5: Hierarchy restructuring algorithm.

Second Stage: Hierarchy Restructuring

Changes in the hierarchy structures always occur after incorporating new observations, which are generally unseen during their initial placement. The restructuring process is performed to adapt the hierarchy to new structures by (1) recovering any misplaced nodes and (2) repairing the homogeneity property that has been violated. To do this effectively, the algorithm pinpoints nodes in the tree that are affected by the change of a node's structure once a new observation is incorporated in the hierarchy. Then, local operators are applied systematically on these affected nodes.

A node is affected if its concept description changes, which is an indication of structural change. The notion of concept descriptions in Definition 1 implies that the affected nodes are the hosting node and its ancestors, that is, all nodes that are in the path from the hosting node to the root inclusive. Obviously, hosting node is the most

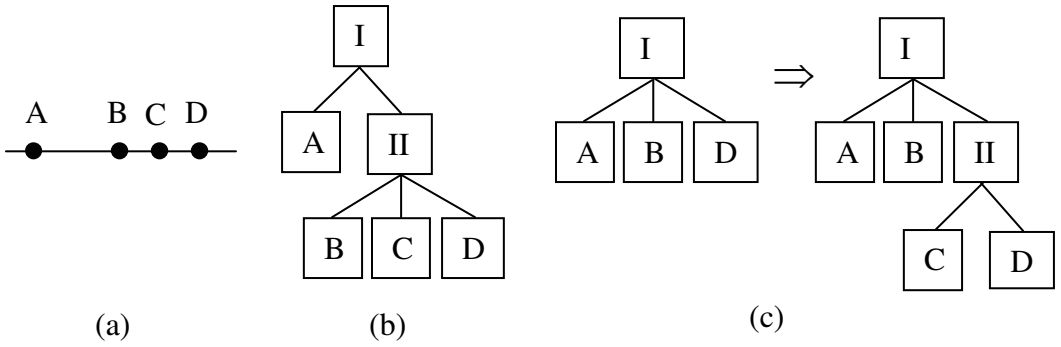


Figure 5.6: An example of structural change from observing a new instance.

affected node, followed by its parent and so on. Figure 5.5 summarizes the hierarchy-restructuring algorithm that performs the restructuring process on the hosting node and its ancestors. The following two sections will discuss steps 4 and 5 described in the figure.

Detection and Recovery of Misplaced Nodes

A hierarchy that meets the homogeneity and monotonicity properties is not unique. The hierarchy restructuring in HOMOGEN is biased toward constructing a hierarchy structure that places a set of homogeneous points into a single cluster rather than in multiple, multi-level clusters. Figure 5.6 illustrates an example that demonstrates the tendency of the first stage of algorithm for separating homogeneous points into a multi-level cluster. Briefly, Figure 5.6a depicts the spatial distribution of four points whose desired target hierarchy structure of these points is given by Figure 5.6b. Learning *A*, *B* and *D* in any order would result in concept structure as depicted by the

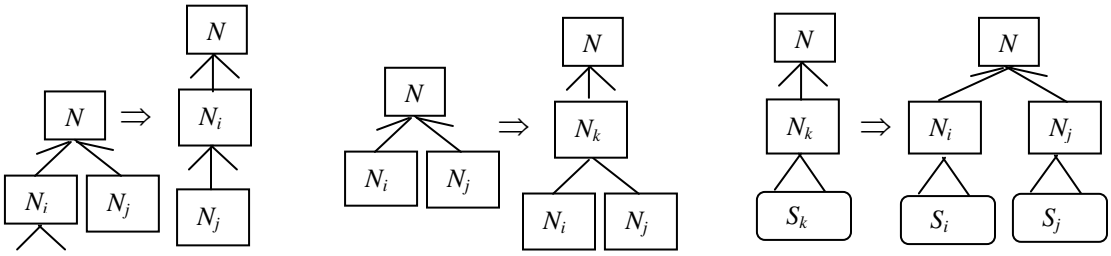


Figure 5.7: Demotion, merging and splitting restructuring operators.

left side of Figure 5.6c. Providing *C* as the last observation would trigger a structural change to the structures of the target hierarchy. Assume *C* is closer to *D* than to *B* so that *C* and *D* form a high-density region on cluster *I*. As shown by the right side of Figure 5.6c, the first stage of algorithm will produce a hierarchy structure that splits homogeneous points (e.g., *B*, *C* and *D*) in two hierarchy levels in that *B* is misplaced as the sibling of cluster *II*, which is supposed to be the child node of cluster *II*, even though the presence of *B* in cluster *I* on the final hierarchy may not necessarily violate the homogeneity and the monotonicity properties.

Stranded at upper hierarchy levels as illustrated above is an inevitable consequence of the first stage of algorithm. As the density of some regions in a cluster increases from observing new points, it will insert new cluster hierarchies on deeper hierarchy levels in an attempt to preserve the homogeneity and the monotonicity properties. As a result, more nodes could be misplaced at higher-level clusters, or more specifically, misplaced as the siblings of other nodes. The following

formally defines this problem and then provides a demotion operator that can eliminate it.

Definition 7 (Misplaced Sibling) Let N_i and N_j be siblings to one another. N_j is said to be misplaced as the sibling of N_i, denoted by **Misplaced_Sibling**(N_i, N_j), if and only if N_j does not form a low-density region on N_i.

Definition 8 (Demotion Operator) Let N_i and N_j be siblings to one another. A demotion operator, denoted by $DEMOTE(N_i, N_j)$, is a process of retracting N_j from its parent and inserting it as a child node of N_i (see Figure 5.7a).

Obviously, if N_j is misplaced as the sibling of N_i, $DEMOTE(N_i, N_j)$ will solve the problem by Definition 7. Since applying a single demotion operator could also lead to further problems to the N_i's remaining siblings, the algorithm checks the rest of the siblings and reapplies the demotion operator, repeatedly, until no misplaced sibling is found.

Figure 5.8 describes the detail process. The restriction on the next sibling chosen in Line 5 guarantees that once the selected node is found to be not a misplaced sibling, then neither do the remaining siblings. If the algorithm terminates by the second condition (i.e., *Siblings* = *null*), which means that N_i is the only child node of its parent, additional minor restructuring is performed (not shown in the algorithm) in order to satisfy the requirement that an internal node must have at least two child nodes.

Algorithm *Detection and Recovery of Misplaced Nodes* (N_i)

1. Let the input N_i be the recipient of demoted nodes.
2. Let *Siblings* ← the set of N_i 's siblings.
3. Let *No_Misplaced_Sibling* ← **false**.
4. Repeat
5. Let $N_j \in$ *Siblings* be the closest node to a child node of N_i .
6. **If Misplaced_Sibling** (N_i , N_j) (i.e., see Definition 7)
7. **Then** $DEMOTE(N_i, N_j)$,
8. Remove N_j from *Siblings*.
9. **Else** Let *No_Misplaced_Sibling* ← **true**.
10. **Until** (*No_Misplaced_Sibling* = **true**) or (*Siblings* = *null*).

Figure 5.8: Misplaced node detection and recovery algorithm.

Homogeneity Property Maintenance

This section describes the process of repairing a cluster whose homogeneity property has been violated. In such a case, some areas in the cluster form high and/or low-density regions. HOMOGEN eliminates a high-density region by merging two nearest nodes using a *merging operator*, which is defined below.

Definition 9 (Merging Operator) Merging operator, denoted by $MERGE(N_i, N_j)$ is $INSERT\_HIERARCHY(N_i, N_j)$ where N_j is a sibling of N_i (see Figure 5.7b).

The merging operator replaces two nodes in a cluster with a single node that is the center of the two nodes. The merging operator, therefore, has a likely effect of lessening the density around the center if the two nodes to be merged are restricted to those with the smallest nearest distance. Moreover, if the smallest nearest distance is further restricted to be below the cluster's lower limit, the merging operator will remove a high-density region from the cluster. Repeating the merging process on these nodes will eventually eliminate all high-density regions.

A low-density region can be removed by splitting the cluster into two or more smaller ones using sparser regions as the cutting points. The process is similar to Zahn's clustering algorithm that removes inconsistent edges on the MST structures to form connected components (1971). The following defines the splitting operation.

Definition 10 (Splitting Operator) Let N_k be a child node of *N*, and S_k be a set of child nodes of N_k (see Figure 5.7c for the illustration). Let θ be a splitting function that divides S_k into two disjoint subsets S_i and S_j, that is, $(S_i, S_j) = \theta(S_k)$ satisfying $S_k = S_i \cup S_j$ and $S_i \cap S_j = \varnothing$. Let $(N_i, N_j) = SPLIT(\theta, N_k)$ where *SPLIT* is a splitting operator. The *SPLIT* operator retracts N_k from *N* and makes N_i and N_j, as *N*'s child nodes where S_i and S_j are the sets of child nodes of N_i and N_j, respectively. If S_i or S_j contains a single child node, then that node becomes N_i or N_j, that is, effectively promoting the child node one level higher in the tree.

To maximally eliminate the low-density regions, the algorithm employs a splitting function θ that selects a cutting point on the middle of a path that connects an object with the farthest distance to its nearest neighbor. Using the MST graph of the cluster being split, the members and the MST structure of each split can be obtained by disconnecting the selected path. If the splitting operation is performed only when the farthest distance to the nearest neighbor exceeds the cluster's upper bound, then recursively applying this operator on each new split will eventually obtain a cluster that is free from low-density regions, in which case the splitting process stops.

Figure 5.9 describes the homogeneity maintenance process of a cluster that combines the merging and splitting operators. The termination conditions of the inner loop (in Line 6) and the algorithm guarantee that the input cluster has neither low-density region nor high-density region. Since a split node (N_i or N_j, in Line 10) that is promoted from a child node of N_k is already homogeneous and its homogeneity property is not affected by the *SPLIT* operator, the recursive calls to the homogeneity maintenance process (Lines 11 and 12) are not applied to this node. Let $S_u \cup N_k$ be the set of child nodes of *N* where S_u is the set of N_k's siblings. Working in a *divide and conquer* fashion, the algorithm receives an input cluster N_k and replaces N_k by a non empty set of homogeneous nodes S_v. That is, the set of *N*'s child nodes is now $S_u \cup S_v$.

Algorithm *Homogeneity Maintenance* (N_k)

1. Let an input N_k be the node that is being examined.
2. Repeat
3. Let N_i and N_j be the pair of neighbors among N_k's child nodes with the closest distance.
4. **If** N_i and N_j *form a high-density region* with respect to N_k,
5. **Then** $MERGE(N_i, N_j)$,
6. Until there is no high-density region found in N_k during the last iteration.
7. Let M_i be the child of N_k with the largest d_i and M_j be M_i's nearest neighbor where d_i is the distance from node *i* to its nearest neighbor.
8. **If** M_i and M_j *form a low-density region* in N_k,
9. **Then** Let $S_k \leftarrow$ the set of N_k's child nodes.
10. Let $(N_i, N_j) = SPLIT(\theta, N_k)$.
11. **If** $N_i \notin S_k$ **Then** *Call* Homogeneity Maintenance (N_i).
12. **If** $N_j \notin S_k$ **Then** *Call* Homogeneity Maintenance (N_j).

Figure 5.9: Homogeneity maintenance algorithm.

5.3.4 Time Complexity Analysis of HOMOGEN

Let *B* be the average branching factor of the tree[1] and *D* be the dimension of the data points (observations). For data with nominal attribute values, *D=AV* where *A* is the

[1] The analysis assumes that the notion of homogeneity in Definition 3 is defined over tighter bound functions. As discussed in Section 5.3.1, tighter bound functions will generate tree structures with better representational powers, that is, smaller branching factors. In all experiments $\mu \pm \sigma$ is used as the bound functions, which indeed construct trees with well-behaved branching factors ranging from 2.5 to 4.5.

number of attributes and V is the average number of attribute's values. D can be associated with the cost of calculating the distance between two objects, or the cost of updating the concept description of a node. Moreover, let n denote the number of observations that have been previously incorporated in the hierarchy. Thus, $\log B_n$ is the average depth of the hierarchy. The most expensive process is rebuilding the MST structure[2] every time the concept descriptions are modified, which is currently dominated by recalculating the distances of all pairs of child nodes (i.e., B^2D).

Finding the initial location during the first stage of algorithm involves searching the closest leaf node using a *beam* search through the hierarchy, performing *upward search* and then inserting the observation into the hosting node. Assume P is the *beam size*. The cost for determining the closest distance to a child node on each level in the hierarchy and on each beam path is BD, and therefore the total cost for finding the closest leaf node is $PBD \log B_n$. The *upward search* requires only $BD \log B_n$ time whereas inserting a single observation into a hosting node involves updating the concept description and the MST structure of the hosting node and its ancestors. The last step requires $(1 + B^2)D \log B_n$ time. Thus, the update time of the first stage of algorithm is $(B^2 + (P+1)B + 1)D \log B_n$, or $O(B^2 D \log_B n)$.

---

[2] Currently the implementation employs Prim's algorithm (Corment, Leiserson, & Rivest, 2001) to rebuild the MST structure, which has an every-case time complexity of $\Theta(B^2)$. Fortunately, there exists an incremental MST algorithm (Fredericson, 1985) with $\Theta(\sqrt{B-1})$ update time that could be used to improve the efficiency of the MST update.

Now, the analysis proceeds to the time complexity of the hierarchy restructuring (see Figure 5.5 for the algorithm description). Let *crntNode* be the node being restructured. The cost of recovering the misplaced siblings consists of two major components. The first component is finding the closest node to a node's child node (see line 5 in the algorithm described in Figure 5.8), which is amount to B^2D time. The second one is updating the concept descriptions (i.e., D time) and rebuilding the MST structures (i.e., B^2D time) of *crntNode* and its parent due to applying the *demotion* operator. On a worst-case scenario, the number of misplaced *crntNode*'s siblings is at most $(B-1)$ and thus requires $(B-1)(B^2D+2B^2+2D) = 3B^3D-3B^2D+2BD-2D$ time. Next, the MST structure update time is also the major cost during the homogeneity maintenance process (recall the description of the algorithm in Figure 5.9). Applying the *merging* and *splitting* operators requires updating two and three concept descriptions as well as their MST structures, respectively. On a worst case scenario, there will also be at most $(B-2)$ splitting operations on *crntNode* including its splits. Therefore, the cost of the homogeneity maintenance process originating from *crntNode* is at most $(B-2)(3B^2D+3D) = 3B^3D-6B^2D+3BD-6D$ time. Thus, the time for misplaced nodes restructuring and the homogeneity maintenance process is $6B^3D-9B^2D+5BD-8D$. Since the hierarchy restructuring is performed on the hosting node's ancestors along the path to the root, the total cost requires $(6B^3-9B^2+5B-8)D\log_B n$ time. This provides a time complexity of $B^3D\log_B n$ for the second stage algorithm.

Hence, $O(B^3 D \log_B n)$ is the legitimate time complexity for incorporating a single observation. This is one order higher than the time complexity of Fisher's COBWEB with respect to the branching factor B, which is $O(B^2 D \log_B n)$ (Fisher, 1987). The actual time of HOMOGEN's algorithm could be less because the number of child nodes that need to be restructured can be anywhere from none to $(B-1)$. Currently B^2D time, the MST update time could be improved to BD by maintaining the calculated distances, recalculating only those that are affected, and applying the Fredericson's MST incremental update algorithm. This possible improvement, however, comes with the price of maintaining more complicated data structures (Fredericson, 1985).

Given a sequence of N observations, the total cost to incorporate all observations is $B^3 D \sum_{n=1}^{N} \log_B N < B^3 DN \log_B N$. This gives the complexity of $O(N \log N)$, which is basically the same as COBWEB's time complexity (Fisher, 1987) and is comparable to the incremental version of WITT system (Hanson & Bauer, 1989). As another comparison, the time complexity of typical agglomerative methods is $O(N^2)$. The incremental algorithms are generally more efficient because these approaches can take advantage the tree structures generated during the clustering process. This privilege, however, is not possessed by the agglomerative methods.

5.4 Evaluating the Concept Formation Algorithm

This section describes the experimentation of HOMOGEN. The objective is to evaluate the system's performance by examining the quality of concepts (or clusters) hierarchy it generates. To avoid confusion, the evaluation is divided into two parts. The first part investigates the behavior of various restructuring strategies employed by the concept formation algorithm using synthetic and natural data sets, which represent structured data sets, and then compares its performance with other incremental systems. The second part evaluates the system in clustering text documents (i.e., unstructured data set). In this part, the system's performance is compared with those of HAC methods, the most common hierarchical clustering algorithms applied in this domain. The following defines several measures employed for quantifying the hierarchy quality.

5.4.1 Quantifying the Hierarchy Quality

The evaluation uses both internal and external criteria to quantify the hierarchy quality produced by HOMOGEN. A hierarchy quality that is based on the internal criterion measures the compliance of the hierarchy to the *monotonicity* and *homogeneity* properties. Due to its subjectivity, this measure is used only in evaluating the behavior of various components in HOMOGEN. Alternatively, an external criterion-based quality measure quantifies the hierarchy quality with respect to its match with an expected hierarchy structure (Jain & Dubes, 1988). The latter measure is more objective and can be used for comparison with the hierarchy

qualities produced by other systems. This measure is also employed to confirm the utility of the above two properties.

Internal Criterion-based Measure

In this measure the hierarchy quality is quantified by calculating the percentage of nodes in the hierarchy that satisfy the homogeneity property or the monotonicity property. Let *Non_Leaf _Nodes* be all internal nodes and the root. The percentage of nodes satisfying the homogeneity property by Definition 3 is given by Equation 5.2 below, which is the fraction of non leaf nodes that are found to be homogeneous.

$$Homogeneity = \frac{Homogeneous \quad Non\_Leaf\_Nodes}{\#Non\_Leaf\_Nodes} \times 100\% \qquad (5.2)$$

Meanwhile, the percentage of nodes satisfying the monotonicity property by Definition 2 is the fraction of monotonic internal nodes. Equation 5.3 gives the formulae needed.

$$Monotonicity = \frac{\#Monotonic \quad Internal \quad Nodes}{\#Internal \quad Nodes} \times 100\% \qquad (5.3)$$

Since leaf nodes always satisfy the two properties, these nodes in Equations 5.2 and 5.3 are not counted. The root node is also not counted in Equation 5.3.

External Criterion-based Measure

Cluster validation methods that are based on external references measure the degrees of overlap between partitions generated by a clustering algorithm and predefined structures. The degrees of match between two partitions can be calculated using *Rand*

index (Rand, 1971}, *Jaccard coefficient* (Theodoridis & Koutroumbas, 1999), *Hubert's γ-statistic* (Hubert & Arabie, 1985) or *Fowlkes and Mallows index* (Fowlkes & Mallows, 1983); among others. However, measures that have been developed to utilize external references are all fundamentally limited to flat partitions. The hierarchy quality is usually computed from the partition obtained by cutting *dendogram* generated by an HAC algorithm (Theodoridis & Koutroumbas, 1999; Fowlkes & Mallows, 1983), which results in a different value of quality measure on different specified cutting level.

This dissertation devises two methods for quantifying the quality of a cluster hierarchy that do not require cluster partitioning. Instead, the approaches are to search the best match cluster in the hierarchy that corresponds to a target (predefined) cluster. The first method is to measure the hierarchy quality that also considers the organizational structure of the discovered, distinct clusters. This method is applicable on data sets whose hierarchical structures are well known. The second one only measures the quality of distinct clusters found in the hierarchy.

Measuring the Quality of Hierarchy Structures

Given a hierarchy H_L produced by a system, the quality of H_L is quantified by measuring the degree of its match with a known target hierarchy H_T. Generally speaking, the degree of match between H_T and H_L is calculated by the number of nodes in H_T, except the root node, that match with their corresponding nodes in H_L.

Furthermore, a node in H_L is said to be the corresponding node in H_T if both nodes match *conceptually* and *structurally*.

Let $N_T \in H_T$ and $N_L \in H_L$ be nodes in the target hierarchy H_T and in the hierarchy produced by a system H_L, respectively, where both hierarchies are derived from the same set of observations. Let $\mathcal{E}(N)$ denote the set of observations (singleton nodes) that are descendants of node *N*. The generalized *Jaccard coefficient* (Miyamoto, 1990) is used to measure the degree of conceptual match between nodes N_T and N_L, denoted by $CMatch(N_T, N_L)$.

$$CMatch(N_T, N_L) = \frac{\left\| \mathcal{E}(N_T) \cap \mathcal{E}(N_L) \right\|}{\left\| \mathcal{E}(N_T) \cup \mathcal{E}(N_L) \right\|} \tag{5.4}$$

CMatch measures the overlap of two concepts from the cardinality ratio between shared and distinct observations covered by the concepts. For each target node N_T in H_T, let N_L^* be the corresponding node in H_L such that[3]:

$$N_L^* = \underset{(N_L \in H_L) \wedge (N_L \neq Root)}{\arg\max} \left\{ CMatch(N_T, N_L) \right\} \tag{5.5}$$

Then, the degree of structural match between N_T and N_L, denoted by $SMatch(N_T, N_L^*)$, is defined as the degree of conceptual match between the parents of N_T and N_L^*.

[3] Ties are handled by selecting the first encountered node with maximum value. Note that multiple target nodes could correspond to a single node in H_L. Since the objective is to measure the quality of target node reconstruction in H_L and not to partition H_L, this is still acceptable. Besides, its occurrence is extremely rare only when the target nodes are poorly reconstructed in H_L.

$$SMatch(N_T, N_L^*) = CMatch(Parent(N_T, Parent(N_L^*)) \quad (5.6)$$

Finally, by incorporating the conceptual match as well as the structural match above, the degree of match between H_T and H_L, denoted by $HMatch(H_T, H_L)$, is computed as follows:

$$HMatch(H_T, H_L) = \sum_{(N_T \in H_T) \wedge (N_T \neq Root)} CMatch(N_T, N_L^*) \cdot SMatch(N_T, N_L^*) \quad (5.7)$$

The root node in Equation 5.7 is not included because this node always contains the same set of observations as those covered by the root node of H_L. The maximum score varies depending on the number of target nodes defined in the target hierarchy.

Measuring the Quality of Distinct Clusters

In this measure, the hierarchy generated by a clustering algorithm is examined whether a distinct target cluster can be rediscovered. The degree of match between the target cluster and its corresponding cluster in the hierarchy, measured using the *CMatch* above, is then weighted according to the target cluster size. The final value is obtained by averaging the weighted *CMatch* over all target clusters, similar to the *micro-averaging* measuring technique (Yang *et al.*, 2000). More specifically, let *DATA* be the set of all observations, and $TC_i \in TC$ be the i^{th} target cluster in a set of target clusters *TC*. Let $\mathcal{E}(TC_i)$ denote the set of observations belonging to the target cluster TC_i such that $DATA = \bigcup_i \mathcal{E}(TC_i)$ for $\forall\, TC_i \in TC$ and $\mathcal{E}(TC_i) \cap \mathcal{E}(TC_j) = \varnothing$. Moreover, let H_L be a hierarchy produced by a system using all observations in *DATA*. For each $TC_i \in TC$, let N_L^* be the corresponding node in

H_L and be determined similarly as in Equation 5.5. The quality of H_L is then calculated as an *accuracy* measure denoting the percentage of match between the target clusters and their corresponding clusters in H_L, as defined by Equation 5.8 below[4].

$$Accuracy(T_C, H_L) = \frac{\sum_{TC_i \in TC} \|\mathcal{E}(TC_i)\| \times CMatch(TC_i, N_L^*)}{\|DATA\|} \times 100\% \qquad (5.8)$$

	Synthetic Data Sets			Natural Data Sets [a]		
	Grid	*Triangle*	*Symbol*	*Soybean Small*	*Soybean Large*	*Voting*
#Observations	288	108	27	47	307	435
#Target Clusters [b]	38	12	12	5	19	2
#Distinct Clusters	24	9	9	4	19	2
#Target Hierarchy Levels	4	2	2	2	–	–
#Attributes [c]	2	2	3	35	35	15
Dimension Size	2	2	39	76	132	48
Attribute Value Types	Cont	Cont.	Nom.	Nom.	Nom.	Nom.
Distance Functions	L_2	L_2	L_1	L_1	L_1	L_1

[a]From UCI repository of machine learning database (Blake & Merz, 1998)
[b]#TargetClusters = #Distinct Clusters + #Internal Nodes, except the root node, that groups the distinct concepts and their larger groups.
[c]The number of attributes does not include the target (class) attribute.

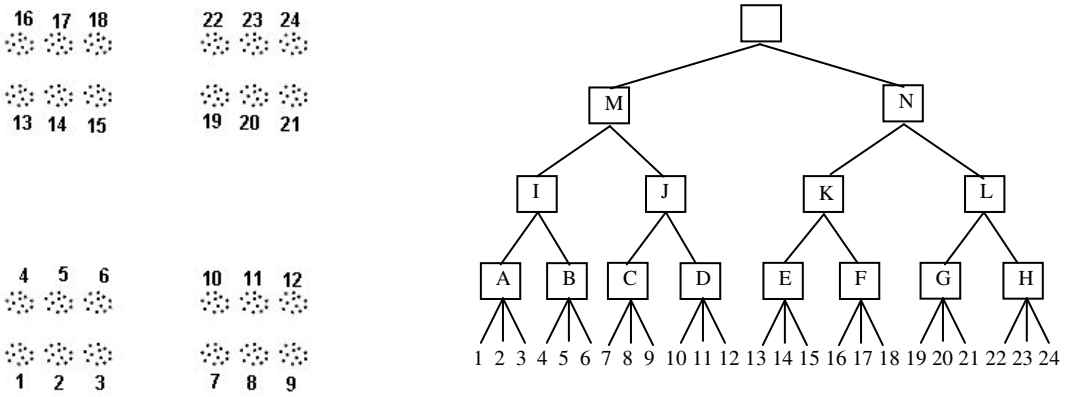
Table 5.1: Summary of non text data sets.

[4] Both $HMatch\ (H_T, H_L)$ and $Accuracy\,(T_C, H_L)$ are asymmetric measures.

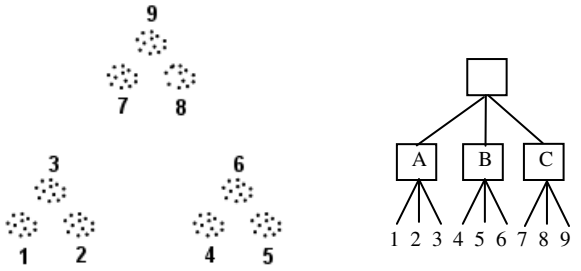
5.4.2 Experiments in Non Text Domains

The experiment uses six data sets as summarized in Table 5.1. The data sets *Grid*, *Triangle*, *Symbol*, and *Soybean Small* have known, clear target hierarchy structures while the hierarchy structure of the *Soybean Large* is unknown. Since the *Voting* data set contains only two target classes, it has the simplest hierarchy structure. Figure 5.10 shows the target hierarchy of the first four data sets; three are from synthetic domains. The first four data sets are used to evaluate the performance of HOMOGEN in discovering both the distinct clusters and their organizational structures inherent in the data sets.

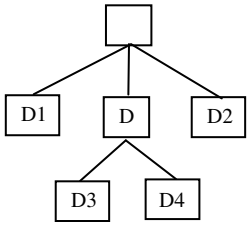
The experiments are run by providing a stream of observations to the incremental systems. The hierarchy quality produced by the system is measured once the last observation has been processed. To determine the appropriate tightness of the bound functions, HOMOGEN is run using the *Triangle* data set and the variance factor k is varied from 0.3 to 2 in all nodes with three or more child nodes. The lower bound constant k_L is also varied from 0.1 to 0.9 and from 1.1 to 2 for the upper bound constant k_U particularly for nodes with two child nodes in the Heuristic Rule 1. From these experiments, k=1, k_L=2/3, and k_U=3/2 are found to be among those that give good measures of hierarchy quality. These settings are then fixed for other data sets in the rest of experiments.



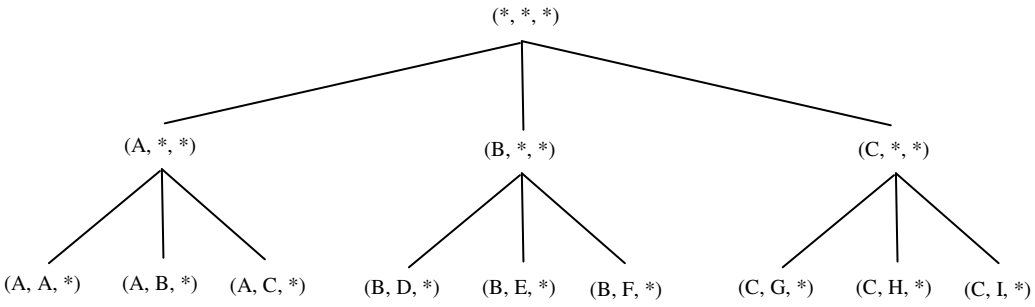
Grid



Triangle



Soybean Small



Symbol

Figure 5.10: Target hierarchy structures of four data sets.

The experiments are performed in two ordering scenarios: *random* and *bad* orderings. The observation in random ordering is selected randomly from one of the unseen observations regardless of the observations' classes. In *bad ordering*, the stream is ordered by observations' classes (Fisher, Xu, & Zard, 1992) where observation of a different class will not be given until all observations of the same class have been processed. In each case the experiment results are averaged over 25 trials.

Preliminary Experiments

The first experiments investigate the ability of HOMOGEN to construct a hierarchy satisfying the homogeneity and monotonicity properties. Table 5.2 describes the effects of applying various restructuring techniques on approximating the two hierarchy properties. The percentages of the homogeneity clusters and the monotonic nodes are calculated using Equations 2 and 3, respectively, and are averaged over 25 runs on random ordering. "+" and "−" denote significant improvement (and degradation, respectively) of performances, measured at most at 0.012 levels, relative to those achieved by running only the first stage algorithm.

As shown in the second column of the table, the heuristics employed during the first stage generate hierarchies that tend to be more monotonic (i.e., the percentages of monotonic nodes are much larger than those of the homogeneous nodes). Applying nodes misplacement restructuring during the second stage improves the hierarchy monotonicity but also reduces the percentage of homogeneous nodes.

Stage I: *Initial Observation Placement*	√	√	√	√
Stage II: *Misplaced Node Restructuring*		√		√
Stage II: *Homogeneity Maintenance*			√	√
Homogeneous Nodes (%)				
Grid	61.52	51.40^{-}	100.00^{+}	100.00^{+}
Triangle	62.03	51.01^{-}	100.00^{+}	100.00^{+}
Symbol	88.71	83.47^{-}	100.00^{+}	100.00^{+}
Soybean Small	71.79	56.24^{-}	100.00^{+}	100.00^{+}
Monotonic Nodes (%)				
Grid	98.08	99.33^{+}	93.97^{-}	98.16
Triangle	98.12	99.74^{+}	94.11^{-}	98.48
Symbol	100.00	97.81^{-}	97.53^{-}	100.00
Soybean Small	96.18	99.31^{+}	89.61^{-}	95.81

Table 5.2: The effect of restructuring techniques on achieving the homogeneity and monotonicity properties.

The homogeneity maintenance process, on the contrary, can repair all the homogeneity violations although it also decreases the number of monotonic nodes. Nonetheless, performing misplaced nodes recovery that is followed by the homogeneity maintenance process prevents degrading the monotonicity property from the latter process with respect to the percentages of monotonic nodes achieved by first stage process. Additionally, combining these two restructuring strategies in the proper order preserves the hierarchy homogeneity property.

The next experiments observe the effects of restructuring processes on the hierarchy quality with respect to its match with a known target hierarchy structure. Table 5.3 summarizes this observation. The hierarchy quality is measured by using Equation 5.7, averaged over 25 trials. The maximum quality scores are based on

Stage I: Initial Placement	√	√	√	√	√	Max Scores
Stage II Rest. Method:						
− *Misplace Nodes*		√		√	√	
− *Homogeneity Maint.*			√	√	√	
Stage II Rest. Scopes:						
− *Hosting Node (HN)*		√	√	√	√	
− *HN's Ancestors*		√	√	√		
Random Ordering						
Grid	35.38	35.17	35.58	38.00^{+}	30.81^{-}	38
Triangle	11.60	11.97^{+}	11.65	12.00^{+}	10.81^{-}	12
Symbol	9.31	6.37^{-}	11.68^{+}	12.00^{+}	12.00^{+}	12
Soybean Small	4.26	4.43	4.55^{+}	4.68^{+}	4.37	5
Bad Ordering						
Grid	31.75	32.74	33.80^{+}	37.97^{+}	32.25	38
Triangle	11.93	12.00	11.87	12.00	11.28^{-}	12
Symbol	9.79	6.34^{-}	11.76^{+}	12.00^{+}	12.00^{+}	12
Soybean Small	4.31	4.59^{+}	4.58	4.61^{+}	4.48	5

Table 5.3: The effect of various restructuring processes on the hierarchy quality.

#target clusters described in Table 5.1. The improvement "+" and degradation "–" of performances over those in the second column are statistically significant at 0.05 levels.

As shown on the second column of the table, the performances achieved without performing any further restructuring process are not optimal. Additionally, misplaced nodes restructuring could improve or degrade the hierarchy quality, which could be related to the fact that this restructuring process improves the monotonicity property and also degrades the homogeneity property (i.e., see Column 3 of Table 5.2. Moreover, applying only the homogeneity maintenance process is likely to improve

the hierarchy quality. As combining both restructuring techniques increases the percentages of nodes satisfying the homogeneity and monotonicity properties (i.e., see the last column of Table 5.2), it is reasonable to expect that the full restructuring processes would significantly improve the hierarchy quality. The fifth column of Table 5.3 confirms this expectation. The results described in this column are produced by the restructuring process that is applied on the hosting node and its ancestors, representing a tradeoff between the local and global approaches. Finally, the sixth column of Table 5.3 reports non-optimal performances obtained when the restructuring process is applied only on the hosting node, which is more similar to the local approaches.

To sum up, it has been empirically shown that the homogeneity and monotonicity are indeed desirable properties. As indicated in the experiment results, improving the hierarchy in satisfying these properties leads to producing a better measure of hierarchy quality that is independent of the hierarchical clustering objectives.

Performance Comparison with Other Incremental Systems

In this section the performances of HOMOGEN are compared with those of COBWEB (Fisher, 1987) and two versions of ARACHNE systems. The first version of ARACHNE, denoted by ARACHNE-L(ocal), implements the original ARACHNE's control strategy as described by McKusick and Langley (1991). This version applies restructuring operators on neighboring nodes that violate the nodes' constraints. The second

version, ARACHNE-G(lobal), extends ARACHNE-L by pushing the power of tree constraints employed by the system further into its limit. In particular, it also globally searches and restructures nodes that do not adhere to the constraints, and iteratively performs this process until all nodes obey the imposed constraints or until a maximum number of global restructuring iterations has been reached.

Table 5.4 provides the performance comparison of HOMOGEN with other incremental systems with respect to the systems' abilities to rediscover distinct clusters inherent in the data and to properly organize the discovered clusters into higher-level clusters. HOMOGEN on the four data sets consistently generates better hierarchy qualities than other systems. On bad ordering, the hierarchy qualities produced by the system are as good as those on random ordering. This evidently indicates that HOMOGEN is relatively insensitive to input ordering. COBWEB, in contrast, suffers from input ordering where its performances drop on bad orderings by approximately 29% and 24% for the *Soybean Small* and *Symbol*, respectively, averaged over the 25 trials. The observation of COBWEB's behavior that is sensitive to input ordering, as was shown in Table 5.4, is consistent with the results reported by Fisher, Xu and Zard, (1992). Furthermore, both versions of ARACHNE are relatively not affected by the input ordering although the performances of these systems drop by approximately 24% on the average on the *Soybean Small* data set. The systems are also unable to reproduce a more complex hierarchy structure such as in the *Grid* data set.

	HOMOGEN	COBWEB	ARACHNE-L	ARACHNE-G	Max Scores
	Random Ordering				
Grid	38.00	–	24.20	24.25	38
Triangle	12.00	–	11.99	12.00	12
Symbol	12.00*	9.62	9.75	11.31	12
Soybean Small	4.67*	4.23	3.81	4.29	5
	Bad Ordering				
Grid	37.97	–	26.65	26.65	38
Triangle	12.00*	–	11.81	11.85	12
Symbol	12.00*	7.12	10.18	11.19	12
Soybean Small	4.61	2.91	2.78	3.30	5

Table 5.4: The quality of hierarchy structures. The quality is measured according to Equation 5.7 averaged over 25 trials.

Next, the ability of systems in rediscovering distinct clusters regardless of the cluster hierarchy is observed. The same experiments as described previously are performed but the hierarchy qualities are measured using Equation 5.8. Now, *Soybean Large* and *Voting* data sets are also included. Table 5.5 summarizes the experiment results, averaged over 25 runs. The table shows that HOMOGEN performs comparably well to or better than the other systems. Consistent with the experiment results on bad ordering scenario described earlier in Table 5.4, the performances of HOMOGEN are even better, the performances of COBWEB are degraded while the ARACHNE's performances could be degraded or improved. Note that on *bad ordering* the underlying partition of the seen instances becomes suddenly unbalanced every time the data stream starts supplying a new class of instances. The performance of COBWEB, which is always worse on *bad ordering*, might have been due to the system's bias against unbalanced partitions (Fisher, 1996), leading to construct

	HOMOGEN	COBWEB	ARACHNE-L	ARACHNE-G
	Accuracy (%) on Random Ordering			
Grid	**100.00**	–	84.50	85.76
Triangle	**100.00**	–	**99.93**	**100.00**
Symbol	**100.00**	87.99	95.74	**99.85**
Soybean Small	**96.00**	**94.03**	83.38	**96.83**
Soybean Large	**59.18**	55.91	47.61	53.66
Voting	**79.07**	75.22	74.10	76.42
	Accuracy (%) on Bad Ordering			
Grid	**99.99**	–	94.43	95.09
Triangle	**100.00**	–	98.55	98.89
Symbol	**100.00**	71.18	96.96	**100.00**
Soybean Small	**97.28**	72.32	67.96	85.92
Soybean Large	**61.61**	50.31	49.74	53.26
Voting	**79.60**	68.40	63.79	75.22

Table 5.5: The quality of distinct clusters. (measured using Equation 5.8) The differences of bold numbers are statistically significant from non-bold numbers on the same row at 0.001 levels.

hierarchy structures with lower quality measures. The consistency of HOMOGEN in maintaining good performance on this ordering is an evidence that its clustering process, guided by the homogeneity property, is much less affected by the temporal change in cluster class distribution.

One can notice from Tables 5.4 and 5.5 that the performances of ARACHNE-G are relatively comparable to those of HOMOGEN on data sets containing distinct and compact clusters (e.g., *Soybean Small*, *Triangle* and *Symbol*). A plausible explanation for this observation is the bias in ARACHNE's restructuring constraints (McKusick & Langley, 1991) that prefer to form a cluster whose members are closer to the cluster center. The clustering of HOMOGEN, in contrast, is relatively not affected by the

cluster shapes. For example, the shapes of clusters *A* through *H* in *Grid* data set are obviously different from the rest of clusters, and HOMOGEN is able to properly identify these clusters. Furthermore, the cluster boundaries on *Voting* and *Soybean Large* are also not clear-cut, indicating the irregularity of cluster shapes and/or the overlap between clusters. Yet HOMOGEN performs better on these data sets. To some extent, this confirms the expectation that the homogeneity property can guide the incremental process of HOMOGEN to reconstruct clusters of fairly arbitrary shapes.

5.4.3 Experiments in Text Domains

A subset of the Reuters-21578 1.0 test collection (this collection is available at UCI KDD Archive (Blake & Merz, 1998) is used for experiments. The original collection contains 21,578 stories divided into 135 topics. Of these stories, 12,902 had been assigned to 118 categories; one category has approximately 4000 documents while most of the categories contain less than ten documents. Among these topics, only six topics are used from the training set part of the *ModApte* split (Apté, Damerau, & Weiss, 1994) with moderate topic sizes in order to avoid a bias toward large topic sizes. Furthermore, since each topic may have multiple topic categories that could confuse the assessment in measuring the cluster quality, the experiments use only the stories from the selected topics that were assigned a single topic category. The number of selected documents is 951 consisting of target topics *Coffee* (90), Crude (253), *Gold* (70), *Interest* (190), *Sugar* (97) and *Trade* (251).

Text document hierarchy is evaluated from the quality of target topic categories found in the hierarchy as measured according to Equation 5.8. The same parameter values for the cluster's bounds are employed as those applied in the previous experiments. The distance between two documents or clusters is measured by Euclidean distance function. Because a document topic is independent of the document length, the concept representation (recall Definition 1) of each node is normalized by Euclidean normalization. Specifically, given a concept description $C = (c_1, c_2, \ldots, c_d)$, the normalized concept description of C is $C' = (c'_1, c'_2, \ldots, c'_d)$ where $c'_j = \frac{c_j}{\sqrt{\sum_{i=1}^{d} c_i^2}}$ and $\sum_{j=1}^{d} c'^2_j = 1$.

Seven variants of HAC (i.e., non-incremental algorithms for hierarchical clustering) are considered for performance comparison. Briefly, HAC initially considers all points in the data set as singleton clusters, and then repeatedly merges two clusters with the closest distance until there is only a single cluster. The seven variants differ from each other in their methods in calculating the distances of a cluster to a non-singleton cluster. Lance and Williams provide recurrent formula for calculating these distances (Lance & Williams, 1967):

$$d_{hk} = \alpha_i d_{hi} + \alpha_j d_{hj} + \beta d_{ij} + \gamma \left| d_{hi} - d_{hj} \right| \qquad (5.9)$$

In the equation above, d_{hk} is the distance between two clusters *h* and *k*, and cluster *k* is the parent of cluster *i* and cluster *j*. More specifically, the clustering process starts by calculating all distances of document pairs using Equation 5.1. The recurrent

	α_i	α_j	β	γ
Single-link	0.5	0.5	0	–0.5
Complete-link	0.5	0.5	0	0.5
Group-average	$\frac{n_i}{n_i + n_j}$	$\frac{n_j}{n_i + n_j}$	0	0
Weighted-average	0.5	0.5	0	0
Centroid	$\frac{n_i}{n_i + n_j}$	$\frac{n_j}{n_i + n_j}$	$-\alpha_i \beta_j$	0
Median Method	0.5	0.5	–0.25	0
Ward's Method	$\frac{n_h + n_i}{n_h + n_i + n_j}$	$\frac{n_h + n_j}{n_h + n_i + n_j}$	$-\frac{n_h}{n_h + n_i + n_j}$	0

Table 5.6: Parameter values for the agglomerative clustering variants. Note that n_m denotes the number of data points that belong to a cluster *m*.

formula above is then employed to calculate the distances between existing clusters and a new non-singleton cluster, after two clusters are merged, using the distances of cluster pairs that have already been computed earlier. By implementing a generic agglomerative clustering, variants are determined from the parameters in the recurrent formula above. Table 5.6 provides the common parameter values for each of the HAC variants (Everitt, Landau, & Leese, 2001; Miyamoto, 1990; Jain & Dubes, 1988).

Document Pre-Processing, Feature Selection and Weighting

Text documents represent a noisy domain in which many features (words) tend to be irrelevant. Feature subset selection and feature weighting are two important processes to deal with this problem and this section describes these two processes. Each document is pre-processed as follows:

- Ignoring case and removing punctuation.
- Extracting unique words and bi-grams (i.e., two-word sequence that occurs at least twice in a document). "*term*" will be used to denote a word or a bi-gram.
- Removing all stop words (e.g., "a", "the", "although", etc.).
- Counting the term frequency TF (i.e., the number of times the term occurs in the document) for each of the remaining unique terms.

Each document is now represented by a feature vector containing a set of unique terms and their term frequencies. Note that the document pre-processing is ordering independent.

The feature selection process is applied to remove irrelevant terms from a document feature vector. Unlike in a *supervised* learning method that can employ *information-theoretic* or other well-grounded approaches for selecting a set of discriminating features from training examples, the nature of the clustering task makes it difficult to apply such methods. Instead, two alternatives of *heuristics* are considered here:

1. **MDF-FS**: minimum document frequency-based feature selection that selects terms occurring in at least *n* documents, and
2. **MTF-FS**: minimum term frequency-based feature selection that selects a term *t* if there exists at least one document in which *t* occurs at least *m* times.

The first alternative, which uses document frequency for filtering non-relevant features, is common in information retrieval, text classification (Joachims, 1997), and text clustering (Dhillon & Modha, 2001). Terms with low document frequency are non-content bearing and thus cannot be used as the discriminatory features. The second alternative assumes that term frequency is an indicator for topical words. Terms thus must appear with high term frequency in at least one document in order to be considered as topical words. Although term frequency has been heavily used for feature weighting, it has been rarely exploited for feature selection process.

The last process is to weigh each selected feature. Two feature weighting methods are considered: *term frequency* (TF) and *term frequency inverse document frequency* (TF-IDF). The former only uses term frequency as the weight of term while the latter also takes into account the document frequency. The TF-IDF has been well studied in Information Retrieval and has been shown to improve the retrieval effectiveness (recall Equation 3.1 in Chapter III). All feature weights are then normalized using Euclidean normalization.

The incremental system performs feature selection and weighting on the fly as it receives a new document to learn. In this system, the statistical information needed for feature selection is derived only from documents that have been previously processed. This clearly poses a problem for the MDF-FS feature selection because at least the first $(n-1)$ seen documents will never be included in the clustering process. Therefore, only the MTF-FS feature selection is used in the incremental system. Although it cannot completely avoid throwing a document out, the likelihood of the

	Accuracy(%)				Total #Features			
Sample Sizes	75	200	400	600	75	200	400	600
mtf = 1	77.14	77.30	69.50	66.62	2151	3769	5875	7270
mtf = 2	80.82	80.70	72.61	71.42	929	1896	3211	4163
mtf = 3	85.85	83.94	77.49	76.20	366	748	1234	1576
mtf = 4	92.14	**93.06**	88.79	87.56	184	396	654	834
mtf = 5	91.25	92.29	**89.94**	**89.49**	116	241	398	510
mtf = 6	**93.18**	92.60	89.44	89.19	79	156	266	342
mtf = 7	91.51	89.95	87.00	87.31	54	103	181	234
mtf = 8	91.75	88.27	85.20	86.01	39	75	131	168
mtf = 9	86.69	84.47	81.62	83.64	30	57	97	126
mtf=10	88.98	83.65	80.55	82.04	25	45	74	98

Table 5.7: The sensitivity of MTF-FS parameter values on HOMOGEN over various sample sizes. The results are averaged over 25 trials. Bold numbers indicate the highest accuracies in their respective sample sizes over various *mtf* values. Note that the feature sizes are the final counts once the system processes the last documents. The actual feature sizes vary during the clustering process and can grow when the system encounters new distinct words by processing more documents.

MTF-FS feature selection for encountering such a problem is much smaller. The TF-IDF weighting method, with similar reason, is also not applicable so that the system only uses the TF weighting method. During the course of incremental learning, a feature that is considered irrelevant in earlier seen documents could become suddenly relevant in more recent documents. To maintain the incremental nature of the system, documents that have already been learned are not reprocessed. In batch systems, the feature selection and weighting processes are performed over all documents well before the clustering process begins.

Experiment Results

First of all, the experiments are conducted to explore the sensitivity of MTF-FS feature selection parameter values on HOMOGEN using smaller sample sizes. The minimum term frequency (*mtf*) values in the experiments are varied from 1 to 10. Setting *mtf* = 1 is identical to running the system without performing feature selection process, and increasing the *mtf* value will reduce the total number of features selected during the clustering process. Table 5.7 describes the experiment results. In general, the system's accuracies improve with increased *mtf* values until optimal points are reached, and then the performances degrade slowly by further reducing the feature sizes. This entails that HOMOGEN's performance is affected by the presence of noise. Reducing the amount of noise by removing non-relevant features improves the system performance. However, aggressively removing the features considered as noise will also eliminate salient features and degrade the system's performance. Clearly, feature selection helps improve the hierarchy qualities provided the right *mtf* values.

The next experiments exploit the peak accuracies that can be achieved by HOMOGEN and the seven HAC variants on the full data set (951 documents consisting of six topics). The experiments are conducted by varying the pairs of feature weighting method (i.e., TF or TF-IDF) and feature selection method (i.e., MTF-FS or MDF-FS) that will be applied in each HAC variant. Because the value of feature selection parameter affects the hierarchy quality, each clustering algorithm is run several times on different values of feature selection parameter and the one that

	Accuracy (%) (*parameter value*)			
Feature Selection	MTF-FS		MDF-FS	
Term Weighting	TF	TF-IDF	TF	TF-IDF
HOMOGEN	89.32 (*5*)	–	–	–
Single-link	70.20 (*9*)	61.71 (*12*)	59.80 (*48*)	62.45 (*64*)
Complete-link	72.81 (*2*)	68.04 (*1*)	72.81 (*4*)	68.01 (*1*)
Group-average	89.16 (*4*)	86.86 (*4*)	81.12 (*2*)	80.24 (*8*)
Weighted-average	88.62 (*5*)	83.94 (*4*)	76.05 (*8*)	80.42 (*32*)
Centroid	83.43 (*14*)	79.05 (*14*)	58.31 (*64*)	50.18 (*80*)
Median Method	74.37 (*8*)	67.80 (*12*)	62.09 (*80*)	61.34 (*64*)
Ward's Method	84.36 (*5*)	83.37 (*6*)	77.19 (*8*)	80.11 (*32*)

Table 5.8: Peak accuracies achieved by HOMOGEN and HAC methods. The accuracy of HOMOGEN is averaged over 25 runs. The italic numbers following the accuracies are the parameter values of their respective feature selection methods (i.e., *mtf* or *mdf* values) that produce the results.

maximizes the hierarchy quality is taken as the representative of the best result of a clustering algorithm. More specifically, the best result is taken by varying the *mtf* values from 1 to 15 for the MTF-FS feature selection, or by varying the minimum document frequency (*mdf*) values to 2, 4, 8, 12, 16, 32, 48, 64, 80 and 96 for the MDF-FS feature selection.

Table 5.8 presents the best results for each variation of feature selection and weighting methods as well as their corresponding parameter settings. The best accuracy from HAC algorithms is achieved by the *Group-average* method (89.16%) and the peak performance attained by HOMOGEN is slightly higher (89.32%). A higher parameter value, shown next to the accuracy in the table, is an indication that the corresponding clustering algorithm is more sensitive to noise since it needs to be more aggressive in removing irrelevant features in order to maximize its performance.

HOMOGEN

	Document Topics						Total
	Coffee	*Crude*	*Gold*	*Interest*	*Sugar*	*Trade*	Docs.
Cluster-1	81.28	0.20	–	0.16	–	0.72	82.36
Cluster-2	0.08	226.72	1.60	1.24	0.16	3.44	233.34
Cluster-3	–	0.08	64.88	–	–	0.08	65.04
Cluster-4	1.04	2.96	0.24	175.32	0.88	6.64	187.08
Cluster-5	0.60	0.32	0.68	–	90.28	0.12	92.00
Cluster-6	0.24	0.96	0.04	5.24	0.04	227.92	234.44
#Excluded docs.	6.76	21.76	2.56	8.04	5.64	12.08	56.84

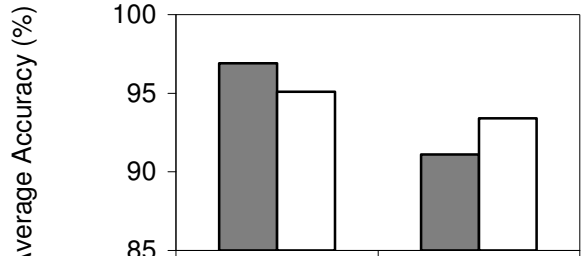
Group-average Hierarchical Agglomerative Clustering

	Document Topics						Total
	Coffee	*Crude*	*Gold*	*Interest*	*Sugar*	*Trade*	Docs.
Cluster-1	83	–	–	–	–	1	84
Cluster-2	1	226	–	–	1	1	229
Cluster-3	–	1	66	–	–	–	67
Cluster-4	3	12	–	186	–	11	212
Cluster-5	–	–	–	–	94	–	94
Cluster-6	–	10	3	2	–	233	248
#Excluded docs.	3	4	1	2	2	5	17

Table 5.9: The confusion matrices of clusters generated by HOMOGEN and Group-average HAC methods.

In this respect and on the data set used in the experiments, *Single-link*, *Centroid* and *Median* methods appear to be very sensitive to irrelevant features. *Complete-link* is the most insensitive method, while the others including HOMOGEN are somewhat in the middle.

The detail results of HOMOGEN and *Group-average* algorithms are provided by Table 5.9. The fractional numbers in HOMOGEN are due to the averaging of the experiment results over 25 trials. Let *precision* be the percentage of correct



	Precision	Recall
HOMOGEN	96.9	91.1
GA-HAC	95.1	93.4

Table 5.10: The precision and recall of HOMOGEN and Group-average (GA)-HAC.

assignment of documents in all found clusters and *recall* be the percentage of correct assignment over all 951 documents, i.e., *micro-average precision* or *recall* (Yang *et al.*, 2000). The precision and recall of HOMOGEN are 96.9% and 91.1%, respectively (see Table 5.10). The group-average algorithm, on the other hand, produces clusters with slightly lower precision (95.1%) but higher recall (93.4%).

5.5 Discussion of Related Work

Previous work has mitigated the effect of input ordering by applying restructuring operators such as cluster *merging*, *splitting*, and *promotion* (Fisher, 1987). The strategies for applying these operators can be broadly divided into *local* and *global* approaches with their advantages and shortcomings. The local approaches apply restructuring operators on the neighborhood of a hosting node (i.e., a node that serves as the parent of a new observation). Systems such as COBWEB (Fisher, 1987),

UNIMEM (Lebowitz, 1987) and INC (Hadzikadic & Yun, 1989} are examples of those employing these restructuring strategies. Fisher's COBWEB selects and applies a restructuring operator that locally maximizes the measure of partition utility value. Lebowit's UNIMEM employs a somewhat less informal method for deciding between restructuring operators, which is based on a *confidence* score and a set of user-specified parameters. Likewise, the restructuring strategy in INC is largely heuristic, basing its decision for applying various restructuring operators on user-defined thresholds over *relevance* and *strength* measures. Although relatively efficient to recover nodes misplaced at neighboring nodes, the local approaches in general suffer from their inability to deal with major structural changes.

The global approaches address the sensitivity issue by iteratively reinserting nodes into the entire hierarchy. As an extreme example, ITERATE redistributes observations on a single-level clustering, which is initially built non-incrementally, until there is no cluster formation change in two consecutive iterations (Biswas, Weinberg, & Fisher, 1998). ITERATE's optimization technique is clearly very expensive. Alternatively, Fisher proposes a hierarchical redistribution method that intermittently performs nodes redistribution on an existing, incrementally built concept hierarchy (Fisher, 1996). The technique represents a hybrid approach that combines incremental and batch methods. It is less expensive but relearning all nodes iteratively makes the algorithm less incremental.

The restructuring strategy in HOMOGEN represents a tradeoff between the local and the global approaches. The system pinpoints nodes whose structures are

potentially affected by the presence of new observations and then applies restructuring operators only to nodes that actually experience structural change. The structural change problems are detected through checking the nodes' conformity with the *homogeneity* and *monotonicity* properties. Intuitively, this strategy improves the ability of the system to recover from major structural changes while preserving the incremental nature of the algorithm.

HOMOGEN's approach that uses a set of conceptual constraints (e.g., the *homogeneity* and *monotonicity* properties) as the guiding principles during the hierarchy restructuring can be related to the ARACHNE (McKusick & Langley, 1991) and the HIERARCH (Nevins, 1995) systems. ARACHNE constructs well-formed concept hierarchies with regard to explicit constraints on the tree structure. A well-organized concept tree in the system is defined as the one that has horizontally and vertically well-placed concepts with respect to a similarity metric. The system applies restructuring operators recursively at neighboring concepts until the two constraints are satisfied locally. Alternatively, the HIERARCH system uses information theoretic considerations to constrain the placement of a node in a hierarchy (Nevins, 1995). The system redistributes any node that violates the constraints as if it is a new object to learn, repeatedly, until every single node in the tree satisfies the imposed constraints. Besides the similarity of ARACHNE's restructuring process to the local approaches, its control structure is not guaranteed to halt theoretically (McKusick & Langley, 1991). Although in lesser extents, the HIERARCH's restructuring strategy can be related to the global approaches.

Unlike the ARACHNE and the HIERARCH systems that rely exclusively on their constraints as the only guiding principles (i.e., both systems apply restructuring operators for the sake of satisfying the given constraints), HOMOGEN also explicitly detects and rectifies structural problems that cannot be recovered by satisfying the imposed constraints. The premise is that no single approach covers all cases, and a complementary approach that addresses a different restructuring objective can be implanted to handle the uncovered cases. Although differing greatly in detail, this idea is similar in spirit to COP-COBWEB (Wagstaff & Cardie, 2000) and COP-KMEANS (Wagstaff, Cardie, Rogers, & Schroedl, 2001), a version of COBWEB (KMEANS) that enforces instance-level hard constraints irrespective to the clustering decision of the main approaches. The instance-level constraints in these systems are heavily dependent on the input domains so that a different set of hard constraints needs to be defined on a different data set. In contrast, HOMOGEN is more general because it deals only with a structural property, allowing it to work across data sets without additional efforts.

Finally, the homogeneity property in HOMOGEN is based on a notion of density. Several density-based algorithms with various notions of density have also been developed mostly for batch clustering methods. For example, a density is determined by the number of neighboring points at a specified radius (Ester *et al.*, 1996), a mathematical model (Hinneburg & Keim, 1998), or the number of points lying inside a cell grid (Agrawal *et al.*, 1998). The foundation of HOMOGEN's notion of density as described earlier is a graph theoretic approach (Jain & Dubes, 1998).

5.6 Summary

The central role of concept hierarchy in the architecture of FEILDS described in Chapter IV makes its construction a critical process. One of the main contributions of this chapter is the description of new concept formation algorithm that exploits the *homogeneity* property coupled with the monotonicity property for incremental induction of hierarchical concepts and clusters from a data stream. Both properties are essential for discovering intrinsic hierarchical structures in which one cannot assume about the shape and the class distribution of clusters.

The other main contribution is providing a comprehensive, in depth empirical evaluation on the performance of the algorithm. It has been experimentally shown that the homogeneity and monotonicity are indeed desirable properties in that improving the hierarchy in satisfying these properties leads to producing a better measure of hierarchy quality that is independent of the hierarchical clustering objectives. Experiments conducted on a variety of domains involving structured and unstructured data sets also indicate the effectiveness of HOMOGEN. The system is relatively insensitive to input ordering and can produce a quality hierarchy structure inherent within the input data. Its performance in the given unstructured data set is also comparable to the best performance achieved by HAC methods.

CHAPTER VI

EVALUATION OF FEILDS

FEILDS, as discussed in Chapter IV, is a computational framework for extending the capability of an existing concept drift learner to deal with variable drift rates. Its main role is to convert a stream of sparsely labeled data (with a rapid drift rate) into one with a slower drift rate that can be conveniently tracked by the learner. This chapter presents an empirical evaluation of FEILDS. The main evaluation objective is to observe the extent to which the performance of the existing concept drift learners can be improved by learning from the stream generated by FEILDS with respect to its performance as a result of learning from the original labeled data stream.

The first section of this chapter describes the experiment data and procedure. Section 6.2 describes three tracking tasks to be used in the experiments. These tracking tasks are modified from those described in Chapter III in order to suit the need of FEILDS's input. The discussion of primary experiments is provided in Section 6.3, followed by the discussions of empirical system behaviors in Sections 6.4, 6.5 and 6.6. Finally, Section 6.7 describes the summary of this chapter.

6.1 Data and Experiment Procedure

All experiments use the same data set as the one employed in the experiments presented in Chapter III. In the rest of this chapter, any experiment reference intended

for those described in Chapter III will be called *previous experiment*. The experiment described in this chapter is denoted by "*current*" *experiment*. The size of test set is 2581, taken from the test set of the *ModApte* split in the Reuters-21578 1.0 collection (exactly the same test set as used in previous experiments). The training set in previous experiments is further split in current experiments into a validation set of size 100 and a training set of size 6352 documents. The validation set is used to empirically determine the concept density threshold for identifying distinct concepts (see Chapter IV Section 4.3.4). Unless mentioned otherwise, all experiments are produced by setting the threshold parameter k in Equation 4.2 to its default value (i.e., k=0.5 or $\theta_{0.5}$). Recall that this default setting maximizes the margins between overfitting and overgeneralization (see Section 4.3.4 in Chapter IV). The experiment uses the training set to generate data streams to be learned by the system.

To observe the performance over time, the data stream is divided into k m-instance sequences. The system performance is measured on the same test set after learning an m-instance sequence. As defined before, the sequence of learning m-instance sequence that is followed for system performance measurement constitutes a *tracking cycle*. Unlike previous experiments in that the system learns only from labeled data, FEILDS in current experiments also allows learning from unlabeled data. Figure 6.1 summarizes the procedure employed in current experiments, which is slightly modified from the summary of FEILDS's approach in Figure 4.3. It ties together various system components and accommodates both the incremental and batch processes needed in the framework.

Input: a data stream ***Stream-*$_{\mathbf{S}}$** generated from the training set.

Initialization:

1. Let ***Stream-*$_{\mathbf{L}}$** = ⟨∅⟩, the sequence of labeled instances.
2. $H = \emptyset$, the concept hierarchy.
3. Determine the density threshold of distinct concepts from the validation set.

Experiment Procedure:

For each *tracking cycle* $i = \{1 \ldots k\}$

1. Process incrementally the i^{th} m-instance sequence from ***Stream-*$_{\mathbf{S}}$**.
   For each instance x from the m-instance sequence
      Update H to incorporate x using incremental concept formation algorithm described in Chapter V.
      If the label q of x is available
         then concatenate ⟨(x, q)⟩ to the tail of ***Stream-*$_{\mathbf{L}}$**.
2. Execute the *concept drift tracker* (CDT) algorithm, described in Chapter IV, to generate the new stream ***Stream-*$_{\mathbf{S}}'$** from current values of ***Stream-*$_{\mathbf{L}}$** and H.
3. Run a selected concept drift learner (e.g., one of the four algorithms described in Chapter III) to learn ***Stream-*$_{\mathbf{S}}'$** and measure the accuracy of the learned concepts on the test set.

Figure 6.1: The procedure of experiment for FEILDS evaluation.

***Stream-*$_{\mathbf{S}}$** in Figure 6.1 denotes a stream of labeled and unlabeled data fed to the system. ***Stream-*$_{\mathbf{L}}$** represents a stream of labeled data extracted from ***Stream-*$_{\mathbf{S}}$**, preserving the relative ordering of labeled data in ***Stream-*$_{\mathbf{S}}$**. Initially empty, the length of ***Stream-*$_{\mathbf{L}}$** grows incrementally when the system sees a labeled instance from ***Stream-*$_{\mathbf{S}}$** (see Step 1 in Figure 6.1). ***Stream-*$_{\mathbf{S}}'$** is the new stream generated by the

concept drift tracker (CDT) component, which contains genuine and artificially labeled data.

During the initialization stage, the procedure empirically determines the concept density threshold from a validation set. This threshold, as described in Chapter IV Section 4.3.4, is used for instance generalization through the concept hierarchy, a process needed by the concept drift tracker algorithm. Initially empty, the concept hierarchy H is updated incrementally when observing each new instance from $\boldsymbol{Stream\text{-}_S}$ regardless of whether the instance is labeled or not. Next, it invokes the CDT algorithm after observing m-instance sequence in order to generate $\boldsymbol{Stream\text{-}_{S'}}$ based on the current value of $\boldsymbol{Stream\text{-}_L}$ and the concept hierarchy built up to that point. An existing concept drift learner is then applied in Step 3 to learn $\boldsymbol{Stream\text{-}_{S'}}$. The system performance is measured in the same way as in previous experiments based on the performance of a selected concept drift learner on a separate test set.

Four concept drift learners are considered for learning the stream $\boldsymbol{Stream\text{-}_{S'}}$: (1) MTDR algorithm, (2) Rocchio algorithm, (3) Window- KNN, and (4) Window-Rocchio. These algorithms have been used for performance comparison against one another in previous experiments, and that is not the case in current experiments. The main idea of FEILDS is to extend an existing concept drift learner for dealing with a few labeled data stream. Therefore, the performance of an existing concept drift learner is expected to improve by learning $\boldsymbol{Stream\text{-}_{S'}}$ over the performance of those that learn only $\boldsymbol{Stream\text{-}_L}$ (i.e., the original labeled data stream) regardless of the

concept drift learner employed. The four concept drift learners with diverse methods above will be used to confirm this expectation.

6.2 Tracking Tasks

The experiments employ similar tracking tasks to the ones summarized in Table 3.6 in terms of the sequence of target concept classes that need to be tracked over time. The evaluation is focused on the system performance when the shortest possible sequence of labeled data, as reflected by ***Stream-***$_{\mathbf{L}}$, is presented. This sequence corresponds to the 5% of labeled instances used in previous experiments.

If data streams used in previous experiments are employed in current experiments, and if only 5% labeled instances in the streams are made available, the rest 95% of the data in the streams can actually serve as unlabeled data that can be utilized by FEILDS. Although still a valid method, the streams contain only a small number of concepts and the portions of instances belong to current target concepts (i.e., relevant unlabeled data) are still relatively high. To make the problem more challenging, current experiments extend the original data streams so that they contain mostly non-target instances while preserving the relative ordering of instances in the original streams.

Tables 6.1−6.5 describe tracking tasks 1-E(xtended) − 5-E; these have been extended from tracking tasks 1−5 used in previous experiments (see Tables 3.1−3.5). While the number of instances at each tracking cycle in the original tracking tasks varies from 1 to 4, the extended tracking tasks contain the same 10-instance sequence

Tracking Cycle				
1 – 20	21 – 40	41 – 60	61 – 80	81 – 100
(*Trade*, +) & 9 others	(*Trade*, –) (*Coffee*, +) & 8 others	(*Coffee*, –) (*Crude*, +) & 8 others	(*Crude*, –) (*Sugar*, +) & 8 others	(*Sugar*, –) (*Acq*, +) & 8 others

Table 6.1: Tracking task 1-E(xtended).

Tracking Cycle			
1 – 20	21 – 40	41 – 60	61 – 80
(*Trade*, +) (*Coffee*, +) & 8 others	(*Trade*, –) (*Coffee*, +) (*Crude*, +) & 7 others	(*Coffee*, –) (*Crude*, +) (*Sugar*, +) & 7 others	(*Crude*, –) (*Sugar*, +) (*Acq*, +) & 7 others

Table 6.2: Tracking task 2-E

Tracking Cycle		
1 – 20	21 – 40	41 – 60
(*Trade*, +) (*Coffee*, +) (*Crude*, +) & 7 others	(*Trade*, –) (*Coffee*, +) (*Crude*, +) (*Sugar*, +) & 6 others	(*Coffee*, –) (*Crude*, +) (*Sugar*, +) (*Acq*, +) & 6 others

Table 6.3: Tracking task 3-E.

per tracking cycle. As a result, the lengths of the data streams in tracking tasks 1-E – 5-E are 1000, 800, 600, 400 and 400, respectively. In contrast, the original stream lengths are 180 for tracking task 1, 220 for tracking tasks 2 & 3, and 40 for tracking tasks 4 & 5. The additional instances in the extended tracking tasks, which are

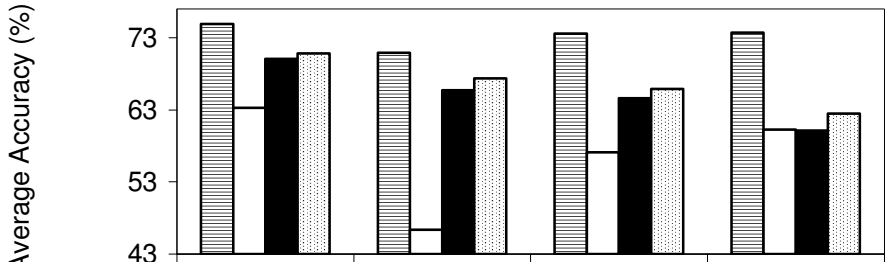
Tracking Cycle	
1 – 20	21 – 40
(*Trade*, +) (*Coffee*, +) & 8 others	(*Coffee*, −) (*Crude*, +) & 8 others
(*m*=2)	(*m*=2)

Table 6.4: Tracking task 4-E.

Tracking Cycle	
1 – 20	21 – 40
(*Trade*, +) (*Coffee*, +) (*Crude*, +) & 8 others	(*Coffee*, −) (Crude, +) (*Sugar*, +) & 8 others
(*m*=3)	(*m*=3)

Table 6.5: Tracking task 5-E.

randomly selected from the training set, belong to non-target concepts so that the portions of non-target instances in tracking tasks 1-E – 5-E are 90%, 80%, 70%, 80% and 70%, respectively. It is worth mentioning that the definition of 5% labeled data in the original streams (e.g., tracking tasks 1–5) corresponds to roughly from 0.9% – 1.1% in tracking tasks 1-E – 5-E. The portions of labeled instances in the extended tracking tasks with respect to instances in the original streams remain the same (5%).



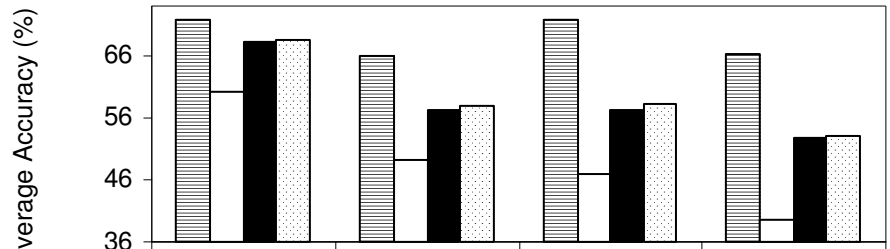
	MTDR	Rocchio	Window-KNN	Window-Rocchio
100%-L	74.90	70.92	73.58	73.71
5%-L	63.29	46.35	57.09	60.26
FEILDS (5%)	70.06	65.71	64.61	60.12
FEILDS (10%)	70.81	67.36	65.90	62.46

Table 6.6: System performances on tracking task 1-E.

6.3 Primary Experiment Results

This section describes the main experiment results that demonstrate the utility of FEILDS. Tables 6.6 – 6.8 summarize the outcomes of previous and current experiments on tracking tasks 1-E – 3-E, respectively. The system performances, as shown in the figures, are the average accuracies from the first tracking cycle to the end, averaged over ten trials (from running ten data streams).

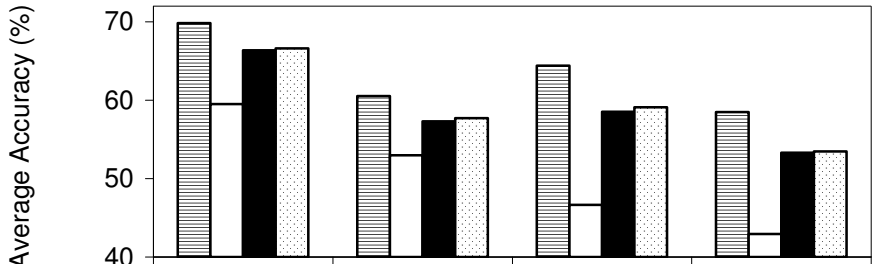
The "100%-L" performances are simply taken from Table 3.8. These results are generated in previous experiments by making the labels of all instances in the original streams available to the concept drift learner. In contrast, the "5%-L" average



	MTDR	Rocchio	Window-KNN	Window-Rocchio
▤ 100%-L	71.80	66.06	71.77	66.40
□ 5%-L	60.12	49.25	46.80	39.49
■ FEILDS (5%)	68.17	57.37	57.36	52.61
▨ FEILDS (10%)	68.46	57.96	58.32	52.99

Table 6.7: System performances on tracking task 2-E.

accuracies, which also serve as the baselines, describe the system performances when given only 5% of labeled data (e.g., ***Stream-*$_{\mathbf{L}}$**) with respect to the number of labeled data used to produce the "100%-L" performances. The "5%-L" performances are obtained from Table 3.10 in previous experiments. The FEILDS rows show the concept drift learner's performances from learning ***Stream-*$_{\mathbf{S}}$′**. As described above, ***Stream-*$_{\mathbf{S}}$′** is the stream generated by the CDT component in the FEILDS architecture, which is also given the same ***Stream-*$_{\mathbf{L}}$** as one of its inputs (its other input is the unlabeled data in ***Stream-*$_{\mathbf{S}}$**). The FEILDS rows provide the main results of current experiments.



	MTDR	Rocchio	Window-KNN	Window-Rocchio
100%-L	69.85	60.54	64.41	58.48
5%-L	59.53	53.00	46.66	42.93
FEILDS (5%)	66.38	57.32	58.55	53.32
FEILDS (10%)	66.63	57.70	59.11	53.48

Table 6.8: System performances on tracking task 3-E.

The difference between "5%-L" and "100%-L" performances represents a room for improvement, the extent to which the "5%-L" performances can be improved by FEILDS; although desirable, it is not realistic to expect that its performance would exceed that of the "100%-L" system. As shown in Tables 6.6 – 6.8, FEILDS can effectively improve the average accuracies of existing concept drift learning algorithms except for the results of FEILDS (5%) employing Window-Rocchio learner on tracking task 1-E. It is worth noting that all the four learning algorithms receive the same ***Stream-s′*** at a given tracking cycle and a tracking task. Therefore, the failure of the Window-Rocchio learner for improving its performance as above is more likely due to the problem within the algorithm itself rather than the quality of the ***Stream-s′***; the other three algorithms do not encounter this problem.

Tables 6.6 – 6.8 also show that the performances of FEILDS can be further improved when given streams with 10% labeled data (again, with respect to the "5%-L" performances). The improvement over FEILDS (5%), however, is not significant. It is likely that most additional labeled data of the same concept category is classified on the same concept node in the concept hierarchy, yielding no additional information.

6.4 Performance over Time

Figures 6.2 – 6.4 depict the MTDR algorithm performances over time on tracking tasks 1-E – 3-E, respectively. Clearly, the FEILDS performances improve over the baseline ("5%-L") performances. Using the same sequence of labeled data as that given to the "5%-L" systems, FEILDS gains its performances as more relevant instances became available, which is expected. Except in the last twenty tracking cycles whose current target topics involve *Acq*, most of the performance gains achieved over the baseline performances are significant, and in some cases are even better than the performances of the "100%-L" systems. In the experiment setting, it is found that *Acq* is the most difficult target concept to learn, causing a drastic performance drop when the systems start to track this target concept. Nonetheless, FEILDS is still able to improve its performances automatically, although rather slower, with the increasing availability of *Acq* documents. This tendency is very encouraging.

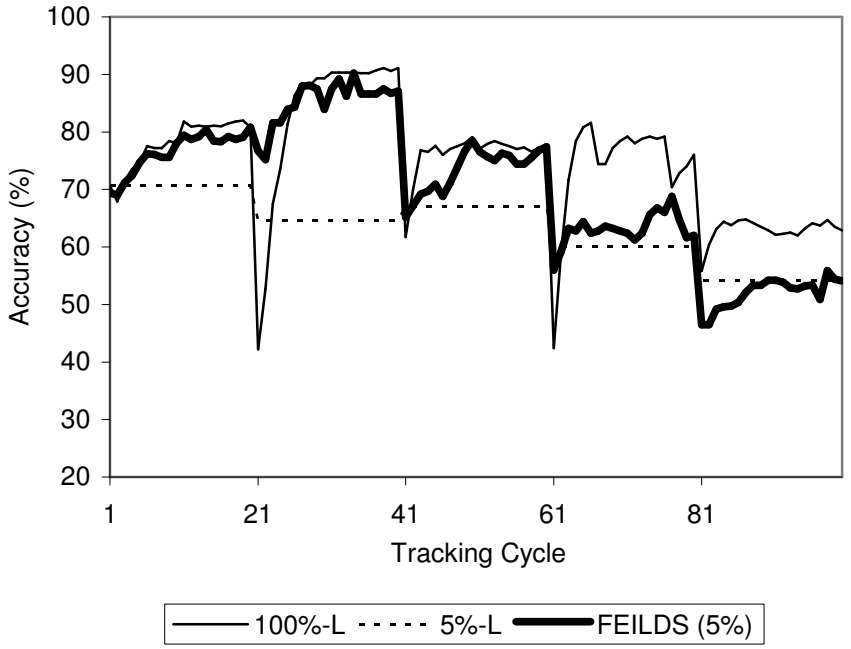


Figure 6.2: The MTDR algorithm performance over time on tracking task 1-E.

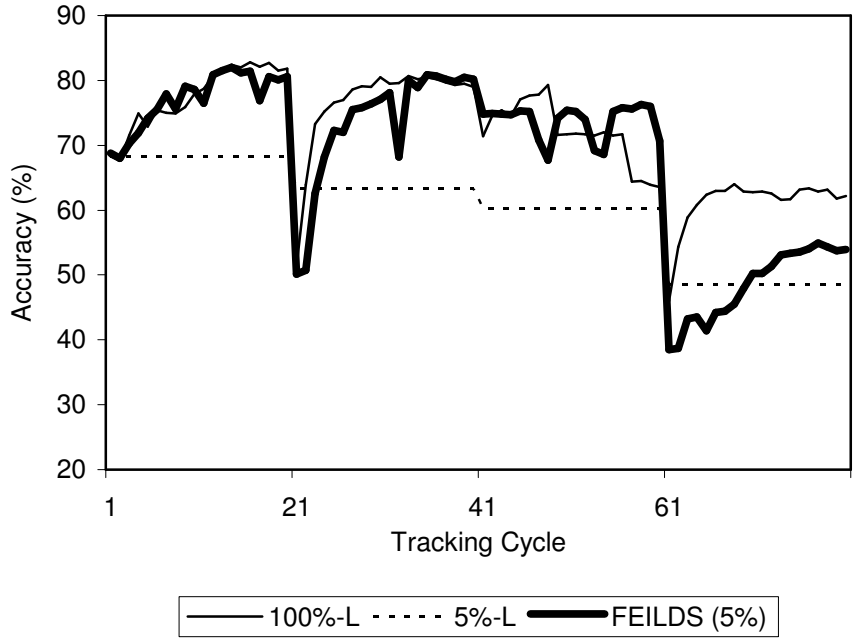


Figure 6.3: The MTDR algorithm performance over time on tracking task 2-E

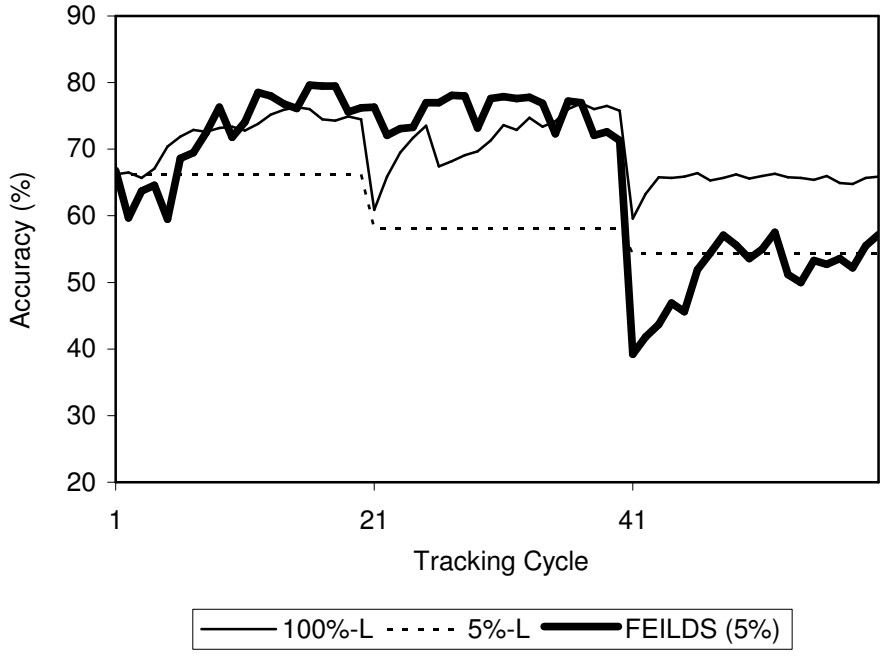
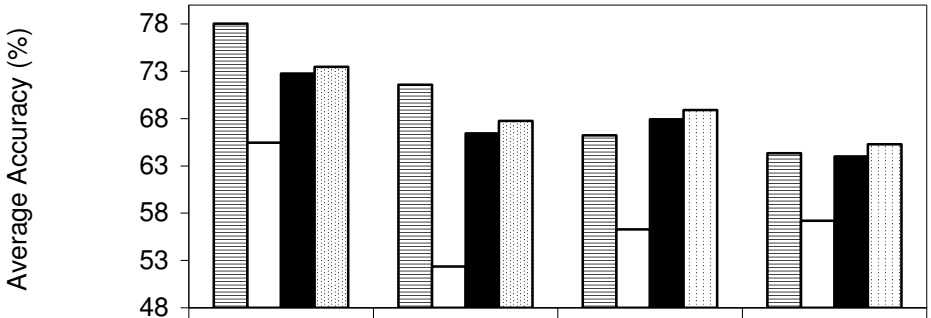


Figure 6.4: The MTDR algorithm performance over time on tracking task 3-E.

As discussed in Section 4.3, the quality of the concept (cluster) hierarchy as well as the accuracy of the instance generalization method could affect the quality of the system's output. Section 6.4 shows that although FEILDS is able to retrieve more relevant unlabeled data, which can improve its performance, some of the unlabeled data retrieved are irrelevant or incorrectly labeled, which degrades system's performance. It is likely that this noise prevents FEILDS's performance from being better than the performance of "100%-L" system on tracking tasks 1-E – 3-E.

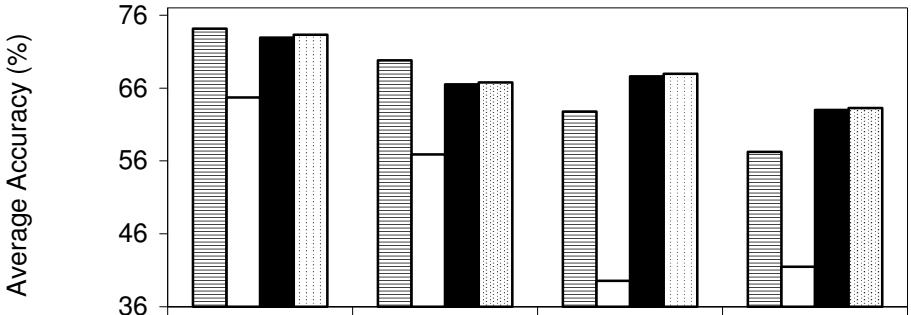


	MTDR	Rocchio	Window-KNN	Window-Rocchio
100%-L	78.04	71.57	66.22	64.35
5%-L	65.45	52.35	56.29	57.20
FEILDS (5%)	72.75	66.44	67.91	64.00
FEILDS (10%)	73.46	67.74	68.91	65.28

Table 6.9: System performances on tracking task 4-E. The differences of means between 100%-L and FEILDS (5% & 10%) in the Window-Rocchio, as well as between 100%-L and FEILDS 5% in the Window-KNN are not statistically significant (measured using the paired two-tailed *t* test).

6.5 Overcoming the Persistence Assumption Problem

Tables 6.9 and 6.10 summarize the performance of the four algorithms on tracking tasks 4-E and 5-E, respectively. Like in tracking tasks 1-E – 3-E, the performance of the MTDR and Rocchio algorithms significantly improves over the baseline ("5%-L") performance but cannot surpass the performances of "100%-L" systems. Deviating from these typical results, interestingly, the performance of the Window-KNN and Window-Rocchio algorithms is at least comparable (in tracking task 4-E) to and is even better (in tracking task 5-E) than that of the "100%-L" system.



	MTDR	Rocchio	Window-KNN	Window-Rocchio
100%-L	74.13	69.81	62.78	57.25
5%-L	64.69	56.90	39.55	41.50
FEILDS (5%)	72.92	66.53	67.61	63.00
FEILDS (10%)	73.33	66.77	67.95	63.26

Table 6.10: System performances on tracking task 5-E.

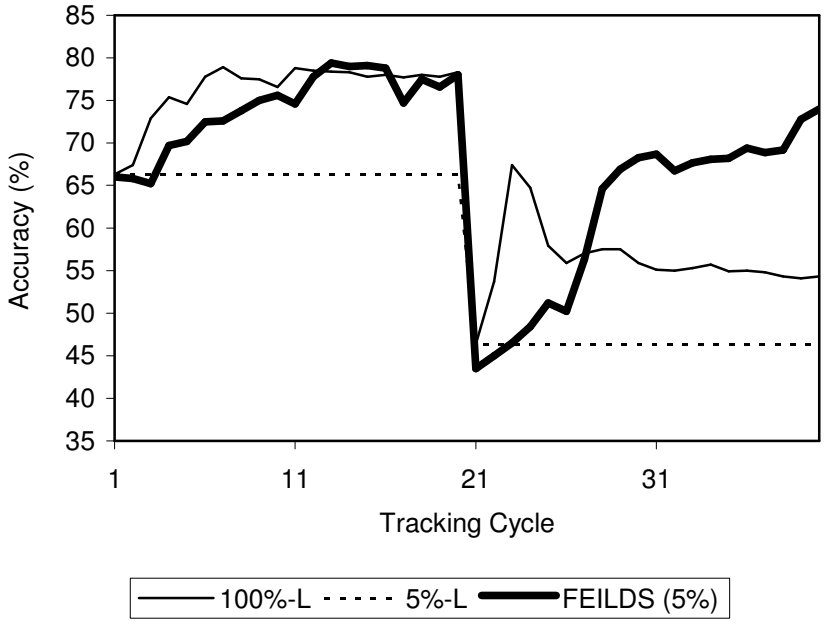


Figure 6.5: The Window-KNN algorithm performance over time on tracking task 4-E.

Figure 6.5 depicts the performance over time provided by the Window-KNN algorithm on tracking task 4-E. Recall that tracking tasks 4(-E) and 5(-E) require the persistence assumption in order to properly track the tasks because these tasks have to track the *Trade*, long-live, topic. As has been discussed in Chapter III, the "100%-L" system based on the window-KNN algorithm is not able to retain relevant older examples (e.g., *Trade* documents given during the first twenty tracking cycles) when a concept change occurs at the 21st tracking cycle, stumbling the system performance during the rest of the tracking cycles. FEILDS as shown in Figure 6.5 can avoid this problem. The accuracy of the Window-KNN algorithm improves over time after the concept change transition.

The "100%-L" performances provided by the Window-KNN and Window-Rocchio algorithms on tracking tasks 4-E and 5-E obviously suffer from being not in conformity with the persistence assumption needed to track these tasks. However, the problem can be addressed by learning from the stream generated by FEILDS. In addition to dealing with fewer labeled data, FEILDS generates a new stream that complies with the persistence assumption. Specifically, the new stream explicitly retains the older relevant examples, allowing the Window-KNN and Window-Rocchio algorithms to learn with better accuracies. This explains why the algorithm can achieve performances of at least comparable to the performances achieved by the "100%-L" systems.

6.6 The Sensitivity of Threshold in Instance Generalization Method

This section explores the sensitivity of threshold for recognizing the concept category of an instance and its impact on the system's performance. To do this, the experiments are re-run by varying the threshold values other than the default setting. The first part of this section examines the quality of the expanded data set $\mathbf{S}'$ produced by the CDT component of the system. The second part shows the system's average accuracies on the test set after learning the stream ***Stream-S′***.

As described in Chapter IV Section 4.3.3, a threshold that selects too specific or too general a concept node could introduce noise, and affect the coverage of the target instances retrieved. The quality of the set $\mathbf{S}'$ is thus expressed in terms of *noise* and *coverage*. The former denotes the percentage of instances in $\mathbf{S}'$ that are incorrectly labeled, while the latter refers to the percentage of target instances in $\mathbf{S}'$ over all target instances currently maintained in the concept hierarchy. These two measures are calculated cumulatively from the first tracking task to the end over ten trials. Specifically, let $\mathbf{S}'_{\mathbf{i,j}}$ be the set of expanded instances generated at the j^{th} tracking cycle during the i^{th} trial, and let $e_{i,j}$ be the number of instances in $\mathbf{S}'_{\mathbf{i,j}}$ that is incorrectly labeled. The noise is calculated as follows:

$$Noise = \frac{\sum_{i,j} e_{i,j}}{|\mathbf{S}'_{i,j}|} \times 100\% \qquad (6.1)$$

	θ_0	$\theta_{0.5}$ (default)	$tf = \theta_{0.5} / \theta_0$
Tracking task 1-E	1.106	1.144	1.03
Tracking task 2-E	1.109	1.150	1.04
Tracking task 3-E	1.103	1.141	1.03

Table 6.11: Automatic threshold value selection.

Furthermore, let $c_{i,j}$ be the number of target instances that are correctly labeled in $\mathbf{S'_{i,j}}$, and $h_{i,j}$ be the number of target instances currently maintained in the concept hierarchy. The coverage measure is defined by:

$$Coverage = \frac{\sum_{i,j} c_{i,j}}{h_{i,j}} \times 100\% \tag{6.2}$$

Table 6.11 reports the absolute threshold values obtained empirically from the validation sets. As described in Section 4.3.4, θ_0 refers to the threshold value calculated by setting k in Equation 4.2 to 0, while $\theta_{0.5}$ is the default thresholding scheme (k=0.5). For readability, a thresholding factor $tf = \frac{\theta}{\theta_0}$ will be used to describe a relative threshold with respect to θ_0. For example, the last column of Table 6.11 provides the threshold factor of $\theta_{0.5}$ on each tracking cycle. Thus, $tf < 1$ (*resp.*

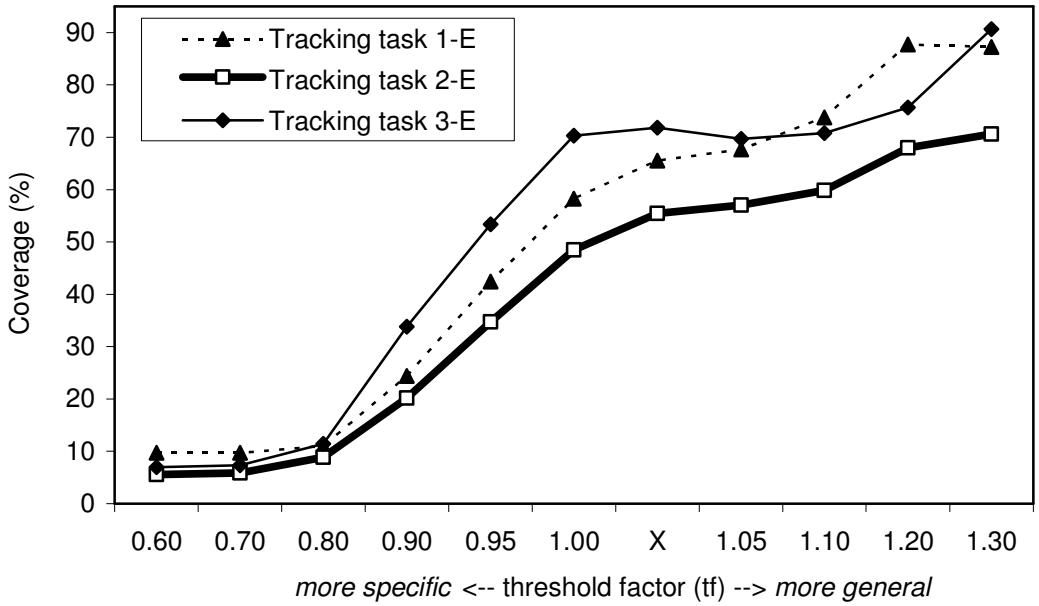


Figure 6.6: The effect of varying threshold values on the coverage of $\mathbf{S}'$.

$tf > 1$) represents a threshold value that would lead the instance generalization function to select a more specific (*resp*. more general) concept node.

Figures 6.6 depicts the coverage of the system's outputs (e.g., the expanded set $\mathbf{S}'$) over various threshold factors. The X threshold factors in the figure denote the values of threshold generated by the default setting, which, as described in Table 6.11, fall between threshold factors 1 and 1.05. On tracking tasks 1-E − 3-E, the coverage of the system's output increases as expected with the increased threshold factors, and the coverage of smaller thresholds converges to about 5 − 10%. It is likely that the coverage of 5% − 10% is obtained mainly from the labeled target data

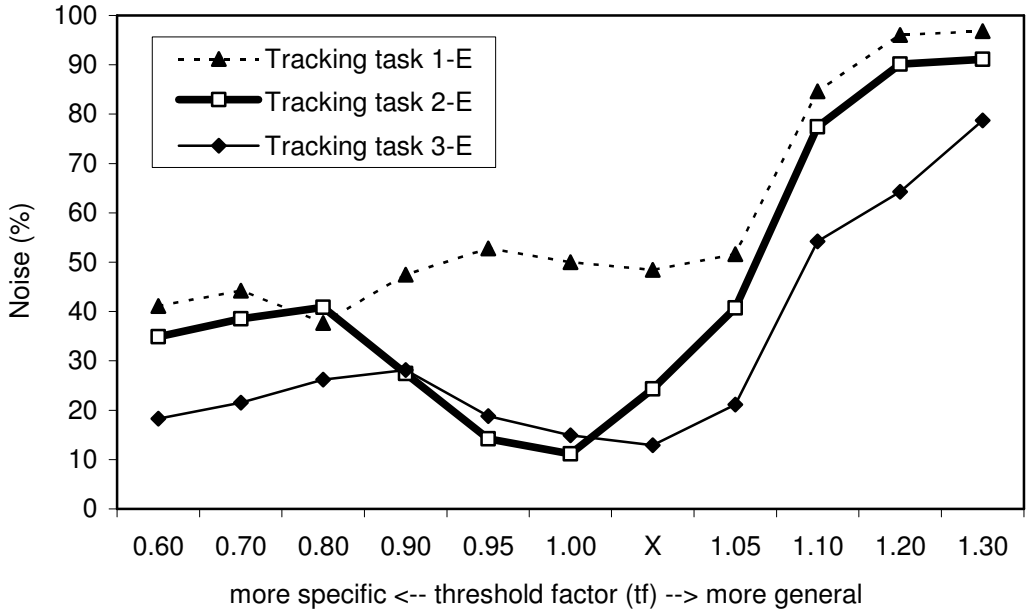


Figure 6.7: The effect of varying threshold values on the noise of $\mathbf{S}'$.

given to the system. Figure 6.6 also presents an interesting observation in that while the coverage of the default setting is relatively high, the slope of the curve around the default setting is relatively low.

The effect of threshold values on the noise of the system's output is depicted by Figure 6.7. Lower threshold factors ($tf < 0.9$) generate a relatively high noise but not as high noise as produced by higher threshold factors ($tf > 1.1$). Except on tracking task 1-E, the threshold factors within the range of 0.95 – 1.05 generate valleys that contain good tradeoffs between small noise and high coverage. It is not surprising that the default threshold value $\theta_{0.5}$ (the threshold factors X in Figure 6.7)

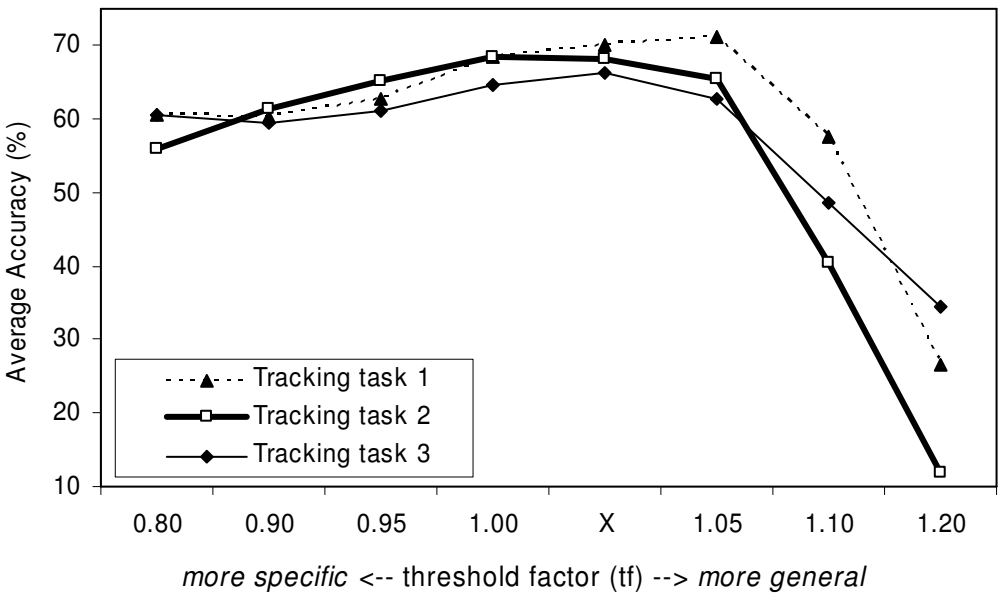


Figure 6.8: The effect of varying threshold values on the average accuracies of the MTDR algorithm.

is always in the range. Although high threshold yields high coverage, it also generates high percentage of noise whose negative effects might outweigh the benefit of having high coverage. Similarly, low threshold produces a relatively low noise but its low coverage might not help improve the system performance.

Figure 6.8 summarizes the performances of the MTDR algorithm over various threshold values. The average accuracies in the figure denote the algorithm performance on the test set, averaged over ten trials, after learning from the stream ***Stream-***$\mathbf{S}'$. As mentioned above, the threshold factor X also represents the results from using the default threshold parameter value ($\theta_{0.5}$). As shown in the figure, smaller

threshold values (lower than θ_0 or at tf=1) do not help improve the algorithm's performances because no new information can be provided, converging to the baseline average accuracies. However, higher threshold factors ($tf > 1.1$) are also prone to producing a detrimental effect that degrades the algorithm's average accuracies even much worse than the baselines. The default threshold setting improves the algorithm's performance over θ_0 ($tf = 1$) in 2 out of 3 cases. In addition, the default setting is still safe enough for not by accident selecting concept nodes that are too general.

6.7 Summary

This chapter empirically evaluates the utility of FEILDS. The emphasis of the evaluation is on observing its effectiveness in improving the performance of an existing algorithm for learning concept drift from a stream with sparsely labeled data. The experiments employ five tracking tasks in previous experiments that are further expanded to include many more irrelevant unlabeled data.

The main experiment results show that FEILDS is indeed able to extend the capability of concept drift learners, successfully improving most of their performances when learning with a very small amount of labeled data. This improvement is partly a result of the FEILDS's ability to take advantage of relevant unlabeled data as these become available over time. The experiment results also show that the improvement achieved by adding more labeled data is not significant, indicating that the performance as achieved by an existing concept drift learning

algorithm from learning with a complete labeled data is unlikely to be recovered. It further confirms the claim made in Chapter IV in that the number of labeled data to FEILDS is no longer relevant as long as its minimum quantity is already satisfied.

To sum up, current implementation of FEILDS is useful in the presence of minimal labeled data but could not effectively take advantage additional labeled data if provided.

CHAPTER VII

CONCLUSIONS

This dissertation has presented three major contributions: a concept drift learning algorithm for tracking multiple user interest categories, a general method for extending a concept drift learning algorithm to deal with a stream containing sparsely labeled data, and a concept formation algorithm for incremental construction of concept hierarchy. This chapter summarizes each contribution and discusses several extensions to the work.

7.1 Major Contributions

Algorithm for tracking multiple interest categories. The MTDR algorithm has been developed for learning the dynamics of tracking multiple interest categories under the assumption that a full set of examples is available for learning. The algorithm also satisfies the persistence assumption regarding the user interests, modifying the interest category representations only when explicitly told to do so from the relevance feedback examples. Conceptually, the algorithm extends the typical single window-based concept drift learning approaches by maintaining multiple window sets. Each set is used for deriving a distinct target concept, and is composed of large and small windows. The algorithm learns a target concept by combining the target concept representations from both large and small windows; this is a novel method. The MTDR algorithm is a realization of the above general method

with implicit windowing mechanism. It has been shown that the MTDR algorithm outperforms the Rocchio algorithm and the single window-based approaches particularly when tracking multiple target concepts simultaneously. The performances of all algorithms, however, are severely degraded when the number of labeled data is significantly reduced.

General Method for Extending Concept Drift Learning Algorithm. The strong assumption about the availability of training data has inspired the development of FEILDS, a general method for extending the capability of existing concept drift learning algorithms to deal with few labeled data. From the Computational Learning Theory perspective, the crux of the method is to convert a learning problem with rapid drift rate that is difficult to track into one with a slower drift rate, which is easier to learn by existing learners. The FEILDS architecture consists of three main entities: (1) a concept formation system (CFS), (2) a concept hierarchy, and (3) a concept drift tracker (CDT). The CFS component incrementally constructs a concept hierarchy from the input stream of labeled and unlabeled data in an unsupervised mode. Utilized mainly by the CDT component, the concept hierarchy serves as the knowledge base for the entire system. The CDT component analyzes the labeled data stream, removes any conflicting examples and then expands the remaining labeled data with relevant unlabeled data. The experimental results show the effectiveness of FEILDS, which greatly improves the performance of existing learners in learning from incomplete labeled data stream.

Concept Formation Algorithm. Since the concept hierarchy is a critical entity in the FEILDS architecture, its construction process deserves a careful treatment. This dissertation has developed a new concept formation algorithm so-called HOMOGEN. The new algorithm exploits the *homogeneity* and *monotonicity* properties for incremental induction of concept hierarchy from a data stream. Both properties are essential for discovering intrinsic hierarchical structures. Experiments conducted on natural, artificial and text documents data sets indicate the effectiveness of HOMOGEN. The system is relatively insensitive to input ordering and is able to produce a quality hierarchy structure inherent within the input data. Its performance in text document collection is also comparable to the best performance achieved by typical hierarchical agglomerative clustering methods. It is no wonder that this new algorithm highly contributes to the success of FEILDS.

7.2 Extensions to Current Works

The utility of the main idea behind the MTDR algorithm (e.g., multiple window sets, and combining large & small windows) has been shown in the information filtering domain. Its effectiveness in other domains will be an interesting investigation. Depending on the kind of concept representation that is most suitable for the domain, applying the algorithm in other domains could require some modifications. The most notable one is the definition of similarity between two concepts. The use of *cosine* coefficient in the current MTDR algorithm is due to the popularity and the effectiveness of this method for measuring the similarity of concepts in vector space

model. Another possible modification is an alternative method for combining two concept representations (i.e., those derived from examples in large and small windows). Currently applying a linear combination of feature weights, the method would need to be appropriately adjusted if the concept is represented as, for example, Boolean-valued features.

Several research issues regarding FEILDS's development are also worthy of further study. The first possibility is exploring alternative methods for generalizing an instance through the concept hierarchy. Current generalization method as described in this dissertation requires a modest effort for preparing the validation set in order to empirically determine the threshold value of generalization node. At one end of the spectrum in terms of effort, making the process fully automatic such as applying a heuristic would be the most desirable method. The most difficult problem with this approach is finding the appropriate heuristic rules; for example, how to practically and effectively define the notion of distinct concept. At the other end of the spectrum, one can use a manually crafted domain theory to guide the selection of the most appropriate concept in the concept hierarchy. Although promising when it involves a small number of concepts, this approach is not scalable for a much larger number of concepts particularly in the text domain where the variety of concept is virtually unlimited.

The second possibility is to improve the efficiency of HOMOGEN, the concept formation system currently employed by FEILDS. As described in Chapter V, the most time-consuming process with the current implementation is the reconstruction of

minimum spanning tree (MST) of objects that defines the concept density information. Applying incremental MST algorithm (Fredericson, 1985) would likely improve the efficiency of the concept hierarchy construction.

REFERENCES

Agrawal, R., Gehrke, J., Gunopulos, D., & Raghavan, P. (1998). Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the 1998 ACM-SIGMOD International Conference on Management of Data*, pp. 94–105. New York, NY: ACM Press.

Allan, J. (1996). Incremental relevance feedback for information filtering. In *Proceedings of the Nineteenth International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 270–278. New York, NY: ACM Press.

Allan, J., Papka, R., & Lavrenko, V. (1998). On-line new event detection and tracking. In *Proceedings of the Twenty-first International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 37–45. New York, NY: ACM Press.

Apté, C., Damerau, F., & Weiss, S. M. (1994). Automatic learning of decision rules for text categorization. *ACM Transactions on Information Systems*, *12*, 233–251.

Auer, P., & Warmuth, M. K. (1998). Tracking the best disjunction. *Machine Learning*, *32*, 127–150.

Balabanović, M. (1997). An adaptive web page recommendation service. In *Proceedings of the First International Conference on Autonomous Agents*, pp. 378–385. New York, NY: ACM Press.

Balabanović, M. (1998). *Learning to Surf: Multi-Agent Systems for Adaptive Web Page Recommendation*. Doctoral dissertation, Stanford University, Menlo Park, CA: Department of Computer Science.

Bartlett, P. L., David, S. B., & Kulkarni, S. R. (1996). Learning changing concepts by exploiting the structure of change. In *Proceedings of the Ninth Annual Workshop on Computational Learning Theory*, pp. 131–139. New York, NY: ACM Press.

Barve, R. D., & Long, P. M. (1997). On the complexity of learning from drifting distributions. *Information and Computation*, *138*, 170–193.

Billsus, D., & Pazzani, M. (1999). A personal news agent that talks, learns and explains. In *Proceedings of the Third International Conference on Autonomous Agents*, pp. 268–275. New York, NY: ACM Press.

Biswas, G., Weinberg, J., & Fisher, D. (1998). ITERATE: A conceptual clustering algorithm for data mining. *IEEE Transactions on Systems, Man, and Cybernetics*, *28*, 100–111.

Blake, C., & Merz, C. (1998). *UCI Repository of machine learning databases*. http://www.ics.uci.edu/_mlearn/MLRepository.html, University of California, Irvine, CA: Department of Information and Computer Science.

Blum, A., & Chalasani, P. (1992). Learning switching concepts. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 231–242. New York, NY: ACM Press.

Blum, A., & Chawla, S. (2001). Learning from labeled and unlabeled data using graph min-cuts. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 19–26. San Mateo, CA: Morgan Kaufmann.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pp. 92–100. New York, NY: ACM Press.

Blummer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1989). Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM*, *36*, 929–965.

Bousquet, O., & Warmuth, M. K. (2002). Tracking a small set of experts by mixing past posteriors. *Journal of Machine Learning Research*, *3*, 363–396.

Buckley, C., Salton, G., Allan, J., & Singhal, A. (1995). Automatic query expansion using SMART: TREC-3. In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, pp. 69–80. NIST Special Publication 500-225.

Chen, C. C., Chen, M. C., & Sun, Y. (2002). PVA: A self-adaptive personal view agent. In Special Issue on Automated Text Categorization, *Journal of Intelligent Information Systems*, *18*, 173–194.

Chen, L., & Sycara, K. (1998). WEBMATE: Personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents*, pp. 132–139. New York, NY: ACM Press.

Cormen, T. H., Leiserson, C. E., & Rivest, R. L. (2001). *Introduction to Algorithms*. Cambridge, MA: MIT Press.

Crouch, C. J., Crouch, D. B., Chen, Q., & Holtz, S. J. (2002). Improving the retrieval effectiveness of very short queries. *Information Processing and Management*, *38*, 1–36.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithms. *Journal of the Royal Statistical Society*, Series B., *39*, 1–38.

Dhillon, I. S., & Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. *Machine Learning*, *42*, 143–175.

Ester, M., Kriegel, H. P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231. Menlo Park, CA: AAAI Press.

Everitt, B. S., Landau, S., & Leese, M. (2001). *Cluster Analysis*. New York, NY: Oxford University Press Inc.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, *2*, 139–172.

Fisher, D. (1996). Iterative optimization and simplification of hierarchical clusterings. *Journal of Artificial Intelligence Research*, *4*, 147–180.

Fisher, D., Pazzani, M., & Langley, P. (1991). *Concept Formation: Knowledge and Experience in Unsupervised Learning*. San Mateo, CA: Morgan Kaufmann.

Fischer, G., & Stevens, C. (1991). Information access in complex, poorly structured information spaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 63–70. New York, NY: ACM Press.

Fisher, D. H., Xu, L., & Zard, N. (1992). Ordering effects in clustering. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 163−168. San Mateo, CA: Morgan Kaufmann.

Fowlkes, E. B., & Mallows, C. L. (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, *78*, 553–569.

Frederickson, G. (1985). Data structures for on-line updating of MST, with applications. *Siam Journal on Computing*, *14*, 781–798.

Gabbay, D. M, Hogger, C. J., & Robinson, J. A. (1995). *Handbook of Logic in Artificial Intelligence and Logic Programming: V4. Epistemic and Temporal Reasoning*. New York, NY: Oxford University Press.

Gennari, J., Langley, P., & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, *40*, 11–61.

Guha, S., Rastogi, R., & Shim, K. (1998). CURE: An efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM-SIGMOD International Conference on Management of Data,* pp. 73–84. New York, NY: ACM Press.

Hadzikadic, M., & Yun, D. (1989). Concept formation by incremental conceptual clustering. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 831–836. San Mateo, CA: Morgan Kaufmann.

Hanson, S. J., & Bauer, M. (1989). Conceptual clustering, categorization, and polymorphy. *Machine Learning, 3*, 343–372.

Harries, M.B., & Horn, K. (1998). Learning stable concepts in a changing world. In G. Antoniou, A. Ghose and M. Truczszinski (Eds.), *Lecture Notes on Artificial Intelligence 1359: Learning and Reasoning with Complex Representation*, pp. 106–122. Berlin; New York, NY: Springer-Verlag.

Harries, M. B., Sammut, C., & Horn, K. (1998). Extracting hidden context. *Machine Learning*, *32*, 101-128.

Helmbold, D. P., & Long, P. M. (1994). Tracking drifting concepts by minimizing disagreement. *Machine Learning*, *14*, 27–45.

Herbster, M., & Warmuth, M. K. (1998). Tracking the best expert. *Machine Learning, 32*, 151–178.

Hinneburg, H., & Keim, D. (1998). An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pp. 58–65. Menlo Park, CA: AAAI Press.

Hubert, L. J., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, *2*, 193–218.

Hull, D. A. (1998). The TREC-7 filtering track: Description and analysis. In *Proceedings of the Seventh Text Retrieval Conference (TREC-7),* pp. 33–56. NIST Special Publication 500-242.

Iwayama, M. (2000). Relevance feedback with a small number of relevance judgments: Incremental relevance feedback vs. document clustering. In *Proceedings of the Twenty-third ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pp. 10–16. New York, NY: ACM Press.

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall.

Jansen, B. J., Spink, A., & Saracevic, T. (2000). Real life, real users and real needs: A study and analysis of users queries on the web. *Information Processing and Management*, *36*, 207–227.

Jardine, N., & Sibson, R. (1971). *Mathematical Taxonomy*. New York, NY: John Wiley & Sons.

Joachims, T. (1997). A probabilistic analysis of the rocchio algorithm with tf-idf for text categorization. In *Proceedings of the Fourth International Conference on Machine Learning*, pp. 143–151. San Mateo, CA: Morgan Kaufmann.

Karypis, G., Han, E. H., & Kumar, V. (1999). CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *Computer*, *32*, 68–75.

Klinkenberg, R. (1999). Learning drifting concepts with partial user feedback. *Beiträge zum Treffen der GI-Fachgruppe 1.1.3 Maschinelles Lernen (FGML-99)*, Perner, Petra and Fink, Volkmar (ed.).

Klinkenberg, R. (2001). Using labeled and unlabeled data to learn drifting concepts. In *IJCAI-01 Workshop on Learning from Temporal and Spatial Data*. http://www-ai.cs.uni-dortmund.de/DOKUMENTE/klinkenberg_2001a.pdf.

Klinkenberg, R., & Joachims, T. (2000). Detecting concept drift with support vector machine. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 487–494. San Mateo, CA: Morgan Kaufmann.

Klinkenberg, R., & Renz, I. (1998). Adaptive information filtering: Learning in the presence of concept drifts. In *AAAI Workshop on Learning for Text Categorization*, pp. 33-40.

Kuh, A., Petsche, T., & Rivest, R.L. (1991). Learning time-varying concepts. *Advances in Neural Information Processing Systems*, *3*, 183–189.

Lam, W., Mukhopadhay, S., Mostafa, J., & Palakal, M. (1996). Detection of shifts in user interests for personalized information filtering. In *Proceedings of the Nineteenth ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pp. 317–325. New York, NY: ACM Press.

Lance, G. N., & Williams, W. T. (1967). A general theory of classificatory sorting strategies 1 hierarchical systems. *Computer Journal*, *9*, 373–380.

Lang, K. (1995). NEWSWEEDER: Learning to filter news. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 331–339. San Mateo, CA: Morgan Kaufmann.

Lebowitz, M. (1987). Experiments with incremental concept formation: UNIMEM. *Machine Learning*, *2*, 103–138.

Lewis, D. D., & Ringuette, M. (1994). A comparison of two learning algorithms for text categorization. In *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, pp. 81–93. Las Vegas, NV: University of Nevada, Information Science Research Institute.

Littlestone, N., & Warmuth, M. K. (1994). The weighted majority algorithm. *Information and Computation, 108*, 212–261.

Long, P. M. (1998). The complexity of learning according to two models of a drifting environment. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pp. 116–125. New York, NY: ACM Press.

McKusick, K. B., & Langley, P. (1991). Constraints on tree structure in concept formation. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 810– 816. San Mateo, CA: Morgan Kaufmann.

Mitchell, T.M. (1997). *Machine Learning*. New York, NY: McGraw-Hill.

Mitra, M., Singhal, A., & Buckley, C. (1998). Improving automatic query expansion. In *Proceedings of the Twenty-first ACM-SIGIR International Conference on Research and Development in Information Retrieval*, pp. 206–214. New York, NY: ACM Press.

Miyamoto, S. (1990). *Fuzzy Sets in Information Retrieval and Cluster Analysis*. Boston, MA: Kluwer Academic Publishers.

Moukas, A., & Zacharia, G. (1997). Evolving a multi-agent information filtering solution in AMALTHEA. In *Proceedings of the First International Conference on Autonomous Agents*, pp. 394–403. New York, NY: ACM Press.

Nevins, J. (1995). A branch and bound incremental conceptual clusterer. *Machine Learning*, *18*, 3–22.

Nigam, K., & Ghani, R. (2000). Analyzing the effectiveness and applicability of co-training. In *Proceedings* of *the Ninth International Conference on Information and Knowledge Management*, pp. 86–93. New York, NY: ACM Press.

Nigam, K., McCallum, A., Thrun, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning*, *39*, 103–134.

Pazzani, M., & Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, *27*, 313–331.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *44*, 846–850.

Rocchio, J. J. (1971). Relevance feedback in information retrieval. In G. Salton, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pp. 313–323. Englewood Cliffs, NJ: Prentice-Hall.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, *4*, 288–297.

Salton, G., & McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. New York, NY: McGraw-Hill.

Schlimmer, J. C., & Granger, R. H. (1986). Beyond incremental processing: Tracking concept drift. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 502–507. Menlo Park, CA: AAAI Press.

Seo, Y. W., & Zhang, B. T. (2000). Learning user's preferences by analyzing web-browsing behaviors. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pp. 381–387. New York, NY: ACM Press.

Sheth, B. D. (1993). *A Learning Approach to Personalized Information Filtering*. Master thesis, Massachusetts Institute of Technology, Cambridge, MA: Department of Electrical Engineering and Computer Science.

Tan, A., & Teo, C. (1998). Learning user profile for personalized information dissemination. In *Proceedings of International Joint Conference on Neural Network*, pp. 183–188. New York, NY: IEEE.

Theodoridis, S., & Koutroumbas, K. (1999). *Pattern Recognition*. San Diego, CA: Academic Press.

Wagstaff, K., & Cardie, C. (2000). Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 1103–1110. San Mateo, CA: Morgan Kaufmann.

Wagstaff, K., Cardie, C., Rogers, S., & Schroedl, S. (2001). Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 577–584. San Mateo, CA: Morgan Kaufmann.

Widmer, G. (1997). Tracking context changes through meta-learning. *Machine Learning*, *3*, 259–286.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning*, *23*, 69–101.

Widyantoro, D. H. (1999). *Modeling and Learning User Profile in Personalized News Agent*. Master thesis, Texas A&M University, College Station, TX: Department of Computer Science.

Widyantoro, D. H., Ioerger, T. R., & Yen, J. (1999). An adaptive algorithm for learning changes in user Interests. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, pp. 405–412. New York, NY: ACM Press.

Widyantoro, D. H., Ioerger, T. R., & Yen, J. (2001). Learning user interest dynamics with a three-descriptor representation. *Journal of the American Society for Information Science and Technology*, *52*, 212–225.

Widyantoro, D. H., Ioerger, T. R., & Yen, J. (2002). An incremental approach to building a cluster hierarchy. In *Proceedings of the Second IEEE International Conference on Data Mining*, pp. 705–708. New York, NY: IEEE.

Widyantoro, D. H., Ioerger, T. R., & Yen, J. (2003). Tracking changes in user interests with a few relevance judgments. In *Proceedings of the Twelfth ACM International Conference on Information and Knowledge Management* (CIKM-2003), pp. 548–551. New York, NY: ACM Press.

Widyantoro, D. H., Yin, J., Seif El-Nasr, M., Yang, L., Zacchi, A., & Yen, J. (1999). ALIPES: A swift messenger in cyberspace. In *Proceedings of the AAAI Spring'99 Symposium on Intelligent Agents in Cyberspace*, pp. 62–67. Menlo Park, CA: AAAI Press.

Witten, I. H., Moffat A., & Bell T. C. (1994). *Managing Gigabytes: Compressing and Indexing Documents and Images*. New York, NY: Van Nostrand Reinhold.

Xu, J., & Croft, W. B. (1996). Query expansion using local and global document analysis. In *Proceedings of the Nineteenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 4–11. New York, NY: ACM Press.

Yan, W. T., & Garcia-Molina, H. (1999). The SIFT information dissemination system. *ACM Transactions on Database Systems*, *24*, 529–565.

Yang, Y., Ault, T., Pierce, T., & Lattimer, C. (2000). Improving text categorization methods for event tracking. In *Proceedings of the Twenty-third International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 65–72. New York, NY: ACM Press.

Yang, Y., Carbonell, J. D., Brown, R. D., Pierce, T., Archibald, B. T., & Liu, X. (1999). Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems: Special Issue on Applications of Intelligent Information Retrieval*, *14*, 32–43.

Yang, Y., Pierce, T., & Carbonell, J. (1998). A study on retrospective and on-line event detection. In *Proceedings of the Twenty-first International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pp. 28-36. New York, NY: ACM Press.

Zahn, C. T. (1971). Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, *C 20*, 68–86.

Zhang, T., & Oles., F. J. (2000). A probability analysis on the value of unlabeled data for classification problems. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 1191–1198. San Mateo, CA: Morgan Kaufmann.

VITA

Dwi Hendratmo Widyantoro graduated cum laude from the Institut Teknologi Bandung, Indonesia, in 1991, obtaining his B.S. in Computer Science. After spending two years working in a software industry, he joined the Department of Informatics Engineering of the Institut Teknologi Bandung in 1993 as a teaching staff member. In 1997, Widyantoro began his graduate study in the Department of Computer Science at Texas A&M University and was awarded M.S. degree in Computer Science in 1999. During his attendance at Texas A&M University, he has published at least twelve conference and/or journal papers. His permanent address is at Jl. Puri Cipageran Indah I Blok A-36, Cimahi 50411 Indonesia.