

Relevant Data Expansion for Learning Concept Drift from Sparsely Labeled Data

Dwi H. Widyantoro
Department of Informatics Engineering
Institut Teknologi Bandung
Bandung 40132
INDONESIA
dwi@if.itb.ac.id

John Yen
School of Information Sciences and Technology
The Pennsylvania State University
University Park, PA 16802
U.S.A.
jyen@ist.psu.edu

ABSTRACT

Keeping track of changing interests is a natural phenomenon as well as an interesting tracking problem because interests can emerge and diminish at different time frames. Being able to do so with a few feedback examples poses an even more important and challenging problem because existing concept drift learning algorithms that handle the task typically suffer from it. This paper presents a new computational Framework for Extending Incomplete Labeled Data Stream (FEILDS), which extends the capability of existing algorithms for learning concept drift from a few labeled data. The system transforms the original input stream into a new stream that can be conveniently tracked by the existing learning algorithms. The experiment results reveal that FEILDS can significantly improve the performances of a Multiple Three-Descriptor Representation (MTDR) algorithm, Rocchio algorithm and window-based concept drift learning algorithms when learning from a sparsely labeled data stream, with respect to their performances without using FEILDS.

Keywords: concept learning, relevance feedback, information filtering.

I. INTRODUCTION

Being able to infer the most up-to-date user interests is of great importance because it can help select and deliver new relevant information to users timely. Users often do not know what kind of new interesting information that will become available and when. The information providers also have no any knowledge about the information needs of users. Information agents can fill in this gap, and the ability to automatically track the change of user interests over time plays a vital role in such agents.

Tracking the evolution of user interests is a problem instance of concept drift learning. The majority of existing concept drift learning algorithms typically require 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 [20][21][22][29][32][33][34][35]. Although unlabeled data are widely available in the information-filtering domain, acquiring labeled data in this domain is indeed still very prohibitive. For example, casual users tend to be unwilling to provide the relevance feedback needed to label the data [18]. It is not questionable that learning concept drift from a sequence of few labeled data poses an important problem.

In this paper we address the above issue, which represents the main paper contribution. We describe and evaluate FEILDS, a computational Framework for Extending Incomplete Labeled Data Stream. One of its inputs is the original labeled data stream, assumed to have a minimal number of labeled data, that would normally be the input to a concept drift learner. It generates a new data stream that has been expanded using relevant unlabeled data. The existing concept drift learner then learns from the new data stream rather than the original stream. Hence, FEILDS modularly extends the capability of existing concept drift learning algorithms by modifying their input without modifying any part of the algorithms. In the absence of labeled

data, FEILDS is able to improve the performance of existing algorithms as more relevant unlabeled data become available from the input stream.

The rest of this paper is organized as follows. Sections II and III describe the background and the representatives of existing concept drift learning algorithms, respectively. Next in Section IV, we present our approach to deal with sparsely labeled data for learning concept drift. Our approach is then evaluated in Section V. Finally, we provide a discussion of related works in Section VI, followed by conclusions in Section VII.

II. FOUNDATION

A. Stable Concept versus Concept Drift Learning

Concept learning is a process of inferring a Boolean-valued function from a set of input and output examples [26], i.e., $f: X \rightarrow \{1, 0\}$ where X is the input space. In the information-filtering domain, the input is a document d and the output is the document relevance (either *relevant* or *irrelevant*). Conventional (stable) concept learning assumes that the target function is static, that is, the relevance values of all documents with the same topic category are the same. The values of target function are often referred to as the data (concept) labels. The input and output examples technically can be learned in any order.

Concept drift learning is a concept learning in which the target function changes over time. For example, the relevance of documents of the same topic category could change from time to time because a user can change his/her interest. Hence, the target function 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 *drift rate* in concept drift learning denotes the probability that two successive target concepts disagree on a randomly drawn example [4], i.e., $\mathbf{Prob}(f_t \neq f_{t-1})$. Intuitively, lower drift rates correspond to learning from data streams whose target concepts change less frequently (more labeled data are available for learning the same target function before it changes), and vice versa. 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.

B. Basic Representation and Computation of Text Documents

Information-filtering problem mostly involves text documents as the source of data. A common representation of a text document is based on the vector space model in which a text document is described by a feature vector in n -dimensional space [37]. Let d be a text document, then $\{(t_1, w_1), (t_2, w_2), \dots, (t_n, w_n)\}$ is the feature vector (or descriptor) of d where t is a term (word) that occurs in d and w is the weight of term t .

Term weighting is concerned with the assignment of a value to a term that represents the degree of importance of the term in a document. *Term Frequency-Inverse Document Frequency (tf-idf)* is the most well studied weighting scheme. It is based on the assumption that terms that occur in fewer documents are better discriminators. Let tf_i be the frequency of occurrence of term t_i in a document d , and let df_i be the number of documents to which the term occurs (the corresponding document frequency). The importance of word i , denoted by w_i , is expressed as follows [30]:

$$w_i = \frac{tf_i}{\sum_j tf_j} \log \left(\frac{N}{df_i} \right) \quad (1)$$

where N is the number of documents in the collection and $\sum_j tf_j$ is the length of document. The document length is used to normalize term frequency in order to avoid favoring long documents over short documents.

A document similarity measure assesses the degree to which a document matches a reference feature vector. The most widely used similarity measure in the vector space model is the *cosine* formula [30]. It calculates the cosine of the angle between two feature vectors, irrespective of their length. The similarity between the feature vectors of two documents d_i and d_j according to the cosine formula is given by:

$$\text{sim}(d_i, d_j) = \cos \theta(d_i, d_j) = \frac{d_i \cdot d_j}{|d_i| |d_j|} \quad (2)$$

A set of documents is considered relevant to a topic category if the cosine similarity of the two in vector space representation is high (closer to one).

III. EXISTING CONCEPT DRIFT LEARNING ALGORITHMS

In this section we describe three representatives of existing concept drift learning algorithms. All algorithms are set up to work in an incremental setting within the information-filtering domain. The input of these algorithms is a labeled instance $\langle d, fb \rangle$, which is a pair of document d and its relevance feedback fb .

A. Rocchio Relevance Feedback Algorithm

Originally developed to improve the quality of retrieval results based on relevance feedback, Rocchio algorithm represents a classical approach for learning user interests. Given a set of relevant and irrelevant documents, a new query is reformulated from the query of preceding

retrieval request. Let $Q_t = \{(t_1, w_1), (t_2, w_2), \dots, (t_n, w_n)\}$ be a query at an iteration t . The relevance feedback process then generates the new query for the next retrieval iteration $Q_{t+1} = \{(t_1, w'_1), (t_2, w'_2), \dots, (t_n, w'_n)\}$ with altered weights w'_i .

The following is the original form of the Rocchio algorithm [29]:

$$Q_t = Q_{t-1} + \beta \frac{1}{n_{pos}} \sum_{pos} d_i - \gamma \frac{1}{n_{neg}} \sum_{neg} d_i \quad (3)$$

where $\beta + \gamma = 1$, n_{pos} is the number of relevant documents, and n_{neg} is the number of non-relevant documents. The role of parameters β and γ is to determine the amount of influence of relevant documents relative to irrelevant documents in query modification.

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 document d_{pos} or non-relevant document d_{neg} at a time, practically setting $n_{pos} = n_{neg} = 1$ in the Rocchio algorithm. The similarity of a document with the new query (which now represents a user's interest) is calculated using the *cosine* coefficient above. The Rocchio representation is inherently suitable for tracking a single target concept. As will be described shortly, this algorithm is also adopted for modeling a long-term interest in the MTDR algorithm.

B. MTDR Algorithm

MTDR algorithm is our concept drift learning algorithm that has been designed to track multiple target topics. Fig. 1 summarizes the MTDR algorithm (see [35] for more detail discussion of its motivation, characteristics and the algorithm description). The algorithm maintains $MTDR = \{TDR_1, TDR_2, \dots, TDR_m\}$ representation where each *TDR* (**T**hree-**D**escriptor

MTDR Algorithm ($\langle d, fb \rangle$: a relevance feedback document)

Let TDR_j be the j^{th} interest category model of $MTDR$ (see its description in text),
 M be the maximum number of TDR s maintained in $MTDR$, and
 θ is the decision threshold constant (0,1) for creating a new interest category model.

/* find the most similar interest category model */

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 ($s < \theta$)

/* d is different enough from all existing models */

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

/* update all models similar to d */

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

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

Fig. 1. MTDR algorithm.

Representation) is a distinct interest category concept. The parameter M in the algorithm limits the number of category models that can be generated in a multiple three-descriptor model.

An interest category model (TDR) is represented by a long-term and a short-term interest models. The long-term interest is modeled by a long-term descriptor LTD , which is updated using the following learning rule adopted from the Rocchio algorithm:

$$LTD_t = LTD_{t-1} + \beta \cdot d_{pos} - (1 - \beta) \cdot d_{neg} \quad (4)$$

where $0 \leq \beta \leq 1$. The short-term interest is modeled by a pair of descriptors ($PosD, NegD$) 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. The update of the positive descriptor in response to a positive feedback document d_{pos} is carried out as follows:

$$PosD_t = (1 - \alpha)PosD_{t-1} + \alpha d_{pos} \quad (5)$$

where $\alpha = (0,1)$ is the learning rate. A similar computation is defined for learning from a negative feedback document by exchanging $PosD$ and $NegD$. The similarity between a document d and an interest category TDR is a balanced mixture of interests according to the long-term and short-term models, defined by

$$\text{sim}(d, TDR) = 0.5(\text{sim}(d, LTD) + \text{sim}(d, PosD) - \text{sim}(d, NegD)) \quad (6)$$

Finally, the similarity between a document d and a given $MTDR$ is obtained from the maximum similarity to d for any TDR in $MTDR$. That is,

$$\text{sim}(d, MTDR) = \max_i \{\text{sim}(d, TDR_i)\} \quad (7)$$

C. Window-based Learning Algorithms

Window-based algorithm is a typical concept drift learning method that adapts to concept drift by sliding a window over recent examples and relearning a target concept from examples within the window [20][21][22][32][33]. Fig. 2 summarizes this algorithm. It consists of two main components typically exist in an online learner. The first component simply maintains two lists of (1) incoming documents and (2) the outcomes of predicting the relevance of incoming documents (this will be used for performance monitoring later whenever needed). The second component performs the actual learning. For simplicity, target concepts are regenerated periodically after seeing r new documents by relearning n most recent feedback documents. n is the window size determined by the *GetWindowSize()* function. The *LearnTargetConcept()* function induces target concepts from n recent examples using a selected base learner.

The *GetWindowSize()* function implements an adaptive window-adjustment heuristic. Typical heuristics [20][22][32] adjust the window sizes based on the changes in the system's predictive

Window-based Algorithm ($\langle d, fb \rangle$: a relevance feedback document)

Initialization:

$S = \langle \emptyset \rangle$, a list of relevance feedback documents in order of arrival.

$C = \text{null}$, target concept.

$P = \langle \emptyset \rangle$, 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 ($C \neq \text{null}$)

Let $p = 1$ if $Prediction(d)$ equals fb (i.e., correctly predict the relevance of d) or let $p = 0$ otherwise.

Concatenate p at the end of P .

Target Concept Learning:

$n = GetWindowSize(P)$.

$D_{LIST} = \text{Get the most recent } n \text{ documents from } S$.

$C = LearnTargetConcept(D_{LIST})$.

Fig. 2. Window-based concept drift learning algorithm.

performance. The window size is increased when the predictive performance improves or is stable, and is quickly decreased when observing a sudden performance drop. The details of these heuristics are generally domain-dependent. Fig. 3 describes the heuristic implemented in this paper, which has the same performance-based adaptation principle as those in the existing heuristics but has been customized to suit our experiment setting. The predictive performance is calculated from the outcomes of ten past predictions.

In this paper we experiment with two widely used base learners: *Rocchio* and k Nearest Neighbor (KNN) for implementing the *LearnTargetConcept()* function, namely **Win-Rocchio** and **Win-KNN** algorithms, respectively. In the Win-Rocchio algorithm, the positive and negative examples in D_{LIST} , which contains the n most recent examples, are equally weighed and the learning process in the Rocchio algorithm is performed using Eq. 4. Let $D_{Rocchio}$ be the concept

Window-Adjustment-Heuristic Algorithm (P: $\{110\}^*$, a sequence of prediction results)

Initialization:

$Accuracy_0 = 0$.

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

$WindowSize_0 = \#PastPred$.

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

$$\text{Let } Accuracy_t = \frac{\sum_{i=\|P\|-\#PastPred+1}^{\|P\|} P_i}{\#PastPred}$$

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

$WindowSize_t = WindowSize_{t-1} + r$. /* increase the window to include unaccounted r new examples */

Else

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

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

$WindowSize_t = \max \{2, Accuracy_t \cdot WindowSize_{t-1}\}$

Else

/* predictive performance is stable */

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

$WindowSize_t = WindowSize_{t-1} + 1$ /* increase the window size by one */

Else

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

$WindowSize_t = \max \{2, Accuracy_t \cdot WindowSize_{t-1}\}$

Fig. 3. Window-adjustment heuristic algorithm.

generated by the Rocchio algorithm. Prediction of class label is performed by thresholding the similarity between a document d and the Rocchio descriptor $D_{Rocchio}$. Hence, the $Prediction()$ function (see Fig. 2) in the Win-Rocchio algorithm is defined by

$$Prediction_{Rocchio}(d) = \begin{cases} 1 & \text{if } \text{sim}(d, D_{Rocchio}) \geq \theta_R \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

where θ_R is the decision threshold for the Rocchio classifier.

The Win-KNN method learns by simply storing all 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:

$$Prediction_{KNN}(d) = \arg \max_{v \in \{0,1\}} \sum_{d_i \in D_{KNN}} \text{sim}(d, d_i) \cdot \delta(v, fb(d_i)) \quad (9)$$

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

IV. EXTENDING INCOMPLETE LABELED DATA STREAM

Now we present FEILDS: a new computational Framework for Extending Incomplete Labeled Data Stream in concept drift learning. FEILDS is intended to extend existing algorithms for learning concept drift from a data stream with a few labeled data. We first discuss the existing theoretical finding that motivates our approach, and then describe the details of the algorithm.

A. Theoretical Observation

It has been shown [4][15] that reducing the sample size in stable concept learning would undercut the learner ability to approximate target concepts and hence increase the classification error. The problem is exacerbated in concept drift learning because the task also involves adaptation to possible concept drift, which is typically exploited from the given examples. The following theoretical observation attempts to justify why learning concept drift from a few labeled data is problematic, shedding light on to our solution.

Helmhold and Long provide theoretical bounds on the allowable drift rates that guarantee tractability with an error of at most ϵ as follows [15]:

$$\Delta \leq \frac{c \epsilon^2}{d \ln(1/\epsilon)} \quad (10)$$

where $c > 0$, Δ is the *drift rate* (see Section II), and d is the *Vapnik-Chervonenkis* dimension of a concept/hypothesis [9]. As discussed in Section II, a slower *drift rate* corresponds to having a

longer sequence of labeled data with the same target concept, and vice versa. Because c and d values are constant, the bounds imply that the tracking problem is more difficult (producing higher error rates) on learning with fewer labeled instances per target concept (higher drift rates). Reducing the rate of drift according to Eq. 10 is apparently the only option for improving the performance of a concept drift learner. FEILDS takes this general approach.

B. Overview of Approach

Inspired by the success of techniques that combine labeled and unlabeled data in stable concept learning [7][8][12], 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. 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 system input, unlike in typical concept drift learning setting, is a stream *Stream-S* of labeled data \mathbf{L} and unlabeled data \mathbf{U} . Only a few instances' labels are made available to the learner while the majority of unlabeled data are irrelevant. Let $\mathbf{S} = \{x_1, \dots, x_n\}$ be a set of instances taken from the input stream such that $\mathbf{S} = \mathbf{L} \cup \mathbf{U}$ and $\mathbf{L} \cap \mathbf{U} = \emptyset$. The label values in concept drift learning can change over time so that some of the labeled instances in \mathbf{L} may be no longer relevant at a later time. 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 topic category is judged irrelevant. The crux of our approach is to identify the set of labeled data $\mathbf{L}_R \subseteq \mathbf{L}$ that are still truly relevant, and to expand every instance in \mathbf{L}_R with relevant unlabeled data. Specifically, for each $x_i \in \mathbf{L}_R$, let $\mathbf{U}_i \subseteq \mathbf{U}$ be the corresponding subset of unlabeled data with the same underlying concept as that of x_i . It then

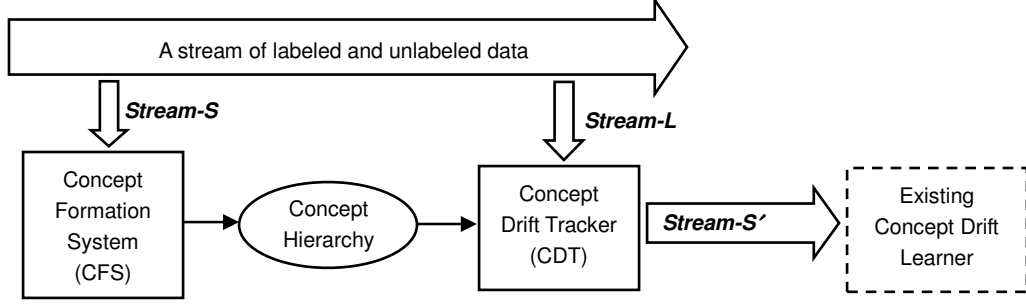


Fig. 4. FEILDS architecture.

uses the set $S' = \{x_i \cup U_i \mid x_i \in L_R\}$ to generate a new stream *Stream-S'* and assigns the label of each unlabeled instance $x_j \in U_i$ with x_i 's label.

Identifying the set of relevant labeled instances L_R requires knowledge about the concept behind each instance in L . However, the concept (topic category) that underlies a given instance (feedback document) is typically unknown to the system and cannot be induced reliably from only a small set of examples. To address this problem, FEILDS employs a concept hierarchy for providing a means to associate an instance with its concept category. The following section describes the idea in greater details.

C. FEILDS Architecture

Fig. 4 depicts the architecture of FEILDS. It consists of three main entities: (1) a concept formation system, (2) a concept hierarchy, and (3) a concept drift tracker. A concept formation system (CFS) incrementally constructs the concept hierarchy by organizing the input stream into cluster hierarchy along with the corresponding concepts. During the course of learning, the concept hierarchy grows dynamically as the system receives more data from the stream.

The concept drift tracker (CDT) component is invoked only when needed by a concept drift learner. It takes as input a hierarchy of concepts and a stream of labeled examples L , and makes

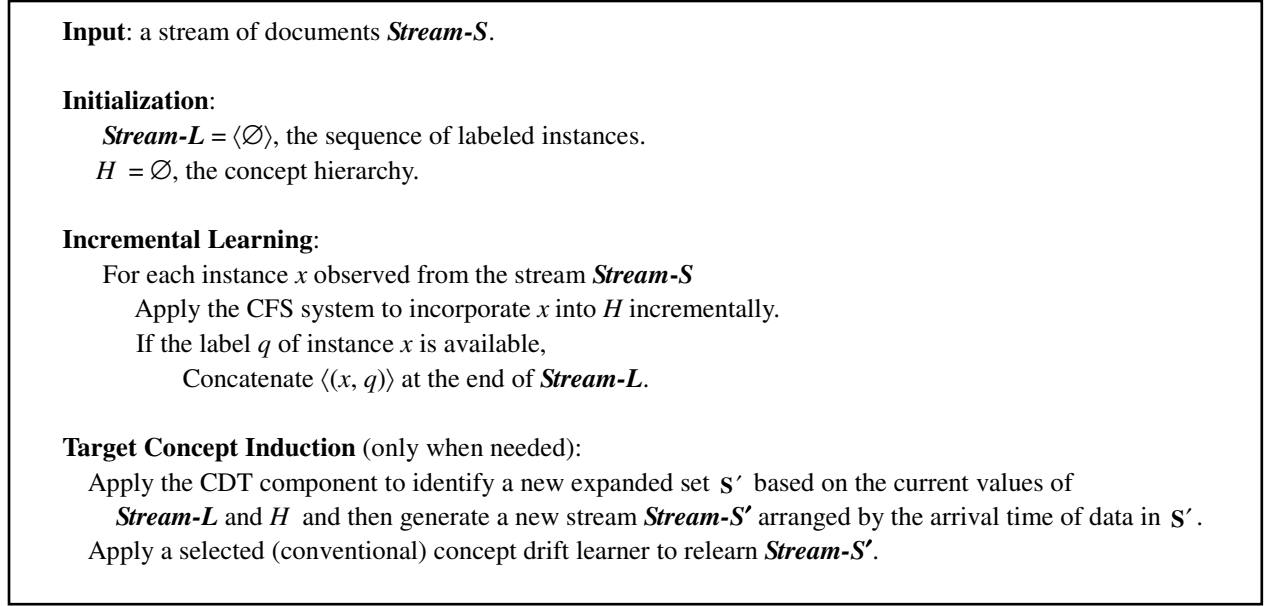


Fig. 5. The summary of FEILDS's approach.

inferences about S' that, as described above, contains the expanded relevant data. This component outputs a new stream *Stream-S'* by rearranging all instances in S' according to the instance arrival times. A (conventional) concept drift learning algorithm such as those described in Section III can then be used to relearn the new stream. Fig. 5. summarizes the interactions among these components.

1) *Instance-Concept Category Associations*: The concept hierarchy in the FEILDS architecture is a tree structure with the following characteristics: (1) all leaf nodes represent instances (e.g., text documents), and (2) all internal nodes represent concept nodes that generalize their descendants. Let X be the instance space and C be the concept node space. The following defines two functions needed by the CDT component for utilizing the concept hierarchy:

- $\delta: X \rightarrow C$. For an instance $x \in X$, let $\mathbf{A}_x = \{c_1, \dots, c_n\} \cup x$ for each $c_i \in C$ be the set of x and its ancestors where c_n is x 's parent, c_{i-1} is c_i 's parent and c_1 is the root. Given x , $\delta(x)$ returns a

concept node $c \in \mathbf{A}_x$ that best represents the concept category of x .

- $\mathcal{E} : C \rightarrow X^*$. This function returns all instances that belong to a concept node $c \in C$, that is, all leaf nodes that are descendants of a node representing the input concept node c .

The δ and \mathcal{E} functions can be used together to identify relevant data, for example, $\mathcal{E}(\delta(x))$.

While \mathcal{E} is straightforward, δ is not. The δ function still requires a method for selecting the most appropriate concept node from the concept hierarchy. This implies that the selected concept node is neither *too general* (i.e., an ancestor of the correct node) nor *too specific* (i.e., a descendant of the correct node). Over generalization, although could increase the coverage of correctly labeled instances, could also mistakenly include unintended concept categories, introducing more false positive or false negative instances. Selecting too specific a concept node, on the other hand, could lead the CDT component to miss concept drift and would reduce the coverage of correctly labeled instances. We assume that a concept node that distinctively partitions instances is the best candidate.

In this work we employ a recently developed concept formation algorithm [36] (although other suitable algorithms can also be employed, we found that [36] performed the task better in text domain). One of the concept node properties in the concept hierarchy is the *concept* (or *cluster*) *density*, which is calculated from the average distance to the nearest neighbor among the child nodes. The concept density in the hierarchy tends to decrease at higher-level nodes.

We identify distinct concept nodes by thresholding the concept density information whose cut-off point is empirically determined from a validation set. The method is similar to the process in *proximity dendrogram* cutting that identifies clusters in the cluster hierarchy according to dissimilarity levels [17]. First, a concept hierarchy is incrementally built from a stream of data

in the validation set. Because the categories of all instances are known in the validation set, distinct concept nodes can be accurately recognized from the concept hierarchy. The threshold is then calculated from the average density of these distinct concept nodes.

More specifically, let H be the concept (cluster) hierarchy generated from the validation set containing a set of concept categories T . Let $c_t \in H$ be a concept node in the hierarchy that corresponds to concept category $t \in T$. The concept node c_t is identified from H by:

$$c_t = \arg \max_{c \in H} \left\{ \sum_{x \in \mathcal{E}(c)} m(x, t) - \sum_{t' \in T - \{t\}} \sum_{x \in \mathcal{E}(c)} m(x, t') \right\} \quad (11)$$

where $m: X \times T \rightarrow \{1, 0\}$ is a binary matching function such that $m(x, t) = 1$ if $x \in \mathcal{E}(t)$, or 0 otherwise. Hence, c_t maximizes the difference between the numbers of instances that are members of t and non- t concept categories. Now let μ_t be the average distance to the nearest neighbor among c_t 's child nodes; μ represents the concept density in the concept hierarchy. Thus, a higher μ_t (the average distance) value corresponds to a lower-density concept, and vice versa. Let $\mu_{t's \text{ parent}}$ be the density of c_t 's parent. Taking μ_t as the threshold runs the risk of over-fitting to a particular concept category while selecting $\mu_{t's \text{ parent}}$ is completely inappropriate because it also covers the instances of other concept categories (i.e., over-generalization). To avoid these problems, we select a threshold value between μ_t and $\mu_{t's \text{ parent}}$, averaged over all concept categories:

$$\theta_k = \frac{1}{|T|} \sum_{t \in T} \max \left\{ \mu_t, \mu_t + k \cdot (\mu_{t's \text{ parent}} - \mu_t) \right\} \quad (12)$$

where $0 \leq k \leq 1$. We set $k = 0.5$ by default, which maximizes the margins between over-fitting and over-generalization.

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

- a) $\mu_p < \theta_k$ for $\forall p \in c$'s descendants, and
- b) $\mu_c \leq \theta_k \leq \mu_{c's \text{ parent}}$, or at least one of c 's siblings is a distinct concept.

These conditions virtually cut the concept hierarchy into non-overlapping concept nodes each of which represents the concept category of its descendant leaf nodes, implementing the δ function above.

2) *Concept Drift Tracker*: The main CDT task is to infer a subset of labeled instances that are still truly relevant (e.g., L_R) and then expand the subset with relevant unlabeled data. For clarity, we will illustrate the process within the information-filtering domain where *topic category* (e.g., document topic) will be used to denote the concept category that can be associated with an instance (e.g., document). Let $\text{Stream-L} = \langle (d_1, j_1), \dots, (d_n, j_n) \rangle$ be the stream of documents d_i with its relevance judgment j_i . A document at the left side arrives earlier. The relevance judgment is either 1 (*positive*) or 0 (*negative*) and corresponds to the data label described earlier.

The CDT component processes its input in four steps. During the first step, it transforms the problem of tracking multiple topic categories into several sub-problems of tracking a single topic category. Specifically, Stream-L is partitioned into several, shorter sequences according to their contexts (i.e., all documents in the same partition belong to the same topic category). The relative ordering of documents in the original stream is also maintained in each partition.

Example 1. Let $\text{Stream-L} = \langle (d_1, 1), (d_2, 0), (d_3, 0), (d_4, 1), (d_5, 0), (d_6, 0), (d_7, 1), (d_8, 1) \rangle$. Suppose that the sets $\{d_2, d_6\}, \{d_1, d_4, d_5\}, \{d_3, d_7, d_8\}$ belong to topics c_1, c_2 and c_3 , respectively. That is, $\delta(d_2) = \delta(d_6) = c_1$ in the first set, and so on. Then, Stream-L is partitioned into $\text{Stream-L}_1 = \langle (d_2, 0), (d_6, 0) \rangle$, $\text{Stream-L}_2 = \langle (d_1, 1), (d_4, 1), (d_5, 0) \rangle$ and $\text{Stream-L}_3 = \langle (d_3, 0), (d_7, 1), (d_8, 1) \rangle$.

The second step normalizes sequence partitions. Since each partition generated by the first step contains documents of the same topic, two or more consecutive documents with the same

relevance value constitute a redundancy. Current step eliminates such a redundancy by repeatedly dropping a document whose judgment is the same as that of document following directly in the sequence.

Example 2. Using the results from *Example 1*, d_2 will be dropped from *Stream-L*₁ because its subsequent document, d_6 , has the same relevance judgment. Similarly, documents d_1 and d_7 are also removed from *Stream-L*₂ and *Stream-L*₃, respectively. The normalized sequence partitions generated are *Stream-L*₁'= $\langle(d_6,0)\rangle$, *Stream-L*₂'= $\langle(d_4,1),(d_5,0)\rangle$ and *Stream-L*₃'= $\langle(d_3,0), (d_8,1)\rangle$.

The third step infers the relevance of each topic. We adopt the *persistence assumption* in temporal reasoning to perform this process. The assumption states that once a fact becomes true it remains true thenceforth until the fact is negated [13]. Consequently, the relevance of a topic category can be simply inferred from the relevance value given by the last document in the normalized sequence partition generated by the second step. In the information-filtering domain that typically concerns only with relevant documents, a sequence of two documents of the same topic can be dropped if the latest document is not relevant and is in conflict with the relevance value of preceding document (i.e., $\langle(d_i,1),(d_j,0)\rangle$).

Example 3. From *Example 2*, topic c_1 , as represented by *Stream-L*₁', is irrelevant while topic c_3 is relevant. *Stream-L*₂' is dropped for the above reason.

The last step retrieves instances from concept hierarchy, using the \mathcal{E} function, that are relevant to the remaining topic categories. A new stream *Stream-S*' is then generated from the retrieved instances and is ordered by the instance arrival times. Each instance's relevance value in *Stream-S*' is set to that of its topic category.

Example 4. Let $\{d_2,d_6,d_9,d_{11}\}=\mathcal{E}(c_1)=\mathcal{E}(\delta(d_6))$ and $\{d_3,d_7,d_8,d_{10},d_{12}\}=\mathcal{E}(c_3)=\mathcal{E}(\delta(d_8))$ be the sets of

documents relevant to topics c_1 and c_3 from *Example 3*. The last step will generate a new stream $\text{Stream-S}' = \langle (d_2, 0), (d_3, 1), (d_6, 0), (d_7, 1), (d_8, 1), (d_9, 0), (d_{10}, 1), (d_{11}, 0), (d_{12}, 1) \rangle$.

V. EXPERIMENTS

A. Data and Document Collection

In all experiments we used a subset of the Reuters-21578 1.0 corpus [6], selected among documents in the *ModApte* split [2] that have been assigned a single topic category. The test set contained 2581 documents consisting of 59 topics, and was used to measure the model's accuracy. The rest of the training set in the *ModApte* split, which contained 6452 documents, was further divided into training set (6352 documents) and validation set (100 documents). The training set was used to generate data streams. The validation set was used only in experiments related to the evaluation involving FEILDS.

All documents were pre-processed by removing stop words, stemming the remaining words, identifying bigrams (i.e., a sequence of two stemmed words) and counting term frequencies [37]. The document terms were then weighed according to the *tf-idf* method (see Eq. 1).

B. Experiment Procedure

The goal of experiments is to observe the system performance as target concepts (i.e., current topic of interests) change from time to time. Accordingly, the system is presented with a stream of 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*. Fig. 6 describes the procedure employed in the

Input: a data stream *Stream-S* generated from the training set.

Initialization:

1. Let *Stream-L* = $\langle \emptyset \rangle$, i.e., 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* $i = \{1 \dots K\}$

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

Fig. 6. The procedure of experiment for FEILDS evaluation.

experiments. During the initialization stage, it empirically calculates the threshold needed for realizing the instance generalization function in FEILDS.

To measure the accuracy of the system, we use the standard *recall-precision break-even point* [25], a value at which precision is equal to recall. The system accuracy is calculated from the percentage of target test documents ranked within the top n documents (where n is set to maximum number of test documents in desired categories during the current tracking cycle). 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 given as follows.

$$Accuracy_t = \frac{P}{n} \times 100\% \quad (13)$$

The *average accuracy* value is calculated by averaging the system accuracy from the first tracking task to the end.

TABLE1

The description of tracking tasks and the evolution of target concepts over twenty-tracking-cycle periods.

	Tracking Cycles				
	1 – 20	21 – 40	41 – 60	61 – 80	81 – 100
Tracking Task 1	(<i>Trade</i> , +) & 9 others	(<i>Trade</i> , –) (<i>Coffee</i> , +) & 8 others	(<i>Coffee</i> , –) (<i>Crude</i> , +) & 8 others	(<i>Crude</i> , –) (<i>Sugar</i> , +) & 8 others	(<i>Sugar</i> , –) (<i>Acq</i> , +) & 8 others
Tracking Task 2	(<i>Trade</i> , +) (<i>Coffee</i> , +) & 8 others	(<i>Trade</i> , –) (<i>Coffee</i> , +) (<i>Crude</i> , +) & 7 others	(<i>Coffee</i> , –) (<i>Crude</i> , +) (<i>Sugar</i> , +) & 7 others	(<i>Crude</i> , –) (<i>Sugar</i> , +) (<i>Acq</i> , +) & 7 others	
Tracking Task 3	(<i>Trade</i> , +) (<i>Coffee</i> , +) (<i>Crude</i> , +) & 7 others	(<i>Trade</i> , –) (<i>Coffee</i> , +) (<i>Crude</i> , +) (<i>Sugar</i> , +) & 6 others	(<i>Coffee</i> , –) (<i>Crude</i> , +) (<i>Sugar</i> , +) (<i>Acq</i> , +) & 6 others		

C. Tracking Tasks

The data streams given to systems are generated according to a tracking task, a scenario that describes the evolution of topic of interests over time. The changes in topic of interests 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 set are 75, 22, 121, 25 and 696, respectively. Table 1 provides three tracking tasks used in our experiments. Tracking task 1 simulates the evolution of a single target concept. Tracking tasks 2 and 3 learn two and three target concepts, respectively. Each column in the table describes the number and the topic of documents in the m -document sequence that is processed at each tracking cycle ($m = 10$ in all tracking tasks). In tracking task 1, for example, tracking cycles 21–40 process 10-document sequences each of which contains one *Trade* document, one *Coffee* document, and eight other (*non-target topic*) documents, randomly selected from the training set and randomly ordered in the sequence. Documents with “+” or “-” sign represent labeled data, and the data labels are made available to the system only at a few tracking cycles. The other documents (without “+”

or “-”) represent unlabeled data. When the labels of labeled data are not given to the system, these data from the system’s point of view are also considered as unlabeled. Each tracking cycle uses a new set of previously unseen documents.

For simplicity, target concepts are made stable for periods of twenty tracking cycles. Feedback document sets that mark the beginning of change in target concepts are given at the first tracking cycles during the twenty-tracking cycle periods, that is, at tracking cycles 1, 21, 41 and so on. Topics with positive (+) labels indicate current target concepts at the respective tracking cycles. Data with positive labels indicate relevant documents and are used to establish new (or emphasize the existing) target concepts. The negative labels demote previously established target concepts. For example, a positive *Trade* document in tracking task 1 is given during the first tracking cycle to establish a new interest in *Trade* topic. The document set provided during the 21st tracking cycle contains one positive *Coffee* document and one negative *Trade* document, changing the target concept from *Trade* to *Coffee*.

D. Experiment Setup

Four concept drift learners, as described in Section III, are considered for learning the stream *Stream-S'* generated by FEILDS: (1) MTDR algorithm, (2) Rocchio algorithm, (3) Win-KNN, and (4) Win-Rocchio. The main idea of FEILDS is to extend an existing concept drift learner for dealing with a few labeled data. Therefore, the performance of an existing concept drift learner is expected to improve by learning *Stream-S'* over the performance of those that learn only *Stream-L* (i.e., the original labeled data stream). The four diverse learning methods above will be used to confirm this expectation.

In all experiments we set $\theta = 0.175$, $M = 8$, $\beta = 0.1$ and $\alpha = 0.3$ in the MTDR algorithm [35], and we set $\beta = 0.1$ in the Rocchio algorithm [35]. To avoid selecting non-optimal parameters in

the Win-Rocchio algorithm, we vary the threshold θ_R in its *Prediction()* function from 0.025 to 0.35 at 0.025 intervals, and use the best results for performance comparison. Similarly, we also vary the k value of the KNN base learner from 2 to 20 in the Win-KNN algorithm.

FEILDS is run by making only 5% of the labeled data available to the system. Specifically, we provide data labels only at the first tracking cycles during the twenty-tracking-cycle periods; the performances of concept drift learners without using FEILDS are not expected to change until the next twenty tracking cycles. We also run concept drift learners without FEILDS that are provided a full (100%) labeled data set in order to see how well FEILDS (with only 5% labeled data) can recover the performance.

E. Results

Fig. 7–9 summarize the outcomes of experiments on tracking tasks 1–3, respectively. The 100%-L and 5%-L performances result from running concept drift learners without FEILDS that learn from 100% and 5% labeled data, respectively. The 5%-L performances also serve as the baselines while the 100%-L can be viewed as the empirical upper-bound. The FEILDS performances are generated from the same streams that produce the 5%-L performances, except that FEILDS also utilizes the presence of unlabeled data, which is not the case in the 5%-L systems. In all tracking tasks, the 5%-L and 100%-L systems simply ignore the unlabeled data.

As expected, the 100%-L systems outperform the others since these systems receive a full set of labeled data. All algorithms in 5%-L systems severely suffer from being unable to maintain their accuracies when presented with minimal labeled data. The difference between 100%-L and 5%-L performances represents a room for improvement, the extent to which the 5%-L performance can be improved by FEILDS. As shown in the figures, FEILDS can effectively improve the performances of existing concept drift learning algorithms (5%-L systems) except

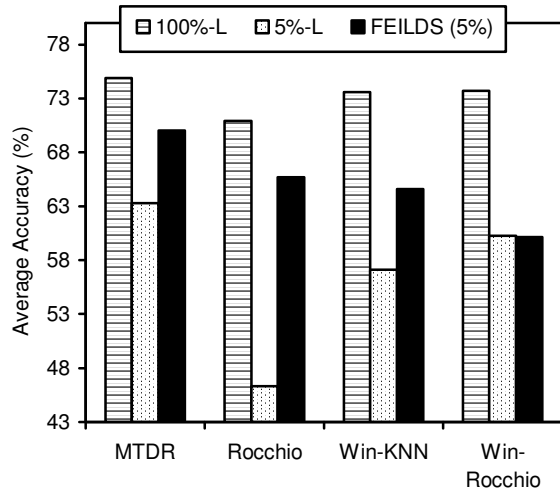


Fig. 7. Performance on tracking task 1.

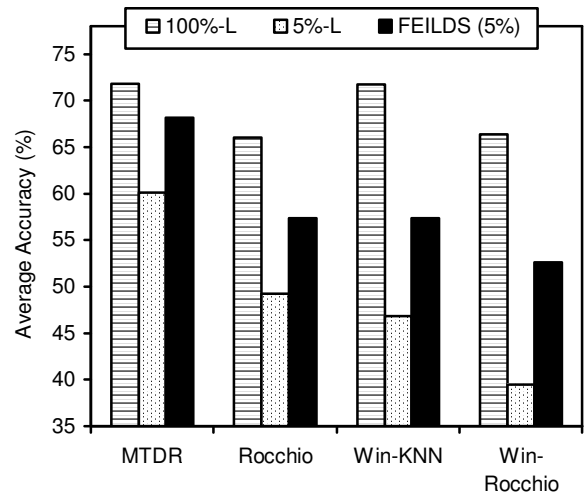


Fig. 8. Performance on tracking task 2.

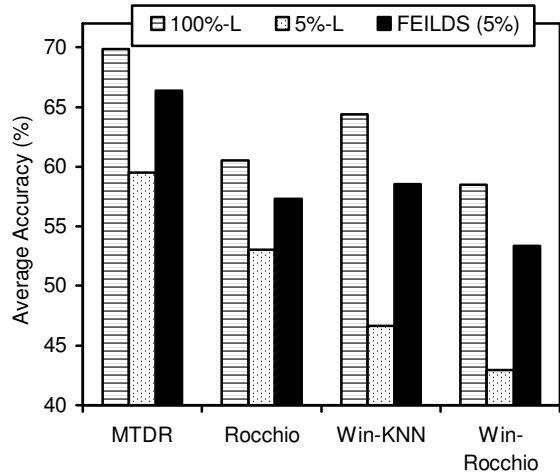


Fig. 9. Performance on tracking task 3.

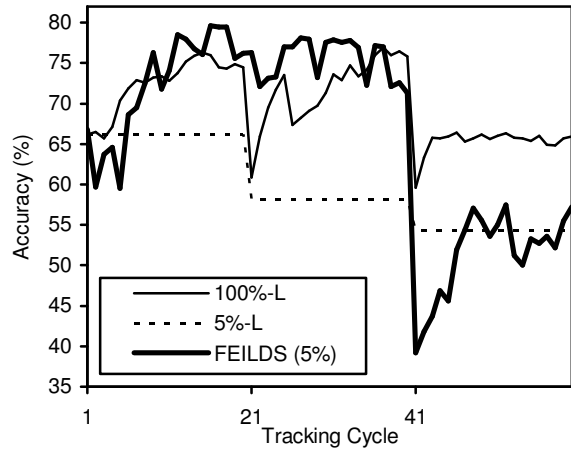


Fig. 10. Performance over time of the MTDR algorithm on tracking task 3.

the performance of FEILDS (5%) employing Win-Rocchio learner on tracking task 1. It is worth noting that all the four learning algorithms receive the same *Stream-S'* at a given tracking cycle and tracking task. Therefore, the failure of the Win-Rocchio learner for improving its performance as above is more likely due to the problem within the algorithm itself.

Fig. 10 depicts an example of performance over time provided by the MTDR algorithm on tracking task 3. Except in the last twenty tracking cycles whose target topics involve *Acq*, most

of the performance gains achieved by FEILDS over the baseline performances are significant, and in some cases are even better than the performances of 100%-L systems. Acq represents the most difficult topic to learn. Nonetheless, the fact that FEILDS is still able to improve its performances automatically, although rather slower, with the increasing availability of Acq documents is the behavior expected from the system.

The quality of the concept (cluster) hierarchy and the accuracy of the instance generalization method could affect the quality of the system's output. Although FEILDS is able to retrieve more relevant unlabeled data (i.e., by adjusting the parameter in the instance generalization method; we will show this shortly), some of the unlabeled data retrieved could be irrelevant or incorrectly labeled. It is likely that this noise prevents the performance of FEILDS from being better than that of the 100%-L system.

The main experiment results above are produced by setting the threshold for instance generalization to its default value (i.e., $k=0.5$ or $\theta_{0.5}$ in Eq. 12). Exploratory experiments were further conducted to observe the sensitivity of the threshold values on (1) the quality of the set S' generated by the CDT component, and (2) the learner's performance from learning *Stream-S'*. The quality of S' is measured in terms of *noise* and *coverage*. The *noise* is the percentage of instances in S' that are incorrectly labeled, while the *coverage* is the percentage of correctly labeled instances in S' over all target instances in the concept hierarchy. The information for calculating these two measures is cumulatively counted from the first tracking task to the end (i.e., akin to the *micro-averaging* technique), averaged over ten trials.

Fig. 11 and 12 provide the results of this experiment. For readability, a threshold factor $tf = \frac{\theta}{\theta_0}$ is introduced to describe a relative threshold with respect to θ_0 . In this respect, the

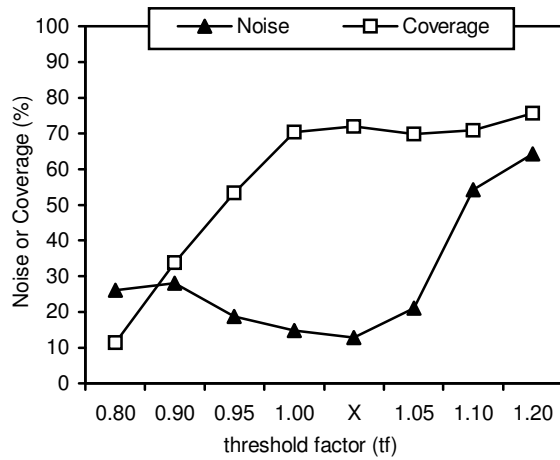


Fig. 11. The quality of S' over threshold factor on tracking task 3.

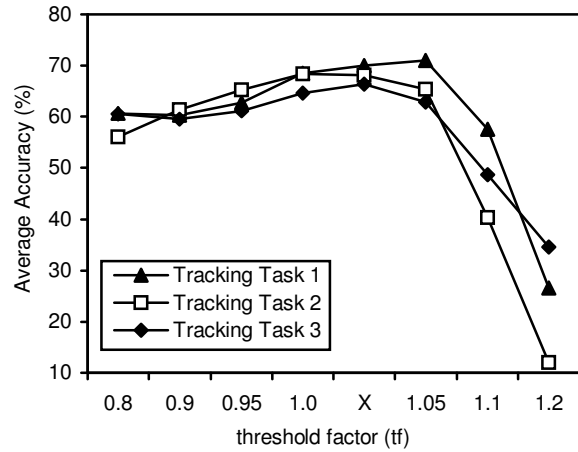


Fig. 12. The performance of MTDR algorithm over threshold factor.

threshold factors of the default settings, denoted by X in the figures, are in the (1.0, 1.05) range.

Fig. 11 presents typical results that confirm the expectation discussed earlier. A higher threshold increases the coverage of S' . Too high a threshold ($tf > 1.1$) also introduces high percentage of noise. The default settings provide a good tradeoff between low noise and high coverage.

Fig. 12 summarizes the performances of the MTDR algorithm from learning the stream *Stream-S'* over various threshold values. It shows that the default settings yield relatively stable performances at high average accuracies. Smaller thresholds do not help improve the algorithm's performances, converging to the baseline average accuracies. Higher threshold factors ($tf > 1.1$) are prone to a detrimental effect that degrades the algorithm's average accuracies even much worse than those of the baselines.

VI. DISCUSSION OF RELATED WORKS

User interest modeling has been an active research area in the Information Retrieval [1][16][38], Intelligent Agents [3][5][11], and Machine Learning [24] communities, to name a few. Researchers address the problem of changing user interests by modeling user interests as compositions of long-term and short-term interests [5][34], relearning examples of recent window [20][21][22], systematically decaying the importance of older feedback [1][3][10], or by employing an evolutionary algorithm [28]. Theoretical results on concept drift learning mainly establish bounds on the drift rate and the sample complexity by making simplifying assumption about the kind of drift that can occur [4][15]. Practical techniques adapt to concept drift by applying concept weight thresholding [31], adaptively adjusting the window size [32], and explicitly detecting hidden contexts through meta-learning [33] or off-line learning [14].

Although intended for learning stable concepts (static user interests), there has been a large number of algorithms that exploit unlabeled data in the presence of incomplete labeled data. These include the Expectation Maximization [12], Co-training [8] and Graph Min-cut [7] algorithms. An evolutionary algorithm has been recently incorporated to iteratively adjust the class membership of unlabeled data [23]. In Information Retrieval, automatic query expansion often exploits unlabeled data from the top ranked documents [27].

Transductive SVM (TSVM) is employed as the base learner in the window-based concept drift learning algorithm [19], allowing the system to incorporate unlabeled data in the test set during the learning process. However, this approach has never been evaluated using fewer labeled data, making the effectiveness of this method remains unclear. Besides, the use of TSVM classifier still poses some controversy such as the validity of using a test set as a means of inductive inference, and the possibility of maximizing wrong margins [39].

VII. CONCLUSIONS

We have described and evaluated a new method for modularly extending the capability of existing concept drift learning algorithms for learning with minimal labeled data. The experiment results show that FEILDS is able to automatically improve the performances of existing algorithms over time as more relevant unlabeled data become available. The main disadvantage of our approach is that it also introduces extra computational costs because it has to reproduce a new stream every time its input changes and the learner must relearn the new stream from scratch. The overall performance of FEILDS also depends greatly on two major factors: (1) the quality of its concept hierarchy and (2) the ability to correctly recognize the category of an instance from the concept hierarchy. This suggests that an attempt to improve the current approach or to apply it to other application domains can be focused on the re-engineering or inventing a new method that would improve those two factors. This is the subject of our future work.

REFERENCES

- [1] Allan, J. (1996). Incremental Relevance Feedback for Information Filtering. In *Proc. of the 19th International Conf. on Research and Development in Information Retrieval*, pp 270-278.
- [2] Apt'e, C., Damerau, F. and Weiss, S. M. (1994). Automatic Learning of Decision Rules for Text Categorization. *ACM Transactions on Information Systems*, 12 (3): 233-251.
- [3] Balabanović, M. (1997). An Adaptive Web Page Recommendation Service. In *Proc. of the 1st International Conf. on Autonomous Agents*, pp. 378-385.
- [4] Bartlett, P.L., David, S.B. and Kulkarni, S.R. (1996). Learning Changing Concepts by Exploiting the Structure of Change. *Computational Learning Theory*, pp. 131-139.
- [5] Billsus, D. and Pazzani, M. (1999). A Personal News Agent that Talks, Learns and Explains. In *Proc. of the 3rd International Conf. on Autonomous Agents*, pp. 268-275.
- [6] Blake, C. and Merz, C. (1998). UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/_mlearn/MLRepository.html, University of California, Irvine, Department of Information and Computer Sciences.

- [7] Blum, A. and Chawla, S. (2001). Learning from Labeled and Unlabeled Data using Graph Mincuts. In *Proc. of the 18th International Conf. on Machine Learning*, pp. 19-26.
- [8] Blum, A. and Mitchell, T. (1998). Combining Labeled and Unlabeled Data with Co-Training. In *Proc. of the 11th Annual Conf. on Computational Learning Theory*, pp. 92-100.
- [9] Blummer, A. Ehrenfeucht, A., Haussler, D. and Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis Dimension. *Journal of the ACM*, 36 (4):929-965.
- [10] Chen, C. C., Chen, M. C. and Sun, Y. (2002). PVA: A Self-Adaptive Personal View Agent. In Special Issue on Automated Text Categorization, *Journal of Intelligent Information Systems*, 18(2-3):173-194.
- [11] Chen, L. and Sycara, K. (1998). WebMate: Personal Agent for Browsing and Searching. In *Proc. of the 2nd International Conf. on Autonomous Agents*, pp. 132-139.
- [12] Dempster, A.P., Laird, N.M. and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithms. *Journal of the Royal Statistical Society, Series B.*, 39(1): 1-38.
- [13] Gabbay, D.M, Hogger, C. J. and Robinson, J.A. (1995). Handbook of Logic in Artificial Intelligence and Logic Programming: V4. Epistemic and Temporal Reasoning. New York: Oxford University Press.
- [14] Harries, M.B., Sammut, C. and Horn, K. (1998). Extracting Hidden Context. *Machine Learning*, 32(2): 101-128.
- [15] Helmbold, D.P. and Long, P.M. (1994). Tracking Drifting Concepts by Minimizing Disagreement. *Machine Learning*, 14(1): 27-45.
- [16] Hull, D. A. (1998). The TREC-7 Filtering Track: Description and Analysis. In E. M. Voorhees and D. K. Harman (Ed.), NIST Special Publication 500-242: *The 7th Text Retrieval Conf. (TREC-7)*, pp. 33-56.
- [17] Jain, A.K. and Dubes, R.C. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, N.J.: Prentice Hall.
- [18] Jansen, B. J., Spink, A. and 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(2): 207-227.
- [19] 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.
- [20] Klinkenberg, R. and Joachims, T. (2000). Detecting Concept Drift with Support Vector Machine. In *Proc. of the 17th International Conf. on Machine Learning*, pp. 487-494.
- [21] 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.).
- [22] Klinkenberg, R. and Renz, I. (1998). Adaptive Information Filtering: Learning in the Presence of Concept Drifts. In *AAAI Workshop on Learning for Text Categorization*, pp. 33-40.
- [23] Kothari, R. and Jain, V. (2002). Learning from Labeled and Unlabeled Data. In *Proc. of the 2002 International Joint Conf. on Neural Networks*, pp. 2803-2808.

- [24] Lang, K. (1995). News Weeder: Learning to Filter News. In *Proc. of the 12th International Conf. on Machine Learning*, pp. 331-339.
- [25] Lewis, D.D. and Ringuette, M. (1994). A Comparison of Two Learning Algorithms for Text Categorization. In *Proc. of the 3rd Annual Symposium on Document Analysis and Information Retrieval*, pp. 81-93.
- [26] Mitchell, T.M. (1997). *Machine Learning*. New York : McGraw-Hill.
- [27] Mitra, M., Singhal, A. and Buckley, C. (1998). Improving Automatic Query Expansion. In *Proc. of the 21st Conf. on Research and Development in Information Retrieval*, pp. 206 - 214.
- [28] Moukas, A. and Zacharia, G. (1997). Evolving a Multi-agent Information Filtering Solution in AMALTHEA. In *Proc. of the 1st International Conf. on Autonomous Agents*, pp. 394-403.
- [29] Rocchio, J.J. (1971). Relevance Feedback in Information Retrieval. In G. Salton, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pp. 313-323.
- [30] Salton, G. and McGill, M.J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill Publishing Company.
- [31] Schlimmer, J.C. and Granger, R.H. (1986). Beyond Incremental Processing: Tracking Concept Drift. In *Proc. of the 5th National Conf. on Artificial Intelligence*, pp. 502-507.
- [32] Widmer, G. and Kubat, M. (1996). Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23 (1): 69-101.
- [33] Widmer, G. (1997). Tracking Context Changes through Meta-Learning. *Machine Learning*, 3:259-286.
- [34] Widyanoro, D.H., Ioerger, T.R. and Yen, J. (1999). An Adaptive Algorithm for Learning Changes in User Interests. In *Proc. of the 8th International Conf. on Information and Knowledge Management*, pp. 405-412.
- [35] Widyanoro, D.H., Ioerger, T.R. and Yen, J. (2001). Learning User Interest Dynamics with a Three-Descriptor Representation. *Journal of the American Society for Information Science*, 52(3):212-225.
- [36] Widyanoro, D.H., Ioerger, T.R. and Yen, J. (2002). An Incremental Approach to Building a Cluster Hierarchy. In *Proc. of the 2nd IEEE International Conf. on Data Mining*, pp. 705-708.
- [37] Witten, I.H., Moffat A. and Bell T.C. (1994). *Managing Gigabytes: Compressing and Indexing Documents and Images*. New York, NY: Van Nostrand Reinhold.
- [38] Yang, Y., Carbonell, J.D., Brown, R.D., Pierce, T., Archibald, B.T. and Liu, X. (1999). Learning Approaches for Detecting and Tracking News Events. *IEEE Intelligent Systems: Special Issue on Applications of Intelligent Information Retrieval*, 14(4):32-43.
- [39] Zhang, T. and Oles, F.J. (2000). A Probability Analysis on the Value of Unlabeled Data for Classification Problems. In *Proc. of the 17th International Conf. on Machine Learning*, pp. 1191-1198.